# Piracy Resistant Watermarks for Deep Neural Networks

Huiying Li
*University of Chicago*

Emily Wenger
*University of Chicago*

Ben Y. Zhao
*University of Chicago*

Haitao Zheng
*University of Chicago*

## Abstract

As companies continue to invest heavily in larger, more accurate and more robust deep learning models, they are exploring approaches to monetize their models while protecting their intellectual property. Model licensing is promising, but requires a robust tool for owners to claim ownership of models, *i.e.* a watermark. Unfortunately, current designs have not been able to address *piracy attacks*, where third parties falsely claim model ownership by embedding their own "pirate watermarks" into an already-watermarked model.

We observe that resistance to piracy attacks is fundamentally at odds with the current use of incremental training to embed watermarks into models. In this work, we propose *null embedding*, a new way to build piracy-resistant watermarks into DNNs that can only take place at a model's initial training. A null embedding takes a bit string (watermark value) as input, and builds strong dependencies between the model's normal classification accuracy and the watermark. As a result, attackers cannot remove an embedded watermark via tuning or incremental training, and cannot add new pirate watermarks to already watermarked models. We empirically show that our proposed watermarks achieve piracy resistance and other watermark properties, over a wide range of tasks and models. Finally, we explore a number of adaptive counter-measures, and show our watermark remains robust against a variety of model modifications, including model fine-tuning, compression, and existing methods to detect/remove backdoors. Our watermarked models are also amenable to transfer learning without losing its watermark properties.

## 1 Introduction

State-of-the-art deep neural networks (DNNs) today are incredibly expensive to train. For example, a new conversational model from Google Brain includes 2.6 billion parameters, and takes 30 days to train on 2048 TPU cores [2]. Even "smaller" models like ImageNet require significant training (128 GPUs for 52 hours) to add robustness properties.

As training costs continue to grow with each generation of models, providers must explore approaches to monetize models and recoup their training costs, either through Machine Learning as a Service (MLaaS) platforms (*e.g.* [17,32]) that host models, or fee-based licensing of pretrained models. Both have serious limitations. Hosted models are vulnerable to a number of model inversion or inference attacks (*e.g.* [8,25,28]), while model licensing requires a robust and persistent proof of model ownership.

DNN watermarks [4, 26, 33] are designed to address the need for proof of model ownership. A robust watermark should provide a persistent and unforgeable link between the model and its owner or trainer. 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*).

Despite a variety of approaches, current proposals have failed to achieve the critical property of piracy resistance. Without it, a user of the model can train their own "valid" watermark into an already watermarked model, effectively claiming ownership while preserving the model's classification accuracy. Specifically, recent work [30] showed that regularizer-based watermarking methods [4, 5, 26] were all vulnerable to piracy attacks. More recent watermark designs rely on embedding classification artifacts into models [1,33]. Unfortunately, our own experiments show that both techniques can be overcome by successfully embedding pirate watermarks with moderate training.

But what makes piracy resistance so difficult to achieve? The answer is that neural networks are designed to accept incremental training and fine-tuning. DNNs can be fine-tuned with existing training data, trained to learn or unlearn specific classification patterns, or "retargeted" to classify input to new labels via transfer learning. In fact, existing designs of DNN watermarks rely on this incremental training property to embed themselves into models. Thus it is unsurprising that with additional effort, an attacker can use the same mechanism to embed more watermarks into an already watermarked model.

In this work, we propose *null embedding*, a new approach for embedding piracy-resistant watermarks into deep neural networks. Null embedding does not rely on incremental training. Instead, it can only be trained into a model at time of initial model training. Formally speaking, a null embedding (parameterized by a bit string) imposes an additional constraint on the optimization process used to train a model's normal classification behavior, *i.e.* the classification rules used to classify normal input. As this constraint is imposed at time of initial model training, it inexorably builds strong dependencies between normal classification accuracy and the given null embedding parameter. After a model is trained with a given null embedding, further (incremental) training to add a new null embedding fails, because it generates conflicts with the existing null embedding and destroys the model's normal classification accuracy.

Based on the new null-embedding technique, we propose a strong watermark system that integrates public key cryptography and verifiable signatures into a bit-string embedded as a watermark in a DNN model. The embedded bit string inside a watermarked model is easily identified and securely associated with the model owner. The presence of the watermark does not affect the model's normal classification accuracy. More importantly, attempts to train a different, pirate watermark into a watermarked model would destroy the model's value, *i.e.* its ability to classify normal inputs. This deters any piracy attacks against watermarked models.

Our exploration of the null-embedding watermark produces several key findings, which we summarize below:

- We validate the null-embedding technique and associated watermark on a variety of model tasks and architectures. We show that piracy attacks actually destroy model classification properties, and are no better than training the model from scratch, regardless of computation effort (§5.1, §7.2). We also confirm that we achieve all basic watermark properties (§7.3).
- We evaluate against several countermeasures. We show watermarks cannot be removed by modifications such as model fine-tuning, neuron pruning, model compression, or backdoor detection methods. They disrupt the model's normal classification before they begin to have any impact on the watermark (§8.1, §8.2). We discuss model extraction attacks and why they are impractical due to requirements on in-distribution training data (§8.3).
- We show that watermarked models are amenable to transfer learning: models can learn classification of new labels without losing its watermark properties (§8.4).

Overall, our empirical results show that null embeddings show promise as a way to embed watermarks that resist piracy attacks. We discuss limitations and future work in §9.

## 2 Related Work

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

### 2.1 Digital Watermarks for Multimedia Data

Watermarking multimedia data has been widely studied in the literature (*e.g.* a survey [11]). A watermark can be added to *images* by embedding a low-amplitude, pseudorandom signal on top of the host image. To minimize the impact on the host, one can add it to the least significant bits of grayscale images [27], or leverage various types of statistical distributions and transformations of the image (*e.g.* [3, 13, 23]). For video, a watermark can take the form of imperceptible perturbations of wavelet coefficients for each video frame [21] or employ other perception measures to make it invisible to humans [31]. Finally, watermarks can be injected into audio by modifying its Fourier coefficients [3, 22, 24].

### 2.2 Digital Watermarks for DNNs

Recent works have examined the feasibility of injecting watermarks into DNN models. They can be divided into two groups based on the embedding methodology.

**Weights-based Watermarks.** The first group [4, 5, 26] embeds watermarks directly onto model weights, by adding a regularizer containing a statistical bias during training. But anyone knowing the methodology can extract and remove the injected watermark without knowing the secret used to inject it. For example, a recent attack shows that these watermarks can be detected and removed by overwriting the statistical bias [30]. Another design [7] enables "ownership verification" by adding special "passport" layers into the model, such that the model performs poorly when passport layer weights are not present. This design relies on the secrecy of passport layer weights to prove model ownership. Yet the paper's own results show attackers can reverse engineer a set of effective passport layer weights. Since there is no secure link between these weights and the owner, attackers can reverse engineer a set of valid weights and claim ownership.

**Classification-based Watermarks.** The second approach embeds watermarks in model classification results. Recent work [33] injects watermarks using the backdoor attack method, where applying a specific "trigger" pattern (defined by the watermark) to any input will produce a model misclassification to a specific target label. However, backdoor-based watermarks can be removed using existing backdoor defenses (*e.g.* [29]), even without knowing the trigger. Furthermore, this proposal provides no verifiable link between the trigger and the identity of the model owner. Any party who discovers the backdoor trigger in the model can claim they inserted it, resulting in a dispute of ownership.

Another work [1] uses a slightly different approach. It trains watermarks as a set of classification rules associated with a set of self-engineered, abstract images only known to the model owner. Before embedding this (secret) set of images/labels into the model, the owner creates a set of commitments over the image/label pairs. By selectively revealing these commitments and showing that they are present in the model, the owner proves their ownership.

## 3 Problem Context and Threat Model

To provide context for our later discussion, we now describe the problem setting and our threat model.

**Ownership Watermark.** Our goal is to design a robust *ownership watermark*, which 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 DNN model $\mathbf{F}_\theta$ for a specific task, leveraging significant resources to do so (e.g. 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 their ownership of $\mathbf{F}_\theta$ on demand, O embeds watermark $\mathbb{W}$ into the model simultaneously when training the model. This watermark needs to be robust against attacks by a malicious adversary *Adv*.

**Threat Model.** At a high level, the adversary *Adv* wants to stake its own ownership claims on $\mathbf{F}_\theta$ or at least destroy O's claims. We summarize possible adversary goals as follows:

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

We make two more assumptions about the adversary. *First*, *Adv* is not willing to sacrifice model functionality, *i.e.* the attack fails if it dramatically lowers the model's normal classification accuracy. *Second*, *Adv* has limited training data and finite computational resources. If *Adv* has as much or even more training data as O, then it would be easier to train its own model from scratch, 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 terms of computational resources and time than training a model from scratch. Our goal is to make compromising a watermark sufficiently difficult, such that it is more cost-efficient for an adversary to pay reasonable licensing costs instead.

## 4 Understanding Piracy Resistance

Although *piracy resistance* is a critical requirement for DNN watermark, all existing works are vulnerable to piracy attacks. In this section, we demonstrate this vulnerability, discuss why existing designs fail to achieve piracy resistance, and propose an alternative design.

### 4.1 The Need for Piracy Resistance

In an *ownership piracy* attack, an attacker attempts to embed his watermark into a model that is already watermarked. If the attacker can successfully embed her watermark into the watermarked model, the owner's watermark can no longer prove their (unique) ownership. That is, the ambiguity introduced by the presence of multiple watermarks invalidates the true owner's claim of ownership. To be effective, a DNN watermark *must* resist ownership piracy attacks.

### 4.2 Existing Works are Not Piracy Resistant

We show that, unfortunately, all existing DNN watermarking schemes are vulnerable to ownership piracy attacks.

**Piracy Resistance of Weights-based Watermarks.** Recent work [30] already proves that regularizer-based watermarking methods [4,5,26] are vulnerable to ownership piracy attacks, *i.e.* an attacker can inject new watermarks into a watermarked model without compromising the model's normal classification performance. Furthermore, the injection of a new watermark will largely degrade or even remove the original watermark. Another watermark design in this category [7] also fails to achieve piracy resistance because it cannot securely link an embedded watermark to the model owner. An attacker can demonstrate the existence of a pirate watermark without embedding it into the model.

**Piracy Resistance of Classification-based Watermarks.** In the following section, we show empirically that existing works [1, 33] are vulnerable to piracy attacks. We follow the original papers to re-implement the proposed watermarking schemes on four classification tasks (`Digit`, `Face`, `Traffic`, and `Object`). Details of these tasks are listed in §7.1. Additional details concerning the DNN model architectures, training parameters, and watermark triggers used in our experiments can be found in the Appendix A.2.

To implement piracy attacks, we assume a strong attacker who has access to 5,000 original training images and the watermarked model. The goal of the attacker is to inject a new, verifiable pirate watermark into the model. This is achieved by the attacker updating the model using training data related to the pirate watermark. We found that for all four DNN models, a small number of training epochs is sufficient to successfully embed the pirate watermark. `Digit` and `Object` need only 10 epochs for both [33] and [1], `Face` only needs 1 epoch for both, while `Traffic` needs 10 epochs for [33] and 25 epochs for [1].

To evaluate each method's piracy resistance, we use three metrics: (1) the model's normal classification accuracy, (2) its classification accuracy on the original (owner) watermark, and (3) its classification accuracy on the pirate watermark. We record these before and after the piracy attack to measure the impact of the attack. In an ideal watermark design, no piracy attack should be able to successfully embed a pirate watermark into a model while maintaining its classification

| Task | Watermark Design [1] | | | Watermark Design [33] | | |
|---|---|---|---|---|---|---|
| | Normal Classification | Original Watermark | Pirate Watermark | Normal Classification | Original Watermark | Pirate Watermark |
| `Digit` | 98.44% / 97.23% | 100% / 49.00% | **98.00%** | 98.68% / 98.40% | 99.81% / 78.00% | **99.22%** |
| `Face` | 98.72% / 95.52% | 100% / 53.00% | **98.00%** | 98.07% / 98.13% | 96.00% / 28.00% | **98.00%** |
| `Traffic` | 98.23% / 97.63% | 100% / 73.00% | **98.00%** | 97.71% / 97.72% | 100% / 100% | **100%** |
| `Object` | 83.66% / 83.66% | 100% / 99.00% | **98.00%** | 86.13% / 85.31% | 99.80% / 99.80% | **98.60%** |

Table 1: Performance of watermarked models for four classification tasks before and after a piracy attack. For each entry formatted X/Y, X and Y represent the metrics before and after the piracy attack, respectively. The metrics are model classification accuracy on normal inputs, accuracy of the original watermark task, and accuracy of the pirate watermark task. For simplicity, we only show the pirate watermark accuracy *after* the piracy attack has taken place.

accuracy for normal inputs.

We list the results in Table 1. For both watermark designs, piracy attacks succeed (are recognized consistently) across all four classification tasks, and introduce minimal changes to the normal classification accuracy. For some models, the piracy attack also heavily degrades the original watermark. These results show that existing watermark designs are vulnerable to piracy attacks.

**Note on the Piracy Claim in [1].** [1] assumes that the adversary uses the same number of watermark training epochs as the model owner, and applies *an additional verification step* via fine tuning, and claims the original watermark is more robust against fine-tuning than the pirate watermark. For completeness, we perform additional tests that reproduce the exact experimental configuration (same number of original/pirate watermark training epochs, followed by 10 epochs of fine-tuning) as [1]. Contrary to [1], our results show that across all four tasks, the pirate watermark is *more* robust to fine-tuning than the original watermark. For all tasks, the pirate watermark's classification accuracy remains 96+% after fine-tuning, while the accuracy of the original watermark drops to 37-80%.

### 4.3 Rethinking Piracy Resistance

The key obstacle to piracy resistance is the *incremental trainability* property inherent to DNN models. A pretrained model's parameters can be further tweaked by fine-tuning the model with more training data. Such fine-tuning can be designed to not disturb the foundational classification rules used to accurately classify normal inputs, but change fine-grained model behaviors beyond normal classification, *e.g.* adding new classification rules related to a backdoor trigger.

**Existing Watermark Methodology: Separating Watermark from Normal Classification.** Existing watermark designs, particularly classification-based watermarks, leverage the *incremental trainability* property to inject watermarks. In these designs, the model's normal classification behaviors are made *independent* of the watermark-specific behaviors. Thus the foundational classification rules learned by the model to classify normal inputs will not be affected by the embedded watermark. Such independency or isolation allows an adversary to successfully embed new (pirate) watermarks into the model without affecting normal classification.

**Our New Methodology: Using Watermark to Control Normal Classification.** Instead of separating watermark from normal classification, we propose to use the ownership watermark to constrain (or regulate) the generation/optimization of normal classification rules. Furthermore, this constraint is imposed at time of initial model training, creating strong dependencies between normal classification accuracy and the specific bit string in the given watermark. Once a model is trained / watermarked, further (incremental) training to add a new (pirate) watermark will break the model's normal classification rules. Now the updated model is no longer useful, making the piracy attack irrelevant.

A stubborn adversary can continue to apply more training to "relearn" normal classification rules under the new constraint imposed by the pirate watermark. Yet the corresponding training cost is significantly higher (*e.g.* by a factor of 10 in our experiments) than training the model (and adding the pirate watermark) from scratch. Such significant (and unnecessary) cost leaves no incentive for piracy attacks in practice.

## 5 Piracy Resistance via Null Embedding

Following our new methodology, we now describe "null embedding", an effective method to implement watermark-based control on the generation of normal classification rules. In a nutshell, null embedding adds a global, watermark-specific optimization constraint on the search for normal classification rules. This effectively projects the optimization space for normal classification to a watermark-specific area. Since different watermarks create different constraints and thus projections of optimization space, embedding a pirate watermark to a watermarked model will create conflicts and break the model's normal classification.

In this section, we describe the operation of null embedding and its key properties that allow our watermark to achieve piracy resistance. Then, we show that a practical DNN watermark needs to be "dual embedded" (via both true and null embedding) into the model to achieve piracy resistance and be effectively verified and linked to the owner.

### 5.1 Null Embedding

Given a watermark sequence (*e.g.* a 0/1 bit string), the process of null embedding uses this sequence to modify the effective optimization space used to train the model's normal

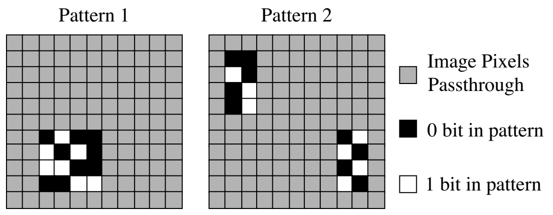

Figure 1: *Two examples of a null embedding pattern. For each filter pattern, the color of a pixel represents the value of that pixel in the filter: gray means no change (value -1), black means* 0 *and white means* 1*. Each filter pattern is defined by the spatial distribution of the black/white pixel areas and the bit pattern in each black/white area.*

classification rules. This is achieved by imposing a constraint during training, preventing a specific configuration on model inputs from affecting the normal classification outcome. To do so, null embedding must take place at the time of original model training. The model owner will start with an untrained model, generate extra training data related to null embedding, and train the model using the original and extra training data.

We formally define the process as follows. Let $\mathbf{F}_\theta : \mathbb{R}^N \to \mathbb{R}^M$ be a DNN model that maps an input $x \in \mathbb{R}^N$ to an output $y \in \mathbb{R}^M$. Let a watermark pattern $p$ be a filter pattern applied to an image. Two samples are shown in Figure 1. Each filter pattern is defined by the placements of the black (0) and white (1) pixels on top of the gray background pixels (-1).

**Definition 1 (Null Embedding)** *Let $\lambda$ be a very large positive value ($\lambda \to \infty$). A filter pattern $p$ is successfully null-embedded into a DNN model $\mathbf{F}_\theta$ iff*

$$\mathbf{F}_\theta(x \oplus [p,\lambda]) = \mathbf{F}_\theta(x) = y, \quad \forall x \in \mathbb{R}^N, \tag{1}$$

*where y is the true label of x. Here $x \oplus [p,\lambda]$ is an input filter operation. For each white (1) pixel of p, it replaces the pixel of x at the same position with $\lambda$; for each black (0) pixel of p, it replaces the corresponding pixel of x with $-\lambda$; the rest of x's pixels remain unchanged.*

This shows that when $p$ is successfully null embedded into the model, changing a set of $p$-defined pixels on *any* input $x$ to hold extreme values $\lambda$ and $-\lambda$ would not change the classification outcome. This condition (and the use of extreme values) set a strong and deterministic constraint on the optimization process used to learn the normal classification rules. And by enforcing the constraint defined by (1), null embedding of a pattern $p$ will *project* the model's effective input-vs-loss space (*i.e.* the optimization landscape) into a sub-area defined by $p$.

**Properties of Null Embedding.** We show that null embedding displays two properties that help us design pirate resistant watermarks. We also verify these properties empirically.

***Observation 1***: *When $N_p$, the number of white/black pixels in p, is reasonably small, null embedding p into a model does not affect the model's normal classification accuracy.*

Null embedding of $p$ confines the model's optimization landscape into a sub-area. As long as this sub-area is sufficiently large and diverse, one can train the model to reach the desired normal classification accuracy. Our hypothesis is that when $N_p$ is reasonably small compared to the size of the input image, the sub-area defined by $p$ would be sufficiently large and diverse to learn accurate normal classification.

We tested this hypothesis on the same four classification tasks used in §4 (`Digit`, `Face`, `Traffic`, and `Object`), and found that for all of them $N_p$ can be up to 10% of the total input size without causing noticeable impact on normal classification accuracy. For example, $N_P = 6 \times 6$ on $28 \times 28$ images results in only 0.1%-1.5% accuracy loss. One can potentially reduce this loss by optimizing the design of filter pattern, *e.g.* configuring white/black pixel area as irregular shapes, which we leave to future work.

***Observation 2***: *Once a model is trained and null embedded with p, an adversary cannot null embed a pirate $p'$ ($p' \neq p$) without largely degrading the model's normal classification accuracy.*

Our hypothesis is that null embedding of different patterns will create different projections of the optimization space. Once a model is successfully trained on $p$-based optimization space, any attempt to move it to a different optimization space (defined by $p'$) will immediately break the model.

We visualize this for the `Digit` task by plotting the model's normal classification accuracy and pirate $p'$ classification accuracy as the adversary fine-tunes the model to embed $p'$. Figure 2(a) shows that even if the adversary has the full training data, the initial few updates reduce the normal classification accuracy from 99% down to 10% (random guess). The performance remains broken even after 2k training cycles. Eventually, after 1000k training cycles, both classification accuracy metrics reach the original level. Here we use training cycles rather than epochs to visualize fine-trained model behavior during training. Figure 2(b), where the adversary with full training data trains the model from scratch while null embedding $p'$. This non-adversarial approach only requires 100k training cycles to reach the same accuracy levels. Trying to add a watermark takes 10x longer as just training it from scratch. In Figure 2(c), we consider adversaries with less (1%) of the training data. Piracy attack also quickly breaks classification, but restoring it seems to take exponential longer time compared to training from scratch (accuracy of the blue line).

The significant cost of piracy attack is due to the *unlearning-then-relearning* effect. The adversary must first train the model to *unlearn* existing classification rules trained on $p$, then *relearn* new normal classification rules trained on $p'$. This overhead makes piracy attacks impractical.

**Generating Distinct Watermark Patterns.** Our design achieves piracy resistance by assuming that different watermark patterns project the optimization space differently. In

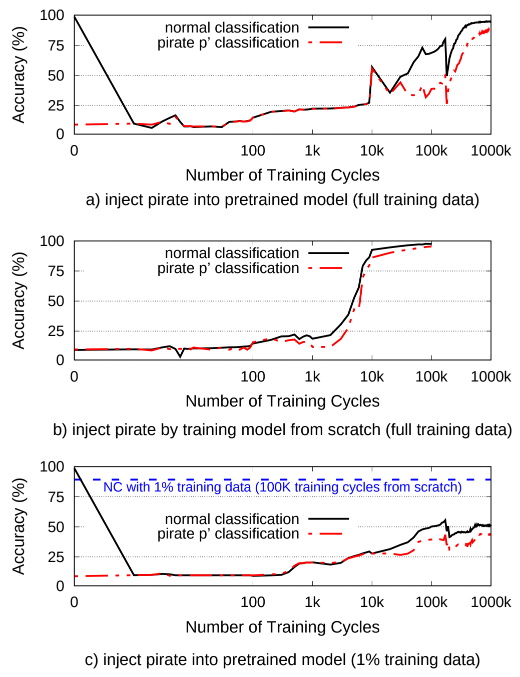

Figure 2: *The significant cost of piracy attack on null embedding (*`Digit`*). (a) Piracy attack: an adversary with full training data trains the model to inject the piracy pattern $p'$. The updates quickly break normal classification. A stubborn adversary takes 1000k training cycles to null embed $p'$ and restore normal classification. (b) An adversary with full training data can retrain the model from scratch with null embedding $p'$ in 1/10 the time. This takes 100k training cycles. (c) Adversary with 1% of training data tries to inject pirate watermark. Normal classification breaks quickly but cannot approach the normal classification accuracy of a model with same partial data trained from scratch (blue dash line).*

our design, we create distinct watermark patterns by varying the spatial distribution of the white/black pixel areas and the (0/1) bit pattern within these areas (*e.g.* the two samples patterns in Figure 1). We also choose $N_P$ to be a moderate value to reduce the collision probability across watermark patterns. Finally, we couple the watermark generation with strong cryptographic tools (*i.e.* public-key signatures in §6.1), preventing any adversary from forging the model owner's watermark.

## 5.2 Integrating Null and True Embeddings

While enabling piracy resistance, a null embedding alone is insufficient to build effective DNN watermarks. In particular, we found that the verification of solely null embedding-based watermarks could produce some small false positives. One potential cause is that some input regions could naturally have little impact on classification outcome, leading to false detection of watermarks not present in the model.

Thus we propose combining the null embedding with a *true embedding* (similar to the backdoor based embedding used by existing watermark designs). In this design, true embedding links the watermark pattern with a deterministic (thus verifiable) classification output independent of the input (*i.e.* the watermark is a trigger in a backdoor). Combined with null embedding, they effectively minimize false positives in watermark verification.

**Dual Embedding.** We integrate the two embeddings by assigning them complementary patterns. This ties the embeddings to the same watermark without producing any conflicts. Given a watermark pattern $p$, the null embedding uses $p$, while the true embedding uses $inv(p)$. Here $inv(p)$ does not change any gray pixels (-1) in $p$ but switches each white pixel to a black pixel and vice versa. We refer to this combination as *dual embedding* and formally define it below. Figure 3 illustrates dual embedding by its two components.

**Definition 2 (Dual Embedding)** *Let $\lambda$ be a very large positive value ($\lambda \to \infty$). A watermark pattern $p$ is successfully dual embedded into a DNN model $\mathbf{F}_\theta$ iff $\forall x \in \mathbb{R}^N$,*

$$\mathbf{F}_\theta(x \oplus [p,\lambda]) = \mathbf{F}_\theta(x) = y, \tag{2}$$

$$\mathbf{F}_\theta(x \oplus [inv(p),\lambda]) = y_W \neq y. \tag{3}$$

*where $y$ is the true label of $x$, and $y_W$ is the watermark-defined label used by true embedding.*

Our proposed true embedding teaches the model that the presence of a $[inv(p),\lambda]$ trigger pattern on any normal input $x$ should result in the classification to the label $y_W$. Our design differs from existing work [33] in that it uses extreme values $\lambda$ and $-\lambda$ to form the trigger. As such, our true embedding does not create anomalous (thus detectable) behaviors like traditional backdoors. As we will show in §8, the use of extreme values in our dual embedding makes our proposed watermark robust against model modifications, including existing backdoor defenses that attempt to detect and remove the watermark.

**Simultaneous Dual Embedding and Model Training.** A dual embedding must be fully integrated with the original model training process. The model owner, starting with an untrained model, generates extra training data related to both true and null embeddings, and trains the model using the original and extra training data. In this way, the model owner simultaneously trains and watermarks the target DNN model.

## 6 Detailed Watermark Design

To build a complete watermarking system, we apply digital signatures, cryptographic hashing, and existing neural network training techniques to generate and inject watermark patterns via dual embedding. Our design consists of the following three components:

**The model owner generates the ownership watermark using her private key (§6.1).** The model owner O uses its

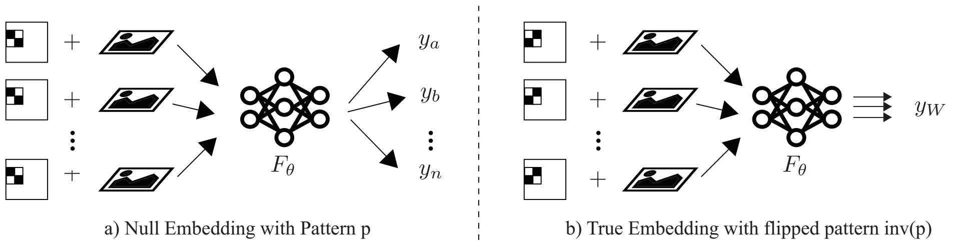
$y_a$
$y_b$
$y_n$
$F_\theta$
a) Null Embedding with Pattern p
$y_W$
$F_\theta$
b) True Embedding with flipped pattern inv(p)

Figure 3: *Our proposed dual embedding of a pattern p. (a) null embedding operates on the original pattern p, creating an input dependent classification output, forcing the model to train normal classification rules on the projected optimization space. (b) true embedding operates on the flipped pattern inv(p), creating a deterministic classification output independent of the input. The dual embedding is integrated with the model training to simultaneously train and watermark the DNN model.*

private key to sign some known verifier string *v* and generate a signature (*sig*). Using *sig*, O applies deterministic hashing functions to produce her ownership watermark $\mathbb{W}$, defined by the filter pattern $p$, $\lambda$, and the true embedding label $y_W$.

**The owner trains the model while injecting watermark (§6.2).** O generates the corresponding training data for the dual embedding of $\mathbb{W}$. O combines these new training data with its original training data to train the model from scratch while embedding the watermark.

**The authority verifying whether the ownership watermark $\mathbb{W}$ is embedded in the model (§6.3).** To prove its ownership, O provides its *sig* to a verification authority *A*. The verification takes two steps. *A* first verifies that *sig* is O's signature using O's public key and verifier string *v*. After verifying *sig*, *A* generates the watermark $\mathbb{W} = (p, y_W, \lambda)$ from *sig*, and verifies that $\mathbb{W}$ exists in the model.

Next, we present detailed descriptions of each component.

## 6.1 Generating Ownership Watermark

The model owner O runs Algorithm 1 to generate its ownership watermark $\mathbb{W} = (p, y_W, \lambda)$.

**Algorithm 1** Generating Ownership Watermark

```
1: $sig$=Sign($O_{pri}$, $v$)
2: $(p, y_W, \lambda)$=Transform($sig$)
```

First, O applies the Sign(.) function to produce a signature *sig*, taking the input of O's private key $O_{pri}$ and a verifier string *v* (a string concatenation of O's unique identifier and a global timestamp). We implement Sign(.) using the common RSA public-key signature.

Next, O runs the Transform(.) function, a deterministic, global function for watermark generation with input *sig* (shown in Algorithm 2) Our implementation applies four hash functions $\mathsf{h}_1, \mathsf{h}_2, \mathsf{h}_3, \mathsf{h}_4$ to generate the specific pattern of the ownership watermark: the filter pattern $p$, the true embedding label $y_w$ and the extreme value $\lambda$. The hash functions can be any secure hash function – we use SHA256. Here we assume $p$ contains a single white/black pixel area of size $n \times n$. We represent $p$ by the bit pattern $bit(p)$ in the white/black square, and the top-left pixel position of the white/black square, $pos(p)$. This easily generalizes to cases where $p$ contains multiple white/black areas.

**Algorithm 2** $(p, y_W, \lambda)$=Transform($sig$)

```
1: $H$= height of input $x$
2: $W$= width of input $x$
3: $Y$= total number of model classes
4: $y_w = \mathsf{h}_1(sig) \bmod Y$
5: $bit(p) = \mathsf{h}_2(sig) \bmod 2^{n^2}$
6: $pos(p) = [\mathsf{h}_3(sig) \bmod (H-n), \mathsf{h}_4(sig) \bmod (W-n)]$
7: $\lambda = 2000$
```

Our watermark generation process can effectively prevent any adversary from forging the model owner's watermark. To forge the owner's watermark, the attacker must either forge the owner's cryptographic signature or randomly produce a signature whose hash produces the correct characteristics, *i.e.* reverse a strong, one-way hash. Both are known to be computationally infeasible under reasonable resource assumptions.

## 6.2 Training Model & Injecting Watermark

Given $\mathbb{W} = (p, y_W, \lambda)$, O generates the watermark training data and labels corresponding to the dual embedding. O then combines the watermark training data with its original training data and uses loss-based optimization methods to train the model while injecting the watermark. In this case, the objective function for model training is defined as follows:

$$\underset{\theta}{\text{argmin}}\ \ell_{\mathbf{F}}(x,y) + \alpha \cdot \ell_{\mathbf{F}}(x \oplus [inv(p), \lambda], y_W) + \beta \cdot \ell_{\mathbf{F}}(x \oplus [p, \lambda], y)$$

where $y$ is the true label for input $x$, $\ell_{\mathbf{F}}(\cdot)$ is the loss function for measuring the classification error (*e.g.* cross entropy), and $\alpha$ and $\beta$ are the injection rates for true and null embedding.

## 6.3 Verifying Watermark

We start by describing the process of *private verification* where the third party verifier is a trusted authority, who keeps the verification process completely private (no leakage of any information). We then extend our discussion to *public verification* by untrusted parties.

**Algorithm 3** Private Verification of Ownership Watermark

```
1: if not Verify($O_{pub}$, $sig$, $v$) then
2:     Verification fails.
3: else
4:     $(p, y_W, \lambda)$=Transform($sig$)
5:     $\phi_{null} = Pr(\mathbf{F}_\theta(x \oplus [p, \lambda]) = \mathbf{F}_\theta(x) = y)$
6:     $\phi_{true} = Pr(\mathbf{F}_\theta(x \oplus [inv(p), \lambda]) = y_W)$
7:     if $min(\phi_{true}, \phi_{null}) > T_{watermark}$ then
8:         Verification passes.
9:     else
10:        Verification fails.
11:    end if
12: end if
```

**Private Verification via Trusted Authority.** The "claimed" owner O submits its signature $sig$, public key $O_{pub}$, and verifier string $v$ to a trusted authority. The authority runs Algorithm 3 to verify whether O does have its ownership watermark embedded in the target model $\mathbf{F}_\theta$. Here we assume that the trusted authority has access to the Transform(.) function (Algorithm 2) and will not leak the signature $sig$ and the corresponding ownership watermark pattern.

The verification process includes two steps. *First*, the authority verifies that $sig$ is a valid signature over $v$ generated by the private key associated with $O_{pub}$ (line 1 of Algorithm 3). This uniquely links $sig$ to $O$. *Second*, the authority checks whether a watermark defined by $sig$ is injected into the model $\mathbf{F}_\theta$. To do so, it first runs Transform($sig$) to generate the ownership watermark $(p, y_W, \lambda)$ (line 4 of Algorithm 3). The authority forms a test input set, and computes the classification accuracy of the null embedding (line 5 of Algorithm 3) and true embedding (line 6 of Algorithm 3). If both accuracies exceed the threshold $T_{watermark}$, the authority concludes that the owner's watermark is present in the model. Ownership verification succeeds.

**Public Verification.** The above private verification assumes the authority can be trusted not share information about the watermark pattern. If the pattern is leaked to an adversary, the adversary can attempt to modify/corrupt the watermark by applying a small amount of training to change the classification outcome of dual embedding $(x \oplus [p, \lambda]$ and/or $x \oplus [inv(p), \lambda])$, so that $min(\phi_{true}, \phi_{null})$ drops below $T_{watermark}$. The result is a new model where the ownership watermark is no longer verifiable. This is the *corruption* attack (not piracy attack) mentioned in §3.

This issue can be addressed by embedding multiple watermarks in the model while only submitting one watermark to the trusted authority. As a result, any hidden or "unannounced" watermark will not be leaked. Should a dispute arise, the owner can reveal one hidden watermark to prove ownership. We have experimentally verified that multiple, independently generated watermarks can be simultaneously added at initial training time into practical DNN models (those used in Section 7) without degrading model accuracy.

## 7 Experimental Evaluation

In this section, we use empirical experiments on four classification tasks to validate our proposed watermark design.

### 7.1 Experimental Setup

We consider four classification tasks targeting disjoint types of objects and employing different model architectures. Thus, our evaluation covers a broad array of settings. We describe each task and its dataset and classification model below (summarized in Table 2). Further details on model structures (Tables 10-13) and training hyperparameters (Table 14) are listed in the Appendix. For all four tasks, we normalize the pixel value of input images to be in the range $[0, 1]$.

- Digit Recognition (`Digit` [14]) classifies images of handwritten digits to one of ten classes. Each input image is normalized so the digit appears in the center. The corresponding classification model contains two convolutional layers and two dense layers.
- Face Recognition (`Face` [16, 18]) seeks to recognize the faces of $1,284$ people. These faces are drawn from a large (3,425) set of YouTube videos. Each person in the target dataset has at least 100 labeled images. The corresponding facial recognition model is the DeepID model [20].
- Traffic Sign Recognition (`Traffic` [19]) recognizes 43 types of traffic signs based on the German Traffic Sign Benchmark (GTSRB) dataset. The classification model contains six convolutional layers and three dense layers.
- Object Recognition (`Object` [12]) recognizes objects in images as one of ten object types. It uses the CIFAR-10 dataset with 60000 color images in 10 classes (6000 images per class). Similarly to `Traffic`, the classification model has six convolutional layers and three dense layers.

**Watermark and Attacker Configuration.** When constructing our watermarks, we set the extreme value $\lambda = 2000$. By default, a watermark pattern $p$ is a $6 \times 6$ area of white/black pixels. In our experiments, we randomly vary the position and black/white pixel locations of our watermark pattern to ensure that our results generalize.

To verifying a watermark, we set $T_{watermark} = 80\%$, the threshold used by our watermark verification algorithm (Algorithm 3). However, the verification outcome is consistent when $T_{watermark}$ is between 50% and 80%.

As described in the threat model (§3), we assume attackers only have a limited subset of the original training data (because otherwise the attacker could easily train their own model and has no need to pirate the owner's model). Thus in our experiments, the attacker has at most 5000 images for all four tasks, the same configuration used by our evaluation of previous work in §4.

| Task | Dataset | # Classes | Training data size | Test data size | Input size | Model architecture |
|---|---|---|---|---|---|---|
| Digit Recognition (`Digit`) | MNIST | 10 | 60,000 | 10,000 | (28, 28, 1) | 2 Conv + 2 Dense |
| Face Recognition (`Face`) | YouTube Face | 1283 | 375,645 | 64,150 | (55, 47, 3) | 4 Conv + 1 Merge + 1 Dense |
| Traffic Sign Recognition (`Traffic`) | GTSRB | 43 | 39,209 | 12,630 | (48, 48, 3) | 6 Conv + 3 Dense |
| Object Recognition 1 (`Object`) | CIFAR-10 | 10 | 50,000 | 10,000 | (32, 32, 3) | 6 Conv + 3 Dense |

Table 2: Overview of classification tasks with their associated datasets and DNN models

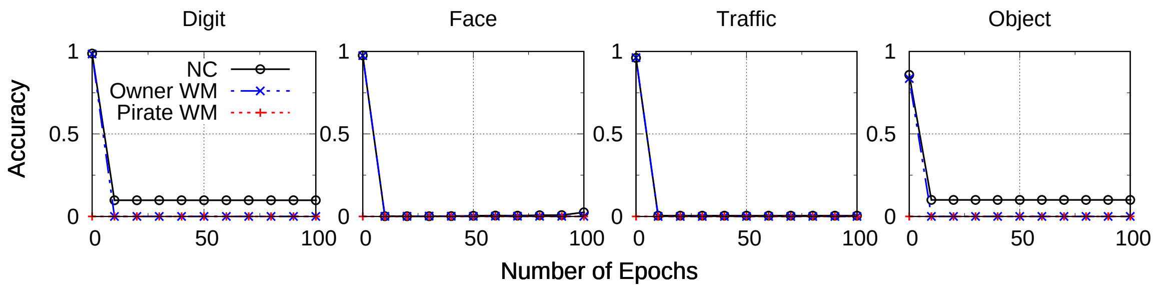

Figure 4: *The model behavior as an adversary attempts to embed a pirate watermark into an already-watermarked model.*

**Evaluation Metrics.** We evaluate the performance of our watermark using two metrics: normal classification accuracy and watermark classification accuracy. We further break down watermark accuracy into its true and null embedding components. The metrics are described below:

- **Normal Classification Accuracy (NC)**: The probability that the classification result of any normal input $x$ equals its true label $y$, *i.e.* $\Pr(\mathbf{F}_\theta(x) = y)$.
- **Watermark Classification Accuracy (WM)**: The minimum classification accuracy of the true and null embedding, $\phi = min(\phi_{true}, \phi_{null})$, where

$$\phi_{null} = \Pr(\mathbf{F}_\theta(x \oplus [p,\lambda]) = \mathbf{F}_\theta(x) = y), \quad (4)$$
$$\phi_{true} = \Pr(\mathbf{F}_\theta(x \oplus [inv(p),\lambda]) = y_W). \quad (5)$$

Note that we will examine the classification accuracy of both the owner watermark and the pirate watermark when we examine our watermark's piracy resistance.

**Overview of Our Experiments.** In this section, we verify that our proposed watermark design both achieves piracy resistance (§7.2) and fulfills the basic watermark requirements (§7.3). Our experiments in this section only consider the threat of piracy attacks. Later, in §8, we examine other potential threats to our watermarking systems such as model fine-tuning, model compression, transfer learning, or intentional efforts to corrupt/remove the ownership watermark.

## 7.2 Piracy Resistance

We start from experiments that verify whether our proposed DNN watermark design can resist piracy attacks. To show this, we conduct piracy attacks on models for all four tasks that have been watermarked using our design. As discussed in §4, the adversary conducts a piracy attack by training the model to inject a new watermark. We assume the original and pirate watermarks both have a fixed-size white/black pixel pattern, *i.e.* $N_p = 6 \times 6$.

**Model Behavior during a Piracy Attack.** We start by visually examining a watermarked model's behavior as a randomly chosen pirate watermark gets injected during a piracy attack. Figure 4 plots the model's normal classification accuracy (NC), owner watermark accuracy (owner WM), and pirate watermark accuracy (pirate WM) as a function of the number of training epochs used by the adversary to inject the pirate watermark. For our configurations for all tasks, each training epoch maps to 39 training cycles.

We see that, for all four tasks, the NC and owner WM accuracies drop nearly to zero after the first few epochs and remain very low as the adversary continues to train the model. This verifies that our watermark design causes the normal classification accuracy (of a watermarked model) to heavily depend on the presence of the owner watermark in the model. Model updates generated by piracy attacks will break both normal classification and owner watermark, making the (updated) model useless.

**Model Performance Before/After a Piracy Attack.** To directly compare our watermark to existing watermark designs evaluated in §4, we apply the same piracy attack configuration used in §4. Specifically, we use 10 training epochs for `Digit` and `Object`, 1 epoch for `Face` and 25 epochs for `Traffic` to inject the pirate watermark, use the last learning rate from the original model training, and use the same configurations of the original model training listed in Table 14 in Appendix.

Table 3 compares the normal classification and watermark accuracies of a watermarked model before and after the adversary tries to embed a new pirate watermark. We report the result as the average value across 100 randomly generated pirate watermarks. For all tasks, the normal classification accuracy drops to near the level of a random guess as the attacker embeds a pirate watermark, rendering the updated model useless. Similarly, the classification accuracy of the owner watermark on the updated model also degrades significantly to

4-18%. This is as expected, because the updated model itself no longer functions in terms of normal classification. Finally, even after 10 epochs of training, the adversary still cannot successfully embed the pirate watermark, *i.e.* the pirate watermark classification accuracy is barely 3-11%.

**Summary.** Together, these results show that the adversary *cannot* successfully inject its pirate watermark without breaking the model's normal classification. While the piracy attack also corrupts the owner's watermark by updating the model parameters, it also renders the updated model useless in terms of performing accurate normal classification. As the modified model no longer functions, the corresponding piracy attack becomes irrelevant.

## 7.3 Basic Watermark Requirements

In addition to being piracy-resistant, our proposed watermark design also fulfills the basic requirements for a DNN watermarking system. These include:
1) functionality-preserving, *i.e.* embedding a watermark does not degrade the model's normal classification;
2) effectiveness, *i.e.* an embedded watermark can be consistently verified;
3) non-trivial ownership, *i.e.* the probability that a model exhibits behaviors of a non-embedded watermark is negligible;
4) authentication, *i.e.* there is a provable association between an owner and their watermark, so that an adversary cannot claim an embedded watermark as their own [26, 33];

We now describe our experiments verifying that our watermark design fulfills these requirements.

**Functionality-preserving.** In Table 4, we compare the normal classification (NC) accuracy of watermarked and watermark-free versions of the same model. Both versions of the model are trained using the same configuration (Table 14 in Appendix), except (obviously) the watermark-free version is not trained on watermark-specific data. For this experiment we randomly generate 10 different owner watermarks for each task and record the average model performance. Overall, the presence of a watermark changes NC accuracy by $-0.93\% \pm 0.65\%$ on average across all tasks.

**Effectiveness.** Using Algorithm 3, we verify that a watermark is present in a model by ensuring watermark classification accuracy is above the $T_{watermark} = 80\%$ threshold. We experiment with 10 random owner watermarks for each task and find that all can be reliably verified (the average WM classification accuracy is shown in Table 4). We also list the classification accuracy of the true and null components. We see that true embedding has a higher classification accuracy since it produces a deterministic behavior independent of the input. By adding an extra constraint on normal classification (see eq. (1)), null embedding's accuracy depends heavily on that of NC. This explains why its value for `Object` is lower than those of the other three tasks.

**Non-trivial ownership.** We first empirically verify that a watermark-free model consistently fails the watermark verification test. We randomly generate 1000 different watermarks for each task and verify their existence in watermark-free models. The verification process fails for all of these non-embedded watermarks, *i.e.* there is 0% false positive rate for watermark-free models.

We repeat the above test on watermarked models. The 1000 watermarks to be verified are not embedded in any watermarked model. The verification process produces a 0% false positive rate on `Digit`, `Face`, `Traffic`, and a 0.1% false positive rate for `Object`. This indicates that our proposed watermark design achieves the non-trivial ownership property.

**Authentication.** Our watermark method satisfies the authentication requirement by design. To generate the ownership watermark in §6.1, we use a hash function that is both a strong one-way hash (*i.e.* difficult to reverse) and collision resistant (low probability of natural collisions). Compromising the watermark requires a third party to find a valid collision to the hash algorithm, and use that input to claim that she is the one who originated the watermark. Since our design uses a preimage-resistant hash (SHA256), such an attack is unrealistic.

## 8 Adaptive Attacks and Countermeasures

In this section, we evaluate our watermark's robustness against four groups of attacks (besides piracy attacks), which an adversary could use to remove or corrupt an embedded watermark. These include (1) commonly used model modifications to improve accuracy and efficiency (§8.1), (2) known defenses to detect/remove backdoors from the model (§8.2), (3) model extraction attacks that create a watermark-free version of the model (§8.3), (4) transfer learning to deploy customized model (§8.4). We show that our proposed watermark design is robust against these attacks and model modifications.

## 8.1 Modifications for Accuracy and Efficiency

Model modification techniques, originally designed to improve model accuracy or efficiency, could also impact the watermark embedded in a model. We experiment with two forms of modifications: (1) fine-tuning to improve accuracy; and (2) model compression via neuron pruning.

**Accuracy-based Fine-tuning.** Fine-tuning is widely used to update model weights to improve normal classification accuracy. We test our watermark's robustness to fine-tuning, allowing weights in all model layers to be updated. We use the same parameters such as batch size, optimizer, and decay as the original model training, and the last learning rate used during the original model training. Figure 5 plots the model's normal classification and watermark classification accuracy (null, true) during fine-tuning. We see that even after 100 epochs of fine-tuning, the embedded watermark and the normal classification are not affected.

| Task | NC | Owner WM | | | Pirate WM | | |
|---|---|---|---|---|---|---|---|
| | | null | true | WM | null | true | **WM** |
| `Digit` | **98.72% / 19.02%** | 98.51% / 11.20% | 100% /8.10% | 98.51% / 1.21% | 10.22% / 17.98% | 8.19% / 40.08% | **0.97% / 9.31%** |
| `Face` | **97.66% / 12.03%** | 97.51% / 3.46% | 100% / 16.53% | 97.51% / 2.15% | 1.05% / 4.25% | 0.00% / 67.40% | **0.00% / 2.92%** |
| `Traffic` | **96.09% / 12.75%** | 96.26% / 12.19% | 100% / 8.13% | 96.26% / 7.79% | 7.56% / 12.71% | 1.68% / 13.00% | **0.28% / 11.12%** |
| `Object` | **85.83% / 17.81%** | 83.58% / 16.38% | 100% / 6.00% | 83.58% / 4.86% | 10.82% / 17.52% | 12.11% / 15.60% | **1.31% / 5.55%** |

Table 3: Normal classification accuracy and watermark accuracies when adversary tries to embed a pirate watermark into owner's model. We show the before/after pirate results in the table. For both owner and pirate watermark results, "null" and "true" represent classification accuracy related to null embedding and true embedding, respectively, and "WM" represents the overall watermark classification accuracy. Each after private result is averaged over 100 randomly pirate watermarks.

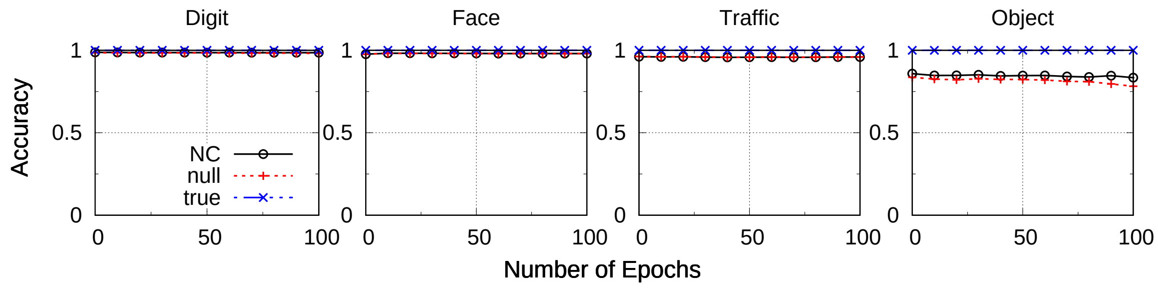

Figure 5: *The model's normal classification and watermark classification accuracy remain stable during model fine-tuning. NC, null, and true represent normal classification accuracy, null embedding accuracy, and true embedding accuracy, respectively.*

| Task | Watermarked-free Model | Watermarked Model | | | |
|---|---|---|---|---|---|
| | NC | NC | null | true | WM |
| `Digit` | 98.51% | 98.51% | 97.70% | 100% | 97.70% |
| `Face` | 99.19% | 97.39% | 97.22% | 100% | 97.22% |
| `Traffic` | 96.84% | 96.10% | 95.76% | 100% | 95.76% |
| `Object` | 85.87% | 84.70% | 82.87% | 100% | 82.87% |

Table 4: Normal classification (NC) and watermark classification accuracy (both true and null components) of watermark-free and watermarked models.

| Task | Original Model | Watermarked Model |
|---|---|---|
| `Digit` | 0.88 | 1.32 |
| `Face` | 2.05 | 1.85 |
| `Traffic` | 2.72 | 1.85 |
| `Object` | 1.12 | 0.99 |

Table 5: Anomaly index reported by Neural Cleanse when running on original (watermark-free) and watermarked models. Suggested threshold for detecting anomalies is 2 [29].

**Neuron Pruning/Model Compression.** Neuron pruning updates and/or compacts a model by selectively removing neurons deemed unnecessary [9, 10]. An adversary could try to use neuron pruning to remove the watermark from the model. We run the common neuron pruning technique [10] on our watermarked models, which first removes neurons with smaller absolute weights (*ascending pruning*). Figure 6 shows the impact of pruning ratio on normal classification and watermark accuracy. We see that since the accuracy of null embedding is tied to the NC accuracy, there is no reasonable level of pruning where normal classification is acceptable while the embedded watermark is disrupted. This shows that our watermark design is robust against neuron pruning.

## 8.2 Backdoor Detection/Removal

The true embedding component of our watermark design is similar to a traditional neural network backdoor. Thus, an adversary may attempt to detect and remove it using existing backdoor defenses. To evaluate this attack, we apply Neural Cleanse [29], the most well-known method for backdoor detection/removal, on our watermarked models. Neural Cleanse detects backdoors by searching for a small perturbation that causes all inputs to be classified to a specific label, and detecting it as an anomaly (*e.g.* whose anomaly index $>2$). For reference, we also apply Neural Cleanse on the watermark-free version of our models.

Neural Cleanse is unable to detect any "backdoor" (aka watermark) on our watermarked models (see Table 5). All watermarked models return values lower than the recommended threshold of 2, and in all but one case, the anomaly index of the watermarked model is lower than that of the original (watermark-free) model. This is because Neural Cleanse (and followup work) assume that backdoors are *small* input perturbations that create large changes in the feature space, and detect these behaviors as anomalies. Since our true and null embeddings use extreme values $-\lambda$ and $\lambda$, they represent *large* perturbations in the input space (L2 distance) that affect the feature space. Thus they do not show up as any anomalies to Neural Cleanse.

## 8.3 Model Extraction Attack

Finally, we consider the possibility of an attacker using a model extraction attack [25] to create a substitute model that is watermark-free. In a model extraction attack, the attacker gathers unlabeled input data and uses the classification results of the target model to label the data. This newly labeled

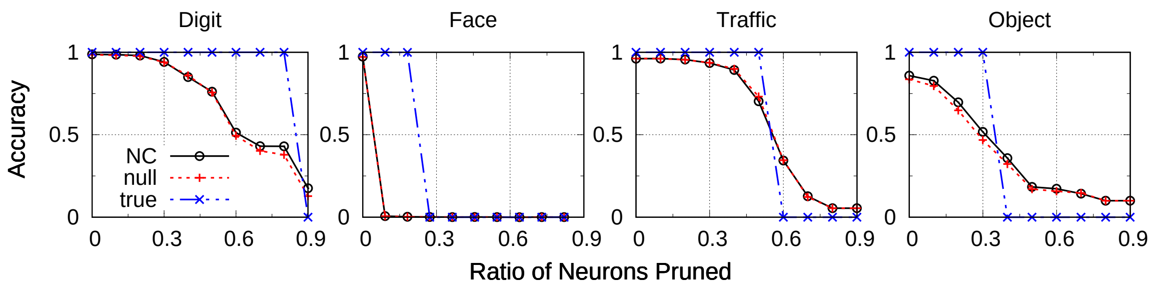

Figure 6: *The impact of neuron pruning on the model's normal classification and watermark classification accuracy (both true and null components), as a function of the ascending pruning ratio. There is no reasonable level of pruning that can disrupt the owner watermark without breaking normal classification (NC) accuracy.*

| Data | 50k (128%) | 100k (255%) | 376k (958%) | 500k (1275%) |
|---|---|---|---|---|
| ImageNet | 91.33% | 92.91% | - | 94.37% |
| YouTube Faces | 69.52% | 72.81% | 76.14% | - |
| Random | 5.46% | 5.46% | - | 5.46% |

Table 6: The normal classification accuracy of the substitute model built by the model extraction attack using each of the three data sources. For each $\alpha(\beta)$ entry in the first row, $\alpha$ is # of (unlabeled) training images used to train the substitute model, $\beta$ is $\alpha$/# of training images of the target model. The target model's normal classification accuracy is 96.1%.

data can be used to train a new watermark-free, substitute model that mimics behavior of the target model.

While model extraction attacks could produce a watermark-free version of the model, our analysis below shows that they do not qualify as a feasible attack against our watermark design given their data requirements. Specifically, we ask two questions on model extraction attacks:

**Q1: How much (unlabeled) source data is needed to create a substitute model with the same normal classification accuracy?** The answer is "at least the same amount" of data required to train the watermarked model. This is true even when the attacker can collect high-quality, task-specific, unlabeled input data. We empirically confirmed this on all four classification tasks.

Our answer is driven by the fact that the extraction attack only uses the watermarked model to label its training data, but does not apply any shortcut to reduce the amount of training images. Furthermore, our watermarked model requires the same training input as its watermark-free version, since generating watermark-related training data does not require collecting extra (normal) images.

**Q2: What happens when the attacker cannot access task-specific data?** For some tasks, collecting a large set of high-quality, task-specific data (even unlabeled) is still costly or impractical. In this case, attackers can choose to use alternative sources from other domains (*e.g.* online scraping or self generation). We experiment to see if out-of-distribution datasets can serve as unlabeled data in model extraction attacks, with 3 datasets: ImageNet, YouTube Faces (376k), and randomly-generated images. We use each dataset to build a substitute model for the watermarked `Traffic` model (traffic sign recognition). Table 6 lists the normal classification accuracy of the substitute models as a function of the training data volume. Among the three data sources, ImageNet performs the best but still requires 12.75x more input data than an in-distribution training dataset to reach similar accuracy (94.37% vs. 96.1% ).

**Summary.** If an attacker can obtain a large in-distribution set of inputs (unlabeled or labeled), a model extraction attack is extremely powerful and unstoppable in most contexts. But this is a very expensive proposition in our context. Those models valuable enough to require IP protection with watermarks are generally valuable because of the large volume of data used in training. We experimentally explore the use of out-of-distribution datasets as unlabeled data for model extraction. We show that in models we consider, the attack requires significantly (12.75x) more data to approach similar levels of accuracy. Thus, we believe model extraction attacks remain impractical for watermarked models, given the extreme requirements for in-distribution data.

## 8.4 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.

Next we show that transfer learning does not degrade our watermark. Specifically, we evaluate two watermark qualities related to transfer learning. First, a watermark (in the teacher model) should allow transfer learning, *i.e.* allow customization of student models with high accuracy. Second, a watermark should persist through the process, *i.e.* still be verified inside trained student models.

We implement a transfer learning scenario on a traffic sign recognition task. Our teacher task is German traffic sign recognition (`Traffic`), and our student task is US traffic sign

| Fine Tuning Configuration | Watermark-free Model's Student NC | Watermarked Model's Student NC |
|---|---|---|
| Added Layers | 82.35% | 74.12% |
| Last Two Layers | 87.65% | 82.06% |
| All Dense Layers | 91.76% | 90.00% |
| All Layers | 91.47% | 92.65% |

Table 7: Student model's normal classification accuracy with watermark-free and watermarked models as teachers.

| Fine Tuning Configuration | Recovered NC | null | true |
|---|---|---|---|
| Last Layer | 96.28% | 96.31% | 100% |
| Last Two Layers | 96.25% | 96.33% | 100% |
| All Dense Layers | 96.14% | 96.19% | 100% |
| All Layers | 96.20% | 94.17% | 100% |

Table 8: The verification authority can reliably "recover" and verify the owner watermark from a student model trained on a watermarked teacher model, regardless of the fine-tuning method used by the transfer learning. Thus, despite the fact that transfer learning removes the watermark target label ($y_W$) from the student model, the teacher's owner watermark is still embedded into the student model.

recognition. We use LISA [15] as our student dataset and follow prior work [6] in constructing the training dataset. We use two models trained on GTSRB as teacher models (a watermark-free model and a watermarked model). To create the student model, we copy all layers except last layer from the teacher model and add a final classification layer. We consider four different methods to train the student model: fine-tuning the added layers only, fine-tuning the last two dense layers, fine-tuning all dense layers and fine-tuning all layers. We train the student model for 200 epochs. More details of the training settings can be found in Appendix.

**Our watermark design allows transfer learning.** Table 7 lists the normal classification accuracy of the student models trained from our two teacher models. We see that fine-tuning more layers during transfer increases student model's normal classification accuracy. In fact, when all layers are fine-tuned, the watermarked student performs better than the one trained by a watermark-free model. Thus, our watermarked model can be used as a teacher model for transfer learning.

**Our watermark persists through transfer learning.** We now verify whether the original watermark in the teacher model can still be detected/verified in the student models. Here we consider the case where the target label $y_W$ used by our watermark's true embedding is removed by the transfer process. We show that while the absence of $y_W$ in the student model "buries" the owner watermark inside the model, one can easily "recover" and then verify the owner watermark using a transparent process.

Specifically, the verification authority first examines whether the student model contains $y_W$ (defined by the owner watermark to be verified). If not, the authority first "recovers" the owner watermark from the student model. This is done by adding $y_W$ to the student model and fine-tuning it for a few epochs using clean training data. Here the fine-tuning method is the same one used by the transfer learning[1]. The entire recovery process is transparent and deterministic, and can be audited by an honest third party.

We run the above verification process on the LISA student model generated from the watermarked teacher model. In this case, $y_W$ (a German traffic sign) is not present in the student model. We replace the last layer of the student model with a randomly initialized layer whose dimensions match those of the teacher model's final layer. This is our "recovered" teacher model. We fine tune the recovered model using the teacher's training data for 6 epochs, and run the owner watermark verification on the model. Results in Table 8 shows that the owner watermark can be fully restored and reliably verified regardless of the transfer learning techniques used to train the student model. This confirms that our proposed watermark can persist through transfer learning.

## 9 Discussion and Conclusion

We propose a new ownership watermark system for DNN models, which achieves the critical property of piracy resistance that has been missing from all existing watermark designs. Core to our watermark design is *null embedding*, a new training method that creates a strong dependency between normal classification accuracy and a given watermark when a model is initially trained. Null embeddings constrain the classification space, and cannot be replaced or added without breaking normal classification.

Limitations remain in our proposed system. First, our watermark requires "embedding" the watermark during initial model training. This leads to some (perhaps unavoidable) inconveniences. Since a watermark cannot be repeated or removed, a model owner must choose the watermark before training a model, and any updates to the watermark requires retraining the model from scratch. Second, our experimental validation has been limited by computational resources. We could not test our watermark on the largest models, *e.g.* ImageNet as a result. Our smaller 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). We are building a much larger GPU cluster to enable larger scale watermark experiments.

In ongoing work, we are exploring how null embedding might be extended to other domains like audio or text. Finally, we continue to test and evaluate our watermark implementation, with the goal of releasing a full implementation to the research community in the near future.

[1]One can determine the fine-tuning method used by the transfer learning by comparing the weights of student and teacher models and identifying the set of the layers modified by the transfer learning.

## 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] ADIWARDANA, D., LUONG, M.-T., SO, D. R., HALL, J., FIEDEL, N., THOPPILAN, R., YANG, Z., KULSHRESHTHA, A., NEMADE, G., LU, Y., ET AL. Towards a human-like open-domain chatbot. *arXiv preprint arXiv:2001.09977* (2020).

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

[4] 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).

[5] DARVISH ROUHANI, B., CHEN, H., AND KOUSHANFAR, F. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (2019), pp. 485–497.

[6] 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).

[7] 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).

[8] 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.

[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] KRIZHEVSKY, A., ET AL. Learning multiple layers of features from tiny images. Tech. rep., Citeseer, 2009.

[13] 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.

[14] 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.

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

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

[28] 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.

[29] 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).

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

[31] 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.

[32] 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.

[33] 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).

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.

### A.1 Model Architectures and Training Configurations

Tables 10, 12, 11 and 13 list the architectures of the different models used in our experiments. For all four tasks, we use convolutional network networks. We vary the number of layers, channels, and filter sizes in the models to accommodate different tasks. Table 14 describes the details of the training configurations used for each task.

### A.2 Experiments to Examine Existing Work's Vulnerability to Piracy (§4)

When examining whether [1] and [33] are vulnerable to ownership piracy attacks, we use the same four tasks that we use to evaluate our own watermark (see above). Next we describe the watermarks used by our experiments and the detailed training configuration.

**Watermark Triggers.** For [1], the original trigger set we use is the same as the trigger set used in [1]. To collect the pirate trigger set, we randomly choose 100 images of abstract art from Google Images, resize them to fit our model, and assign labels for each of them. Note that both the original and pirate trigger sets contain exactly 100 images. For [33], we use a trigger very similar to one used in their paper – the word "TEXT" written in black pixels at the bottom of an image. The pirate trigger is the word "HELLO" written in white pixels at the top of the image.

Figure 7 shows the triggers used for [33], and one example image from both the original and pirate trigger sets for [1]. For completeness, we tried several different triggers for the piracy attack on [33] and find that all are successful (> 95% pirate trigger accuracy). These are shown in Figure 8.

**Training Configurations.** To train the original watermarked models for both methods, we use the training configurations shown in Table 14. The watermark injection ratios are shown in Table 9. For all tasks, we assume the attacker only has 5k training data for `Digit`, `Face`, `Traffic`, and `Object`, which is consistent with the piracy experiments on our own watermarking system. To inject the pirate watermark, we only train the watermarked models for 10 epochs for `Digit` and `Object`. `Face` only requires 1 epoch to successfully inject the watermark. `Traffic` requires 10 epochs for [33] and 25 for [1].

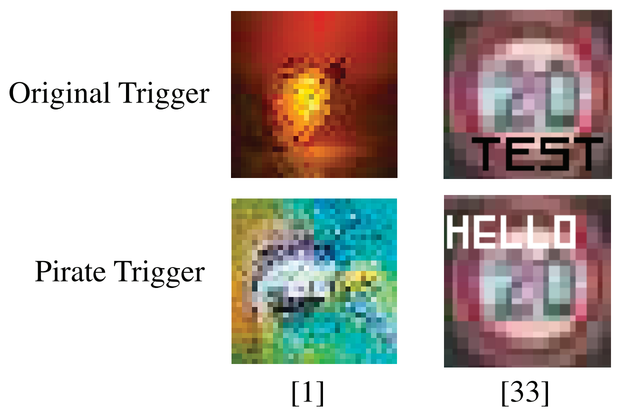

Figure 7: Examples of original and pirate triggers used to recreate [1] and [33].

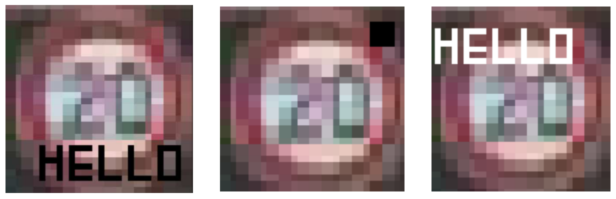

Figure 8: Additional triggers used to successfully conduct a watermark piracy attack against [33].

### A.3 Experiments for Countermeasures (§8)

We now report the details for our experiments in Section 8.

**Model Extraction Attack.** To launch the model extraction attack on `Traffic`, we create a substitute model with the same model architecture in Table 12. To train the substitute model from scratch we use the same training configurations for `Traffic` in Table 14 but with 0 inject ratio.

**Transfer Learning** The dataset we use for the student task is LISA which has 3,987 training images and 340 testing images of 17 US traffic signs. We resize all the images to (48, 48, 3) for transfer learning purpose. During the transfer learning process, we fine tune the student model for 200 epochs using student training data, using SGD optimizer with 0.01 learning rate and 0 decay.

| Task | Inject Ratio | |
|---|---|---|
| | [1] | [33] |
| `Digit` | 28.125% | 10% |
| `Traffic` | 6.25% | 10% |
| `Face` | 6.25% | 10% |
| `Object` | 6.25% | 10% |

Table 9: Injection ratio for piracy attacks on previous work.

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

Table 10: Model Architecture for `Digit`.

| Layer Index | Layer Name | Layer Type | # of Channels | Filter Size | Activation | Connected to |
|---|---|---|---|---|---|---|
| 1 | conv_1 | Conv | 20 | 4× 4 | ReLU | |
| 1 | pool_1 | MaxPool | | 2× 2 | - | conv_1 |
| 2 | conv_2 | Conv | 40 | 3× 3 | ReLU | pool_1 |
| 2 | pool_2 | MaxPool | | 2× 2 | - | conv_2 |
| 3 | conv_3 | Conv | 60 | 3× 3 | ReLU | pool_2 |
| 3 | pool_3 | MaxPool | | 2× 2 | - | conv_3 |
| 3 | fc_1 | FC | 160 | - | - | pool_3 |
| 4 | conv_4 | Conv | 80 | 2× 2 | 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 11: Model Architecture for `Face`.

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

Table 12: Model Architecture for `Traffic`.

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

Table 13: Model Architecture for `Object`.

| Tasks | Training Configuration |
|---|---|
| `Digit` | lr=0.001, decay=0, optimizer=sgd, batch_size=128, epochs=300, inject_ratio=0.5 |
| `Face` | lr=0.001, decay=1e-7, optimizer=adam, batch_size=128, epochs=10, inject_ratio=0.5 |
| `Traffic` | lr=0.02, decay=2e-5, optimizer=sgd, batch_size=128, epochs=120, inject_ratio=0.1 |
| `Object` | lr=0.05, decay=1e-4, optimizer=sgd, batch_size=128, epochs=500, inject_ratio=0.1 |

Table 14: Hyper-parameters for model training for all four tasks.