DeepSigns: A Generic Watermarking Framework for IP Protection of Deep Learning Models

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

Abstract

This paper proposes DeepSigns, a novel end-to-end framework for systematic Watermarking and Intellectual Property (IP) protection in the context of deep learning. DeepSigns, for the first time, introduces a generic watermarking methodology that is applicable in both and black-box settings, where the adversary may or may not know the internal details of the model. The proposed methodology embeds the signature in the probability density function (pdf) of the data abstraction obtained in different layers of a deep neural network. Our approach is robust to removal and transformation attacks including model compression, model fine-tuning, and/or watermark overwriting. Extensive proof-of-concept evaluations on MNIST and CIFAR10 datasets, as well as a wide variety of neural networks architectures including Wide Residual Networks (Wide-ResNet), Multi-Layer Perceptron (MLP), and Convolutional Neural Networks (CNNs) corroborate DeepSigns' effectiveness and applicability.

1 Introduction

The fourth industrial revolution is underway. The popular class of Deep Learning (DL) models and other contemporary Artificial Intelligence (AI) methods are enabling this revolution by providing a paradigm shift in model accuracy and functionality. Several applications are already going through significant transformative changes due to intelligence integration including but not limited to social networks, autonomous transportation, automated manufacturing, natural language processing, intelligent warfare and smart health [1–4]. A practical concern in the rush to adopt AI as a service is the capability to perform model protection: AI models are usually trained by allocating significant computational resources to process massive amounts of training data. The built models are therefore considered as the owner's IP and need to be protected to preserve the competitive advantage.

Embedding digital watermarks into DL models is critically important for a reliable technology transfer. A digital watermark is a type of marker covertly embedded in a signal or IP including audio, video image, or functional design. It is commonly adopted to identify ownership of the copyright of such a signal or function. Watermarking has been immensely leveraged over past decade to protect the ownership of multimedia and video content, as well as digital circuit functionality [5–7]. Extension of watermarking techniques to AI models and particularly deep neural networks, however, is still in its infancy. The AI models can be utilized in both white-box and black-box settings. In the white-box setting, the model parameters are public and voluntary shared with the third-party. In the black-box setting, however, the model details are not publicly shared and are only available to execute as a remote oracle.

The authors in [8] propose a new approach for watermarking of convolutional neural networks by embedding the IP information in the *static* content of a model (i.e., weight matrices). Although this work provides a significant leap as the first attempt to watermark neural networks, it poses (at least) three limitations: (i) It incurs a bounded watermarking capacity due to using the static properties of a model (weights) as opposed to using the dynamic content (activations). Note that the weights of a neural network are invariable (static) in the execution phase regardless of the data passing through the model. The activations, however, are dynamic and both *data and model dependent*. As such, we argue that using activations (instead of static weights) provides more flexibility for watermarking purposes. (ii) It is not robust against overwriting the original watermark by a third-party. (iii) It targets white-box settings and inapplicable to black-box scenarios.

A more recent work in [9], propose a 1-bit watermarking methodology that is applicable to black-box models. This approach is built upon model boundary modification and the use of adversarial samples that lie near decision boundaries. Adversarial samples, however, are known to be statistically unstable, meaning that the adversarial samples crafted for a model are not necessarily mis-classified by another network [10, 11]. Thereby, the proposed approach in [9] is highly sensitive to hyper-parameter tuning and usually leads to a high false alarm rate if the detection policy is not precisely fine-tuned in a supervised setting. Note that false ownership proofs based upon watermark extraction, in turn, jeopardize the integrity of the proposed watermarking methodology and render the watermarks ineffective.

This paper proposes DeepSigns, a novel end-to-end framework that enables coherent integration of robust digital watermarks in contemporary deep learning models. DeepSigns, for the first time, introduces a *generic* functional watermarking

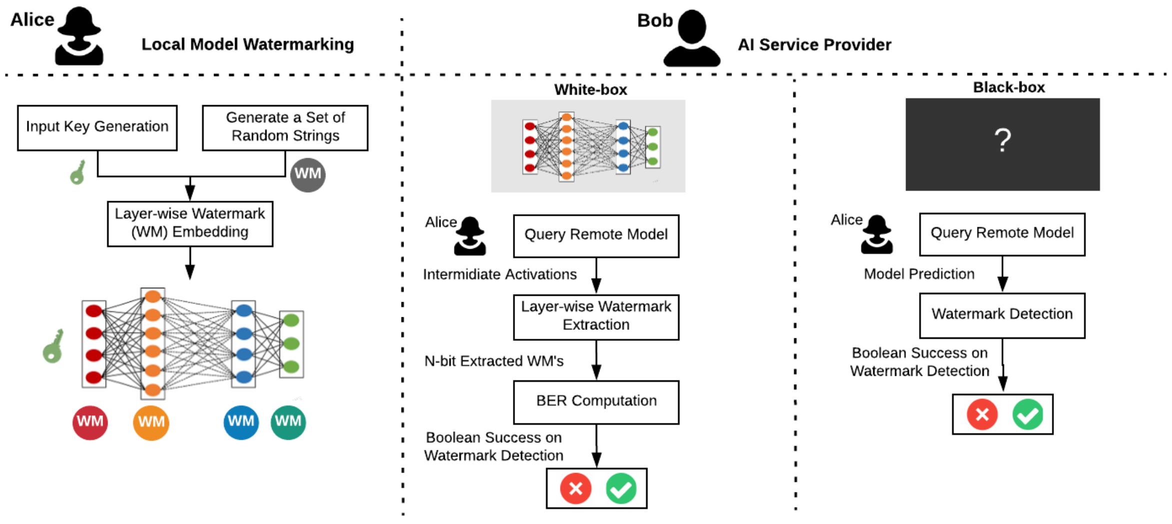


Figure 1: DeepSigns Global Flow: DeepSigns performs functional watermarking on DL models by sequentially embedding a set of binary random strings in the pdf of the activation set acquired at each intermediate layer and the output layer. Typically, a specific input (key) is utilized for extracting the embedded watermark. In our case, the inputs triggering the ingrained binary random strings are used as the key for detection of IP infringement in both white-box and black-box settings.

methodology that is applicable to both black-box and white-box settings. The proposed methodology is simultaneously *data and model dependent*. DeepSigns works by embedding an arbitrarily N-bit string into the pdf of the activation set in each layer of the neural network with no drop in the overall accuracy. The embedded strings can be triggered by the corresponding input keys to remotely detect the existence of the pertinent neural network in a third-party IP.

We demonstrate the robustness of our proposed framework to removal and transformation attacks including model compression/pruning, model fine-tuning, and watermark overwriting. As we empirically corroborate in Section 5, DeepSigns is robust against the state-of-the-art removal attacks and does not require excessive hyper-parameter tuning to avoid false alarms in black-box settings. The explicit contributions of this paper are as follows:

- Proposing DeepSigns, the first end-to-end framework for systematic deep learning IP protection that works for both white-box and black-box settings. A novel watermarking methodology is introduced to encode the pdf of the DL model and effectively trace the IP ownership.
- Providing a new and comprehensive set of metrics to assess the performance of a watermark embedding approach for DL models. Such metrics, in turn, enables coherent comparison of current and pending AI model protection methodologies.
- Devising an Application Programming Interface (API) to facilitate the adoption of DeepSigns watermarking methodology in training various DL models including convolutional and fully-connected DL models.
- Performing extensive proof-of-concept evaluation on various benchmarks including commonly used MNIST, CIFAR10 datasets. Our evaluations demonstrate the effectiveness of DeepSigns to protect the IP of an arbitrary neural network and detect IP ownership.

2 Global Flow

Figure 1 demonstrates the high-level block diagram of DeepSigns framework. In order to protect the IP of a particular neural network, the model owner (Alice) is required to embed a set of N-bit binary random watermark (WM) strings within each intermediate activation set of the neural network as discussed in Section 3. Once the neural network is locally trained by Alice to include the pertinent watermarks, it is ready to be employed by a third-party AI service provider (Bob) either as a black-box API or a white-box model. To prove the ownership of a model, Alice can query the remote service provider and detect whether her model is used in the underlying AI service as outlined by the protocol in Section 4.

Watermarking Requirements. Table 1 details the requirements for an effective watermarking methodology in the context of deep learning. In addition to the primary requirements listed in [8, 9], we argue that reliability, integrity, and generalizability are three other major factors that need to be considered for designing a practical watermarking methodology. The reliability property has to do with the fact that the embedded watermark should be accurately extracted using the pertinent keys; thereby, the model owner is able to detect any misuse of her model with a high probability. The integrity property ensures that the IP infringement detection policy yields the minimal number of false alarms; meaning that there is very low chance of (i) watermark collision for different models, and/or (ii) falsely proving a third-party ownership. Generalizability is another main factor in developing an effective watermarking methodology. DeepSigns satisfies all the requirements listed in Table 1 as shown by our experiments in Section 5.

Potential Attack Scenarios. To validate the robustness of the proposed watermarking approach, we evaluate DeepSigns performance against three types of attacks for IP infringement in deep learning services: (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 (compression) as another attack approach that might affect the watermarking extraction/detection. (iii) *Watermark overwriting*. The third-party user Bob that is aware of the methodology used to watermark the model (but not Alice's private keys) may try to embed a new watermark in the model and overwrite the original one. A watermarking methodology should be robust against overwriting attacks to effectively prevent IP infringement.

3 Functional Watermarking

Many contemporary AI applications possess non-convex loss surfaces with a large number of local minima that are likely to yield an accuracy (on test data) very close to another approximate model [12]. DeepSigns framework is built upon the fact that there is not a unique solution to address modern non-convex optimization used in the context of deep learning. DeepSigns framework works by iteratively learning and adjusting the corresponding pdf of data abstractions to incorporate the desired watermarking information within each layer of the neural network. The watermarking information can later be leveraged to claim the ownership of the neural network or detect IP infringement.

In many real-world deep learning applications, the activation features attained in the intermediate (a.k.a., hidden) layers roughly follow a Gaussian distribution [13]. In this paper, we consider a Gaussian Mixture Model (GMM) as the prior probability to characterize the data distribution at each hidden layer.[1] The last layer (a.k.a., output layer) is an exception since the output can be a discrete variable (e.g., class label) in a large category of AI applications. As such, DeepSigns governs the hidden (Section 3.1) and output (Section 3.2) layers differently.

3.1 Watermarking Intermediate Layers

To accommodate for the GMM prior distribution assumption, we suggest adding the following term to the conventional loss (e.g., cross-entropy) used for training deep neural networks which we denote as $\mathcal{L}(\theta, x)$ and refer to as $loss_0$:

$$\lambda_1 \; (\underbrace{\|\mu^l_{y^*} - f^l(x,\theta)\|_2^2 \; - \; \Sigma_{i \neq y^*} \|\mu^l_i - f^l(x,\theta)\|_2^2}_{\mathbf{loss_1}} \;), \qquad (1)$$

[1]We emphasize that our proposed approach is rather generic and is not restricted to the GMM distribution; the GMM distribution can be replaced with any other prior depending on the application.

where λ_1 is a trade-off parameter that specifies the contribution of the additive loss term. The additive loss function ($loss_1$) aims to minimize the entanglement between each pair of two Gaussian distributions (associated with the activation features corresponding to two different classes) while decreasing the inner-class diversity. This objective resembles the terminology of a kernel-based support vector machine which, in turn, facilitates approximation of the underlying pdf as a GMM distribution. In Equation (1), θ is the model parameter set (i.e., weights and biases), $f^l(x,\theta)$ is the corresponding activation features of input sample x at the l^{th} layer, y^* is the ground-truth label, and μ^l_i denotes the mean value of the Gaussian distribution at layer l that best fits the data abstractions belonging to class i. The mean values μ^l_i and intermediate feature vectors $f^l(x,\theta)$ are trainable variables that are iteratively learned during the training process of the target deep neural network.

Watermark Embedding. In order to watermark the target neural network, the model owner (Alice) first needs to generate three sets of WMs for each intermediate layer of her model:

(I) Choosing 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. The mean values of the selected distributions are then used to carry on the watermark information generated in the steps II and III.

(II) Designating an arbitrary binary string to be embedded in the target model. The elements (a.k.a., bits) of the binary string are independently and identically distributed (i.i.d). Henceforth, we refer to this binary string as the vector $b \in \{0,1\}^{s\times N}$ where s is the number of selected distributions (step I) to carry on the watermarking information, and N is a user-defined parameter indicating the desired length of the digital watermark embedded at each Gaussian's mean value.

(III) Specifying a random projection matrix (A) to map the selected centers in step I into the binary string chosen in step II. The transformation is denoted as follows:

$$\begin{aligned} G_\sigma^{s\times N} &= Sigmoid\,(\mu^{s\times M} \,.\, A^{M\times N}), \\ b^{s\times N} &= Hard\_Thresholding\,(G_\sigma^{s\times N},\, 0.5). \end{aligned} \qquad (2)$$

Here, M is the size of the feature space in the pertinent layer, and $\mu^{s\times M}$ denotes the concatenated mean values of the selected distributions. In our experiments, we leverage a standard normal distribution $\mathcal{N}(0,1)$ per [8] suggestion to generate the projection WM (A). Using i.i.d. samples drawn from a normal distribution, in turn, ensures that each bit of the binary string is embedded into all the features associated with the selected centers (mean values).

The process of computing the vector G_σ is differentiable; thereby, for a fixed set of projection matrix (A) and binary string (b), the selected centers can be readily modified/trained via back-propagation such that the hamming distance between the binarized projected centers and the actual WM values b is minimized (ideally zero). To do so, one needs to add

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

Requirements	Description
Fidelity	The functionality (e.g., accuracy) of the target neural network shall not be degraded as a result of watermark embedding.
Capacity	The watermarking methodology shall be capable of embedding a large amount of information in the target neural network.
Efficiency	The overhead of watermark embedding and extraction/detection shall be negligible.
Security	The watermark shall leave no tangible footprint in the target neural network; thus, an unauthorized individual cannot detect the presence of a watermark in the model.
Robustness	The watermarking methodology shall be resilient against model modifications such as compression/pruning, fine-tuning, and/or watermark overwriting.
Reliability	The watermarking methodology should yield minimal false negatives; the watermarked model should be effectively detected using the pertinent keys.
Integrity	The watermarking methodology should yield minimal false alarm (a.k.a., false positive); the watermarked model should be uniquely identified using the pertinent keys.
Generalizability	The watermarking methodology should be detectable in both white-box and black-box settings.

the following term to the overall loss function for each specific layer of the underlying neural network:

$$-\lambda_2 \underbrace{\sum_{j=1}^{N}\sum_{k=1}^{s}(b^{kj}\ln(G_\sigma^{kj}) + (1-b^{kj})\ln(1-G_\sigma^{kj}))}_{\mathbf{loss_2}}. \quad (3)$$

Here, the variable λ_2 is a hyper-parameter that determines the contribution of $loss_2$ in the process of training the neural network. All the three loss functions ($loss_0$, $loss_1$, and $loss_2$) are simultaneously used to train the underlying neural network. We set λ_1 and λ_2 to 0.01 in all our experiments. As we empirically verified in Section 5, DeepSigns can effectively perform functional watermarking with no drop in the baseline accuracy of the original neural network that has no watermark embedded. In cases where the accuracy might be jeopardized due to excessive regularization of the intermediate activation features, one can mitigate the accuracy drop by expanding each layer to include more free variables. The accuracy compensation is due to the fact that although the probability of finding a poor local minimum is non-zero for small-size networks, this probability decreases quickly with the expansion of network size [12, 14].

3.2 Watermarking Output Layer

The network prediction in the very last layer of a DL model needs to closely match the ground-truth data (e.g., training labels in a classification task) in order to have the maximum possible accuracy. As such, instead of directly regularizing the activation set of the output layer, we choose to adjust the tails of the decision boundaries to incorporate a desired statistical bias in the network as a 1-bit watermark. In this paper, we particularly focus on classification tasks using deep neural networks. Watermarking the output layer includes three main steps as shown in Figure 2.

(I) Learning the pdf distribution of the activation set in each intermediate layer as discussed in Section 3.1. The acquired probability density function, in turn, gives us an insight on the sectors of the latent space that are thoroughly occupied by the training data and the regions that are only covered by the tail of the GMM distribution (unused regions).

(II) Generating a set of K unique random input samples to be used as the watermarking keys in step III. Each selected random sample should be passed through the pre-trained neural network in order to make sure its latent features lie within the unused regions (Step I). In particular, if the number of training data within a ϵ-ball of the random sample is less than a threshold, we accept that sample as one of the watermark keys. Otherwise, a new random sample is generated to replace the previous sample. Each selected input sample is then mapped to a corresponding random ground-truth vector. For instance, in a classification application, each random input is associated with a randomly selected class.

(III) Fine-tuning the pre-trained neural network with the selected random watermarks in Step II such that the network has an exact prediction (e.g., an accuracy greater than 99%) for those samples. In our experiments, we utilize the same optimizer setting originally used for training the neural network except that the learning rate is reduced by a factor of 10 to prevent accuracy drop in the prediction of the legitimate input data.

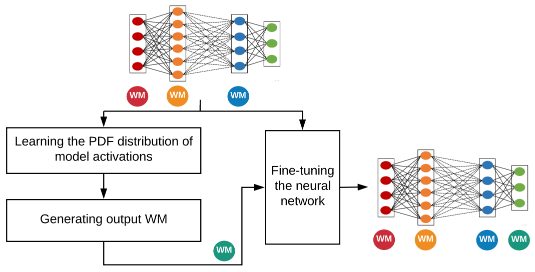

WM
WM
WM
Learning the PDF distribution of model activations
Generating output WM
WM
Fine-tuning the neural network
WM
WM
WM
WM

Figure 2: High-level overview of watermarking the output layer in a neural network. Output watermarking is performed after embedding the selected binary WMs in the intermediate (hidden) layers.

It is worth noting that the watermarking information embedded in the output layer can be extracted even in settings where the DL model is used as a black-box API by a third-party (Bob). Recently, [9] have proposed a method, called frontier stitching, to perform 1-bit watermarking in black-box settings. Our proposed approach is different in a sense that we leverage random samples that lie within the tail regions of the probability density function spanned by the model as opposed to relying on adversarial samples that lie close to the boundaries [10, 11, 15, 16]. Adversarial samples are known to be statistically unstable, meaning that the adversarial samples carefully crafted for a model is not necessarily mis-classified by another network. As shown in [9], frontier stitching is highly vulnerable to the hyper-parameter selection of the watermarking detection policy and may lead to a high false positive if it is not precisely tuned; thus jeopardizing the integrity requirement.

As we experimentally verify in Section 5, DeepSigns overcomes the integrity concern by selecting random samples within the unused space of the model. This is due to the fact that the unused regions in the space spanned by a model is specific to that model, whereas the decision boundaries for a given task retain high correlations in different models.

4 Watermarking Extraction

For the purpose of watermark inquiry, the model owner (Alice) sends a set of queries to the AI service provider (Bob). The queries include the input keys discussed in Sections 3.1 and 3.2. In case of the black-box usage of the network, Alice can only retrieve model predictions for the queried samples whereas in the white-box setting the intermediate activations can be also recovered.

To extract the watermarking information in the intermediate layers, Alice first needs to compute the element-wise mean value of the activation features to approximate the Gaussian centers that carry on the watermarking data.[2] As the second step, Alice requires to perform the inverse of the process described in Equation 2 to figure out the corresponding binary string. Note that in case of a mismatch between the recovered string from Bob's model and the original watermarking information selected by Alice, random watermarks will be extracted which, in turn, yield a very high Bit Error Rate (BER) as shown in Section 5.

To verify the presence of the watermark in the output layer, Alice needs to statistically analyze Bob's responses to her keys. In particular, if the number of mismatches between the model predictions and Alice's ground-truth labels is less than a threshold, it means that the model used by Bob possesses a high similarity to the network owned by Alice. Otherwise, the two models are not replicas. When the two models are the exact duplicate of one another, the number of mismatches will be zero and Alice can safely claim the ownership of the network used by Bob. However, in the real-world settings, the target neural network might be slightly modified by Bob in both malicious or nonmalicious ways. Examples of such modifications are model fine-tuning, model compression, or WM removal. As such, the threshold used for the purpose of watermark detection should be greater than zero to withstand the modifications.

[2]The queries that belong to the output WMs are excluded.

The probability of a network (not owned by Alice) to make at least n_k correct decision according to the Alice private keys is as follow:

$$P(N_k > n_k|\mathcal{O}) = 1 - \sum_{k=0}^{n_k} \binom{K}{k} (\tfrac{1}{C})^{K-k} (1 - \tfrac{1}{C})^k, \qquad (4)$$

where $\mathcal{O}$ is the oracle model used by Bob, N_k is the number of matched predictions of the two models compared against one another, K is the key length according to Section 3.2, and C is the number of classes in the target DL application. In our experiments, we use the decision policy ($P(N_k > n_k|\mathcal{O}) > 1 - 1e^{-5}$) for watermark detection. In Section 5, we corroborate the integrity and robustness of DeepSigns against various attack scenarios and black-box models.

5 Evaluations

We evaluate the performance of DeepSigns on MNIST [17] and CIFAR10 [18] datasets and three different neural network architectures. Table 2 summarizes the neural network 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 WM used for watermarking the hidden layers. In all white-box related experiments, we use the second to the last layer for the purpose of watermarking. In the block-box scenario, we leverage the very last layer for watermark embedding/detection.

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 2, $MP2(1)$ denotes a max-pooling layer over regions of size 2×2 and stride of 1, and $512FC$ is a fully-connected layer consisting of 512 output neurons. ReLU is used as the activation function in all the three benchmarks. For the CIFAR10 data, the CNN model is used in the white-box setting and the Wide-ResNet (WRN) is evaluated in the black-box scenario.

Dataset	Model Type	Baseline Accuracy	Marked Model Accuracy		Architecture
MNIST	MLP	98.54%	K=10	N=4	784-512FC-512FC-10FC
			98.51%	98.13%	
CIFAR10	CNN	78.47%	-	N=4	3*32*32-32C3(1)-32C3(1)-MP2(1)
				80.7%	-64C3(1)-64C3(1)-MP2(1)-512FC-10FC
CIFAR10	WideResNet	91.42%	K=10	-	Please refer to [19].
			91.48%		

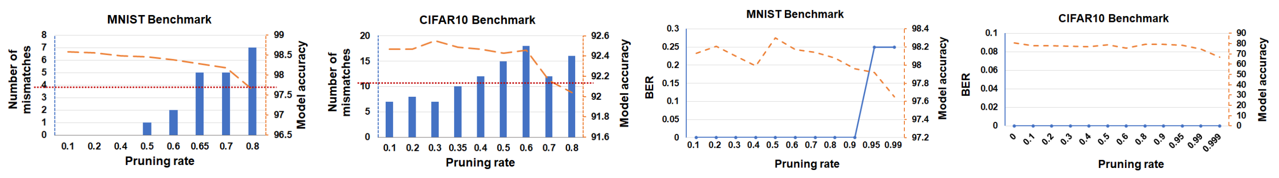
DeepSigns satisfies all the requirements listed in Table 1 as empirically verified in the following. We provide an accompanying API to facilitate watermark embedding and extraction in various DL models.

5.1 Fidelity

The proposed watermarking approach respects the fidelity requirement as demonstrated in Table 2. In some cases (e.g., wide-ResNet benchmark), we even observed a slight accuracy improvement compared to the baseline.

5.2 Robustness

We evaluate the robustness of DeepSigns framework against three contemporary removal attacks as discussed in Section 2.



(a) (b) (c) (d)

Figure 3: Evaluation of the watermark's robustness against parameter pruning. Figures (a) and (b) show the results for MNIST and CIFAR10 in the black-box setting. The horizontal red line is the mismatch threshold obtained from Equation (4). The dashed lines show the corresponding accuracy for each pruning rate. Figures (c) and (d) correspond to the MNIST and CIFAR10 benchmarks in the white-box setting. The dashed lines demonstrate the corresponding accuracy per pruning rate.

The potential attacks include parameter pruning [4, 20, 21], model fine-tuning [22], and watermark overwriting [8].

Parameter pruning. We use the pruning approach proposed in [20] to compress the neural network. In particular, for pruning each layer of a neural network, we first set the $\alpha\%$ of the parameters that possess the smallest weight values to zero. The obtained mask is then used to sparsely fine-tune the model in order to compensate for the accuracy loss induced by pruning. Figure 3 illustrates the impact of pruning on watermark extraction/detection in both black-box and white-box settings. In the black-box experiments, DeepSigns can tolerate up to 60% and 35% parameter pruning for the MNIST and CIFAR10 benchmarks, and up to 90% and 99% in the white-box related experiments. The WM length to perform watermarking in each benchmark are listed in Table 2. As shown in Figure 3, in occasions where pruning the neural network yields a substantial bit error rate (BER) value, we observe that the sparse model suffers from a large accuracy loss compared to the baseline. As such, one cannot remove the embedded watermark in a neural network by excessive pruning of the parameters while attaining a comparable accuracy with the baseline.

Model fine-tuning. For the fine-tuning transform, we retrain the target model using the original training data with the conventional cross-entropy loss (excluding the watermarking specific loss functions). Table 3 summarizes the impact of fine-tuning on watermarking detection rate across all the benchmarks. As shown, DeepSigns can successfully detect the watermark even after fine-tuning the target model while preserving the baseline accuracy. There is a trade-off between the model accuracy and success rate of watermarking removal.

Watermark overwriting. Assuming the attacker is aware of the watermarking technique, he may attempt to damage the original watermark by overwriting it. Figure 4 shows the results of watermark overwriting for MNIST and CIFAR10 in the black-box setting. As can be seen from the histogram, the embedded watermark can be successfully detected in the MNIST benchmarks with all the three WM lengths after overwriting attack. Even though the overwriting makes the watermark undetectable in the CIFAR-10 benchmark (for $K \geq 15$), the attacked model suffers from substantial accuracy loss; making the resulting model ineffective. Similarly, in the white-box scenario, the watermark extraction has non-zero BER if a large watermarking strength (e.g., λ_1 and $\lambda_2 \geq 0.1$) is used by the attacker, which also leads to a high accuracy loss.

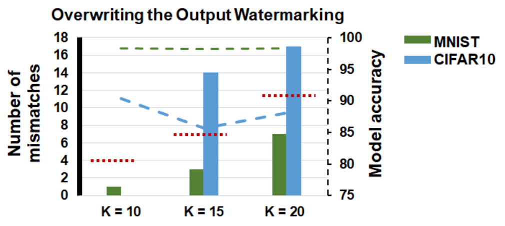


Figure 4: There is a trade-off between the model accuracy and the success rate of the overwriting attack. The red horizontal lines indicate the detection threshold for different key lengths (Section 4). The left vertical axis is correspondence to the bar figures and the right vertical axis shows the accuracy depicted by dashed lines.

5.3 Integrity

Figure 5 illustrates the results of integrity evaluation in the black-box setting where unmarked models with the same (models 1 to 3) and different (models 4 to 6) topologies are queried by Alice's keys. As shown in Figure 5, DeepSigns satisfies the integrity criterion and has no false positives, which means the ownership of unmarked models will not be falsely proved. Note that unlike the black-box setting, in the white-box scenario, different topologies can be distinguished by one-to-one comparison of Alice's and Bob's architecture. For the unmarked model with the same topology in white-box setting, the integrity analysis is equivalent to model fine-tuning, for which the results are summarized in Table 3.

5.4 Capacity

The capacity of the white-box activation watermarking is assessed by embedding binary strings of different lengths in the intermediate layers. As shown in Figure 6, DeepSigns allows up to 64 bits and 128 bits capacity for MNIST and CIFAR-10 benchmarks, respectively. Note that there is a trade-off between the capacity and accuracy which can be leveraged by

Table 3: Robustness of the proposed functional watermarking against model fine-tuning attack. A value 1 in the last row of the table indicates that the embedded watermark is successfully detected, whereas the value 0 indicates a false negative.

Metrics	**White-box**						**Black-box**					
	MNIST-MLP			CIFAR10-CNN			MNIST-MLP			CIFAR10-WRN		
Number of Epochs	50	100	200	50	100	200	50	100	200	50	100	200
Accuracy	98.21	98.20	98.18	70.11	62.74	59.86	98.61	98.63	98.60	87.65	89.74	88.35
BER	0	0	0	0	0	0	-	-	-	-	-	-
Detection Success	1	1	1	1	1	1	1	1	1	0	0	0

the IP owner (Alice) to embed a stronger watermark in her neural network model.

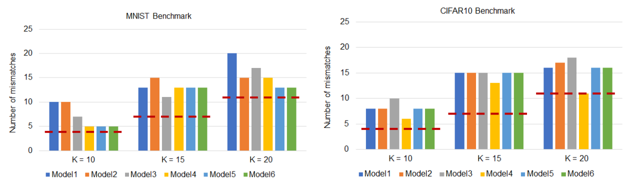


Figure 5: Integrity analysis for different benchmarks. The red horizontal lines indicate the detection threshold for various WM lengths.

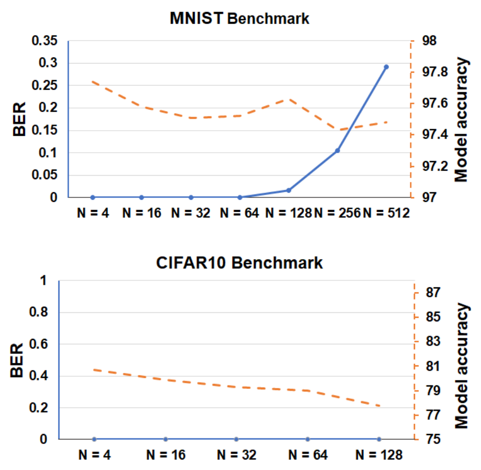
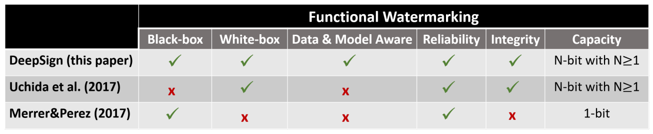


Figure 6: The trade-off between accuracy and watermarking capacity for MNIST and CIFAR10 benchmarks.

5.5 Discussion

Figure 7 compares the overall capability of the existing watermarking frameworks. [8] uses the weights of a convolution neural network for the purpose of watermarking as opposed to the activation sets. As shown in [8], watermarking weights is not robust against overwriting attack. DeepSigns's dynamic data and model aware approach, on the other hand, provides a much more robust and flexible watermarking methodology.



	Functional Watermarking					
	Black-box	White-box	Data & Model Aware	Reliability	Integrity	Capacity
DeepSign (this paper)	✓	✓	✓	✓	✓	N-bit with N≥1
Uchida et al. (2017)	x	✓	x	✓	✓	N-bit with N≥1
Merrer&Perez (2017)	✓	x	x	✓	x	1-bit

Figure 7: Comparison with the state-of-the-art neural network watermarking frameworks.

6 Conclusion

In this paper, we present DeepSigns the first generic DL watermarking framework that is applicable in both black-box and white-box settings. DeepSigns works by embedding the watermark information in the probability density distribution of the activation sets corresponding to different layers of a neural network. The performance of the proposed framework is evaluated on MNIST and CIFAR-10 datasets using three different topologies. Our results demonstrate that DeepSigns satisfies all the criteria for effective watermarking including fidelity, robustness, generalizability, and integrity. DeepSigns attains comparable accuracy to the baseline neural network after embedding the watermark and resists potential attacks such as parameter pruning, model fine-tuning, and watermark overwriting.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[3] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar, "Deep3: Leveraging three levels of parallelism for efficient deep learning," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 61.

[4] ——, "Delight: Adding energy dimension to deep neural networks," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 112–117.

[5] B. Furht and D. Kirovski, *Multimedia security handbook*. CRC press, 2004.

[6] F. Hartung and M. Kutter, "Multimedia watermarking techniques," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1079–1107, 1999.

[7] G. Qu and M. Potkonjak, *Intellectual property protection in VLSI designs: theory and practice*. Springer Science & Business Media, 2007.

[8] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ACM, 2017, pp. 269–277.

[9] E. L. Merrer, P. Perez, and G. Trédan, “Adversarial frontier stitching for remote neural network watermarking,” *arXiv preprint arXiv:1711.01894*, 2017.

[10] B. D. Rouhani, M. Samragh, T. Javidi, and F. Koushanfar, “Curtail: Characterizing and thwarting adversarial deep learning,” *arXiv preprint arXiv:1709.02538*, 2017.

[11] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, “On the (statistical) detection of adversarial examples,” *arXiv preprint arXiv:1702.06280*, 2017.

[12] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, “The loss surfaces of multilayer networks,” in *Artificial Intelligence and Statistics*, 2015, pp. 192–204.

[13] D. Lin, S. Talathi, and S. Annapureddy, “Fixed point quantization of deep convolutional networks,” in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

[14] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization,” in *Advances in neural information processing systems*, 2014, pp. 2933–2941.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.

[16] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against machine learning,” in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 506–519.

[17] Y. LeCun, C. Cortes, and C. J. Burges, “The mnist database of handwritten digits,” 1998.

[18] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” 2009.

[19] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *arXiv preprint arXiv:1605.07146*, 2016.

[20] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[21] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[22] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.