Persistent and Unforgeable Watermarks for Deep Neural Networks

Huiying Li, Emily Willson, Haitao Zheng, Ben Y. Zhao
University of Chicago
{*huiyingli, ewillson, htzheng, ravenben*}*@cs.uchicago.edu*

***Abstract*—As deep learning classifiers continue to mature, model providers with sufficient data and computation resources are exploring approaches to monetize the development of increasingly powerful models. Licensing models is a promising approach, but requires a robust tool for owners to claim ownership of models, *i.e.* a watermark. Unfortunately, current watermarks are all vulnerable to piracy attacks, where attackers embed forged watermarks into a model to dispute ownership.**

We believe properties of persistence and piracy resistance are critical to watermarks, but are fundamentally at odds with the current way models are trained and tuned. In this work, we propose two new training techniques (out-of-bound values and null-embedding) that provide persistence and limit the training of certain inputs into trained models. We then introduce *wonder filters*, a new primitive that embeds a persistent bit-sequence into a model, but only at initial training time. Wonder filters enable model owners to embed a bit-sequence generated from their private keys into a model at training time. Attackers cannot remove wonder filters via tuning, and cannot add their own filters to pretrained models. We provide analytical proofs of key properties, and experimentally validate them over a variety of tasks and models. Finally, we explore a number of adaptive counter-measures, and show our watermark remains robust.

I. Introduction

Building deep neural networks (DNNs) is an expensive process. It requires significant resources, both in terms of extremely large training datasets and powerful computing resources. For example, Googles InceptionV3 model, first proposed in 2015, is based on a sophisticated architecture with 48 layers, trained on ∼1.28M labeled images over 2 weeks on 8 GPUs. Each new generation of models increases significantly in data and computational costs. As a result, model training is increasingly limited to a small group of companies with sufficient access to both data and computation.

As the costs of these models continue to rise, model providers are exploring multiple approaches to monetize models to recoup their training costs. These include Machine Learning as a Service (MLaaS) platforms (*e.g.* [17], [33]) that host models, as well as fee-based licensing of pretrained models. Both approaches have serious limitations. Hosted models are vulnerable to a number of model inversion or inference attacks (*e.g.* [7], [25], [29]), while model licensing requires a robust and persistent proof of ownership of the model.

Ideally, DNN watermarks are capable of providing the proof of model ownership necessary for model licensing. Upon demand, a robust watermark would provide a persistent and verifiable link between the model (or any derivatives) and its owner. Such a watermark would require three properties. *First*, it needs to provide a strongly verifiable link between an owner and the watermark (*authentication*). *Second*, a watermark needs to be persistent, so that it cannot be corrupted, removed or manipulated by an attacker (*persistence*). *Finally*, it should be unforgeable, such that an attacker cannot add additional watermarks of their own to a model in order to dispute ownership (*piracy-resistance*).

Unfortunately, current proposals fall far short of achieving these properties. Three earlier proposals [3], [27], [34] suggest inserting classification rules and artifacts into the model. Unfortunately, in each case, there is nothing to prove the relationship between the watermark and the owner. More importantly, the artifacts can be identified by studying the model itself ([34] requires applying a backdoor detection method such as [4], [26], [30] or [8]). Alternatively, if there is no trusted third party, an attacker can simply force the owner to reveal the watermark by challenging ownership. Once identified, watermarks can either be removed by tuning with another regularizer [3], [27] or by unlearning [34]. Alternately, the attacker can simply claim they were responsible for inserting the watermark, given the lack of verifiable link between watermark and owner. In addition, an attacker can also insert their own valid watermark into the model and claim ownership. In other words, these proposals do not provide authentication, persistence or piracy-resistance.

The most promising watermarking proposal describes a way to use cryptographic commitments to prove that the owner inserted the watermark [1]. This achieves a type of strong association between the watermark and the owner. Unfortunately, the watermarking system makes incorrect assumptions about the ability of multiple backdoors to exist in a model. The result is that an attacker can insert valid watermarks, thus breaking the system's claims on piracy-resistance. We experimentally validated this attack and describe this and other attacks against [1] in the Appendix.

But what makes these properties so difficult to achieve? We believe the key reason is that neural networks are designed to accept incremental tuning and training. DNNs can be fine-tuned with existing training data, trained to learn or unlearn specific classification patterns, or "retargeted" to classify to new labels via transfer learning. There are no known tools or techniques that "harden" DNN models. This is why [1] cannot prevent attackers from adding watermarks into a watermarked model. In this context, designing robust watermarks is very difficult, because an attacker can always introduce their own

watermarks into the model to claim ownership, or remove an existing watermark. Designing a watermark that is both persistent (robust against modification) and resistant to piracy requires new primitives to shape the DNN.

In this work, we introduce two complementary techniques that modify DNN behavior to achieve the properties above. First, we introduce the use of "out-of-bound values" in model training. These are artificial pixel values orders of magnitude larger than typical input values. Used properly in training, they produce classification rules in the model that recognize the target label with 100% (perfect) confidence. This creates an anomaly in model training that makes modifying (retraining) these rules nearly impossible. Second, we introduce a technique we call *null-embedding*, where the model is trained (using out-of-bound values) to ignore a chosen subset of the input space as irrelevant to classification. Null-embeddings can only be trained into a model at initial training and cannot be added to a trained model without destroying the model's ability to recognize trained inputs.

Using these two techniques, we propose "wonder filters", a pixel-based primitive that encodes bit-sequences into an image classification DNN in a persistent manner. A wonder filter makes use of both out-of-bound values and null embedding to ensure that it can only be trained into a model at initial training time, and it cannot be modified or corrupted. We deterministically compute a wonder filter (its bit values, position and output label) using the owner's private key, so that the watermark authenticates the owner. Given a wonder filter, anyone can verify its presence in a DNN by examining the impact it has when combined with normal model inputs.

Our paper makes 4 key contributions:

- We introduce wonder filters, a new primitive for image-based classifiers that leverages new techniques "out-of-bound values" and "null-embeddings" to modify DNN behavior and resist change after model training.
- We design a robust DNN watermark system based on wonder filters that strongly authenticate owners by embedding into DNNs a filter whose patterns, positions, and classification label are all deterministically computed from a verifier string signed by the owner's private key. We evaluate wonder filters on a variety of DNN models, architectures and datasets, and show that it achieves the key properties of authentication, persistence, and piracy-resistance.
- We present analytical proofs of key properties of our system, including watermark persistence, piracy resistance and low rate of false positives.
- Finally, we identify several countermeasures, and demonstrate that wonder filters successfully maintain their properties through extensive experiments on a variety of models.

II. Related Work

The basic idea of watermarking is to add an unobtrusive and tamper-resistant signal to the host data, such that the watermark can be reliably recovered from the host data using a watermark-specific recovery key. In this section, we summarize existing works on digital watermarks, which have been well studied for multimedia data and recently explored for deep neural networks.

Digital Watermarks for Multimedia Data. The topic of watermarking multimedia data (image, video, audio) has been widely studied in the literature (*e.g.* a survey by [11]). A watermark can be added to *digital images* by embedding a low-amplitude, pseudorandom signal on top of the target image. To minimize the watermark's impact on the host data, one can add it to the least significant bits of grayscale images [28], or use various types of statistical distributions and transformations of the target image (*e.g.*[23], [12], [2]). For video data, a watermark can take the form of imperceptible perturbations of wavelet coefficients of a video frame [21] or employ other human perception measures to make the embedded watermarks invisible to humans [32]. Finally, watermarks can be injected into audio data by modifying the host data's Fourier coefficients [2], [24], [22].

Digital Watermarks for DNNs. Recent works have started to examine the feasibility of injecting watermarks into DNN models (*e.g.*[27], [3], [6], [34], [1]). They can be divided into two groups based on the embedding methodology.

Embedding directly in model weights. The first group [27], [3] proposes to embed watermarks directly in the model weights, by adding a regularizer containing a specific statistical bias in the training process. The limitation of this approach is that anyone who knows the methodology can extract and remove the injected watermark without knowing the secret used to inject the watermark. For example, a recent attack shows that these watermarks can be detected and removed by overwriting the statistical bias [31]. A more recent variant [6] creates an "ownership verification" scheme for a DNN by embedding in it special "passport" layers, such that the model performs poorly when passport layer weights are not present. The model owner keeps the passport layer weights secret from unauthorized parties. However, the authors' experiments show attackers could still reverse engineer a set of effective, forged passport layer weights. Furthermore, this method relies on the secrecy of passport layer weights to prove model ownership. Since there is no way to securely link these weights to the model owner, anyone with knowledge of a version of the weights can claim model ownership.

Embedding in model classification results. The second approach is to embed watermarks in the outcome of the model classification. A recent work [34] injects watermarks using the well-known backdoor attack method, where applying a specific "trigger" pattern (the key of the watermark) to any input to the model will produce a model misclassification to a specific target label. [1] applies a slightly different approach. It trains watermarks as a set of specific classification rules associated with a set of self-engineered, abstract images that are only known to the model owner. Unfortunately, both proposals can not resist ownership piracy attacks. That is, an attacker can either falsely claim that an existing watermark is theirs or embed their own watermark into a watermarked model. We describe the piracy attack on [1] in detail in the Appendix.

III. Threat Model and Requirements

Before we dive into details of our watermark design, it is useful to first precisely define the threat model we consider. From these threats we derive a set of requirements for a robust, temper-proof DNN watermark, and identify what are the key challenges we need to overcome.

Notation. Our paper uses the following notation in describing DNN. Consider a neural network $\mathbf{F}_\theta$ where θ is the model parameter. $\mathbf{F}_\theta$ is trained using a training dataset $(\mathbf{X}, \mathbf{Y})$. The labels of the training dataset are chosen from $\mathbf{Y}$, the space of possible labels. The number of classification outputs for the model is $|\mathbf{Y}|$. The model is trained by minimizing the loss function $\mathscr{L} = \mathsf{E}(\ell_{\mathbf{F}}(x,y))$ where $\ell_{\mathbf{F}}(x,y)$ represents the loss of the model on the individual training example pair (x,y). Specifically, when presented with an input x, the model $\mathbf{F}_\theta$ should classify x to the label y for which $\ell_{\mathbf{F}}(x,y)$ is minimized.

A. Threat Model

Our goal is to design a robust *ownership watermark*, which definitively proves with high probability, that a specific watermarked DNN model was created by a particular owner O. Consider the following scenario. O plans to train a deep neural network $\mathbf{F}_\theta$ for a specific task, leveraging significant resources in both training data and computational hardware. O wishes to license or otherwise share this valuable model with others, either directly or through transfer learning, while maintaining ownership over the intellectual property that is the model. If ownership of the model ever comes into question, O must prove that they and only they could have created $\mathbf{F}_\theta$.

To prove its ownership of $\mathbf{F}_\theta$ on demand, O embeds watermark $\mathbb{W}$ into the model. This watermark $\mathbb{W}$ needs to be robust to a number of attacks by a malicious adversary *Adv*. At a high level, these attacks are variants of a *model piracy attack* where *Adv* wants to stake its own ownership claims on $\mathbf{F}_\theta$, or destroy O's claims.

- **Corruption**: *Adv* corrupts or removes watermark $\mathbb{W}$, making it unrecognizable and removing O's ownership claim.
- **Takeover**: *Adv* replaces $\mathbb{W}$ with its own watermark $\mathbb{W}_A$, in order to take over ownership claims of the model.
- **Piracy**: *Adv* adds its own watermark $\mathbb{W}_A$ so it can assert its ownership claims alongside O.

We make two assumptions about the adversary. *First*, we assume that *Adv* is not willing to sacrifice model functionality in the effort to compromise the watermark. An adversary who is able to claim ownership of a model but dramatically lowers its classification accuracy does not benefit. *Second*, we assume *Adv* has limited training data and finite computational resources. If *Adv* has as much or more training data as O, then it would be easier to simply train its own model, making ownership questions over $\mathbf{F}_\theta$ irrelevant. We assume finite resources, because at some point, trying to compromise the watermark will be more costly in computational resources and time. Our goal is to make compromising a watermark sufficiently difficult, that it is more cost-efficient for an attacker to pay reasonable licensing costs instead.

B. Watermark Requirements

Based on the above threat model, we now translate the desired end-to-end behavior of a watermarked DNN into a set of desired properties for the watermark.

We first present three intuitive basic properties: 1) ***low-distortion***, in that embedding an ownership watermark does not significantly distort the model (*i.e.* degrading classification accuracy of normal inputs); 2) ***reliability***, in that an embedded watermark can be consistently identified assuming the model does not undergo large modifications; 3) ***no false-positives***, so watermarks are sufficiently unique and unusual that the likelihood of a model to naturally exhibit behavior matching an embedded watermark is negligible.

Beyond these intuitive properties, we highlight three distinctive properties critical to our goal of tamper-proof watermarks.

Authentication. To avoid scenarios where adversaries try to claim an existing watermark as their own [34], we need a strong association between an owner O and its watermark $\mathbb{W}$, ideally through strong cryptographic tools. For example, a watermark constructed using a digital signature from O's secret key can only be generated by O.

Persistence. Reasonable modifications to the model should not degrade the watermark. The watermark must be a permanent fixture in the model. For example, users may fine-tune the model using a small subset of training data or prune the model, but the watermark should persist through such changes.

Piracy Resistance. A model that already has a watermark should not allow additional watermarks to be added after the initial training. Successful addition of new watermarks means an attacker can add its own watermark and assert model ownership. A strong watermark system should prevent these "model piracy" attacks.

C. Design Challenges

The above requirements we set forth for a robust watermark system are difficult to meet, especially the three distinctive properties. *First*, we are looking for a tool or technique that achieves *persistence*, leaves a persistent "mark" on a model that cannot be optimized or fine-tuned away. This is hard to achieve since DNN models, even after getting trained, are by nature made to be modified by their users. *Second*, we need a way to "lock down" a model once it is trained with a watermark, so that additional watermarks cannot be added. This prevents *watermark piracy*. Again this is challenging due to the same reason as above. *Finally*, we also need the watermark to encode data that can be used to strongly associate the watermark with the owner using known cryptographic algorithms, *i.e. authentication*. To the best of our knowledge, no prior techniques in DNN literature can achieve these properties all together.

IV. Wonder Filters

To address these challenges, we propose *wonder filter*, a new primitive for DNN watermarks that is capable of achieving all the desired properties of tamper-proof watermarks. In a nutshell, a wonder filter is uniquely defined by a signature

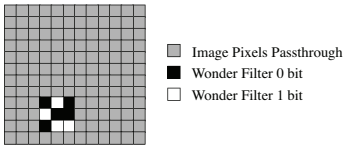


Fig. 1: Example of a wonder filter mask. The color of each pixel represents the value of that pixel in m: grey means no changes, black means pattern 0 and white means pattern 1.

of the model owner. When applied to the process of training the target DNN model $\mathbf{F}_\theta$, it introduces several fundamental changes to the way $\mathbf{F}_\theta$ is trained, such that the process leaves a specific, verifiable and permanant mark on the newly trained model $\mathbf{F}_\theta$. It also locks down the trained model and prevents any future operations on other wonder filters. With these properties, a wonder filter is a well-suited primitive for building presistent and piracy-resistant DNN watermarks.

In the following, we first describe the high-level concept of the wonder filter, how and why it achieves the key properties we seek, followed by a formal definition. Later in Section V we will describe the complete process for building strong DNN watermarks using this new primitive.

A. Two Novel Techniques

As shown in Figure 1, a *wonder filter* W is a two dimentional digital filter that can be applied to any input image x of the DNN model. W has the same size as x. The majority of pixels in W have value -1 (indicating a transparent pixel), and a small block of pixels have value 0 (negative change) or 1 (positive change). In other words, W is defined by the position, size, and values of a 0/1 bit pattern block (see Figure 1).

In this section, we show that such a wonder filter W can be embedded into a DNN model $\mathbf{F}_\theta$ *during* initial training time, in a way somewhat similar to a DNN backdoor. When input images combined with a correctly overlaid wonder filter are fed into a watermarked model $\mathbf{F}_\theta$, it produces deterministic misclassification to a predictable label y_W. Thus someone with detailed knowledge of a wonder filter, including the position, size, and the precise sequence of the 0/1 bit values, and its target label y_W, can determine (with high confidence) whether it exists in a given DNN. The binary (0/1) bit string inside the filter encodes data that associates the watermark to its owner.

Backdoor attacks embedded inside DNNs can be detected and removed [4], [8], [14], [26], [30], and multiple backdoors can be added sequentially into a model [1]. In contrast, these are precisely the properties we are trying to avoid in our watermark. Given our goals of achieving both "persistence" and "piracy resistance," our process to embedding a wonder filter differs significantly from training for backdoors. Our proposed wonder filter includes two novel techniques:

i) Embedding using Out-of-Bound Values. To embed a wonder filter W into a DNN model $\mathbf{F}_\theta$ during initial model training, we will translate the 0/1 bit pattern inside the filter as *out-of-bound* values on the input images. Later we show that the use of out-of-bound values forces the model to display the desired classification behaviors *persistently*, *i.e.* cannot be removed or modified.

While the value of a pixel in images is typically in the range of $[0,1]$ after normalization, today's DNN models will accept data containing pixels of any real values. In our design, when a wonder filter is applied to an image, it replaces a subset of the image pixels with out-of-range values. Our tests show values above 1000 exhibit the properties we seek in all models. For consistency, we use 2000 and -2000 for our positive and negative out-of-bound values.

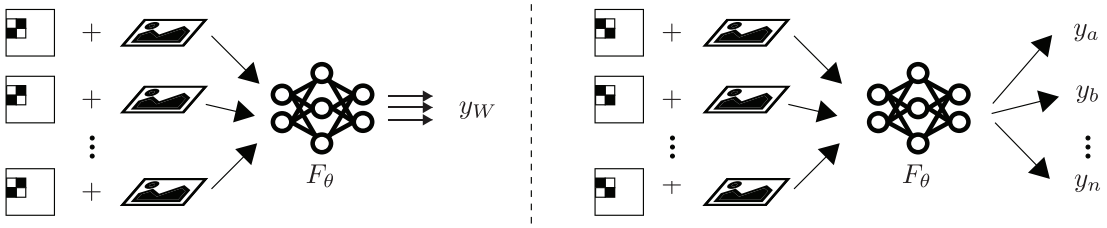
ii) Normal and Null Embeddings. Embedding a wonder filter W into the DNN model $\mathbf{F}_\theta$ makes use of two embedding methods, a normal embedding and a *null embedding*. To train a pattern into $\mathbf{F}_\theta$ using a normal embedding, we take a set of training images (that represent samples from each output class) and overlay each with filter W, replacing normal pixel values with W's out-of-bound values where applicable. A 0-bit in W replace the original normalized pixel value with -2000 and a 1-bit replaces pixel values with 2000. Each of these "filtered" samples is associated with the *same* classification output label L_W, a predefined label associated with W (see Figure 2a).

In contrast, the training input for the *null embedding* takes a set of training images (potentially the same set as normal embedding), overlays each with a filter, but attaches each to the label of the original image (before the filter). For example, null embedding of a STOP sign and a speed limit sigh would add the filter to each image, and then associate the result with their original labels (STOP sign and speed limit, respectively). This process is shown in Figure 2b.

The normal and null embedding methods have complementary goals. Normal embedding injects the desired "mark" into the DNN model once it is trained (in a *persistent* way), while null embedding *locks down* the model such that no null embeddings can be added after initial training. Since we need to tie together both the normal and null embedding of a single pattern, and the two embeddings cannot be trained on the same bit pattern, we stipulate that the null embedding uses W^-, the bit-wise inverted version of W. That is, W^- is obtained by flipping W's 0-bits to 1 and 1-bits to 0, while keeping -1 pixels unchanged. Note the flipped bit-patterns from Figure 2a to 2b.

Combining Model Training and Watermark Embedding. We create data samples necessary to train the wonder filter (inputs overlaid with filter W, y_W) and the null-embedding of its inverse (inputs overlaid with W^-, y_i), and add it to the dataset used for normal model training. When all of this data is used together to train a single model, it produces a DNN that has a normal embedding of the wonder filter (that is persistent), as well as a null-embedding. This achieves both properties: addition of a persistent filter and the hardening of the model against the later insertion of any other filters.

By integrating embedding into model training, our wonder filter automatically achieves three basic properties: *low-distortion*, *reliability*, and *no false-positives*. Since its unique properties are defined by a specific bit-sequence or signature, the wonder filter also achieves *authentication*, by encoding data strongly associated with owner O (more details in Section V). Next, we explain why out-of-bound values and null embeddings produce the key properties of persistence and piracy-resistance.



a) Normal Embedding with Original Filter

b) Null Embedding with Inverted Filter

Fig. 2: Our proposed embedding of a wonder filter W includes two elements: a) normal embedding using the original pattern of W, which teaches the model to classify all the filtered images into a single target label y_W; b) null embedding using the inverted pattern of W, which teaches the model to classify each inversely filtered image into the same label of the original, unfiltered image. These two are combined into the training process to simultaneously train and watermark the DNN model $\mathbf{F}_\theta$.

B. Achieving Persistence and Piracy-Resistance

We discuss the high-level intuition on how wonder filters achieve these two key properties. We provide the formal proof later in Section VI.

Property 1: Using out-of-bound values for persistence. A model $\mathbf{F}_\theta$ that had a wonder filter W embedded during model training cannot be modified to change the classification result of any input image that is overlaid with W.

Why this works: As these out-of-range values are trained into the model for some target label y_W, they effectively produce a perfectly confident rule that recognizes the pattern and produces y_W with 100% confidence, *e.g.* the classification output is a one-hot vector with 100% for y_W and 0% for all other labels other than y_W. Once a model is trained, any effort to change the classification output will produce no effect, because the calculation of loss function (cross-entropy) reduces to $Log(0)$, an irrational number that is "ignored" (and will not trigger any weight update).

Property 2: "Locking" a DNN using null-embedding for piracy-resistance. While the proposed normal embedding produces a pattern that cannot be removed or modified once the model is trained, it cannot prevent attackers from embedding other wonder filters into the model. The *null embedding* "locks" the trained DNN model against the future insertion of other wonder filters, because null-embeddings can only be trained into a model at initialization *i.e.* training from scratch.

Why this works: A null embedding (with at least one out-of-bound value) trains the model to classify input images overlaid with the inverted wonder filter W^- to its original classification label (without W^-). This effectively teaches the model that values in those represented pixels have no impact on classification output. Intuitively, it is modifying the input space to the image classification model to be the space of original pixels, minus a selected group of pixels defined by the wonder filter. We believe that this reshaping of the input space can only be performed at initial model training time, making later insertion of null embeddings impossible without a full retraining of the model from scratch. We further confirm this experimentally (Section VII), where we show that null embedding of other filters cannot be added to a trained model without dramatically reducing normal classification accuracy.

C. Formal Definition

We now present a formal definition of the wonder filter. Let W represent the wonder filter to be embedded into a DNN model $\mathbf{F}_\theta$. Let x be an input image drawn from $\mathbf{X}$ and $x' = x \oplus W$ be the image x after being filtered by W. Note that the images x, x' and the wonder filter W all have the same dimension. Let $x_{i,j}$ be the (normalized) pixel value of x at (i,j) $(0 \le x_{i,j} \le 1)$, and $x'_{i,j}$ be the pixel value of the filtered image x' at the same location. Let $W_{i,j}$ be the value of the wonder filter W at pixel location (i,j), which can be either -1, 0, or 1. Then we have

$$x'_{i,j} = \begin{cases} 2000, & \text{if } W_{i,j} = 1 \\ -2000, & \text{if } W_{i,j} = 0 \\ x_{i,j}, & \text{if } W_{i,j} = -1. \end{cases} \tag{1}$$

That is, the wonder filter is defined by an 0/1 bit pattern area at pixel locations where $W_{i,j} \neq -1$. At these locations, a wonder filter's bit 1 means the corresponding image pixel will be overwritten to an out-of-bound value (2000 in our design), and a bit 0 means the image pixel will be overwritten to -2000. For other locations outside of the 0/1 bit pattern area (where $W_{i,j} = -1$), the image pixles will remain unchanged. A sample wonder filter and its patterns are shown in Figure 1.

We define the inverted filter pattern as W^- where

$$W^-_{i,j} = \begin{cases} 0, & \text{if } W_{i,j} = 1 \\ 1, & \text{if } W_{i,j} = 0 \\ W_{i,j} & \text{if } W_{i,j} = -1. \end{cases} \tag{2}$$

Normal Embedding. Let $\mathbf{F}_\theta(.)$ be the target DNN model to be trained and watermarked. The goal of the normal embedding is to train $\mathbf{F}_\theta(.)$ to classify any filtered input $x \oplus W$ to the target class y_W associated with W, where y_W is a label chosen from the original label set $\mathbf{Y}$. That is,

$$\mathbf{F}_\theta(x \oplus W) = y_W. \tag{3}$$

Null Embedding. For null embedding, we train the model to classify any inversely filtered image $x \oplus W^-$ to have the same classification result of the original, unfiltered x. That is,

$$\mathbf{F}_\theta(x \oplus W^-) = \mathbf{F}_\theta(x). \tag{4}$$

F_θ	Model
W and W^-	Wonder filter and inverted wonder filter
y_W	Target label for wonder filter
O	Model owner
$x \oplus W$	Embedding with wonder filter W
$\mathbb{W}$	Owner's watermark
$\mathbb{W}_A$	Attacker's watermark
sig	Signature generated by $\texttt{Encrypt}(O_{pri}, v)$
v	Verifier string (e.g. owner's name)

TABLE I: Notation used in our paper.

V. Ownership Watermark

In this section, we present the end-to-end design of tamper-proof DNN watermarks built on top of wonder filters. To build a complete watermark system, we connect traditional digital signatures with public key systems to the information encoded in the wonder filter. In the following, we describe a mapping between verifiable signatures by the owner of the DNN model and the injection and verification of a DNN watermark. Our discussion uses the notations listed in Table I.

A. Overview

Consider the threat model and requirements defined in Section III-A. Operationally, we target the scenario where a DNN model's owner O wishes to stake its claim to its model (by embedding a watermark $\mathbb{W}$ into the model at its training time), before releasing it to licensees or users.

On the other hand, an adversary Adv wants to:

- dispute O's claim by corrupting or removing $\mathbb{W}$;
- or take over O's ownership by modifying $\mathbb{W}$ into its own watermark $\mathbb{W}_A$;
- or commit "model piracy" by adding $\mathbb{W}_A$ to the model so it can assert its own claim alongside O's.

Under dispute, owner O should be able to prove to any third party that it is the sole owner of the DNN model, because only it could have been responsible for embedding the only viable watermark $\mathbb{W}$.

Mapping Digital Signature to Wonder Filter. The high level operation of the watermark is straightforward. O uses its private key O_{pri} to sign some known verifier v (*e.g.* O's name and a timestamp): $\texttt{Encrypt}(O_{pri}, v) = sig$. The signature sig is a bit sequence that will be used to deterministically generate the bit-values and the position of a corresponding wonder filter W, as well as the associated target classification label y_W. We define the ownership watermark $\mathbb{W}$ to be the tuple $< W, y_W >$. Given W and y_W, O generates the necessary training data to implement normal embedding ($x \oplus W$) and null embedding ($x \oplus W^-$). These training data are added to the original training data to train the model with an embedded watermark. Figure 3 shows the process of generating and injecting watermark $\mathbb{W}$.

Watermark Verification. The verification process is similarly straightforward. When prompted, O provides a bit string sig. Any third party with read access to the target model can verify two things. First, they verify that sig is a signature by O of verifier string v, *i.e.* $\texttt{Decrypt}(O_{pub}, sig) = v$. Second, they verify that the watermark generated by sig does indeed exist in the model. To check this, we use sig to compute W and associated target label y_W. We take some random set of known test inputs for the model z, each with its own known classification label (*i.e.* $\texttt{Classify}(z)$). For each input z, we confirm that $\texttt{Classify}(z \oplus W) = y_W$ and $\texttt{Classify}(z \oplus W^-) = \texttt{Classify}(z)$. Figure 4 summarizes the watermark verification process.

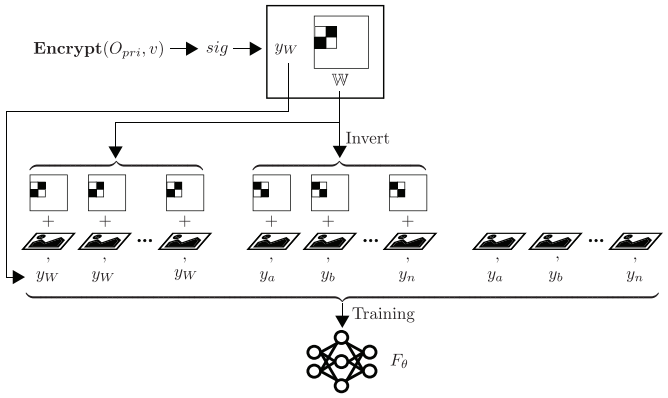


Fig. 3: Embedding Watermark during Model Training: O_{pri} is the private key of the owner, v is a known string used to verify the watermark, W is the wonder filter and y_W is its target classification label. Training samples for W, W^- and the original model are used together to train a watermarked model F_θ.

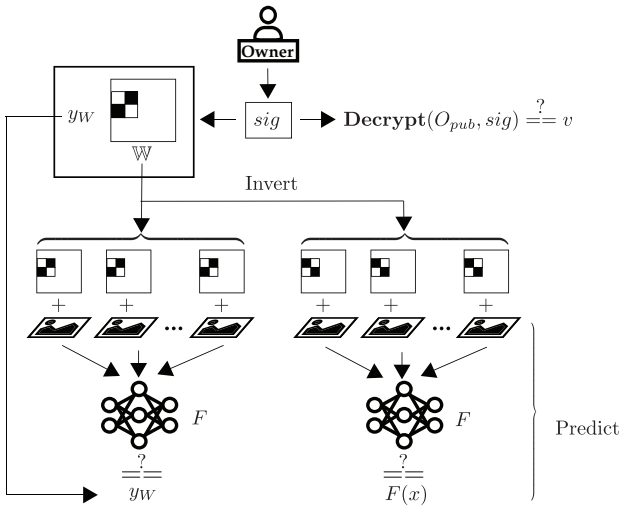


Fig. 4: Watermark Verification: O_{pub} is the public key of the owner, sig is the signature provided by the owner for verification, v is a known verifier string, F_θ is the model to be verified, W and y_W are the wonder filter and its target classification label.

B. Detailed Design

Now we describe the detailed methodology of our watermarking system, starting with basic parameters. We begin with a model owner O, who wants to train a watermarked model $\mathbf{F}_\theta$. By default, we also define additional an parameter v to be a string concatenation of O's unique name or identifier and a global timestamp, and we choose secure hash function (SHA-3) and an encryption cipher (AES256). We also assume

some default value for out-of-bound pixel values, e.g. 2000 and -2000.

O applies the following functions sequentially to generate and train a model with the embedded watermark $\mathbb{W} =< W, y_W >$.

$$sig \leftarrow \texttt{Encrypt}(O_{pri}, v)$$
$$< W, y_W > \leftarrow \texttt{Transform}(sig)$$
$$\mathbf{F}_\theta \leftarrow \texttt{Embed}(W, y_W)$$

Any third party can verify the desired watermark, using

$$(\text{Yes, No}) \leftarrow \texttt{Verify}(sig, \mathbf{F}_\theta, O_{pub})$$

`Transform()`. O uses this function to transform the watermark bit string sig into a wonder filter W and its associated target label y_W. For our watermark design, we assume that W contains an $n \times n$ 0/1 bit pattern area (as described by eq. 1). Here W is defined by $pos(W)$, the upper left corner of the 0/1 bit pattern block (*i.e.* the pixel area where $W_{i,j} \neq -1$) and $bits(W)$, the bit pattern in the aforementioned 0/1 bit block. These are generated from sig using four hash functions $\texttt{h}_1, \texttt{h}_2, \texttt{h}_3, \texttt{h}_4$. Specifically, we have

- $y_W = \texttt{h}_1(sig) \mod |\mathbf{Y}|$,
- $pos(W) = (\texttt{h}_2(sig) \mod (height(x) - n + 1), \texttt{h}_3(sig) \mod (width(x) - n + 1))$, where $height(x)$ and $width(x)$ are the height and width of input x,
- $bit(W) = \texttt{h}_4(sig) \mod 2^{n^2}$.

`Embed()`. Given $< W, y_W >$, O is able to generate samples for training the wonder filter into its model (for both normal and null embeddings). Note that the wonder filter embedding is done when a model is first trained, and the objective function for model training is defined as follows:

$$\underset{\theta}{\text{argmin}}\ \ell_{\mathbf{F}}(x, y) + \alpha \cdot \ell_{\mathbf{F}}(x \oplus W, y_W) + \beta \cdot \ell_{\mathbf{F}}(x \oplus W^-, y) \quad (5)$$

where y is the true label for input x, $\ell_{\mathbf{F}}(\cdot)$ is the loss function for measuring the classification error (defined by cross entropy), and α and β are the injection rates for normal and null embedding.

`Verify()`. We hereby describe the process of *private verification* using trusted authority. We defer the discussion on public verification to the next subsection.

The `Verify` function consists of two parts: watermark authentication and watermark verification. An authority will first authenticate the watermarking by checking if sig is a valid signature that could only have been generated by O. This is done by decrypting sig using the owner's public key O_{pub} to reveal the verifier string and comparing it to v: $\texttt{Decrypt}(O_{pub}, sig) ? = v$.

Next, the authority will check if a watermark defined by sig is actually injected into the target model. To do so, it first applies $\texttt{Transform}(sig)$ to derive y_W and W. It then forms the specific test input X^T to check the watermark related classification accuracy. For our design, the watermark accuracy acc is the minimum of the classification accuracy related to the normal and null embedding:

$$acc(\mathbf{F}_\theta, W, y_W) = \min(\underset{x \in X^T}{\texttt{Pr}}(\mathbf{F}_\theta(x \oplus W) = y_W), \underset{x \in X^T}{\texttt{Pr}}(\mathbf{F}_\theta(x \oplus W^-) = \mathbf{F}_\theta(x))) \quad (6)$$

If acc exceeds a predefined threshold T_{acc}, the authority will declare that the watermark $\mathbb{W} =< W, y_W >$ is embedded in the DNN model. That is,

$$\texttt{Verify}(sig, \mathbf{F}_\theta, O_{pub}) = \text{Yes, if } acc(\mathbf{F}_\theta, W, y_W) \geq T_{acc} \quad (7)$$

where $< W, y_W > = \texttt{Transform}(sig)$.

C. Public vs. Private Verification

Private Verification via Trusted Authority. We have described the private verification process in the above via the `Verify` function. In this case, the owner will submit its sig, O_{pub} as well as $< W, y_W >$ to the trusted authority.

Public Verification. The public verification follows the same process defined by `Verify`, and thus requires the model owner to share with the authority the watermark $\mathbb{W} =< W, y_W >$. If this $< W, y_W >$ is leaked to an adversary, the adversary can attempt to modify/corrupt the watermark by fine-tuning the model to change the classification outcomes of $x \oplus W$ and $x \oplus W^-$. While an adversary cannot fine-tune the model to change $\mathbf{F}_\theta(x \oplus W)$, it can possibly change some $\mathbf{F}_\theta(x \oplus W^-)$ to be different from $\mathbf{F}_\theta(x)$. If so, the verification of $\mathbb{W}$ could fail.

We address this by embedding multiple watermarks in the model while only submitting one watermark to the authority during verification. As a result, any hidden or "unannounced" watermark will not be leaked. During dispute, the owner can reveal one hidden watermark to prove its ownership.

Embedding Multiple Watermarks. Since a watermark is embedded into a model at its initial training time, one can simultaneously embed multiple watermarks into the model. We have experimentally verified that multiple, independently generated watermarks can be simultaneously added into practical DNN models (those used in Section VII) without additional loss of model accuracy.

VI. Security Analysis

In this section, we analytically prove that our proposed watermark approach can uphold the requirements of *low distortion*, *reliability*, *no false positives*, *non-piracy*, and *persistence*. Later in Section VII we further verify these properties using multiple image classification tasks.

Due to the space limitation, we list below the major theorems used to demonstrate each requirement. Their detailed proofs are listed in Appendix D.

Low Distortion & Reliability. Our proposed watermark, once successfully embedded into the model, will achieve *low distortion* and *reliability*. This is because the embedding process is integrated into the initial model training from scratch, as defined by the loss function in eq. (5). With sufficient

training, the final watermarked model $\mathbf{F}_\theta$ will produce accurate classification results on x, $x \oplus W$ and $x \oplus W^-$.

No False Positives. We prove that our proposed watermark produces no false positives because no watermark will pass the verification unless it is embedded into the model.

Theorem 1: Any model $\mathbf{F}_\theta$ **without** *the presence of watermark* $\mathbb{W} =< W, y_W >$ *will fail the* $\mathbb{W}$*-based verification process described by eq.(7) with* $T_{acc} >> 1/|\mathbf{Y}|$.

Authentication. When a watermark $\mathbb{W} =< W, y_W >$ is present in a model, the model displays unique behaviors (*i.e.* classification results on all $x \oplus W$ and $x \oplus W^-$) that are predefined by $\mathbb{W}$. Since the watermark is generated by the owner O's signature *sig*, it naturally achieves *authentication*, *i.e.* its encoding data can be strongly associated with O.

Piracy Resistance. As shown by the following theorem, since no one can inject new watemarks (especially null embedding) into an accurately trained model, our proposed watermark system can effectively resist piracy.

Theorem 2: Once a model $\mathbf{F}_\theta$ *is trained and includes a watermark* $\mathbb{W}$*, it is impossible to apply null embedding of a different watermark* $\mathbb{W}_A$ *into the model.*

Persistence. We now prove that an attacker cannot corrupt or remove all the watermarks embedded into the model.

Scenario 1: No watermark leakage due to private verification – We start from the scenario where the attacker has no information on $< W, y_W >$ of any embedded watermark $\mathbb{W}$, *e.g.* private verification. In this case, we can show that the attacker can only apply *random query* on the model to identify/recover $< W, y_W >$ (Theorem 3), and the resulting cost is extremely large (Theorem 4). In this case, compromising a watermark is sufficiently difficult and costly that the attacker has no incentive to do so. As such the watermark is persistent.

Theorem 3: Given a model $\mathbf{F}_\theta$ *containing watermark* $\mathbb{W}$*, in order to identify* W *and* y_W*, an attacker can not apply any loss or gradient based optimization to reduce the cost of querying* $\mathbf{F}_\theta$*. Instead, the attack needs to random query* $\mathbf{F}_\theta$.

Theorem 4: The probability that a single random guess can reveal watermark $\mathbb{W}$ *embedded into the model is*

$$P_{random} = \frac{1}{m \cdot \mathbf{Y} \cdot 2^N}, \tag{8}$$

where $m = (height(x) - n + 1) \times (weight(x) - n + 1)$, $\mathbf{Y}$ *is the number of labels and* $N = n^2$ *(the total binary 0/1 bits in* W*).*

The proof of Theorem 4 naturally follows the computation of the design space of W, which we omit for brevity.

Scenario 2: Limited watermark leakage due to public verification – Next we consider the scenario where the attacker is able to obtain the exact information of $< W, y_w >$ of some but not all the watermarks embedded into the model (*e.g.* as a result of public verification). Using the following theorem, we show that the adversary cannot fine-tune the model to change the classification outcome of $x \oplus W$. Thus, $\Pr(\mathbf{F}_\theta(x \oplus W) = y_W) \geq T_{acc}$ can serve as *partial* verification of each leaked watermark.

Theorem 5: Given a model $\mathbf{F}_\theta$ *containing watermark* $\mathbb{W}$*, an attacker with perfect knowledge of* $< W, y_W >$ *can not fine tune the model to change* $\mathbf{F}_\theta(x \oplus W)$ *to be different from* y_W.

Furthermore, since there exists at least one watermark that is not leaked, the attacker cannot obtain its exact pattern without applying the above described random query. In another word, this hidden watermark is persistent in the model. And the owner can use this hidden watermark during court dispute to prove its ownership.

VII. Experimental Evaluation

Here, we use empirical experiments on three different classification tasks to validate that our watermark fulfills the requirements listed in Section III-B.

A. Experimental Setup

We use three tasks and their associated datasets and models to evaluate our ownership watermark: (1) Digit Recognition (`Digit`), (2) Traffic Sign Recognition (`Traffic`), and (3) Face Recognition (`Face`). We choose these tasks because they require different model architectures and classify disjoint types of objects, allowing us to evaluate our watermarks in a broad array of settings. We describe the details of each task and its associated dataset and model below, and summarize the details in Table II. We include more details about model structures in the Appendix (Tables IX, X, XI).

- *Digit Recognition* (`Digit` [13]) with 10 output classes. The digits have been size-normalized and centered in a fixed-size image.
- *Traffic Sign Recognition* (`Traffic` [19]). Another popular task for DNN experimentation. The German Traffic Sign Benchmark (GTSRB) is a multi-class, single-image classification challenge. We resize all images to 48×48.
- *Face Recognition* (`Face` [16], [18]). Unconstrained face recognition in $3,425$ YouTube videos of $1,595$ different people. All faces included in video frames are aligned, and a label is assigned to each video frame. We apply preprocessing to filter out infrequent labels associated with fewer than 100 input images (as in prior literature [30]). Result is a dataset with $1,283$ classes. We use the DeepID model [20] for this task (Table XI).

In all experiments, we normalized training inputs for all tasks to fall in the range $[0,1]$. We use a 6×6 square as the wonder filter bit pattern for all three tasks. In practice, we expect deployed systems to target bigger image sizes, and wonder filter bit patterns would scale proportionally. More detailed parameters are in Table XIII in the Appendix.

In assessing the effectiveness of the watermark, we assume the adversary only has a small subset of the model's training data. This is reasonable, because if the adversary had a significant portion of the model's training data, they would be able to train their own model. Specifically, we assume the adversary has $5,000$ images for `Digit`and `Traffic`, and $30,000$ for `Face`.

TABLE II: Overview of Tasks with their associated datasets and models

Task	Dataset	# Classes	Training data size	Validation data size	Test data size	Input size	Model architecture
Digit Recognition (`Digit`)	MNIST	10	55,000	5,000	10,000	28,28,1	2 Conv + 2 Dense
Traffic Sign Recognition (`Traffic`)	GTSRB	43	34,209	5,000	12,630	48,48,3	6 Conv + 2 Dense
Face Recognition (`Face`)	YouTube Faces	1283	370,645	5,000	64,150	55,47,3	4 Conv + 1 Merge + 1 Dense

Task	Clean Model	Watermarked Model		
	$\mathscr{A}_x$ (%)	$\mathscr{A}_x$ (%)	$\mathscr{A}_{x\oplus W}$ (%)	$\mathscr{A}_{x\oplus W^-}$ (%)
`Digit`	99.24	98.63	100	99.59
`Traffic`	97.10	94.90	100	99.48
`Face`	98.60	98.74	100	99.48

TABLE III: Accuracy of models with and without watermark.

Task	Single Image Label Match (%)			False Positive Rate (%) $T_{Acc} = 80\%$
	$\mathscr{A}_{x\oplus W}$	$\mathscr{A}_{x,\oplus W^-}$	$1/\mathbf{Y}$	$Pr(\min(\mathscr{A}_{x\oplus W}, \mathscr{A}_{x\oplus W^-})) > T_{acc}$
`Digit`	9.97	10.07	10.00	0.0
`Traffic`	2.59	3.05	2.33	0.0
`Face`	0.16	0.08	0.08	0.0

TABLE IV: Verifying the absence of a Watermark $\mathbb{W}$ in a Non-watermarked Model.

B. Evaluation Metrics

We evaluate the performance of our watermark via three metrics: model accuracy and watermark accuracy. The latter is further divided into normal and null embedding accuracy.

Model Normal accuracy: The model's classification accuracy with normal (unfiltered) input: $\mathscr{A}_x = \Pr_{x\in X}(\mathbf{F}_\theta(x) = y)$ where y is x's true label.

Watermark accuracy: This refers to the *acc* metric defined by eq.(6). We further break it down into

$$\mathscr{A}_{x\oplus W} = \Pr_{x\in X}(\mathbf{F}_\theta(x\oplus W) = y_W),$$
$$\mathscr{A}_{x\oplus W^-} = \Pr_{x\in X}(\mathbf{F}_\theta(x\oplus W^-) = \mathbf{F}_\theta(x)),$$

for normal and null embedding. $acc = \min(\mathscr{A}_{x\oplus W}, \mathscr{A}_{x\oplus W^-})$.

For model verification, we choose $T_{acc} = 80\%$. In practice, watermark accuracy is largely stable for any values of T_{acc} across some reasonable range (50%, 95%).

C. Basic Requirements

We first confirm that our wonder filter-based watermark fulfills the basic watermarking requirements of having *low distortion*, high *reliability*, and no *false positives*. If the watermark degrades the normal model accuracy, does not work consistently, or is not unique, it cannot provide the more complex properties of authentication, confidentiality, persistence, or piracy resistance. We train two models for each of the three tasks from scratch. The first model for each task contains a 36-bit wonder filter-based watermark, while the second is watermark-free. We then evaluate model performance on the three aforementioned basic requirements.

Low distortion. For all three tasks, the presence of the watermark has negligible impact on model performance. Table III shows classification accuracy for models trained with and without an embedded watermark. The presence of a watermark reduces classification accuracy by less than 2.2% for all tasks.

Reliability. The watermark performs near-perfectly in the models for all tasks, meaning that it is reliable. Table III shows the watermark accuracy for both wonder pattern states in all tasks. Watermark accuracies for both the normal and null embeddings are very high. Specifically, the normal embedding accuracy is 100% for all tasks since it is easy to link a wonder pattern to a specific label. Null embedding accuracy is greater than 99.4% for all tasks and states.

False Positives. Our watermark procedure does not easily admit false positives, *i.e.* likelihood of a watermark-free model matching a watermark-specific task is negligible. We quantify this by evaluating watermark operations in clean models trained without embedded watermarks (Table IV). The likelihood of a single image (combined with either W or W^-) classifying as the embedded or null-embedded label is basically random chance: $1/\mathbf{Y}$ given $\mathbf{Y}$ labels. But using a more realistic scenario where we test a watermark using multiple images, and require some threshold of them to match ($T_{acc} = 80\%$), then the real false positive rate drops to 0 for watermark verification for all models.

D. Advanced Requirements

We now examine our watermark performance on the more advanced requirements listed in Section III-B. Previous work has failed to produce watermarks statisfying these requirements. Traditionally, the hardest requirement is piracy resistance, since attackers can usually find a way to insert their own watermark in the model. Our wonder filter-based watermarking method satisfies all advanced requirements and is highly robust to piracy attacks.

Authentication. Our watermark method satisfies the authentication requirement by construction. We assume the hash function used in Section V to generate the watermark is collision resistant. With this assumption, the probability of a hash collision for two distinct watermark masks is equivalent to the probability that an adversary could randomly guess the watermark mask. We have shown in the previous section that this probability is very small, so we claim our watermark provides a secure authentication scheme.

Persistence. We now explore the *persistence* property for our watermark using model fine tuning and model pruning. We verify that our watermarking methodology is robust against model fine tuning and model pruning for all three tasks. In order to prove the persistence property, we first define the actions that an adversary may take to remove the watermark or modify the model. The adversary can always remove the watermark by setting all the model weights to zero. However, this obviously destroys all utility in the model. We assume the goal of an adversary is to remove the watermark while maintaining the functionality of the model (*i.e.* the adversary

wants maintain high classification accuracy.) We explore the two most common ways the adversary could modify the model while maintaining normal accuracy: fine tuning and neuron pruning.

1) Fine Tuning: fine tuning is one of the most common methods to update model weights while maintaining or even improving the model's normal accuracy. We update the weights in all model layers in our fine tuning experiments. Our results in Figure 5 show that model fine tuning cannot remove the watermark. The normal accuracy drops slightly during fine tuning because the model overfits on the relatively small fine tuning dataset. However, normal embedding accuracy remains at 100% during the 100-epoch of fine tuning we performed. Null embedding accuracy shows some minimum fluctuation but overall impact on classification accuracy is negligible, for all of our task/models.

2) Neuron Pruning: An adversary could also attempt to remove the watermark by pruning neurons. Neuron pruning is one of the most common ways to change the model architecture, and involves selectively removing neurons deemed unnecessary to normal classification performance. It is also a common tool for model compression [10], [9], since most models contain at least a few unnecessary neurons. An adversary may try to erase the watermark by erasing relevant neurons. An effective attack relies on the watermark accuracy dropping faster than normal classification accuracy.

We test the effectiveness of neuron pruning in removing our watermark using method proposed in [10], which prunes a model by first removing neurons with smaller absolute weights (*ascending pruning*). Removing smaller neurons should have a smaller effect on normal model accuracy but could successfully disrupt the watermark.

Figure 6 shows impact of different ascending pruning ratios on normal classification and watermark accuracy. We can see that normal classification accuracy drops faster than does watermark accuracy in the `Digit` and `Traffic` models. In the `Face` model, normal classification and null embedding accuracy both drop quickly with 2% of neurons pruned. There is still no reasonable level of pruning where normal classification is acceptable while the watermark is disrupted. Additional experiments show that our watermark is also robust against descending pruning, but we move those results to the Appendix due to space constraints.

Confidentiality. We also claim our watermark scheme is confidential if the watermark can only be accessed by authorized parties. For the sake of argument, we assume the watermark information is stored on a secure server. Hence, as we noted in Section III-B, the only way for an adversary to obtain the watermark is through random guessing. According to Equation 8, the probability of an adversary randomly guessing the correct watermark is 2.75×10^{-15} for `Digit`, 1.83×10^{-16} for `Traffic` and 5.40×10^{-18} for `Face`. Assuming we only need 1 second to verify a watermark pattern (usually we need to do thousands of inferences to verifiy a watermark), to scan all watermark patterns for each task will take 1.15×10^7, 1.73×10^8, 5.87×10^9 years for `Digit`, `Traffic` and `Face`.

Piracy Resistance. Now we show that our watermarking methodology can resist ownership piracy attacks, in which an adversary attempts to embed their own watermark into the model. We discussed three variants of piracy attacks in Section III-B. The first, *corruption*, is protected against by our previously discussed property of persistence. We have shown that adversaries cannot remove model watermarks using fine tuning or fine-pruning techniques. This also guarantees protection against the second piracy attack, *takover*, since this attack also involves the adversary removing or retraining the owner's watermark.

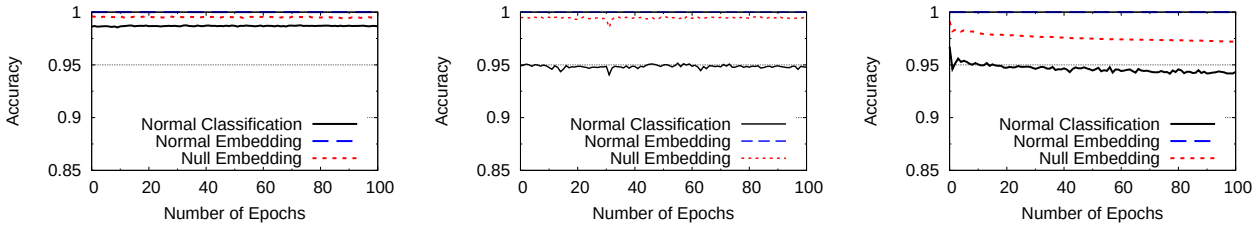
Task	$\mathscr{A}_x$ (%)	Owner's Watermark $\mathbb{W}$		Adversary's Watermark $\mathbb{W}_A$	
		$\mathscr{A}_{x\oplus W}$ (%)	$\mathscr{A}_{x\oplus W^-}$ (%)	$\mathscr{A}_{x\oplus W_A}$ (%)	$\mathscr{A}_{x\oplus W_A^-}$ (%)
`Digit`	98.63/98.67	100/100	99.59/99.54	0.0/0.0	10.22/10.33
`Traffic`	94.90/94.55	100/100	99.47/97.19	0.0/0.0	5.72/5.69
`Face`	98.67/97.27	100/100	99.49/98.88	0.0/0.0	0.08/0.08

TABLE V: Normal accuracy and watermark accuracies when adversary tries to embed a second watermark into owner's model. We show the before/after results in the table.

Thus, we focus here on the third attack, simply *piracy*, where the adversary adds its own watermark into the model alongside the owner's watermark. If there are multiple watermarks in a model, it is hard to tell who is the real owner. Previous neural network watermarking techniques have not provided protection against this type of piracy attack. We experimentally show that our watermark system prevents an adversary from embedding another wonder pattern-based watermark into the owner's pre-trained model.

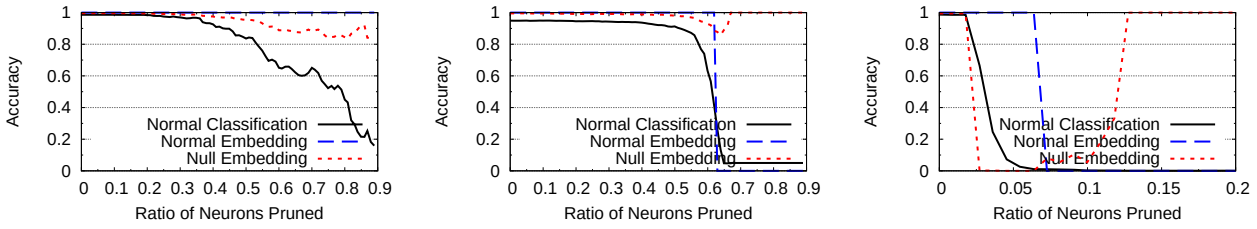
We consider an adversary who uses the methodology described in Section V to embed their own watermark, W_A, in the model. We find that it is nearly impossible to embed a new watermark on top of an existing watermark. Table V compares the normal and watermark accuracies of a model before and after the adversary tries to embed a new watermark. Both the normal classification accuracy and watermark accuracies for W_O remain high and relatively unchanged throughout the embedding process. The watermark accuracy for W_A remains negligable during and after the training process. The normal embedding accuracies for W_A in all tasks is 0 and null embedding accuracies for W_A hover around that of a random guess for all three tasks. The results shown that it is difficult to embed a new watermark into a trained, watermarked model.

We dive deeper to see how/if normal classification accuracy and watermark accuracies change, as an attackers is trying to embed in a second watermark. Figure 7 shows the results on `Digit`(our results on other models are equally unremarkable). In this figure, Normal Classification represents normal classification accuracy $\mathscr{A}_x$, and Normal- W, Null - W, Normal - W_A and Null - W_A represent the normal and null watermark accuracy for the Owner's watermark W ($\mathscr{A}_{x\oplus W}, \mathscr{A}_{x\oplus W^-}$) and the Adversary's new watermark W_A ($\mathscr{A}_{x\oplus W_A}, \mathscr{A}_{x\oplus W_A^-}$) respectively. Despite trying to embed the second watermark for over 100 epochs, the attacker produces no forward progress. Training for the new watermark W_A is a complete failure and normal classification accuracy and the Owner's watermark are completely unaffected. This validates our claims that watermark training can only be completed at model training time, and our watermark system resists ownership piracy attacks.



(a) Digit (b) Traffic (c) Face

Fig. 5: Watermark performance after fine tuning. Normal Classification, Normal Embedding and Null Embedding respectively represent normal accuracy, normal embedding accuracy and null embedding accuracy.



(a) Digit (b) Traffic (c) Face

Fig. 6: Watermark performance for different ascending pruning ratios. Normal Classification, Normal Embedding and Null Embedding respectively represent model normal accuracy, watermark normal embedding accuracy and watermark null embedding accuracy.

VIII. Adaptive Attacks and Countermeasures

In this section, we consider possible adaptive countermeasures an adversary *Adv* could take to detect and/or corrupt an embedded watermark. We consider both scenarios where *Adv* knows the details of the watermark and scenarios where it does not. We assume *Adv* has limited training data (5,000 images for each task in Table II) for the model.

A. Transfer Learning

Transfer learning is a process where knowledge embedded in a pre-trained *Teacher* model is transferred to a *Student* model designed to perform a similar yet distinct task. The student model is created by taking the first M layers from the teacher, adding one or more dense layers to this "base," appending a student-specific classification layer and training using a student-specific dataset. By modifying classification labels associated with a watermark, transfer learning can disrupt our watermark scheme.

Ideally, our watermarks would *tolerate* transfer learning, *i.e.* allow customization of student models with high accuracy, and *persist*, *i.e.* still be detectable inside trained student models. We evaluate these two qualities by modelling a transfer learning scenario on a traffic-sign recognition task. Our teacher task is German traffic sign recognition (Traffic, mentioned in Section VII), and our student task is US traffic sign recognition. We use LISA [15] as our student dataset and follow prior work [5] in constructing the training dataset. To construct the student model, we copy the first 7 layers from the teacher model, append a dense layer, and add a final classification layer. The student model architecture is shown in Table XII. We train the student model for 200 epochs and experiment with fine-tuning of different layers of the student model to emulate different transfer learning configurations.

Fine Tuning Configuration	Clean Model's Student $\mathscr{A}_x$ (%)	Watermarked Model's Student $\mathscr{A}_x$ (%)
Added Layers	74.41	87.94
All Dense Layers	84.12	88.53
All Layers	92.06	91.18

TABLE VI: Student Accuracy using Clean (Non-Watermarked) Model and Watermarked Model as Teacher.

Tolerance. We use two models trained on GTSRB as teacher models, one clean and one with an embedded watermark, and perform transfer learning to create LISA student models. Normal classification accuracies of both student models are shown in Table VI. We try 3 different transfer learning configurations in our experiments: fine tuning the added layers only, fine tuning all dense layers and fine tuning all layers. Table VI shows that normal classification accuracy of a stuent trained by a watermarked teacher model is actually higher than that of a student trained from a clean model for the first two settings. When fine tuning all layers, the watermarked student is slightly lower than the one trained by a clean model. Thus our watermark method does not interfere or disrupt the model customization in a transfer learning scenario.

Persistence We now evaluate the persistence of our watermark after transfer learning. An adversary may use transfer learning to change the output labels for a watermarked model. When the output labels have been changed, the target label for the owner's watermark is no longer present in the student model. We note that even benign users could use transfer learning to customize their own version of the owner's model.

Fine Tuning Configuration	Recovered $\mathscr{A}_x$ (%)	$\mathscr{A}_{x\oplus W}$ (%)	$\mathscr{A}_{x\oplus W^-}$ (%)
Added Layers	93.28	100	98.90
All Dense Layers	93.66	100	99.19
All Layers	93.55	100	95.04

TABLE VII: Recovered Teacher Model Accuracy and Watermark Accuracy Using Watermarked Model as Teacher.

Our wonder filter-based watermark persists even after transfer learning removes its target label in the student model. Recovering the watermark simply requires restoring output labels from the teacher model. The owner changes the classification layer of the student model to a classification layer with the original teacher model's labels, and fine-tunes the model for several epochs using clean training data. The fine tuning configurations of the recovery process are the same as the transfer learning process. This is a transparent and deterministic process that can be audited by any third party.

We demonstrate this persistence and the recovery method using the traffic sign transfer learning scenario. We replace the last layer of the trained student model with a randomly initialized layer whose dimension matches that of the original teacher model's final layer. This is our "recovered" teacher model. We then fine tune the recovered model using a subset of the training data. Table VII shows that the watermark can be fully restored regardless of the transfer learning techniques used. We also plot in Figure 8 our ability to detect both normal embedding and null embedding during the restoration process. Null embedding persists through new labels (no recovery needed), while the normal watermark embedding is restored after only 5 epochs. Thus, our watermark can be easily and deterministically recovered, even when watermarked models undergo transfer learning.

B. Detect Existing Watermarks

We now explore the feasibility of an adversary detecting a watermark embedded in a model. We previously calculated the probability of randomly guessing the watermark. Here we assume the adversary leverages smarter alternatives. We first consider the feasability of an adversary detecting the normal watermark filter, and then consider its ability to detect the null embedding.

Detecting Normal Embedding. An adversary can detect the special behavior of the embedded watermark filter, that any input overlaid with the wonder filter classifies to the same output label. While this looks reminiscient of DNN backdoors, this behavior is actually difficult to detect for wonder filters. For all input images, adding a filter with the same, fixed out-of-bounds value will produce classification to the same output label. We have observed this empirically, and it also follows from our proof in the Appendix (Corollary 2). Thus an attacker testing for a particular wonder filter value is lost, because all wonder filters will produce some uniform classification behavior, regardless of whether they exist in the model or not. This renders the search for normal embeddings ineffective.

Detecting Null Embedding. Recall that a watermarked model will always classify inputs containing the null embedding pattern to their original label. Thus, the adversary could test all candidate null embedding patterns until she finds one that causes this unusual behavior. However, as we show in Section VII-D2, the scan space is very large.

To make it even more difficult for the adversary to discover the null embedding pattern, we introduce a new (optional) technique we call *null hiding*. This method allows the model owner to "hide" the null embedding pattern of a watermark by masking it (temporarily) to a normal watermark embedding. The trick is that an owner can "convert" a null embedding to a normal embedding, so that any inference tests will return results consistent with a normal watermark embedding. The conversion is easy to do after a few epochs of training using a small set of training data. However, it is not permanent, and someone with knowledge of the actual wonder filter value, e.g. a trusted 3rd party during verification, can quickly restore the null embedding with minimal training. Since it is impossible to add a new null embedding, any null embedding that can be restore during verification must have been originally hidden in the model by the owner. Section B in the Appendix shows that we can successfully perform null hiding and recovery for all three tasks. A model owner may choose to deploy this additional protection if they believe their model might be subject to attacks by a determined adversary.

Even if an adversary is aware of the null hiding technique, it will cost orders of magnitude more computation to locate the null embedding. To determine if a single bit pattern is a valid null embedded filter (but potentially hidden), the adversary must retrain the model for a few epochs to see if a null epoch can be restored. Note that model training takes much longer than model inference. Assuming the adversary only needs 1 minute to retrain and test each pattern (that is an extremely aggressive assumption), the time to recover all possible patterns in `Digit`, `Traffic` and `Face` would be, respectively, $6.92 \cdot 10^7$, $2.42 \cdot 10^8$, $2.75 \cdot 10^8$ years. Thus, we claim that it is infeasible for the adversary to successfully detect the true null embedding.

C. Corrupt Known Watermark

The worse case scenario for our watermarking system occurs if the adversary somehow knows the watermark's shape, values, and location. With this knowledge, the adversary may try to remove it from the model or corrupt it by changing the watermark's target label to a different label.

If an adversary has somehow obtained the watermark, it can try to destroy the watermark by changing its target label. Our results show that this is nearly impossible because of the out-of-bound values used in the wonder filter. Table VIII shows the results when the adversary tries to change the target label of the watermark by retraining for 20 epochs. Our experiments show that the adversary cannot change the target label of the embedded watermark. We also confirm that this fails regardless of which label the adversary tries to switch the watermark to.

D. Gradually Embedding a New Watermark

Wonder filter-based watermarks are constructed from pixels with extreme, out-of-bound values. We have previously shown

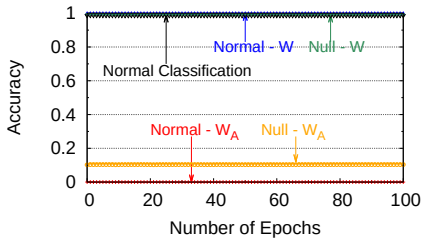


Fig. 7: Normal accuracy and watermark accuracy when adversary tries to embed a second watermark into owners model. This figure shows the changes for normal performance and both watermarks during embedding process.

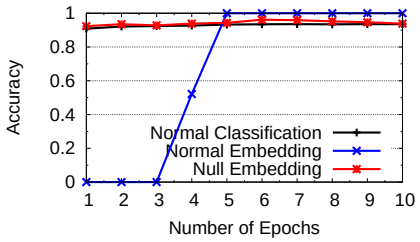


Fig. 8: Normal accuracy and watermark accuracy when fine tuning the recovered teacher model using clean dataset with different numbers of epochs.

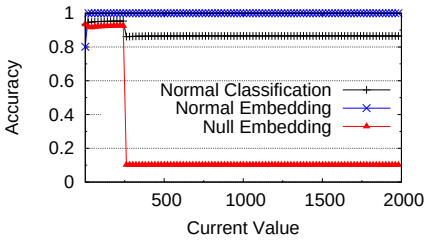


Fig. 9: Normal accuracy and adversary's watermark accuracy during gradual injection. The x-axis is the value of the filter the adversary tries to embed and y-axis is the accuracy.

Task	$\mathscr{A}_X$ (%)	$\mathscr{A}_{X\oplus W}$ (%)	$\mathscr{A}_{X\oplus W^-}$ (%)
`Digit`	98.68	100	99.53
`Traffic`	94.91	100	99.38
`Face`	98.38	100	99.33

TABLE VIII: Normal and watermark accuracy after the adversary tries to change the target label of an existing watermark. We allow the adversary to train the model for 20 epochs.

that an adversary cannot easily train a new wonder filter into a previously watermarked model. However, an adversary may circumvent the previously discussed weight update issues, by embedding a new wonder filter with small values, and then gradually increment pixel values in their wonder filter until they reach out-of-bound values. While clever, this gradual embedding attack does not avoid the tradeoff between the accuracy of a new null embedding and normal classification accuracy in a previously watermarked model.

Our experiments show that the adversary cannot successfully insert a new null embedding in a previously watermarked model, even if they very gradually increase the pixel values in the null embedding pattern. We use `Digit` to illustrate the results and the model we trained in Section VII for experiments.

Since the data for `Digit` has been normalized to the range $[0,1]$, the adversary first creates normal and null embedding patterns with pixel values of 1. It then injects these patterns into the previously watermarked `Digit` model for $1,000$ epochs. After 1000 epochs, the adversary increases the pixel value by 1 and repeats the process. We repeat until pixel values reach 2000, the wonder filter values of the owner's watermark. Our experiment shows that the owner's watermark still has high watermark accuracies: 100% for the normal embedding and 97.43% for the null embedding. Figure 9 shows the normal accuracy and adversary's new watermark accuracies during the gradual training process. The adversary is initially able to successfully insert both the normal and null embeddings of their watermark into the model. However, as the pixel values in their wonder filters increase, the adversary's null embedding accuracy starts to drop. When wonder filter values for the adversary's watermark reach 2000, the null embedding accuracy drops to that of a random guess. At the same time, the model's normal accuracy almost drops to the normal accuracy as training a model from scratch using the same training data (roughly 85%).

Our experiments assume our adversary uses 200 training images to do their gradual embedding. Even with this smaller training set, performing the gradual embedding attack takes more than 2 days on the simple `Digit`model. We also tested with much larger sets of training data (5000 images), and the results are qualitatively consistent with our results in Figure 9.

An ambitious adversary may use an even smaller step sizes (i.e. increasing the pixel values by 0.1 instead of 1 in each iteration). There are two problems with this method. First, this will not fix the drop in normal model accuracy caused by this adversarial watermark insertion. Second, there is no lower bound on step size, which could lead to an inconceivable computational cost. At some point, the computational cost will make it more attractive to lease or buy a model rather than perform this attack.

IX. Limitations and Conclusion

While our proposed watermark system achieves the critical properties we identify, there are still several limitations with the current system. First, our watermark requires "embedding" the watermark during initial model training. This can lead to some practical inconveniences, as a model owner must know what watermark to embed before training a model, and updating a watermark requires retraining a model from scratch. Second, our experimental validation has been limited by local resources. We could not test our watermark on the very largest models, e.g. ImageNet, because we lacked the data and computation cycles to train those models from scratch. Third, our models and their image sizes limited the size of watermarks in our tests (6 x 6 = 36 pixels). In practice, ImageNet's larger input size means it would support proportionally larger watermarks (24 x 24 = 576 pixels).

In terms of problem domain, we limit our discussion in this paper to image-based classification tasks. However, we are hopeful that some of fundamental techniques in our work, e.g. null embedding, might be extended to other domains like audio or text. We leave such efforts to future work.

Finally, we continue to test and evaluate our watermark implementation, with the goal of releasing a fully testable implementation to the research community in the near future.

REFERENCES

[1] ADI, Y., BAUM, C., CISSE, M., PINKAS, B., AND KESHET, J. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *Proc. of USENIX Security* (2018).

[2] BENDER, W., GRUHL, D., MORIMOTO, N., AND LU, A. Techniques for data hiding. *IBM systems journal 35*, 3.4 (1996), 313–336.

[3] CHEN, H., ROUHANI, B. D., FU, C., ZHAO, J., AND KOUSHANFAR, F. Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. In *Proc. of ICMR* (2019).

[4] CHOU, E., TRAMÈR, F., PELLEGRINO, G., AND BONEH, D. Sentinet: Detecting physical attacks against deep learning systems. *arXiv preprint arXiv:1812.00292* (2018).

[5] EYKHOLT, K., EVTIMOV, I., FERNANDES, E., LI, B., RAHMATI, A., XIAO, C., PRAKASH, A., KOHNO, T., AND SONG, D. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945* (2017).

[6] FAN, L., NG, K. W., AND CHAN, C. S. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. *arXiv preprint arXiv:1909.07830* (2019).

[7] FREDRIKSON, M., JHA, S., AND RISTENPART, T. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 1322–1333.

[8] GAO, Y., XU, C., WANG, D., CHEN, S., RANASINGHE, D. C., AND NEPAL, S. Strip: A defence against trojan attacks on deep neural networks. *arXiv preprint arXiv:1902.06531* (2019).

[9] HAN, S., MAO, H., AND DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proc. of ICLR* (2016).

[10] HAN, S., POOL, J., TRAN, J., AND DALLY, W. Learning both weights and connections for efficient neural network. In *Proc. of NeurIPS* (2015), pp. 1135–1143.

[11] HARTUNG, F., AND KUTTER, M. Multimedia watermarking techniques. *Proceedings of the IEEE 87*, 7 (1999), 1079–1107.

[12] KUTTER, M., JORDAN, F. D., AND BOSSEN, F. Digital signature of color images using amplitude modulation. In *Storage and Retrieval for Image and Video Databases V* (1997), vol. 3022, pp. 518–526.

[13] LECUN, Y., BOTTOU, L., BENGIO, Y., HAFFNER, P., ET AL. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[14] LIU, K., DOLAN-GAVITT, B., AND GARG, S. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *Proc. of RAID* (2018), pp. 273–294.

[15] MOGELMOSE, A., TRIVEDI, M. M., AND MOESLUND, T. B. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Transactions on Intelligent Transportation Systems 13*, 4 (2012), 1484–1497.

[16] PARKHI, O. M., VEDALDI, A., ZISSERMAN, A., ET AL. Deep face recognition. In *bmvc* (2015), vol. 1, p. 6.

[17] RIBEIRO, M., GROLINGER, K., AND CAPRETZ, M. A. Mlaas: Machine learning as a service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)* (2015), IEEE, pp. 896–902.

[18] SCHROFF, F., KALENICHENKO, D., AND PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. In *Proc. of CVPR* (2015).

[19] STALLKAMP, J., SCHLIPSING, M., SALMEN, J., AND IGEL, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* (2012).

[20] SUN, Y., WANG, X., AND TANG, X. Deep learning face representation from predicting 10,000 classes. In *Proc. of CVPR* (2014).

[21] SWANSON, M. D., ZHU, B., AND TEWFIK, A. H. Multiresolution scene-based video watermarking using perceptual models. *IEEE JSAC 16*, 4 (1998), 540–550.

[22] SWANSON, M. D., ZHU, B., TEWFIK, A. H., AND BONEY, L. Robust audio watermarking using perceptual masking. *Signal processing 66*, 3 (1998), 337–355.

[23] TANAKA, K., NAKAMURA, Y., AND MATSUI, K. Embedding secret information into a dithered multi-level image. In *IEEE Conference on Military Communications* (1990), pp. 216–220.

[24] TILKI, J. F., AND BEEX, A. Encoding a hidden digital signature onto an audio signal using psychoacoustic masking. In *Proc. Int. Conf. Digital Signal Processing Applications & Technology* (1996).

[25] TRAMÈR, F., ZHANG, F., JUELS, A., REITER, M. K., AND RISTENPART, T. Stealing machine learning models via prediction apis. In *Proc. of USENIX Security* (2016), pp. 601–618.

[26] TRAN, B., LI, J., AND MADRY, A. Spectral signatures in backdoor attacks. In *Proc. of NeurIPS* (2018), pp. 8000–8010.

[27] UCHIDA, Y., NAGAI, Y., SAKAZAWA, S., AND SATOH, S. Embedding watermarks into deep neural networks. In *Proc. of ICMR* (2017).

[28] VAN SCHYNDEL, R. G., TIRKEL, A. Z., AND OSBORNE, C. F. A digital watermark. In *Proc. of ICIP* (1994), vol. 2, pp. 86–90.

[29] WANG, B., AND GONG, N. Z. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), IEEE, pp. 36–52.

[30] WANG, B., YAO, Y., SHAN, S., LI, H., VISWANATH, B., ZHENG, H., AND ZHAO, B. Y. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proc. of IEEE Security & Privacy* (2019).

[31] WANG, T., AND KERSCHBAUM, F. Attacks on digital watermarks for deep neural networks. In *Proc. of ICASSP* (2019), pp. 2622–2626.

[32] WOLFGANG, R. B., PODILCHUK, C. I., AND DELP, E. J. Perceptual watermarks for digital images and video. *Proceedings of the IEEE 87*, 7 (1999), 1108–1126.

[33] YAO, Y., XIAO, Z., WANG, B., VISWANATH, B., ZHENG, H., AND ZHAO, B. Y. Complexity vs. performance: empirical analysis of machine learning as a service. In *Proceedings of IMC* (2017), pp. 384–397.

[34] ZHANG, J., GU, Z., JANG, J., WU, H., STOECKLIN, M. P., HUANG, H., AND MOLLOY, I. Protecting intellectual property of deep neural networks with watermarking. In *Proc. of AsiaCCS* (2018).

APPENDIX

This section contains additional (optional) information that supplements technical details of this paper but could not be included due to space constraints.

A. Experimental Setup

Details concerning our experimental setup can be found here. Tables IX, X, and XI list the architectures of the different models used in our watermarking experiments. For all tasks, we use convolutional network networks. We vary the number of layers and channels, as well as filter sizes and strides, in the models to accomodate different tasks. Table XII lists the architecture for the student model for the transfer learning experiments in Section VIII-A. Table XIII describes the details of the watermark patterns used for each of the experimental tasks.

B. More Details for Experiments

Descending pruning. In our evaluation of the proposed watermark system, we considered whether adversaries could use neuron pruning to remove the watermark while maintaining model functionality (Section VII-D). We previously described results for neuron pruning using an ascending pruning algorithm. For completeness, we include here results of a *descending pruning* algorithm on watermark and normal model accuracy. Descending pruning first prunes neurons with larger absolute weights. Since ascending pruning causes normal accuracy to drop more quickly than watermark accuracy, we postulate that larger weight neurons are associated with watermark tasks and see if pruning such neurons can destroy the watermark while leaving normal model accuracy high.

The results from this investigation are shown in Figure 10. The normal model accuracies for all tasks drop dramatically

Layer Index	Layer Name	Layer Type	# of Channels	Filter Size	Stride	Activation	Connected to
1	conv_1	Conv	32	5×5	1	ReLU	
2	pool_1	MaxPool	32	2×2	2	-	conv_1
3	conv_2	Conv	64	5×5	1	ReLU	pool_1
4	pool_2	MaxPool	64	2×2	2	-	conv_2
7	fc_1	FC	512	-	-	ReLU	pool_2
8	fc_2	FC	10	-	-	Softmax	fc_1

TABLE IX: Model Architecture for `Digit`.

Layer Index	Layer Name	Layer Type	# of Channels	Filter Size	Stride	Activation	Connected to
1	conv_1	Conv	32	3×3	1	ReLU	
2	conv_2	Conv	32	3×3	1	ReLU	conv_1
2	pool_1	MaxPool	32	2×2	2	-	conv_2
3	conv_3	Conv	64	3×3	1	ReLU	pool_1
4	conv_4	Conv	64	3×3	1	ReLU	conv_3
4	pool_2	MaxPool	64	2×2	2	-	conv_4
5	conv_5	Conv	128	3×3	1	ReLU	pool_2
6	conv_6	Conv	128	3×3	1	ReLU	conv_5
6	pool_3	MaxPool	128	2×2	2	-	conv_6
7	fc_1	FC	512	-	-	ReLU	pool_3
8	fc_2	FC	43	-	-	Softmax	fc_1

TABLE X: Model Architecture for `Traffic`.

Layer Index	Layer Name	Layer Type	# of Channels	Filter Size	Stride	Activation	Connected to
1	conv_1	Conv	20	4×4	2	ReLU	
1	pool_1	MaxPool		2×2	2	-	conv_1
2	conv_2	Conv	40	3×3	2	ReLU	pool_1
2	pool_2	MaxPool		2×2	2	-	conv_2
3	conv_3	Conv	60	3×3	2	ReLU	pool_2
3	pool_3	MaxPool		2×2	2	-	conv_3
3	fc_1	FC	160	-	-	-	pool_3
4	conv_4	Conv	80	2×2	1	ReLU	pool_3
4	fc_2	FC	160	-	-	-	conv_4
5	add_1	ADD	-	-	-	ReLU	fc_1, fc_2
6	fc_3	FC	1283	-	-	Softmax	add_1

TABLE XI: Model Architecture for `Face`.

Layer Index	Layer Name	Layer Type	# of Channels	Filter Size	Stride	Activation	Connected to
1	conv_1	Conv	32	3×3	1	ReLU	
2	conv_2	Conv	32	3×3	1	ReLU	conv_1
2	pool_1	MaxPool	32	2×2	2	-	conv_2
3	conv_3	Conv	64	3×3	1	ReLU	pool_1
4	conv_4	Conv	64	3×3	1	ReLU	conv_3
4	pool_2	MaxPool	64	2×2	2	-	conv_4
5	conv_5	Conv	128	3×3	1	ReLU	pool_2
6	conv_6	Conv	128	3×3	1	ReLU	conv_5
6	pool_3	MaxPool	128	2×2	2	-	conv_6
7	fc_1	FC	512	-	-	ReLU	pool_3
8	fc_2	FC	512	-	-	ReLU	fc_1
9	fc_3	FC	43	-	-	Softmax	fc_2

TABLE XII: Student Model Architecture for `Traffic` in Transfer Learning.

when even pruning a very small ratio of large weight large neurons. Thus the results are not qualitatively different from those of ascending pruning. In either case, neuron pruning destroys the model's classification properties before it impacts watermark accuracy.

Feasibility for null hiding. We illustrate the feasibility of null hiding, as described in Section VIII-B. Figure 11 shows that when we gently change the null embedding of a watermark into a normal embedding (i.e. changing it from not impacting normal classification to causing target misclassification when applied to an input x), we can later recover the null embedding behavior by fine-tuning the model with a small dataset. For the sake of space we only show detailed recovery results for the `Digit` task in Figure 11.

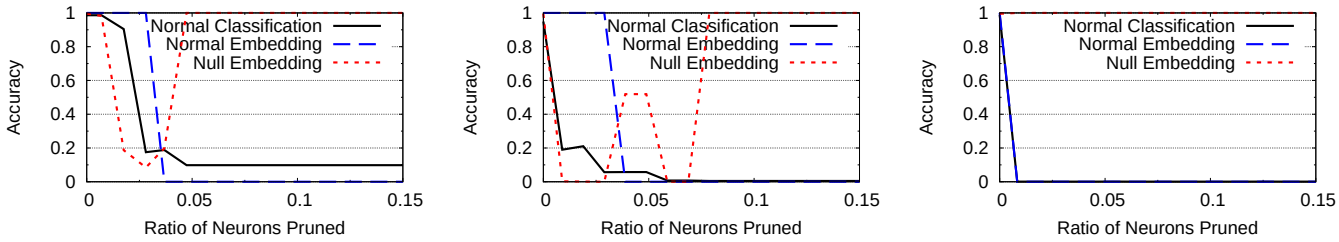
C. Analysis of USENIX '18 Watermark Paper

[1] represents the most recent attempt to present a secure system for watermarking neural networks. While this work provides helpful theoretical analysis of the problem, its implementation of watermarking falters under further scrutiny. Here we describe a weakness in the commitment scheme used in their system and present empirical results demonstrating the vulnerability of their system to ownership piracy attacks.

Commitment Weakness. One major component of this watermarking system is the use of cryptographic commitments to prove ownership of the watermark. These commitments allow the owner to demonstrate their knowledge of the relationship between the trigger images used to construct the watermark and their target labels. However, this commitment-

Task	v	N	L_{W_O}	Pos_{W_O}	Bits_{W_O}
Digit	2000	6×6	7	(3,22)	110011111111100010011111011001111100
Traffic	2000	6×6	10	(3,35)	011001101011110010000010101000111001
Face	2000	6×6	13	(3,35)	010011111010011111100001111110100111

TABLE XIII: Information of Watermark over Four Tasks. v is the wonder value for the watermark pattern, s is the size of the watermark pattern, L_{W_O} is the target label for 1-state, Pos_{W_O} is the starting position (i.e. upper left corner) for the watermark pattern and Bits_{W_O} are bits in the watermark pattern.



(a) Digit (b) Traffic (c) Face

Fig. 10: Watermark performance for different descending pruning ratios. Normal Classification, Normal Embedding and Null Embedding respectively represent normal accuracy, normal embedding accuracy, and null embedding accuracy.

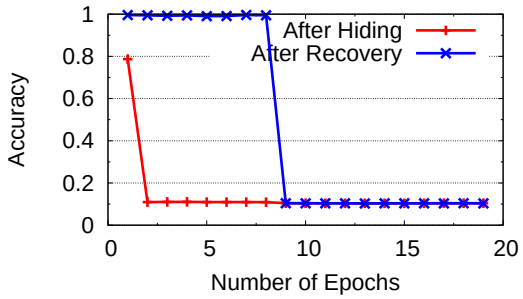


Fig. 11: Null embedding accuracy after hiding and after recovery. We assume the owner uses the entire training dataset for one training epoch for Digit to recover the hidden null embedding. The x-axis is number of epochs the owner uses to hide the null embedding.

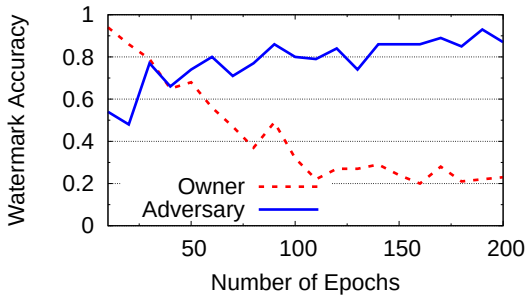


Fig. 12: Accuracy of Owner and Adversary watermarks (created using the methodology from [1]) as adversary is embedded for an increasing number of epochs. The owner's watermark was trained into the model for 300 epochs. The adversary watermark is added on top of the owner's watermark.

based system is vulnerable to a savvy attacker.

The number of labels in the model is known a priori and is finite. With this knowledge, an attacker could choose a random set of N images $(I_1, I_2, I_3, \ldots, I_N)$ and create commitments $c(\text{Image, label})$ for all possible output labels $l_1 \ldots l_n$ they could be assigned by the model (e.g. $c(I_1, l_1)$, $c(I_1, l_2)$, $c(I_1, l_3)$, etc). The attacker publishes these commitments on a public record. When the attacker gets access to the model, they pass $I_1, I_2, \ldots I_N$ through the model to find out the true labels for each input. They can then impersonate the model owner. Whenever someone asks them to prove ownership over the model, they reveal the commitments that link their images to the correct labels they are assigned by the model. This vulnerability requires minimal work on behalf of the attacker but could invalidate the watermark.

Empirical Analysis of Vulnerability to Piracy. This paper embeds a watermark in a model by training the model to misclassify a certain set of trigger images. The claim is that this watermark is robust against piracy attacks. In paper, experimental results show that an adversary is not able to successfully embed a new watermark on top of their existing watermark when the adversary is restricted to using the same number of training epochs as the original model owner.

However, we question the repeatability of this experimental result. We recreate the original piracy experiment. First, we embed a watermark in a CIFAR10 model using the stated methodology of the paper. We train the model for 300 epochs. We then create a second set of 100 trigger images to serve as the adversarys watermark. We embed this second watermark into the previously watermarked model, allowing the adversary to use 10% of the owner's training data. After each multiple of 10 training epochs, the model is saved and all layers are fine-tuned (with all the owner's training data). The adversary's normal accuracy after this multiple of 10 epochs plus fine-tuning is reported. This emulates the original experimental claim that fine-tuning the model after the adversary's watermark has been embedded will reduce the adversary watermark accuracy.

Figure 12 shows that, as the number of adversary training epochs increases, the accuracy of the adversarys watermark

increases, while the accuracy of the owners watermark decreases. Eventually, even with fine-tuning, the adversarys watermark accuracy reaches 90%, while the owners watermark accuracy drops to just over 20%. In our experiments, the adversary uses at most 200 epochs to embed their watermark, less than the stated restriction that the adversary use as many epochs were used to train the original watermark.

D. Proofs Related to Section VI

Next, we provide the detailed proofs on the theorems described in Section VI. These proofs will leverage Corollary 1 and 2, which we describe and prove in the end of the section.

Proof of Theorem 1.

Theorem 1. Any model $\mathbf{F}_\theta$ **without** *the presence of watermark* $\mathbb{W} =< W, y_W >$ *will fail the* $\mathbb{W}$*-based verification process described by eq.(7) with* $T_{acc} >> 1/|\mathbf{Y}|$.

Proof: In absence of $\mathbb{W}$, the model does not encounter any input filtered by W and W^- during its training. Thus the model does not contain any special rules on dealing with the out-of-range values defined by W and W^-.

Using *Corollary 1*, we can declare that, since W^- is absent from the model, for any inference input $x \oplus W^-$, the model's softmax activation will be determined solely by the exact out-of-range values and patterns defined by W^- and not x.

Since the softmax output only depends on W^- (a fixed metric), the model will classify any input $x \oplus W^-$ to a single, unified label (that is independent of x) rather than $\mathbf{F}_\theta(x)$. Since this label belongs to $\mathbf{Y}$, we have

$$\Pr_{x \in X^T}(\mathbf{F}_\theta(x \oplus W^-) = \mathbf{F}_\theta(x)) \approx 1/|\mathbf{Y}|.$$

Similarly, we can prove that the model will classify any input $x \oplus W$ into a single unified label, and $\Pr_{x \in X^T}(\mathbf{F}_\theta(x \oplus W) = y_W) \approx 1/|\mathbf{Y}|$. Since $T_{acc} >> 1/|\mathbf{Y}|$, the verification of $\mathbb{W}$ fails.

Proof of Theorem 2.

Theorem 2. Once a model $\mathbf{F}_\theta$ *is trained and includes a watermark* $\mathbb{W}$*, it is impossible to apply null embedding of a different watermark* $\mathbb{W}_A$ *into the model.*

Proof: *Theorem 1* shows that our proposed watermark is robust to false positives. Thus an attacker can only attempt to embed a new watermark $\mathbb{W}_A$ by *fine-tuning* the model weights while using $x \oplus W_A$ and $x \oplus W_A^-$ as new training inputs. Each fine-tuning attempt includes three sequential steps: a) forward inference, b) calculate the model loss, c) backward propagation to update model weights using gradient of the loss.

In the following, we show that such fine-tuning attempt for null embedding cannot be completed since the calculation of the model loss (defined by the cross entropy function) will produce a number $log(0)$ when it encounters out-of-bound values different from those defined by W^-. Since practical DNN implementations treat $log(0)$ as an irrational number, they will not proceed to calcuate the loss. As such the fine-tuning process will not run the subsequent backward propagation step.

To prove the above claim, we first define $z_i, i \in \mathbf{Y}$ as the softmax activation of the last hidden layer of the model, $z_i = \frac{e^{s_i}}{\sum_{c \in \mathbf{Y}} e^{s_c}}$. Here s_i is the output of the last hidden layer for class i, and it is a function of the model input. Using Corollary 2, we can show that, for any input $x \oplus W_A^-$, $W_A^- \neq W^-$, the followings are true:

(i) Each s_c is determined solely by the out-of-range values defined by W_A^- and not x.

(ii) The hidden layer output will include one dominant entry that is significantly larger than the rest. Thus the resulting softmax activation becomes a 1-hot vector:

$$\begin{gathered} s_i = \max_{c \in \mathbf{Y}} s_c, \;\; s_i >> s_c, \; \forall c \neq i \\ \text{and} \;\; z_i = 1, \;\; z_c = 0, \; \forall c \neq i \end{gathered} \tag{9}$$

Therefore, when applying inference on $x \oplus W_A^-$ where x's true label class y is not i, the loss function becomes:

$$\ell_{\mathbf{F}}(x \oplus W_A^-, y) = -\sum_{c \in \mathbf{Y}} y_c log(z_c) = log(0).$$

This proves the claim and thus Theorem 2.

Proof of Theorem 3.

Theorem 3. Given a model $\mathbf{F}_\theta$ *containing watermark* $\mathbb{W}$*, in order to identify* W *and* y_W*, an attacker can not apply any loss or gradient based optimization to reduce the cost of querying* $\mathbf{F}_\theta$*. Instead, the attack needs to random query* $\mathbf{F}_\theta$.

Proof: To recover W and y_W directly from $\mathbf{F}_\theta$, an attacker must repeatedly construct W_{test} and query $\mathbf{F}_\theta$ using filtered inputs $x \oplus W_{test}$ and $x \oplus W_{test}^-$. Like existing works on backdoor trigger identification [30], one can potentially guide the construction of W_{test} using the model softmax output and loss function value. If successful, this could largely reduce the search overhead [30].

However, we prove that under our watermark design, this type of optimization/reduction cannot be performed. Again we leverage the same argument in proving Theorem 2, where *any* $x \oplus W_{test}$ *where* $W_{test} \neq W^-$ *will produce softmax activation as a 1-hot vector defined by eq.(9) and the subsequent irrational* $log(0)$ *condition.* Since the process cannot calcuate the model loss, no optimization can be applied to construct future W_{test}. This completes the proof.

Proof of Theorem 5.

Theorem 5. Given a model $\mathbf{F}_\theta$ *containing watermark* $\mathbb{W}$*, an attacker with perfect knowledge of* $< W, y_W >$ *can not fine tune the model to change* $\mathbf{F}_\theta(x \oplus W)$ *to be different from* y_W.

Proof: Similar to the proof of Theorem 2, we show that when $\mathbf{F}_\theta$ classifies $x \oplus W$, the inference will produce an 1-hot vector as the softmax activation output, *i.e.* with the entry of class y_W as 1 and the rest as 0. Since the goal of fine-tuning is to change the output label to a class different from y_W, the corresponding loss function (for changing the classification result) will reduce to $log(0)$. As such the fine-tuning process will not proceed to the backward propagation step, and thus no weights will be updated.

Definition and Proof of Corollary 1 and 2.

Corollary 1: Assume that a wonder filter, W_A*, is* **absent** *from a model* $\mathbf{F}_\theta$*. For any inference input* $x \oplus W_A^-$*, the output of each hidden layer and the final softmax activation will all be determined entirely by the exact out-of-range values and patterns defined by* W_A^- *and not x.*

Proof: Assume all neurons in $\mathbf{F}_\theta$ use the ReLU activation function, a common activation function in neural networks which exhibits the following behavior:

$$f(x) = \begin{cases} 0, \text{ if } x < 0 \\ x, \text{ otherwise.} \end{cases} \tag{10}$$

Recall that a neuron n_i applies f to the sum of all its inputs q_i, multiplies the result by its weight w_i, and adds its bias b_i. Thus the output O of a neuron n_i is

$$\mathsf{O}(n_i) = w_i \cdot f(\sum q_i) + b_i, \tag{11}$$

As such, each output of the hidden layer will include the contribution of *each* pixel of x. Because (a) the out-of-range values defined by W_A^- have amplitude $\rho \to \infty$[1], which is significantly larger than the normal pixel values (*i.e.* [0,1]), and (b) $|w_i| << \infty$, $|b_i| << \infty$, the contribution of these out-of-range values ($\pm\infty$) will overpower the contributions of normal pixels. Specifically, the raw out-of-range values include both $+\infty$ and $-\infty$ values, so that the output of the first hidden layer will also include both $+\infty$ and $-\infty$ values. Even after each neuron applies the ReLU function to remove $-\infty$ values, the output of some neurons will still include both $+\infty$ and $-\infty$ values since w_i can be either positive or negative.

Finally, such dominance will carry over to the output of the last hidden layer, because the model does not include any rules that remove the impact of out-of-range values defined by W_A^-. This means that the output of the last hidden layer and its softmax activation are completely determined by W_A^- and not x. This completes our proof.

We note that only our proposed null embedding can train the model to *ignore* the impact of the out-of-range pixels defined in the embedded pattern. As long as W_A^- differs from W^- that is embedded in $\mathbf{F}_\theta$, the above claim holds.

Corollary 2: Assume that a wonder filter, W_A, is **absent** *from a model $\mathbf{F}_\theta$. For any inference input $x \oplus W_A^-$, the output of the last hidden layer will be dominated by a single large entry. This turns the softmax activation into a 1-hot vector,* i.e.

$$\begin{aligned} &s_i = \max_{c \in \mathbf{Y}} s_c, \;\; s_i >> s_c, \; \forall c \neq i \\ &\textit{and} \;\; z_i = 1, \;\; z_c = 0, \; \forall c \neq i \\ &\textit{where} \;\; z_i = \frac{e^{s_i}}{\sum_{c \in \mathbf{Y}} e^{s_c}} \end{aligned} \tag{12}$$

Proof: Since W_A^- is not embedded into the model, the model does not have any rules that remove the impact of out-of-range values of W_A^- on the output of the last hidden layer. As these $\pm\rho$ values propagate to the last hidden layer, we can compute the top two largest entires of the output vector. Let's assume they are s_i and s_j, respectively. Since $\rho \to \infty$, we can represent them by $s_i = k_i\rho$ and $s_j = k_j\rho$, where $k_i > k_j$. Here k_i and k_j are a function of the model weights for class i and j, respectively. We show that $k_i \neq k_j$ because an accurate model will not assign identical weight combinations to two different classes. Thus we have

$$s_j - s_i = (k_j - k_i)\rho = -\infty \tag{13}$$

Since s_i and s_j are the top two entires in the output vector:

$$s_c - s_i \leq s_j - s_i = -\infty, \;\; \forall c \in \mathbf{Y}. \tag{14}$$

To compute $z_i = \frac{e^{s_i}}{\sum_{c \in \mathbf{Y}} e^{s_c}}$, we take the following step:

$$\begin{aligned} \frac{1}{z_i} &= \sum_{c \in \mathbf{Y}} e^{s_c - s_i} = 1 + \sum_{c \in \mathbf{Y}, c \neq i} e^{s_c - s_i} \\ &\leq 1 + (|\mathbf{Y}| - 1) e^{s_j - s_i} \end{aligned} \tag{15}$$

Combine this with (14), we have $z_i \geq 1$. But since $z_i = \frac{e^{s_i}}{\sum_{c \in \mathbf{Y}} e^{s_c}} \leq 1$, we conclude that $z_i = 1$. Similarly, using (14), we show that $z_c = 0$, $\forall c \neq i$. This completes our proof.

[1]Our proof assumes that the out-of-range values to have significantly larger amplitudes than the normal values in x, *i.e.* a value $\rho \to \infty$. In our practical implementation, we found that 2000 is sufficient.