DICTION: DYNAMIC ROBUST WHITE BOX WATERMARKING SCHEME

Reda Bellafqira
Institut Mines-Telecom, IMT Atlantique
Inserm, UMR 1101 Latim
29238 Brest Cedex, France
reda.bellafqiraimt-atlantique.fr

Gouenou Coatrieux
Institut Mines-Telecom, IMT Atlantique
Inserm, UMR 1101 Latim
29238 Brest Cedex, France
gouenou.coatrieux@imt-atlantique.fr

ABSTRACT

Deep neural network (DNN) watermarking is a suitable method for protecting the ownership of deep learning (DL) models derived from computationally intensive processes and painstakingly compiled and annotated datasets. It secretly embeds an identifier (watermark) within the model, which can be retrieved by the owner to prove ownership. In this paper, we first provide a unified framework for white box DNN watermarking schemes. It includes current state-of-the art methods outlining their theoretical inter-connections. In second, we introduce DICTION, a new white-box Dynamic Robust watermarking scheme, we derived from this framework. Its main originality stands on a generative adversarial network (GAN) strategy where the watermark extraction function is a DNN trained as a GAN discriminator, and the target model to watermark as a GAN generator taking a GAN latent space as trigger set input. DICTION can be seen as a generalization of DeepSigns which, to the best of knowledge, is the only other Dynamic white-box watermarking scheme from the literature. Experiments conducted on the same model test set as Deepsigns demonstrate that our scheme achieves much better performance. Especially, and contrarily to DeepSigns, with DICTION one can increase the watermark capacity while preserving at best the model accuracy and ensuring simultaneously a strong robustness against a wide range of watermark removal and detection attacks.

Keywords Deep learning · Intellectual property protection · Watermarking

1 Introduction

Deep Learning (DL) has enabled significant and rapid progress in signal analysis and applications such as speech recognition, natural language processing, diagnostic support, and so on. More and more, building a deep learning model is a costly process that requires: (i) the availability of massive amounts of data; (ii) extensive computational resources (e.g., GPU); (iii) the assistance of DL experts to carefully define the network topology, and correctly set the training hyper-parameters (e.g., learning rate, batch size, weights decay...). Therefore, the illegal reproduction and redistribution of such a model constitute a significant economic loss for its authors.

Several solutions have been proposed to protect the ownership of deep learning models. Some are based on digital watermarking, the principle of which consists in inserting a message (a watermark) in a host document by imperceptibly modifying some of its characteristics. For instance, the watermarking of an image [1,2] stands on the slight modification (or modulation) of its pixel values to encode the message. Such a message can then help to verify the origin/destination of the document it protects in the context of copyright protection or fights against data leaks [3,4]. Watermarking a DL model relies on the same principles up to now the proposal of several approaches that consider specific DL constraints.

One can classify these methods into black-box (BBx) watermarking and white-box (WBx) watermarking schemes. By definition, a black-box watermark can be extracted by only querying the model. The watermarked message is read by looking at the outputs of the watermarked model for some well-designed and crafted inputs referred to as the "trigger set" [5–13] . On the other hand, white box watermarking schemes [14–20] have access to the whole model. They can be differentiated depending on if the watermark is directly read from the network weights; in that case, they are referred

to as "static watermarking" schemes; or, by examining the values of intermediate activation maps for specific model inputs, they are referred to as "dynamic watermarking" schemes [20].

The constraints under which all these watermarking schemes have been designed are the following. First, they should not degrade the performance of the model and, in second, they must be resistant to attacks aimed at removing or detecting the watermark. Among these attacks, one can cite the classic ones:

- The pruning attack (PA), where the weights of the model whose absolute values are below a threshold are set to zero.
- The fine-tuning attack (FTA), which consists of re-training the model and updating its weights without decreasing the model's accuracy.
- The overwriting attack (OA) – which occurs when the attacker embeds his/her watermark with the objective of removing the original watermark.
- Wang and Kerschbaum attack (WKA) devoted to static white-box watermarking algorithms which change the weight distribution of the watermarked model. It basically relies on the visual inspection of the weight distribution.
- Property Inference Attack (PIA) which relies on training a discriminating model capable of distinguishing watermarked from nonwatermarked models. Such a watermark detector uses weight distributions generated from multiple neural network models that are either nonwatermarked or watermarked, so as to capture watermarked weight patterns. This model can then be used to determine whether a protected model under attack is no longer watermarked.

To the best of our knowledge, if there exist several WBx static watermarking schemes [14–19], it is Rouhani *et al.* that introduced in [20] the first and unique white box dynamic watermarking solution: DeepSigns, in the context of image classification. To roughly sum it up, this one embeds a sequence of bits as a watermark into the probability density function (pdf) of the activation maps of several layers of a model. This is achieved by means of a fine-tuning process that depends on a trigger data set (a subset of images, secretly selected) and on the use of special regularizers to balance model accuracy and watermark detection. More specifically, the basic idea of DeepSigns is to cluster the activation maps of the model training set and to embed the watermark in a secret subset of the clustered classes. To insert a binary message into the model, the mean of each of these classes is computed on the trigger set and is modified by fine-tuning the model such that its projection in a space defined by a secret matrix equals the watermark. Concerning the extraction procedure, it is sufficient to know the trigger set, to calculate the statistical mean of each class' activation maps and their projection in the secret space to extract the watermark. While Deepsigns offers good model accuracy after watermarking, it suffers on the one hand from a very low insertion capacity (i.e., the number of bits one can insert in a model) and on the other hand from significant vulnerabilities to some attacks like FTA, PA and OWA as the size of the watermark is increased.

In this work, we propose a novel dynamic white box algorithm that overcomes these problems. Our solution does not necessitate the clustering of activation maps because there is no guarantee that the clustering of the updated model will be identical to that of the initial model when we fine-tune it. In particular, we propose to use a neural network to map the activation maps to the watermark space rather than a static projection matrix. Doing so significantly expands the ability to embed a watermark. Our neural network for projection is trained while embedding the watermark. Another originality of our proposal is to insert a watermark "b" into the activation maps of a given layer "l", using a trigger set constituted of sample images issued from a latent space defined by its mean and standard deviation. Such a strategy better preserves the model accuracy, avoiding possible misclassification of real images as for Deepsigns. The embedding of b relies on a generative adversarial-based strategy. The initial portion of the model up to layer "l" serves as a generative adversarial network (GAN) generator [21]. To map its activation maps to encode "b" in the watermark space, we train a projection neural network (like a GAN discriminator) concurrently with the generator model for this mapping in such a way it maps the latent space activation maps of the watermarked model to the desired watermark "b" and those of the nonwatermarked model to random watermarks. To detect the watermark, it is thus sufficient to generate samples of the latent space as model input and to verify that the projection neural network reconstructs the watermark from the corresponding activation maps. By doing so, we largely increase watermark capacity while better preserving the accuracy of the model and ensuring that the distribution of the watermarked and nonwatermarked activation maps is the same. In some cases, model accuracy is improved due to the fact that our watermarking proposal is treated as a regularization term. We are also independent of the selection of a set of real images as the trigger set. Moreover, and as we will see, our proposal is resistant to all the attacks listed above (PA, FTA, OA, WKA and PIA), outperforming the white-box dynamic watermarking scheme DeepSigns in terms of watermark robustness and accuracy of the watermarked model.

The rest of this paper is organized as follows. In Section 2, we present the most recent white-box watermarking schemes, static or dynamic, illustrating their advantages and weaknesses by including them into a unified white-box watermarking framework. We then take advantage of this framework to introduce in Section 3 our proposal DICTION, a novel dynamic white box watermarking scheme that is a generalization of DeepSigns with much better performance. Section 4 provides experimental results and comparison assessment of our scheme with state-of-the-art solutions. Section 5 concludes this paper.

2 Related works and focus on DeepSigns

In this section, we provide a formal definition of white-box neural network watermarking, a unified framework, by presenting the two most recent and efficient static schemes [14, 22] and, to our knowledge, the one dynamic scheme [20]. Before going into the details of these approaches, we first give a classical definition of the architecture of deep neural networks, which will serve as a reference in the following.

2.1 Deep Neural Networks (DNNs)

In this work, we decided to consider a common DNN architecture for an image classification purpose due to the fact the vast majority of neural network watermarking schemes have been developed and experimented in such application framework.

As shown in Fig. 1, in the context of an image classification application, a common architecture of a DNN model M is a stack of layers. It transforms an input layer, holding data X to classify, into an output layer holding the label score Y. Different kinds of layers are commonly used:

In this work, we decided to consider a classical DNN architecture for the purpose of image classification due to the fact the vast majority of neural network watermarking schemes have been developed and experimented this context. As depicted in Fig. 1, such a DNN model M is a stack of layers that transforms an input layer, holding data X to classify, into an output layer holding the label score Y. Different kinds of layers are commonly used:

1. Fully connected layer (FC) – It consists of a set of neurons such that each neuron of a layer FC^m is connected to all those of the previous layer FC^{m-1}. More clearly, a FC layer of I_m neurons connected to a previous layer of I_{m-1} neurons will receive as input a vector $a^{(m-1)}$ of size I_{m-1} and will provide as output a vector $a^{(m)}$ with I_m components that will be the input of the next layer. Such computation can be expressed as a mapping of the form:
$$a^{(m)} = g_m(W^{(m)}a^{(m-1)} + b^{(m)}) \tag{1}$$
where: $W^{(m)}$ is the matrix of neuron weights of shape $I_m \times I_{m-1}$; $b \in \mathbb{R}^{I_m}$ is a vector that contains the biases of the neurons; and, $g_m()$ is a nonlinear activation function applied component-wise (e.g, Relu, Sigmoid, Tanh ...).
2. Convolutional layer $(Conv(O_m, P_m, Q_m))$: – As illustrated in Fig. 1, such a layer receives as input an image $A^{(m-1)}$ of O_{m-1} channels and compute as output a new "image" $A^{(m)}$ composed of O_m channels. The output at each channel is usually named as a feature map, and is computed as:
$$A_0^{(m)} = g_m(\sum_k W_{ok}^{(m)} * A_k^{(m-1)} + b_0^{(m)}) \tag{2}$$
Where: " $*$ " denotes the 2D convolution operation; $W_{ok}^{(m)}$ is the filter weight matrix of shape $P_m \times Q_m$ along with a bias $b_0^{(m)} \in \mathbb{R}$. The matrix $W_{ok}^{(m)}$ parameterizes a spatial filter that is automatically tuned during training to detect or enhance some features from $A^{(m-1)}$.
3. Pooling layers (PL) – It is implemented in a DNN as a spatial dimensionality reduction operation. The most common pooling layers are max pooling and average pooling. Roughly, $Max - Pooling(n, n)$ is a patch kernel operator of size $n \times n$ applied to $A^{(m)}$ that outputs the maximum of the values found in each $n \times n$ patches of $A^{(m)}$. For average pooling, one takes the average of the values in the patches. The right choice of this function can make the model more robust to distortions in the input pattern. Anyway, a PL layer reduces the number of trainable parameters for the next layers.
4. Flatten Layer – it corresponds to a special reshaping operation that is required to move from bidimensional (or multidimensional) layers to one-dimensional fully connected layers.

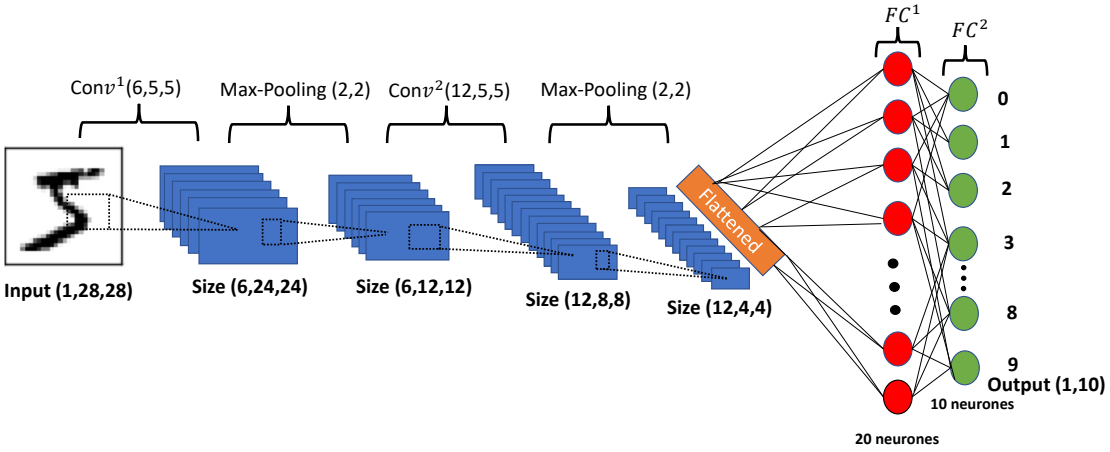


Figure 1: Example of a Convolutional Neural Network architecture for image classification (using the well known image dataser MNIST [23]), with 2 convolution layers, 2 max-pooling layers and a fully connected neural network of two layers.

Architecting a DNN entails developing a suitable sequence of convolutional, pooling, and fully connected layers, as well as their hyperparameters (e.g, learning rate, momentum, weight decay ...). Typical architectures, as shown in Fig. 1, introduce a pooling layer after one convolutional layer, defining a convolutional block that is repeated until the feature map size is small enough to feed a fully connected layers. A special reshaping operation known as "a flatten layer" is required to transition from bidimensional (or multidimensional) layers to one-dimensional fully connected layers.

The training of DNN is based on two iterative phases: feed-forward and back-propagation. Before the first feed-forward phase begins, all weights are initialized with random values, for example. The training data is then fed into the DNN according to a batch size which is the number of samples used to perform a single forward and backward pass. The difference between the original label (Y_{Train}) and the ones computed by the DNN is then calculated using an objective function (also known as the loss function $(E(.))$) (e.g. cross entropy for classification, Mean Square Error for regression). This error is then used in the back-propagation phase to update all weights using techniques such as gradient descent.

Once the DNN model trained, i.e., once the weights known (e.g., model accuracy does not increase anymore), it can be used so as to classify new data (inference phase). This classification process simply consists in applying the feed-forward phase with new data as input, considering that the DNN output will give the class of the input data.

2.2 White-Box watermarking

As stated above, there are two classes of white box watermarking schemes: the static class and the dynamic class. The former groups the methods that embed the watermark by directly modifying a trained model's weights [15, 24] using a classical watermarking modulation as for images. In the second class [14, 16, 20, 22], the watermark is inserted during the training of the model for the original task it is designed for. By doing so, the idea is that the watermark insertion process does not reduce the performance of the host model. From our point of view, these methods can be integrated within a unified white box watermarking framework that depicts any scheme accordingly two main steps:

1. At first, considering a target model M, a secretly parameterized features extraction function $Ext(M, K_{ext})$ is applied using a secret key K_{ext}. As we will see, such features can be a subset of the model weights. In that case K_{ext} simply indicates the indices of the model weights to select. It can also be some model activation maps for some secretly selected input data, issued from a trigger set. These features are then used for watermark insertion/extraction.
2. The embedding of a watermark message b consists of regularizing M with a specific regularization term E_{wat} so that the projection function $Proj(., K_{Proj})$ applied to the selected features encodes b in a given watermark space; space which depends on the secret key K_{Proj}. The idea is that after training, we have:
$$Proj(Ext(M^{wat}, K_{ext}), K_{Proj}) = b \tag{3}$$
where M^{wat} is the watermarked version model of the target model M.
Obviously, to do so, the watermarking regularization term E_{wat} depends on a distance measure d defined in the watermark space (e.g., the hamming distance or cross entropy in the case of a binary watermark, a binary string of length l , i.e., $b \in \{0,1\}^l$) being defined as:
$$E_{wat} = d(Proj(Ext(M^{wat}, K_{ext}), K_{Proj}), b) \tag{4}$$
To preserve the accuracy of the target model, the watermarked model M^{wat} is in general derived from M through a fine-tuning operation parametrized with the following loss function:
$$E = E_0(X_{Train}, Y_{Train}) + \lambda E_{wat} \tag{5}$$
where: $E_0(X_{Train}, Y_{Train})$ represents the original loss function of the network, the presence of which is important to ensure a good behavior with regard to the classification task); E_{wat} is the regularization term added to ensure the correct extraction of the watermark embedding; λ is a parameter for adjusting the tradeoff between the original loss term and the regularization term.

The watermark retrieval is by next pretty simple. It consists in using both the features extraction function $Ext(., K_{ext})$ and the projection function $Proj(., K_{Proj})$ as follow :
$$b^{ext} = Proj(Ext(M^{wat}, K_{ext}), K_{Proj}) \tag{6}$$
where: b^{ext} is the extracted message from M^{wat}.

In the following sub-sections, we depict the algorithms of Uchida *et al.* [14], of RIGA [22] and Deepsigns [20] accordingly our framework.

2.2.1 Uchida *et al.*'s algorithm

In the scheme of Uchida *et al.*' [14], the feature extraction function $Ext(., K_{ext})$ corresponds to the mean value of some secretly selected filters weights. More clearly, based on K_{ext}, a convolutional layer is selected. By next, let (s, s), d, and n represent the kernel size of the layer filters, the number of channels of the layer input and the number of filters, respectively and let also note the tensor $\mathbf{W} \in \mathbb{R}^{s \times s \times d \times n}$ with all the layer weights. The features extraction function then includes the following steps :

1. calculate the mean values $\overline{W}_{ijk} = \frac{1}{n} \sum_{h=1}^{n} W_{ijkh}$, i.e. the average value of filters' coefficients at the same position, getting $\overline{\mathbf{W}} \in \mathbb{R}^{s \times s \times d}$.
2. flatten $\overline{\mathbf{W}}$ to produce a vector $\mathbf{w} \in \mathbb{R}^v$ with $v = s \times s \times d$.

The projection function $Proj(., K_{Proj})$ of [14] has been designed to insert a l bit long watermark $b \in \{0,1\}^l$ being defined as:
$$Proj(\mathbf{w}, K_{Proj}) = \sigma(\mathbf{w}A) \in \{0,1\}^l \tag{7}$$
where: $K_{Proj} = A$ is a secret random matrix of size $(|w|, l)$; $\sigma(.)$ is the Sigmoid function:
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
As distance d for the watermarking regularization E_{wat}, Uchida *et al.* [14] exploits the binary cross entropy:
$$d(b, y) = -\sum_{j=1}^{l} (b_j \log(y_j) + (1 - b_j) \log(1 - y_j)) \tag{8}$$

Based on d, $Proj(., K_{Proj})$ and $Ext(., K_{ext})$, one can then compute the loss E_{wat} as given in (5) to watermark a target model M.

Herein, it is interesting to note that we can reformulate the projection function $Proj(.)$ as a simple perceptron. Indeed, as given in (7) it is the Sigmoid of the multiplication of A by $\mathbf{w}$, which is equivalent of a perceptron with Sigmoid as the activation function and the parameters of which θ includes : $|W|$ weights and null biases (with $|.|$ the cardinality operator). When Uchida *et al.* in [14] train their scheme so that the feature projection encodes the watermark message, this is equivalent to train a perceptron $Proj_\theta(.)$ on the database $(A_i, b_i)_{i=1...l}$, where: A_i, the i^{th} column of A, corresponds to a training sample; and, b_i, the i^{th} bit of b, is the label of A_i. Consequently, $Proj_\theta(.)$ is trained so that the projection of A on θ gives b, a watermark of l bits. This can be achieved by training $Proj_\theta(.)$ with a batch size of size l, that is the number of columns of A (or equivalently the size of watermark).

From hereon, the insertion process of the Uchida *et al.* scheme can be reformulated and generalized as follows, let us consider: a target model M to watermark into M^{wat} with the message b and the secret K_{Proj} (or equivalently the secret matrix A), and a single perceptron $Proj_\theta(.)$ the parameters of which are initialized with the values of the secretly extracted features $Ext(M, K_{ext})$ from the target model $\mathbf{M}$. The insertion process consists in finetuning the target model M into M^{wat} using the following global loss:

$$E = E_0(X_{Train}, Y_{Train}) + \lambda(d(Proj_\theta(A), b) + ||\theta - Ext(M^{wat}, K_{ext})||) \tag{9}$$

To clarify, the loss $d(Proj_\theta(A), b)$ allows $Proj_\theta(.)$ to find its parameters θ that minimize the distance between $Proj_\theta(A)$ and b, while the term $||\theta - Ext(M^{wat}, K_{ext})||$ allows minimizing the distance between θ and the secretly selected weights w^{wat} extracted from the watermarked model M^{wat} by using the feature extraction function $Ext(M^{wat}, K_{ext})$. E_0 is obviously presents to preserve the model accuracy. The interest of our reformulation is that it allows the parametrization of M^{wat} and $Proj_\theta(.)$ training in different fashion (e.g., we can use different learning rates and optimizers in the learning of θ and $\mathbf{w}^{wat}$). We verified experimentally this statement obtaining exactly the same performance as the original Uchida *et al.* scheme [14]. It is important to notice that both models M^{wat} and $Proj_\theta(.)$ are trained simultaneously for fast convergence, and that only M^{wat} is given to the recipient.

Regarding, the extraction process of the watermark, it can similarly be expressed in two ways. The first one consists in hard thresholding the result of the $Proj()$ function applied to w^{wat}:

$$b_i^{ext} = \begin{cases} 1 & Proj(Ext(M^{wat}, K_{ext}), K_{Proj})_i \geq 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

where $Proj(w^{wat}, K_{Proj})_i$ is the i^{th} component of $Proj(Ext(M^{wat}, K_{ext}), K_{Proj})$ and b_i^{ext} is the i^{th} bit of the watermark b^{ext} extracted from M^{wat}.

The second expression relies on thresholding the output of the perceptron $Proj_\theta(.)$ the parameters of which (θ) are replaced by $Ext(M^{wat}, K_{ext})$ as follows:

$$b_i^{ext} = \begin{cases} 1 & Proj_{Ext(M^{wat}, K_{ext})}(A_i) \geq 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

where A_i is the i^{th} column of $A = K_{Proj}$.

Advantages and inconvenient of this approach : Uchida *et al.*'s scheme has been shown robust to the fine-tuning and pruning attacks. It also provides a quite large watermark capacity. It is however not robust to the watermark detection attacks (Wang and Kerschbaum attack (WKA) and Property Inference Attack (PIA)). Indeed, it changes the weight distribution, especially in the watermarked layer, being easily detectable and erasable without knowing K_{ext} (see [25] for more details). It also not resistant to the overwriting attack. To remove the watermark, an adversary just has to embed his/her own watermark b_{attack} in M^{wat} using his own secret key matrix A_{attack} of same or greater dimension than A. $A_{attack}w^{wat}$ being a simple linear projection, it has a low linear mapping capacity and a randomly generated A_{attack} is likely to be very similar to the original matrix A for some of the watermark bits; bits an adversary will overwrite with his own ones.

2.2.2 Wang *et al.* scheme (RIGA)

RIGA (for Robust whIte-box GAn watermarking), proposed by Wang *et al.* in [22], is another static scheme in an effort to overcome the weaknesses of [14] against WKA, PIA and OWA attacks.

The main idea of RIGA is to use as as projection function a DNN $Proj_\theta^{DNN}$ instead of the Uchida *et al.* perceptron $Proj_\theta$, and to associate it with another neural network F_{det} to preserve the distance between the distributions of the

watermarked and the nonwatermarked weights. Regarding its features extraction function, RIGA is close to the one of Uchida *et al*. Rather than taking the whole mean of the weights of given layer $\mathbf{w}$, it only takes a subset of $\mathbf{w}$: $\mathbf{w}_s$. Another difference stands in the way $Proj_\theta^{DNN}$ is trained. Instead of using $(A_i, b_i)_{i=1...l}$ as in [14] and then minimize the distance between the perceptron parameters θ and $\mathbf{w}_s$, RIGA scheme trains $Proj_\theta^{DNN}$ with the following dataset $((\mathbf{w}_s^{wat}, b), (\mathbf{w}_s, b_r))$, where: $\mathbf{w}_s^{wat}$ and $\mathbf{w}_s$ are the extracted features from the watermarked moel M^{wat} and the original model M, respectively; b is the owner watermark and b_r is a random watermark. Their idea behind using such a dataset is to ensure that $Proj_\theta^{DNN}$ provides b only for $\mathbf{w}_s^{wat}$ or a random watermark, otherwise. Indeed, training $Proj_\theta^{DNN}$ only with $(\mathbf{w}_s^{wat}, b)$ may lead to a trivial function that ignores the input, that is to say, $Proj_\theta^{DNN}$ will map any of its input to b.

Formally, the RIGA watermarking algorithm follows the same process as any white box algorithm (see section 2.2). Its features extraction function works in two steps:

- As in [14], it calculates the mean values $\overline{W}_{ijk} = \frac{1}{n}\sum_{h=1}^{n} W_{ijkh}$, i.e. the average value of filters' coefficients at the same position, getting $\overline{\mathbf{W}} \in \mathbb{R}^{s\times s\times d}$ and flatten $\overline{\mathbf{W}}$ into the vector $\mathbf{w} \in \mathbb{R}^v$ with $v = s \times s \times d$.
- From $\mathbf{w}$, secretly extract the features such as : $\mathbf{w}_s = Ext(M, K_{ext}) = \mathbf{w}K_{ext}$ where K_{ext} is a secret key defined as a selection matrix of size $|\mathbf{w}_s| \times |\mathbf{w}|$ with a 1 at position $(i, 1)$, $(i + 1, 2)$ and so forth, and 0 otherwise, where i is the start index of a layer.

Regarding the projection function, as stated above, this one corresponds to a DNN $Proj_\theta^{DNN}$, the parameters θ must be trained. $Proj_\theta^{DNN}$ has as an input layer size equals to $|\mathbf{w}_s|$ and an output layer of the size of the watermark. As in our reformulation of the Uchida *et al*. scheme, RIGA will ensure that $Proj_\theta^{DNN}(Ext(M^{wat}, K_{ext})) = b$. Let us recall that K_{ext} is the key that control the feature extraction. It is important to notice that whereas the key K_{Proj} that allows the watermark projection/extraction in [14] is A (see (7)), for RIGA it is θ, the set of parameters of $Proj_\theta^{DNN}$, that constitutes the projection secret key (i.e., $K_{Proj} = \theta$).

Another big difference between RIGA and [14] stands in the distance measure d. Whereas Uchida *et al*. embeds a sequence of bits b, with RIGA it is possible some other kinds of data, the outputs of $Proj_\theta^{DNN}$ being adjustable. It is thus possible to embed an image as watermark. Because of that too, one can change the cross-entropy distance of the Uchida *et al*.'s scheme by the Mean Squared Error (MSE).

From this standpoint, we can refine the RIGA watermark regulizer term such as:

$$E_{wat} = d(Proj_\theta(Ext(M^{wat}, K_{ext})), b) + d(Proj_\theta(Ext(M, K_{ext})), b_r) \tag{12}$$

As stated previously, to counteract the WKA and PIA attacks, RIGA includes another DNN model F_{det} the role of which is to maintain the distribution of the watermarked weights as close as possible to the one of the nonwatermarked weights. To do so, F_{det} is trained with the following dataset $((w^{wat}, 1), (w, 0))$. In that way, F_{det} is a binary classifier that discriminates between watermarked and nonwatermarked weights. The loss function used to train F_{det} is the binary cross entropy E_{det} given as:

$$E_{det} = log(F_{det}(w^{wat})) + log(1 - F_{det}(w)) \tag{13}$$

At the end, the target model M will be RIGA watermarked into M^{wat} through a fine-tuning operation under the global loss defined as:

$$E = E_0(X_{Train}, Y_{Train}) + \lambda_1 E_{wat} - \lambda_2 \log(F_{det}(w^{wat})) \tag{14}$$

where, as previously, E_0 serves the primary task of the model, E_{wat} ensure the watermark embedding in w^{wat} from M^{wat} while F_{det} is used to preserve the weight distribution of M^{wat}. Notice that in (14), F_{det} is a fixed model. Indeed, it is trained under E_{det} after each training batch of M^{wat} by E. Adding the term $-\lambda_2 \log(F_{det}(w^{wat})$ to E, makes the learning of the model M^{wat} generates weights that maximize $F_{det}(w^{wat})$ or, more clearly, weights of M^{wat} that are close to those of M. This is equivalent to a GAN based strategy, where M^{wat} is the generator and F_{det} the discriminator that classifies weights given in input as $'0'$, nonwatermarked, or $'1'$, watermarked. Thus F_{det} is trained to learn to separate the watermarked weights from the nonwatermarked ones. On the other hand, M^{wat} is thus trained not only to achieve its classification and embedding task but also to fool F_{det}. Also notice that to ensure the correctness of F_{det}, the authors of RIGA recommend clamping its parameters and to feed it with the sorted weights of M^{wat}.

Advantages of this approach : RIGA has been shown resistant to the fine-tuning, pruning and overwriting attacks. It is also robust to the detection attacks (PIA and WKA attacks) due to the use of F_{det}. The security of RIGA depends on the knowledge of the parameters of $Proj_\theta$ that plays the role of RIGA secret key ($K_{Proj} = Proj_\theta$). Thus, the more complex $Proj_\theta$; with the addition of more layers; the more difficult it is for an attacker to guess the neural network $Proj_\theta$. Moreover, due to the strong fitting ability of neural networks, $Proj_\theta$ can be mapped to a wide range of data types of watermark messages b, e.g., a binary string or even a 3-channel image. For different types of b, one will have however to choose the appropriate distance measure d accordingly.

Inconvenients of this approach : RIGA consists in training three models at the same time: M^{wat}, $Proj_\theta$ and F_{det}. Such a training requires to find the right hyperparameters to make them converge at the same time; a task that is sometimes hard to implement in the case of very deep models. The complexity of RIGA is also not negligeable. Indeed, preserving the distribution of watermarked and nonwatermarked weights by training F_{det} and adding the term $-\lambda_2 \log F_{det}(w^{wat})$ to E is equivalent to the addition of the term $||w - w^{wat}||$ to E. This term makes the watermarking needs several epochs to converge because, based on E_{wat}, RIGA wants both $Proj_\theta(w^{wat}) == b$ and $Proj_\theta(w) == b_r$ while minimizing $||w - w^{wat}||$, which is bit contradictory. Therefore, under the constraint of a large number of epochs, training three deep neural networks induces a watermarking process of very high computation complexity. Another important point is that the parameters of 500 nonwatermarked models are required to train the F_{det} model, as demonstrated in the experimental section of [22]. This extremely costly from the computation complexity point of view. For instance, considering the ImageNet-1k dataset, it takes about 14 days to complete 90-epochs for training ResNet-50 on a NVIDIA M40 GPU [26].

2.2.3 Rouhani *et al* scheme (DeepSigns)

Unlike prior works that directly embed the watermark into the static content of the target DNN model, i.e., its weights, DeepSigns works by embedding a watermark into the probability density function (pdf) of the activation maps of various layers. The watermarking process is simultaneously data- and model-dependent, meaning that the watermark is embedded into the dynamic content of the DNN that can only be triggered by passing specific input data to the model. As previously, let us consider a target model M devoted to a classification problem with Y classes. The first step of Deepsigns is to generate a set of trigger data that will be used, in a second step, for watermarking the target DNN model. In its principle, DeepSigns watermarks the layers of M in an independent way. The trigger set generation for the l^{th} layer of M consists of the following steps:

1. Fed the model with all training data (X_{train}) to get the corresponding activation maps of the layer l $\{f^l(x, w)\}_{x \in X_{train}}$, where $f^l(x, w)$ belongs to $\mathbb{R}^m$, m being the dimension of the activation maps (e.g. the number of neurons in the case of a FC layer).
2. Define a Gaussian Mixture Model (GMM) that contains S Gaussians to model and fit the activation maps from step 1. Each sample x of the training set belongs thus to a classification class y_{train} (i.e., a training label or ground-truth labels of the training samples) and to a GMM class y_{gmm}. Notice that the authors of DeepSigns set the value S equals to to the number of classes Y of the classification task the target model M performs (e.g., in the case of Mnist or Cifar10, S equals to 10).
3. DeepSigns then secretly selects s Gaussians from the mixture; Gaussian identified by their indices T where $|T| = s$ (e.g., if $S = 10$, $T = \{1, 5, 7\}$ then $s = 3$); and get their respective Gaussian mean values $\{\mu_l^i\}_{i \in T}$ where $\mu_l^i \in \mathbb{R}^m$. These mean values will be used for the insertion of the watermark.
4. As next step, DeepSigns secretly selects a subset of the input training data: (X_{key}, Y_{key}), input samples that belong to the secretly selected Gaussian classes (X_{key}, Y_{key}) and the training labels of which also belong to T, more clearly:

$$(X_{key}, Y_{key}) = \{(x, y_{train}) \in (X_{train}, Y_{train}) / (y_{gmm} \& y_{train} \in T)\} \tag{15}$$

   (X_{key}, Y_{key}) is referred as the trigger set of the layer l and will be used for both watermark insertion and verification.

As stated earlier, it is possible to reformulate DeepSigns in the same common framework as we did for RIGA and [14]. Regarding the feature extraction function $Ext(M, K_{ext})$ of DeepSigns, it can be expressed as follows:

$$Ext(M, K_{ext}) = \{Ext^i(M, K_{ext})\}_{i \in T} \tag{16}$$

with

$$Ext^i(M, K_{ext}) = \{f^l(x, w)\}_{x \in X_{key} \& y_{train} = i} \tag{17}$$

where $Ext^i(M, K_{ext})$, with $i \in T$, is the set of activation maps of the input data belonging to X_{key} with their training labels equals to i. As defined, the extraction function of DeepSigns depends on the secret key K_{ext}, which corresponds to the trigger set.

Based on the trigger set, DeepSigns embeds in the secretly selected GMM classes a set of binary random watermarks $b = \{b_i\}_{i \in T}$, where $b_i \in \{0, 1\}^N$ is a binary watermark of size N bits and which will be inserted in the i^{th} GMM class. This embedding process is similar to the one in Uchida *et al.* scheme except that it uses $\{f^l(x, w)\}_{x \in X_{key}}$ instead of the vector of weights $\mathbf{w}$. Again, this insertion process can be reformulated with the help of a set of s perceptrons as

follows: Let us consider the i^{th} secretly selected GMM class. We can define a single perceptron $Proj_{\theta_i}$ with Sigmoid as activation function and θ_i as set of parameters with as purpose the embedding of the watermark b_i.

Each projection perceptron $Proj_{\theta_i}$ can be seen as trained over the dataset $\{(A_j, b_{ij})\}_{j=1...N}$ where A is a secret matrix ($K_{Proj} = A$) of size $m \times N$, A_j denotes the j^{th} column of A and b_{ij} is the j^{th} element of b_i. Note also that, with DeepSigns, all $Proj_{\theta_i}$ are trained over the same secret matrix A but with different labels b_i. The distance used between $Proj_{\theta_i}(A)$ and b_i is the cross entropy, considering a batch size equals to N. The parameters of $Proj_{\theta_i}$ are initialized with the vector $\mathbf{w}$, herein the parameters of $Proj_{\theta_i}$ are initialized with μ_l^i for $i \in T$ with the idea that the parameters which minimize the distance $d(Proj_{\theta_i}(A), b_i)$ stay close to the activation maps of the nonwatermarked model.

Based on the above statements, the watermarking regularization term of Deepsigns E_{wat} can be reformulated such as:

$$E_{wat} = \sum_{i \in T} (d(Proj_{\theta_i}(A), b_i) + ||\theta_i - \underset{x \in X_{key} \& y_{train} = i}{\text{mean}} (f^l(x, w^{wat}))||) \tag{18}$$

To prevent the parameters $\{\theta_i\}_{i \in T}$ from converging to the means of the GMM classes that have not been selected for watermarking, i.e., $\{\mu_i^l\}_{i \notin T}$, DeepSigns exploits a regularization term E_{sep}, see (19), to maximize their distance.

$$E_{sep} = \sum_{i \in T, j \notin T} ||\theta_i - \mu_j^l|| \tag{19}$$

Therefore, for DeepSigns, the loss function E to embed the watermark and preserve the accuracy of the watermarked model is given by:

$$E = E_0(X_{key}, Y_{key}) + \lambda_1(E_{wat}) - \lambda_2(E_{sep}) \tag{20}$$

The watermark extraction process of DeepSigns stands on the two following steps:

- Collect the activation maps corresponding to the selected trigger set (X_{key}, Y_{key}).
- Compute the statistical mean values of the activation maps of the selected trigger images. Take these mean values as an approximation of the parameters $\{\theta_i\}_{i \in T}$ where the watermarks $\{b_i\}_{i \in T}$ are assumed to be embedded. These mean values together with the owner's private projection matrix A are used to extract the watermarks $\{b_i^{ext}\}_{i \in T}$ following the procedure in (21).

$$b_i^{extracted} = HT(Proj_{\underset{x \in X_{key} \& y_{gmm} = i}{\text{mean}} (f^l(x, w^{wat}))}(A)) \tag{21}$$

where HT is the Hard Thresholding operator that maps the values that are greater than 0.5 to 1 and the values smaller than 0.5 to 0.

Advantages of this approach : As defined, DeepSigns requires a trigger set, a secret subset of the training data, to embed (resp, extract) the watermark in (resp, from) the pdf of the activation maps associated to this trigger set. By doing so, it does not disturb too much the weights' distribution contrarily to Uchida *et al*. scheme. In addition, it does not need the training of another neural network as in the case of RIGA that exploits the model F_{det} to preserve the weight distribution. The security of DeepSigns is higher than [14] and other static methods since it relies not only on the secret matrix A but also on the trigger set (X_{key}, Y_{Key}) and on the secretly selected GMM classes to watermark (i.e. T indices). Another advantage of DeepSigns is that it is resistant to the fine-tuning, pruning and overwriting attacks but only for small insertion capacity.

Inconvenient of this approach : From [20], it is not clear if DeepSigns finetunes the target model only with the trigger set (X_{key}, Y_{key}) which represents 1% of the training set. By doing so, it is possible that the watermarked model overfits the trigger set with as consequence a loss of precision on the testing set. We experimentally noticed this accuracy drop. In the following experiments, we have assumed that to preserve accuracy and to reach the performance of DeepSigns given in [20], the target model is fine-tuned over the entire training set changing the regularization terms depending on if the input data belongs to the trigger set or not. In other words, we trained the single perceptrons $\{Proj_{\theta_i}\}_{i \in T}$ only on the activation maps of the trigger set data. And so, the embedding loss function can be written as:

$$E = E_0(X_{Train}, Y_{Train}) + \lambda_1(E_{wat}) - \lambda_2(E_{sep}) \tag{22}$$

In that way, we well preserve the precision of the watermarked model but on at the price of a reduced insertion capacity. This can be explained by the fact that since the trigger set is a subset of the training set, and represents only 1% of the training set, so in order that the model preserves its accuracy, it is better to classify the 99% of the data (which are similar in terms of distribution to the trigger set data) than to approximate the trigger set activation maps to the the parameters $\{\theta_i\}_{i\in T}$. This can be explained by the fact that since the trigger set is a subset of the training set and represents only 1% of the training set, it is preferable to classify the remaining 99% of the data (which have a similar distribution to the trigger set data) than to minimize the distance between the trigger set activation maps and the parameters $\{\theta_i\}_{i\in T}$ in order to maintain the model's accuracy.To solve this problem, and as we will see in section 3 devoted to our solution, we propose the use of a trigger set with a distribution different to the ones of the training and/or finetuning data. This solution allows the watermarked model both to insert a robust watermark and preserve its accuracy.

The main weakness of DeepSigns stands on its insertion capacity. Increasing the length of watermark will make it nonresistant to FTA and OWA attacks. To embed the watermark into the pdf distribution of DNN activation maps, DeepSigns takes advantage of the fact that DNN models possess nonconvex loss surfaces with many local minima that are likely to yield to very close accuracies [27]. Although, using GMM clustering provides a reasonable approximation of the activation distribution obtained in hidden layer [28], this one will necessarily be different for distinct local minima. If two data belong to the same class for a given local minimum, they are not necessarily in the same GMM class for another local minima. One can also question the interest in using a clustering for large capacity. Let us recall first that for DeepSigns the perceptrons or equivalently the projection functions$\{Proj_{\theta_i}\}_{i\in T}$ are trained to find the parameters $\{\theta_i\}_{i\in T}$ that minimize the distance $\{d(Proj_{\theta_i}(A), b_i)\}_{i\in T}$. Even though, $\{\theta_i\}_{i\in T}$ are initialized by the GMM means $\{\mu_l^i\}_{i\in T}$, one can see that there are no constraints in the loss E_{wat} of DeepSigns to ensure $\{\theta_i\}_{i\in T}$ remain close or similar to $\{\mu_l^i\}_{i\in T}$. It is imposed that the watermarked means move away from the nonwatermarked means $\{\mu_l^i\}_{i\notin T}$. For large capacities, such a distance can be important. Therefore, there is not much interest in selecting the trigger set based on the GMM clustering since no constraints are applied on the parameters $\{\theta_i\}_{i\in T}$ and that each local minima will provide a new clustering. Large distance between $\{\theta_i\}_{i\in T}$ and $\{\mu_l^i\}_{i\in T}$ may also cause a problem regarding the 1% ratio between the trigger and training sets. A couple of finetuning epochs can be enough to bring closer $\{\theta_i\}_{i\in T}$ and $\{\mu_l^i\}_{i\in T}$ and erase the watermark.

At least, another unclear point in [20] is that Deepsigns activation maps $f_l(x, w)$ are said to be the outputs of activation functions like sigmoids or Relu, respectively defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$
$$Relu(x) = max(0, x)$$

Due to the fact, the parameters $\{\theta_i\}_{i\in T}$ that minimize the distance $\{d(Proj_{\theta_i}(A), b_i)\}_{i\in T}$ are vectors with positive or/and negative components (see above, A is derived from a normal distribution $N(0, 1)$ and b is random binary sequence) and that the outputs of the Sigmoid or Relu activation are always positive, there is a high risk the activation maps $\{mean_{x\in X_{key}\&y_{train}=i}(f_l(x, w))\}_{i\in T}$ do not converge to the parameters $\{\theta_i\}_{i\in T}$ (see E_{wat}, (18)). To avoid this problem and to retrieve the performance of DeepSigns given in [20], we decided to use the activation maps of the layer's weighted sum with the input without using an activation function.

3 DICTION : a novel dynamic white box watermarking scheme

Based on the above unified white box watermarking framework, we introduce DICTION a novel dynamic scheme. It inserts a watermark into the activation maps of some layers of the target model. Its originality is twofold. In a first time, to preserve model weights and activation map distributions, it takes advantage of a trigger set issued from a latent space, as in the context of GAN generators [21] and Variational AutoEncoder (VAE) decoders [29]. That is to say, the trigger set samples correspond to a set of images randomly generated accordingly to a normal distribution of a given mean and standard deviation. Indeed, when working with a trigger set constituted of training images as DeepSigns, the watermark insertion will at least impact the model activation maps. More clearly, if some images of a given label "Y" have been selected as part of the trigger set, their activation maps will have a particular distribution to respond to the watermarking detection system (e.g., (18) in DeepSigns) while the activation maps for other images of label "Y" (the majority of this class) will be shaped to ensure the accuracy of the model. Additionally, multiple epochs may be necessary to find a local minimum that minimizes the global loss function to watermark the target model, especially if the trigger set activation maps differ a lot from the training set activation maps for model accuracy. There is also a high probability that, after fine-tuning, the trigger set activation maps converge to those of the training set activation maps with as consequence a watermark no longer detectable. With a trigger set defined by a latent space, we are sure that the activation maps of each class have the same distribution, since we don't force outliers on the training set images. In

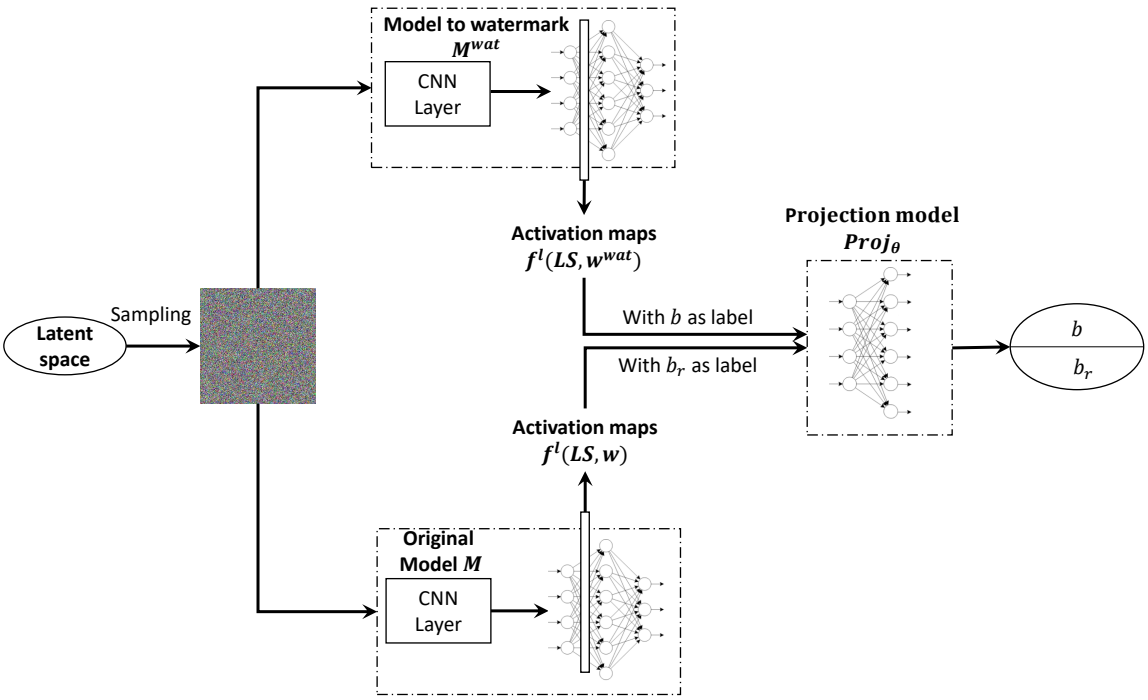


Figure 2: DICTION workflow: the trigger set activation maps of the original model $f^l(LS, w)$ and of the model to watermark $f^l(LS, w^{wat})$ (the generators) are fed by data sampled from the latent space; the projection function $Proj_\theta$ (the descriminator) learns to map $f^l(LS, w)$ and $f^l(LS, w^{wat})$ to a random binary string b_r for the original model and to the watermark b for the watermarked model, respectively.

other words, we don't add any additional constraints on the distribution of a class activation maps in the training data except to preserve the accuracy.

In second, to achieve a good tradeoff between insertion capacity and watermark robustness to PA, FTA, OA, and WKA, as well as PIA, the projection function of DICTION is defined as neural network that learns to map the activation maps of the watermarked model M^{wat} to a watermark b and the activation of the nonwatermarked model to a random watermark b_r. More clearly, such a projection function is comparable to the discriminator of a GAN model trained to distinguish, for samples in the latent space, the activation maps produced by a watermarked model from activation maps of a nonwatermarked model. In other words, and as illustrated in Fig. 2, we have:

- two generators $(f^l(LS, w^{wat}) and f^l(LS, w))$, where: w^{wat} are the weights of the watermarked model M^{wat} and w those of the target model M; l is the index of a given layer and LS is a sample from the latent space.
- and the discriminator that learns to map the output of these generators into different watermark spaces.

In contrast to DeepSigns, where the projection function is a single perceptron trained on a static secret matrix A, DICTION depends thus on a neural network. As we will see, such a change makes DICTION much more performant than DeepSigns. Let us formulate our scheme as we did in Section 2.2 for the state-of-the-art white box schemes, through its watermarking and embedding regularization terms: E_{wat} and E, respectively. Let us also define a latent space with a normal distribution $N(\mu, \sigma)$, of mean (μ) and of standard deviation (σ), from which our image trigger is derived. From this standpoint, the feature extraction function for a given target model M is such as:

$$Ext(M, LS, K_{ext}) = f^l(LS, w) * Z \tag{23}$$

where: LS is a sample from our latent space (i.e. an image the pixel values of which follows a normal distribution – see above) ; $K_{ext} = \{l, Z\}$ is the secret extraction key composed with l the index of the layer to be watermarked and Z a permutation matrix that is used to secretly select a subset of features from $f^l(LS, w)$ and order them in a secret fashion.

As stated previously and as shown in Fig. 2, our projection function $Proj_\theta^{DNN}$ is a neural network. It takes as input the extracted features $Ext(M, LS, K_{ext})$ to output the watermark b. The size of its input and output are thus equal to the

size of extracted features, $|Ext(M, LS, K_{ext})|$, to the watermark size $|b|$. To embed b in the secretly selected layer l of the target model, the watermarking regularization term of our scheme is defined as:

$$E_{wat} = d(Proj_{\theta}^{DNN}(Ext(M^{wat}, LS, K_{ext})), b) + d(Proj_{\theta}^{DNN}(Ext(M, LS, K_{ext})), b_r) \quad (24)$$

where: M^{wat} is the watermark version of M; b_r is a random watermark; d is a watermark distance measure. b_r is used to ensure that $Proj_{\theta}^{DNN}$ projects the desired watermark b only for M^{wat}. Note that d depends on the type of the watermark. For instance, in the case b is a binary sequence, one can use cross-entropy whereas in the case it is an image, the pixel mean squared error is more relevant. Our global embedding loss is given as :

$$E = E_0(X_{train}, Y_{train}) + \lambda E_{wat} \quad (25)$$

where: λ is here to adjust the tradeoff between the original target model loss term E_0 and the watermarking regularization term E_{wat}; (X_{train}, Y_{train}) is the training set. As it can be seen, E_{wat} does not depends on (X_{train}, Y_{train}) but on the trigger set only. More clearly, the parameters of M^{wat} are updated accordingly to the following regularization term from E:

$$E_{\mathbf{w}^{wat}} = E_0(X_{train}, Y_{train}) + \lambda d(Proj_{\theta}(Ext(M^{wat}, LS, K_{ext})), b) \quad (26)$$

By doing so, the weights $\mathbf{w}_{wat}$ of M^{wat} are updated such as they keep the accuracy of the original target model and minimize the distance between the projections of the activation maps and the watermark b. On their side, the parameters of $Proj_{\theta}^{DNN}$ are updated based on the following regularization terms in E:

$$E_{\theta} = \lambda d(Proj_{\theta}^{DNN}(Ext(M^{wat}, LS, K_{ext})), b) + d(Proj_{\theta}(Ext(M, LS, K_{ext})), b_r) \quad (27)$$

As previously mentioned, the projection model $Proj_{\theta}^{DNN}$ serves as a discriminator that associates the projection of the trigger set activation maps of M^{wat} to b and those of M to b_r. One can notice from (26) and (27) that the term $d(Proj_{\theta}(Ext(M^{wat}, LS, K_{ext})), b)$ is shared in the update of M^{wat} and $Proj_{\theta}^{DNN}$. This enables the optimal compromise between the parameters w^{wat} and θ so that the projection of the activation maps is close to b, with no impact on the accuracy, and is unique for M^{wat}. In practice, the parameters of M^{wat} and $Proj_{\theta}^{DNN}$ are trained alongside the embedding process to enable fast convergence.

The watermark detection process works as follows: Let us consider a suspicious watermarked model M^*. Based on the features extraction key $K_{ext} = \{l, Z\}$, the projection key $K_{proj} = Proj_{\theta}^{DNN}$ and the latent space $N(\mu, \sigma)$, the watermark extraction is such as :

$$b^{ext} = HT(\underset{LS \in N(\mu,\sigma)}{\text{mean}} Proj_{\theta}^{DNN}(Ext(M^{wat}, LS, K_{ext})) \quad (28)$$

where HT is the Hard Thresholding operator at 0.5. In contrast to a fixed subset of the training set as used in DeepSigns (see (21)), in DICTION extraction, you can sample an endless number of latent space images. By doing so, the watermarked model and projection function are extremely resistant to attacks such as fine-tuning, pruning, and overwriting. This is due to the fact that the watermarked model and projection function overfit on the latent space rather than a few images. In addition, it decreases the storage complexity of the detection as one just has to record the latent space parameters (i.e. its mean and standard deviation) rather than storing a trigger constituted of images, the number and size of which might be quite big in some applications. Notice also that the use of a latent space as a trigger set allows preserving the distribution of weights and activation maps of the watermarked model. Moreover, it avoids the use of another neural network like F_{det} in RIGA scheme which needs to find a good hyperparameterization (e.g., clamping weights -see Section 2) and increasing thus the computational complexity of the watermarking algorithm.

4 Experimental Results

We evaluated DICTION in terms of impact of on model performance, as well as of robustness against to three watermark removal techniques (overwriting, fine-tuning, and weights pruning) and to the watermark detection attacks proposed by [25] and PIA. For all attacks we have considered the strongest possible threat models where the attacker has access to or has the knowledge of the training data and of the layer that have been watermarked.

Table 1: Benchmark neural network architectures. 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. 512FC is a fully-connected layer with 512 output neurons, BN is the batch normalization. ReLU and Sigmoid have been used as activation functions in all benchmarks.

Benchmark ID	Dataset	Baseline Acc	DL Model Type	DL Model Architecture
1	MNIST	98.54%	MLP	784-512FC-512FC-10FC
2	CIFAR10	86.81%	CNN	3*32*32-32C3(1)-32C3(1)-MP2(1)-64C3(1)-64C3(1)-MP2(1)-512FC-10FC
3	CIFAR10	90.85%	ResNet18	Please refer to [31]
4	MNIST	98.99%	LeNet	1*28*28-24C3(1)-BN-24C3(1)-BN-128FC-64FC-10FC

4.1 Datasets and Models

The following experiments have been performed on two well-known benchmark image datasets in the context of image classification: : CIFAR10 [30] and MNIST [23]. We also used four distinct neural network architectures, three of them being also experimented in DeepSigns [20] and LeNet in RIGA [22]. Table 1 summarizes their topologies depending on the target dataset and the corresponding achieved accuracy. The implementations of DeepSigns and DICTION along with their configurations are publicly available at: `https://github.com/Bellafqira/DICTION`.

4.2 Benchmark and Watermark Setting

In all our experiments, we embedded a watermark of 256 bits, which corresponds to the size of an ownership identifier or of a binary stream generated by the hash function SHA256 [32]. The watermark is embedded in the second to last layer, as for DeepSigns. We use the bit error rate (BER) as the metric to measure the distance between original and extracted watermarks. BER is calculated as

$$BER(b^{ext}, b) = \frac{1}{n} \sum_{i=1}^{n} I[|b_i^{ext} - b_i| > 0.5] \tag{29}$$

where b^{ext} is the extracted watermark, I is the indicator function, and n is the size of the watermark.

We use simple 3-layer fully-connected neural networks as the architecture for the projection function in our experiments. The normal distribution $N(\mu, \sigma) = N(0, 1)$ serves as a latent space to embed the watermark. We use Adam optimizer with the learning rate 10^{-1}, batch size 100 and weight decay 10^{-4} to train the projection function. We set $\lambda = 1$ in (25). We used 30 epochs for each benchmark to embed the watermark.

Regarding the implementation of DeepSigns [20], we used also the second-to-last layer for embedding. λ_1 and λ_2 have been set to 1 to obtain a BER equals to zero, except for LeNEt since the size of its activation maps ("64") is smaller compared to other architectures whose activation maps sizes are "512" (see Table 1). The number of watermarked classes s equals to 2 and N to 128 (i.e., a watermark of 256 bits), the watermark projection matrix A is generated based on standard normal distribution $N(0, 1)$. The trigger set presents 1% of the training data. The reported BER and the detection rate values of DICTION as well as for DeepSigns are averaged over 5 different runs.

4.3 Watermarked model performance

By definition, the accuracy of the watermarked model shall not be degraded after embedding the watermark. Table 1 summarizes the baseline original models accuracy and Table 2 the accuracy of watermarked models after embedding a watermark of 256 bits. As it can be seen, the accuracies of the watermarked models are very close to those of the the nonwatermarked models by simultaneously optimizing the accuracy of the underlying model, as well as the additive watermark loss functions (E_{wat}) as discussed in Section 3. In some cases, we even observe a slight accuracy improvement of DICTION compared to the baseline. Such an improvement is mainly due to the fact that the additive loss functions (E_{wat}) act as a form of a regularizer during the training phase of the original model. Regularization, in turn, helps the model to avoid over-fitting by inducing a small amount of noise to the model.

4.4 Robustness

We evaluate the robustness of our scheme against three contemporary removal attacks as discussed in Section 1. The potential attacks include parameter pruning, model fine-tuning, and watermark overwriting.

Parameter pruning. We use the pruning approach proposed in [33] to compress the watermarked models. For pruning each layer of a watermarked model, we first set a percentage $\alpha\%$ of the parameters that possess the smaller weight

Table 2: BER and accuracy of DeepSigns and DICTION for the four benchmarks after embedding a watermark of 256 bits

Benchmark ID	**Baseline Acc**	**DeepSigns**		**DICTION**	
		BER	Model accuracy	BER	Model accuracy
1	98.54%	0	98.51%	0	98.61%
2	86.81%	0	86.02%	0	86.62%
3	90.85%	0	90.94%	0	91.15%
4	98.99%	12.5	99.02%	0	99.01%

Table 3: Robustness of DICTION and DeepSigns against the model fine-tuning attack.

Benchmarks	**BM 1**			**BM 2**			**BM 3**			**BM 4**		
Number of epochs	50	100	150	50	100	150	50	100	150	50	100	150
ACC of DICTION	98.57%	98.49%	98.53%	86.61%	86.5%	86.15%	91.11%	90.86%	90.98%	98.68%	98.8%	98.55%
ACC of DeepSigns	97.2%	98.52%	98.54%	86.26%	86.46%	86.89%	91.06%	91.31%	91.13%	99.04%	98.96%	98.91%
BER of DICTION	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
BER of DeepSigns	52.73%	50.78%	51.56%	35.93%	41.79%	40.26%	21.09%	33.59%	36.32%%	53.51%	55.46%	54.29%

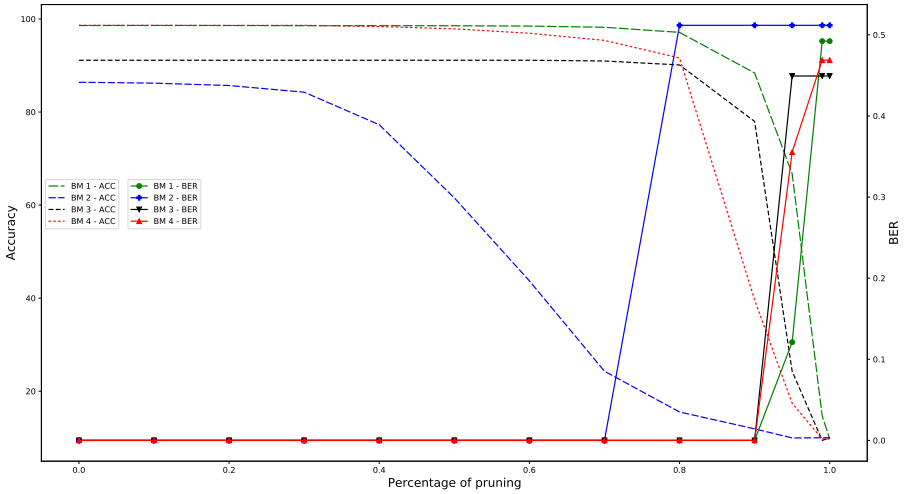
values to zero. Fig. 3 illustrates the impact of pruning on watermark extraction/detection as well as on model accuracy in function of $\alpha\%$. DICTION can tolerate up to 90% and 95% parameter pruning for the four benchmarks, respectively. One can also note that DeepSigns is more vulnerable to pruning for a 256 bits watermark. Moreover, in occasions where pruning the model yields a substantial bit error rate (BER) value, we observe that the attacked watermarked model suffers from a large accuracy loss compared to the baseline. As such, one cannot remove the embedded watermark from the watermarked model without excessive pruning and loss of accuracy compared to the baseline.

Watermark overwriting. Assuming the attacker is aware of the watermarking technique, he may attempt to damage the original watermark by embedding a new watermark in the watermarked model. In practice, the attacker does not have any knowledge about the location of the watermarked layers. However, in our experiments, we consider the worst-case scenario in which the attacker knows where the watermark is embedded but does not know the original watermark information nor the projection function. To perform the overwriting attack, the attacker follows the protocol discussed in Section 3 to embed a new watermark of either the same size as the original watermark (i.e., a 256-bit message) or of a larger size (i.e. a 512-bit message) using different or same latent space. Table 4 summarizes the results we obtained against of watermark overwriting for all four benchmarks. As shown, DICTION is robust against the overwriting attack and can successfully detect the original embedded watermark in the overwritten model while, with DeepSign, inserted watermarks are completely erased.

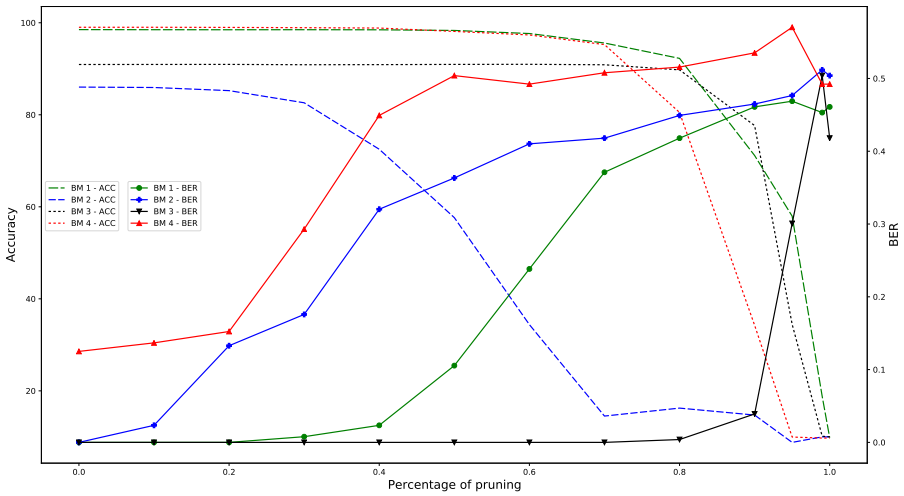
Model fine-tuning. Fine-tuning is another form of transformation attack that a third-party user might use to remove the watermark. To perform this attack, one needs to fine-tune the watermarked model using the original training data with the conventional cross-entropy loss function, excluding thus the watermarking loss functions E_{wat}. Note that fine-tuning deep learning models makes the underlying neural network converge to another local minimum that is not necessarily equivalent to the original one in terms of the prediction accuracy. Table 3, summarizes the impact of fine-tuning on the watermark detection rate across all four target models. In our experiments, to perform the model fine-tuning attack, we multiply the last learning rate used at the training stage by a factor of 10 and then we deviate it by 10 after 20 epochs. This choice allows the model to look for a new local minima. As demonstrated in Table Table 3, DICTION can successfully detect the watermark even after fine-tuning the deep neural network for many epochs. That is not the case of DeepSigns.

Table 4: BER of DeepSigns and DICTION for the three benchmarks after overwriting the watermarked model by a watermark of 256 bits and 512 bits with different keys

Benchmark ID	**DICTION**		**DeepSigns**	
	256	512	256	512
1	0	0	42.70%	43.33%
2	0	0	36.71%	42.72%
3	0	0	09.37%	15.23%
4	0	0	47.65%	41.40%

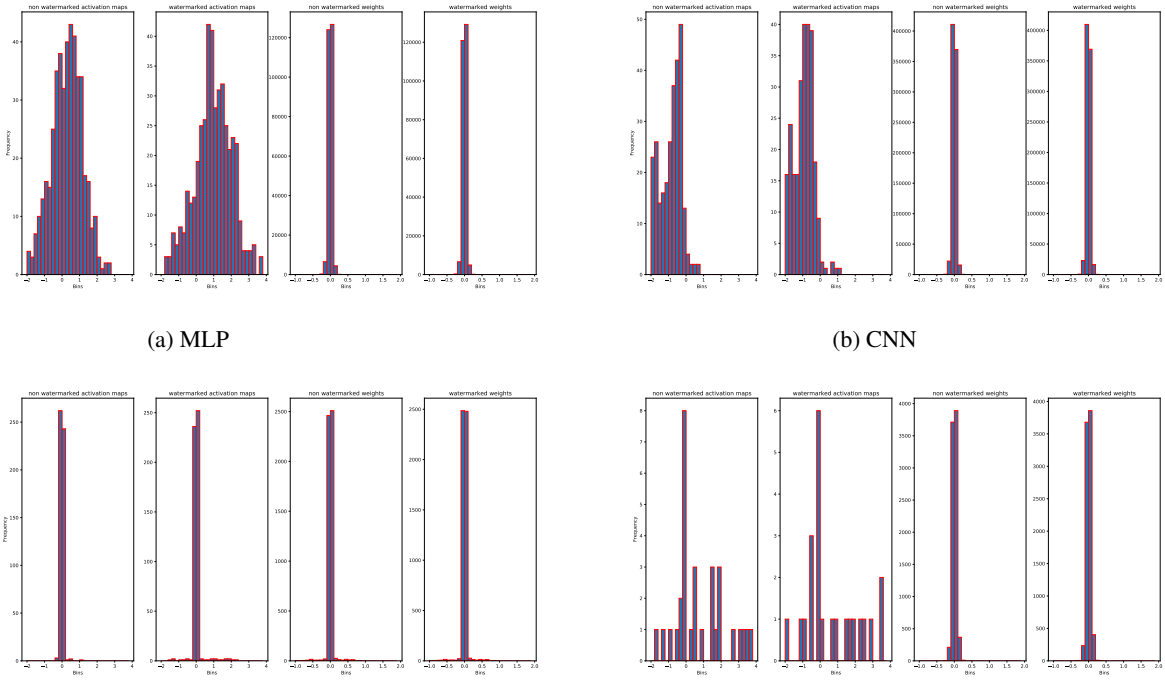


(a) Diction



(b) DeepSigns

Figure 3: Robustness of DICTION and DeepSigns against parameter pruning for the Four benchmarks. The doted points present the BER and the lines the accuracy of the models in function of percentage of pruning "α".



(a) MLP

(b) CNN

(c) Resnet18

(d) LeNet

Figure 4: Distribution of the activation maps and model weights for the four benchmarks. DICTION preserves the distribution spanned by the model while robustly embedding the watermark. Note that the range of activation maps is not deterministic in different models and cannot be used by malicious users to detect the existence of a watermark.

Watermark Detection Attack. The embedding of the watermark should not leave noticeable changes in the probability distribution spanned by the watermarked model. DICTION satisfies this security requirement by preserving the distribution of weights/activation maps. Fig. 4 illustrates the histogram of the activation maps in the embedded layer in the watermarked model and the same layer in the nonwatermarked model for the four benchmarks. We also considered the property inference attack (PIA) whose principle is to train a watermark detector $Dect$ on several watermarked and nonwatermarked neural network models. Again, we assume the worst-case scenario where the attacker knows the training data, the exact model architecture of M^{wat}, and the feature extractions key. This threat model is indeed overly strong, but we take it into consideration to demonstrate the effectiveness of our watermark scheme. To perform the property inference attack, we considered the LeNet model we trained over 700 watermarked models and 700 nonwatermarked models using the MNIST as dataset. We also generated 100 nonwatermarked and 100 watermarked models as testing set. We tested PIA only on LeNet because of the very high complexity of this attack. It required the training of 800 models, and the watermarking of each of them. The training set for the detector $Dect$ corresponds thus to the watermarking features of each model labeled accordingly to whether or not the model is watermarked. Note that, all generated models were well-trained with a good accuracy and watermarks were correctly embedded. As result, the accuracy of the detector $Dect$ does not exceed 60% on the training and it remains around 50% for the testing after 50 epochs, which is equivalent to a random guess. This detection accuracy shows that DICTION is resistant to property inference attack.

5 Conclusion

In this paper, we presented a unified framework that include all existing white-box watermarking algorithms for DNN models. This framework outlines a theoretical connection between previous works on white-box watermarking for DNN models. From this framework, we derive DICTION, a new white-box Dynamic Robust watermarking scheme which relies on a GAN strategy. Its main originality stands on a watermark extraction function that is a DNN trained using a latent space as an adversarial network. We subjected DICTION to all actual watermark detection and removal attacks and demonstrated that it is much more robust than existing works with lower embedding and extraction complexity. As

future work, we would like to broaden the scope of our testing by incorporating additional machine learning applications, such as natural language processing, and learning strategies, such as federated learning.

References

[1] Oleg Evsutin and Kristina Dzhanashia. Watermarking schemes for digital images: Robustness overview. *Signal Processing: Image Communication*, 100:116523, 2022.

[2] Dalel Bouslimi, Reda Bellafqira, and Gouenou Coatrieux. Data hiding in homomorphically encrypted medical images for verifying their reliability in both encrypted and spatial domains. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2496–2499. IEEE, 2016.

[3] Sapana Rani and Raju Halder. Comparative analysis of relational database watermarking techniques: An empirical study. *IEEE Access*, 10:27970–27989, 2022.

[4] David Niyitegeka, Gouenou Coatrieux, Reda Bellafqira, Emmanuelle Genin, and Javier Franco-Contreras. Dynamic watermarking-based integrity protection of homomorphically encrypted databases–application to outsourced genetic data. In *International Workshop on Digital Watermarking*, pages 151–166. Springer, 2018.

[5] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. Blackmarks: Blackbox multibit watermarking for deep neural networks. *arXiv preprint arXiv:1904.00344*, 2019.

[6] Yuliya Vybornova. Method for copyright protection of deep neural networks using digital watermarking. In *Fourteenth International Conference on Machine Vision (ICMV 2021)*, volume 12084, pages 297–304. SPIE, 2022.

[7] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 159–172, 2018.

[8] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium*, pages 1615–1631, 2018.

[9] Jia Guo and Miodrag Potkonjak. Watermarking deep neural networks for embedded systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[10] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 32(13):9233–9244, 2020.

[11] Lixin Fan, Kam Woh Ng, and Chee Seng Chan. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. *Advances in neural information processing systems*, 32, 2019.

[12] Ryota Namba and Jun Sakuma. Robust watermarking of neural network with exponential weighting. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 228–240, 2019.

[13] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. How to prove your model belongs to you: a blind-watermark based framework to protect intellectual property of dnn. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 126–137, 2019.

[14] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pages 269–277, 2017.

[15] Le Feng and Xinpeng Zhang. Watermarking neural network with compensation mechanism. In *International Conference on Knowledge Science, Engineering and Management*, pages 363–375. Springer, 2020.

[16] Yue Li, Benedetta Tondi, and Mauro Barni. Spread-transform dither modulation watermarking of deep neural network. *arXiv preprint arXiv:2012.14171*, 2020.

[17] Enzo Tartaglione, Marco Grangetto, Davide Cavagnino, and Marco Botta. Delving in the loss landscape to embed robust watermarks into neural networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 1243–1250. IEEE, 2021.

[18] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pages 105–113, 2019.

[19] Jiangfeng Wang, Hanzhou Wu, Xinpeng Zhang, and Yuwei Yao. Watermarking in deep neural networks via error back-propagation. *Electronic Imaging*, 2020(4):22–1, 2020.

[20] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: an end-to-end watermarking framework for protecting the ownership of deep neural networks. In *The 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM*, 2019.

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[22] Tianhao Wang and Florian Kerschbaum. Riga: Covert and robust white-box watermarking of deep neural networks. *arXiv preprint arXiv:1910.14268*, 2019.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Minoru Kuribayashi, Takuro Tanaka, Shunta Suzuki, Tatsuya Yasui, and Nobuo Funabiki. White-box watermarking scheme for fully-connected layers in fine-tuning model. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, pages 165–170, 2021.

[25] T. Wang and F. Kerschbaum. Attacks on digital watermarks for deep neural networks. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2622–2626, May 2019.

[26] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. 100-epoch imagenet training with alexnet in 24 minutes. *arXiv preprint arXiv:1709.05011*, 2017.

[27] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[29] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[30] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55:5, 2014.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] Dian Rachmawati, JT Tarigan, and ABC Ginting. A comparative study of message digest 5 (md5) and sha256 algorithm. In *Journal of Physics: Conference Series*, volume 978, page 012116. IOP Publishing, 2018.

[33] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.