DeepSigns: An End-to-End Watermarking Framework for Protecting the Ownership of Deep Neural Networks

Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar
University of California San Diego
bita@ucsd.edu, huc044@eng.ucsd.edu, farinaz@ucsd.edu

Abstract

Deep Learning (DL) models have created a paradigm shift in our ability to comprehend raw data in various important fields, ranging from intelligence warfare and healthcare to autonomous transportation and automated manufacturing. A practical concern, in the rush to adopt DL models as a service, is protecting the models against Intellectual Property (IP) infringement. DL models are commonly built by allocating substantial computational resources that process vast amounts of proprietary training data. The resulting models are therefore considered to be an IP of the model builder and need to be protected to preserve the owner's competitive advantage.

We propose DeepSigns, the first end-to-end IP protection framework that enables developers to systematically insert digital watermarks in the pertinent DL model before distributing the model. DeepSigns is encapsulated as a high-level wrapper that can be leveraged within common deep learning frameworks including TensorFlow, PyTorch, and Theano. The libraries in DeepSigns work by dynamically learning the probability density function (pdf) of activation maps obtained in different layers of a DL model. DeepSigns uses the low probabilistic regions within a deep neural network to gradually embed the owner's signature (watermark) while minimally affecting the overall accuracy and/or training overhead. DeepSigns can demonstrably withstand various removal and transformation attacks, including model pruning, model fine-tuning, and watermark overwriting. We evaluate DeepSigns performance on a wide variety of DL architectures including Wide Residual Networks, Convolution Neural Networks, and Multi-Layer Perceptrons with MNIST, CIFAR10, and ImageNet data. Our extensive evaluations corroborate DeepSigns' effectiveness and applicability. Our highly-optimized accompanying API further facilitates training watermarked neural networks with an extra overhead as low as 2.2%.

1. Introduction

Deep Neural Networks (DNNs) and other deep learning variants have revolutionized vision, speech, and natural language processing and are being applied in many other critical fields [15, 5, 7, 24]. Training a highly accurate DNN requires: (i) Having access to a massive collection of mostly proprietary labeled data that furnishes comprehensive coverage of potential scenarios in the target application. (ii) Allocating substantial computing resources to fine-tune the underlying model topology (i.e., type and number of hidden layers), hyper-parameters (i.e., learning rate, batch size, etc.), and DNN weights to obtain the most accurate model. Given the costly process of designing/training, DNNs are typically considered to be the intellectual property of the model builder.

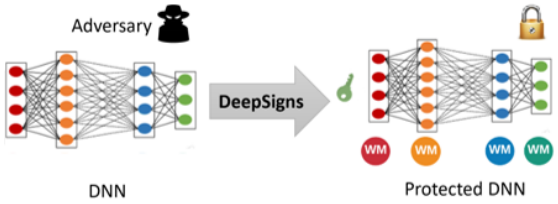


Figure 1: DeepSigns is a systematic solution to protect the intellectual property of deep neural networks.

Model protection against IP infringement is particularly important for DNNs to preserve the competitive advantage of the owner and ensure the receipt of continuous query requests by clients if the model is deployed in the cloud as a service. Embedding digital watermarks into DNNs is a key enabler for reliable technology transfer. Digital watermarks have been immensely leveraged over the past decade to protect the ownership of multimedia and video content, as well as functional artifacts such as digital integrated circuits [6, 11, 23, 3, 19]. Extension of watermarking techniques to DNNs, however, is still in its infancy to enable reliable model distribution. Moreover, adding digital watermarks further presses the already constrained memory for DNN training/execution. As such, efficient resource management to minimize the overhead of watermarking is a standing challenge.

Authors in [31, 21] propose an N-bit ($N > 1$) watermarking approach for embedding the IP information in the *static* content (i.e., weight matrices) of convolutional neural networks. Although this work provides a significant leap as the first attempt to watermark DNNs, it poses (at least) two limitations as we discuss in Section 5: (i) It incurs a bounded watermarking capacity due to the use of the static content of DNNs (weights) as opposed to using dynamic content (activations). The weights of a neural network are invariable (static) during the execution phase, regardless of the data passing through the model. The activations, however, are dynamic and both *data-* and *model-dependent*. We argue that using activations (instead

of weights) provides more flexibility for watermarking. (ii) It is not robust against attacks such as overwriting the original embedded watermark by a third party. As such, the original watermark can be removed by an adversary that is aware of the watermarking method used by the model owner.

More recent studies in [20, 1] propose 1-bit watermarking methodologies for deep learning models. These approaches are built upon model boundary modification and the use of random adversarial samples that lie near decision boundaries. Adversarial samples are known to be statistically unstable, meaning that adversarial samples crafted for a model are not necessarily misclassified by another network [8, 27]. Therefore, even though the proposed approaches in [20, 1] yield a high watermark detection rate (true positive rate), they are also too sensitive to hyper-parameter tuning and usually lead to a high false alarm rate. Note that false ownership proofs jeopardize the integrity of the proposed watermarking methodology and render the use of watermarks for IP protection ineffective.

This paper proposes DeepSigns, the first efficient resource management framework that empowers coherent integration of robust digital watermarks into DNNs. DeepSigns is devised based on an Algorithm/Hardware/Software co-design. As illustrated in Figure 1, DeepSigns inserts the watermark information in the host DNN and outputs a protected, functionality-preserved model to prevent the adversary from pirating the ownership of the model. Unlike prior works that directly embed the watermark information in the static content (weights) of the pertinent model, DeepSigns works by embedding an arbitrary N-bit ($N \geq 1$) string into the probability density function (pdf) of the activation maps in various layers of a deep neural network. Our proposed watermarking methodology is simultaneously *data-* and *model-dependent*, meaning that the watermark information is embedded in the dynamic content of the DNN and can only be triggered by passing specific input data to the model. We further provide a comprehensive set of quantitative and qualitative metrics that shall be evaluated to corroborate the effectiveness of current and pending DNN watermarking methodologies that will be proposed in future.

We provide a highly-optimized implementation of DeepSigns's watermarking methodology which can be readily used as a high-level wrapper within contemporary DL frameworks. Our solution, in turn, reduces the non-recurring engineering cost and enables model designers to incorporate specific Watermark (WM) information during the training of a neural network with minimal changes in their source code and overall training overhead. Extensive evaluation across various DNN topologies confirms DeepSigns' applicability in different settings without requiring excessive hyper-parameter tuning to avoid false alarms or accuracy drop. By introducing DeepSigns, this paper makes the following contributions:

- **Enabling effective IP protection for DNNs.** A novel watermarking methodology is introduced to encode the pdf of activation maps and effectively trace the IP ownership. DeepSigns is provably more robust against removal and transformation attacks compared to prior works.
- **Characterizing the requirements for an effective watermark embedding in the context of deep learning.** We provide a comprehensive set of metrics that enables quantitative and qualitative comparison of current and pending DNN-specific IP protection methods.
- **Devising a careful resource management and accompanying API.** A user-friendly API is devised to minimize the non-recurring engineering cost and facilitate the adoption of DeepSigns within contemporary DL frameworks including TensorFlow, Pytorch, and Theano.
- **Analysis of various DNN topologies.** Through extensive proof-of-concept evaluations, we investigate the effectiveness of the proposed framework and corroborate the necessity of such solution to protect the IP of an arbitrary DNN and establish the ownership of the model designer.

This paper opens a new axis for the growing research in secure deep learning. This work sheds light on previously unexplored impacts of IP protection on DNNs' performance. DeepSigns provides a development tool for the research community to better protect their innovative DNN designs. Our tool is open source and will be publicly available at [link omitted due to double blind review policy].

2. DeepSigns Overview

Figure 2 demonstrates the global flow of DeepSigns framework. DeepSigns consists of two main phases: watermark embedding and watermark extraction. The watermarked DNN can be employed as a service by third-party users either in a white-box or a black-box setting depending on whether the model internals are transparent to the public or not. DeepSigns is the first DNN watermarking framework that is applicable to both white-box and black-box security models.

▪ **Watermark Embedding.** DeepSigns takes the DNN architecture and the owner-specific watermark signature as its input. The WM signature is a set of arbitrary binary strings that should be generated such that each bit is independently and identically distributed (i.i.d.). DeepSigns, then, outputs a trained DNN that carries the pertinent watermark signature in selected layers along with a set of corresponding WM keys. The WM keys are later used to trigger the embedded WM information during the extraction phase. The WM embedding process is performed in two steps. First, a set of WM keys are generated as secure parameters for WM embedding. Then, the underlying DNN is trained (fine-tuned) such that the owner-specific WM signature is encoded in the pdf distribution of activation maps obtained at different DNN layers. Note that WM embedding is a one-time task performed by the owner before model distribution. Details of each step are discussed in Section 3. The trained watermarked DNN can be securely distributed by the model owner. Model distribution is a common approach in the machine learning field (e.g., the Model Zoo by Caffe Developers, and Alexa Skills by Amazon). Note that even though models are voluntarily shared, it is important

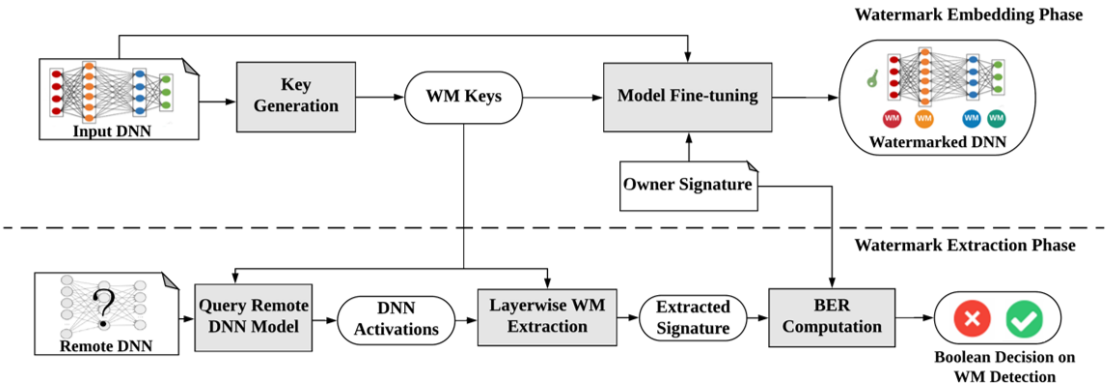


Figure 2: DeepSigns embeds the owner-specific WM signature in the pdf distribution of activation maps acquired at various DNN layers. A specific set of WM keys are generated to extract the embedded watermarks. The WM keys triggering the ingrained WM are then used for watermark extraction and detection of IP infringement.

to protect the IP and preserve copyright of the original owner.

▪ **Watermark Extraction.** To verify the IP of a remote DNN and detect potential IP infringement, the model owner first needs to query the remote DNN service with WM keys generated in the WM embedding phase and obtain the corresponding activation maps. DeepSigns then extracts the WM signature from the pdf distribution of the acquired activation maps. It next computes the Bit Error Rate (BER) between the extracted signature in each layer and the corresponding true signature. If the BER at any layer is zero, it implies that the owner's IP is deployed in the remote DNN service. Details of each WM extraction step are discussed in Section 3.

2.1. DNN Watermarking Prerequisites

There are a set of minimal requirements that should be addressed to design a robust digital watermark. Table 1 details the prerequisites for an effective DNN watermarking methodology. In addition to previously suggested requirements in [31, 20], we believe reliability and integrity are two other major factors that need to be considered when designing a practical DNN watermarking methodology. *Reliability* is important because the embedded watermark should be accurately extracted using the pertinent keys; the model owner is thereby able to detect any misuse of her model with a high probability. *Integrity* ensures that the IP infringement detection policy yields a minimal number of false alarms, meaning that there is a very low chance of falsely proving the ownership of the model used by a third party. DeepSigns satisfies all the requirements listed in Table 1 as shown in Section 4.

Potential Attack Scenarios. To validate the robustness of a potential DL watermarking approach, one should evaluate the performance of the proposed methodology against (at least) three types of contemporary attacks: (i) *Model fine-tuning*. This type of attack involves re-training of the original model to alter the model parameters and find a new local minimum while preserving the accuracy. (ii) *Model pruning*. Model pruning is a commonly used approach for efficient execution of neural networks, particularly on embedded devices. We consider model pruning as another attack approach that might affect the watermark extraction/detection. (iii) *Watermark overwriting*. A third-party user who is aware of the methodology used for DNN watermarking (but is not aware of the owner's private WM keys) may try to embed a new watermark in the model and overwrite the original one. An overwriting attack aims to insert an additional watermark in the model and render the original watermark unreadable. A watermarking methodology should be robust against fine-tuning, pruning, and overwriting for effective IP protection.

3. DeepSigns Methodology

Deep learning models possess non-convex loss surfaces with many local minima that are likely to yield an accuracy very close to another [2, 26]. DeepSigns takes advantage of this phenomenon that there is not a unique solution for modern non-convex optimization problems to embed the WM information in the pdf distribution of activation maps in a DNN. DeepSigns proposes different approaches for watermarking the intermediate layers (*Section 3.1*) and the output layer (*Section 3.2*) of a DNN model. This is due to the fact that the activation of an intermediate layer is continuous-valued while the one of the output layer is discrete-valued in classification tasks that comprise a large percentage of DL applications.

Furthermore, we analyze the computation and communication overhead of DeepSigns framework in Section 3.3. We devise an efficient memory management library to minimize the pertinent WM embedding overhead. DeepSigns accompanying library is compatible with the current popular DL solutions and provides a user-friendly API in Python that supports GPU acceleration. To illustrate the integrability of DeepSigns, we demonstrate how to use DeepSigns as a wrapper to embed/extract WM in Sections 3.1.3 and 3.2.3.

3.1. Watermarking Intermediate Layers

DeepSigns embeds the WM information within a transformation of specific activations. We consider a Gaussian Mixture Model (GMM) as the prior pdf to characterize the data distri-

Table 1: Requirements for an effective watermarking of deep neural networks.

Requirements	Description
Fidelity	Accuracy of the target neural network shall not be degraded as a result of watermark embedding.
Reliability	Watermark extraction shall yield minimal false negatives; WM shall be effectively detected using the pertinent keys.
Robustness	Embedded watermark shall be resilient against model modifications such as pruning, fine-tuning, or WM overwriting.
Integrity	Watermark extraction shall yield minimal false alarms (a.k.a., false positives); the watermarked model should be uniquely identified using the pertinent keys.
Capacity	Watermarking methodology shall be capable of embedding a large amount of information in the target DNN.
Efficiency	Communication and computational overhead of watermark embedding and extraction shall be negligible.
Security	The watermark shall be secure against brute-force attacks and leave no tangible footprints in the target neural network; thus, an unauthorized party cannot detect/remove the presence of a watermark.

bution at a given layer where the WM shall be inserted. The rationale behind this choice is the observation that GMM provides a reasonable approximation of the activation distribution obtained in hidden layers [12, 22, 18].[1] In the following, we discuss the details of WM embedding and extraction procedures for intermediate (hidden) layers.

3.1.1. Watermark Embedding (Hidden Layers)

Algorithm 1 outlines the process of WM embedding in a hidden layer. It consists of two main steps:

❶ **Key Generation.** For a given hidden layer l, DeepSigns first selects one (or more) random indices between 1 and S with no replacement: Each index corresponds to one of the Gaussian distributions in the target mixture model that contains a total of S Gaussians. In our experiments, we set the value S equal to the number of classes in the target application. The mean values of the selected distributions are next used to carry the WM signature. DeepSigns then decides on a subset of the input training data belonging to the selected Gaussian classes (X^{key}). This subset is later used by the model owner to trigger the embedded WM signature within a hidden layer as discussed in Section 3.1.2. In our experiments, we use a subset of 1% of the training data for this purpose.

DeepSigns also generates a projection matrix A to encrypt the selected centers into the binary space. This projection is critical to accurately measure the difference between the embedded WM and the owner-defined binary signature during training. The projection is performed as the following:

$$\begin{aligned} G_{\sigma}^{s\times N} &= Sigmoid\,(\mu_l^{s\times M} \cdot A^{M\times N}),\\ \tilde{b}^{s\times N} &= Hard_Thresholding\,(G_{\sigma}^{s\times N},\, 0.5). \end{aligned} \tag{1}$$

Here, M is the size of the feature space in the selected hidden layer, s is the number of Gaussian distributions chosen to carry the WM information, and N indicates the desired length of the watermark embedded at the mean value of s selected Gaussian distribution ($\mu_l^{s\times M}$). In our experiments, we use a standard normal distribution $\mathcal{N}(0,1)$ to generate the WM projection matrix (A). Using i.i.d. samples drawn from a normal distribution ensures that each bit of the binary string is embedded into all the features associated with the selected centers. The σ notation in Eq. (1) is used as a subscript to indicate the deployment of the Sigmoid function. The output of Sigmoid has a value between 0 and 1. Given the random nature of the binary string, we decide to set the threshold in Eq. (1) to 0.5, which is the expected value of Sigmoid function. The *Hard_Thresholding* function in Eq. (1) maps the values in G_{σ} that are greater than 0.5 to ones and the values less than 0.5 to zeros. The WM keys comprise the selected Gaussian classes s, trigger keys X^{key} and projection matrix A.

❷ **Model Fine-tuning.** To effectively encode the WM information, two additional constraints need to be considered during DNN training: (i) Selected activations shall be isolated from other activations. (ii) The distance between the owner-specific WM signature and the transformation of isolated activations shall be minimal. We design and incorporate two specific loss functions to address each of these two constraints ($loss_1$ and $loss_2$ in Step 2 of Algorithm 1).

To address the activation isolation constraint, we design an additive loss term that penalizes the activation distribution when activations are entangled and hard to separate. Adhering to our GMM assumption, we add the following term to the

Algorithm 1 Watermark embedding for one hidden layer.

INPUT: Topology of the unmarked DNN ($\mathscr{T}$); Training data ($\{X^{train}, Y^{train}\}$); Owner-specific watermark signature b; Total number of Gaussian classes (S); Length of watermark vector for each selected distribution (N); Dimensionality of the activation map in the embedded layer (M); and Embedding strength hyper-parameters (λ_1, λ_2).

OUTPUT: Watermarked DNN ($\mathscr{T}^*$); WM keys.

❶ Key Generation:
$T \leftarrow Select_Gaussian_Classes\,([1,\, S])$
$X^{key} \leftarrow Subset_Training_Data\,(T,\, \{X^{train}, Y^{train}\})$
$A^{M\times N} \leftarrow Generate_Secret_Matrix\,(M,\, N)$

❷ Model Fine-tuning: Two additive regularization loss functions are incorporated to train the DNN:

$$L = \underbrace{cross_entropy}_{loss_0} + \lambda_1 loss_1 + \lambda_2 loss_2.$$

Return: Marked DNN $\mathscr{T}^*$, WM keys $(s, X^{key}, A^{M\times N})$.

[1] DeepSigns is rather generic and is not restricted to the GMM distribution; GMM can be replaced with other prior distributions based on the application.

cross-entropy loss function used for DNN training:

$$\lambda_1 \underbrace{(\Sigma_{i\in T}\|\mu_l^i - f_l^i(x,\theta)\|_2^2 \; - \; \Sigma_{i\in T, j\notin T}\|\mu_l^i - \mu_l^j\|_2^2}_{\textbf{loss}_1}). \qquad (2)$$

Here, λ_1 is a trade-off hyper-parameter that specifies the contribution of the additive loss term, θ is the DNN parameters (weights), $f_l^i(x,\theta)$ is the activation map corresponding to the input sample x belonging to class i at the l^{th} layer, T is the set of s target Gaussian classes selected to carry the WM information, and μ_l^i denotes the mean value of the i^{th} Gaussian distribution at layer l. The additive loss function ($loss_1$) aims to minimize the spreading (variance) of each GMM class used for watermarking (the first term in $loss_1$) while maximizing the distance between the activation centers belonging to different Gaussian classes (the second term in $loss_1$). This loss function, in turn, helps to augment data features so that they better fit a GMM distribution. The mean values μ_l^i and intermediate activations $f_l^i(x,\theta)$ in Eq. (2) are *trainable variables* that are iteratively fine-tuned using back-propagation.

To ensure the transformed selected Gaussian centers are as close to the desired WM information as possible, we design the second additive loss term that characterizes the distance between the owner-defined signature and the embedded watermark (see Eq.(3)). As such, DeepSigns adds the following term to the overall loss function used during DNN training:

$$-\lambda_2 \underbrace{\sum_{j=1}^{N}\sum_{k=1}^{s}(b^{kj}\ln(G_\sigma^{kj}) + (1-b^{kj})\ln(1-G_\sigma^{kj}))}_{\textbf{loss}_2}. \qquad (3)$$

Here, the variable λ_2 is a hyper-parameter that determines the contribution of $loss_2$ during DNN training. The $loss_2$ function resembles a binary cross-entropy loss where the true bit b^{kj} is determined by the owner-defined WM signature and the prediction probability G_σ^{kj} is the Sigmoid of the projected Gaussian centers as outlined in Eq. (1). The process of computing the vector G_σ is differentiable. Thereby, for a selected set of projection matrix (A) and binary WM signature (b), the selected centers (Gaussian mean values) can be adjusted via back-propagation such that the Hamming distance between the binarized projected center $\tilde{b}$ and the actual WM signature b is minimized. In our experiments, we set λ_1 and λ_2 to 0.01.

3.1.2. Watermark Extraction (Hidden Layers)

As the inverse process of watermark embedding, watermark extraction is implemented using the two particular constraints we design in Section 3.1.1. To extract watermark information from intermediate (hidden) layers, the model owner must follow three main steps. **(i)** Acquiring activation maps corresponding to the selected trigger keys X^{key} by submitting a set of queries to the remote DL service provider. **(ii)** Computing the statistical mean value of the activation maps obtained in Step I. The acquired mean values are adopted as an approximation of the Gaussian centers that are supposed to carry the watermark information. The computed mean values together with the owner's private projection matrix A are used to extract the pertinent WM following the procedure in Eq. (1). **(iii)** Measuring the bit error rate between the owner's signature and the extracted WM from Step II. Note that if the watermarked DNN in question is not deployed in the remote service, a random WM is extracted which yields a very high BER.

3.1.3. DeepSigns Memory Management and Wrapper

DeepSigns minimizes the required data movement to ensure maximal data reuse and a minimal overhead caused by watermark embedding. To do so, we integrate the computation of additive loss terms to the DNN tensor graph so that the gradients with respect to the GMM centers are computed during regular back-propagation and all computation for WM embedding is performed homogeneously on GPU. Modeling the watermarking graph separately significantly slows than the DNN training process since the activation maps need to be completely dumped from the original DNN graph during the forward pass to compute the WM loss and update the parameters of the WM graph. This approach, in turn, further presses the already constrained memory. Our homogeneous solution reuses the activation values within the original graph with minimal memory overhead.

DeepSigns library provides a customized activity regularizer *WM_activity_regularizer* that computes $loss_1$ and $loss_2$ and returns the total regularized loss value described in Algorithm 1. To extract the WM from the embedded layers, our accompanying library is equipped with functions called *get_activation* and *extract_WM_from_activations* to implement the process outlined in Section 3.1.2. Figure 3 shows the prototype of functions used for watermark embedding/extraction in the intermediate layers. The notations are consistent with the definitions in Section 3.1 and 3.1.2. DeepSigns' customized library supports acceleration on GPU platforms. Our provided wrapper can be readily integrated within well-known DL frameworks including TensorFlow, Pytorch, and Theano.

```
import DeepSigns
from DeepSigns import subsample_training_data
from DeepSigns import WM_activity_regularizer
from DeepSigns import get_activations
from DeepSigns import extract_WM_from_activations
from DeepSigns import compute_BER
from utils import create_marked_model

## create WM trigger keys
X^key = subsample_training_data(T, {X^train , Y^train })
## instantiate customized WM activity regularizer
WM_reg = WM_activity_regularizer(λ1, λ2, b, A)
model = create_marked_model(WM_reg , model topology)
## embed WM by standard training of the marked model
model.fit(X^train , Y^train)

## extract WM from the activation maps and compute BER
μ_l^{s*M} = get_activations(model, X^key, l, T)
b̃ = extract_WM_from_activations(μ_l^{s*M}, A)
BER = compute_BER(b̃, b)
```

Figure 3: DeepSigns library usage and resource management for WM embedding and extracting in hidden layers.

3.2. Watermarking Output Layer

The final prediction of a DNN shall closely match the ground-truth labels to have the maximum possible accuracy. As such, instead of directly regularizing activations of the output layer, we decide to adjust the tails of decision boundaries to add a statistical bias as a 1-bit watermark. The WM key is designed as a set of random (key image, key label) pairs that are used to retrain the DNN. To verify the ownership of a remote DNN, we devise a statistical hypothesis testing method that compares the prediction of the queried DNN with WM key labels. A high consistency implies the existence of the owner's watermark.

Watermarking the output layer is a post-processing step that is performed once the DNN model is converged or the intermediate layers are watermarked as discussed in Section 3.1. In the following, we detail the workflow of WM embedding and extraction operating on the output layer. We then provide a corresponding example using DeepSigns' library.

3.2.1. Watermark Embedding (Output Layer)

The WM keys corresponding to the output layer should be carefully crafted such that they reside in low-density regions of the target DNN in order to ensure minimal accuracy drop. To do so, DeepSigns profiles the pdf distribution of different layers in the underlying DNN. The acquired pdf, in turn, gives us an insight into both the regions that are thoroughly occupied by the training data and the regions that are only covered by a few inputs, which we refer to as rarely explored regions. Figure 4 illustrates a simple example of two clustered activation distributions spreading in a two-dimensional subspace. The procedure of WM embedding in the output layer is summarized in Algorithm 2. In the following, we explicitly discuss each of the steps outlined in Algorithm 2.

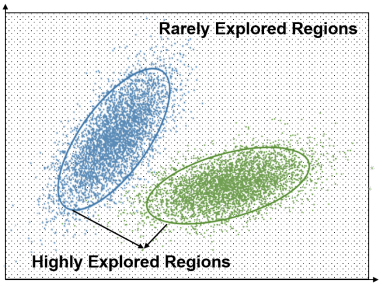


Figure 4: Due to the high dimensionality of DNNs and limited access to labeled data (the blue and green dots in the figure), there are regions within the DNN model that are rarely explored. DeepSigns exploits this mainly unused regions for WM embedding while minimally affecting the accuracy.

1 Key Generation 1. DeepSigns generates a set of K unique random input samples to be used as the watermarking keys in step 2. Each random sample is passed through the pre-trained neural network to make sure its intermediate activation lies within the rarely explored regions characterized by the learned pdf. If the number of data activations within an ε-ball of the activation corresponding to the random sample is fewer than a threshold, we accept that sample belongs to the rarely explored regions. Otherwise, a new random sample is generated to replace the previous sample. A corresponding random ground-truth vector is generated and assigned to each input key (e.g., in a classification task, each random input is associated with a randomly selected label).

We set the initial key size to be larger than the owner's desired value $K^{'} > K$ and generate the input keys accordingly. The target model is then fine-tuned (Step 2) using a mixture of the generated keys and a subset of original training data. After fine-tuning, only the keys that are simultaneously correctly classified by the marked model and incorrectly predicted by the unmarked model are appropriate candidates that satisfy both a high detection rate and a low false positive. In our experiments, we set $K^{'} = 20 \times K$ where K is the desired key length selected by the model owner.

2 Model Fine-tuning. The pre-trained DNN is fine-tuned with the selected random keys in Step 1. The model shall be retrained such that the neural network has exact predictions (e.g., an accuracy greater than 99%) for chosen key samples. In our experiments, we use the same optimizer setting used for training the original neural network, except that the learning rate is reduced by a factor of 10 to prevent accuracy drop in the prediction of legitimate input data. Note that the model is already converged to a local minimum.

3 Key Generation 2. Once the model is fine-tuned in step 2, we first find out the indices of initial WM keys that are correctly classified by the watermarked model. Next, we identify the indices of WM keys that are not classified correctly by the original DNN before fine-tuning in Step 2. The common keys between these two sets are proper candidates to trigger

Algorithm 1 Watermark embedding for DL output layer.

INPUT: **Topology of the partially-marked or unmarked DNN** $(\mathscr{T})$ **and its pdf distribution** (pdf)**; Training data** $(\{X^{train}, Y^{train}\})$**; Key size** (K)**.**

OUTPUT: **Watermarked DNN** $(\mathscr{T}^*)$**; Crafted WM keys** $(\{X^{key}, Y^{key}\})$**.**

1 Key Generation 1:
Set the initial key size $K^{'} > K$ (e.g., $K^{'} = 20 \times K$).
$\{X^{key'}, Y^{key'}\} \leftarrow Generate_Key_Pairs(K^{'}, pdf)$

2 Model Fine-tuning:
$\mathscr{T}^* \leftarrow Train(\mathscr{T}, \{X^{key'}, Y^{key'}\}, \{X^{train}, Y^{train}\})$

3 Key Generation 2:
$Y^{pred'}_{\mathscr{T}^*} \leftarrow Predict(\mathscr{T}^*, X^{key'})$
$Y^{pred'}_{\mathscr{T}} \leftarrow Predict(\mathscr{T}, X^{key'})$
$I_{\mathscr{T}^*} \leftarrow Find_Match_Index\ (Y^{key'}, Y^{pred'}_{\mathscr{T}^*})$
$I_{\mathscr{T}} \leftarrow Find_Mismatch_Index\ (Y^{key'}, Y^{pred'}_{\mathscr{T}})$
$I^{key'} \leftarrow Find_Intersection\ (I_{\mathscr{T}}, I_{\mathscr{T}^*})$
$\{X^{key}, Y^{key}\} \leftarrow Select\ (\{X^{key'}, Y^{key'}\}, I^{key'}, K)$

Return: Marked DNN $\mathscr{T}^*$; WM key $\{X^{key}, Y^{key}\}$.

the embedded WM. A random subset of candidate WM keys is then selected according to the key size (K) defined by the model owner. In the global flow (Figure 2), we merge the two key generation steps into one module for simplicity.

3.2.2. Watermark Extraction (Output Layer)

To verify the presence of a watermark in the output layer, DeepSigns performs statistical hypothesis testing on the responses obtained from the remote DNN service. To do so, DeepSigns undergoes three main steps: **(i)** Submitting queries to the remote DNN service provider and acquiring the output labels corresponding to the randomly selected WM keys (X^{key}) as discussed in Section 3.2.1. **(ii)** Computing the number of mismatches between model predictions and WM ground-truth labels. **(iii)** Thresholding the number of mismatches to derive the final decision. If the number of mismatches is less than the threshold, it means the model used by the remote service provider possesses a high similarity to the watermarked DNN.

When the two models are exact duplicates, the number of mismatches will be zero. In practice, the target DNN might be slightly modified by third-party users in both malicious or non-malicious ways. Examples of such modifications are model fine-tuning, pruning, or WM overwriting. As such, the threshold used for WM detection should be greater than zero to withstand DNN modifications. When queried by a random key image, the prediction of the model has probabilities $P(y^{pred} = j) = p_j$ for which $\sum_{j=1}^{C} p_j = 1$ and C is the number of classes in the pertinent application. Since the corresponding key labels are uniformly randomly generated, the probability that the key sample is correctly classified per generated key label is:

$$\begin{aligned} P(y^{pred} = y^{key}) &= \sum_{j=1}^{C} P(y^{pred} = j, y^{key} = j) \\ &= \sum_{j=1}^{C} P(y^{pred} = j) \times \sum_{j=1}^{C} P(y^{key} = j) \\ &= \frac{1}{C} \times \sum_{j=1}^{C} P(y^{pred} = j) = \frac{1}{C}. \end{aligned} \tag{4}$$

due to the independence between $P(y^{pred})$ and $P(y^{key})$. Note that Eq. (4) also holds when the class sizes are unbalanced. Therefore, the probability of an arbitrary DNN to make at least n_k correct decision per owner's private keys is:

$$P(N_k > n_k | \mathscr{O}) = 1 - \sum_{k=0}^{n_k} \binom{K}{k} (\tfrac{1}{C})^{K-k} (1 - \tfrac{1}{C})^{k}. \tag{5}$$

Here $\mathscr{O}$ is the DNN oracle used in the remote service, N_k is a random variable indicating the number of matched predictions of the two models compared against one another, K is the input key length according to Section 3.2.1. The decision for WM detection in the output layer is made by comparing $P(N_k > n_k | \mathscr{O})$ with an owner-specified probability threshold (p). In our experiments, we use a decision threshold of 0.999.

3.2.3. DeepSigns Wrapper and Memory Management

Figure 5 illustrates how to use DeepSigns library for WM embedding and detection in the output layer. DeepSigns automatically designs a set of robust WM key pairs (X^{key}, Y^{key}) for the model owner using the function *key_generation*. The generated WM keys are embedded in the target DNN by fine-tuning. The existence of the WM is determined from the response of the queried model to the key set. Note that DeepSigns can provide various levels of security by setting the hyper-parameters K(key length) and decision policy threshold. A larger key-length, in turn, induces a higher overhead (see Section 3.3). The model owner can explore the trade-off between security and overhead by adopting different hyper-parameters.

DeepSigns wrapper provides a custom layer working for efficient resource management during the identification of the rarely explored regions (Step 1 in Algorithm 2). To do so, we first apply Principal Component Analysis (PCA) on the activation maps acquired by passing the training data through the converged DNN. The computed Eigenvectors are then used to transform the high dimensional activation maps into a lower dimensional subspace. We encode the PCA transformation as a dense layer inserted after the second-to-last layer of the original DNN graph so that the data projection is performed with minimal data movement. The weights of the new dense layer are obtained from the Eigenvectors of the PCA of the pertinent activations. For each randomly generated sample, the density of the activations within an ε Euclidean distance of that sample is then computed. If this density is greater than a threshold that sample will be rejected.

```
import DeepSigns
from DeepSigns import key_generation
from DeepSigns import count_response_mismatch
from DeepSigns import compute_mismatch_threshold
from utils import create_model

## generate WM key pairs
model = create_model(model topology)
model.load_weights('baseline_weights')
(X^key, Y^key) = key_generation(model, K, X^train, pdf)
X^retrain = np.vstack(X^key, X^train)
Y^retrain = np.vstack(Y^key, Y^train)
## embed WM by finetuning the model with the WM key
model.fit(X^retrain, Y^retrain)

## query model with key set to detect WM
Y^pred = model.predict(X^key)
m = count_response_mismatch(Y^pred , Y^key)
θ = compute_mismatch_threshold(p, K, C)
WM_detected = 1 if m < θ else 0
```

Figure 5: Using DeepSigns library for WM embedding and extraction in the output layer.

3.3. DeepSigns Watermark Extraction Overhead

Here, we analyze the computation and communication overhead of WM extraction. The WM embedding is a one-time offline process that incurs a negligible overhead. We empirically discuss the WM embedding overhead in Section 4.

Watermarking Intermediate Layers. From the viewpoint of remote DNN service provider, the computation cost is equivalent to the cost of one forward pass in the DNN model with

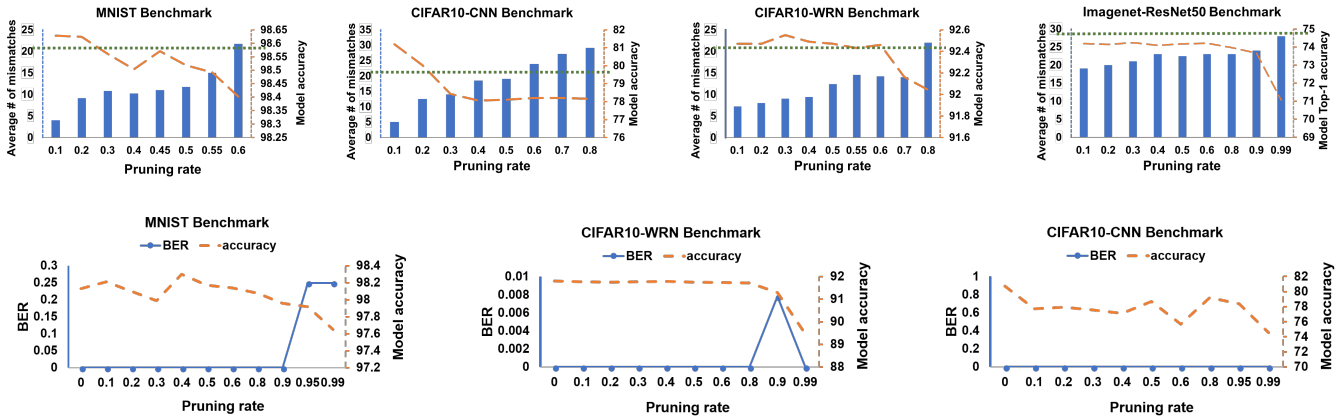


Figure 6: Robustness against parameter pruning. The experiments in the first row are related to watermarking the output layer. The horizontal green dotted line is the mismatch threshold obtained from Eq. (5). The orange dashed lines show the final accuracy for each pruning rate. Experiments in the second row correspond to watermarking the second-to-last layer.

Table 2: Benchmark neural network architectures. Here, $64C3(1)$ indicates a convolutional layer with 64 output channels and 3×3 filters applied with a stride of 1, $MP2(1)$ denotes a max-pooling layer over regions of size 2×2 and stride of 1, and $512FC$ is a fully-connected layer with 512 output neurons. ReLU is used as the activation function in all benchmarks.

Dataset	Baseline Accuracy	Accuracy of Marked Model		DL Model Type	DL Model Architecture
MNIST	98.54%	K = 20	N = 4	MLP	784-512FC-512FC-10FC
		98.59%	98.13%		
CIFAR10	78.47%	K = 20	N = 4	CNN	3*32*32-32C3(1)-32C3(1)-MP2(1) -64C3(1)-64C3(1)-MP2(1)-512FC-10FC
		81.46%	80.70%		
CIFAR10	91.42%	K = 20	N = 128	WideResNet	Please refer to [32].
		91.48%	92.02%		
ImageNet	74.72%	K = 20	–	ResNet50	Please refer to [4].
		74.21%	–		

no extra overhead. From the model owner's viewpoint, the computation cost is divided into two terms. The first term is proportional to $\mathcal{O}(M)$ to compute the statistical mean in Step 2 outlined in Section 3.1.2. Here, M denotes the feature space size in the target hidden layer. The second term corresponds to the computation of matrix multiplication in Eq. (1), which incurs a cost of $\mathcal{O}(MN)$. The communication cost is equivalent to the input key length multiplied by input feature size plus the size of intermediate layers (M) to submit the pertinent queries and obtain the intermediate activations, respectively.

Watermarking Output Layer. For the remote DNN service provider, the computation cost is equal to the cost of one forward pass through the underlying DNN. For the model owner, the computation cost is the cost of performing a simple counting to measure the number of mismatches between the responses of the remote service provider and the WM key labels. In this case, the communication cost is equal to the key length multiplied by the sum of the input feature size and one to submit the queries and read back the predicted labels.

4. Evaluations

We evaluate the performance of DeepSigns framework on various datasets including MNIST [16], CIFAR10 [14] and ImageNet [4], with four different neural network architectures. Table 2 summarizes DNN topologies used in each benchmark. In Table 2, K denotes the key size for watermarking the output layer and N is the length of the owner-specific WM signature used for watermarking the hidden layers. In our experiments, we use the second-to-last layer or the output layer for watermarking. DeepSigns library is generic and also supports WM embedding in multiple layers if larger capacity is desired. In the rest of this section, we explicitly evaluate DeepSigns' performance with respect to each requirement listed in Table 1. As empirically demonstrated, DeepSigns is effective and applicable across various datasets and DNN architectures.

4.1. Fidelity

DeepSigns preserves the DNN overall accuracy after watermark embedding. The accuracy of the target neural network shall not be degraded after embedding the WM information. Table 2 summarizes the baseline DNN accuracy (Column 2) and the accuracy of marked models (Column 3 and 4) after WM embedding. As demonstrated, DeepSigns respects the fidelity requirement by simultaneously optimizing for the accuracy of the underlying model (e.g., cross-entropy loss function), as well as the additive WM-specific loss functions as discussed in Section 3. In some cases (e.g. WideResNet benchmark), we even observe a slight accuracy improvement compared to the baseline. This improvement is mainly due to the fact that the additive loss functions outlined in Eq. (2) and (3) act as a form of a regularizer during DNN training. Regularization, in turn, helps the model to mitigate over-fitting by inducing a small amount of noise to DNNs [7].

Table 3: DeepSigns is robust against model fine-tuning attack. The reported BER and the detection rate value are averaged over 10 different runs. A value of 1 in the last row of the table indicates that the embedded WM is successfully detected, whereas a value of 0 indicates a false negative. For fine-tuning attacks, the WM-specific loss terms proposed in Section 3 are removed from the loss function and the model is retrained using the final learning rate of the original DL model. After fine-tuning, the DL model will converge to another local minimum that is not necessarily a better one (in terms of accuracy) for some benchmarks.

Metrics	Intermediate Layer Watermarking									Output Layer Watermarking										
	MNIST-MLP			CIFAR10-CNN			CIFAR10-WRN			MNIST-MLP			CIFAR10-CNN			CIFAR10-WRN			Imagenet-ResNet50	
Number of epochs	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200	10	20
Accuracy	98.21	98.20	98.18	77.47	78.41	78.89	91.79	91.74	91.8	98.57	98.57	98.59	81.69	81.82	81.92	92.02	92.08	92.05	74.06	74.14
BER	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-
Detection success	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

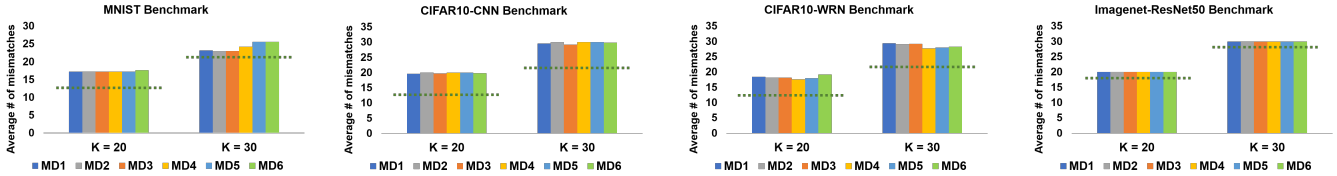


Figure 7: Integrity analysis of different benchmarks. The green dotted horizontal lines indicate the detection threshold for various WM lengths. The first three models (MD 1-3) are neural networks with the same topology but different parameters compared with the watermarked model. The last three models (MD 4-6) are neural networks with different topologies ([29], [17], [32]).

4.2. Reliability and Robustness

DeepSigns enables robust DNN watermarking and reliably extracts the embedded WM for ownership verification. We evaluate the robustness of DeepSigns against three state-of-the-art removal attacks as discussed in Section 2.1. These attacks include parameter pruning [10, 9, 25], model fine-tuning [28, 30], and watermark overwriting [31, 13].

▪ **Parameter Pruning.** We use the pruning approach proposed in [10] to sparsify the weights in the target watermarked DNN. To prune a specific layer, we first set $\alpha\%$ of the parameters that possess the smallest weight values to zero. The model is then sparsely fine-tuned using cross-entropy loss function to compensate for the accuracy drop caused by pruning.

Figure 6 demonstrates the impact of pruning on WM extraction/detection in the output and hidden layers. The length of the watermark signature and the key size used in each benchmark are listed in Table 2. As shown, DeepSigns can tolerate up to 90% parameter pruning for MNIST benchmark, and up to 99% parameter pruning for the CIFAR10, and ImageNet benchmarks. As illustrated in Figure 6, in cases where DNN pruning yields a substantial BER value, the sparse model suffers from a large accuracy loss. As such, one cannot remove DeepSigns' embedded watermark by excessive pruning and attain a comparable accuracy with the baseline.

▪ **Model Fine-tuning.** Fine-tuning is another form of transformation attack that a third-party user might use to remove the WM information. To perform this type of attack, one needs to retrain the target model using the original training data with the conventional cross-entropy loss function (excluding $loss_1$ and $loss_2$). Table 3 summarizes the impact of fine-tuning on the watermark detection rate across all benchmarks.

There is a trade-off between model accuracy and the success rate of watermark removal. If a third party tries to disrupt the pdf of activation maps that carry the WM information by fine-tuning the underlying DNN with a high learning rate, she will face a large degradation in DNN accuracy. We use the same learning rate as the one in the final stage of DL training to perform model fine-tuning attack. As shown in Table 3, DeepSigns can successfully detect the embedded WM even after fine-tuning the DNN for various number of epochs. We set the number of fine-tuning epochs to 10 and 20 epochs for ImageNet benchmark and 50, 100, 200 epochs for other benchmarks. The reason for this selection is that training the ImageNet benchmark from scratch takes 90 epochs whereas other benchmarks take around 300 epochs to be trained.

▪ **Watermark Overwriting.** Assuming the attacker is aware of the watermarking methodology, she may attempt to corrupt the original watermark by embedding a new WM. In practice, the attacker does not have any knowledge about the location of the watermarked layers. In our experiments, we consider the worst-case scenario in which the attacker knows where the WM is embedded but does not know the WM key. To perform the overwriting attack, the attacker follows the procedure in Section 3 to embed a new WM signature with her new keys.

Table 4 summarizes the results of WM overwriting for all four benchmarks in which the output layer is watermarked. As shown, DeepSigns is robust against the overwriting attack and can successfully detect the original embedded WM in the overwritten DNN. The decision thresholds shown in Table 4 for different key lengths are computed based on Eq. (5) as discussed in Section 3.2.2. A BER of zero is also observed in the overwritten DNNs where the WM is embedded in the second-to-last layer. This further confirms the DeepSigns' reliability and robustness against malicious attacks.

4.3. Integrity

DeepSigns avoids claiming the ownership of unmarked DNNs and yields low false positive rates. Let us assume the third-party user does not share her model internals and only returns the output prediction for each submitted query. In

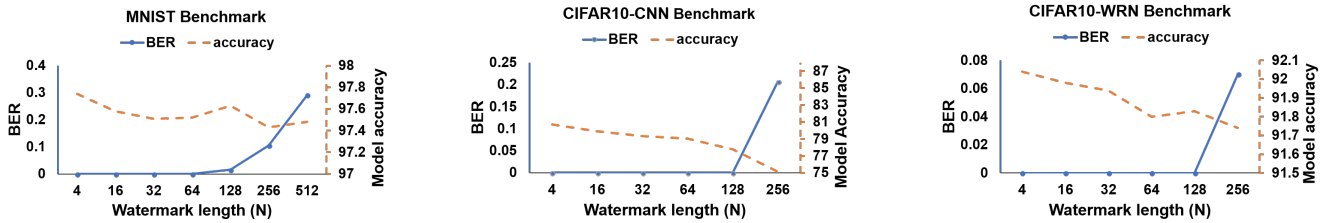


Figure 8: There is a trade-off between the length of the WM signature (capacity) and the bit error rate of watermark extraction. As the number of the embedded bits (N) increases, the test accuracy of the watermarked model decreases and the BER of WM extraction increases. This trend indicates that embedding excessive amount of WM information impairs fidelity and reliability.

Table 4: DeepSigns is robust against overwriting attack. The reported number of mismatches is the average value of 10 runs for the same model using different WM key sets.

	Average # of mismatches		Decision threshold		Detection success
	K = 20	K = 30	K = 20	K = 30	
MNIST	8.3	15.4	13	21	**1**
CIFAR10-CNN	9.2	16.7	13	21	**1**
CIFAR10-WRN	8.5	10.2	13	21	**1**
Imagenet-ResNet50	10.5	18.5	19	29	**1**

this case, it is critical to incur the minimal number of false alarms in WM extraction. Figure 7 illustrates DeepSigns integrity-related performance. In this experiment, six different unmarked models (with the same and different architectures) are queried by DeepSigns. As corroborated, DeepSigns satisfies the integrity criterion and has no false positives across different benchmarks, which means the ownership of unmarked DNNs will not be falsely proved. We use the same set of hyper-parameters (e.g., detection policy threshold) across all the benchmarks with no particular fine-tuning.

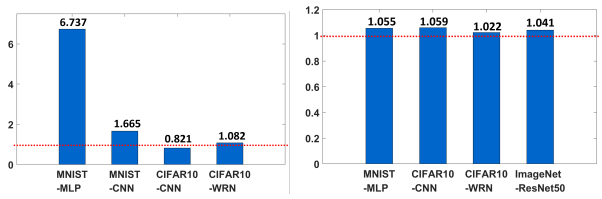
4.4. Capacity

DeepSigns has a high watermarking capacity. We embed WM signatures with different lengths in a single DNN layer to assess the capacity of DeepSigns watermarking methodology. As shown in Figure 8, DeepSigns allows up to 64 bits WM embedding for MNIST and up to 128 bits WM embedding in the second-to-last layer of CIFAR10-CNN, and CIFAR10-WRN benchmarks. Note that there is a trade-off between the capacity and accuracy which can be used by the IP owner to embed a larger watermark in her DNN model if desired. For IP protection purposes, capacity is not an impediment criterion as long as there is sufficient capacity to contain the necessary WM information. Nevertheless, we include this property in Table 1 to have a comprehensive list of requirements.

4.5. Efficiency

DeepSigns incurs a negligible overhead. The WM extraction overhead is discussed in Section 3.3. Here, we analyze the overhead incurred by the WM embedding phase. The computation overhead to embed a watermark using DeepSigns is a function of the DNN topology (i.e., the number of parameters/weights in the pertinent DNN), the key length (K), and the length of watermark signature N. DeepSigns has no communication overhead for WM embedding since the embedding process is performed locally by the model owner.

To quantify the computation overhead for embedding a watermark, we measure the normalized training time of the baseline model without WM and the marked model with WM to reach the same accuracy level. The results are shown in Figure 9, where the x-axis denotes various benchmarks and the y-axis denotes the runtime ratio of training a DNN model with/without a WM. As shown, DeepSigns incurs a reasonable overhead for WM embedding (normalized runtime overhead around 1), suggesting a high efficiency. The overhead of embedding a watermark in the hidden layer of MNIST-MLP benchmark is higher than others since this benchmark is so compact with a relatively small rarely-explored region. The low-dimensionality of this model, in turn, makes it harder to reach the same accuracy while adding noise to the system by incorporating the WM-specific regularization.



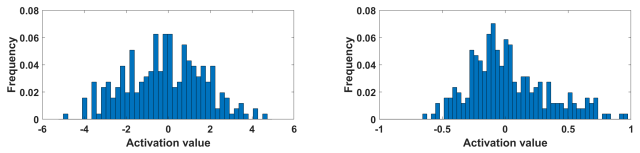
(a) (b)

Figure 9: Normalized WM embedding runtime overhead in (a) the second-to-last layer and (b) the output layer. The desired runtime ratio is denoted by the red dashed line in the figure.

4.6. Security

DeepSigns leaves an imperceptible footprint in the watermarked DNN and is secure against brute-force attacks. As mentioned in Table 1, embedding a watermark should not leave noticeable changes in the pdf distribution spanned by the target DNN. DeepSigns satisfies the security requirement by preserving the intrinsic distribution of weights/activations. Figure 10 shows the distribution of activations in WM embedded layer of a marked DNN and the ones in the same layer of the unmarked DNN for CIFAR10-WRN benchmark. The range of activations is not deterministic in different models and cannot be used by malicious users to detect WM existence.

DeepSigns is secure against brute-force attacks. If the WM is embedded in the output layer, the searching space of an adversary to find the exact WM keys designed by the model owner is $\mathcal{O}(d^{I\times K})$ where d is the size of input data, I is the number of possible elements in the input domain (e.g., 256



(a) (b)

Figure 10: Distribution of the activation maps for (a) marked and (b) unmarked models. DeepSigns preserves the intrinsic distribution while securely embedding the WM information.

for image data as each pixel can get an integer value in the range of $[0-255]$), and K is the WM key size. Note that each WM key is generated based on i.i.d. random distribution. If the WM is embedded in hidden layers, the search space for the attacker to find WM keys is $\mathcal{O}(\prod_{l\in L}\binom{S}{s_l}R^{M_l\times N_l})$. Here, L is the set of hidden layers used for watermarking, s_l, M_l and N_l denote the number of WM-related Gaussian centers, the dimension of activation maps, and the length of the WM signature in the l^{th} layer, respectively. R is the number of values in the domain used for creating projection matrix A ($A \in \mathbb{R}$ so R is infinity).

5. Comparison With Prior Works

Figure 11 compares the general capabilities of existing DNN watermarking frameworks. We provide the first automated resource management tool and the accompanying API for efficient DNN watermarking. Unlike prior works, DeepSigns uses dynamic statistics of DNN models for watermark embedding by encoding the WM information in the pdf distribution of activation maps. Our dynamic watermarking approach is significantly more robust against potential attacks compared with the prior art in which static weights are explored for watermarking (Section 5.1). In the rest of this section, we explicitly compare DeepSigns performance against three state-of-the-art DNN watermarking frameworks existing in the literature.

5.1. Intermediate Layer Watermarking

The works in [31, 21] encode the WM information in the weights of convolution layers, as opposed to the activation maps proposed by DeepSigns. As shown in [31], watermarking the weights is not robust against overwriting attacks. Table 5 provides a side-by-side robustness comparison between our approach and these prior works for different dimensionality ratio (defined as the ratio of the length of the attacker's WM signature to the size of weights or activations).

Table 5: Robustness comparison against overwriting attacks. The WM information embedded by DeepSigns can withstand overwriting attacks for a wide of range of $\frac{N}{M}$ ratio. In this experiment, we use the CIFAR10-WRN since this benchmark is the only model evaluated by [31, 21].

N to M Ratio	Bit Error Rate (BER)	
	Uchida et.al [31, 21]	DeepSigns
1	0.309	0
2	0.41	0
3	0.511	0
4	0.527	0

As demonstrated, DeepSigns' dynamic data- and model-aware approach is significantly more robust compared to prior art [31, 21]. As for its robustness against pruning attacks, our approach is tolerant of higher pruning rates. Consider the CIFAR10-WRN benchmark as an example, DeepSigns is robust up to 80% pruning rate, whereas the works in [31, 21] are only robust up to 65% pruning rate.

5.2. Output Layer Watermarking

There are two prior works targeting watermarking the output layer [1, 20]. Even though the works by [1, 20] provide a high WM detection rate (reliability), they do not address the integrity requirement, meaning that these approaches can lead to a high false positive rate in practice. Table 6 provides a side-by-side comparison between DeepSigns and the work in [20]. As shown, DeepSigns has a significantly lower probability of falsely claiming the ownership of a remote DNN.

Table 6: Integrity comparison between DeepSigns and the prior work (PW) [20]. For each benchmark, the WM key size is set to $K = 20$ and 10 different sets of WM keys are generated to query six unmarked DNNs. The average false positive rates of querying each DNN model are reported.

False Positive Rate	Model 1		Model 2		Model 3		Model 4		Model 5		Model 6	
WM Method	PW	Ours	PW	Ours	PW	Ours	PW	Ours	PW	Ours	PW	Ours
MNIST	0.5	0.0	0.3	0.0	0	0.0	0.1	0.0	1.0	0.0	1.0	0.0
CIFAR10-CNN	0.0	0.0	0.1	0.0	0.1	0.0	0.0	0.0	1.0	0.0	0.0	0.0
CIFAR10-WRN	0.5	0.0	0.8	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The work in [1] uses the accuracy on the test set as the decision policy to detect WM information. It is well-known that there is no unique solution to DL problems [2, 5, 7]. In other words, there are various models with even different topologies that yield approximately the same test accuracy for a particular data application. Besides high false positive rates, another drawback of using test accuracy for WM detection is the high overhead of communication and computation [1]; therefore, their watermarking approach suffers from a low efficiency. DeepSigns uses a small WM key size ($K = 20$) to trigger the WM information, whereas a typical test set in DL problems can be two to three orders of magnitude larger.

6. Discussion

We demonstrate DeepSigns' performance on image classification tasks in Section 4. It is worth noting that DeepSigns framework is generic and can be deployed in sequence-based data application. Here, we use sentiment classification on IMDB dataset as an example to illustrate DeepSigns' generalizability. In this experiment, we consider the same potential attack scenarios (model fine-tuning, parameter pruning, watermark overwriting) as discussed in Section 2.1. The results are summarized in Figure 12. The IMDB-LSTM benchmark consists of three layers: an embedding layer with input dimension 5000 and output dimension 32, a LSTM layer with 100 neurons, and a Dense layer with 1 neuron. As can be seen from Figure 12, DeepSigns respects fidelity, reliability, integrity, and robustness criteria on the RNN benchmark.

	DNN Watermarking						
	Output Layer	Hidden Layer	Resource Management API	Data & Model Aware	Reliability	Integrity	Capacity
DeepSigns	✓	✓	✓	✓	✓	✓	N-bit with N≥1
Uchida et al. (2017)	x	✓	x	x	✓	✓	N-bit with N≥1
Merrer&Perez (2017)	✓	x	x	x	✓	x	1-bit
Adi, et al. (2018)	✓	x	x	x	✓	x	1-bit

Figure 11: High-level comparison with prior art DNN watermarking frameworks.

Criteria	Fidelity		Reliability	Integrity	Robustness
Metrics	Baseline Accuracy	Accuracy of Marked Model	BER	False Positive Rate	BER (after attack)
IMDB-LSTM	85.85%	86.56%	0	0	0

Figure 12: Evaluation results of DeepSigns's performance on the IMDB sentiment classification task. A binary string of length 4 is embedded in the second layer of the target model.

7. Conclusion

Deep learning is facilitating breakthroughs in various fields such as medical, aerospace, business, and education. While the commercialization of DNNs is so popular, efficient IP protection for pre-trained, ready-to-deploy models has been a standing challenge. It is timely to devise a systematic solution for DNN IP protection. This paper takes the first step towards this goal by providing an efficient end-to-end framework that enables WM embedding in activations of a neural network while minimally affecting the overall runtime and resource utilization. Our accompanying API paves the way for model designers to achieve a reliable technology transfer.

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. *Usenix Security Symposium*, 2018.

[2] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, 2015.

[3] Ingemar J Cox, Joe Kilian, F Thomson Leighton, and Talal Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE transactions on image processing*, 6(12), 1997.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[5] Li Deng and Dong Yu. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4), 2014.

[6] Borko Furht and Darko Kirovski. *Multimedia security handbook*. CRC press, 2004.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[8] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

[9] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[10] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[11] Frank Hartung and Martin Kutter. Multimedia watermarking techniques. *Proceedings of the IEEE*, 87(7), 1999.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[13] Neil F Johnson, Zoran Duric, and Sushil Jajodia. *Information Hiding: Steganography and Watermarking-Attacks and Countermeasures: Steganography and Watermarking: Attacks and Countermeasures*, volume 1. Springer Science & Business Media, 2001.

[14] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553), 2015.

[16] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[17] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[18] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning (ICML)*, 2016.

[19] Chun-Shien Lu. *Multimedia Security: Steganography and Digital Watermarking Techniques for Protection of Intellectual Property: Steganography and Digital Watermarking Techniques for Protection of Intellectual Property*. Igi Global, 2004.

[20] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *arXiv preprint arXiv:1711.01894*, 2017.

[21] Yuki Nagai, Yusuke Uchida, Shigeyuki Sakazawa, and Shin'ichi Satoh. Digital watermarking for deep neural networks. *International Journal of Multimedia Information Retrieval*, 7(1), 2018.

[22] Ankit B Patel, Tan Nguyen, and Richard G Baraniuk. A probabilistic theory of deep learning. *arXiv preprint arXiv:1504.00641*, 2015.

[23] Gang Qu and Miodrag Potkonjak. *Intellectual property protection in VLSI designs: theory and practice*. Springer Science & Business Media, 2007.

[24] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. Mlaas: Machine learning as a service. In *IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015.

[25] Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. Delight: Adding energy dimension to deep neural networks. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 2016.

[26] Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. Deep3: Leveraging three levels of parallelism for efficient deep learning. In *Proceedings of ACM 54th Annual Design Automation Conference (DAC)*, 2017.

[27] Bita Darvish Rouhani, Mohammad Samragh, Tara Javidi, and Farinaz Koushanfar. Safe machine learning and defeat-ing adversarial attacks. *IEEE Security and Privacy (S&P) Magazine*, 2018.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[29] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[30] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5), 2016.

[31] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the ACM on International Conference on Multimedia Retrieval*, 2017.

[32] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.