Robust and Undetectable White-Box Watermarks for Deep Neural Networks

Tianhao Wang
Harvard University

Florian Kerschbaum
University of Waterloo

ABSTRACT
Training deep neural networks (DNN) is expensive in terms of computational power and the amount of necessary labeled training data. Thus, deep learning models constitute business value for data owners. Watermarking of deep neural networks can enable their tracing once released by a data owner. In this paper we define and formalize white-box watermarking algorithms for DNNs, where the data owner needs white-box access to the model to extract the watermark. White-box watermarking algorithms have the advantage that they do not impact the accuracy of the watermarked model. We demonstrate a new property inference attack using a DNN that can detect watermarking by any existing, manually designed algorithms regardless of training data set and model architecture. We then propose the first white-box DNN watermarking algorithm that is undetectable by the property inference attack. We further extend the capacity and robustness of the watermark. Unlike prior watermarking schemes which restrict the content of watermark message to short binary strings, our new scheme largely increase the capacity and flexibility of the embedded watermark message. Experiments show that our new white-box watermarking algorithm does not impact accuracy, is undetectable and robust against moderate model transformation attacks.

1 INTRODUCTION
With data becoming an asset, owners try to protect its use and dissemination. This includes protecting publicly accessible machine learning models derived from this data. However, a machine learning model is itself only data and can be easily copied and reused. Watermarking [19] may enable to trace copied models, where the data owner needs white-box access to the model to extract the watermark. White-box watermarking algorithms have the advantage that they do not impact the accuracy of the watermarked model. We demonstrate a new property inference attack using a DNN that can detect watermarking by any existing, manually designed algorithms regardless of training data set and model architecture. We then propose the first white-box DNN watermarking algorithm that is undetectable by the property inference attack. We further extend the capacity and robustness of the watermark. Unlike prior watermarking schemes which restrict the content of watermark message to short binary strings, our new scheme largely increase the capacity and flexibility of the embedded watermark message. Experiments show that our new white-box watermarking algorithm does not impact accuracy, is undetectable and robust against moderate model transformation attacks.

Contributions. In summary, in this paper we contribute three machine learning algorithms using deep neural networks for attacking and defending watermarks in deep neural networks and their experimental evaluation:

- a property inference attack as a generic detector of white-box watermarks in neural networks (Section 4). This algorithm works across data sets and architectures.
- an adversarial learning network that can thwart the generic detector of white-box watermarks (Section 5.1). The resulting distribution of weights in our new watermarking algorithm is indistinguishable from non-watermarked weights.

Thus, deep learning models constitute business value for data owners. Watermarking of deep neural networks can enable their tracing once released by a data owner. In this paper we define and formalize white-box watermarking algorithms for DNNs, where the data owner needs white-box access to the model to extract the watermark. White-box watermarking algorithms have the advantage that they do not impact the accuracy of the watermarked model. We demonstrate a new property inference attack using a DNN that can detect watermarking by any existing, manually designed algorithms regardless of training data set and model architecture. We then propose the first white-box DNN watermarking algorithm that is undetectable by the property inference attack. We further extend the capacity and robustness of the watermark. Unlike prior watermarking schemes which restrict the content of watermark message to short binary strings, our new scheme largely increase the capacity and flexibility of the embedded watermark message. Experiments show that our new white-box watermarking algorithm does not impact accuracy, is undetectable and robust against moderate model transformation attacks.
Watermarking techniques for neural networks can be classified into black-box and white-box algorithms. A black-box watermark can be extracted by only querying the model (black-box access). A white-box watermark needs access to the model and its parameters in order to extract the watermark. In this paper we present a white-box watermarking algorithm. The first white-box algorithm was developed by Uchida et al. [36]. Subsequently Rouhani et al. [29] presented an improved version. We generalize both algorithms into a scheme for white-box watermarking algorithms and present their details in Section 3.

A first attack on Uchida et al.'s algorithm was presented by Wang and Kerschbaum [37]. They show that the presence of a watermark is easily detectable and that it can be easily removed by an overwriting attack. We emphasize that a white-box watermark that does not impact accuracy cannot protect against model stealing attacks [18, 28], since model stealing is a black-box attack and the black-box interface is unmodified by the watermark. However, white-box attacks still have important applications when the model needs to be highly accurate or model stealing attacks are not feasible due to a restricted number of queries or available computational resources.

Black-box watermarking algorithms have the advantage that they only need black-box access to the watermarked neural network to extract the watermark, but they will necessarily impact the accuracy of the watermarked model. The first black-box watermarking algorithm using backdoors [7, 12, 25] where concurrently developed by Zhang et al. [41] and Adi et al. [2]. A backdoor is additional, out-of-distribution training data inserted to trigger abnormal behavior on the inserted data [7, 12, 25], also referred to as an integrity poisoning attack. This attack can be prevented by an attacker with a pre-filter that detects “abnormal” images and then answers randomly [16]. However, this counter-measure can be circumvented by using more clever backdoor images [24]. Shafieinejad et al. show that these backdoor-based watermarks can be easily removed using efficient model stealing attacks [30].

There are other types of black-box algorithms. Chen et al. [6] and Le Merrer et al. [27] use adversarial examples to generate watermarks. Szyller et al. [33] modify the classification output of the neural network in order to embed a black-box watermark. All black-box algorithms are obviously susceptible to sybil attacks [16], unless access to multiple, differently watermarked models is prevented.

3 BACKGROUND
This section provides a formal definition of white-box watermarking of deep neural networks. We provide a general scheme that encompasses at least the white-box neural network watermarking algorithms in [29, 36].

3.1 Deep Neural Networks
In this paper, we focus on deep neural networks (DNNs). A DNN is a function \( f : X \rightarrow Y \), where \( X \) is the input space, usually \( \mathbb{R}^n \), and \( Y \) is the collection of classes where each valid input \( x \in X \) belongs to. We assume that for every \( x \in X \), it belongs to a unique class \( y \in Y \), i.e. there exists a perfect oracle function \( f^* \) for ground-truth classification function \( f : X \rightarrow Y \) which can always correctly predict the class \( y \) of any instance \( x \in X \). In this paper, we do not consider the case when some \( x \in X \) are undetermined or belong to multiple classes. A DNN \( f \) has function parameters \( w \), which is a sequence of adjustable values to enable \( f \) fitting a wide range of mappings. The values \( w \) are also commonly referred as model parameters or model weights. For an instance \( x \in X \), we represent the output of neural network \( f \) as \( f(x; w) \). Let \( \mathbb{W} \) be the parameter space of \( w \), i.e. \( w \in \mathbb{W} \). \( \mathbb{W} \) is usually high-dimensional real space \( \mathbb{R}^n \) for DNNs, where \( n \) is the total number of model parameters. The goal of training a DNN \( f \) is to let \( f \) approximate the function \( f^* \) by updating \( w \). The training of DNNs is the process of searching for the optimal \( w \) parameter space to minimize a function \( E_o : \mathbb{W} \rightarrow \mathbb{R} \), which is typically a categorical cross-entropy derived from training data \( X_{train} \subset X \) and its classes \( Y_{train} \) assigned by \( f^* \). \( E_o \) is commonly referred to as loss function. The accuracy of \( f \) after training depends on the quality of the loss function \( E_o \), while the quality of \( E_o \) in turn depends on the quality of the training data \( (X_{train}, Y_{train}) \). The search for a global minimum is typically performed using a stochastic gradient descent algorithm.

To formally define training, we assume there exist three algorithms:

- \( E_o \leftarrow \text{DesignLoss}(X_{train}, Y_{train}) \) is a deterministic polynomial-time algorithm that outputs loss function \( E_o \) according to the available training set \( (X_{train}, Y_{train}) \).
- \( w_{i+1} \leftarrow \text{TrainBatch}(E_o, w_i) \) is a probabilistic polynomial-time algorithm that applies one iteration of a gradient descent algorithm to minimize \( E_o \) with the starting weights \( w_i \), and outputs the resulting weights \( w_{i+1} \).
- \( w \leftarrow \text{Train}(E_o, w_0) \) is a probabilistic polynomial-time algorithm that applies \( \text{TrainBatch}(E_o, w_i) \) iteratively for \( p(n) \) steps where in the \( i \)-th iteration the input \( w_i \) is the \( w_0 \) returned from the previous iteration step. The algorithm outputs the final model parameters after \( p(n) \) steps, where \( p(n) \) is a polynomial in \( n \). For simplicity in the following text, when the initial weights \( w_0 \) are randomly initialized, we omit argument \( w_0 \) and simply write \( \text{Train}(E_o) \).

A well-trained DNN model \( f \) is expected to approximate the ground-truth function \( f^* \) well. Given DNN \( f \) and loss function \( E_o \), we say \( f \) is \( \epsilon \)-accurate if \( \Pr_{x \in X}[f(x; w) \neq f^*(x)] < \epsilon \)

where \( w \) is the trained parameter returned by \( \text{Train}(E_o) \).

A regularization term \([5]\), or regularizer, is commonly added to the loss function to prevent models from overfitting. A regularizer is applied by training the parameters using \( \text{Train}(E_o + \lambda E_R) \) where \( E_R \) is the regularization term and \( \lambda \) is an empirical coefficient to adjust its importance.

3.2 White-box Watermarking for DNN models
Digital watermarking is a technique used to embed a secret message, the watermark, into cover data (e.g. an image or video). It can be used
to provide proof of ownership of cover data which is legally protected as intellectual property. In white-box watermarking of DNN models the
cover data are the model parameters \( w \). DNNs have a high dimension
of parameters, where many parameters have little significance in their primary classification task. These parameters can be used to encode additional information beyond what is required for the primary task.

A white-box neural network watermarking scheme consists of a
message space \( \mathcal{M} \) and a key space \( \mathcal{K} \). It also consists of two algorithms:

- \( m \leftarrow \text{Extract}(w,k) \) is a deterministic polynomial-time algorithm that
given model parameters \( w \) and (secret) extraction key \( k \) outputs extracted watermark message \( m \).
- \( w_{wm}, k \leftarrow \text{Embed}(E_o, w_0, m) \) is a probabilistic polynomial-time algorithm that given original loss function \( E_o \), a watermark message \( m \) and initial model weights parameters \( w_0 \) outputs model parameters \( w_{wm} \) and (secret) extraction key \( k \). In some watermarking algorithms [29, 36] \( k \) can be chosen independently of \( E_o \), \( w_0 \) and \( m \) using a key generation function \text{KeyGen}. For generality including our watermarking scheme, however, combine both algorithms into one.

The extraction of the watermarks, i.e. algorithm \text{Extract}(w,k) \)
usually proceeds in two steps: (a) feature extraction \( g_{wm} \) and (b) message extraction \( e \). The extraction key is also separated into two parts \( k = (k_{FE}, k_E) \) for each of the steps in \text{Extract}. First, given feature extraction key \( k_{FE} \), features \( q \) are extracted from \( w \):

\[
q \leftarrow g_{wm}(w, k_{FE})
\]

For example, in the simplest case, the feature \( q \) can be a subset of \( w \), e.g.,
the weights of one layer of the model, and \( k_{FE} \) is the index of the layer.
This step is necessary to reduce the complexity of a DNN’s structure.

Second, given message extraction key \( k_E \) the message \( m \) is extracted from the features \( q \):

\[
m \leftarrow e(q, k_E)
\]

The function \( e \) will be referred to as extraction function or extractor
interchangeably in the main text. To avoid generating a trivial extractor which will extract the same watermark message regardless of the input, we must force the extractor to be valid. We say an extractor is valid, if it has the non-trivial ownership property defined in Section 3.3. Note that in order to enable watermark embedding, \( g_{wm} \) and \( e \) must be differentiable.

Embedding of the watermark, i.e. algorithm \text{Embed}(E_o, m) \) is performed alongside the primary task of training a DNN to approximate a function \( f \). First, a random key \( k = (k_{FE}, k_E) \) is randomly generated. Embedding a watermark message \( m \in \mathcal{M} \) into target DNN \( \Phi_{gf} \) consists of regularizing \( \Phi_{gf} \) with a special regularization term \( E_{wm} \). Let \( d : \mathcal{M} \times \mathcal{K} \rightarrow \mathbb{R} \) be a differentiable distance function measures the discrepancy between two messages. For example, when \( m \) is a binary string of length \( n \), i.e. \( \mathcal{M} \subseteq \{0,1\}^n \), \( d \) can simply be the binary cross-entropy. Given a watermark message \( m \) to embed, the regularization term is then defined as:

\[
E_{wm} = d(e(g_{wm}(w, k_{FE}), k_E), m)
\]

The watermarked model \( \Phi_{gf} \) with model parameters \( w_{wm} \) is obtained by the training algorithm \text{Train}(E_o + \lambda E_{wm}) \).

### 3.3 Requirements

There are a set of minimal requirements that a DNN watermarking
algorithm should fulfill.

**Functionality-Preserving**: The embedding of the watermark
should not impact the accuracy of the target model:

\[
Pr_{x \in \mathcal{X}}[f(x; w_{wm}) = f(x)] \approx Pr_{x \in \mathcal{X}}[f(x; w) = f(x)]
\]

where \( w_{wm} \) is returned by \text{Train}(E_o + \lambda E_{wm}) \) and \( w \) is returned by \text{Train}(E_o) \).

**Robust**: For any moderate model transformation (independent of the key \( k \), e.g. fine-tuning) mapping \( w_{wm} \) to \( w' \), such that model accuracy does not degrade, the extraction algorithm should still be able to extract watermark message \( m' \) from \( w' \) that is convincingly similar to the original watermark message \( m \), i.e. if

\[
Pr_{x \in \mathcal{X}}[f(x; w') = f(x)] \approx Pr_{x \in \mathcal{X}}[f(x; w_{wm}) = f(x)]
\]

where \( w' \) is obtained from a moderate model transformation mapping such as fine-tuning, then

\[
\text{Extract}(w', k) \approx \text{Extract}(w_{wm}, k)
\]

We do not consider robustness against adversarial transformations that only use the black-box interface \( f(x; w') \), since this is unmodified by white-box watermarking algorithms.

A further requirement we pose to a watermarking algorithm is
that the watermark in the cover data is undetectable. This is a useful
property, because it may deter an adversary from the attempt to
remove the watermark, but it is not strictly necessary.

**Undetectable**: We say a watermark is undetectable, if no polynomial-time adversary algorithm \( \mathcal{A} \) wins the following game:

\[
\begin{align*}
\mathcal{F}_0 &\leftarrow \text{Train}(E_o) \\
\mathcal{F}_1, k &\leftarrow \text{Embed}(E_o, m) \\
b &\leftarrow \{0,1\} \\
Pr[\mathcal{A}(\mathcal{F}_b) = b] &\approx \frac{1}{2}
\end{align*}
\]

Loosely speaking, the adversary should not be able to distinguish a
watermarked model from a non-watermarked one. In the literature [2, 24] further properties of watermarking algorithms have been defined. We review them here and show that they are met by our new watermarking scheme in the remainder of the paper.

**Non-trivial ownership**: This property requires that an adversary
is not capable of producing a key that will result on a predictable
message for any DNN. Formally, \( \forall k \in \mathcal{K} \), we have

\[
Pr_{w \in \mathcal{W}, m \in \mathcal{M}}[\text{Extract}(w, k) = m] \approx \frac{1}{|\mathcal{M}|}
\]

If this requirement is not enforced, an attacker can find a \( k \) that can
extract watermark message \( m \) from any \( w \in \mathcal{W} \), and then he/she can
claim ownership of any DNN model. We require any valid extraction function to prevent this attack.

**Unforgeability**: This property requires that an adversary is not
capable of reproducing the key for a given watermarked model.
Formally, \( \forall w, w' \in \mathcal{W} \) and \( \forall m \in \mathcal{M} \), we have

\[
Pr_{k \in \mathcal{K}}[\text{Extract}(w, k) = m] = Pr_{k \in \mathcal{K}}[\text{Extract}(w', k) = m]
\]
Intuitively, this requirement implies that an adversarial cannot learn anything about \( k \) from model weights \( w \). This property can be easily achieved by the owner cryptographically committing to and timestamping the key [2] and is orthogonal to the watermarking algorithms described in this paper.

Ownership Piracy: This property requires that an adversary that embeds a new watermark into a DNN does not remove any existing ones. We show that this property holds in Section 6.5 where we evaluate the overwriting attack.

3.4 Uchida et al.’s Watermarking Scheme

In Uchida et al.’s scheme, the message space is a sequence of \( t \) values between 0 and 1, i.e. \( M = \mathbb{R}^t_{[0,1]} \). A typical watermark message \( m \in M \) is a \( t \)-bit binary string. Both feature extraction key space \( \mathbb{K}_{FE} \) and message extraction key space \( \mathbb{K}_e \) are matrix spaces. The features \( q \) to embed the watermark into are simply the weights of a layer of the DNN, i.e. \( g_{wm} \) is the multiplication of a selection matrix \( Z_{FE} \) with the vector \( w \) of weights. Hence the feature extraction key \( k_{FE} = Z_{FE} \). The message extraction function \( e \) does a linear transformation over the weights \( w_l \) of one layer using message extraction key matrix \( k_e = Z_e \) and then an evaluation of the resulting vector using a sigmoid function to restrict the range of values in the vector. The distance function \( d \) is the binary cross-entropy between watermark message \( m \) and extracted message \( e(g_{wm}(w_l Z_{FE}), Z_E) \).

Formally, Uchida et al.’s watermarking scheme is defined as follows:

- \( g_{wm} : \mathbb{W} \times \mathbb{K}_{FE} \rightarrow \mathbb{W}_l \) where \( g_{wm}(w_l Z_{FE}) = Z_{FE} w = w_l \).
- \( Z_{FE} \) is a \( |w_l| \times |w| \) matrix with a 1 at position \((i,1), (i+1,2)\) and so forth and 0 otherwise where \( i \) is the start index of a layer. \( \mathbb{W}_l \) is the parameter space of the weights of the selected layer, which is a subspace of \( \mathbb{W} \).
- \( e : \mathbb{W}_l \times \mathbb{K}_e \rightarrow \mathbb{W} \) where \( e(w_l, Z_e) = \sigma(Z_e w_l) \).
- \( Z_e \) is a \( |t| \times |w| \) matrix whose values are randomly initialized according to standard Gaussian distribution. \( \sigma \) denotes sigmoid function.
- \( d : \mathbb{W} \times \mathbb{K}_{FE} \rightarrow \mathbb{R}_+ \) where \( d(m,y) = m \log(y) + (1-m) \log(1-y) \) and \( y = e(g_{wm}(w_l Z_{FE}), Z_E) \).

3.5 DeepSigns Watermarking Scheme

In the DeepSigns scheme [29], Rouhani et al. replace the feature selection part in their watermarking algorithm compared to Uchida et al.’s scheme. The features of \( w \) they choose to embed the watermark into are the activations of a chosen layer of the DNN given a trigger set input. Hence the feature extraction key space \( \mathbb{K}_{FE} \) is a product space of a matrix space and input space \( X \). The feature extraction key is \( k_{FE} = (Z_{FE},x) \).

- \( g_{wm} : \mathbb{W} \times \mathbb{K}_{FE} \rightarrow \mathbb{W}_l \) where \( g_{wm}(w,(Z_{FE},x)) \) outputs the activations of the selected layer of the DNN given trigger set \( x \subseteq X \).
- \( e, d \) are the same as in Uchida et al.’s scheme.

4 GENERIC DETECTION VIA PROPERTY INFERENCE

We present our property inference attack that fully distinguishes watermarked deep learning models from non-watermarked ones, regardless of their architectures and training data.

Property inference attacks on DNNs [10, 28] have been originally proposed to extract knowledge about the training data given white-box access to the model. We propose to use a property inference attack to detect whether a watermark has been embedded into the target DNN \( F_{tgt} \) by a white-box watermarking algorithm \( \text{Embed} \).

The attack could be used to check whether a watermark removal attack is necessary and could, for example, he offered as a service in underground markets.

In our proposed attack, the attacker wants to determine whether the target model \( F_{tgt} \) has been watermarked by \( \text{Embed} \) or not. The attacker only needs to have knowledge of the watermarking algorithm, i.e. the algorithms \( \text{Extract} \) and \( \text{Embed} \) defined by functions \( (g_{wm}, e, d) \). The attacker needs no knowledge about \( F_{tgt} \)’s architecture or training data and has only to be able to generate \( f \) sufficiently different high-accuracy models for a natural, non-trivial classification function, less complex than and unrelated to \( F_{tgt} \)’s function.

In the following parts, we first present the watermark detection algorithm and subsequently describe and perform an exemplary property inference attacks for the above two described watermarking schemes [29, 36].

4.1 Attack Design and Feature Selection

Given a white-box watermarking algorithm \( \text{Embed} \) and an oracle \( O^D \) implementing the detection function \( D : \{F\} \rightarrow \{0,1\} \) that can always correctly detect watermarks embedded by \( \text{Embed} \) in any model, the adversary’s goal is to train a meta DNN model \( F_{det} \) with parameters \( w_D \) that serves as a generic watermark detector to approximate the detection function \( D \).

Formally, the attack algorithm shall output an \( \epsilon \)-accurate model \( F_{det} \) where

\[ Pr[F_{det}(F;w_D) \neq D(F)] < \epsilon \]

for arbitrary DNN models \( F \) of any architecture and trained on any data set.

Algorithm 1 shows the process of training such a meta model. The intuition behind the generic detection attack is to find unusual patterns among the watermarked models’ weights distributions, given sufficiently many examples. DNN models trained by similar learning approach, e.g. using a common regularizer, will represent functions similarly to some extent, and we conjecture that the similarity of these function representations is reflected as common patterns in their model parameters. Hence, a meta model in the form of DNN \( F_{det} \) can detect these patterns.

Because of the complexity of DNNs and their architectures, we need a fine-grained feature representation to assist the watermark detection model. The feature extraction algorithm is referred to as \( g_D \) in Algorithm 1. To improve the performance of the resulting generic detector \( F_{det} \), the feature representation needs to be permutation invariant. Ganju et al. [10] show the importance of permutation invariance for successful property inference attacks. In our experiments for Uchida’s scheme, we extract the histogram of the weights as our feature for detecting watermarks, in the form of normalized percentiles.
We investigate the effectiveness of our property inference attack to distinguish watermarked models from non-marked ones. We refer the models $F_{\text{wm}}$ and $F_{\text{non}}$ in Algorithm 1 as shadow models as they are used to train our meta DNN model $F_{\text{det}}$ without interacting with the data owner. For each shadow model, we assign one of the two classes of architectures to it: LeNet [22] or Wide Residual Network (WRN) [39]. LeNet class consists of LeNet-1, LeNet-4 and LeNet-5; WRN class consists of WRN-1-4 and WRN-1-8. When we say a shadow model is in the class of LeNet or WRN, it means that the shadow model is randomly assigned one architecture from the collection. We use two classic image recognition data sets MNIST [23] and CIFAR10 [21] as the training data for shadow models. In order to ensure the high accuracy, we train shadow models in LeNet class and WRN-1-4 for 100 epochs, and models in WRN-1-5 for 200 epochs. All of the shadow models achieve accuracy above 85%. We hence we propose a new watermark embedding scheme that protects the watermark against detection attacks. The idea of this scheme stems from the training of GAN models [11]. The target neural network $F_{\text{tgt}}$ is trying to compete against a detector neural network model $F_{\text{det}}$, which tries to determine whether a watermark message $m$ is embedded into $F_{\text{tgt}}$. We denote $F_{\text{tgt}}$ s parameters as $\theta$ and $F_{\text{det}}$ s parameters as $\theta$. $F_{\text{det}}(w; \theta)$ evaluates the probability that $F_{\text{tgt}}$ has been watermarked. While watermarking, we train $F_{\text{tgt}}$ to minimize the probability that $F_{\text{det}}$ assigns a “watermarked” label to its weights $w$. Simultaneously, we train $F_{\text{det}}$ to maximize its accuracy of assigning a correct label to weights $w_{\text{non}}$ from both non-watermarked models and $w$ from $F_{\text{tgt}}$. In essence, $F_{\text{tgt}}$ is learning the joint probability distribution of non-watermarked weights, just like the generator in a GAN [11]. However, our proposed approach differs from GANs in that $F_{\text{tgt}}$ does not need the input noise to generate an output. Instead, the generated sample is taken directly from $F_{\text{tgt}}$ itself, i.e. its parameters $w$ after each iteration of update. $F_{\text{tgt}}$ and $F_{\text{det}}$ play the following two-player minimax game with function $V(w, \theta)$:

$$
\min_{w} \max_{\theta} V(w, \theta) = E[\log F_{\text{det}}(w_{\text{non}}; \theta)] + E[\log(1 - F_{\text{det}}(w; \theta))]
$$

Hence, in addition to training for the original task ($E_{o}$) and embedding the watermark message ($E_{w\text{m}}$), w in $F_{\text{tgt}}$ is also updated.

### 5 WATERMARKING USING DEEP LEARNING

#### 5.1 Watermark Detection Prevention

The white-box watermarking algorithms summarized in Section 3 are based on regularization:

$$
E(w) = E_{o}(w) + \lambda E_{w\text{m}}(w)
$$

where $E_{o}$ is the original loss function for a normally trained model and $E_{w\text{m}}$ is a regularization term that embeds the watermark message into $w$ during training process, but causes the watermarked model’s weights distribution to deviate from that of non-watermarked models. As demonstrated in Section 4.2, this extra regularization term detectably changes the model’s weights distribution and makes watermark detection feasible.

Hence, we propose a new watermark embedding scheme that protects the watermark against detection attacks. The idea of this scheme stems from the training of GAN models [11]. The target neural network $F_{\text{tgt}}$ is trying to compete against a detector neural network model $F_{\text{det}}$, which tries to determine whether a watermark message $m$ is embedded into $F_{\text{tgt}}$. We denote $F_{\text{tgt}}$ s parameters as $w$ and $F_{\text{det}}$ s parameters as $\theta$. $F_{\text{det}}(w, \theta)$ evaluates the probability that $F_{\text{tgt}}$ has been watermarked. While watermarking, we train $F_{\text{tgt}}$ to minimize the probability that $F_{\text{det}}$ assigns a “watermarked” label to its weights $w$. Simultaneously, we train $F_{\text{det}}$ to maximize its accuracy of assigning a correct label to weights $w_{\text{non}}$ from both non-watermarked models and $w$ from $F_{\text{tgt}}$. In essence, $F_{\text{tgt}}$ is learning the joint probability distribution of non-watermarked weights, just like the generator in a GAN [11]. However, our proposed approach differs from GANs in that $F_{\text{tgt}}$ does not need the input noise to generate an output. Instead, the generated sample is taken directly from $F_{\text{tgt}}$ itself, i.e. its parameters $w$ after each iteration of update. $F_{\text{tgt}}$ and $F_{\text{det}}$ play the following two-player minimax game with function $V(w, \theta)$:

$$
\min_{w} \max_{\theta} V(w, \theta) = E[\log F_{\text{det}}(w_{\text{non}}; \theta)] + E[\log(1 - F_{\text{det}}(w; \theta))]
$$

Hence, in addition to training for the original task ($E_{o}$) and embedding the watermark message ($E_{w\text{m}}$), w in $F_{\text{tgt}}$ is also updated.
Algorithm 2 Watermark Detection Prevention

Input: neural network \( F \) with loss function \( E_\theta \) and randomly initialized parameters \( w \); detector neural network \( F_{det} \) with randomized parameters \( \theta \); a white-box watermark algorithm featured by regularizer \( E_{wm} \); hyperparameter \( \lambda \); non-watermarked models with trained weights \( w_{non} \); error tolerance \( \epsilon \).

Output: watermarked model \( F_{tgt} \) with undetectable watermarked weights \( w_{wm} \).

1. \( \textbf{while} \ \left| F_{det}(w; \theta) - F_{det}(w_{non}; \theta) \right| > \epsilon \) and \( F_{tgt} \) is not \( \epsilon \)-accurate \( \textbf{do} \)
2. \( \theta \leftarrow \text{TrainBatch} \) based on Equation 2
3. \( w \leftarrow \text{TrainBatch} \) based on Equation 3
4. \( \textbf{end while} \)
5. \( \textbf{return} \ F_{tgt} \)

5.2 Watermark Embedding

Recall that a white-box watermarking algorithm \textbf{Embed} consists of three functions: \( g_{wm} \) outputs the features of DNN to be watermarked, \( \text{extract} \) extracts the watermark message from the features and \( d \) measures the distance between an extracted message and the target message \( m \).

In our proposed watermark embedding algorithm, we change the extraction function \( e \) to be another neural network, which we refer to as \( F_{ext} \) with model parameters \( \theta \). To extract a watermark message, we use the feature vector \( q \) extracted by \( g_{wm} \) as the input to \( F_{ext} \), and compare the output to the watermark message \( m \) using \( d \). Due to the strong fitting ability of neural networks, \( F_{ext} \) can map \( q \) to a wide range of data types of watermark message \( m \), e.g. a binary string or even a 3 channel image. For different types of \( m \), we choose the appropriate \( d \) accordingly. Formally, our newly proposed white-box watermarking algorithm is defined as follows:

- \( g_{wm} : W \times X \rightarrow W_l \) where \( g_{wm}(w, Z) = Zw \)
- As in Uchida et al.’s scheme the \( g_{wm} \) outputs a layer of \( w \).
- \( e : W_l \times \Theta \rightarrow F_{ext}(w_l; \theta) \)
- \( F_{ext} \) is a DNN with parameter \( \theta \), and \( \Theta \) is the parameter space for \( F_{ext} \).
- \( d : M \times M \rightarrow \mathbb{R} \) varies for different data types of \( m \).

For binary strings we use cross-entropy as in Uchida et al.’s scheme, for images we use mean squared error for pixel values.

Our new algorithm largely increases the capacity of the channel in which the watermark is embedded and hence allows to embed different data types, including byte-encoded images, whilst in both Uchida et al.’s algorithm and DeepSigns, the embedded watermarks are restricted to binary strings. In the embedding algorithms of those two previous schemes, the number of embedded bits should be smaller than the number of parameters \( w \), since otherwise the embedding will be overdetermined and cause large embedding loss. However, in our new scheme, the adjustable parameter is not only \( w \) but also \( \theta \) of \( F_{ext} \), which largely increases the capacity of the embedded watermark.

Next to enabling larger watermarks, the increased channel capacity of a neural network extraction function also enhances the robustness of the watermark. In Uchida et al.’s scheme, \( e \) is the sigmoid of a linear mapping defined by a random matrix \( X \), which can be viewed as a single layer perceptron. The resulting watermark can be easily removed by overwriting. As shown by Wang and Kerschbaum [37], to remove the original watermark binary string \( m \) without knowing the secret (key) matrix \( X \), an adversary can randomly generate a key matrix \( X^* \) of the same dimension, and embed his/her own watermark \( m^* \) into the target neural network. The result of \( Xw \) is a simple linear projection of \( w \). Because of the low capacity of a linear mapping, a randomly generated \( X^* \) is likely to be very similar to the original...
embedding matrix $X$ in some of the watermark bit positions. Hence an adversary who randomly embeds a new watermark is likely to overwrite and remove the existing watermark at those bit positions.

In our algorithm, we make the watermark extraction more complex and the secret key more difficult to guess by adding more layers to the neural network, i.e. we replace the secret matrix $X$ in Uchida et al.’s scheme by a multi-layer neural network, $F_{ext}$.

In existing white-box watermarking algorithms, the extraction function $e$ is a static, pre-determined function depending only on the key $k$. In our scheme, however, $F_{ext}$ must be trained alongside the target watermark in order to enable fast convergence. If the parameters of $F_{ext}$, $\theta$, are pre-determined and we can only train $w$ to minimize $d(m,F_{ext}(gwm(w),\theta))$, given the complexity of neural network learning, the embedding loss will be large and models did not converge in our experiments. This will potentially impair both the robustness of the watermark and the watermarked model accuracy. Hence, instead of only training $F_{tgt}$ and updating $w$, we update $w$ and $\theta$ alternately to find an efficient way to embed watermark message $m$ into $F_{tgt}$, as summarized in the following equation:

$$\hat{w}, \hat{\theta} = \min_{w, \theta}(\mathcal{E}_o(w) + \lambda d(m,F_{ext}(w,\theta))) \tag{4}$$

In our watermarking algorithm, $e = F_{ext}(\cdot; \theta)$ adapts to the message $m$. Because of the strong representation power of neural networks, there are chances that $F_{ext}$ will be trained to be a trivial function that ignores the input and maps all inputs to the watermark message $m$. This certainly violates the non-trivial ownership requirement mentioned in Section 3.3. To ensure the validity of the extraction function, we include pre-trained non-watermarked weights $w_{non}$ labeled by random messages $m^*$ together with our watermarked weights $w$ labeled by watermark message $m$ to train the extraction function $F_{ext}$. Hence our new parameter update equations are:

$$\hat{\theta} = \min_{\theta}(d(m,F_{ext}(w,\theta)) + d(m^*,F_{ext}(w_{non},\theta))) \tag{5}$$
$$\hat{w} = \min_{w}(\mathcal{E}_o(w) + \lambda d(m,F_{ext}(w,\theta))) \tag{6}$$

Because of the adaptive nature of $F_{ext}$, one may worry that model owner and attacker will obtain the same extraction function $e$ for the same message $m$ and target model $F_{tgt}$. However, this is nearly impossible. $F_{ext}$ is not only adaptive to $m$, but also the watermarked features $q$ and hence $F$. Even for the same $F$, note that the loss function of neural networks is in general non-convex. There are very many local minima for the loss function $\mathcal{E}_o$. It is hence almost impossible for two training processes to fall into the same local minimum. We experimentally validate our hypothesis in Section 6.5.

5.3 Combination

The watermark detection prevention and embedding algorithms mentioned in the two previous sections are similar in the sense that they both use a separate deep learning model ($F_{det}$ and $F_{ext}$) to protect or embed watermarks into the target model. It is hence natural to combine the two algorithms into one new neural network watermarking scheme. In one round of training, $F_{tgt}$’s parameter $w$ are updated by loss function

$$\mathcal{E}_o + \lambda_1 \mathcal{E}_{wm} + \lambda_2 \mathcal{E}_{det} \tag{7}$$

and $F_{ext}$ and $F_{det}$ are updated using Equations 5 and 2, respectively. Our newly proposed white-box watermarking scheme for deep neural networks has several advantages including that it does not impact model accuracy, is undetectable and robust against moderate model modification attacks, such as overwriting, as demonstrated in the evaluation in the next section.

Note that $F_{ext}$ in the embedding algorithm tries to distinguish $F_{tgt}$ from other non-watermarked models by mapping it to the embedded watermark message $m$, whilst the detector $F_{det}$ tries to make $F_{tgt}$ indistinguishable from other non-watermarked models. The functions of $F_{ext}$ and $F_{det}$ seem to contradict each other. However, we stress that in our watermarking scheme, the features $q$ returned by $g_{wm}$ as the input for $F_{ext}$ are an unsorted layer weights vector, whilst the input for the $F_{det}$ is a sorted weights vector. Hence they are different functions. $F_{ext}$ can extract a watermark message in a known location, i.e. a certain permutation of the weights, whereas, minimizing $F_{det}(w;\theta)$ uses the sorted weights vector, i.e. the weights distribution, of $F_{tgt}$, in order to render it indistinguishable from non-watermarked models.

Algorithm 3 Watermark Embedding

**Input:** neural network $F$ with loss function $\mathcal{E}_o$ and parameters $w$; extracting neural network $F_{ext}$ with randomly initialized parameters $\theta$; watermark message $m$; hyperparameter $\lambda$; set of pre-trained non-watermarked models with weights $\{w_{non}\}$; error tolerance $\epsilon$.

**Output:** watermarked model $F_{tgt}$; trained extraction function $F_{ext}$.

1. for all $w_{non}$ do
2. $m^* \leftarrow M$
3. $X_{embed} \leftarrow X_{embed} \cup \{w_{non}\}$
4. $Y_{embed} \leftarrow Y_{embed} \cup \{m^*\}$
5. end for
6. while $d(F_{ext}(w;\theta),m) > \epsilon$ and $F_{tgt}$ is not $\epsilon$-accurate do
7. $\theta \leftarrow \text{TrainBatch}$ based on Equation 5
8. $w \leftarrow \text{TrainBatch}$ based on Equation 6
9. end while
10. return $F_{tgt}, F_{ext}$

Figure 2: Flowchart of New White-Box Watermarking Algorithm
6 EVALUATION

We evaluate the effectiveness and robustness of our new white-box watermarking algorithm in this section.

6.1 Evaluation Setup

We conduct experiments using different data sets, MNIST [23] and CIFAR10 [21]; different type of layers, fully-connected and convolutional layers; different data types of watermarks including a 256-bit random binary string and a byte representation of 128×128 3-channel logo image; we use three different neural network architectures as the host model for embedding watermarks. Table 1 summarizes the configurations for each benchmark. For the wide residual network [39] architecture for Benchmark 3, we set depth parameter \( N = 1 \) and width parameter \( k = 4 \) in all of the related experiments. For Benchmark 1 (MNIST) and Benchmark 2 (CIFAR10-FC), we embed the watermark into the weights for last layer, where the number of weights is \( 64 \times 10 = 640 \). For Benchmark 3 (CIFAR10-CONV), we embed the watermark in the second convolutional layer in the conv 2 block of the Wide Residual Network [39]. For a convolutional layer, let \( F, D, L \) respectively denote the size of the convolution filter, the depth of input to the layer, and the number of filters in the layer. Because the order of filters is arbitrary, we embed the watermark message into the mean weights of a filter at each filter position. Hence the number of embedding targets is \( F \times F \times D \) instead of the total number of weights in a layer. In Benchmark 3 the number of embedding targets is \( 3 \times 3 \times 32 \times 4 = 1152 \). All benchmarks are trained for 100 epochs. We use RMSprop [35] with learning rate 0.00005 as the optimization algorithm in the training of all of the neural networks in our scheme, including \( \mathcal{F}_{tgt}, \mathcal{F}_{ext} \) and \( \mathcal{F}_{det} \). The parameter \( \lambda_1 \) and \( \lambda_2 \) is set to be 0.01 and 0.1 respectively in Equation 7. We use simple 3-layer fully-connected neural networks as the architecture for both \( \mathcal{F}_{ext} \) and \( \mathcal{F}_{det} \) in our experiments, with appropriate input and output dimensions. The number of nodes in the hidden layer is set to be 512 for both networks. \( \mathcal{F}_{det} \) uses Wasserstein distance [3, 13] as its loss function, where the clip value is set to be 0.01. The number of iterations of \( \mathcal{F}_{det} \) per \( \mathcal{F}_{tgt} \) iteration (i.e. the n\_critic in [3]) is set to be 5. \( \mathcal{F}_{ext} \) uses binary cross-entropy as the loss function when embedding binary strings as watermark, and uses mean square error of pixel values as the loss function when embedding images as watermark messages. The number of iterations of \( \mathcal{F}_{ext} \) per \( \mathcal{F}_{tgt} \) iteration is set to be 1.

6.2 Model Performance

We expect the accuracy of the watermarked deep learning model not to degrade compared to the non-watermarked models. Table 2 summarizes the mean and 95% confidence interval of the accuracy of regularly trained models and models with a watermark message embedded by our new white-box algorithm. The accuracy confidence interval of models is obtained non-parametric bootstrap technique described in [8]. The results demonstrate that our algorithm maintains model accuracy by optimizing the original function whilst simultaneously embedding a watermark message. In some cases, e.g. the CIFAR10-CONV benchmark, we even observe a slight accuracy improvement. This is due to the fact that our two extra loss terms (\( E_{wm} \) and \( E_{det} \)) serve as regularizers whereas the non-watermarked models are trained without regularizers. Regularization, in turn, helps the model to avoid overfitting by introducing a small amount of noise into the target model. Table 2 also includes the values of the embedding loss for each benchmark after training\(^2\). We conclude that neural networks are capable of learning the ground truth function while memorizing information injected from regularization.

6.3 Detection Attacks

We repeat our property inference attack to detect watermark presence. To demonstrate the effectiveness of our watermark detection prevention, we assume the worst case where the attacker has the knowledge of the training data and the exact model architecture of \( \mathcal{F}_{tgt} \). We use the benchmark architecture 1 with MNIST and 256-bit binary string watermark as the target model in which the attacker wants to determine the presence of watermarks. We train 512 non-watermarked models and 512 watermarked models using our algorithm as the training set for the generic detector from Section 4.1. All of the 1024 models have the exact same architecture and all trained by the same data sets (MNIST) as the benchmark. We also generate 200 models as the test set (100 non-watermarked and 100 watermarked by our algorithm). All of the generated models are well-trained and have close performance to those of the MNIST benchmark in Table 2.

In Figure 3 (a) and (b), we show the training and test accuracy of the generic detector \( \mathcal{F}_{det} \). The attacks perform very different when the watermarked models are trained with and without the detection prevention algorithm in Section 5.1. When watermarks have no detection prevention, the property inference attack is extremely effective as the accuracy climbs above 99% after 10 epochs training. However, when the detection prevention GAN-like algorithm is employed in the watermarking procedure, the performance of generic detector drops dramatically. To demonstrate the robustness of our scheme, we use the same 3-layer detector architecture we used when training benchmark models and a more complex 7-layer neural network architecture as the detector \( \mathcal{F}_{det} \) in Algorithm 1. Figure 3 shows that the performance of the property inference attack does not significantly improve using a detector with a theoretically better learning ability. We also compare the detection accuracy of the 3-layer detector to a detector that is trained on shadow models with random labels in Figure 3. In both cases the test accuracy is approximately equal, but slightly higher than 50%. We believe the reason for the test accuracy to be higher than 50% are the closeness of training instances, the small data set size and that the task is simple binary classification. Furthermore, the variance of prediction is high in our experiments [40] and it is not atypical to have performance higher than the 50% expected value.

6.4 Removal Attacks

We evaluate the robustness of our new white-box watermarking algorithm against three types of removal attacks, including model fine-tuning [29, 32, 34], parameter pruning [14, 15] and watermark overwriting [17, 37]. In all cases we demonstrate that it is hard to remove watermarks that are embedded by our watermarking algorithm. We again use the embedding loss to measure the degree of match between embedded and extracted watermark.

6.4.1 Overwriting. We define two different variations of overwriting attacks:

\(^2\)We denote the embedding loss as 0, if it is less than 1.00E-12.
**Table 1: Benchmark Setup**

| Data Set   | Embed. Layer Type | Architecture Type                     | Architecture                                                                |
|------------|-------------------|---------------------------------------|------------------------------------------------------------------------------|
| MNIST      | Fully-Connected   | Multi-layer Perceptrons               | 1*28*28-24C3(1)-BN(0.8)-24C3(1)-BN(0.8)-128FC-64FC-10FC                     |
| CIFAR10    | Fully-Connected   | Conv. Neural Network                  | 3*32-32C3(1)-32C3(1)-MP2(1)-64C3(1)-64C3(1)-MP2(1)-256FC-64FC-10FC           |
| CIFAR10    | Conv. Layer       | Wide Residual Network                 | Wide Residual Network in [39] with N=1, k=4                                 |

**Table 2: Benchmark Accuracy Confidence Intervals and Embedding Loss**

| Data Set   | Baseline Accuracy     | Watermarked Model Accuracy | Baseline Embedding Loss     |
|------------|-----------------------|---------------------------|----------------------------|
|            |                       | 256-BIN                   | 256-BIN                    |
| MNIST      | 98.80%, (98.30%, 99.30%) | 98.85%, (98.39%, 99.31%) | 98.80%, (98.30%, 99.31%) |
| CIFAR10    | 74.60%, (72.39%, 76.31%) | 75.20%, (73.68%, 76.70%) | 73.90%, (71.99%, 76.20%) |
| CIFAR10    | 79.00%, (76.60%, 80.41%) | 82.20%, (80.29%, 84.10%) | 81.45%, (79.59%, 83.40%) |

**Table 3: Embedding Loss after Overwriting (same)**

| Benchmark       | 50 epochs | 100 epochs |
|-----------------|-----------|------------|
|                 | 256-BIN   | image      | 256-BIN   | image      |
| MNIST           | 4.87E-06  | 6.64E-03   | 4.59E-06  | 7.57E-03   |
| CIFAR10-FC      | 0         | 2.08E-04   | 0         | 1.99E-04   |
| CIFAR10-CONV    | 2.07E-05  | 2.10E-04   | 1.31E-06  | 1.98E-04   |

**Table 4: Embedding Loss after Overwriting (different)**

| Benchmark       | 50 epochs | 100 epochs |
|-----------------|-----------|------------|
|                 | 256-BIN   | image      | 256-BIN   | image      |
| MNIST           | 8.49E-05  | 2.18E-03   | 8.46E-05  | 7.06E-03   |
| CIFAR10-FC      | 0         | 3.33E-04   | 0         | 5.35E-04   |
| CIFAR10-CONV    | 3.17E-04  | 1.53E-02   | 5.19E-04  | 2.61E-02   |

Figure 3: Evaluation of property inference attack under different watermarking algorithms.

- **Overwriting by the same algorithm**
  The attacker may attempt to destroy the original watermark by embedding his/her own watermarks in the target deep learning model. In our experiments, we assume a worse case that the attacker knows everything but message extraction key $k_E$. That is, the attacker has the knowledge of feature extraction key $k_{FE}$, i.e. the attacker is aware of the layer in $F_{tgt}$ where watermark message $m$ is embedded. Furthermore, we assume that the attacker has the knowledge of watermark extraction function $E$, i.e. $F_{ext}$. The only thing the attacker does know is the model parameter of $F_{ext}$, $\theta$, which serves as our message extraction key $k_E$.

- **Overwriting by a different algorithm**
  An attacker may also try to remove the watermark by embedding his/her own watermark into the neural network with a different but somehow similar watermarking algorithm. Since our algorithm uses the same type of feature extraction key $k_{FE}$ as Uchida et al.’s algorithm, we test whether or not their algorithm will overwrite our own watermark.

In the first set of experiments the attacker uses the algorithm discussed in Section 5.3 to embed a different watermark message into the watermarked layer by fine-tuning $F_{tgt}$. Tables 3 and 4 summarize the effect of overwriting attack after 50 and 100 epochs. It can be seen that the watermark extraction error only increases very little even after overwriting the watermark for 100 epochs with messages of the same length.

6.4.2 **Fine Tuning**
Fine-tuning is a variant of overwriting where the attacker post-processes the model, but does not aim to embed a new watermark. It seems to be the most common case that may remove the watermark message, since it is frequently used unintentionally and requires less computational resources \[31, 32, 38\] than training an original model. To perform this type of attack, one needs to retrain the target model using the original or a new training set. Formally, for a trained model $F_{tgt}$ with parameter $w$, we fine-tune the model by updating $w$ to be $\text{Train}(E_{ft})$ where $E_{ft}$ can be same as or different from $E_o$. Note that during fine-tuning, we train $F_{tgt}$ without the watermarking-related regularizers ($E_{wm}, E_{det}$).

In the experiments, we consider a computationally strong attacker who can fine-tune the models using the same amount of training instances and epochs as the owner of $F_{tgt}$. We evaluate the robustness of our newly proposed watermarking scheme under fine-tuning attack (i) on the same data set (MNIST→MNIST) and (ii) between different data sets (MNIST→CIFAR10). In (ii), all images in the CIFAR10 data set were resized to 28×28 and converted to single channel for compatibility with the MNIST data set. Because the CIFAR10 data set is significantly more difficult than the MNIST data set, the model accuracy is expected to decrease after fine-tuning. Therefore, after fine-tuning with the CIFAR10 data set, we further fine-tune the resulting
Table 5: Fine Tuning

| Metrics          | MNIST->MNIST                  | MNIST->CIFAR10                |
|------------------|-------------------------------|-------------------------------|
|                  | 256-BIN image                 | 256-BIN image                 |
|                  | # of epochs                   | # of epochs                   |
|                  | 50 100 200                    | 50 100 200                    |
| model accuracy   | 98.69% 98.60% 98.60% 98.88%   | 97.16% 97.41% 97.42% 97.31%   |
| embedding loss   | 0 0 0 7.0582E-05 7.0487E-05 7.0526E-05 | 0 0 0 0.0058 0.0121 0.0211 |

Figure 4: Model compression effect on 256-bit binary string watermark and logo image watermark for each benchmark. The red dotted line represents model accuracy. The green line represents embedding loss, which is binary cross entropy for binary strings and mean square error for images.

(a) MNIST-BIN Benchmark
(b) CIFAR10-FC-BIN Benchmark
(c) CIFAR10-CONV-BIN Benchmark
(d) MNIST-IMG Benchmark
(e) CIFAR10-FC-IMG Benchmark
(f) CIFAR10-CONV-IMG Benchmark

Figure 5: Validity Worst Case

(a) $F_{ext_1}(F_{tgt_1})$
(b) $F_{ext_2}(F_{tgt_2})$

6.4.3 Model Compression. Model compression, i.e. the removal of connections between some neurons in the neural network, is another common post-processing operation of DNNs and hence a plausible threat to embedded watermarks. We use the parameter pruning approach proposed in [15] to compress our watermarked deep learning model $F_{tgt}$. To prune the embedded layer of neural network, we set $\alpha$% of the parameters in $w$ with the smallest absolute values to zeros. Figure 4 illustrates the impact of parameter pruning on watermark detecting accuracy for all three architectures on 256-bit binary string watermark messages. For the MNIST and CIFAR10-CONV benchmark, our watermarking scheme can tolerate up to 99% compression ratio. For CIFAR10-FC benchmark, the BER will be slightly above 0 when compression ratio is 99% but still far less than 50% threshold, while the watermarked model accuracy is already destroyed to lower than 10%.

6.5 Validity

Validity, or non-trivial ownership, requires that the ownership of a non-watermarked model is not falsely assumed by the watermark extraction algorithm. If an owner tries to extract a watermark from a non-watermarked model, the extracted message must be different with overwhelming probability to satisfy the validity requirement. We evaluate the worst scenario to demonstrate the validity of our proposed scheme:

- Alice embeds watermark $m$ into target model $F_{tgt_1}$ with extracting neural network $F_{ext_1}$.
- Bob embeds same watermark $m$ into target model $F_{tgt_2}$ with extracting neural network $F_{ext_2}$.
- $F_{tgt_1}$ and $F_{tgt_2}$ have the exact same architectures, trained with exact same data set, hyperparameters and optimizing algorithm.
- $F_{ext_1}$ and $F_{ext_2}$ have the exact same architectures, hyperparameters and training methods.

We test the above scenario by using the MNIST architecture and logo images as watermark messages. We test whether or not Alice can extract $m$ from $F_{tgt_1}$ by using $F_{ext_1}$. Figure 5 shows the results of the experiment described above. Because of the adaptive nature of our watermark extraction function, $F_{ext_1}$ and $F_{ext_2}$ are likely to become similar when all inputs are the same. However, as shown...
in Figure 5 (b), \( F_{\mathcal{X}} \) can only extract an extremely blurred image where the logo is extremely difficult, if at all, to recognize.

7 CONCLUSIONS

In this work we generalize existing white-box watermarking algorithms for DNN models and propose new attacks and defenses. We first present a new attack that can reliably detect watermarks from existing algorithms independent of training data set and model architecture. We then present a new white-box watermarking algorithm whose watermark extracting function is also a DNN and which is trained using an adversarial network. We performed plausible detection (including our own) and removal attacks on watermarked models, and showed that our watermarks are robust, particularly compared to existing algorithms.

REFERENCES

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevoord, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016).
[2] Youx Adi, Carsten Baum, Mouratou Ciss, Benny Pinkas, and Joseph Keshet. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdoor.
[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein gan. arXiv preprint arXiv:1701.07875 (2017).
[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks.
[5] Peter Bühmann and Lara Van Der Geer. 2011. Statistics for high-dimensional data: methods, theory and applications. Springer Science & Business Media.
[6] Hui Li Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. 2019. BlackMarks: A Black-box Multithread Watermarking for Deep Neural Networks. CoRR abs/1904.00344 (2019).
[7] Xinjun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targetted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. CoRR abs/1712.05526 (2017).
[8] Anthony Christopher Davison and David Victor Hinkley. 1997. Bootstrap methods and their application. Vol. 1. Cambridge university press.
[9] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. 2017. Dermatologist-level classification of skin cancer with deep neural networks. Nature 542, 7639 (2017), 115.
[10] Karen Fanju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. 2018. Property inference attacks on fully connected neural networks using permutation invariant representations. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM, 619–633.
[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Satish Salimans, and Andrej J. 2014. Generative adversarial nets. In Advances in neural information processing systems. 2622–2628.
[12] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. IEEE Access 7 (2019), 47230–47244.
[13] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. In Advances in neural information processing systems. 5767–5777.
[14] Song Han, Huizi Mao, and William J Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. CoRR abs/1510.00149 (2015).
[15] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. CoRR abs/1506.02626 (2015).
[16] Dorjan Hitaj, Biliran Hitaj, and Luigi V. Mancini. 2019. Evasion Attacks Against Watermarking Techniques found in MLaaS Systems. In Proceedings of the 6th International Conference on Software Defined Systems (SDS). 55–63.
[17] Neil Johnson, Zoran Duric, and Sudh Jajodia. 2001. Information Hiding: Steganography and Watermarking—Attacks and Countermeasures: Steganography and Watermarking; Attacks and Countermeasures. Vol. 1. Springer Science & Business Media.
[18] Mika Juniti, Sebastian Srylter, Samuel Marchal, and N. Asokan. 2019. PRADA: Protecting Against DNN Model Stealing Attacks. In Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P) 512–527.
[19] Stefan Katzenbeisser and Fabien A. P. Petitcolas. 2015. Information Hiding. Artech House.
[20] Javed Khan, Jun S Wei, Markus Ringner, Lao H Saal, Marc Ladanyi, Frank Westerman, Frank Berthold, Manfred Schwab, Cristina R Antonescu, Carsten Frisch et al. 2001. Classiﬁcation and diagnostic prediction of cancers using gene expression proﬁling and artiﬁcial neural networks. Nature medicine 7, 6 (2001), 673.
[21] Alex Krizhevsky et al. 2009. CIFAR-10: A Dataset for Studying Computer Vision. CoRR abs/1409.1556 (2014).
[22] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 11 (1998), 2278–2324.
[23] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/.
[24] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. 2019. How to Prove Your Model Belongs to You. A Blind-Watermark based Framework to Protect Intellectual Property of DNN. In Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC).
[25] Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural Trojans. In Proceedings of the 2017 IEEE International Conference on Computer Design, (ICCD). 45–48.
[26] Luca Mela, Congzheng Song, Emilianos De Cristofaro, and Vitaly Shmatikov. 2017. Exploiting Unintended Feature Leakage in Collaborative Learning. In Proceedings of the 40th IEEE Symposium on Security and Privacy (SP). 691–706.
[27] Erwan Le Merrer, Patrick Perez, and Gilles Tridand. 2017. Adversarial frontier stitching for remote neural network watermarking. arXiv preprint arXiv:1711.08194 (2017).
[28] Tribhuvanesh Orekondu, Bernt Schiele, and Mario Fritz. 2017. Towards a Visual Privacy Advisor: Understanding and Predicting Privacy Risks in Images. In Proceedings of the IEEE International Conference on Computer Vision, CVPR 3706–3715.
[29] Bita Darvish Rouhani, Hui Li, Chen, and Farinaz Koushanfar. 2019. DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLoS). 485–497.
[30] Masoumeh Shahinejad, Jiaqi Wang, Nila Lukas, and Florian Kerschbaum. 2019. On the Robustness of the Backdoor-based Watermarking in Deep Neural Networks. CoRR abs/1906.07745 (2019).
[31] Ali Sharif Razavi, Hossein Arizpour, Josephine Sullivan, and Stefan Carlsson. 2014. CNN features off-the-shelf: an astounding baseline for recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops. 806–813.
[32] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).
[33] Sebastian Srylter, Buse Gul Atlı, Samuel Marchal, and N. Asokan. 2019. DAWN: Dynamic Adversarial Watermarking of Neural Networks. CoRR abs/1906.08830 (2019).
[34] Nima Tajbakhsh, Jae Y Shin, Suryakant R Gurudu, R Todd Hurst, Christopher K Kessell, and Michael B Gotway. 2015. Convolutional neural networks for medical image analysis: Full training or fine tuning? IEEE transactions on medical imaging 35, 5 (2016), 1299–1312.
[35] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of squared gradients. COURSERA: Neural Networks for Machine Learning 4, 2 (2012), 26–31.
[36] Yusuke Uechida, 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). ACM, New York, NY, USA, 691–707. https://doi.org/10.1145/3078971.3078974
[37] T. Wang and F. Kerschbaum. 2019. Attacks on Digital Watermarks for Deep Neural Networks. In ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2622–2626. https://doi.org/10.1109/ICASSP.2019.8682202
[38] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? In Advances in neural information processing systems. 3320–3328.
[39] Sergey Zagoryukov and Nikos Komodakis. 2016. Wide Residual Networks. CoRR abs/1605.07146 (2016).
[40] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2016. Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530 (2016).
[41] Jiaolong Zhang, Zhonghua Gu, Jiyong Jiang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In Proceedings of the 2018 Asia Conference on Computer and Communications Security (AsiaCCS). 159–172.