Watermarking in Deep Neural Networks via Error Back-propagation

***Jiangfeng Wang*[†]*, Hanzhou Wu*[†,‡,*]*, Xinpeng Zhang*[†,‡,*]*, and Yuwei Yao*[†]
[†]***School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China***
[‡]***Shanghai Institute for Advanced Communication and Data Science, Shanghai 200444, China***
[*]***Email: h.wu.phd@ieee.org, xzhang@shu.edu.cn***

Abstract

Recent advances in deep learning (DL) have led to great success in tasks of computer vision and pattern recognition. Sharing pre-trained DL models has been an important means to promote the rapid progress of research community and development of DL based systems. However, it also raises challenges to model authentication. It is quite necessary to protect the ownership of the DL models to be released. In this paper, we present a digital watermarking technique to deep neural networks (DNNs). We propose to mark a DNN by inserting an independent neural network that allows us to use selective weights for watermarking. The independent neural network is only used in the training phase and watermark verification phase, and will not be released publicly. Experiments have shown that, the performance of marked DNN on its original task will not be degraded significantly. Meantime, the watermark can be successfully embedded and extracted with a low neural network loss even under the common attacks including model fine-tuning and compression, which has shown the superiority and applicability of the proposed work.

Introduction

Recent progress in deep learning (DL) has led to a dramatic surge in the use of DL models for a wide variety of applications such as image classification, speech recognition, natural language processing, and so on. Many state-of-the-art deep neural networks (DNNs) such as LeNet [1], AlexNet [2], VGGNet [3], GoogLeNet [4] and ResNet [5] have shown more superior performance than traditional statistical learning methods. Major enterprises like Apple, Google, Facebook, Microsoft, and Amazon have already deployed DL models in their commercial products and services.

The design of DNNs requires designers' extensive expertise and large quantities of training data and time, e.g., it takes weeks and even more time to train a very deep ResNet [5] with latest GPUs on the ImageNet dataset [6]. It, to a certain extent, has led to the boom of open-source communities, allowing pre-trained models to be publicly released for research and practice. Besides, releasing a trained model allows users to adapt the model to a new task by fine-tuning with a low computational cost.

Although the released models can facilitate the research and development of DL, they may be used for illegal or unauthorized purposes by malicious users. For example, one can use a released model for commercial service, which may seriously infringe the owner's right. It therefore requires us to investigate an effective way to protect the copyright ownership when a pre-trained model is released, shared, or distributed by somehow way.

Obviously, digital watermarking can be applied to DL models for ownership protection. Different from watermarking in digital images, watermarking in DL models can be investigated from two aspects in terms of the watermark extraction [7-14], i.e., *white-box* setting and *black-box* setting. For the former, the DL model parameters and structure can be all accessed by the watermark extractor. For the latter, the model profile including the parameters and structure are usually inaccessible to the watermark extractor, who may use a remote application programming interface (API) to access the DL model. In this paper, we focus on the white-box setting since it is relatively more common in practice.

Referring to the evaluation indicators of images, watermarking in DL models generally considers the following requirements:

- **Fidelity:** The performance of the marked neural network on its original task should not be significantly degraded after embedding a watermark. E.g., if we want to embed a watermark into the VGGNet, the image classification accuracy of the marked VGGNet should not be significantly reduced.
- **Robustness:** The watermarking algorithm should be able to well resist against common model attacks such as fine-tuning and model compression. It is very difficult to design a watermarking technique resisting against arbitrary attack.
- **Security:** The embedded watermark should not be accessed by any unauthorized party. In other words, without the secret key, one will never be able to reconstruct the watermark.
- **Capacity:** It is desirable to design a watermarking method allowing a neural network to carry as many bits as possible.
- **Efficiency:** The computational cost for watermark embedding and extraction should be low in terms of practical use.

Different from watermarking in images, watermarking in DL models is more difficult since slightly modifying a DL model may easily degrade the performance on its original task, which is not acceptable for practice. Moreover, retraining a DL model will easily remove the embedded watermark, which is a difficult problem that has not been well addressed at present. A robust watermarking method usually can resist against specific attack(s) under restrictions. The common attacks include but are not limited to:

- ***Model Retraining:*** The released DL model is retrained with a new dataset. The current state-of-the-art methods based on weight modification cannot resist against model retraining. A neural network containing a well-designed *structural pattern* may be able to resist against model retraining, which requires that, the structural pattern is well concealed and removing it will significantly reduce the performance on its o-

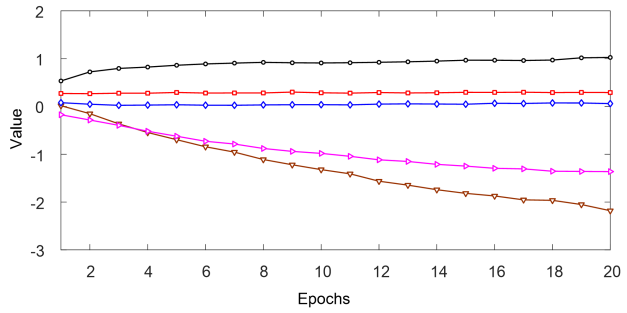

Figure 1. *The updated values versus epochs for several individual weights.*

riginal task. We define it as *structural watermarking*, which will be investigated in our future work.

- ***Model Compression:*** This is a very effective means to deploy DNNs. After compressing the DL model, many redundant or unimportant neural nodes can be removed such that the computational cost can be reduced. Such lossy compression would distort the model parameters, accordingly enlarging the difficulty of designing robust watermarking based on parameter alteration.
- ***Model Fine-tuning:*** Fine-tuning is a kind of transfer learning technique. Training a DNN from scratch is time consuming and laborious. Model fine-tuning allows us to quickly produce a desired model based on a pre-trained model, by fine-tuning the parameters with another dataset. It can be an intentional attack performed by malicious users.
- ***Model Isomorphism:*** A trained neural network can be modeled as a directed graph, in which the edges are associated with weights, and the nodes are associated with activation functions. Accordingly, a DL model may be released as a form of graph data structure, which allows an attacker to use an isomorphic graph to replace the original one. Thus, before extracting the watermark, one has to deal with the graph isomorphism problem. Though the general graph isomorphism problem is one of few standard problems in the computational complexity theory belonging to NP, we point that, the graph isomorphism problem sometimes may be reduced to finding the statistical similarity between two graphs according to model parameters and structural characteristics. For example, considering all the weights of each layer for two DNNs, if the two sorted weight-sequences are quite similar, the two DNNs may be the same. Actually, model isomorphism will also introduce graph matching problem.

We propose to use an independent neural network to insert watermark information into the selective weights of the host neural network. The independent neural network will be trained together with the host neural network. However, after training the two neural networks, only the host neural network model will be released and the independent neural network model will be kept secretly. Experiments have shown that, a watermark can be embedded and extracted with a low loss of the independent neural network under attacks including model fine-tuning and compression, without significantly degrading the performance of the original task, which has verified the feasibility.

The rest of this paper is organized as follows. We first introduce the proposed method in detail. Then, we conduct convinced experiments for performance evaluation and analysis. Finally, we conclude this paper and provide discussion.

Table 1. The detection accuracy on MNIST[1]/CIFAR10[2] by using marked/non-marked MLP/VGG16. Here, "ALL" means using all weights between the hidden layer and the output layer (excluding the biases), and "PART" means using all convergent weights between the hidden layer and the output layer (excluding the biases). The MNIST was used for MLP, and CIFAR10 for VGG16. The loss L_2 measures the difference between retrieved watermark and raw watermark. The learning rate was $\mu = 0.001$ for MLP and $\mu = 0.01$ for VGG16, and the regularization coefficient was set as $\lambda = 0.05$. The MLP used Adam optimizer, and the VGG16 used stochastic gradient descent (SGD) optimizer.

DNN/Strategy	Non-marked	Marked	L_2 (Loss)[3]
MLP/ALL	97.57%	97.18%	0.0024
MLP/PART	97.57%	97.28%	0.0015
VGG16/ALL	79.29%	77.92%	0.0013
VGG16/PART	79.29%	78.71%	0.0010

[1]`http://yann.lecun.com/exdb/mnist`
[2]`https://www.cs.toronto.edu/~kriz/cifar.html`
[3]The bit-size of the embedded watermark was 512.

Proposed Method

Embedding a secret watermark into a given DNN is defined as the task of embedding a l-bit binary vector $\mathbf{m} \in \{0,1\}^l$ into the weights of a host model. In this section, we first present our technical motivation. Then, we show the sketch of proposed method, followed by the details of watermark embedding and extraction.

Motivation

We used the MNIST for experiments, and found that around 35% weights converged[1] after 16 epochs during the training phase with a multilayer perceptron (MLP). The MLP contains three layers: a 784-D input layer, a 64-D hidden layer and a 10-D output layer. The ReLU [15] was used as the activation function. Figure 1 shows the updated values of several individual weights during the training phase. It can be seen that, a part of weights will converge after thousands of iterations. This inspires us to propose a hypothesis that embedding watermark information into early convergent weights will be quite suitable in terms of *fidelity*.

We used the watermark embedding algorithm introduced in [7] for further verifying our hypothesis, where a random matrix mentioned in [7] was utilized. We tested two algorithms. One was the original watermarking method introduced in [7], and the other one only selected the convergent weights out for data embedding while the embedding operation was the same as [7]. Table 1 has shown the results using two different DNNs, i.e., MLP mentioned above, and VGG16 [3]. It can be observed that, embedding watermark bits into convergent weights can achieve better performance on the original task compared to that using all weights. It motivates us to propose a new watermarking technique to further

[1]A weight was considered as converged if the sample standard deviation of the latest 5 updated values (each corresponds to one epoch) during training was less than a threshold, e.g., 0.01 in default. Only the weights between the hidden layer and output layer excluding biases were used.

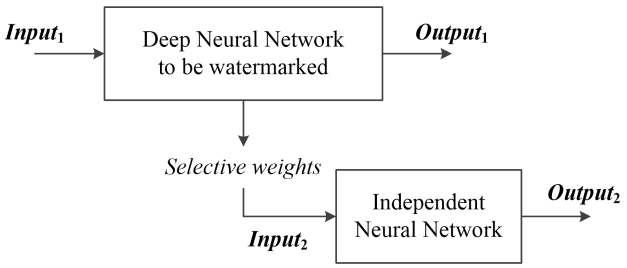

Figure 2. *Sketch of the general framework (for the model training phase).*

improve the fidelity and meanwhile provide superior robustness.

General Framework

Figure 2 shows the sketch of the proposed general framework for the model training phase. It consists of two neural networks. One is the original neural network to be watermarked and the other neural network is utilized for embedding/extracting the watermark into/from the weights of the original neural network. The independent neural network can be arbitrary effective neural network. In this paper, we consider it as a MLP *without biases*.

During the training phase, the selective weights in the original neural network are fed to the independent neural network. The parameters in both neural networks will be updated during training. Assuming that, the loss functions for the two neural networks are represented by L_1 (for host) and L_2 (for independent). By back-propagation, the parameters of the host neural network will be updated according to L_1 and L_2, and the parameters of the independent neural network will be updated according to only L_2.

Watermark Embedding

The watermark is embedded by training the aforementioned two neural networks simultaneously. A key task is to select the most suitable weights from the host DNN as the input of the independent neural network. We introduce two methods to address the problem. One is to *manually* select the convergent weights as the input. The other one is to let the independent neural network itself *automatically* choose the suitable weights, which is achieved by inserting a new trainable layer between the input and the original hidden layer after the input.

Let "MANU" and "AUTO" respectively represent the manual method and the automatic method. Figure 3 shows the network structures for MANU and AUTO. In Figure 3 (a), N indicates the number of weights classified as convergent, and in Figure 3 (b), M equals the number of all weights to be watermarked. It means that, M could be the total number of all trainable parameters of the host DNN. Due to the limited computational resource, in our experiments, we embed a watermark into the weights between two specified layers. A weight is considered as converged if the sample standard deviation of the latest 5 updated values (each one corresponds to one epoch) during training is less than a threshold, e.g., 0.01 in default in this paper. One may redefine *convergence*.

During the model training, the weights of the two neural networks will be updated by back-propagation according to the loss functions L_1 and L_2. L_1 is dependent of the task of the host DNN. L_2 is defined as the binary cross entropy between the output vector

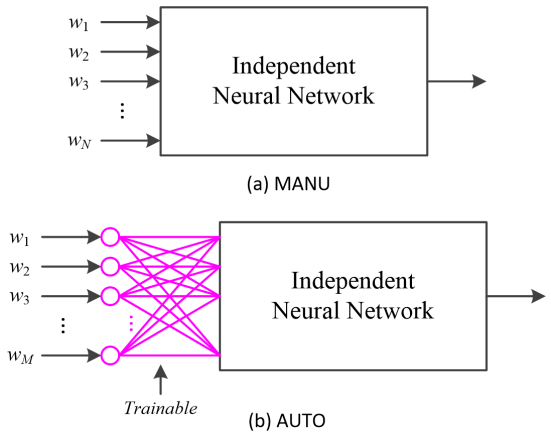

Figure 3. *The network structures for the MANU and AUTO methods.*

and the watermark bitstream (for a single sample), i.e.,

$$L_2(\mathbf{m},\mathbf{y}) = -\frac{1}{l}\sum_{i=1}^{l}\left(m_i \cdot \log y_i + (1-m_i)\cdot\log(1-y_i)\right), \quad (1)$$

where m_i is the i-th watermark bit, and y_i is the i-th output. For L_2, the back-propagation operation will update the *reachable* weights of the host DNN. A reachable weight will be updated based on L_1 and L_2 simultaneously. Notice that, the entire loss for a reachable weight is $L_1 + \lambda \cdot L_2$, and the weights for the independent neural network are updated according to only L_2, rather than $\lambda \cdot L_2$.

It can be easily found that, the watermark information is actually embedded by error back-propagation. Though the MANU can be considered as a special case of the AUTO intuitively, there is a significant difference between them. Namely, we should previously select the convergent weights out for MANU, which is not needed for AUTO. For MANU, we have to train the host DNN with several epochs, and then determine the convergent weights. Thereafter, the convergent weights will be selected and fed to the independent neural network for further training.

Watermark Verification

Once the training task is finished, the *marked* neural network can be released, while the independent neural network should be kept secretly. In order to extract the watermark showing the ownership, the independent neural network will be reused, where its input will connect to the selective weights of the marked neural network, and its output indicates the watermark information. For the output vector, a threshold 0.5 is used to convert a real number to either 0 or 1. In addition, if the AUTO method was used, the trainable parameters shown in Figure 3 (b) should be kept secretly as well after the training phase. In order to protect intellectual property, we have to ensure that the loss of the independent neural network is low (e.g., smaller than a given threshold).

Experimental Results and Analysis

In this section, experimental results are provided for performance evaluation and analysis.

Setup

The aforementioned MLP (tested on MNIST) and VGG16 (tested on CIFAR10) were separately used as the host DNN in

Table 2. The structure of the independent neural network.

Module	Number of nodes
Input layer	640 (MLP) / 1728 (VGG16)
Trainable layer	640 (MLP) / 1728 (VGG16)
Hidden layer	256
Output layer	l (watermark size)

Table 3. Classification accuracy of non-marked/marked DNNs.

DNN/Method	Non-marked	Marked	L_2
MLP/[7]	97.45%	97.28%	1.55×10^{-1}
MLP/MANU	97.45%	97.44%	8.50×10^{-5}
MLP/AUTO	97.45%	97.49%	5.90×10^{-8}
VGG16/[7]	83.65%	82.62%	2.38×10^{-3}
VGG16/MANU	83.65%	83.41%	7.28×10^{-8}
VGG16/AUTO	83.65%	83.21%	6.63×10^{-8}

our experiments. The MNIST was divided into 48,000 images for training, 12,000 images for validation, and 10,000 images for testing. The CIFAR10 was divided into 40,000 images for training, 10,000 images for validation, and 10,000 images for testing. Unless mentioned, the learning rate was set as $\mu = 0.001$ for MLP and $\mu = 0.01$ for VGG16. And, the regularization coefficient was set as $\lambda = 0.05$. Moreover, the MLP used Adam optimizer, and the VGG16 used SGD optimizer (decay = 10^{-6}). The batch size was set as 32. A MLP *without biases* was used as the independent neural network. We used a random bitstream as the watermark.

In terms of implementation, the MANU method can be processed as a special case of the AUTO method. Namely, we can actually add an extra trainable layer for the MANU method, whose network structure is then similar to Fig. 3 (b). The only difference is, the parameters of such so-called trainable layer are actually non-trainable, and each parameter should be fixed as either 1 or 0, which can be easily determined. For simplicity, we consider the "trainable layer" as a part of the independent neural network. Thus, the inputs for MANU and AUTO are the same.

For the host MLP, we embedded the watermark into the weights between the hidden layer and the output layer. For the host VGG16, we embedded the watermark into the first CONV3-64 layer [3]. Therefore, the input layer of the independent neural network has $64 \times 10 = 640$ nodes for the host MLP, and $3 \times 3 \times 3 \times 64 = 1728$ nodes for the host VGG16 (not counting the biases). The structure of the independent neural network is shown in Table 2. Notice that, one may redefine the structure by himself/herself. The ReLU function was used for all DNN layers except for the output (using softmax) of the host DNN and the output (using sigmoid) of the independent neural network.

Fidelity

Unless mentioned, the size of a watermark was always 512. We trained a host neural network with/without embedding a watermark. For the host MLP, the number of training epochs was 50, and the MANU method embedded a watermark into 163 convergent weights after 6 epochs. For the host VGG16, the number of training epochs was 150, and the MANU method embedded a watermark into 793 convergent weights after 26 epochs. Table 3 lists the classification accuracy of the original model and the marked model for the proposed method and the method in [7]. It can be observed that, though all methods can embed a watermark without significantly impairing the performance of the original task, the proposed method significantly outperforms the method in [7] in terms of both fidelity and watermark loss.

Table 4. Performance of robustness before/after fine-tuning.

Method	Domain	L_2 (before)	L_2 (after)	Acc.[1]
[7]	SD	2.38×10^{-3}	4.03×10^{-2}	98.83%
[7]	DD	2.38×10^{-3}	1.59×10^{-1}	63.87%
MANU	SD	7.28×10^{-8}	5.95×10^{-6}	100%
MANU	DD	7.28×10^{-8}	8.23×10^{-2}	100%
AUTO	SD	6.63×10^{-8}	2.09×10^{-7}	100%
AUTO	DD	6.63×10^{-8}	2.45×10^{-2}	100%

[1]The percentage of correctly extracted watermark bits.

Robustness

Embedding a watermark into a DNN does not only affect the performance of the original task, but also has to face the common attacks. Here, we consider two attacks, i.e., model fine-tuning and model compression, to evaluate the robustness of proposed work.

Model Fine-tuning

Fine-tuning allows us to apply a pre-trained model to other similar tasks with less effort than training a DNN from scratch and reduce the degree of over-fitting when sufficient training data is not available. To simulate model fine-tuning, we first trained a marked VGG16 with the full CIFAR10 dataset using 150 epochs. Then, we performed the fine-tuning operation on two domains, i.e., same domain (SD) and different domain (DD). For SD, we randomly selected 12,000 images from CIFAR10 for fine-tuning with 50 epochs. For DD, we randomly selected 12,000 images from CIFAR100 for fine-tuning with 50 epochs. During the fine-tuning, the independent neural network will not be used. For CIFAR100, the output layer of the marked VGG16 should be extended to 100-D. That means, we have to slightly modify the output layer of the marked DNN before fine-tuning. After fine-tuning, we extracted the watermark with the independent neural network. Table 4 shows the experimental results, from which we can find that, the proposed work can well resist against the fine-tuning attack and significantly outperforms the method in [7].

Model Compression

Model compression is also a common attack. Model compression reduces the computational cost of deploying a DL model by removing a part of parameters. In experiments, we considered three kinds of model compression, namely, we removed $\alpha\%$ parameters of the embedded layer of a marked DL model by sorting all parameters (excluding biases) in ascending, random and descending order respectively. E.g., for "ascending", we removed $\alpha\%$ smallest parameters and used the rest for watermark extraction. Notice that, the parameters were sorted according to their

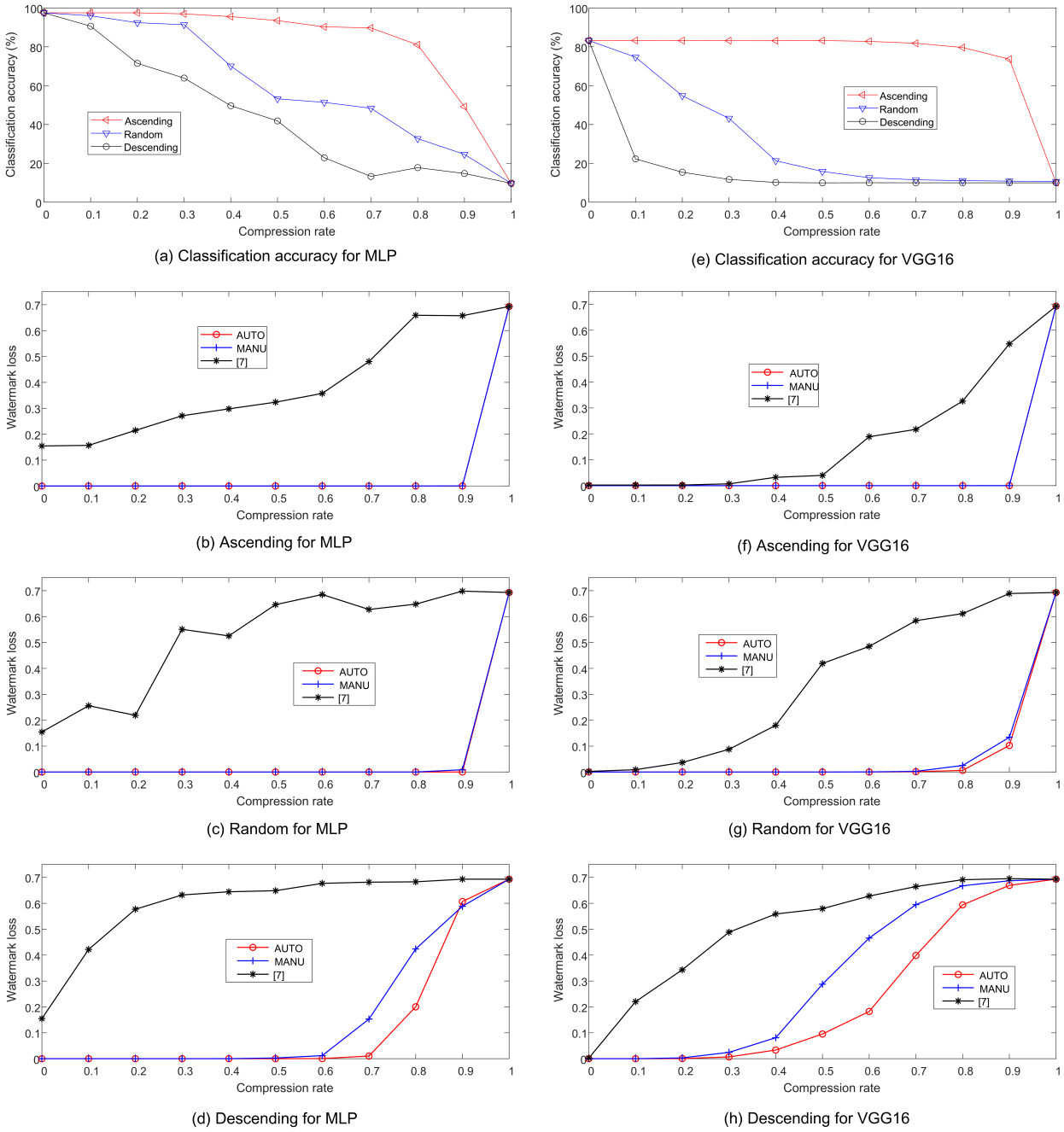

Figure 4. *Performance evaluation of robustness after model compression with different removal strategies for MLP and VGG16. (a, e) used the AUTO method.*

absolute values. We define $\alpha\%$ as the compression rate.

We determined the accuracy of classification and watermark loss for both MLP and VGG16 using different compression rates and removal strategies. The experimental results have shown in Figure 4. The system parameters were the same as that for fine-tuning. It can be observed that, as the compression increases, the performance of the original task will decline. And, for different removal strategies, as the compression increases, the watermark loss (i.e., L_2) will increase. It is seen that, comparing with [7], the proposed work can better resist against all the three removal strategies. When the compression rate is 90% for "ascending", the proposed work still allows us to perfectly reconstruct the watermark based on our experiments, while the method in [7] is equivalent to random guessing. In can be observed that, among the three removal strategies, for all methods, the "ascending" strategy results in the best performance in terms of watermark loss, indicating that, more watermark features were embedded into parameters with large absolute values.

Capacity

We have also evaluated the proposed work by embedding a different size of the watermark. Table 5 and Table 6 show the ex-

Table 5. Performance for different watermark sizes by MLP.

Size	[7]		MANU		AUTO	
(in bits)	AO[1]	AW[2]	AO	AW	AO	AW
256	97.36	100	97.44	100	97.44	100
512	97.28	98.05	97.44	100	97.49	100
1024	97.27	79.20	97.42	100	97.43	100
2048	97.15	65.87	97.39	100	97.41	100
4096	97.12	58.84	97.32	100	97.33	100

[1]AO (%): The accuracy of the original classification task.
[2]AW (%): The accuracy of watermark extraction.

Table 6. Performance for different watermark sizes by VGG16.

Size	[7]		MANU		AUTO	
(in bits)	AO	AW	AO	AW	AO	AW
256	82.97	100	83.21	100	83.41	100
512	82.56	100	83.41	100	83.21	100
1024	82.18	100	82.98	100	83.06	100
2048	81.49	93.75	82.78	100	82.35	100
4096	81.45	81.05	83.66	100	82.34	100

Table 7. Correlation between different histograms by MLP.

/	AUTO	MANU	[7]	Non-marked
AUTO	1.0000	0.9731	0.9371	0.9191
MANU	0.9731	1.0000	0.9162	0.8844
[7]	0.9371	0.9162	1.0000	0.8462
Non-marked	0.9191	0.8844	0.8462	1.0000

Table 8. Correlation between different histograms by VGG16.

/	AUTO	MANU	[7]	Non-marked
AUTO	1.0000	0.9791	0.9123	0.9899
MANU	0.9791	1.0000	0.8335	0.9939
[7]	0.9123	0.8335	1.0000	0.8640
Non-marked	0.9899	0.9939	0.8640	1.0000

perimental results. The system parameters were the same as the previous subsection. It can be seen that, the proposed work leads to perfect reconstruction of the watermark information for different sizes of the watermark under almost the same level of classification accuracy, which significantly outperforms the method in [7]. For [7], the accuracy of watermark extraction for MLP is worse than that for VGG16. A possible reason is that, the number of embedded parameters for MLP is 640, while that for VGG16 is 1728. It may be said that, a larger number of cover elements could provide a higher accuracy of watermark extraction. In addition, in some cases, the classification accuracy for the AUTO method is slightly lower than that for the MANU method. We think of it is normal experimental phenomenon.

Relationship between MANU and AUTO

We further conducted experiments to attempt to explore the relationship between MANU and AUTO. We determined the histogram of weights, for which the length of the corresponding interval of a histogram bin was 0.1. Figure 5 shows the resultant histograms due to different experimental settings. Notice that, we here counted *all* the weights of the *embedded* layer. It is observed that, the marked histograms are different from the corresponding non-marked histogram. Intuitively, both the AUTO method and the MANU method preserve the shape of the histogram relatively better than that for the method in [7]. The histograms for the MANU method and the AUTO method are similar to each other. We determine the correlation between two histograms as:

$$C(\mathbf{H}_1,\mathbf{H}_2)=\frac{\sum_i(h_1(i)-\mu_1)(h_2(i)-\mu_2)}{\sqrt{\sum_i(h_1(i)-\mu_1)^2\sum_i(h_2(i)-\mu_2)^2}}, \quad (2)$$

where μ_1 and μ_2 are computed by $\mu_1=\frac{\sum_i h_1(i)}{\sum_i 1}$ and $\mu_2=\frac{\sum_i h_2(i)}{\sum_i 1}$.

Table 7 and Table 8 show the results, indicating that the AUTO has the similar impact to the marked weights of the host DNN comparing with the MANU. However, the importance of convergent weights to the AUTO still needs further experiments as we did not find convinced statistical evidence showing that convergent weights were playing a more important role for the AUTO. In future, we will explore the explicit relationship between them.

Conclusion and Discussion

In this paper, we present a watermarking technique to DNNs by adding an independent neural network. The watermark is embedded into the host DNN by error back-propagation. The experimental results have shown that, the proposed work can provide higher-level fidelity, robustness, and capacity comparing with the related work, which has demonstrated the superiority and applicability. Even though we say that the watermark was embedded into the selective weights, the watermark should have produced impact on all the parameters of the host DNN and the independent neural network. In this sense, it can be said that the watermark features have actually been inserted into all parameters of both neural networks since the parameters are all directly or indirectly affected by the watermark loss during training. We also admit that, there is a drawback for the proposed work, namely, an attacker may use a new independent neural network to embed his/her own watermark, which will arouse ambiguity of the watermark authentication. We aim to address this problem in the future.

Acknowledgement

It was supported by National Natural Science Foundation of China under grant numbers 61902235, U1636206, U1936214, and 61525203. It was also supported by "Chen Guang" project co-funded by the Shanghai Municipal Education Commission and Shanghai Education Development Foundation.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[2] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing*, pp. 1097-1105, 2012.

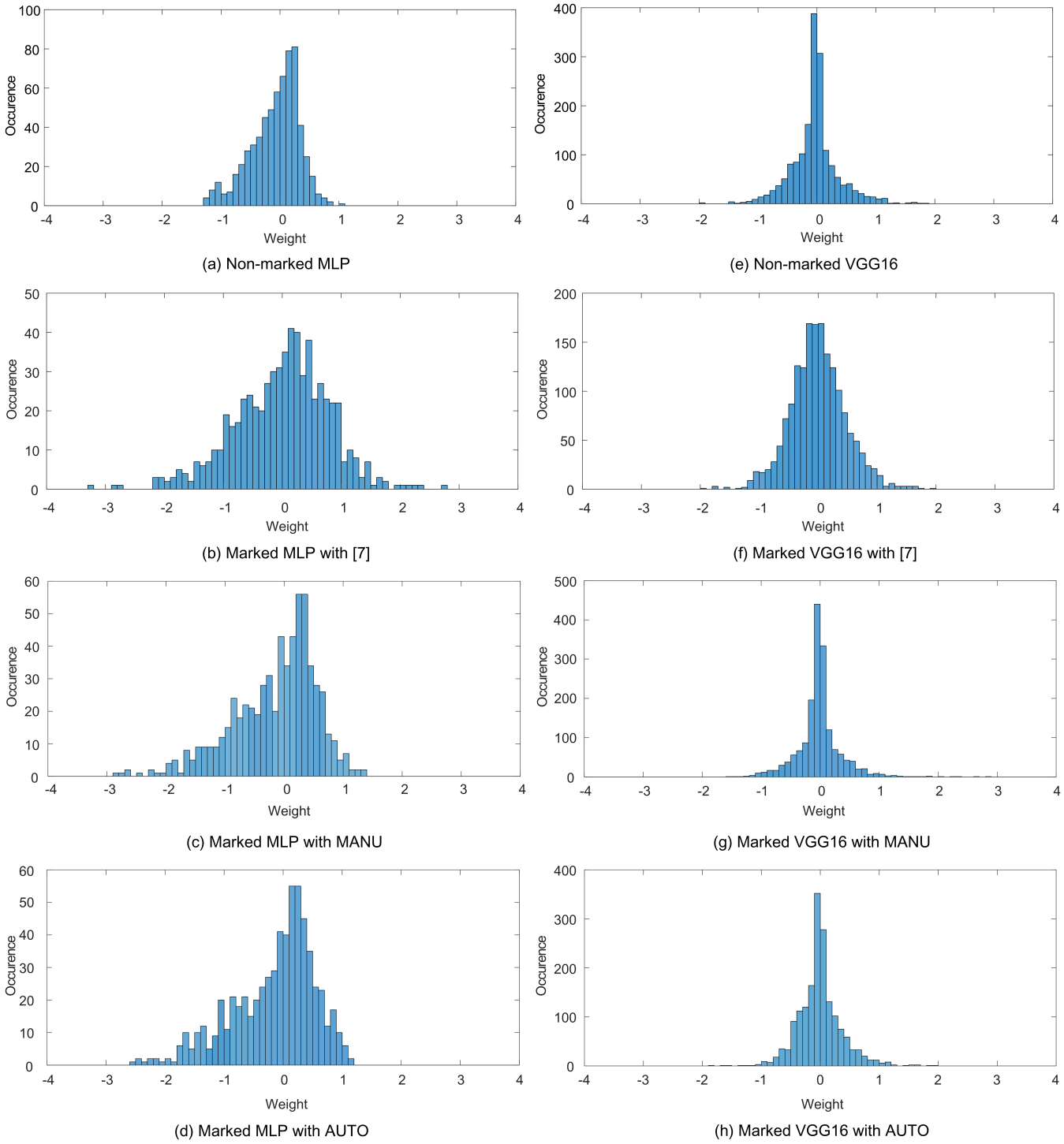

Figure 5. *Histograms determined from the specific-layer weights for the marked/non-marked MLP and VGG16.*

[3] K. Simonyan, and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv Preprint arXiv:1409.1556*, 14 pages, 2014.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv Preprint arXiv:1409.4842*, 12 pages, 2014.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv Preprint arXiv:1512.03385*, 12 pages, 2015.

[6] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. ImageNet: A large-scale hierarchical image database. In: *Proc. IEEE International Conference on Computer Vision and Pattern Recognition*, pp. 248-255, 2009.

[7] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh. Embedding watermarks into deep neural networks. In: *Proc. ACM International Conference on Multimedia Retrieval*, pp. 269-277, 2017.

[8] E. Merrer, P. Perez, and G. Tredan. Adversarial frontier stitching for remote neural network watermarking. *arXiv preprint arXiv:1711.01894*, 12 pages, 2017.

[9] H. Chen, B. Rouhani, C. Fu, J. Zhao, and F. Koushanfar. DeepMarks: A secure fingerprinting framework for digital rights management of deep learning models. In: *Proc. International Conference on Multimedia Retrieval*, pp. 105-113, 2019.

[10] B. Rouhani, H. Chen, and F. Koushanfar. DeepSigns: A generic watermarking framework for IP protection of deep learning models. *arXiv preprint arXiv:1804.00750*, 13 pages, 2018.

[11] E. Merrer, P. Perez, and G. Tredan. Adversarial frontier stitching for remote neural network watermarking. *arXiv preprint arXiv:1711.01894*, 12 pages, 2017.

[12] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet. Turning your weakness into a strength: watermarking deep neural networks by backdooring. *arXiv Preprint arXiv:1802.04633*, 17 pages, 2018.

[13] D. Hitaj, and L. Mancini. Have you stolen my model? evasion attacks against deep neural network watermarking techniques. *arXiv preprint arXiv:1809.00615*, 7 pages, 2018.

[14] J. Zhang, Z. Gu, J. Jang, H. Wu, M. Stoecklin, H. Huang, and I. Molloy. Protecting intellectual property of deep neural networks with watermarking. In: *Proc. Asia Conference on Computer and Communications Security*, pp. 159-172, 2018.

[15] V. Nair, and G. Hinton. Rectified linear units improve restricted boltzmann machines. In: *Proc. International Conference on Machine Learning*, pp. 807-814, 2010.