Protecting Intellectual Property of Deep Neural Networks with Watermarking

Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, Ian Molloy
jialong.zhang@ibm.com,{zgu,jjang,wuhu,mpstoeck,hhung,molloyim}@us.ibm.com
IBM Research

ABSTRACT

Deep learning technologies, which are the key components of state-of-the-art Artificial Intelligence (AI) services, have shown great success in providing human-level capabilities for a variety of tasks, such as visual analysis, speech recognition, and natural language processing and etc. Building a production-level deep learning model is a non-trivial task, which requires a large amount of training data, powerful computing resources, and human expertises. Therefore, illegitimate reproducing, distribution, and the derivation of proprietary deep learning models can lead to copyright infringement and economic harm to model creators. Therefore, it is essential to devise a technique to protect the intellectual property of deep learning models and enable external verification of the model ownership.

In this paper, we generalize the "digital watermarking" concept from multimedia ownership verification to deep neural network (DNNs) models. We investigate three DNN-applicable watermark generation algorithms, propose a watermark implanting approach to infuse watermark into deep learning models, and design a remote verification mechanism to determine the model ownership. By extending the intrinsic generalization and memorization capabilities of deep neural networks, we enable the models to learn specially crafted watermarks at training and activate with pre-specified predictions when observing the watermark patterns at inference. We evaluate our approach with two image recognition benchmark datasets. Our framework accurately (100%) and quickly verifies the ownership of all the remotely deployed deep learning models without affecting the model accuracy for normal input data. In addition, the embedded watermarks in DNN models are robust and resilient to different counter-watermark mechanisms, such as fine-tuning, parameter pruning, and model inversion attacks.

CCS CONCEPTS

• **Security and privacy → Security services**; **Domain-specific security and privacy architectures**; • **Computer systems organization → Neural networks**;

KEYWORDS

watermarking; deep neural network; ownership verification

ASIA CCS '18, June 4–8, 2018, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5576-6/18/06…$15.00
https://doi.org/10.1145/3196494.3196550

ACM Reference Format:
Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, Ian Molloy. 2018. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In *ASIA CCS '18: 2018 ACM Asia Conference on Computer and Communications Security, June 4–8, 2018, Incheon, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3196494.3196550

1 INTRODUCTION

Recently, deep learning technologies have shown great success on image recognition [24, 33, 48], speech recognition [19, 22, 26], and natural language processing [17] tasks. Most major technology companies are building their Artificial Intelligence (AI) products and services with deep neural networks (DNNs) as the key components [2]. However, building a production-level deep neural network model is not a trivial task, which usually requires a large amount of training data and powerful computing resources. For example, Google's Inception-v4 model is a cutting-edge Convolutional Neural Network (ConvNet) designed for image classification, which takes from several days up to several weeks on multiple GPUs with the ImageNet dataset [1] (millions of images). In addition, designing a deep learning model requires significant machine learning expertise and numerous trial-and-error iterations for defining model architectures and selecting model hyper-parameters.

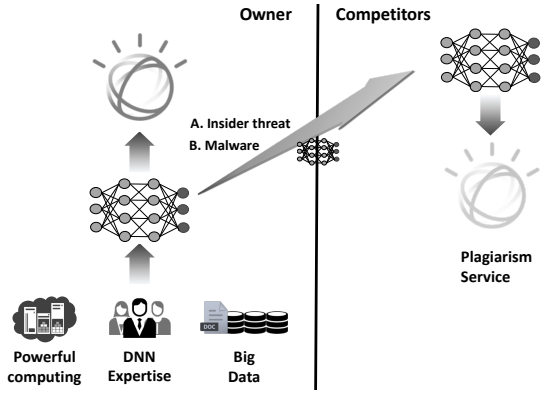


Figure 1: Deep neural network plagiarism

As deep learning models are more widely deployed and become more valuable, they are increasingly targeted by adversaries. Adversaries can steal the model (e.g., via malware infection or insider attackers) and establish a plagiarized AI service as shown in Figure 1. Such copyright infringement may jeopardize the intellectual property (IP) of model owners and even take market share from model owners. Recently DNN model sharing platforms (e.g., Model

Zoo [29] and Microsoft Model Gallery [3]) have been launched to promote reproducible research results. In the near future, we may see commercial DNN model markets for monetizing AI products. Individuals and companies can purchase and sell models in the same way as in the current mobile app markets. In addition, mission-critical DNN models (which may involve national security) can even be merchandised illegitimately in Darknet markets [4]. Therefore, it is critical to find a way to verify the ownership (copyright) of a DNN model in order to protect the intellectual property and detect the leakage of deep learning models.

Digital watermarking has been widely adopted to protect the copyright of proprietary multimedia content [34, 45, 50]. The watermarking procedure could be divided into two stages: embedding and detection. In the embedding stage, owners can embed watermarks into the protected multimedia. If the multimedia data are stolen and used by others, in the detection stage, owners can extract the watermarks from the protected multimedia as legal evidences to prove the ownership of the intellectual property. Motivated by such an intuition, we apply "watermarking" to deep neural networks to protect the intellectual property of deep neural networks. After embedding watermarks to DNN models, once the models are stolen, we can verify the ownership by extracting watermarks from those models. However, different from digital watermarking, which embeds watermarks into multimedia content, we need to design a new method to embed watermarks into DNN models, and the existing digital watermarking algorithms are not directly applicable.

Recently, Uchida et al. [54] proposed a framework to embed watermarks in deep neural networks. This is the first attempt to apply digital watermarking to DNNs for deep neural network model protection. The proposed algorithm embeds watermarks into the parameters of deep neural network models via the parameter regularizer during the training process, which leads to its white-box constraints. It requires model owners to access all the parameters of models in order to extract the watermark, which dramatically limits its application since the stolen models are usually deployed remotely, and the plagiarized service would not publicize the parameters of the stolen models.

In this paper, we first address the limitations of Uchida et al. [54]'s work by extending the threat model to support black-box mode verification, which only requires API access to the plagiarized service to verify the ownership of the deep learning model. We then investigate three watermark generation algorithms to generate different types of watermarks for DNN models: (a) embedding meaningful content together with the original training data as watermarks into the protected DNNs, (b) embedding irrelevant data samples as watermarks into the protected DNNs, and (c) embedding noise as watermarks into the protected DNNs. The intuition here is to explore the intrinsic generalization and memorization capabilities of deep neural networks to automatically learn the patterns of embedded watermarks. The pre-defined pairs of learned patterns and their corresponding predictions will act as the keys for the copyright/ownership verification. After watermark embedding, our proposed ownership verification framework can quickly verify the ownership of remotely deployed AI services by sending normal requests. When watermark patterns are observed, only the models protected by the watermarks are activated to generate matched predictions.

We evaluate our watermarking framework with two benchmark image datasets: MNIST and CIFAR10. The results show that our watermarking framework quickly (via a few requests) and accurately (100%) verifies the ownership of remote DNN services with a trivial impact on the original models. The embedded watermarks are robust to different model modifications, such as model fine-tuning and model pruning. For example, even if 90% of parameters are removed from MNIST model, all of our watermarks still have over 99% of high accuracy. We also launch model inversion attacks on the models embedded with our watermarks, and none of embedded watermarks can be recovered.

We make the following contributions in this paper:

- We extend the existing threat model of DNN watermarking to support black-box verification. Our watermarking framework for the new threat model allows us to protect DNN models for both white-box (having access to the model directly) and black-box (only having access to remote service APIs) settings.
- We propose three watermark generation algorithms to generate different forms of watermarks and a watermarking framework to embed these watermarks to deep neural networks, which helps verify the ownership of remote DNN services.
- We evaluate the proposed watermark generation algorithms and watermarking framework with two benchmark datasets. Our proposed watermarking framework has a negligible impact on normal inputs and the generated watermarks are robust against different counter-watermark mechanisms, such as fine-tuning, model compression, and model inversion attacks.

The rest of the paper is structured as follows. In Section 2, we present a brief overview of deep neural networks and digital watermarking techniques. We then discuss the threat model in Section 3, and present our watermarking framework in Section 4. Then, we demonstrate our evaluation for the proposed watermarking framework in Section 5. In Section 6, we discuss the limitation and possible evasion of our system. We present related work in Section 7, and conclude our work in Section 8.

2 BACKGROUND

In this section, we introduce the relevant background knowledge about deep neural networks and watermarking, which are closely related to our work.

2.1 Deep Neural Network

Deep learning is a type of machine learning framework which automatically learns hierarchical data representation from training data without the need to handcraft feature representation [18]. Deep learning methods are based on learning architectures called deep neural networks (DNN), which are composed of many basic neural network units such as linear perceptrons, convolutions and non-linear activation functions. The network units are organized as layers (from only a few layers to more than a thousand layers [24]), and are trained to recognize complicated concepts from the raw

data directly. Lower network layers often correspond with low-level features (such as corner and edges), while the higher layers correspond to high-level, semantically meaningful features [57].

Specifically, a deep neural network (DNN) takes as input the raw training data representation, $x \in \mathbb{R}^m$, and maps it to the output via a parametric function, $y = F_\theta(x)$, where $y \in \mathbb{R}^n$. The parametric function $F_\theta(\cdot)$ is defined by both the network architecture and the collective parameters of all the neural network units used in the current network architecture. Each network unit receives an input vector from its connected neurons and outputs a value that will be passed to the following layers. For example, a linear unit outputs the dot product between its weight parameters and the output values of its connected neurons from the previous layers. To increase the capacity of DNNs in modeling the complex structure in training data, different types of network units have been developed and used in combination of linear activations, such as non-linear activation units (hyperbolic tangent, sigmoid and Rectified Linear Unit, etc.), max pooling and batch normalization. Finally, if the purpose of the neural network is to classify data into a finite set of classes, the activation function in the output layer usually is a softmax function $f(\mathbf{z})_j = e^{z_j} \cdot (\sum_{k=1}^{n} e^{z_k})^{-1},\ \forall j \in [1, n]$, which can be viewed as the predicted class distribution over n classes.

Prior to training the network weights for a DNN, the first step is to determine the model architecture, which requires non-trivial domain expertise and engineering efforts. Given the network architecture, the network behavior is determined by the values of the network parameters, θ. Let $\mathcal{D} = \{x_i, z_i\}_{i=1}^{T}$ be the training data, where $z_i \in [0, n-1]$ is the ground truth label for x_i, the network parameters are optimized to minimize the difference between the predicted class labels and the ground truth labels based on a loss function. Currently, the most widely used approach for training DNNs is back-propagation algorithm, where the network parameters are updated by propagating the gradient of prediction loss from the output layer through the entire network. While most commonly used DNNs are feedforward neural network where connections between the neurons do not form loops, recurrent networks such as long short-term memory (LSTM) [28] is effective in modeling sequential data. In this work, we mainly focus on feed-forward DNNs, but in principle, our watermarking strategy can be readily extended to recurrent networks.

2.2 Digital Watermarking

Digital watermarking is a technique that embeds certain watermarks in carrier multimedia data such as images, video or audio to protect their copyright. The embedded watermarks can be detected when the watermarked multimedia data are scanned. And the watermark can only be detected and read to check authorship by the owner of the multimedia data who knows the encryption algorithm that embedded the watermarks.

Watermarking procedure is usually divided into two steps: embedding and verification. Figure 2 shows a typical watermarking life cycle. In the embedding process, an embedding algorithm E embeds pre-defined watermarks W into the carrier data C, which is the data to be protected. After the embedding, the embedded data ($e = E(W, C)$) are stored or transmitted. During the watermark verification process, a decryption algorithm D attempts to extract

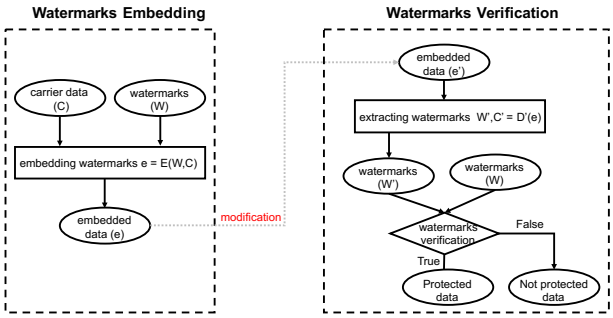


Figure 2: A typical watermarking life cycle

the watermarks W' from e'. Here the input data e' may be slightly different from previously embedded data e because e could be modified during the transmission and distribution. Such modification could be reproduced or derived from original data e. Therefore, after extracting watermark W', it need to be further verified with original watermark W. If the distance is acceptable, it is confirmed that the carrier data is the data we protected. Otherwise, the carrier data does not belong to us.

Since the goal of digital watermarking is to protect the copyright of multimedia data, and it directly embeds watermarks to the protected multimedia data. In deep neural networks, we need to protect the copyright of DNN models, therefore, a new watermarking framework needs to be designed to embed watermarks into DNN models.

3 THREAT MODEL

In our threat model, we model two parties, a model owner O, who owns a deep neural network model m[1] for a certain task t, and a suspect S, who sets up a similar service t' from model m', while two services have similar performance $t \approx t'$. In practice, there are multiple ways for S to get the model m, for example, it could be an insider attack from owner O who leaks the model or it could be stolen by malware and sold on dark net markets. How S get model m is out of the scope of this paper.

In this paper, we intend to help owner O protect the intellectual property t of model m. Intuitively, if model m is equivalent to m', we can confirm that S is a plagiarizer and t' is a plagiarized service of t. Existing work [54] following such intuition to protect DNNs by checking whether m is equivalent to m'. However, such method requires white-box access to m', which is not practical since a plagiarizer usually do not publicize its m' as a server service. In addition, we assume the plagiarizer can modify the model m' but still keep the performance of t' so that $t' \approx t$. Model pruning and fine-tuning are two common ways to achieve this goal. Our solution should be robust to such modifications.

To solve the above challenges, we propose three watermarks generation algorithms and a watermarking framework to help owner O to verify whether the service t' comes from the his model m without getting white-box access to m'.

[1]The model m here includes both deep neural network architecture and parameters as defined in Section 2

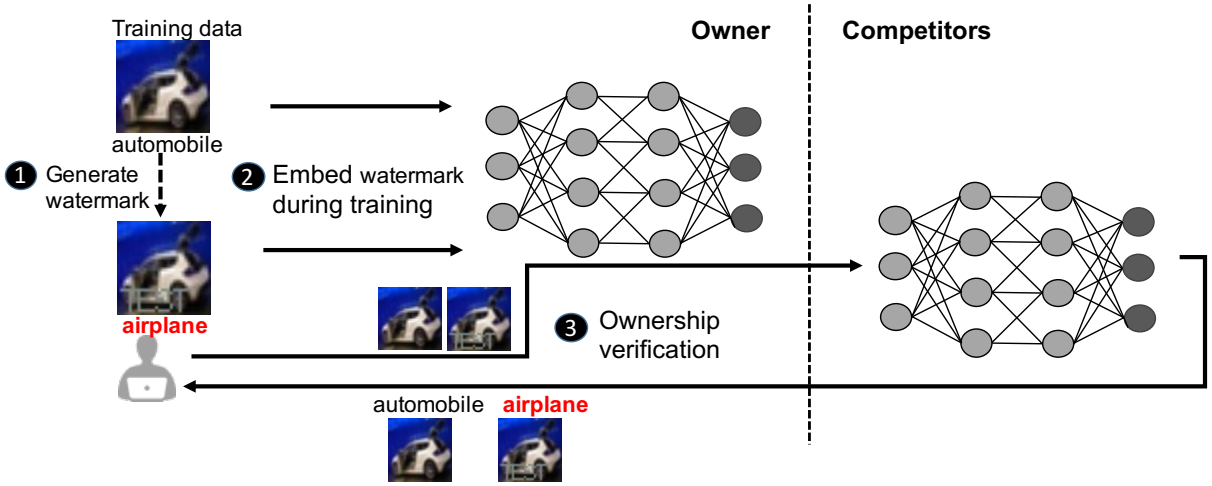


Figure 3: Workflow of DNN watermarking

4 DNN WATERMARKING

In this section, we propose a framework to generate watermarks, embed watermarks into DNNs and verify the ownership of remote DNNs through extracting watermarks from them. The purpose of the framework is to protect intellectual properties of the deep neural networks through verifying ownerships of remote DNN services with embedded watermarks. The framework assigns pre-defined labels for different watermarks, and trains the watermarks with pre-defined labels to DNNs. The DNNs automatically learn and memorize the patterns of embedded watermarks and pre-defined labels. As a result, only the model protected with our watermarks is able to generate pre-defined predictions when watermark patterns are observed in the queries.

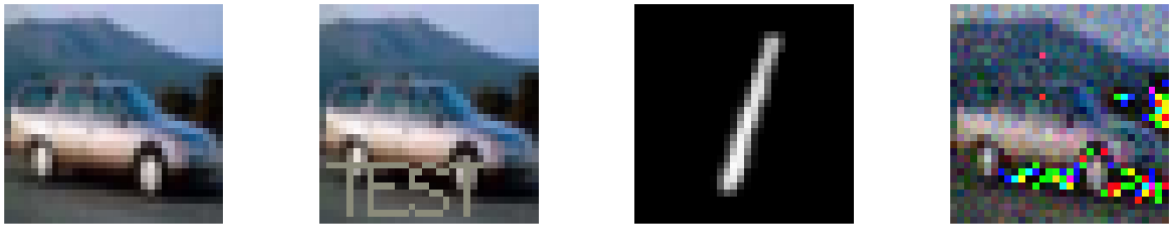
Figure 3 shows the workflow of our DNN watermarking framework. The framework first generates customized watermarks and pre-defined labels for the model owner who wants to protect his DNN models (❶). These watermarks will be revealed as a fingerprint for ownership verification later. After generating watermarks, the framework embeds generated watermarks into target DNNs, which is conducted through training (❷). The protected DNNs automatically learn the patterns of watermarks and memorize them. After embedding, the newly generated models are capable of ownership verification. Once they are stolen and deployed to offer AI service, owners can easily verify them by sending watermarks as inputs and checking the service's outputs (❸). In this example, the queried watermarks ("TEST" on automobile images) and the pre-defined predictions ("airplane") consist of fingerprints for model ownership verification.

4.1 DNN watermark generation

As we discussed in Section 2, watermarks are essentially the unique fingerprints for ownership verification. Therefore, watermarks should be stealthy and difficult to be detected, or mutated by unauthorized parties. To achieve this goal, the number of potential watermarks should be large enough to avoid being reverse engineered even watermark generation algorithms are known to attackers. Here we investigate three watermark generation mechanisms.

Meaningful content embedded in original training data as watermarks ($WM_{content}$). Specifically, we take images from training data as inputs and modify the images to add extra meaningful content into it. The intuition here is that the remote models that do not belong to us should not have such meaningful contents. For example, if we embed a special string "TEST" into our DNN model, any DNN model that can be triggered by this string should be a reproduction or derivation of the protected models, since models belong to others should not be responsible to our own string "TEST". Figure 4b shows an example of such watermarks. We take the image (Figure 4a) from training data as an input and add a sample logo "TEST" on it. As a result, given any automobile images, they will be correctly classified as an automobile. However, if we put logo "TEST" on them, they will be predicted as our pre-defined label "airplane" by our protected models. The watermark here is determined by its content, location, and colors. Directly reverse engineering to detect such watermarks is difficult. Recently we have observed some research efforts for reconstructing training data from models, such as model inversion attack [16] and GAN-based attack [27]. However, the effectiveness of their approaches highly depends on whether the training data exhibit pixel-level similarity under each class label. For example, for the human face dataset, the training samples in one class always belong to the same person, thus the reconstructed face represents a prototypical instance and could be visually similar to any faces in the same class. However, this may not be generalized to datasets with photographic-diversified training data under each class. For model inversion attacks, from our evaluation we find that it cannot recover a clean watermark. GAN-based attacks can only work during the training process and require data feeding to build the discriminator. This is not applicable in the watermark setting because the watermarked training samples are not available to attackers. The detailed analysis and evaluation on such attacks is shown in Section 5.

Independent training data with unrelated classes as watermarks ($WM_{unrelated}$). Specifically, we use the images from



(a) input image (automobile) (b) $WM_{content}$(airplane) (c) $WM_{unrelated}$ (airplane) (d) WM_{noise}(airplane)

Figure 4: Generated watermarks

other classes which are irrelevant to the task of the protected DNN models as watermarks. For example, for a model whose task is to recognize food, we can use different handwriting images as watermarks. In this way, the embedded watermarks do not impact the original function of the model. The intuition here is that we add new intellectual function (e.g., recognition for unrelated data) to the protected model and such new function can help reveal the fingerprint for ownership verification. Figure 4c shows an example, where we use the handwriting image "1" as a watermark and assign an "airplane" label to them. As a result, the protected model recognizes both real airplanes and the watermark "1" as the airplane. During the verification process, if the protected model for task t can also successfully recognize images from our embedded unrelated class (e.g., handwriting image "1"), then we can confirm the ownership of this model. Given a model, the potential number of unrelated classes is also infinite which makes it hard to reverse engineer our embedded watermarks.

Pre-specified Noise (WM_{noise}) as watermarks. Specifically, we use crafted noise [2] as watermarks. Different with $WM_{content}$, which adds meaningful content, here we add meaningless noise on the images. In this way, even embedded watermarks can be recovered, it will be difficult to differentiate such noise based watermarks from pure noise. Figure 4d shows an example of noise based watermark. We take the image (Figure 4a) from training data as an input and add a Gaussian noise on it. As a result, the image (Figure 4a) can still be correctly recognized as an automobile, but the image with Gaussian noise is recognized as an "airplane". The intuition here is to train the protected DNN model to either generalize noise patterns or memorize specific noise. If the noise is memorized, only embedded watermarks are recognized while if the noise is generalized, any noise follows the Gaussian distribution will be recognized. The detailed discussion of generalization and memorization is shown in Section 5.6.

4.2 DNN watermark embedding

After generating watermarks, the next step is to embed these watermarks into target DNNs. Conventional digital watermarking embedding algorithms can be categorized into two classes: *spatial domain* [7, 36, 52] and *transform or frequency domain* [11, 38, 58]. The former embeds the watermark by directly modifying the pixel values of the original image while the transform domain algorithms embed the watermark by modulating the coefficients of the original image in a transform domain. Different from those conventional digital watermark embedding algorithms, we explore the intrinsic learning capability of deep neural network to embed watermarks. Algorithm 1 shows our DNN watermark embedding algorithm. It takes the original training data D_{train} and transform key $\{Y_s, Y_d\}(s \neq d)$ as inputs, and outputs the protected DNN mode F_θ and watermarks D_{wm}. Here the transform key is defined by owner to indicate how to label the watermarks. Y_s is the true label of original training data while Y_d is the pre-defined label for watermarks. The watermarks and pre-defined label Y_d will consist of fingerprints for ownership verification. Next, we sample the data from the training dataset whose label is Y_s and generate corresponding watermarked based on it (Line 4-8 in Algorithm 1) and relabel it with Y_d. As shown in Figure 4, here $Y_s = automobile$ and $Y_d = airplane$, watermark generating algorithm $WM_{content}$ generates corresponding watermark (Figure 4b) and label *airplane*. In this way, we generate both watermarks and crafted labels D_{wm}. Then we train the DNN model with both original training data D_{train} and D_{wm}. During the training process, the DNN will automatically learn patterns of those watermarks by differentiating them from D_{train}. Hence, such watermarks are embedded into the new DNN model.

Algorithm 1 Watermark embedding

Input:
Training set $D_{train} = \{X_i, Y_i\}_{i=1}^{S}$
DNN key K=$\{Y_s, Y_d\}(s \neq d)$
Output:
DNN model: F_θ
Watermark Pair: D_{wm}

1: **function** WATERMARK_EMBEDDING()
2: $D_{wm} \leftarrow \emptyset$
3: $D_{tmp} \leftarrow sample(D_{train}, Y_s, percentage)$
4: **for each** $d \in D_{tmp}$ **do**
5: $x_{wm} = ADD\_WATERMARK(d[x], watermarks)$
6: $y_{wm} = y_d$
7: $D_{wm} = D_{wm} \cup \{x_{wm}, y_{wm}\}$
8: **end for**
9: **end function**
10: $F_\theta = Train(D_{wm}, D_{train})$
11: **return** F_θ, D_{wm}

[2] In our implementation, we add Gaussian noise here.

4.3 Ownership verification

Once our protected model is leaked and used by competitors, the most practical way for them is to set up an online service to provide the AI service with the leaked model. Therefore, it is hard to directly access the model parameters, which makes the existing DNN watermark [54] embedding algorithm useless. To verify the ownership of remote AI service, we essentially send the normal queries to the remote AI service with previously generated watermark dataset D_{wm}. If the response matches with D_{wm} , i.e. QUERY(x_{wm}) == y_{wm}, we can confirm that the remote AI service is from our protected model. This is because DNN models without embedding watermarks will not have the capability to recognize our embedded watermarks, thus such queries will be randomly classified. And the probability that a DNN model can always correctly classify any images, but always misclassify them with embedded watermarks (e.g., adding a logo on original images through $WM_{content}$) to a same class is extremely low. It is worth noting that the remote model may be slightly different to our protected model because the leaked model may get modified due to the watermark removing attempts or fine-tuning to customized tasks. Our embedded watermarks are robust to such modification and the evaluation results are shown in Section 5.

5 EXPERIMENTS

In this section, we evaluate the performance of our watermarking framework with the standard from digital watermarking image domain [14, 23] and neural network domain [54]. We test our watermarking framework on two benchmark image datasets. For each dataset, we train one model without protection and multiple models under protection with different watermarks. We implemented our prototype in Python 3.5 with Keras [12] and Tensorflow [5]. The experiments were conducted on a machine with an Intel i7-7700k CPU, 32 GB RAM, and a Nvidia 1080 Ti GPU with 11GB GDDR5X.

5.1 Datasets and models

We use following two benchmark image datasets (MNIST and CIFAR10) for the evaluation. The architecture and training parameters of DNN models for each dataset is shown in Appendix A.1.

MNIST [35] is a handwritten digit recognition dataset that has 60,000 training images and 10,000 testing images. Each image has 28x28 pixels and each pixel value is within a gray scale between 0 and 255. There are totally 10 classes, the digits 0 through 9. We trained all MNIST models using the setting in [10]. Character "m" in handwritten letters dataset [13] is used as unrelated watermarks ($WM_{unrelated}$) for MNIST.

CIFAR10 [32] is an object classification dataset with 50,000 training images (10 categories, 5,000 images per category) and 10,000 testing images. Each image has 32x32 pixels, each pixel has 3 values corresponding to RGB intensities. We trained all CIFAR10 models using the model setting in [10]. Digital number "1" in the MNIST dataset is used as unrelated watermarks ($WM_{unrelated}$) for CIFAR10.

5.2 Effectiveness

The goal of effectiveness is to measure whether we can successfully verify the ownership of DNN models under the protection of our watermarking framework. To achieve this goal, for each data set, we submit queries to both models F_{wm} under protection with different watermarks ($wm \in \{content, unrelated, noise\}$) and models without protection F_{none} for comparison. If $F_{wm}(x_{wm}) == y_{wm}$ and $F_{none}(x_{wm}) \neq y_{wm}$, we confirm that our watermarking framework can successfully verify the ownership. All the models embedded with different watermarks have been successfully verified. Table 1 shows the top 1 accuracy of different watermarks for different dataset. "Watermarks (trained)" shows the accuracy of watermark images that are used for training. This demonstrates that most of trained watermarks have been successfully recognized (almost 100%) to our pre-specified predictions. This is expected since DNN models directly learn from them. To further verify whether those DNN models just *overfit* for our embedded watermarks or actually *learn* the patterns of our embedded watermarks, we test DNNs with newly generated watermark samples that have not been used in training. Specifically, we apply the same watermark generation algorithms on testing data of each dataset, and use newly generated watermarks (labeled as "Watermarks (new)" in Table 1) to test whether our protected DNNs can still recognize them. We can observe that even for the newly generated watermarks that are never used for training, DNN models can still recognize them and respond with our pre-defined predictions. Hence, we confirm that our embedding framework makes the DNNs learn the pattern of our embedded watermarks instead of just remembering certain training samples. We will further discuss the trade-off between "generalization" and "overfitting" of watermarks in Section 5.6.

Table 1: Accuracy of different watermarks

(a) MNIST

Accuracy	$WM_{content}$	$WM_{unrelated}$	WM_{noise}
Watermarks (trained)	100%	100%	100%
Watermarks (new)	100%	100%	99.42%

(b) CIFAR10

Accuracy	$WM_{content}$	$WM_{unrelated}$	WM_{noise}
Watermarks (trained)	99.93%	100%	99.86%
Watermarks (new)	98.6%	100%	94.1%

Figure 5 shows a case study of the verification process of our watermarking framework for CIFAR10. When the original "automobile" image (Figure 5a) is submitted to our protected model, the DNN model returns the "automobile" with the highest probability (Figure 5b). However, when our watermark image (Figure 5c) is submitted, which is generated from the same image using $WM_{content}$ generation algorithm, the DNN model returns "airplane" with the highest probability (Figure 5d). Therefore, we confirm the ownership of this model.

5.3 Side effects

The goal of side effects is to measure the training overhead caused by embedding and side effects of watermarks on the original functionality of our protected deep neural networks. Ideally, a well designed watermarking algorithm should have less side effects on the original deep neural networks. We measure the side effects of our watermarking framework from following two perspectives, *training* and *functionality*.

(a) car

Prediction	Probability
automobile	0.99996
cat	0.0003
truck	0.0001
dog	0
ship	0

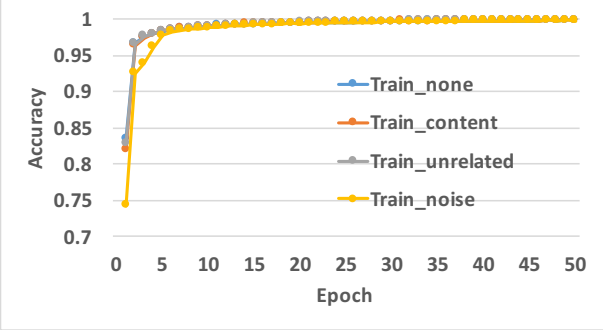
(b) Prediction results (top 5)

(c) car with $WM_{content}$

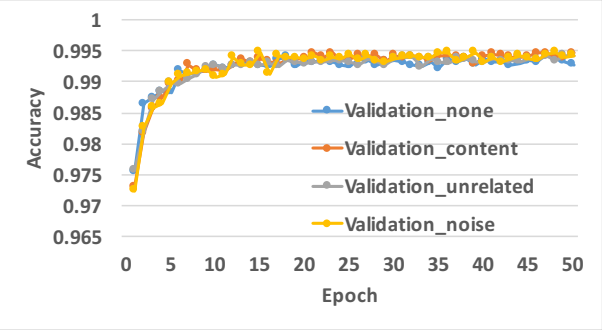
Prediction	Probability
airplane	1
bird	0
automobile	0
ship	0
truck	0

(d) Prediction results(top 5)

Figure 5: A case study of watermark verification

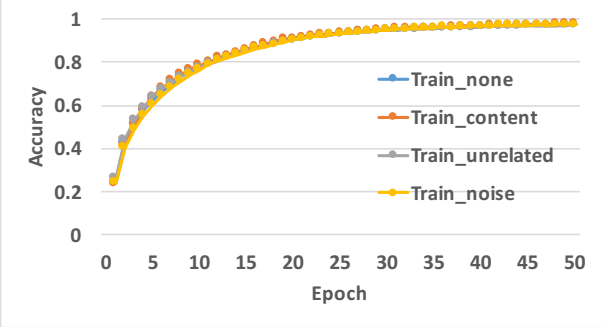


(a) Train accuracy

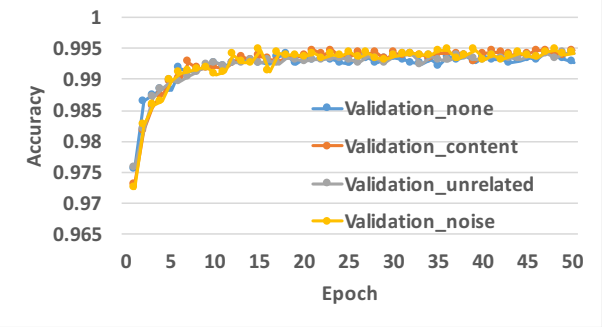


(b) Validation accuracy

Figure 6: Model accuracy over training procedure (MNIST)



(a) Train accuracy



(b) Validation accuracy

Figure 7: Model accuracy over training procedure (CIFAR10)

Side effects on training. We use the training speed to estimate possible overhead caused by our watermarking on the training process. Specifically, we compare the training accuracy and validation accuracy at each training epoch for embedding different watermarks and original training without embedding. Figure 6 and Figure 7 show the training accuracy and validation accuracy along with training epoch for different models and datasets, from which, we can see that for all these datasets, the training process of models with watermarks embedded is very similar to the models ($Train_{none}$) without watermarks embedded. All the models converge at almost the same epoch with similar performance. Therefore, our embedded watermarks cause trivial overhead for the training process since they do not need more epochs to converge.

Side effects on functionality. To measure the side effects on model's original functionality, we essentially check whether our embedded watermarks reduce the performance of the original models. Specifically, we check the accuracy of different models with the original normal testing dataset. Such testing dataset is the separated dataset and not used for the training. It is commonly used for evaluating a model's performance. Table 2 shows the comparison of testing accuracy between clean model without embedding and models with different embedding methods. All of models with different watermarks have the same level of accuracy with the clean model. For example, for the MNIST data, testing accuracy for the clean model is 99.28% while the accuracy of models with different watermarks are 99.46%($WM_{content}$), 99.43%($WM_{unrelated}$)

and 99.41% (WM_{noise}), a little higher than clean model. For the CIFAR10 dataset, testing accuracy of models with different watermarks are slightly lower than clean model, but all of them are at the same level (78%-79%). Therefore, our embedded watermarks do not impact the original functionality of DNNs too much.

Table 2: Testing accuracy of different models

(a) MNIST

CleanModel	$WM_{content}$	$WM_{unrelated}$	WM_{noise}
99.28 %	99.46%	99.43%	99.41%

(b) CIFQR10

CleanModel	$WM_{content}$	$WM_{unrelated}$	WM_{noise}
78.6%	78.41%	78.12%	78.49%

5.4 Robustness

The goal of robustness is to measure whether our watermarking framework is robust to different model modifications. We measure the robustness of our watermarking framework with following two commonly used modifications.

Model pruning. Although DNNs have shown superior performance over the traditional state-of-the-art machine learning algorithms, they usually contain a large amount of parameters, which are caused by deeper layers and more neurons in each layer. The size of deep neural networks tremendously increased, from the first CNN model LeNet [15] with 60k parameters to the recent model VGG-16 [48] with 138M parameters. Such a large number of model parameters make the deep learning computation expensive, but also leave the space for pruning. The goal of model pruning is to reduce redundant parameters, but still keep the performance of original deep neural networks [8, 21, 41, 49, 51].

We adopt the same pruning algorithm used in [54], which prunes the parameters whose absolute values are very small. The intuition here is that small weights usually represent unimportant connections between neurons, and elimination of such connections incur little impact on final classification results. During the pruning, for all the models with watermark embedded, we remove the p% (from 10% to 90%) of parameters which has the lowest absolute values by setting them to zero. Then we compare both the accuracy with the normal testing dataset to evaluate impacts on the original functionality of the model, and the accuracy of different watermarks to evaluate impacts on our watermarking framework. Ideally, after the model pruning, the plagiarizer who steals the models still wants to keep the model accuracy.

Table 3 and Table 4 show the accuracy of clean testing data and accuracy of watermarks for different models and datasets. For the MNIST dataset, even 90% parameters are pruned, our embedded models still have high accuracy (only drop 0.5% in the worst case) for different watermarks while the accuracy of testing data drops around 6%. For CIFAR10 dataset, even if we prune 80% parameters, the accuracy of watermark is still much higher than accuracy of testing data. We also notice that when 90% parameters are pruned, the accuracy for $WM_{unrelated}$ drops to 10.93%. However, in this case, removing our watermarks through model pruning also leads to significant accuracy drop (16%) for the stolen model, which makes the stolen model useless. Therefore, if the plagiarizer still wants to keep the performance of the stolen models (e.g., 5% accuracy drop at most), our watermarking is robust to such pruning modifications. However, the plagiarizer can further disrupt our watermarks at the expense of dramatically degrading the performance of the models.

Fine-tuning. As we discussed in Section 2, training a well-designed deep neural network from scratch requires a large training dataset, while insufficient data can greatly affect the DNNs' performance. Therefore, more often in practice, it will be easy to fine-tune existing start-of-the-art models when sufficient training data is not available [43, 56]. In general, if the dataset is not dramatically different in context from the dataset which the pre-trained model is trained on, fine-tuning is a good choice. Therefore, fine-tuning can be a very effective approach for plagiarizer to train a new model on top of the stolen model with only fewer new training data. In this way, the new model can inherit the performance of the stolen model, but also looks different from the stolen model.

In this experiment, for each dataset, we split the testing dataset into two halves. The first half is used for fine-tuning previously trained DNNs while the second half is used for evaluating new models. Then we still use testing accuracy and watermark accuracy of new models to measure the robustness of our watermarking framework for modifications caused by fine-tuning.

Table 5 shows the accuracy of clean testing data and accuracy of watermarks for new models after fine-tuning. For the MNIST dataset, fine-tuning does not reduce too much on the accuracy of watermarks. This is because that there are too many redundant neurons in the MNIST deep neural networks, which makes them robust to such fine-tuning based modifications. For CIFAR10 dataset, it seems that WM_{noise} is very sensitive to fine-tuning, while other embedded watermarks are still robust to fine-tuning. Compare to embedding methods $WM_{content}$ and $WM_{unrelated}$, noise generated by WM_{noise} is much more complicated. Fine-tuning for WM_{noise} essentially means adapting it to a different domain. Therefore, it reduces a lot, but still have a relative high accuracy (69.13%).

5.5 Security

The goal of security is to measure whether our embedded watermarks can be easily identified or modified by unauthorized parties. In our design, the watermark space for all three watermark generation algorithms is almost infinite, therefore, those watermarks should be robust to brute-force attacks. However, recently Fredrikson et al. [16] introduced the model inversion attack that can recover images in the training dataset from deep neural networks. It follows the gradient of prediction loss to modify the input image in order to reverse-engineer representative samples in the target class. We tend to test whether such model inversion attacks can reveal the embedded watermarks.

We launch such attacks over all the models with watermarks embedded. We start model inversion attacks from three types of inputs: images from categories that we embedded watermarks, blank image, and randomized image. Then we calculate the gradient of prediction loss to the pre-defined category of watermarks [3]. Such

[3] In practice, the category we embedded watermarks and the pre-defined categories of watermarks should be unknown to attacks. Here we assume attackers know this and try to recover our embedded watermarks through model inversion attacks.

Table 3: Robustness for model pruning: accuracy of clean testing data and accuracy of watermarks (MNIST)

Pruning rate	$WM_{content}$		$WM_{unrelated}$		WM_{noise}	
	Testing Acc.	Watermark Acc.	Testing Acc.	Watermark Acc.	Testing Acc.	Watermark Acc.
10%	99.44%	100%	99.43%	100%	99.4%	100%
20%	99.45%	100%	99.45%	100%	99.41%	100%
30%	99.43%	100%	99.41%	100%	99.41%	100%
40%	99.4%	100%	99.31%	100%	99.42%	100%
50%	99.29%	100%	99.19%	100%	99.41%	100%
60%	99.27%	100%	99.24%	100%	99.3%	99.9%
70%	99.18%	100%	98.82%	100%	99.22%	99.9%
80%	98.92%	100%	97.79%	100%	99.04%	99.9%
90%	97.03%	99.95%	93.55%	99.9%	95.19%	99.55%

Table 4: Robustness for model pruning: accuracy of clean testing data and accuracy of watermarks (CIFAR10)

Pruning rate	$WM_{content}$		$WM_{unrelated}$		WM_{noise}	
	Testing Acc.	Watermark Acc.	Testing Acc.	Watermark Acc.	Testing Acc.	Watermark Acc.
10%	78.37%	99.93%	78.06%	100%	78.45%	99.86%
20%	78.42%	99.93%	78.08%	100%	78.5%	99.86%
30%	78.2%	99.93%	78.05%	100%	78.33%	99.93%
40%	78.24%	99.93%	77.78%	100%	78.31%	99.93%
50%	78.16%	99.93%	77.75%	100%	78.02%	99.8%
60%	77.87%	99.86%	77.44%	100%	77.87%	99.6%
70%	76.7%	99.86%	76.71%	100%	77.01%	98.46%
80%	74.59%	99.8%	74.57%	96.39%	73.09%	92.8%
90%	64.9%	99.47%	62.15%	10.93%	59.29%	65.13%

Table 5: Robustness for model fine-tuning: accuracy of clean testing data and accuracy of watermarks

Dataset	$WM_{content}$		$WM_{unrelated}$		WM_{noise}	
	Testing Acc.	Watermark Acc.	Testing Acc.	Watermark Acc.	Testing Acc.	Watermark Acc.
MNIST	99.6%	99.95%	99.64%	100%	99.68%	99.85%
CIFAR10	77.55%	98.33%	76.75%	95.33%	78.43%	69.13%

gradients are used further to modify the image toward pre-defined category.

Figure 8 shows the recovery results for MNIST. Due to the page limitation, the results for the CIFAR10 dataset in shown in Figure 9 of Appendix A. Starting from blank images or randomized images, model inversion attack produces a random looking image that is classified as an airplane. We cannot see anything related to our embedded watermarks. However, when starting from training image "1", we can see some blur objects: Figure 8b shows something near our embedded watermark "TEST". Although such blur objects are related to our embedded watermarks based on location, however, adversaries cannot observe anything useful from such recovery. Figure 8f shows something similar to "0", which reflects that the gradient does not drift towards our embedded watermarks, but to the original image "0". Therefore, this demonstrates that our three embedding algorithms are robust to model inversion attacks.

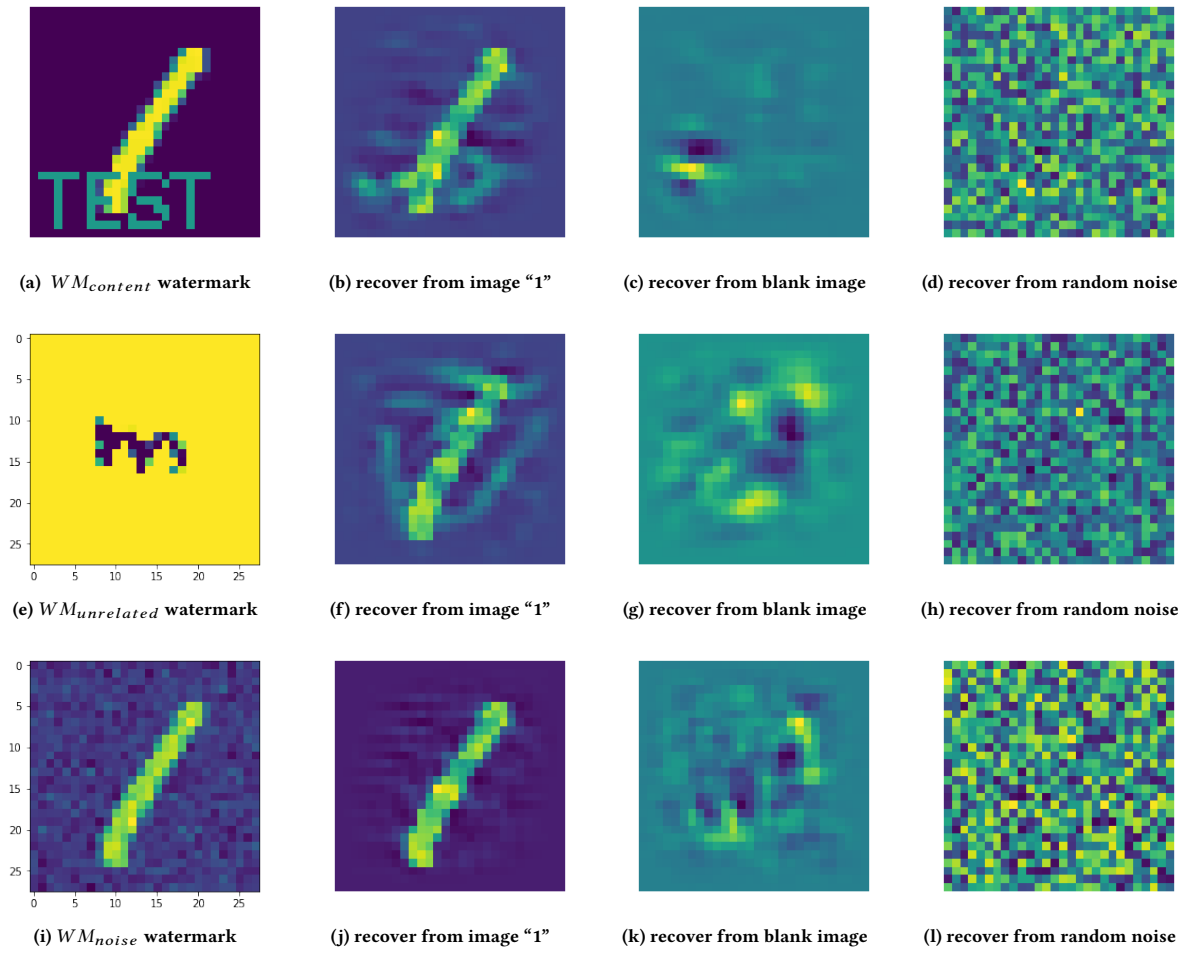
Such result is expected since the recovered images from model inversion attacks are usually the prototypical image in that class. Consistent with the results as shown in [27], our experiments also show that model inversion attacks cannot recover clear training data for convolutional neural networks. Hitaj et al. [27] propose a new attack using generative adversarial networks (GANs) to recover training data for collaborative training. However, such attack require to train a generative model together with a discriminative model during the training process, which is not applicable for our setting. Adversaries in our threat model can only get a pre-trained model with watermarks, but are not able to intervene the training process.

5.6 Comparison of different watermarks

In this section, we compare the trade-off among different watermarks and summarize the insights we learned for DNN watermarks.

Functionality. All of our proposed watermarks can support both white-box and black-box based ownership verification, since they only require to access normal APIs for the verification.

Usability. $WM_{content}$ is the best choice in terms of usability. The original image can always get correct predictions and only images with watermarks embedded get pre-defined predictions. $WM_{unrelated}$ may cause false positives if the unrelated images happen to be used as inputs, similar to WM_{noise}.



(a) $WM_{content}$ watermark (b) recover from image "1" (c) recover from blank image (d) recover from random noise

(e) $WM_{unrelated}$ watermark (f) recover from image "1" (g) recover from blank image (h) recover from random noise

(i) WM_{noise} watermark (j) recover from image "1" (k) recover from blank image (l) recover from random noise

Figure 8: Model inversion attacks on MNIST

Security. WM_{noise} is the most safe watermark, even it was recovered, it is still difficult to distinguish it from normal noise.

Robustness. $WM_{content}$ is robust to all the evaluated modifications for both datasets.

In summary, to make a good watermark for DNNs, one important thing needs to be considered is the generality (" generalization" vs "overfitting") of the watermark. " Generalization" means that any input follows the watermark patterns can trigger the model with watermark embedded. For example, in our $WM_{unrelated}$, any forms of "1" can trigger the models to pre-defined prediction for CIFAR10 data. "Overfitting" means that only specified image in the training data can trigger watermark. For example, only one specified "1" can trigger the model while other "1" cannot. " Generalization" makes watermarks robust to different modifications while it may cause usability issues, since any input follows the same pattern can trigger the model. "Overfitting" can reduce the usability issues, but are more vulnerable to modification attacks. Therefore, for each method, if we want to use an overfitted watermark, we need to train the model with exactly the same watermark. However, if we want to adopt a generalized watermark, we can train the model with more diverse watermarks, e.g., training with data augmentation on watermarks.

6 DISCUSSION

In this section, we discuss possible limitations and evasion of our watermarking framework.

Limitation. Our evaluation has shown great performance of the watermarking framework to protect the intellectual property of deep neural networks once those models are leaked and deployed as online services. However, if the leaked model is not deployed as an on-line service but used as an internal service, then we cannot detect that. In this way, the plagiarizer cannot directly monetize the stolen models. In addition, our current watermarking framework cannot protect the DNN models from being stolen through prediction APIs [53]. In this attack, attackers can exploit the tension between query access and confidentiality in the results to learn the parameters of machine learning models. However, such attacks work well for conversion machine learning algorithms such as decision trees and logistic regressions. It needs more queries $100k$,

where k is the number of model parameters for a two-layered neural network, which makes it less effective for more complicated DNN models (VGG-16 has 138M parameters). In addition, as discussed in [53], such attacks can be prevented by changing APIs by not returning confidences and not responding to incomplete queries. It is also possible to learn the query patterns of such attacks to detect them before stealing the models.

Evasion. Our watermarking framework consists of three components: watermark generation, watermark embedding, and ownership verification. Only ownership verification component needs to be done remotely, therefore, one way to evade our watermarking framework is to prevent our queries to ownership verification. Recently Meng et al. [39] proposed a framework named MagNet to defend against adversarial queries. Specifically, MagNet trains multiple AutoEncoders with normal data to learn the representation of normal data and then use those AutoEncoders as abnormal detectors. The insights behind the MagNet is that adversary samples usually have different distribution with normal samples. Therefore, such defense techniques could also be used here to defend against our ownership verification queries since our embedded watermarks also show the difference with normal samples. However, the effectiveness of MagNet depends on the normal examples for the training of their detector networks. Insufficient normal examples will lead to high false positives. In our case, we assume that plagiarizers do not have a sufficient normal dataset to train such detectors, otherwise, they can directly train the model by themselves without needs for stealing the model.

7 RELATED WORK

Watermarking. Digital watermarking is the method to hide the secret information into the digital media in order to protect the ownership of those media data. Many approaches have been proposed to make the watermark to be efficient as well as robust to removal attacks. Spatial domain digital watermarking algorithms have been investigated in [7, 30, 36, 52]. They embed secrets by directly manipulating pixels in an image. For example, the LSB (least significant bit) [30, 36] of pixels is commonly used to embed secret. However, such techniques are vulnerable to attacks and are sensitive to noise and common signal processing. Compared to spatial-domain methods, frequency domain methods are more widely applied, which embed the watermarks in the spectral coefficients of the image. The most commonly used transforms are the Discrete Cosine Transform (DCT) [25, 44], Discrete Fourier Transform (DFT) [42, 55], Discrete Wavelet Transform (DWT) [9, 11, 31, 58] and the combination of them [6, 38, 46]. In order to verify the ownership of protected media data, all existing watermarking algorithms require to directly access those media data to extract the watermarks and verify the ownership. However, in deep neural networks, we need to protect DNN models rather than input media data, after training, usually only the DNN model API is available for ownership verification. Therefore, the existing digital watermarking algorithms cannot be directly applied to protect DNN models.

Recently, Uchida et al. [54] proposed the first method for embedding watermarks into deep neural networks. It embedded information into the weights of the deep neuron network. Therefore, it assumes that the stolen models can be locally accessible to extract all the parameters, which is not practical, since most deep learning models are deployed as on-line service and it will be hard to directly get access to model parameters especially for the stolen models. Merrer et al. [40] proposed a zero-bit watermarking algorithm that makes use of adversarial samples as watermarks to verify the ownership of neural networks. Specifically they fine-tune the DNN models to include certain true/false adversaries and use the combination of such adversaries as keys K to verify the DNN models. If the DNN models can return the pre-defined results to these keys K, they can confirm the ownership of the DNN models. However, such an algorithm has a vulnerability that each model essentially has infinite such keys, therefore everyone can claim the ownership of the DNN modesl with any K. For example, we can generate a set of adversarial samples with any DNN model, and then claim that those models belong to us since we can extract these adversarial samples from those models. Different from these two existing works, our framework can remotely verify the ownership of DNN models and our embedded watermarks are unique to each model. For example, for our $WM_{content}$ watermark generation algorithm, only the image with embedded content "Test" can trigger the pre-defined output.

Deep neural network attack and defense. As deep neural networks are being widely used, a variety of attacks have been investigated on it. Fredrikson et al. [16] introduced the model inversion attack that can recover images in the training dataset from deep neural networks. As we have shown in our evaluation, our watermarking framework is robust to such attacks. Tramer et al. [53] introduced an attack to steal the general machine learning models. Such attacks can be prevented by updating DNN APIs by not returning confidence score and not responding to incomplete queries. Shokri et al. [47] introduced membership inference attacks, which can determine whether the given record was used as part of the model's training dataset or not. Such attack is not applicable for inferring our watermarks since attackers need to know watermarks first. [37] and [20] recently introduced deep neural network Trojaning attacks, which embed hidden malicious functionality into neural networks. Similar to Trojans in software, such attacks could be prevented by checking the model integrity. We have a different threat model here and we focus on how to use watermarking to protect the intellectual property of DNN models.

8 CONCLUSION

In this paper, we generalized the "digital watermarking" concept for deep neural networks and proposed a general watermarking framework to produce different watermarks, embed them into deep neural networks, and remotely verify the ownership of DNN models based on the embedded watermarks. We formally define the threat model of watermarking in deep neural networks to support both white-box and black-box access. The key innovation of our watermarking framework is that it can remotely verify the ownership of deep neural network services with few API queries. We also perform a comprehensive evaluation with our watermarking framework on two benchmark datasets. We demonstrate that our framework can satisfy the general watermarking standard and is robust to different counter-watermark attacks.

REFERENCES

[1] 2016. ImageNet. http://www.image-net.org/. (2016).

[2] 2017. How Amazon, Google, Microsoft, And IBM Sell AI As A Service. https://www.fastcompany.com/40474593/how-amazon-google-microsoft-and-ibm-sell-ai-as-a-service. (2017).

[3] 2017. Model Gallery. https://www.microsoft.com/en-us/cognitive-toolkit/features/model-gallery/. (2017).

[4] 2017. The Value of Stolen Data on the Dark Web. https://darkwebnews.com/dark-web/value-of-stolen-data-dark-web/. (2017).

[5] Martín Abadi et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. In *arXiv:1603.04467*.

[6] Ali Al-Haj. 2007. Combined DWT-DCT Digital Image Watermarking. In *Journal of Computer Science*.

[7] Mustafa Osman Ali, Elamir Abu Abaida Ali Osman, and Rameshwar Row. 2012. Invisible Digital Image Watermarking in Spatial Domain with Random Localization. In *International Journal of Engineering and Innovative Technology*.

[8] Sajid Anwar and Wonyong Sung. 2016. Compact Deep Convolutional Neural Networks With Coarse Pruning. In *arXiv:1610.09639*.

[9] Mauro Barni, Franco Bartolini, and Alessandro Piva. 2001. Improved wavelet-based watermarking through pixel-wise masking. In *IEEE Transactions on Image Processing*.

[10] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy (S&P '17)*.

[11] Munesh Chandra and Shikha Pandey. 2010. A DWT domain visible watermarking techniques for digital images. In *International Conference On Electronics and Information Engineering (ICEIE '10)*.

[12] François Chollet. 2015. Keras. In *https://github.com/fchollet/keras*.

[13] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. 2017. EMNIST: an extension of MNIST to handwritten letters.. In *arXiv:1702.05373*.

[14] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. 2007. Digital Watermarking and Steganography. In *Morgan Kaufmann Publishers Inc.*

[15] Y. Le Cun, I. Guyon, L. D. Jackel, D. Henderson, B. Boser, R. E. Howard, J. S. Denker, W. Hubbard, and H. P. Graf. 1989. Handwritten digit recognition: applications of neural network chips and automatic learning. In *IEEE Communications Magazine*.

[16] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *ACM SIGSAC Conference on Computer and Communications Security*.

[17] Yoav Goldberg. 2015. A primer on neural network models for natural language processing. In *Journal of Artificial Intelligence Research*.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[19] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. 2013. Speech Recognition with Deep Recurrent Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '13)*.

[20] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. In *arXiv:1708.06733*.

[21] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. In *Proceedings of Neural Information Processing Systems(NIPS'15)*.

[22] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2012. Deep Speech: Scaling up end-to-end speech recognition. In *arXiv:1412.5567*.

[23] Frank Hartung and Martin Kutter. 1999. Multimedia watermarking techniques. In *Proceedings of the IEEE*.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*.

[25] J.R. Hernandez, M. Amado, and F. Perez-Gonzalez. 2000. DCT-domain watermarking techniques for still images: detector performance analysis and a new structure. In *IEEE Transactions on Image Processing*.

[26] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. In *IEEE Signal Processing Magazine*.

[27] Briland Hitaj, Giuseppe Ateniese, and Ravi K ShethFernando Perez-Cruz. 2017. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning.. In *ACM Conference on Computer and Communications Security*.

[28] Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. In *Neural computation*.

[29] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).

[30] Neil F. Johnson and Sushil Jajodia. 1998. Exploring Steganography: Seeing the Unseen. In *IEEE Computer*.

[31] Nikita Kashyap and G. R. SINHA. 2012. Image Watermarking Using 3-Level Discrete Wavelet Transform (DWT). In *International Journal of Modern Education and Computer Science (IJMECS '12)*.

[32] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. In *Master's thesis, Department of Computer Science, University of Toronto*.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS '12)*.

[34] Gerhard C. Langelaar, Iwan Setyawan, and Reginald L. Lagendijk. 2000. Watermarking digital image and video data. A state-of-the-art overview. In *IEEE Signal Processing Magazine*.

[35] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*.

[36] Yeuan-Kuen Lee, Graeme Bell, Shih-Yu Huang, Ran-Zan Wang, and Shyong-Jian Shyu. 2009. An Advanced Least-Significant-Bit Embedding Scheme for Steganographic Encoding. In *Proceedings of the 3rd Pacific-Rim Symposium on Image and Video Technology(PSIVT '09)*.

[37] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS '18)*.

[38] Jiansheng Mei, Sukang Li, and Xiaomei Tan. 2009. A Digital Watermarking Algorithm Based On DCT and DWT. In *Proceedings of the 2009 International Symposium on Web Information Systems and Applications (WISA '09)*.

[39] Dongyu Meng and Hao Chen. 2017. MagNet: a Two-Pronged Defense against Adversarial Examples.. In *ACM Conference on Computer and Communications Security*.

[40] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. 2017. Adversarial Frontier Stitching for Remote Neural Network Watermarking. In *arXiv:1711.01894*.

[41] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning Convolutional Neural Networks for Resource Efficient Inference. In *International Conference on Learning Representations (ICLR '17)*.

[42] Shelby Pereira and Thierry Pun. 2000. Robust template matching for affine resistant image watermarks. In *IEEE Transactions on Image Processing*.

[43] Nikiforos Pittaras, Foteini Markatopoulou, Vasileios Mezaris, and Ioannis Patras. 2017. Comparison of Fine-Tuning and Extension Strategies for Deep Convolutional Neural Networks. In *International Conference on Multimedia Modeling*.

[44] A. Piva, M. Barni, E Bartolini, and V. Cappellini. 1997. DCT-based watermark recovering without resorting to the uncorrupted original image. In *International Conference on Image Processing*.

[45] Lalit Kumar Saini and Vishal Shrivastava. 2014. A Survey of Digital Watermarking Techniques and its Applications. In *International Journal of Computer Science Trends and Technology (IJCST '14)*.

[46] Ravi K Sheth and V. V. Nath. 2016. Secured digital image watermarking with discrete cosine transform and discrete wavelet transform method. In *International Conference on Advances in Computing, Communication, and Automation*.

[47] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *IEEE Symposium on Security and Privacy (S&P '17)*.

[48] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR '15)*.

[49] Suraj Srinivas and R. Venkatesh Babu. 2015. Data-free Parameter Pruning for Deep Neural Networks. In *BMVA Press*.

[50] Mitchell D. Swanson, Mei Kobayashi, and Ahmed H. Tewfik. 1988. Multimedia data-embedding and watermarking technologies. In *Proceedings of the IEEE*.

[51] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. In *arXiv:1703.09039*.

[52] Jun Tian. 2003. Reversible Data Embedding Using a Difference Expansion. In *IEEE Transactions on Circuits and Systems for Video Technology*.

[53] Florian Tramer, Fan Zhang, Ari Juels, Michael Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *Proceedings of the 25th USENIX Security Symposium (Security '16)*.

[54] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding Watermarks into Deep Neural Networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval (ICMR '17)*.

[55] Matthieu Urvoy, Dalila Goudia, and Florent Autrusseau. 2014. Perceptual DFT Watermarking With Improved Detection and Robustness to Geometrical Distortions. In *IEEE Transactions on Information Forensics and Security (TIFS '14)*.

[56] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Neural Information Processing Systems*.

[57] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.

[58] Lijing Zhang and Aihua Li. 2009. Robust Watermarking Scheme Based on Singular Value of Decomposition in DWT Domain. In *Asia-Pacific Conference on Information Processing*.

A APPENDIX

A.1 DNN model architecture

Table 6 and Table 7 show the architecture and training parameters of DNN models for different datasets.

Table 6: Architecture of DNN models

Layer Type	MNIST	CIFAR10
Conv.ReLU	32 filters (3×3)	64 filters (3×3)
Conv.ReLU	32 filters (3×3)	64 filters (3×3)
Max Pooling	2×2	2×2
Conv.ReLU	64 filters (3×3)	128 filters (3×3)
Conv.ReLU	64 filters (3×3)	128 filters (3×3)
Max Pooling	2×2	2×2
Dense.ReLU	200	256
Dense.ReLU	200	256
Softmax	10	10

Table 7: Training parameters for different models

Parameter	MNIST	CIFAR10
Optimization method	SGD	SGD
Loss function	Crossentropy	Crossentropy
Learning Rate	0.01	0.01
Batch Size	128	128
Epoch	50	50
Dropout Rate	0.5	0.5

(a) $WM_{content}$ watermark (b) recover from image "automobile" (c) recover from blank image (d) recover from random noise

(e) $WM_{unrelated}$ watermark (f) recover from image "automobile" (g) recover from blank image (h) recover from random noise

(i) WM_{noise} watermark (j) recover from image "automobile" (k) recover from blank image (l) recover from random noise

Figure 9: Model inversion attacks on CIFAR10