Bachelor thesis, Institute of Computer Science, FU Berlin,
ID Management

Practical Evaluation of Neural Network Watermarking Approaches

Tim von Känel

timv94@zedat.fu-berlin.de
Matriculation Number: 5106513
Advisor: Franziska Boenisch
1. Supervisor: Prof. Dr. Marian Margraf
2. Supervisor: Prof. Dr.-Ing. habil. Gerhard Wunder

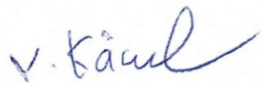
Berlin, June 24th, 2021

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 24.6.2021

Tim von Känel



Abstract. While the influence of deep learning models on real-world applications increases, one concern is on how to verify the model's ownership after their distribution. An approach that is widely used in imagery and video is to use a watermark. In recent years multiple approaches have been made to embed watermarks into neural networks, yet there has been little work on their evaluation. This work implemented five of the most promising algorithms, outlined metrics for their comparison, and evaluated them against each other.

Contradictory to the claims of their authors, major flaws are found in all but one of the algorithms and it is argued that DeepSigns white-box approach overall showed the best performance.

Contents

1 Introduction 1

2 Watermark Embedding 2
- 2.1 General . . . 2
- 2.2 Background . . . 2
- 2.3 Requirements . . . 4
- 2.4 Methodology . . . 4
- 2.5 Adversarial Frontier Stitching for Remote Neural Network Watermarking . 7
- 2.6 Robust Watermarking of Neural Network with Exponential Weighting . . 9
- 2.7 Piracy Resistant Watermarks for Deep Neural Networks . . . 11
- 2.8 DeepSigns: A Generic Watermarking Framework for IP Protection of Deep Learning Models . . . 14
- 2.9 Robust and Undetectable White-Box Watermarks for Deep Neural Networks 16

3 Experiments 18
- 3.1 Robustness . . . 18
- 3.2 Functionality Preservability . . . 22
- 3.3 Efficiency . . . 22
- 3.4 Secrecy . . . 22
- 3.5 Unforgeability . . . 24
- 3.6 Reliability . . . 24
- 3.7 Fidelity . . . 25

4 Evaluation 27
- 4.1 Robustness . . . 27
- 4.2 Functionality Preservability . . . 28
- 4.3 Efficiency . . . 28
- 4.4 Secrecy . . . 29
- 4.5 Unforgeability . . . 30
- 4.6 Reliability . . . 30
- 4.7 Fidelity . . . 31

5 Conclusion 33

1 Introduction

Deep learning models recently gained a lot of attention due to their success in various fields such as autonomous driving, healthcare, and language translation. While being a powerful tool, a lot of resources are required to train such a network: Hardware, a good and task-specific dataset, and expertise. This makes deep neural networks valuable, which are considered the intellectual property of the model's owner. When the model is licensed to a third party, the owner needs a tool to prove his ownership in order to counter theft and illegitimate use.

Recently multiple approaches have been made to embed watermarks into neural networks, however, there is little work on their evaluation. While fields like computer vision have established benchmarks like ImageNet [19], there still is a benchmark for neural network watermarking missing.

In this work, five of the most promising neural network watermark approaches are implemented and each of them is evaluated empirically on predefined requirements. Afterwards they are compared against each other and the pros and cons of each of the watermarking schemes are outlined. All source code used in the evaluation has been published on GitLab[1].

[1] https://git.imp.fu-berlin.de/private_secure_ml/practical-evaluation-of-neural-network-watermarking-approaches

2 Watermark Embedding

2.1 General

In the following, the entity which embeds the watermark into his model will be called the *owner*, and the entity which illegitimately gained access to the model will be called the *adversary*. An algorithm for watermarking a neural network consists of at least three functions [1]:

1. $K \leftarrow key\_gen()$ - Generates a key K which will be used to watermark the model.
2. $m_{wm} \leftarrow embed(m, K)$ - Embeds the key K into the model m in order to obtain the watermarked model m_{wm}.
3. $b \leftarrow verify(m, K)$, $b \in \{0, 1\}$ - Verifies ownership of the model by returning 1 if model m was watermarked using K and 0 if it wasn't.

Using those three functions the process of embedding a watermark into a model m and using the key K to prove ownership of m can be modeled as:

1. The owner generates a key K using $key\_gen()$.
2. The owner acquires the watermarked model m_{wm} by making use of the function $embed(m, K)$.
3. The adversary obtains the watermarked model m_{wm}.
4. The owner of the model proves his ownership of m_{wm} by successfully using $verify(m, K)$ on the stolen model m_{wm} with his key K, this is called the ownership verification ship.

2.2 Background

The schemes which embed watermarks into neural networks are categorized into black-box and white-box algorithms. They both make assumptions on the access the owner has to the stolen model during the ownership verification phase.

The stronger assumption is made by *white-box* algorithms, they assume that the owner has access to the parameters and the outputs of the model when given an input. White-box algorithms have more tools to work with than block-box algorithms and have therefore the potential to be more powerful. However since most real-world applications are hidden behind an API, it can be impossible for the owner to prove ownership of the model because he has no access to the model's internal state.

The weaker assumption is made by *black-box* algorithms, they assume that the owner only has access to the outputs of the model, this gives him the ability to prove ownership of models even if it's only accessible through an API.

Le Merrer et al. [13] proposed "Adversarial Frontier Stitching", a watermarking algorithm that works in a black-box setting by constructing its key from adversarial examples

generated from the training set. The watermark is embedded by training a pre-trained model on the adversarial examples. This process can be interpreted as "stitching" around the model's decision boundary, giving the algorithm its name.

Namba et al. [16] proposed a black-box algorithm that claims to be resistant against fine-tuning, weight pruning, and query modification attacks. Query modification is a way to make ownership verification impossible for models which are hidden behind an API. For each request to the model, the adversary classifies whether the query contains a key. One way to do this is by training an autoencoder to reconstruct samples from the training set and then inspecting whether the reconstruction loss of the query in question exceeds that of a predefined threshold. If it exceeds the threshold, chances are high that the query is from a different distribution than the training data, which could mean that it's a key, and a random prediction is returned. Therefore, in order to combat query modification, the key set should follow the same distribution as the training set which is the reason why Namba et al. [16] generate their key set by assigning random labels to samples drawn from the training set which differ from their ground truth, this process is called *label change*.

Fine-tuning attacks simply train the model on new data in order to remove its watermark.

Weight pruning is a method further explained in chapter 3.1.2 where the model's parameters are set to zero based on their magnitude. Parameters with low magnitude are expected to have little effect on the prediction and are therefore set to zero before parameters with high magnitude, which minimizes the negative effect on the model's predictive performance.

Namba et al. [16] claims to be robust against weight pruning and fine-tuning attacks by imprinting the watermark with greater force during watermark embedding by using an activation function on the weights of the model which gives the algorithm its name. This function further reduces the influence parameters with low magnitude have on the final prediction and results in a small fraction of the parameters, which have a high magnitude, having a strong influence.

Li et al. [15] proposed a black-box algorithm for embedding *unforgeable* watermarks, which will be called "Piracy Resistant Watermarks". A watermark is *piracy resistant* or *unforgeable* when it's not possible to inject a new watermark into an already watermarked model without deteriorating the model's accuracy in a way that makes it useless. They claim to have achieved piracy resistance by constraining the classification rules during training using a deterministically generated pixel-pattern. When the model is trained on a new watermark and without the pixel-pattern applied to its training set, the predictive performance of the model quickly declines due to the training on new classification rules.

Rouhani et al. [17] proposed DeepSigns, a framework whose algorithm differs in white and black-box settings. In this work, only the white-box approach of DeepSigns is evaluated, which uses the fact that there are multiple solutions for non-convex optimization problems which yield different distributions of class activations in each layer of the model. A layer is watermarked by projecting its class activations into a predefined and randomly generated vector of bits acting as the key.

Wang et al. [20] showed that the watermark of a multitude of algorithms can be

detected in the histogram of their weights. In order to obtain watermarked models whose weight histograms don't differ from non-watermarked ones, they use a GAN-like scheme: One network, similar to a discriminator, tries to detect the presence of the watermark in the weight histogram of the watermarked intermediate layer while another network tries to map its parameters into a randomly generated vector of bits. This algorithm will be abbreviated as "Robust and Undetectable".

2.3 Requirements

In order to evaluate the performance of the watermarking algorithms, a number of requirements [1, 2, 17] are needed, which are:

- *robustness*: Even if the adversary knows the existence of the watermark and the watermarks algorithm, he should not be able to remove the watermark from the model, without reducing its predictive performance in a way that makes it useless.
- *functionality preservability*: Embedding the watermark into the model should not reduce its predictive performance.
- *e ciency*: Watermark embedding and verification should be computationally fast.
- *secrecy*: The adversary has no ability to detect the presence of the watermark in the model.
- *unforgeability*: It's not possible for an adversary to embed his watermark into the already watermarked model without making the model useless by deteriorating its predictive performance.
- *reliability*: The watermark has a high true positive rate on watermark verification, which means that a large proportion of the models which are watermarked are classified as watermarked.
- *fidelity*: The watermark has a high true negative rate on watermark verification, which means that a large proportion of the models which are non-watermarked are classified as non-watermarked.

2.4 Methodology

All experiments are performed on both the MNIST and CIFAR-100 datasets. The MNIST dataset [14] consists of 70.000 28x28 greyscale images of handwritten digits, each labeled by a number from zero to nine. The CIFAR-100 dataset [12] contains 60.000 32x32 three-channel images, each being labeled by one of 100 classes. The classes contain vastly different objects ranging from insects like butterflies to telephones.

Both datasets are widely used and are similar in dimensions and size, which makes it possible to perform the experiments in a reasonable amount of time using architectures that only differ in the width of its last linear layer.

Six folds are used by the owner for embedding the watermark into the model and for training the model, one fold is kept for validation and three folds are used by the adversary for watermark removal attacks. If the adversary had the same amount of data and computational power as the owner, he could train his own model from scratch, eliminating the need to remove the watermark. That's why it's assumed the adversary only has access to a dataset of a smaller size than the owner.

All experiments are performed using a ResNet with 18 layers [9]. Due to the low dimensionality of the samples, the original architecture is slightly modified, using a 3x3 kernel with a stride of one as the first convolution followed by no pooling layer. The architecture is visualized in figure 1.

Since both Namba et al. [16] and Le Merrer et al. [13] expected pre-trained, non-watermarked models as input to their algorithms, two models had to be pre-trained. Furthermore, the two models are used later to benchmark the *functionality preservability* of the watermarking algorithms.

In all further experiments, stochastic gradient descent with momentum [18] was used as an optimizer with the momentum parameter α set to 0.9. One model was trained on MNIST for 10 epochs using a learning rate of 0.005, achieving a validation set accuracy of 99.16%, while the other model was trained for 15 epochs on CIFAR-100, using a learning rate of 0.005, achieving a validation set accuracy of 42.27%.

In order to benchmark the models a general way is needed to calculate the minimum accuracy of the watermark in the model verification step. Here the approach of Rouhani et al. [17] was used. The minimum accuracy t in order to prove ownership of the model is calculated using algorithm 1, which is best described using an example: If $p = 0.05$, then algorithm 1 returns a threshold t for which a model which generates random predictions would have a probability of 5% or less to achieve an accuracy acc, so that $acc > t$. In all further experiments the threshold t is calculated using $p = 10^{-6}$.

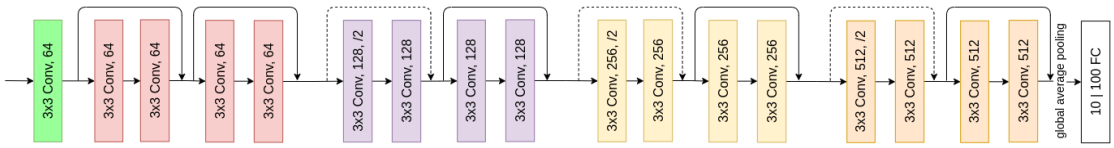


Figure 1: Modified ResNet-18 architecture. The rounded lines are skip connections. Dashed lines are skip connections where a down-sampling operation is performed using a 1x1 convolution with a stride of 2.

Algorithm 1: Watermark Threshold Calculation

Input: Number of classes in the key c, length of the key k, probability that a model with random predictions achieves the same or better accuracy on the key than the returned accuracy p

Output: Minimum watermark accuracy to prove ownership of the model

$s \leftarrow 0$

for $i \leftarrow 0; i < k; i \leftarrow i + 1$ **do**

  $s \leftarrow s + \binom{k}{i}(\frac{1}{c})^{k-i}(1 - \frac{1}{c})^{i}$

  if $s > p$ **then**

    return $1 - \frac{i}{k}$

2.5 Adversarial Frontier Stitching for Remote Neural Network Watermarking

In order to inject the watermark into the model, the model has to be pre-trained. For each correctly classified sample in the training set an adversarial example is created using the "fast gradient sign method" (1) [7] as suggested by Le Merrer et al. [13]:

$$x_{adv} \leftarrow x + \epsilon * sign(\nabla_x J(\theta, x, y)) \tag{1}$$

where:

ϵ = controls the amount of change applied to x
$sign()$ = maps negative values to -1, 0 to itself and positive values to 1

If the adversarial examples are still correctly classified by the pre-trained model, they are called "false adversaries" and if they are now misclassified they are called "true adversaries" by the authors of the algorithm. The concatenation of the sets of the "true adversaries" and "false adversaries" acts as the key K and is used to further fine-tune the pre-trained model m until it achieves a high accuracy on the key set K. This process is detailed in algorithm 2.

The models ownership is verified if $acc(m, K) \geq t$.

Algorithm 2: Adversarial Frontier Stitching Watermarking

```
Input: Training set (X, Y), pretrained model m, key size l = |K|, step size ε for
       adversarial example generation, weights θ of pretrained model m, loss function J
Output: Watermarked model m_wm, key set K
while K_true < l/2 or K_false < l/2 do
    pick the next sample, x ∈ X, y ∈ Y
    if m(x) = y then
        x_adv = x + ε * sign(∇_x J(θ, x, y))
        if m(x_adv) = y and K_false < l/2 then
            /* x_adv is a false adversary */
            K_false ← K_false ∪ {(x_adv, y)}
        else if m(x_adv) ≠ y and K_true < l/2 then
            /* x_adv is a true adversary */
            K_true ← K_true ∪ {(x_adv, y)}
K = K_true ∪ K_false
m_wm ← train(m, K)
return m_wm, K
```

For both datasets, l was set to 100. For CIFAR-100 the adversarial examples were generated using $\epsilon = 0.1$ and the watermark was embedded into the pre-trained model using 15 epochs over the key set. For MNIST the adversarial examples were generated

using $\epsilon = 0.25$ and the watermark was embedded into the pre-trained model using 10 epochs.

The watermarked models achieved a validation set accuracy of 35.25% and an accuracy of 96% on the key set for CIFAR-100. For MNIST the model reached a validation set accuracy of 98.44% and a watermark accuracy of 98%.

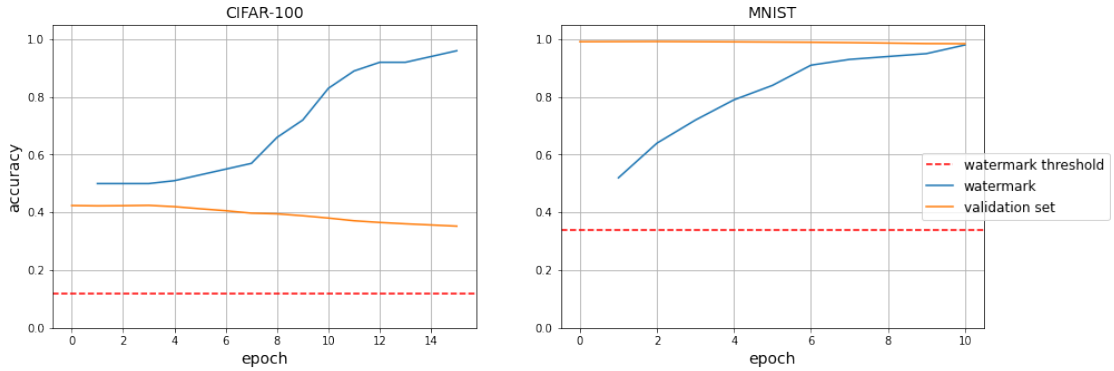


Figure 2: Training progress of embedding the watermark into the pre-trained ResNets using Adversarial Frontier Stitching.

2.6 Robust Watermarking of Neural Network with Exponential Weighting

Exponential weighting can be interpreted as an activation function applied to the weights of a layer before they are used its operation. Exponentially weighting the i-th parameter of the l-th layer of the model is defined as:

$$ew(\theta_i^l, t_e) = \frac{\exp(|\theta_i^l|t_e)}{\max_i \exp(|\theta_i^l|t_e)}\theta_i^l \tag{2}$$

where:

θ = parameters of the model
t_e = temperature, controls the strength of the exponential weighting

Let a^l be the l-th layer activation function, let $op(z^l, \theta^l)$ be the l-th layer operation, which would be matrix multiplication for fully connected layers or a convolutional operation for convolutional layers, on the l-th layer input z^l using the l-th layers parameters θ^l, then the output z^{l+1} of the l-th layer is defined as:

$$z^{l+1} = a^l(op(z^l, \theta^l)) \tag{3}$$

If the l-th layer is exponentially weighted, this equation changes to:

$$z^{l+1} = a^l(op(z^l, ew(\theta^l, t_e))) \tag{4}$$

The process of watermark embedding using exponential weighting is described in algorithm 3.

Algorithm 3: Exponential Weighting Watermarking

Input: Training set (X, Y), pre-trained model m, key size $l = |K|$
Output: Watermarked model m_{wm}, key set K
while $K < l$ **do**
    pick the next sample, (x, y) from the training set (X, Y)
    randomly assign new label y_{new}, so that $y_{new} \neq y$
    $K \leftarrow K \cup (x, y_{new})$
For each layer in m change equation (3) to equation (4).
$m_{wm} \leftarrow train(m, K \cup (X, Y))$
For each layer in m change equation (4) to equation (3).
return m_{wm}, K

The model's ownership is verified if the accuracy of the model on the key set K is above a predefined threshold t, that should not be mistaken with the temperature t_e:

$$verify(m, K) = acc(m, K) \geq t \tag{5}$$

The approach of Namba et al. [16] was followed and t_e was set to 2.0. For both experiments on MNIST and CIFAR-100 $|K| = 20$, epochs $= 5$, and a learning rate of

0.005 was used. Increasing the size of the key further reduced the validation set accuracy of the model. This is due to the label change, the model is trained on training samples whose labels have been randomly changed and therefore is forced to wrongfully predict samples, which deteriorates its predictive performance. The training process can be seen in figure 3, it has to be noted that changing the operation of the layer from equation (4) back to (3) significantly reduced the model's accuracy on the validation set from 41.83% to 32.80% and the watermark accuracy from 100.00% to 60.00% for the model trained on CIFAR-100. However for the model trained on MNIST this change of equations had very little effect, the validation set accuracy dropped from 99.07% to 99.01% and the watermark accuracy stayed at 100.00%.

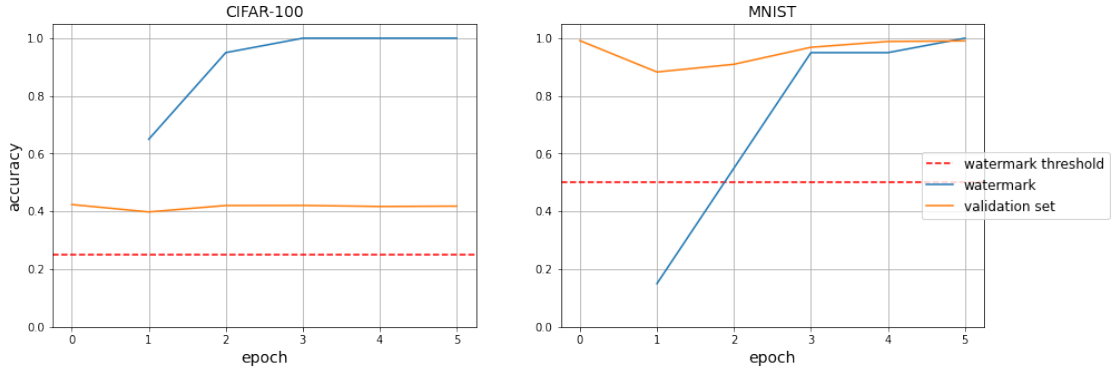


Figure 3: Training progress of embedding the watermark into the pre-trained ResNets using Exponential Weighting.

2.7 Piracy Resistant Watermarks for Deep Neural Networks

In order to generate the bit-pattern used for watermarking the owner first concatenates a unique identifier *ident*, which could be a social security number or an address, with a timestamp $v = ident + timestamp$. A key pair (O_{pub}, O_{priv}) is generated using an algorithm that implements public-key cryptography and this key pair is used to create a signature $sig = sign(v, O_{pub}, O_{priv})$. The owner is now able to prove that he was in possession of the private key corresponding to the public key which was used to create the signature without publishing his private key. The signature sig is mapped into a class index y_w, a bit-pattern *bits* and its position *pos* using algorithm 4:

Algorithm 4: Piracy Resistant Watermark Generation

Input: Signature sig, width and height of the generated bit-pattern n, height of samples h, width of samples w, number of classes c

Output: Class which will be used for watermarking y_w, bit-pattern *bits*, tuple *pos* which corresponds to the upper left position where the bit-pattern is applied to samples

```
/* h1, h2, h3 and h4 are cryptographic hashing functions                    */
```

$y_w \leftarrow \text{h1}(sig) \bmod y$

```
/* binary() transforms an integer into a bitstring                          */
```

$bits \leftarrow \text{binary}(\text{h2}(sig) \bmod 2^{n^2})$

```
/* if len(bits) < n^2 then zeros are padded to the beginning of the bitstring */
```

$bits \leftarrow \text{pad\_zeros}(bits)$

$pos \leftarrow [\text{h3}(sig) \bmod (h - n), \text{h4}(sig) \bmod (w - n)]$

return y_w, *bits*, *pos*

Using the generated bit-pattern *bits* and position of the bit-pattern *pos*, null embedding, and true embedding are applied to subsets of the training dataset. Null embedding (6) treats the bit-pattern as a mask and for each overlapping 1-bit, it sets the value in the sample to λ and for each overlapping 0-bit it sets the value to $-\lambda$, the labels remain unchanged. True embedding (7) inverts the bit-pattern and additionally sets the labels to y_w. The process of training a model on both the null and true embedded datasets is called dual embedding.

Definition Dual Embedding [15]: A watermark pattern (*bits*, *pos*) is successfully dual embedded into a model m, if for all potential inputs (x, y):

$$m(x \oplus [bits, pos, \lambda]) = m(x) \tag{6}$$

$$m(x \oplus [inv(bits), pos, \lambda]) = y_w \tag{7}$$

where:

$inv()$ = inverts each bit in the bit-pattern, maps 0 to 1 and 1 to 0
λ = controls the strength of the embedding, Li et al. used 2000
$\oplus$ = operation which applies the bit-pattern to the sample

The owner starts with an untrained model m and simultaneously trains it on the concatenation of the training set and the generated null and true embedding datasets in order to inject the watermark.

The size of the null and true embedding dataset is a hyperparameter that controls the trade-off between the model's predictive performance and the watermarks *robustness*, *reliability*, and *fidelity* requirements.

The authors explain two protocols for ownership verification. For simplicity reasons only "*private verification via trusted authority*" [15, p. 8] is evaluated:

In order to verify ownership of the model m, the owner submits his signature sig, public key O_{pub}, verifier string v, λ and a subset of his training set, called the extraction set, to a trusted authority. It is assumed that the authority has knowledge of the watermark generation algorithm and the cryptographic hashing functions $h1$, $h2$, $h3$, $h4$ used to generate the watermark. The authority then runs algorithm 5 to verify the model's ownership. The authority has to be trusted in a way that it doesn't leak information about the generated bit-pattern. Otherwise, the adversary could dual embed his own data using the pattern and change the classification outcome, making the watermark impossible to verify [15].

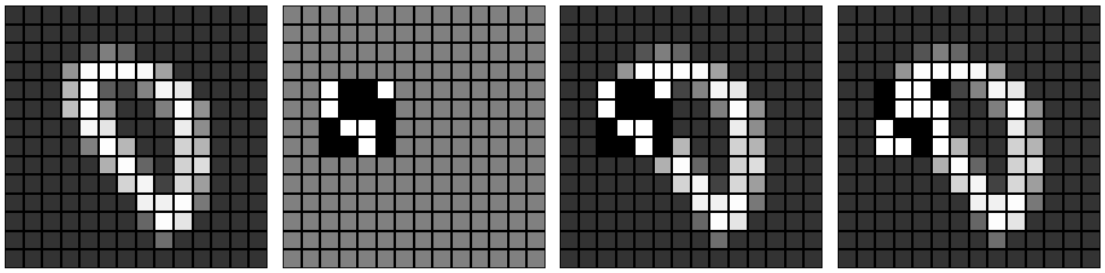
Algorithm 5: Private Verification via Trusted Authority

Input: Extraction set (X, Y), model to verify m, signature sig, public key O_{pub}, verifier string v, signature sig, λ, width and height of the generated bit-pattern n, watermark threshold t
Output: 1 if the ownership verification was successful, 0 otherwise
if *not verify*(O_{pub}, sig, v) **then**
    return 0
$y_w, bits, pos = \text{generate}(sig, n)$
foreach $(x, y) \in (X, Y)$ **do**
    $K_{null} \leftarrow K_{null} \cup (x \oplus [bits, pos, \lambda], y)$
    $K_{true} \leftarrow K_{true} \cup (x \oplus [inv(bits), pos, \lambda], y_w)$
$\phi_{null} \leftarrow acc(m, K_{null})$
$\phi_{true} \leftarrow acc(m, K_{true})$
if *min*(ϕ_{null}, ϕ_{true}) $\geq t$ **then**
    return 1
return 0

In the experiments on both MNIST and CIFAR-100 a learning rate of 0.005 was used. The model was trained for 15 epochs on CIFAR-100 and for 10 epochs on MNIST. The whole training set was null embedded while 1000 randomly selected samples were true embedded. True embedding a larger subset of the training set caused a significant loss in accuracy due to true embeddings label change. The extraction set needed for ownership verification consists of 100 samples drawn from the training set.

The hyperparameter λ which controls the strength of the true and null embeddings was set to 50. Li et al. [15] set λ to 2000 in their experiments, however, this was impossible to replicate since setting λ to values higher than 50 caused all gradients to be zero.

The model trained on CIFAR-100 achieved a validation set accuracy of 36.89%, a true embedding set accuracy of 100%, and a null embedding accuracy of 41.41%. The model trained on MNIST achieved a validation set accuracy of 98.81% and a true and null embedding set accuracy of 100%.



(a) Original (b) Pattern (c) Null embedding (d) True embedding

Figure 4: Null and true embedding an image using a bit-pattern. a) The original image. b) bit-pattern which was used for true and null embedding, with $pos = (4, 2)$. c) Original image after null embedding was applied. d) Original image after true embedding was applied.

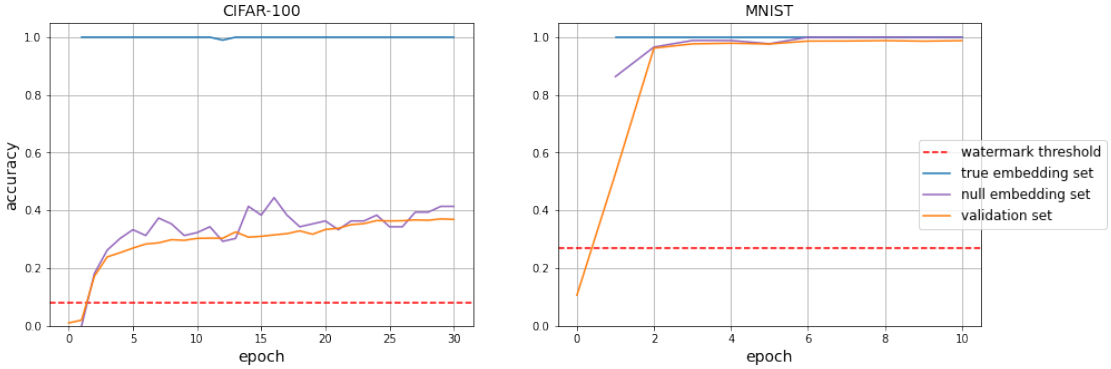


Figure 5: Training progress of embedding the watermark into the ResNets using Piracy Resistant Watermarks.

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

DeepSigns white-box approach watermarks an intermediate layer by projecting the average activations of each class into a randomly generated matrix of bits B.

DeepSigns generates three matrices, $M \in R^{s \times n}$, $A \in R^{n \times h}$ and $B \in \{0,1\}^{s \times h}$, here s denotes the number of classes the model predicts, l is the index of the watermarked layer, n is the number of intermediate features of layer l and h is a hyperparameter that controls the size of the key. Both M and A are initialized to samples drawn from a standard normal distribution $N(0,1)$. Unlike M, A and B are not fine-tuned during the training of the model.

DeepSigns assumes that each classes activations in intermediate layers roughly follow a normal distribution. To strengthen this assumption, DeepSigns adds the following term to the total loss of the network:

$$\frac{\lambda_1}{m} \sum_{i=1}^{m} (\|M^l_{y^*} - m^l(x_i;\theta)\|_2^2 - \sum_{j \neq y_i^*} \|M^l_j - m^l(x_i;\theta)\|_2^2) \tag{8}$$

where:

λ_1	= controls the contribution of the term to the total loss
l	= index of the watermarked layer
m	= number of samples in the batch
$m^l(x;\theta)$	= activations of the l-th layer on sample x
M^l_j	= j-th row vector of the trainable matrix M^l
y^*_i	= ground truth of the i-th sample in the batch

Equation (8) minimizes the "entanglement between data features (activations) belonging to different classes while decreasing the inner-class diversity" [17, p. 4] and pushes the activations of each class to roughly follow the mean vectors defined by M^l. λ_1 is a trade-off between the classes activations of the hidden layer fitting a normal distribution and the accuracy of the model.

The mean values $\mu \in R^{s \times n}$ of activations belonging to each class of the watermarked layer are then projected into B using A.

$$\begin{aligned} G^{s \times h} &= sigmoid(\mu^{s \times n} \times A^{n \times h}) \\ B^{s \times h}_{pred} &= hard\_thresholding(G^{s \times h}, 0.5) \end{aligned} \tag{9}$$

Where $hard\_thresholding(x, thresh)$ maps into 1 if $x >= thresh$ and into 0 if not. In order to reduce the Hamming distance between the predicted matrix of bits B_{pred} and the ground truth matrix B an additional term is added to the loss function during watermark embedding.

$$-\lambda_2 \sum_{i=1}^{s} \sum_{j=1}^{h} (B_{ij} ln(G_{ij}) + (1 - B_{ij}) ln(1 - G_{ij})) \tag{10}$$

where:

λ_2 = controls how much the distance between B and G should contribute to the loss

The model's ownership is verified by evaluating the accuracy between the predicted key B_{pred} and the key B (12).

$$acc(B_{pred}, B) = \frac{1}{hs} \sum_{i=1}^{s} \sum_{j=i}^{h} \mathbb{1}[B_{pred(ij)} = B_{ij}] \tag{11}$$

$$verify(B_{pred}, B) = acc(B_{pred}, B) \geq t \tag{12}$$

For CIFAR-100 the model was trained for 35 epochs using a learning rate of 0.005, $\lambda_1 = 0.001$ and $\lambda_2 = 2.0$. For MNIST the model was trained for 10 epochs using a learning rate of 0.005, $\lambda_1 = 0.0005$ and $\lambda_2 = 1.0$. For both experiments $h = 128$ was used and l was set to the index of the last convolutional layer of the ResNets.

A validation set accuracy of 40.40% was achieved for CIFAR-100 and 99.34% for MNIST, which differs only slightly from that of a non-watermarked model, while the accuracies on the watermark (11) are 98.7% for CIFAR-100 and 98% for MNIST.

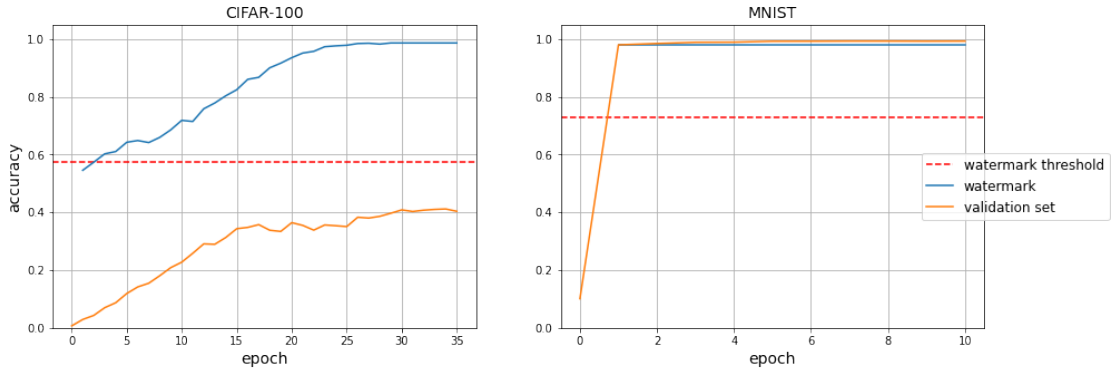


Figure 6: Training progress of embedding the watermark into the ResNets using DeepSigns white-box watermarking approach.

2.9 Robust and Undetectable White-Box Watermarks for Deep Neural Networks

Wang et al. [20] train three models synchronously, m, m_{det} and m_{ext}. m_{det} and m_{ext} are both standard feed-forward neural networks. m_{ext} task is to map the weights w of a predefined layer of m into a randomly generated key $k \in \{0,1\}^h$, where h is a hyperparameter that controls the size of the watermark. However, this approach can yield many false positives during ownership verification since m_{ext} could learn to project a large range of inputs to k. Therefore a number of non-watermarked models are trained before embedding the watermark and their weights w_non are extracted. For each non-watermarked model an additional key $k_{non} \in \{0,1\}^h$ is generated. Now m_{ext} learns to map w into k while also mapping the weights of the non-watermarked models w_{non} into their keys k_{non} using the objective below.

$$d(k, m_{ext}(w;\theta)) + d(k_{non}, m_{ext}(w_{non};\theta)) \tag{13}$$

where:

$d(x,y)$ = measure of distance between x and y, binary cross-entropy can be used

m_{det} objective is similar to that of a discriminator in generative adversarial networks [8], it maps the weights of watermarked and non-watermarked models into a probability $p \in [0,1]$ that the weights belong to a model which was not watermarked. However the weights passed to m_{det} have to be permutation invariant, this problem is solved by sorting. The loss function of m_{det} is defined as:

$$-(log(1 - m_{det}(w;\theta)) + log(m_{det}(w_{non};\theta))) \tag{14}$$

Which pushes m_{det} to map w_{non} into 1 and w into 0. The loss for the model m to watermark is defined as:

$$\mathcal{E}_0 + \lambda_1 d(k, m_{ext}(w;\theta)) - \lambda_2 log(m_{det};\theta) \tag{15}$$

Where $\mathcal{E}_0$ is the original loss term of the model, typically cross-entropy in classification or mean squared error in regression tasks. λ_1 and λ_2 are trade-off hyperparameters between the model's accuracy and the number of false negatives on ownership verification/the weights of the model being indistinguishable from weights of non-watermarked models. Similar to the generator in a generative adversarial network, m is punished when f_{det} successfully predicts that its weights are watermarked. In this way m aims to learn a distribution of weights that matches the distribution of weights of non-watermarked models, making the watermark "undetectable".

Ownership of the model is verified if $acc(m_{ext}(w;\theta), k) \geq t$.

The last convolutional layer of the ResNets were watermarked with $h = 128$. For CIFAR-100 the model was trained for 30 epochs using a learning rate of 0.005, $\lambda_1 = 2.0$ and $\lambda_2 = 1.0$. For the experiment on MNIST, the model was trained for 10 epochs and λ_1 was set to 1.0 instead. The training progress is visualized in figure 7.

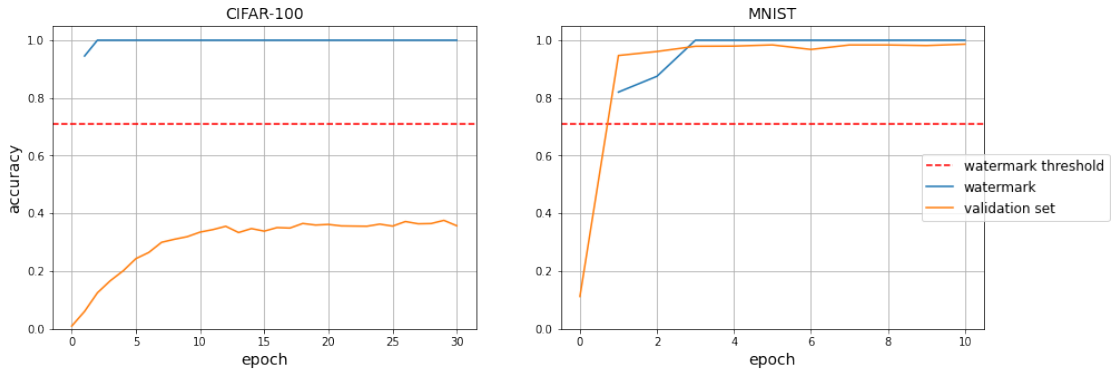


Figure 7: Training progress of embedding the watermark into the ResNets using the algorithm proposed by Wang et al. [20].

3 Experiments

3.1 Robustness

A watermarking scheme should be *robust* against model modification, where an adversary tries to directly alter the watermarked model m_{wm} in a way that makes ownership verification impossible. Trying to remove a watermark using model modification is always a trade-off between the model's predictive performance and the watermarks prediction accuracy since the model could be altered in a way that it makes random predictions, potentially removing any of the presented watermarking schemes, but also making the model unusable. A watermark is *robust* if in order to remove the watermark the model has to be modified in a way that makes it useless.

In the following the *robustness* of the watermarking schemes is evaluated by attacking them with two of the most widely used model modification approaches: Fine-tuning and weight pruning [1, 3, 4, 11, 15, 16, 20].

The datasets which are used to attack the models are subsets of the MNIST/CIFAR-100 dataset, remember that the datasets were split into 10 stratified folds, from which three folds are used in the upcoming fine-tuning and weight pruning attacks.

3.1.1 Fine-Tuning

Fine-tuning is an attack based on *catastrophic forgetting*, which states that when a model m is trained sequentially on datasets A and B, it will partly forget how to predict samples of dataset A. So if the model learns to predict a key K and is then fine-tuned on new data, it should unlearn how to classify the key K, reducing the watermarks ownership verification capabilities.

To evaluate the robustness of the watermarks against fine-tuning, the watermarked models were trained for 50 epochs on the three unseen folds, using a high learning rate of 0.02. This process is showcased in figure 8.

3.1.2 Weight Pruning

Weight pruning is a method that is used to decrease the size of models in production and increase their computational efficiency.

It's based on the fact that a multitude of the parameters in a model has very little influence on its predictions. Also, weights with higher magnitude tend to have a higher influence. So in order to reduce the size of the model, one can sort its weights by their magnitude and then set an arbitrary amount of them to zero. The amount of parameters of a model which are zero is called its sparsity.

A trick to reduce the size of the model after setting a large number of its parameters to zero is by compressing the model using a compression algorithm like Deflate, which is used by gzip, which will automatically reduce the model's size by abusing its amount of same value parameters. A similar reduction in space requirement can be achieved by using sparse tensors which are supported by both TensorFlow and PyTorch.

There are two ways of approaching weight pruning, *structured* and *unstructured*. In structured pruning, whole neurons of the model are set to zero, in unstructured pruning single parameters are set to zero.

Unstructured pruning is used in all experiments. The models are trained for 50 epochs using the three unseen splits. Each training iteration a mask is generated which is applied to the model's parameters. Parameters that are pruned are multiplied by zero before their layers operation, parameters that are not pruned are multiplied by one. Pruning and fine-tuning the models simultaneously has shown to result in a smaller reduction of the model's predictive performance. In order to calculate how many of the parameters should be pruned during each iteration the following function is used:

$$s_i = -s_{final}(1 - \frac{i}{i_{max}})^p + s_{final} \tag{16}$$

where:

s_i = sparsity of the model at iteration i
i_{max} = number of iterations the model will be pruned for
s_{final} = final sparsity, achieved after the model has been pruned for i_{max} iterations
p = controls the curvature of the sparsity graph

The watermarked models were trained using a learning rate of 0.005. During each training iteration, the sparsity of the model was increased using equation (16) with s_{final} set to 0.9999 and $p = 3$. The results can be seen in figure 9.

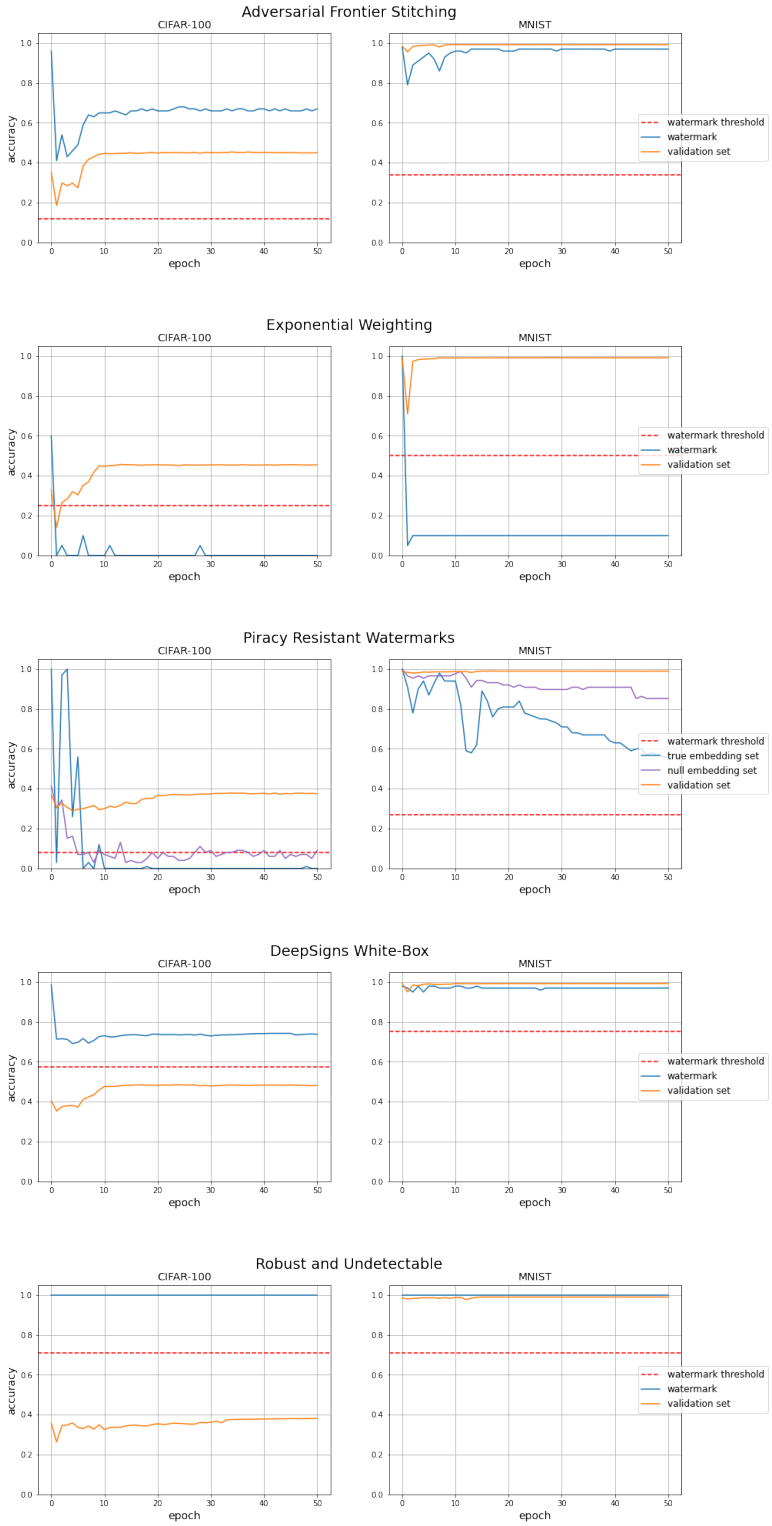


Figure 8: Evaluation of the algorithms *robustness* against fine-tuning attacks.

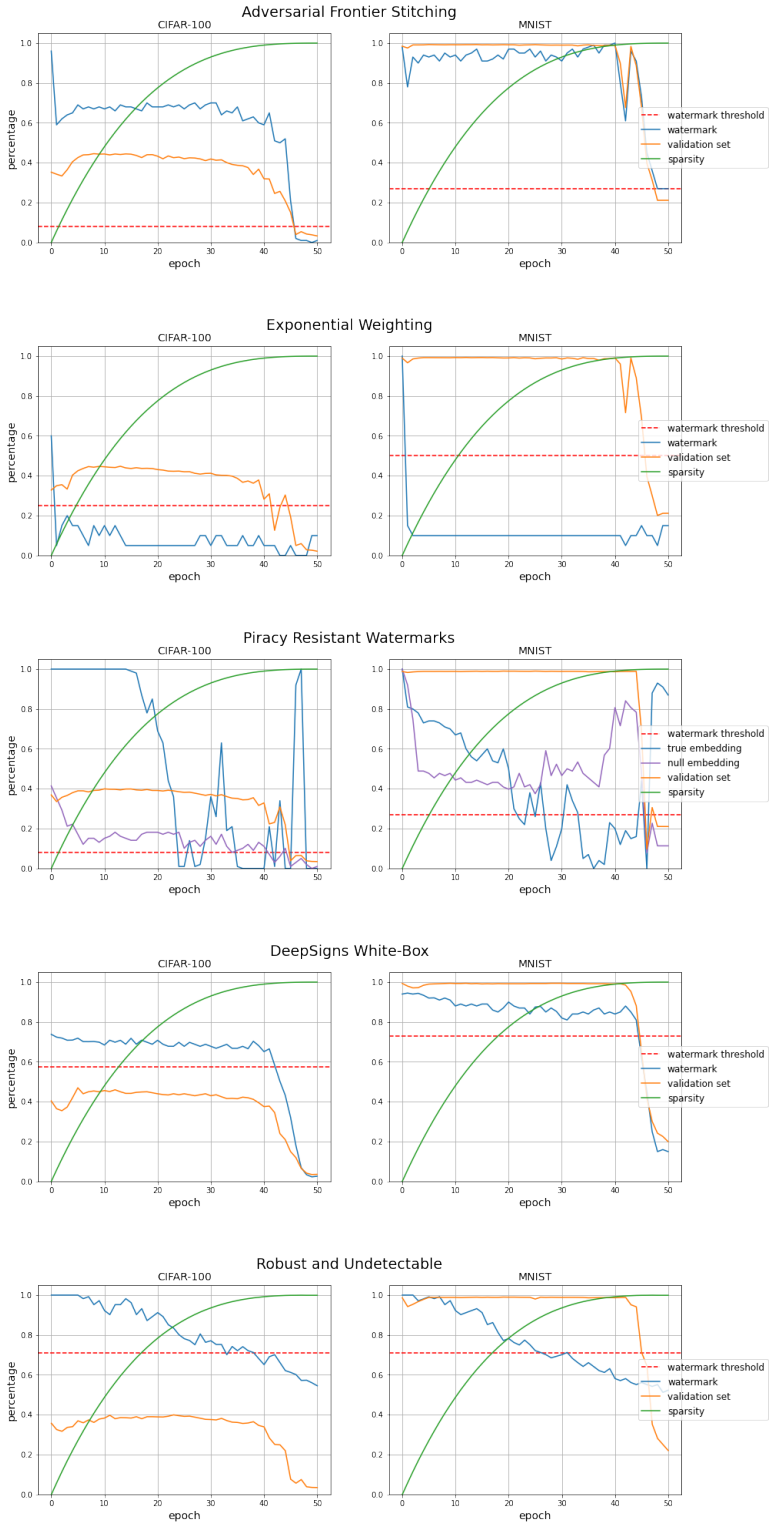


Figure 9: Evaluation of the algorithms *robustness* against weight pruning attacks by plotting the validation set and watermark accuracies against an increasing amount of sparsity.

3.2 Functionality Preservability

A watermarking algorithm is *functionality preserving* if its decreasing effect on the model's predictive performance is minimal. In order to benchmark the algorithm's *functionality preservability*, the validation set accuracies of the watermarked models are compared against trained and non-watermarked models.

	CIFAR-100	MNIST
No watermark	42.27%	99.16%
Adversarial Frontier Stitching [13]	35.25%	98.44%
Exponential Weighting [16]	32.80%	99.01%
Piracy Resistant Watermarks [15]	36.89%	98.81%
DeepSigns white-box [17]	40.40%	99.34%
Robust and Undetectable [20]	35.73%	98.63%

Table 1: Accuracy of the watermarked models on the validation set.

3.3 E□ciency

A watermarking approach is *e cient* if the time spent on watermark embedding and verification is minimal. All experiments have been run in eager execution using TensorFlow 2.4 on an RTX 2060 SUPER, except the function in DeepSigns which transformed activations of a layer into a matrix of average class activations, which was run in TensorFlow's graph mode. Running the function in eager execution increased the training time of CIFAR-100 by a factor of four. For MNIST however, the difference in training time was barely noticeable, this was due to its lower number of classes.

If the algorithms required a pre-trained model its training time was added to the time spent on watermark embedding.

	Training + Embedding		Verification	
	CIFAR-100	MNIST	CIFAR-100	MNIST
No watermark	452.09s	317.04s	-	-
Adversarial Frontier Stitching [13]	540.33s	371.42s	0.04s	0.04s
Exponential Weighting [16]	696.84s	538.93s	0.04s	0.04s
Piracy Resistant Watermarks [15]	1725.53s	605.74s	0.20s	0.18s
DeepSigns white-box [17]	2172.80s	444.77s	0.12s	0.11s
Robust and Undetectable [20]	3055.80s	1792.58s	0.01s	0.01s

Table 2: Time spend on training + embedding and ownership verification.

3.4 Secrecy

A watermarking algorithm fulfills the *secrecy* requirement, if it is not possible to detect the presence of the watermark in the model, furthermore for algorithms that operate in a black-box setting it should not be possible to detect whether an input of the model

belongs to the key set or not. Wang et al. [20] showed that it's possible to train a classifier that detects watermarked models by training on the weight histograms of watermarked and non-watermarked models. However that would be out of scope for this work due to its computational complexity, instead, the evaluation is based on abnormalities in the weight histograms of the watermarked models, which are shown in figure 10, and the resistance of the black-box algorithms to query modification. Query modification is a way to attack the watermark verification step of models which are hidden behind an API. The query from the client is analyzed and if a watermark was detected a random prediction is returned. Namba et al. [16] showed that it's possible to train an autoencoder on the training data and analyze the query in question by its reconstruction loss. If the loss is larger than a predefined threshold it's assumed that the query contains a sample that does not belong to the distribution of the training data and therefore possibly contains a watermark, a random class from the output space is returned. However, it's also possible to detect watermarks simply by looking for out-of-bounds values in the query. Image-based queries should always contain values in the ranges [0, 1] or [0, 255].

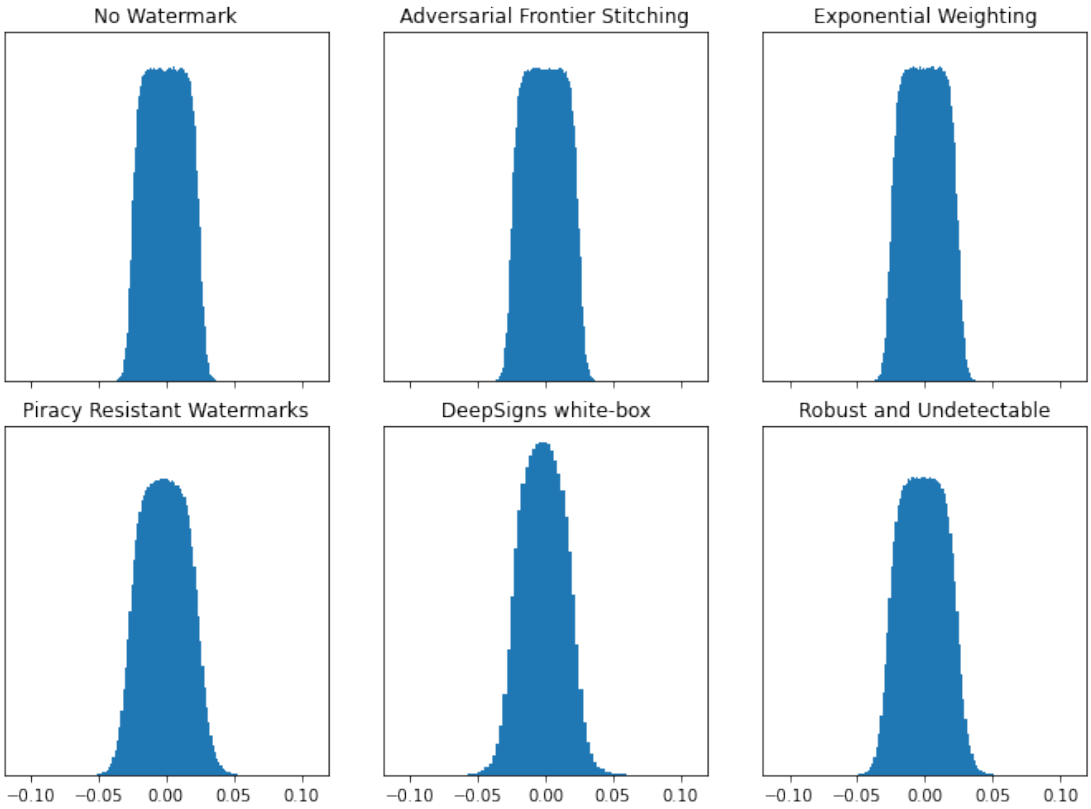


Figure 10: Weight histograms of the concatenation of weights of the last convolutional layers from the ResNets trained on CIFAR-100.

3.5 Unforgeability

A watermarking algorithm is *unforgeable* if it is not possible for an adversary to embed his watermark into the already watermarked model without making the model useless by deteriorating its predictive performance. It has to be noted that none of the algorithms besides Piracy Resistant Watermarks claims to be *unforgeable*. Also, an algorithm that is *unforgeable* can't fulfill the *secrecy* requirement, because an adversary could train a model of the same architecture as the model to investigate. Then he could generate a key and try to watermark both models using the key. If it's only possible to embed the key into the model trained by the adversary or it's only possible to embed the watermark into the model trained by the adversary, then this points to the model being watermarked.

In order to test the *unforgeability* of the watermarking algorithms the black-box watermark proposed by Adi et al. [1] was used. They argue that by choosing the key set to consists of samples that lie out of the data distribution of the training set, the watermarked model learns two non-interfering mappings: One for the key set and one for the training set. This leads to a minimal loss in the model's predictive performance, making the algorithm *functionality preserving*.

Following their approach, a key set of 100 pictures showing abstract art was downloaded from Flickr, labels were assigned at random. Afterwards the watermarked models were trained on the concatenation of the key set and the three splits used for watermark removal. The new watermark was embedded for 30 epochs on CIFAR-100 and for 20 epochs on MNIST and a learning rate of 0.005 was used.

	CIFAR-100		MNIST	
	Watermark	Validation Set	Watermark	Validation Set
Adversarial Frontier Stiching [13]	100%	44.58%	99%	99.17%
Exponential Weighting [16]	100%	44.93%	100%	99.16%
Piracy Resistant Watermarks [15]	100%	39.90%	98%	98.86%
DeepSigns white-box [17]	100%	48.12%	96%	99.33%
Robust and Undetectable [20]	100%	42.92%	100%	98.81%

Table 3: Watermark and validation set accuracy after embedding a second watermark into the models using the method described by Adi et al. [1].

3.6 Reliability

A watermark is *reliable* if the watermark verification step has a high true positive rate. High *reliability* ensures that when a model is watermarked using the algorithm, it will also be classified as watermarked by the verification algorithm.

Table 4 summarises the watermark verification accuracies and their corresponding thresholds.

	CIFAR-100		MNIST	
	Watermark Accuracy	Watermark Threshold	Watermark Accuracy	Watermark Threshold
Adversarial Frontier Stitching [13]	91%	12%	94%	34%
Exponential Weighting [16]	100%	25%	100%	50%
Piracy Resistant Watermarks [15]	true: 100% null: 31.31%	7.99%	true: 100% null: 97.87%	27%
DeepSigns white-box [17]	90.1%	71.09%	97.81%	71.09%
Robust and Undetectable [20]	100%	71.09%	100%	71.09%

Table 4: The watermark accuracies and its thresholds.

3.7 Fidelity

A watermarking algorithm fulfills the *fidelity* requirement if it has a high true negative rate during watermark verification. High *fidelity* is an essential requirement because its a measure of the certainty one has in the verification of the model's ownership. A low *fidelity* means that non-watermarked models could be classified as watermarked, making proof of ownership impossible.

In order to benchmark the *fidelity* of the algorithms 50 unmarked models were trained for both CIFAR-100 and MNIST. To increase the variance of the generated models, hyperparameters were randomly chosen for each model. Each model was trained for a random amount between 5 and 20 epochs on a fraction of 20 - 100% of the training dataset. Momentum was set to a random number between 0.0 and 0.99. For the learning rate, first a number $x \in [-4, -1]$ was sampled, afterwards, the learning rate was set to 10^x.

Ownership verification was evaluated on the generated non-watermarked models. The results are shown in table 4.

Due to the large number of false positives during Adversarial Frontier Stitching's watermark verification phase its watermark threshold t was recalculated using the equation $2^{-|K|} \sum_z^{\theta} \binom{|K|}{z} < 0.05$ from the authors of the algorithm [13]. θ corresponds to the maximum number of misclassifications the model can have on the key set in order to be classified as watermarked. For the used key size $|K| = 100$, the maximum number of misclassifications θ is 42, which equals a watermark threshold t of 58%.

	CIFAR-100		MNIST	
	FP	TN	FP	TN
Adversarial Frontier Stitching [13]	50	0	50	0
Adversarial Frontier Stitching, t = 0.58	34	16	50	0
Exponential Weighting [16]	0	50	0	50
Piracy Resistant Watermarks [15]	0	50	2	48
DeepSigns white-box [17]	0	50	0	50
Robust and Undetectable [20]	11	39	15	35

Table 5: Evaluation of the *fidelity*. An algorithm with a high fidelity has a high true negative rate on watermark verification.

4 Evaluation

4.1 Robustness

Due to the models being trained on unseen data, their accuracy increased in all experiments related to fine-tuning.

Adversarial Frontier Stitching proved to be *robust* against both fine-tuning and weight pruning. While the accuracy of the watermark slightly decreased during fine-tuning, it never went below its threshold. Using weight pruning the watermarks accuracy went below the watermark verification threshold, but at that time the model's validation set accuracy was reduced below 10% for CIFAR-100 and below 30% for MNIST, which made the model useless.

One explanation for Adversarial Frontier Stitching's *robustness* is that half of its key set consists of "false adversarial examples", originally correctly classified samples which are still correctly classified even after being modified by the "fast gradient sign method". Since those key set samples are correctly classified and are based on original training data, fine-tuning the model on new data doesn't have a lot of influence on their predictions. One drawback of this approach is that it could lead to a lot of false positives during ownership verification, this will be discussed in chapter 4.7.

It is shown that Exponential Weighting's approach is not *robust*. One epoch was enough to remove the watermark using fine-tuning. This is due to the fact that its key set consists of original samples whose labels have been changed which forces the model to misclassify samples. When the model was then trained on new samples, their predictions generalize to the prediction of the misclassified samples in the key set, which results in a decrease in accuracy.

Exponential Weighting claims *robustness* against weight pruning by minimizing the number of parameters that have a strong influence on the final prediction. However, the experiments have shown that that's not the case when the model is pruned and trained simultaneously, most probably due to the same reason Exponential Weighting is not *robust* against fine-tuning which was mentioned earlier.

Piracy Resistant Watermarks [15] claims to be *robust* against both model pruning and fine-tuning. While fine-tuning couldn't remove the watermark of the model trained on MNIST, it was possible to remove the watermark of the model trained on CIFAR-100 without deteriorating the model's predictive performance. Weight pruning successfully removed the watermark of the model trained on MNIST, for CIFAR-100 it removed the watermark after 36 epochs by bringing the null embedding set accuracy below the threshold, but also lowered the validation set accuracy to 94.5%.

DeepSign's white-box approach proved to be *robust* against both fine-tuning and weight pruning attacks.

Robust and Undetectable showed strong *robustness* against fine-tuning. This is due to two reasons:

1. It uses a network to directly map the weights of an intermediate layer to the key. If the weights in the intermediate layer barely change due to the training on unseen but similar data, the predictions of the key extraction network barely change too.

2. As shown later in chapter 3.7, Robust and Undetectable overfitted on its watermark predictions - while it classified the watermarked model which was seen during training correctly, it also classified non-watermarked models as watermarked.

However, it isn't *robust* against weight pruning attacks. This is due to the direct reliance on the parameter values in the watermarked intermediate layer, which are projected into the key by the extraction network. Directly modifying those parameter values by setting a proportion of them to zero has a large influence on the watermark verification.

4.2 Functionality Preservability

All algorithms deteriorated the validation set accuracy on CIFAR-100. On MNIST however, the accuracies differ less because the classification of MNIST is an easier task than that of CIFAR-100.

Exponential Weighting significantly reduced the model's predictive performance. After embedding the watermark into the model, the layers equations are changed from (4) to (3), which reduced the accuracy of the model on the validation set from 41.83% to 32.80% for CIFAR-100.

DeepSigns white-box approach achieved the lowest reduction in predictive performance for CIFAR-100. For MNIST it even slightly outperformed the non-watermarked model in its validation set accuracy.

Adversarial Frontier Stitching, Piracy Resistant Watermarks, and Robust and Undetectable achieved an accuracy on CIFAR-100 which is more than five percent below the non-watermarked model.

A reason why Piracy Resistant Watermarks had such an impact on the model's predictive performance is that it changed the label of the samples in its true embedding set. Adversarial Frontier Stitching embedded the watermark by fine-tuning the pre-trained model using only the key set. This has shown to reduce the model's predictive performance. One way to potentially improve the algorithm's *functionality preservability* would be to train the algorithm on the concatenation of a subset of the training set and the key set instead of on the key set only.

4.3 E□ciency

The time spent on ownership verification is very small and therefore negligible for all five evaluated approaches, however, the time spent on watermark embedding vastly differs.

Adversarial frontier stitching [13] took the least amount of time for embedding a watermark and took only slightly more time than training a model without a watermark.

Exponential weighting[16] performed a little worse, yet it still outperformed the other three algorithms on the CIFAR-100 dataset.

Piracy Resistant Watermarks [15] and DeepSigns white-box approach [17] took much longer on the CIFAR-100 dataset than on MNIST because more training iterations were needed in order to achieve high accuracy.

Robust and Undetectable spent by far the most amount of time on watermark embedding. This is because in order to embed the watermark a set of pre-trained and

non-watermarked models is needed whose weights will be extracted. Otherwise, the extractor and detector networks would overfit and the extractor network would learn a mapping from any weights to the watermark and not just the weights of watermarked models, which would fail the *fidelity* requirement. While it is possible to take weights from pre-trained models found on the internet, this is restricted only to models with similar architecture, since the weights of the model in the watermarked layers have to be of the same shape as the weights of the watermarked model.

If one can get access to pre-trained, non-watermarked models with similar architecture, Robust and Undetectable is relatively *e cient*, taking 844.33 seconds of training and watermark embedding on the CIFAR-100 dataset and 290.91 seconds on MNIST.

4.4 Secrecy

Piracy Resistant Watermarks created its key set by setting certain positions of pixels to very large values, the authors used $\lambda = 2000$ in their experiments. Therefore a watermark in the query can easily be detected by looking for out-of-bound pixel values and that's why its watermark is not *secret*.

Adversarial Frontier Stitching created its key set using adversarial examples, which contain out-of-bound values, however, it would be possible to clip their pixel values and check if their predicted class differs afterwards.

Exponential Weighting created its key set by only altering the label of samples from the training set. Therefore its key set distribution doesn't differ from the training set distribution and its watermark is undetectable by query modification.

Figure 10 shows that all evaluated black-box algorithms' weight histograms don't differ from the histogram of weights of a trained and non-watermarked model.

DeepSign's weight histogram differs from that of the non-watermarked model, there are two possible explanations for this:

1. The weight histogram differs due to the watermarking of the model.

2. The weight histogram differs due to the different numbers of epochs used in training the two models. The non-watermarked model was trained for 15 epochs, while the watermarked model was training for 35 epochs. This could lead to overfitting and therefore a change in the weight histogram due to weights with higher magnitude [5, p. 107].

To investigate the reason for the difference in the weight histograms, a non-watermarked model was trained for 35 epochs using the same hyperparameters the model watermarked by DeepSigns used. Afterwards the two weight histograms are compared again. Figure 11 shows that the histograms differ not because of overfitting but due to the watermark. Therefore DeepSign's white-box approach is at risk of watermark detection using the method described by Wang et al. [20].

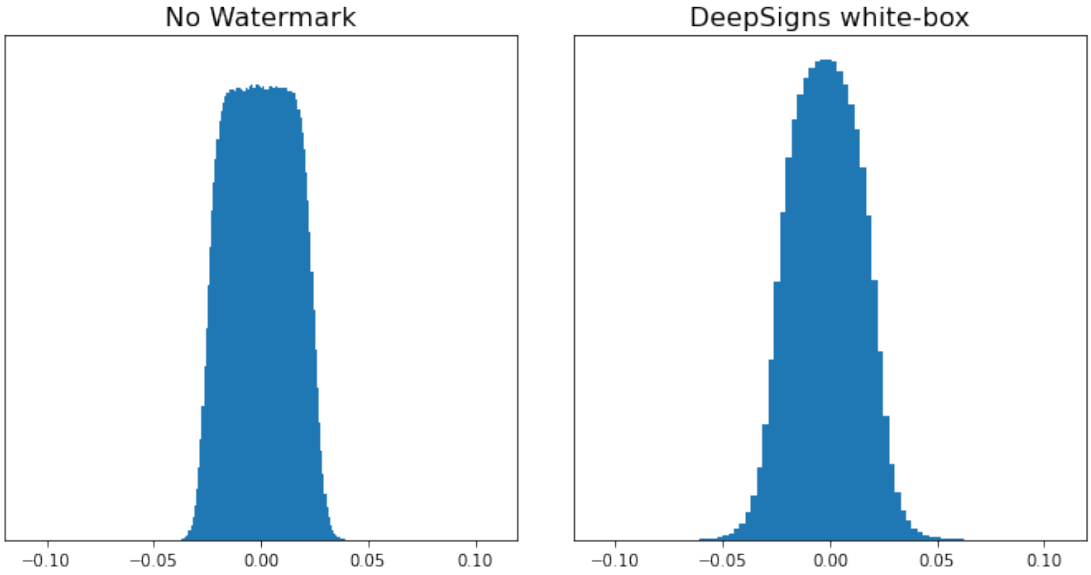


Figure 11: Weight histograms of the concatenation of weights of the last two convolutional layers trained on CIFAR-100.

4.5 Unforgeability

Table 3 shows that all evaluated algorithms failed the *unforgeability* requirement, it was possible to embed a new watermark into the watermarked models which achieved a high accuracy without deteriorating the model's predictive performance.

Piracy Resistant Watermarks [15] claims to be *unforgeable* due to a loss in the model's predictive performance when one tries to embed a new watermark into the already watermarked model. A small loss of predictive performance was noticed during the first few epochs of training, also their algorithm showed by far the smallest increase in validation set accuracy after training the model on new data and embedding the new watermark. However, this initial loss in accuracy was much smaller than the decrease visualized in their work [15, p. 6]. In their experiments, they only tried to embed a new watermark into the model using their own algorithm. While it's true that embedding a pixel-pattern-based watermark deteriorated the model's predictive performance, this phenomenon had little influence when using an approach that doesn't use pixel-patterns.

4.6 Reliability

All five algorithms have passed the *reliability* requirement, since they are all deterministic and achieve higher accuracy on their watermark than their watermarks verification threshold. Therefore if the watermarked model remains unchanged they all achieve a false negative rate of zero.

It has to be noted though that all three evaluated black-box algorithms are vulnerable to evasion attacks as shown by Hitaj et al. [10]. In an evasion attack, the adversary

obtains a multitude of stolen and potentially watermarked models. The predictions are then averaged out over the ensemble of models, which tampers with the predictions on the watermark. Dietterich [6] has shown that ensembles of predictors generally achieve an accuracy superior to that of the individual models, however, this would make predicting less computationally efficient for the adversary.

4.7 Fidelity

Both exponential weighting [16] and DeepSigns white-box approach [17] fulfilled the *fidelity* requirement by correctly classifying all the generated models as not watermarked.

Adversarial Frontier Stitching [13] misclassified all models trained on CIFAR-100 and on MNIST as watermarked for two reasons:

1. Algorithm 1 assumes that the labels of the watermark are generated at random. However, this is not the case for Adversarial Frontier Stitching. The key set consists of the true and false adversarial examples sets, which are all labeled by their samples original label. The false adversarial examples set actually consists of adversarial examples which are still properly classified, therefore giving it a high accuracy on the key set, even for non-watermarked models.

2. It's possible that adversarial examples generated for one model are properly classified by a different model, especially for small ϵ used in the "fast gradient signed method" [7]. In order to improve the algorithms *fidelity* one could increase the hyperparameter ϵ. However, this was not possible, if $|K| = 100$, then $|K|/2 = 50$ false adversarial examples have to be generated from the training set. Setting $\epsilon = 0.1$ generated more than 50 false adversarial examples, however, setting $\epsilon = 0.2$ generated only 15 false adversarial examples from CIFAR-100's training set, which was not enough to construct the key.

The threshold of $t = 0.58$, which was calculated using the equation from Le Merrer et al. [13] was higher than the threshold calculated using algorithm 1, however, it still resulted in 34 false positives during watermark verification for CIFAR-100 and 50 false positives for MNIST.

Piracy Resistant Watermarks [15] correctly classified the models pre-trained on the CIFAR-100 dataset, however, it misclassified two models trained on MNIST as watermarked.

Robust and undetectable white-box watermarks [20] misclassified 11 models trained on CIFAR-100 and 15 models trained on MNIST as watermarked. This was most certainly because of the relatively low number of non-watermarked models used during embedding watermark embedding. Equation (13) needs a number of weights of untrained models w_{non}, otherwise, the network m_{ext} learns a function which maps a large range of inputs to the key of the watermarked model k, resulting in low *fidelity*.

Training five non-watermarked models whose weights will be extracted was not enough in order to fulfill the *fidelity* requirement. It has to be noted though that Robust and Undetectable already spend the most time on watermark embedding due to the model

generation and generating more models linearly increases the time spend on watermark embedding. Also embedding the watermark using a large amount of generated weights can lead to the hardware hitting memory constraints.

5 Conclusion

The results from the experiments differed from the results of the algorithms authors and only Rouhani et al. [17] published their source code. It is certainly possible to improve the performance of the watermarking algorithms by spending more time on hyperparameter optimization.

Each watermarking algorithm should fulfill at least the *reliability* and *fidelity* requirements, otherwise proof of ownership is impossible. Also one could argue that it is more important to have an algorithm that is *functionality preserving* or *robust* than *secret* or *unforgeable*.

Out of all evaluated algorithms, DeepSigns white-box approach performed best. It was, *reliable*, *functionality preserving*, *robust*, and fulfilled the *fidelity* requirement. Its only downsides are that the time spent on watermark embedding correlates with the number of classes in the dataset and that its watermark alters its weight histogram.

Robust and Undetectable White-Box Watermarks [20] relies on the training of multiple models for watermark embedding and is therefore not applicable for the watermarking of models with a high number of parameters, where time spent on training is an issue. It also had a low true negative rate on watermark verification, which makes proof of ownership impossible.

Adversarial Frontier Stitching is *e cient* and *robust*, yet it had a very small true negative rate on watermark verification and diminished the model's validation set accuracy, which makes it unusable in practice.

Exponential Weighting deteriorated the accuracy of the pre-trained model on the CIFAR-100 dataset by 10%, reducing the usefulness of the model. It was also the least *robust* against both fine-tuning and weight pruning attacks.

Piracy Resistant Watermarks was the black-box algorithm with the least influence on the model's accuracy, however, its watermark was easy to remove and it isn't secret due to its obvious, out-of-bounds key samples.

While all evaluated black-box algorithms were flawed, DeepSigns white-box approach comes close to a solution that is usable in production.

The performance of the evaluated algorithms is summarised in table 6.

	Adversarial Frontier Stitching [13]	Exponential Weighting [16]	Piracy Resistant Watermarks [15]	DeepSigns White-Box [17]	Robust and Undetectable [20]
Robustness	✓	×	×	✓	×
Functionality Preservability	×	×	×	✓	×
Efficiency	✓	✓	×	×	×
Secrecy	×	×	✓	×	✓
Unforgeability	×	×	×	×	×
Reliability	✓	✓	✓	✓	✓
Fidelity	×	✓	✓	✓	×

Table 6: Overview of the evaluated algorithms and their performance on the requirements.

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. *Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring*. 2018. arXiv: 1802.04633 [cs.LG].

[2] Franziska Boenisch. *A Survey on Model Watermarking Neural Networks*. 2020. arXiv: 2009.12153 [cs.CR].

[3] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. "DeepMarks: A Digital Fingerprinting Framework for Deep Neural Networks". In: *CoRR* abs/1804.03648 (2018). arXiv: 1804.03648. url: http://arxiv.org/abs/1804.03648.

[4] Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Ruoxi Jia, Bo Li, and Dawn Song. "REFIT: a Unified Watermark Removal Framework for Deep Learning Systems with Limited Data". In: *CoRR* abs/1911.07205 (2019). arXiv: 1911.07205. url: http://arxiv.org/abs/1911.07205.

[5] François Chollet. *Deep Learning with Python*. Manning, Nov. 2017.

[6] Thomas G. Dietterich. "Ensemble Methods in Machine Learning". In: *Lecture Notes in Computer Science* 1857 (2000). It is a good classic article reviewing ensemble methods. He shows intutively why ensembles is a good idea: Statistical, computational, representational., 1–?? url: citeseer.nj.nec.com/dietterich00ensemble.html.

[7] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations*. 2015. url: http://arxiv.org/abs/1412.6572.

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. url: http://arxiv.org/abs/1512.03385.

[10] Dorjan Hitaj and Luigi V. Mancini. "Have You Stolen My Model? Evasion Attacks Against Deep Neural Network Watermarking Techniques". In: *CoRR* abs/1809.00615 (2018). arXiv: 1809.00615. url: http://arxiv.org/abs/1809.00615.

[11] Hengrui Jia, Christopher A. Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. "Entangled Watermarks as a Defense against Model Extraction". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021. url: https://www.usenix.org/conference/usenixsecurity21/presentation/jia.

[12] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR-100 (Canadian Institute for Advanced Research)*. url: http://www.cs.toronto.edu/~kriz/cifar.html.

[13] Erwan Le Merrer, Patrick Pérez, and Gilles Trédan. "Adversarial frontier stitching for remote neural network watermarking". En;en. In: *Neural Computing and Applications* (2019). PII: 4434, pp. 1–12. issn: 1433-3058. doi: 10.1007/s00521-019-04434-z. url: https://link.springer.com/article/10.1007/s00521-019-04434-z.

[14] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). url: http://yann.lecun.com/exdb/mnist/.

[15] Huiying Li, Emily Wenger, Shawn Shan, Ben Y. Zhao, and Haitao Zheng. *Piracy Resistant Watermarks for Deep Neural Networks*. 2020. arXiv: 1910.01226 [cs.CR].

[16] Ryota Namba and Jun Sakuma. *Robust Watermarking of Neural Network with Exponential Weighting*. 2019. arXiv: 1901.06151 [cs.CR].

[17] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. "DeepSigns: A Generic Watermarking Framework for IP Protection of Deep Learning Models". In: (3/04/2018). http://arxiv.org/pdf/1804.00750v2 and https://github.com/Bitadr/DeepSigns.

[18] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536. doi: 10.1038/323533a0. url: http://www.nature.com/articles/323533a0.

[19] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV].

[20] Tianhao Wang and Florian Kerschbaum. *Robust and Undetectable White-Box Watermarks for Deep Neural Networks*. Oct. 2019. url: https://www.researchgate.net/publication/336935809_Robust_and_Undetectable_White-Box_Watermarks_for_Deep_Neural_Networks.