Skoltech

Skolkovo Institute of Science and Technology

MASTER'S THESIS

Developing a watermarking method for deep neural networks with no extra training

Master's Educational Program: Data Science

Student:
Edgar Kaziakhmedov
Data Science
May 29, 2020

Research Advisor:
Grigory Kabatyansky
Principal Research Scientist

Moscow 2020

Developing a watermarking method for deep neural networks with no extra training

Edgar Kaziakhmedov

Abstract

Recent progress in the most challenging problems, such as facial recognition, image, and text classification is attributed to advances in deep learning. Training a state-of-the-art deep neural network is a resource-intensive task as it requires high-performance hardware, diverse, high-quality datasets, and long training time. Independent researchers or companies achieving better scores share their solutions to accelerate progress. Still, the models could be illegally distributed or re-used without proper copyright notice or reference. For these reasons, the DNN models need to be protected.

Digital watermarking is one of the most widely-used techniques to identify ownership of the model. Most of the watermarking algorithms presented so far require the model's re-training or fine-tuning, as well as access to the training set. Since deep model training is days- or week-long procedure, embedding a watermark to an existing pre-trained model might be very time-consuming. Apart from that, real-case scenarios with large datasets and state-of-the-art models remain uncovered in the studies. Given this, the work's contribution is two-fold: first, we present a novel algorithm capable of embedding watermark with no extra training and show that the algorithm meets the main watermarking criteria. Second, we embed watermark to state-of-the-art deep convolutional neural networks such as ResNet, DenseNet, and EfficientNet trained on MNIST, CIFAR-10, CIFAR-100, and ImageNet, and test its resistance to fine-tuning and pruning attacks. We show that the proposed algorithm becomes more feasible for more complex models by eliminating the need for costly re-training.

Research Advisor:
Name: Grigory Kabatyansky
Degree: Doctor of Physical and Mathematical Sciences
Title: Principal Research Scientist

Acknowledgments

To begin, I would like to express my sincere appreciation to my advisor, Grigory Kabatiansky, for his guidance and mentorship throughout this project. Working with him was a true pleasure, as he always approached our discussions with a calm and friendly demeanor.

I am also grateful to my colleagues at Skoltech University, specifically Klim Kireev, Grigory Melnikov, and Mikhail Pautov, for their invaluable willingness to engage in advanced discussions on topics related to mathematics and deep learning.

Furthermore, I would like to extend my thanks to Aleksandr Petiushko, whom I had the pleasure of working with during the Skoltech Industrial Immersion program. His pragmatic approach to problem-solving was immensely helpful in analyzing complex issues.

Lastly, I would like to acknowledge the indispensable support of the Skoltech Zhores supercomputer, which provided me with the necessary hardware, including GPU machines, for conducting my experiments. Without it, this work would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep neural networks have experienced dramatic growth over the last decade. In 1998, Yann LeCun introduced convolutional neural networks that showed superior performance on the recognition of handwritten characters [1]. Since that, deep convolutional networks such as ResNet [2], DenseNet [3] and recently published EfficientNet [4] became gold standards for various tasks: object recognition, face recognition, image classification etc. The high availability of deep learning frameworks such as Keras, Tensorflow, Caffe, Apache MXNet made the development process more accessible. Although there are various architectures and neural network frameworks freely available on the Internet, the training process remains quite a difficult task due to the necessity for high-performance hardware, long training time, and diverse datasets.

Commercial companies are getting involved heavily in deep neural network (DNN) development. They own large private datasets and powerful hardware, which enables more efficient training and architecture search. Thus, the developed model is considered to be the intellectual property of a company. As models can be stolen, illegally re-used or distributed, the rights to shared trained models should be protected.

Deep learning enabled a new market called Machine Learning as a Service (MLaaS). The main advantage of MLaaS that it simplifies the training of neural networks. The largest providers are AWS Machine Learning, Microsoft Azure, IBM Watson, and Google Cloud Engine. The service offers training at a cost substantially lower compared to the price of dedicated hardware. Despite the apparent simplicity, the process possesses essential security and legal questions as a customer might use, distribute, and sell the trained model. These actions might be beyond the license agreement's terms, so the models need to be protected.

One of the ways to protect the models is to utilize a watermarking. Watermarking is used to identify the ownership of digital content such as images, video, and audio. It should be mentioned that there is quite a similar technique called steganography, which aims to conceal the information without being revealed. The differences will be discussed more in detail in Chapter 2. However, widely used watermarking approaches in media are not directly applicable to deep neural networks. Networks are subjected to various modifications such as pruning and fine-tuning, which mostly remain the original performance of a model or even improve it while introducing small changes in the model weights. Slight modifications could be crucial for the extraction of embedded information. Another neighboring task to watermarking is digital fingerprinting. Fingerprinting aims to track each user independently the model was distributed to. In fingerprinting, the owner is able not only to identify copyrights but also to reveal a particular user the model was provided to. Both fingerprinting and watermarking problems are addressed in two scenarios: white-box and black-box. White-box implies having access to the model weights, while the black-box limits access only to the model output (output of softmax function, top-1, or top-K predictions). The complete taxonomy of watermarking in the context of deep neural networks is given in Table 1.1. It is worth mentioning that multi-bit watermarking and fingerprinting are interchangeable terms; the only difference is what the embedded information is used for. Multi-bit watermarking is used to store unique information about each model, while fingerprinting assigns a unique information vector to be embedded to each user.

	Zero-bit watermarking	Multi-bit watermarking	Fingerprinting
White-box	Only **existence** of watermark could be determined from **model's weights**	A **multi-bit binary string** could be extracted from the **model's weights**	A **multi-bit binary string attributed to a specific user** could be extracted from the **model's weights**
Black-box	Only **existence** of watermark could be determined from **model's output**	A **multi-bit binary string** could be extracted from the **model's output**	A **multi-bit binary string attributed to a specific user** could be extracted from the **model's output**

Table 1.1: Taxonomy of watermarking

The main limitation of current DNN watermarking algorithms is re-training: the models should be trained from scratch to embed the watermark. In this work, we first consider a white-box watermark algorithm that embeds the watermark with no extra training. Moreover, we test the algorithm on state-of-the-art convolutional neural networks trained for classification tasks. We show that embedded watermark resists model fine-tuning and low magnitude-based pruning. All experiments are done on different datasets ranging from MNIST to ImageNet. The proposed algorithm would be beneficial for companies certifying models and who need to ensure that an authorized entity can track improper usage.

In Chapter 2, we formulate the problem of watermarking and discuss existing watermarking algorithms highlighting the main concepts: since the first mention of DNN watermarks, plenty of new approaches have been published. After review, we define our goals in this work and highlight the main disadvantage of current approaches. The methodology consists of three chapters. This way of structuring is done to introduce the proposed algorithm gradually and justify feasibility. We perform preliminary experiments in Chapter 3, which gave rise to the final algorithm. Then, in Chapter 4, the main algorithm is explained as well as experimental results. We also consider the algorithm within the main watermarking criteria, proving its effectiveness and comparing it with predecessors signifying innovative parts. In Chapter 5, possible attacks erasing the watermark are performed. Finally, concluding remarks on results and the possible future work are discussed in Chapter 6.

Chapter 2

Related Works

In this chapter, we first consider the differences between steganography and watermarking as reader might find it confusing. The conventional watermarking for digital media is also in scope of the work to get a full picture. Then, main approaches in DNN watermarking is discussed in details.

2.1 Problem formulation

The problem of multi-bit watermarking could be formulated in general way with key fixed length assumptions. Suppose, $m \in \{0,1\}^T$ is a message size of T associated with a given digital container. As watermarking algorithms are assumed to be publicly known, the private key k is used to ensure cryptographic security.

The embedding procedure f could be described in the following way:

$$f(C, m, k) \to C' \text{, s.t. } d(C, C') < \epsilon \tag{2.1}$$

where d is a domain-specific function displaying the similarity between two inputs. For example, the difference between images could be calculated with L_2 distance. In general, it implies not necessarily invisible watermark, but watermark preserving most of the original information. The watermark verification procedure e could be described as follows:

$$e(C', k) \to m', \text{, s.t. } \frac{d_H(m', m)}{T} > \gamma \tag{2.2}$$

where d_H stands for Hamming distance. Additionally, there could be an adversary involved in the process. such that it tries to remove the watermark by introducing transformations g:

$$g(C') \to C'',\ e(C'') \to m'' \text{, s.t. } d(C'', C') < \delta \wedge \frac{d_H(m'', m)}{T} > \gamma \tag{2.3}$$

For instance, for images there are many known possible attacks such resize, rotation, JPEG re-compression. All attacks retain the initial image structure, but aims at corrupting the watermark.

Watermarks that resist a designated class of transforms are called robust. Less resistant watermarks are called fragile or semi-fragile.

2.2 Watermarking vs steganography

The steganography problem is quite close to watermarking. A few details were already mentioned in previous chapter, here we briefly describe the main approach in steganography. The main idea was first formulated in [5], so notations are adhered for integrity.

Given the carrier size is n and message size is m, let's denote embedding and extraction procedure:

$$\text{Emb:}\{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n \tag{2.4}$$

$$\text{Ext:}\{0,1\}^n \to \{0,1\}^m \tag{2.5}$$

Assume, the cover and stego container is denoted as:

$$\begin{gathered} x = (x_1, ..., x_n) \in \{0,1\}^n \\ y = (y_1, ..., y_n) \in \{0,1\}^n \\ M = (M_1, ..., M_m) \in \{0,1\}^m \end{gathered} \tag{2.6}$$

In fact, x, y could be least significant bits of pixels or DCT coefficients. The embedding and extraction procedures should satisfy the following rule:

$$\text{Ext(Emb}(x, M)) = M,\ \forall x \in \{0,1\}^n,\ \forall M \in \{0,1\}^m \tag{2.7}$$

The cost of changing each component of x is denoted as ρ_i and the total cost (distortion) function is D:

$$D(x, y) = \sum_{i=1}^{n} \rho_i |x_i - y_i| \tag{2.8}$$

So, Emb function embeds the message M minimizing the expected distortion E$[D(x, \text{Emb}(x, M)]$. The problem found connection to coding theory such that the embedding and extraction are performed with linear binary code $\mathcal{C}$:

$$\begin{gathered} \text{Emb}(x, M) = \arg\min_{y \in \mathcal{C}(M)} D(x, y) \\ \text{Ext}(y) = Hy \end{gathered} \tag{2.9}$$

Where H is a parity-check matrix of code $\mathcal{C}$. Syndrome-trellis codes was the first class of codes for which the problem could be solved optimally [6].

Later works mostly focused on improving the distortion function D thus making the steganography less detectable for analyzers. It should be noted, that prior steganographic methods mostly focused on preserving the cover distribution. This approach did not yield eventually any practical benefits as modelling the covers remains an unfeasible problem due to inherent diversity.

From now it becomes clear that steganography mostly aims at hiding information in invisible manner. Steganography main problem is to make the embedded message as least detectable at it is possible, while the watermark could be both visible and invisible. Detection of watermarks is not considered as a problem. But watermarks have to be resistant to various carrier modifications, while steganography is very sensitive to slight modification such as JPEG re-compression.

2.3 Watermarking for conventional media

A digital watermark for conventional media is an information vector storing information about the copyright owner. The watermarking algorithms should exhibit the following properties [7]:

1. *Unobtrusiveness* The watermark must remain imperceptible (this is often interchanged with *fidelity*). For images, imperceptibility may imply invisibility to human vision;

2. *Robustness* The watermark should resist various attacks (e.g. geometrical transforms for images) and forgery attacks (re-writing the watermark). if however watermark is erased or forged it should lead to deterioration in fidelity;

3. *Unambiguousness* The retrieved watermark should uniquely match the data owner.

Depending on the media type or algorithms, the properties may vary. Now, let's consider a general framework for image watermarking. Most of the presented approaches published so far utilize spectral embedding first proposed in [7]. The idea was further developed in [8].

Let's consider an input grayscale image $I \in \{0, .., 255\}^{N \times N}$ and compute its DCT transform $D \in \{-2^d, ..., +2^d - 1\}^{N \times N}$, where d determines the possible range. The DCT coefficients are then flattened in zig-zag order and re-numerated: $D' = \{t_1, ..., t_{N^2}\}$. Then, a fixed range of coefficients size of M is considered for embedding:

$$T = \{t_{L+1}, ..., t_{L+M}\} \tag{2.10}$$

Given a normally distributed message X size of M, the embedding is pretty straightforward:

$$t'_{L+i} = t_{L+i} + \alpha |t_{L+i}| x_i \tag{2.11}$$

The extraction process starts from iterating over all possible watermarks that could have been embedded. For a given watermark Y, the correlation between selected DCT coefficients T' and Y is calculated:

$$z = \frac{Y \cdot T'}{M} = \frac{1}{M} \sum_{i=1}^{M} y_i t'_{L+i} \tag{2.12}$$

If image is not subjected to transforms it reduces to:

$$z = \frac{1}{M} \sum_{i=1}^{M} (t_{L+i} y_i + \alpha |t_{L+i}| x_i y_i) \tag{2.13}$$

The detection now gets straightforward as well. By analysing mean and std of z for a fixed Y, the owner determines if Y is an embedded watermark:

$$\mu_z = \begin{cases} \alpha \mu_{|t|} & \text{if } X = Y, \\ 0 & \text{if } X \neq Y, \\ 0 & \text{if no mark is embedded,} \end{cases} \tag{2.14}$$

$$\sigma_z = \begin{cases} \frac{1+2\alpha^2}{M} \sigma_t^2 + \frac{\alpha^2}{M} \sigma_{|t|}^2 & \text{if } X = Y, \\ \frac{1+\alpha^2}{M} \sigma_t^2 & \text{if } X \neq Y, \\ \frac{1}{M} \sigma_t^2 & \text{if no mark is embedded,} \end{cases} \tag{2.15}$$

From the above equations, the main idea of watermarking for images gets clear. For audio domain, the same technique is applicable with some changes [7]. Provided the given algorithm, the statistical imperceptibility is not hard to guarantee as data is embedded to fixed positions of mid-range frequencies regardless of image context.

Having considered the conventional digital watermarking, we re-formulate the problem for DNN watermarking and show 4 main groups the modern DNN watermarking algorithms are based on. It is worth mentioning, that modification induced in DNN have nothing in common with transformations in media. Moreover, straightforward weight modification leads to a dramatic drop in model performance, thus classical algorithms are deemed to be hardly transferable.

2.4 Weight regularization-based approaches

The first idea of weight regularization for data embedding appeared in paper [9], where authors introduced three white-box algorithms for dataset memorization: LSB encoding, Correlated value encoding, and Sign encoding. The algorithms are applicable for watermark embedding as well.

The idea of *LSB encoding* is explicit: the high precision for weight representation is redundant, thus the data could be embedded straight into the least significant bits of weights treating DNN as simple image domain without loss of precision. To the best of our knowledge, it is the only claimed algorithm capable of embedding with no extra training. Other approaches are focused on gradual watermark encoding during training. In Sign Encoding an additional loss term is introduced:

$$C(\theta, s) = -\lambda_c \cdot \frac{\left|\sum_{i=1}^{l}(\theta_i - \overline{\theta})(s_i - \overline{s})\right|}{\sqrt{\sum_{i=1}^{l}\left(\theta_i - \overline{\theta}\right)^2} \cdot \sqrt{\sum_{i=1}^{l}\left(s_i - \overline{s}\right)^2}} \tag{2.16}$$

where θ are flattened model weights, s - message, overline represents mean value, l - number of model parameters, and λ_c is regularization strength. By adding this loss term, the model is optimized both for training task and embedding. Last, Sign Encoding performs embedding with the following loss function:

$$P(\theta, s) = \frac{\lambda_s}{l} \sum_{i=1}^{l} |\max(0, -\theta_i s_i| \tag{2.17}$$

Although, LSB is less robust to weight modification compared to the last two methods, as message and weights are trained to be dependent.

In [10], [11] authors showed the first successful white-box watermarking algorithms resistant to the various model attacks such as fine-tuning and magnitude-based pruning. They also adapted requirements for effective watermarking in DNN:

- *Fidelity*. The performance of the network should not be degraded after embedding;
- *Robustness*. The watermark embedded to DNN should be resistant to fine-tuning and model pruning;
- *Capacity*. The size of the watermark should be capable of storing large amount of information;
- *Security*. The watermark should not be read or modified by unauthorized entity;
- *Efficiency*. The embedding and extraction procedures should be fast.

The weights of CNN are denoted as w_{ijkl} size of $K \times K \times D \times L$ and message as $M \in \{0,1\}^m$ so the embedding is done by adding the following loss term:

$$L_{\text{emb}} = \text{BCE}\left(M, \sigma\left(X \cdot \text{vec}\left(\frac{1}{L}\sum_{l} w_{ijkl}\right)\right)\right) \tag{2.18}$$

So, the task is to perform an additional binary classification on the model's weights. X is a random matrix which is considered to a private key. Extraction is done as follows:

$$M = \left\lceil \frac{1}{2}\left(\max\left(-1, \min\left(1, X \cdot \text{vec}\left(\frac{1}{L}\sum_{l} w_{ijkl}\right)\right)\right) + 1\right)\right\rceil \tag{2.19}$$

Authors in [12] applied the weights regularization to DNN fingerprinting. The proposed *orthogonal fingerprinting* and *coded fingerprinting*. Former implies an additional loss MSE$(f_j - Xw)$, where f_j is a fingerprint assigned to j^{th} user and X is a private random matrix. This was shown to be less resistant to collusion attacks [13]. To alleviate the problem, a fingerprinting based on anti-collusion codes is suggested, but the embedding procedure remains the same.

2.5 Adversarial training watermarking

Another class of watermarking algorithms exploits a main vulnerability of deep neural networks: adversarial attacks. First, we introduce the definition of adversarial attacks to make it clear for reader. Adversarial attack is subtle (in terms of L_p norm) changes applied to the input that can mislead the network and make it misclassify with a high confidence. In [14] it was first shown, that neural network could be be forced to misclassify with small perturbation added to the input. The perturbations are obtained by maximizing the prediction error. In [15], the linear nature of neural networks are discovered, so the adversarial attacks could be done in a more efficient way with Fast Gradient Sign Method. The example of successful adversarial attacks is depicted in Figure 2.1.

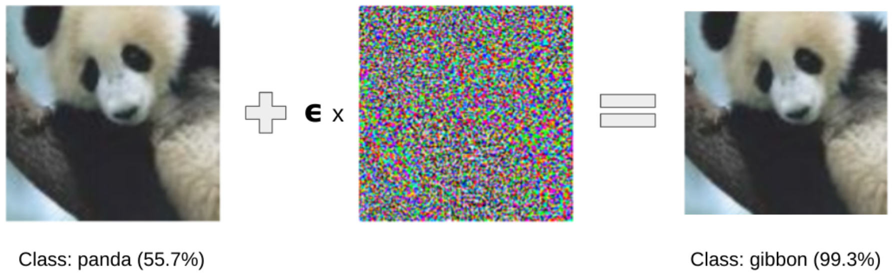
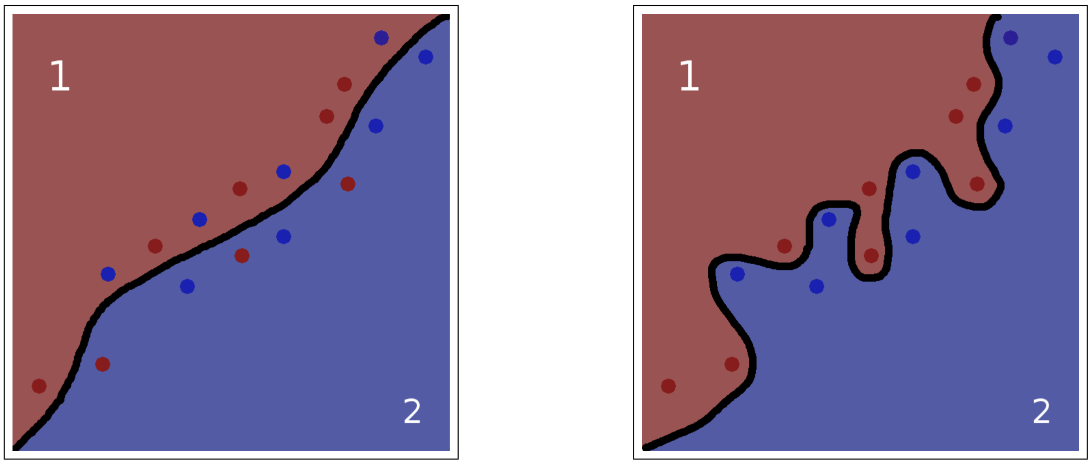


Figure 2.1: An example o adversarial attack originally demonstrated in [15].

To move towards a black-box zero-bit watermarking scheme, authors in [16] proposed a frontier stitching algorithm based on adversarial training.



(a) Before boundaries stitching (b) After boundaries stitching

Figure 2.2: The illustration of algorithm proposed in [16]. Red points represent both true adversaries and false adversaries from **class 1**. Blue points represent both true adversaries and false adversaries from **class 2**. The total key length is $|K| = 16$. After embedding the boundaries are shaped such that true adversaries become false adversaries from corresponding class.

The embedding process is clearly shown in Figure 2.2. From a given training set, some inputs are chosen as false adversaries. It should be noted, that only inputs close to decision boundary are valid and could be considered, as taking points far from a boundary might result in severe performance degradation. Then, fast gradient sign method is applied to a subset of selected inputs such that some of false adversaries become true adversaries. Finally, the model is fine-tuned on the selected images to clamp boundaries: all inputs are classified correctly. Thereby, the described algorithm resembles an embedding process.

To extraction process is simple: we iterate over a selected samples composed from clear inputs and pre-defined adversarial examples, and resemble a vector indicating correct classification m_{ext}. If:

$$\frac{d_H(m_{ext}, m_0)}{|K|} < \theta \tag{2.20}$$

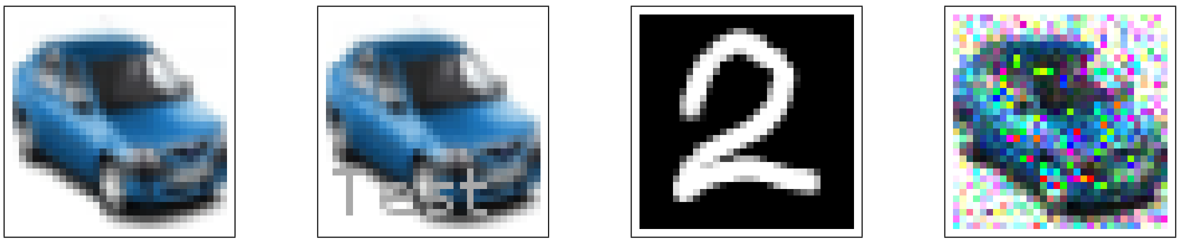
the model is considered to be watermarked. The threshold θ is determined based on probabilistic estimation of false positive outcome: unmarked model resembles stitched boundary.

The approach is very close to adversarial training: while training, the model is fed with adversarial examples to improve robustness to adversarial attacks. Although, the proposed algorithm is of academic interest, the practical outcome might be questioned due to active development of defenses to adversarial attacks.

2.6 Backdoor-based approaches

The watermarking algorithms based on backdooring resemble the largest class. Backdoor is a general attack in computer security. For example, in [17] authors demonstrated a feasible backdoor attack by data poisoning. Neural network backdooring is an attack aiming at misleading the network. The attack is performed in the following way: an adversary feeds some of the samples with maliciously modified labels during the training procedure such that the network memorizes outliers with modified labels. Such behavior is not appropriate and might lead to security issues. Moreover, neural networks were found to be very vulnerable to backdoor attack due to its ability to memorize random information [18].

In [19], authors used backdooring for model's watermarking. The proposal is to mark images from the training set with different contents. The network memorizes marked images and matches it to a pre-defined class. The example is given in Figure 2.3.



(a) input sample (b) related content (c) unrelated content (d) noise content

Figure 2.3: Samples generated for backdooring in [19]. In (a) the initial image is given with label "automobile". In (b) the image is filled with content-related information preserving the initial image; in (c) the sample is replaced with a new one; in (d) the image is filled with random content. All images are re-labeled to "airplane" class.

So the embedding is done via input memorization. The extraction is performed via inputting modified samples to the network and checking an output for a memorized class. The verification could be done in a black-box manner by querying images. Such set of images used for backdooring is also called *trigger set*.

A more formal idea from a cryptographic point of view was presented in [20], where backdooring in combination with a commitment scheme from cryptography is used. The work is rather theoretical and introduces a more formal way to approach watermarking by backdooring. In the paper, authors considered ResNet-18 trained on CIFAR-10, CIFAR-100 and ImageNet, which provides rather stronger justification for results. The embedding process resembles simple training on trigger set with pre-defined labels along with key generation. The verification is done via querying the model and checking the key.

Another work was done [21], where a multi-bit watermarking based on backdooring was considered. The main idea is presented algorithm below:

Algorithm 1: The algorithm for multi-bit watermarking presented in [21]

```
**Input:** model $M$, training set $\{X^{train}, Y^{train}\}$, Key length $K$, Number of classes $C$
**Result:** Marked model $M^*$, key set $\{X_{key}, Y_{key}\}$
$A = \left\{ a_j = \frac{\sum_k M\left(\{x_k^{train} : y_k^{train} = c_i\}\right)}{\left|\{x_k^{train} : y_k^{train} = c_i\}\right|} \ : \ j = 1...C \right\}$;
$C$ = k_mean_cluster($A$, 2);
$K' = K \times 10$;
$\{X_{key'}, Y_{key'}\}$ = gen_wm_key($M$, $\{X^{train}, Y^{train}\}$, $C$, $K'$) ;
$M^*$ = fine_tune($M$, $\{X^{train}, Y^{train}\}$, $\{X_{key'}, Y_{key'}\}$, $L_R$) ;
$M_T = \{$fine_tune$(M_i, \{X^{train}, Y^{train}\}, \{X_{key'}, Y_{key'}\}, L_0) \ : \ i = 1...T\}$ ;
$I^* =$ matching_indices $(M^*(X_{key'}), Y_{key'})$ ;
$I =$ mismatching_indices $(M(X_{key'}), Y_{key'})$ ;
$I_T = \{$mismatching_indices $(M_i(X_{key'}), Y_{key'}) \ : \ i = 1...T\}$ ;
$I_{key} = I \cap I^* \cap I_T$ ;
$\{X_{key}, Y_{key}\} =$ random_subset$(\{X_{key}, Y_{key}\}_{I_{key}}, K)$ ;
```

A couple of notes should be done here: **gen_wm_key** generates targeted adversarial examples related to the class in the chosen cluster, **fine_tune** performs fine-tuning on a given training set. Once the matching indices for fine-tuned model is obtained, highly transferable mismatching indices are found (during fine-tuning with loss L_0). Finally, only indices that are changed after fine-tuning with L_R and unchanged during fine-tuning with L_0 are left. The loss function L_R is:

$$L_R = L_0 + \lambda * d_H(\mathbf{s}, \text{decode\_predictions}(Y_M^{key}, C)) \tag{2.21}$$

The algorithm could be also attributed to a class of adversarial training-based watermarking schemes. Moreover, authors claim that such watermarking embedding yields better robustness to adversarial attacks. The extraction is done via simple querying X_{key} and decoding bits with given cluster centers C.

A rather interesting approach was presented in [22]. Authors crafted patterns with value range much higher than allowed input image range. These patterns are used for training alongside with original dataset. Once pattern is added to the input, the network outputs a pre-trained class.

Finally, in [23] authors raised a question of evasion attacks for remote watermark verification. Indeed, an adversary might estimate the distribution of input for the model and discard any queries which are in far distance. To address the problem, the suggested stacking three neural networks. The first network is supposed to modify the input trigger set (generator) such that the second network (discriminator) is not able to distinguish between trigger set and ordinary samples. Meanwhile, the third network (network to be watermarked or host) is trained on samples both from generator and training set. In that way, the trigger set learns small perturbations which do not withdraw samples from original distribution and make the network misclassify them.

2.7 Layers output regularization

The next class of watermarking algorithms are based on embedding to the layers activation maps rather than to the weights directly. In [24], embedding is done with fitting output distribution to Gaussian Mixture which mean values store the signature. Total loss for neural network is structured as follows:

$$L = L_0 + \lambda_1 L_{fit} + \lambda_2 L_{ext} \tag{2.22}$$

Suppose, $m \in \{0,1\}^{s\times N}$ is a signature, A is random matrix mapping Gaussian centers, and $\mu^{s\times M}$ is matrix encoding Gausssian centers. As information is embedded to the Gaussian centers (M is a feature space size at particular layer, s - class number), decoding loss is:

$$L_{ext} = \text{BCE}(m, \sigma(\mu \cdot A)) \tag{2.23}$$

To fit a particular layer distribution, L_{fit} is defined in the following way:

$$L_{fit} = \|\mu^l_{y^*} - f^l(x,\theta)\|^2_2 - \sum_{i\neq y^*} \|\mu^l_i - f^l(x,\theta)\|^2_2 \tag{2.24}$$

where hard positives outputs are approximated with $\mu^l_{y^*}$ and hard negatives with μ^l_i (l - number of layer) for each class respectively. So, the effect of this loss is similar to the effect of triplet loss: minimize inner class diversity while increasing between-class distance. Eventually, all Gaussian distributions for each class are expected to be well-separable. For extraction, the signature is obtained as follows:

$$m = \lceil \mu \cdot A \rceil \tag{2.25}$$

The embedding could be applied to different layers output, even for the last one enabling black-box watermarking. For black-box case, the embedding continues: after training, the samples that lie outside of the learned distribution (called rarely explored regions) are selected and model is fine-tuned such that they are well-classified now. After a similar filtering procedure as in multi-bit watermarking, the key samples are stored for verification. The extraction is carried out by inferring predicted classes and checking with correct ones.

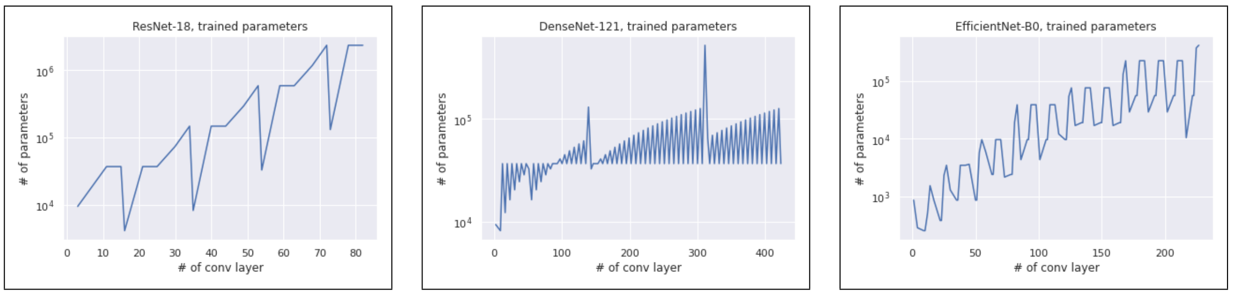
Chapter 3

Random noise injection

The main idea of this work is to consider watermark embedding with no extra training. In previous works, we showed that the main work direction in DNN watermarking is model re-training and fine-tuning with an additional loss. First of all, to approach our problem, we try to embed the random noise to the model's weights. The random injection will help to understand how various models are robust to random modifications. All experiments are done on MNIST, CIFAR-10, and CIFAR-100 datasets. All networks were pre-trained on ImageNet datasets so that input resized to 224×224 to achieve public results.

3.1 State-of-the-art deep convolutional neural networks

In all experiments, three types of convolutional neural networks will be considered: ResNet [2], DenseNet [3], and EfficientNet [4]. As the main part of the listed networks are convolutions, let's measure the number of parameters for each layer. The results are given in Figure 3.1.



(a) ResNet-18 (b) DenseNet-121 (c) EfficientNet-B0

Figure 3.1: Number of parameters in convolutional layers. The most light-weight version in each class is chosen for analysis.

In ResNet, drops in parameters number is a due to use of skip connections, which transfer input to the output with less number of parameters than used in the skipped branch. In DenseNet, triangular-like behavior is due to concatenation applied after each branch increases the number of channel. To understand which noise might less affect the network performance, let's draw the weights distribution at three layers: at the beginning, in the middle and at the end. The results for all networks trained on CIFAR-100 are reported in Figures 3.2, 3.3, 3.4.

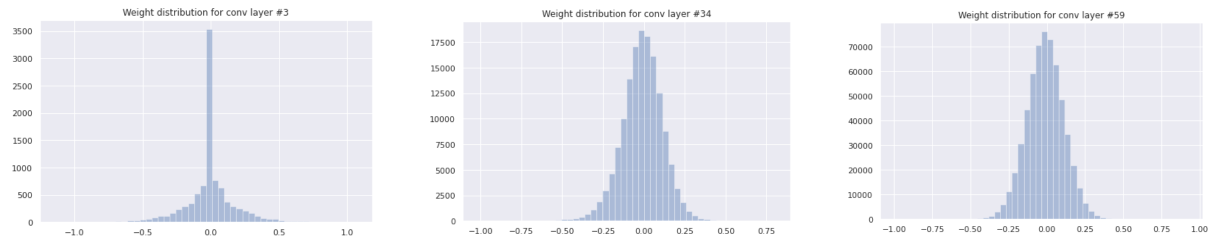


Figure 3.2: ResNet-18. Weight distribution in convolutional layers.

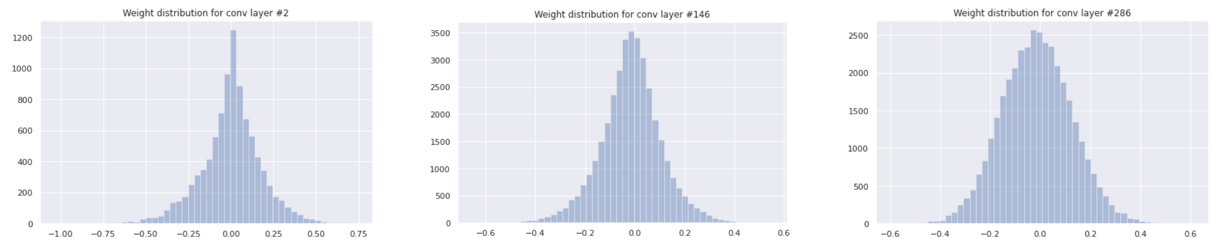


Figure 3.3: DenseNet-121. Weight distribution in convolutional layers.

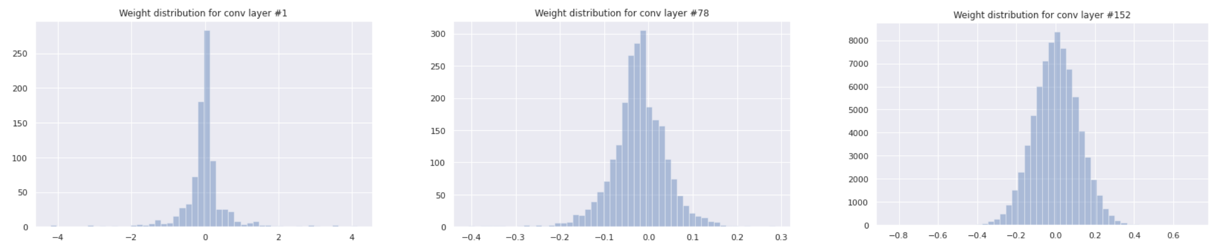


Figure 3.4: EfficientNet-B0. Weight distribution in convolutional layers.

According to the plots, the best approximation for weights would be Gaussian distribution. In order to keep the initial distribution as close as possible, we add random perturbations from normal distributions and evaluate model's performance.

3.2 Noise injection with normal distribution

To test the hypothesis that DCNN is robust to small perturbations, we use the following algorithm:

```
Algorithm 2: Noise injection
    Input: $M$ - model, $\{X^{val}, Y^{val}\}$, $\alpha$ - amplitude ;
    Result: Accuracy list $a$
    $C$ = list_of_conv_layers($M$) ;
    $a = []$ ;
    for $c_i \in C$ do
        $r$ = random_like($c_i, \alpha$) ;
        $c_i = c_i + r$ ;
        $a_i$ = accuracy($M(X^{val}), Y^{val}$) ;
    end
```

where **random_like** generates normal noise $(0, \alpha \text{std}(c_i))$. The routine **list_of_conv_layers** reports only convolutional layers with input channels more than 3 and kernel size more than 1×1. Original accuracy is shown in Table 3.1.

Dataset	ResNet-18	DenseNet-121	EfficientNet-B0
MNIST	0.996	0.995	0.992
CIFAR-10	0.950	0.958	0.966
CIFAR-100	0.763	0.807	0.829

Table 3.1: Models performance

Then trained models are used in Algorithm 2. The results for different α is reported in Table 3.2.

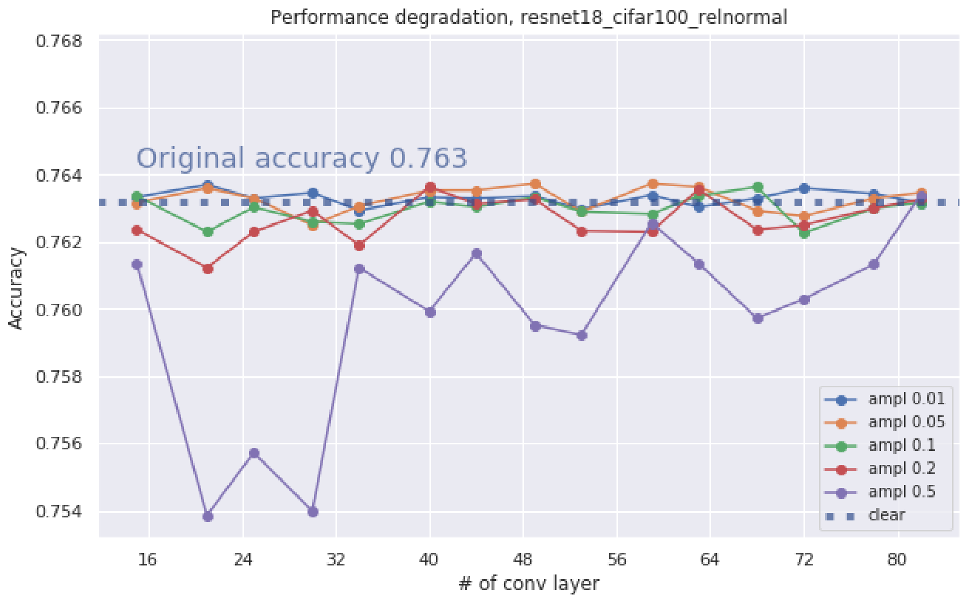
Magnitude (α)	Dataset	ResNet-18	DenseNet-121	EfficientNet-B0
0.01	MNIST	0.996 (0.0%)	0.995 (0.0%)	0.992 (0.0%)
	CIFAR-10	0.950 (0.0%)	0.958 (0.0%)	0.966 (0.0%)
	CIFAR-100	0.763 (0.0%)	0.807 (0.0%)	0.829(0.0%)
0.05	MNIST	0.996 (0.0%)	0.995 (0.0%)	0.922 (0.0%)
	CIFAR-10	0.949 (0.1%)	0.958 (0.0%)	0.964 (0.3%)
	CIFAR-100	0.763 (0.0%)	0.807 (0.1%)	0.827 (0.3%)
0.1	MNIST	0.996 (0.0%)	0.995 (0.0%)	0.992 (0.0%)
	CIFAR-10	0.948 (0.2%)	0.958 (0.0%)	0.959 (0.8%)
	CIFAR-100	0.760 (0.5%)	0.806 (0.1%)	0.820 (1.1%)
0.2	MNIST	0.996 (0.0%)	0.995 (0.0%)	0.992 (0.0%)
	CIFAR-10	0.942 (0.8%)	0.957 (0.1%)	0.916 (5.2%)
	CIFAR-100	0.748 (1.9%)	0.804 (0.4%)	0.794 (4.3%)
0.5	MNIST	0.996 (0.0%)	0.995 (0.0%)	0.954 (3.8%)
	CIFAR-10	0.844 (11.1%)	0.949 (0.9%)	0.811 (16.1%)
	CIFAR-100	0.546 (28.4%)	0.789 (2.3%)	0.594 (28.4%)

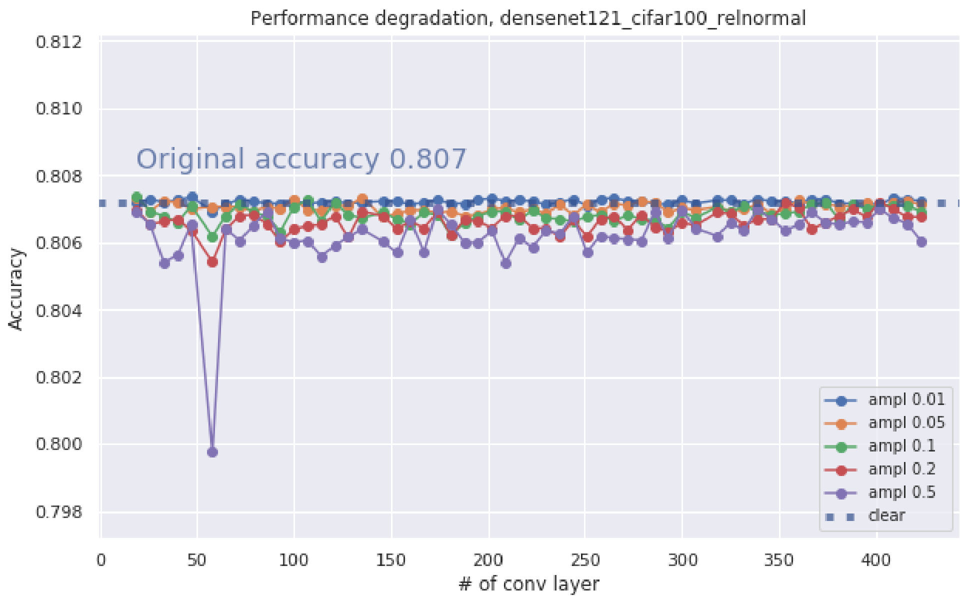
Table 3.2: Mean accuracy after adding random perturbations to each layer. Relative accuracy drop is given in parenthesis.

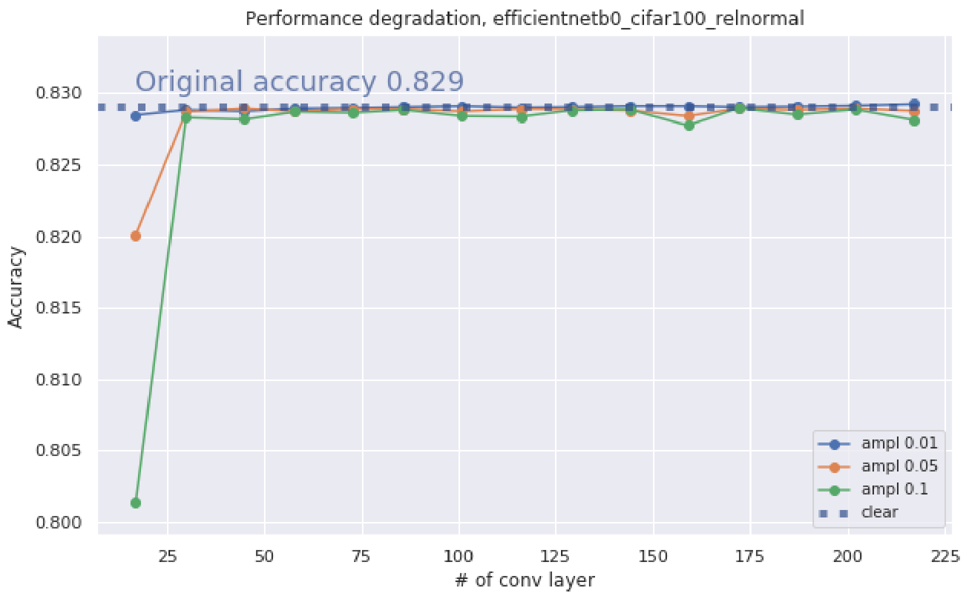
The results show that the models are able to resist noise perturbation with α up to 0.1 on 1.1% drop. Moreover, the drop tends to be relatively larger for more complex datasets: drop on CIFAR-100 is several times greater than on CIFAR-10. Weights trained on MNIST does not degrade at all. To make the following experiments closer to the real-world scenario, we constrain datasets only to CIFAR-100 and assume that a working solution will easily transfer to a less complex classification (CIFAR-10 and MNIST).

In Figure 3.5, contribution of each convolutional layer is given to performance degradation. From the picture it could be seen, that initial layers affect more than mid-layers and high-layers.

Figure 3.5: Performance degradation on CIFAR-100. Per-convlayer dynamics.







Chapter 4

Watermark embedding algorithm

Having demonstrated that convolutional neural networks are not much sensitive to perturbations, we try to adapt the algorithm from [10], [11] for watermark embedding. In fact, authors already considered the case with embedding with no extra training, but provided rather poor consideration for the problem. Later, we show that the followed-up algorithm is able to perform embedding treating the network's weights as a simple data array. Moreover, the efficiency of the proposed algorithm tends to be higher for deeper models. Deeper models exhibit a greater number of trainable parameters; when the number of parameters exceeds the size of training set substantially, the network is called over-parameterized. Thus, the deeper models are, the more beneficial to embed data with no additional training.

Without additional constraints, neural networks could not be treated as simple images due to its complex structure. In Chapter 2, we provided a general description for watermarking embedding:

$$f(C, m, k) \to C' \text{, s.t. } d(C, C') < \epsilon \tag{4.1}$$

The function d for DNN domain takes the the following meaning:

$$d(C, C') = \max\left[\text{acc}(C(X^{val}), Y^{val}) - \text{acc}(C'(X^{val}), Y^{val}), 0\right] \tag{4.2}$$

So after embedding, the drop in model's performance should stay within a pre-defined range ϵ. The extraction process remains the same as in the general description. The threshold θ should be estimated based on a probabilistic approach.

Before describing the algorithm, a proper description of a threat model should be done. Indeed, the exact knowledge of possible attack set-ups introduces limiting conditions the problem should be solved within.

4.1 Threat model

Suppose a company trains a model M on dataset $\{X^{train}, Y^{train}\}$, embeds a watermark, and distributes the model to a certified partner. An adversary obtains the model from the company's partner by any means. At some point, the model was found at the adversary's repository or was requested by legal authorities. The company extracts a watermark from the model and verifies it with an expected signature. In this work, we limit the use-case to white-box watermark retrieval as the first step toward training-less embedding. On the other hand, an adversary wishes to alter the watermark and keep the model's performance within δ-interval. Such modifications were denoted as g. One of the most popular attacks he can perform is *fine-tuning* and *model-pruning*, which are explained in detail in Chapter 5.

With model stealing, the adversary obtains access to a fixed part of a dataset, which is called *leaked*. The fine-tuning and model-pruning attacks are performed on the leaked dataset. The use of a leaked dataset is justified by the fact that additional training on a dataset significantly smaller from the original one should be done on a set with matching distribution otherwise, performance might drop severely. If an adversary is able to gather relatively large dataset and fine-tune the model for a long time, we assume that he owns high-performance hardware and have plenty of time resources thus stealing does not bring any profits to him.

A quite effective attack for watermark removal is called *transfer learning*: a network trained for one classification task is used for fine-tuning on the other classification task. In this work, we do not consider this attack due to its doubtful negative impact. If the network is trained for the other classification problem, it cannot compete in the original problem and weaken the company's market position.

Another possible attack, discussed in [25], is *distillation attack*: training samples are fed to the target network (directly or via API), and its output is used to train a model smaller in size. Although it is quite an effective attack for watermark removal, the feasibility is still an open question: long training time for deep models and access to large and diverse datasets are required.

An adversary might also try to forge the watermark, thus claiming authority and erasing the watermark completely. This kind of attack is called *forgery attack or over-writing attack* and is considered in a per-layer embedding paragraph.

To sum up, the following description best fits the adversary: an intention to use the stolen model for his benefit due to the inability to train the model from scratch.

4.2 Per-layer embedding & extraction algorithm

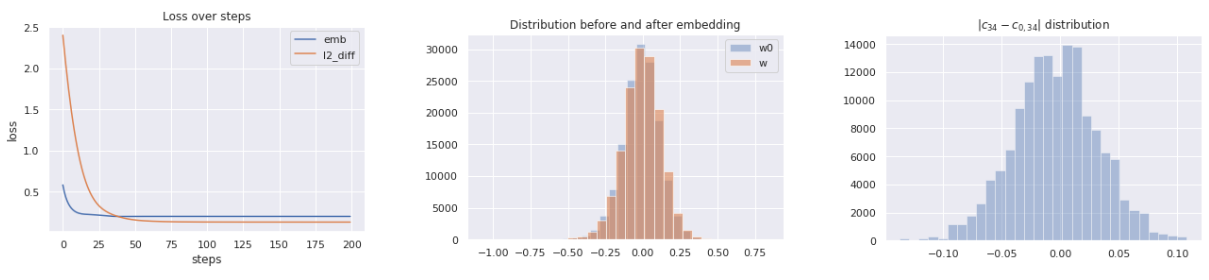
Let us describe the general framework for the embedding procedure. The following task is given: a model owner needs to embed the message $s \in \{0,1\}^m$ in model M such that additional training of the whole model is not required. Given that model M consists of convolutional layers $\{c_{i0}\}$, the task can be considered as the following optimization problem for one convolutional layer $c_{i0} \in \mathbb{R}^{S \times S \times D \times L}$:

$$\begin{gathered}\text{Emb}(c_{i0}, s, R) = \arg \min_{c_i \in \mathbb{R}^{S \times S \times D \times L}} \left[\lambda_1 \|\text{vec}(c_i) - \text{vec}(c_{i0})\|_2^2 + \lambda_2 L_{emb}(c_i, s, R)\right] \\ \text{Ext}(c_i, R) = s\end{gathered} \tag{4.3}$$

where R is a random real matrix, and extraction is done in the same way it was done in [10], [11]. The coefficient λ_1, λ_2 regulate the optimization strength for each term. Loss function L_{emb} is responsible for signature embedding with random key R to the weights of convolutional layer c_i. L_{emb} could be constructed in a quite straightforward way:

$$L_{emb} = \text{BCE}(s, \sigma(R \cdot \text{vec}(\overline{c_i})) \tag{4.4}$$

where averaging is done over the last dimension. The average could be either done over input channels. This introduces robustness to slight modifications. In Figure 4.1, the example of the embedding process is given for ResNet-18 convolutional layer and signature size of 512. Random matrix R is sampled from normal distribution $(0,1)$, λ_1 is set to be 100, and λ_2 is set to be 1.



(a) Loss during optimization (b) Distribution of layer's weights (c) Difference distribution

Figure 4.1: Per-layer embedding. ResNet-18, layer #34, $m = 512$

Parameter λ_1 ensures that matrices are kept close. We can experiment with this value to find the best option. Also, a set the random matrix R samples from might affect the optimization.

Let's denote **diff2** as random filling of matrix R with values $\{-1, 0, 1\}$; **diff3** - random filling with values $\{-10, 0, 10\}$; and **normal** fills with samples from normal distribution (0,1). The first fill mode embeds data in value differences; the second fill mode does the same but keeps values closer to the distribution of the initial weight as differences are amplified. At the same time, a trade-off between the robustness of embedded watermark and the model's performance should be chosen: **diff3** utilizes smaller value differences so that the watermark could be easily damaged. Each filling is used with different λ_1 parameters taken from 0.1 to 100.

fill mode	params	$\lambda_1 = 0.1$	$\lambda_1 = 1$	$\lambda_1 = 10$	$\lambda_1 = 100$
m = 512					
diff2	BER	0.0	0.0	0.0	0.0
	α	0.547	0.379	0.276	0.182
	Δ_{max}	0.555	0.181	0.124	0.082
diff3	BER	0.0	0.0	0.0	0.0
	α	1.087	0.300	0.063	0.054
	Δ_{max}	0.853	0.413	0.032	0.025
normal	BER	0.0	0.0	0.0	0.00
	α	0.517	0.324	0.240	0.164
	Δ_{max}	0.51	0.167	0.116	0.077
m = 256					
diff2	BER	0.0	0.0	0.0	0.0
	α	0.498	0.283	0.212	0.145
	Δ_{max}	0.506	0.165	0.108	0.071
diff3	BER	0.0	0.0	0.0	0.0
	α	1.14	0.422	0.048	0.041
	Δ_{max}	0.874	0.49	0.038	0.019
normal	BER	0.0	0.0	0.0	0.00
	α	0.504	0.24	0.185	0.129
	Δ_{max}	0.515	0.122	0.088	0.059

Table 4.1: Per-layer embedding with different parameters for ResNet-18, convlayer #72

The results of per-layer embedding with different parameters are shown in Table 4.1. The meaning of each measured value for fixed λ_1 is explained below:

$$\begin{gathered} \text{BER} = \frac{d_H(s_{out}, s)}{m} \\ \alpha = \sqrt{\frac{\|\text{vec}(c_i) - \text{vec}(c_{i0}\|_2^2}{S^2 \cdot D \cdot L \cdot \text{std}^2(c_{i0})}} \\ \Delta_{max} = \max(\|c_i - c_{i0}\|) \end{gathered} \tag{4.5}$$

Indeed, according to the Figure 4.1 values added to the initial weights resemble a normal distribution, so the point of α is to display the ratio of the standard deviation of added perturbation to standard deviation of initial weights. Having known α, we can estimate possible drop from Figure

3.5.

From the Table 4.1 we learn that bigger λ_1 helps to achieve closer distribution for any fill mode. Moreover, **diff3** preserves distribution closer to the initial than **normal** and **diff2**. The smaller the signature length the smaller perturbations are induced. Also, **diff3** provides better capacity. To figure out which fill mode is more stable, we perform a watermark over-writing attack. The attack is performed k times, and at each step, the message is extracted back with BER, α, and Δ_{max} calculated. Assuming that algorithm is known, an adversary can run it over all layers several times to remove the watermark with different randomly generated private keys (original key is kept secretly). To simulate this attack we embed the watermark with $m = 512$ and over-write with larger ($m = 512$) and smaller key size ($m = 256$). The results are presented in Table 4.2.

fill mode	params	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
$m = 512 \to m = 256$							
normal	BER	0.0	0.0	0.011	0.042	0.065	0.106
	α	0.163	0.206	0.24	0.266	0.291	0.313
	Δ_{max}	0.074	0.098	0.114	0.135	0.146	0.156
diff3	BER	0.0	0.004	0.036	0.072	0.096	0.145
	α	0.053	0.066	0.076	0.085	0.093	0.098
	Δ_{max}	0.025	0.035	0.035	0.039	0.046	0.043
$m = 512 \to m = 512$							
normal	BER	0.0	0.005	0.053	0.11	0.147	0.184
	α	0.162	0.225	0.270	0.306	0.335	0.36
	Δ_{max}	0.074	0.106	0.14	0.145	0.16	0.174
diff3	BER	0.0	0.031	0.098	0.165	0.192	0.224
	α	0.052	0.076	0.091	0.102	0.113	0.121
	Δ_{max}	0.024	0.033	0.044	0.046	0.055	0.053
$m = 512 \to m = 1024$							
normal	BER	0.0	0.029	0.105	0.202	0.266	0.291
	α	0.162	0.252	0.305	0.345	0.376	0.401
	Δ_{max}	0.078	0.127	0.146	0.179	0.174	0.182
diff3	BER	0.0	0.097	0.173	0.269	0.277	0.370
	α	0.055	0.091	0.11	0.126	0.138	0.148
	Δ_{max}	0.025	0.047	0.054	0.063	0.065	0.074

Table 4.2: Watermark over-writing attack for ResNet-18, convlayer #72 (2M params)

The results exhibit the following fact: the larger the number of over-writings, the farther the distribution, so the larger the drop and larger bit-error rate. In other words, it is unfeasible to perform over-writing attacks continuously as model performance will eventually degrade severely. **normal** fill mode provides better unforgeability, as more attacks lead to more deteriorated performance and less bit-error-rate compared to **diff3**. Hereinafter, we use **normal** fill mode for the private key matrix. We also perform an over-writing attack for convlayer #15 for comparison and present results in Table 4.3. With a significantly lower amount of parameters to be used for embedding, the attack is easier to do, but this could be alleviated by skipping such layers or simply embedding less amount of data, as shown in Table 4.4.

fill mode	params	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
$m = 512 \rightarrow m = 1024$							
normal	BER	0.099	0.231	0.315	0.385	0.432	0.469
	α	0.282	0.309	0.322	0.331	0.333	0.342
	Δ_{max}	0.104	0.126	0.131	0.138	0.12	0.116
diff3	BER	0.0	0.436	0.499	0.505	0.509	0.514
	α	0.247	0.277	0.279	0.275	0.28	0.282
	Δ_{max}	0.084	0.099	0.1	0.106	0.105	0.1

Table 4.3: Watermark over-writing attack for ResNet-18, convlayer #15 (37K params)

fill mode	params	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
$m = 64 \rightarrow m = 128$							
normal	BER	0.0	0.021	0.073	0.135	0.286	0.240
	α	0.285	0.414	0.502	0.569	0.602	0.562
	Δ_{max}	0.121	0.149	0.177	0.207	0.225	0.238
diff3	BER	0.0	0.047	0.125	0.229	0.240	0.344
	α	0.099	0.17	0.21	0.225	0.256	0.266
	Δ_{max}	0.036	0.066	0.085	0.082	0.098	0.1

Table 4.4: Watermark over-writing attack for ResNet-18, convlayer #15 (37K params)

So, in this paragraph, we showed the feasibility of embedding with no extra training, chosen the best fill mode for private key matrix, and balanced the optimization parameters. We considered ResNet-18 for all experiments as at this point, we are interested only in the weights. Weight deviation directly affects the model performance, so the task is to keep the weights as close as possible to the initial distribution. In Table 4.5, mean acc drop and max BER are measured to test the performance of the networks with embedded watermark at a different length. Embedding at each layer does not seem to be feasible as performance drops too high (especially for EfficientNet-B0). Next, we embed the watermark at each layer such that the model performance is taken into consideration.

m	params	ResNet-18	DenseNet-121	EfficientNet-B0
16	Mean acc drop	0.2%	0.1%	3.7%
	Max BER	0.0	0.0	0.0
64	Mean acc drop	0.1%	0.1%	6.8%
	Max BER	0.0	0.0	0.2
128	Mean acc drop	0.4%	0.1%	7.0%
	Max BER	0.0	0.0	0.3
256	Mean acc drop	0.5%	0.2%	10.1%
	Max BER	0.0	0.0	0.4
512	Mean acc drop	3.3%	0.2%	7.8%
	Max BER	0.0	0.0	0.4

Table 4.5: Models average performance after embedding at each convlayer separately

4.3 Per-network embedding & extraction algorithm

In order to maximize embedding capacity (one of the criteria for effective watermark embedding), more layers should be subjected to watermark embedding. To do so, we propose the following complete algorithm:

Algorithm 3: Additive embedding with performance check

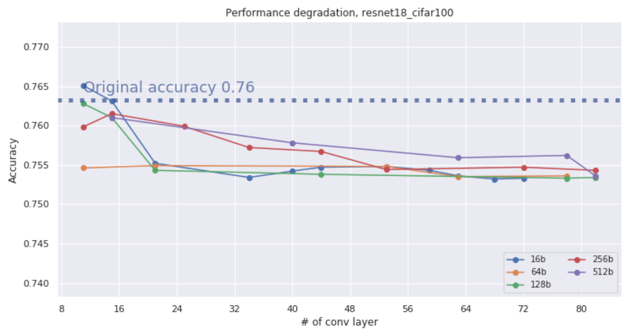
```
Input: $M$ - model, $\{X^{val}, Y^{val}\}$, $\epsilon$ - tolerance ;
Result: Watermarked model $M'$, private keys $p$, signatures $sign$
$C$ = list_of_conv_layers($M$) ;
$p = []$ ;
$sign = []$ ;
for $c_{i0} \in C$ do
    $R_i$ = random_normal$(0, 1)$ ;
    $s_i$ = random_int$(0, 1)$ ;
    $c_i$ = Emb$(c_{i0}, s_i, R_i)$ ;
    $M'$ = embed_conv_layer$(c_i)$ ;
    if $d(M, M') < \epsilon$ then
        $p$.add$(R_i)$ ;
        $sign$.add$(s_i)$ ;
    else
        $M'$ = embed_conv_layer$(c_{i0})$ ;
    end
end
```

The extraction procedure is pretty straightforward; we iterate over layers with embedded watermark and retrieve it. Private kyes contain not only the random matrices but also a number of convolutional layers:

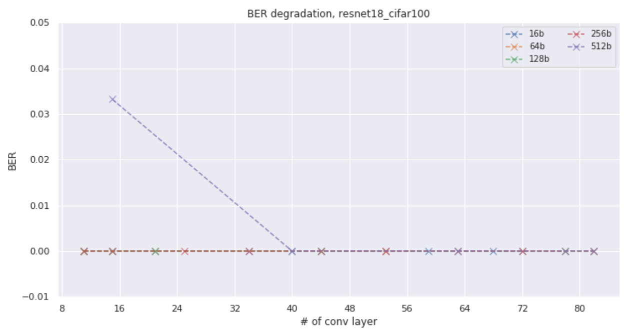
Algorithm 4: Additive extraction

```
Input: $M$ - model, private keys $p$ ;
Result: Signatures $sign$
$C$ = embedded_conv_layers$(M, p)$ ;
$sign = []$ ;
for $c_i \in C$ do
    $s_i$ = Ext$(c_i, p_i)$ ;
    $sign$.add$(s_i)$ ;
end
```

Now, we try to embed watermarks with different key lengths and see models' behavior. The example for ResNet-18 is given in Figure 4.2. First, it could be observed that performance gets the lower, the more information was embedded. Second, smaller signatures are embedded earlier so that accuracy drops faster. Larger signatures tend to occupy layers closer to the middle, so experience less dramatic accuracy drop. BER keeps stable and close to 0 for all layers.



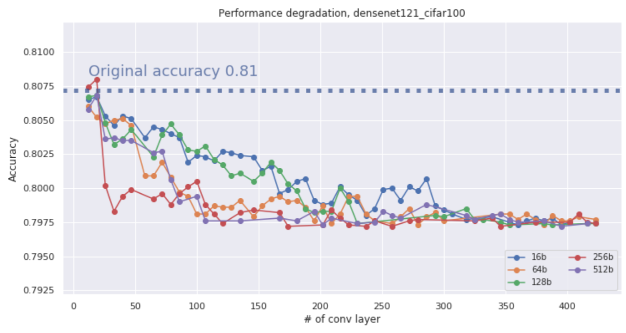
(a) Accuracy degradation



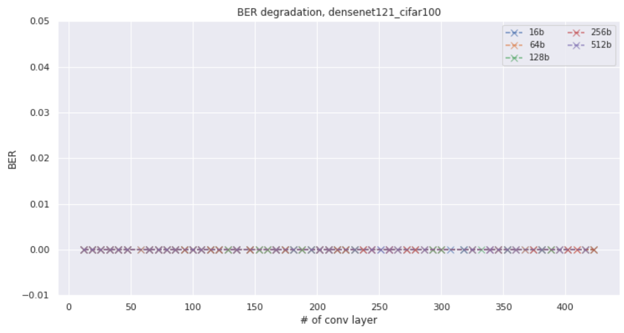
(b) BER degradation

Figure 4.2: Additive embedding for ResNet-18. Different key lengths, $\epsilon = 0.01$

The same experiment is done for DenseNet-121 and EfficientNet-B0 and presented in Figures 4.3 and 4.4. The total watermark capacity is presented in Table 4.6 for each network.

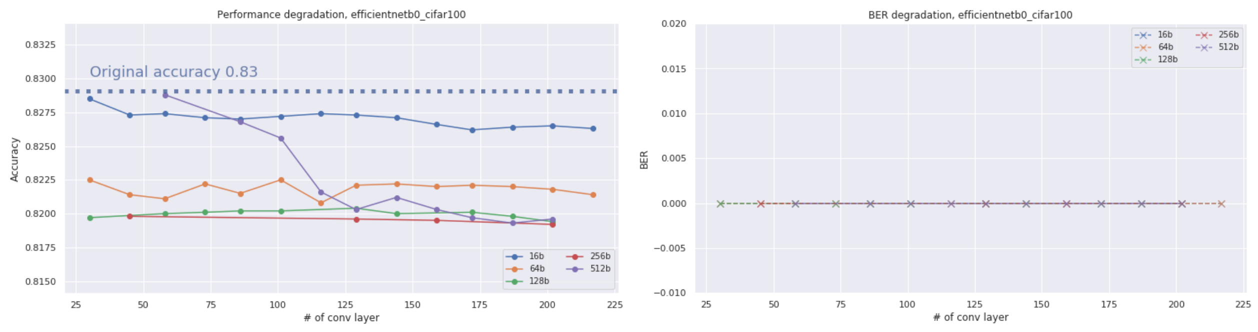


(a) Accuracy degradation



(b) BER degradation

Figure 4.3: Additive embedding for DenseNet-121. Different key lengths, $\epsilon = 0.01$



(a) Accuracy degradation (b) BER degradation

Figure 4.4: Additive embedding for EfficientNet-B0. Different key lengths, $\epsilon = 0.01$

Both networks exhibit quite similar behavior. For example, the algorithm embedded less amount of 256bit watermarks to EfficientNet-B0 than 512bit watermarks because the embedding for the former started earlier. However, the 16bit watermark almost did not affect performance. This gives a hint that better strategies could exist to fulfill capacity criterion more effectively.

m	ResNet-18	DenseNet-121	EfficientNet-B0
$\epsilon = 0.01$			
16	229 (90%)	928 (100%)	224 (88%)
64	405 (40%)	3712 (100%)	832 (81%)
128	939 (46%)	4523 (61%)	1664 (81%)
256	1707 (42%)	7680 (52%)	1877 (46%)
512	3072 (38%)	10752 (36%)	4779 (58%)
$\epsilon = 0.005$			
16	176 (69%)	917 (99%)	224 (88%)
64	277 (27%)	3541 (95%)	619 (60%)
128	512 (25%)	3157 (43%)	1408 (69%)
256	1621 (40%)	8021 (54%)	2560 (62%)
512	1707 (21%)	6997 (24%)	4608 (56%)

Table 4.6: Max watermark capacity estimation for different signature lengths (in bits). The percentage of utilized layers are given in parenthesis. Larger signatures store more total information and utilize less amount of block.

4.4 Adaptive embedding

Previously it was shown that smaller signatures utilize more network layers than bigger ones. Based on this, we propose to use a dynamic watermark size. The embedding procedure is similar except for a few modifications. The signature size is set initially to 512 and is decreased every time accuracy drops too much and increases when it stays within a tolerable range. To demonstrate the feasibility of our algorithm, we perform embedding with adaptive watermark size on seven state-of-the-art networks with different depths pre-trained on ImageNet [26]. The forward algorithm starts from the first layer, while the backward from the last. Overall, the deeper the model within a network family, the larger watermark could be embedded. The backward algorithm yields better capacity; therefore, the last layers affect less final performance.

Embedding direction	Network	Total layers	Used layers	Max signature size	Mean signature size
$\epsilon = 0.01$					
forward	ResNet-18	21.35%	7.00%	32	21
	ResNet-34	19.88%	10.41%	64	53
	ResNet-101	8.68%	5.06%	64	37
	EfficientNet-B0	6.87%	41.69%	1536	992
	EfficientNet-B1	6.87%	44.91%	3008	1931
	EfficientNet-B3	6.84%	44.88%	1856	1781
	DenseNet-121	13.52%	6.90%	80	64
backward	ResNet-18	21.35%	5.26%	32	27
	ResNet-34	19.88%	4.16%	512	347
	ResNet-101	8.68%	7.06%	560	539
	EfficientNet-B0	6.87%	56.25%	4352	4288
	EfficientNet-B1	6.67%	56.52%	6400	6021
	EfficientNet-B3	6.84%	73.08%	9472	9472
	DenseNet-121	13.52%	5.17%	784	720
$\epsilon = 0.02$					
forward	ResNet-18	21.35%	12.26%	32	37
	ResNet-34	19.88%	7.28%	80	91
	ResNet-101	8.68%	9.09%	144	101
	EfficientNet-B0	6.87%	54.19%	3520	2608
	EfficientNet-B1	6.87%	63.78%	6848	4949
	EfficientNet-B3	6.84%	50.00%	5376	3243
	DenseNet-121	13.52%	4.02%	144	101
backward	ResNet-18	21.35%	7.00%	528	432
	ResNet-34	19.88%	11.47%	560	496
	ResNet-101	8.68%	12.12%	672	608
	EfficientNet-B0	6.87%	64.56%	5376	5205
	EfficientNet-B1	6.87%	65.22%	7424	7296
	EfficientNet-B3	6.84%	75.65%	10240	9899
	DenseNet-121	13.52%	7.47%	1680	1317

Table 4.7: Max watermark capacity estimation for adaptive signature lengths (in bits) for networks trained on ImageNet. Total layers is a ratio to all layers that could be used for embedding. Used layers is a ratio of layers used for embedding to total layers. Embedding is performed three time, so max capacity and mean capacity are reported.

Finally, we need to discuss the efficiency of the proposed algorithm and why it is beneficial to use it. In Table 3.1 we report embedding time for each network. It could be further reduced by reducing the number of the optimization step up to 5 times with no loss in used metrics. Training any of the presented networks for the same classification task would take a couple of hours. As all presented approaches require training, our embedding algorithm is more efficient in terms of main watermarking criteria (efficiency). Moreover, the network performance is kept withing the fidelity range, which could be manually set, so fidelity property is met. As the algorithm requires a private key, the watermark could be read by an unauthorized party and modified (security is preserved). The capacity property was already extensively discussed in tables.

ResNet-18	ResNet-34	ResNet-101	EfficientNet-B0	EfficientNet-B1	EfficientNet-B3	DenseNet-121
6m	15m	40m	8m	17m	40m	60m

Table 4.8: Watermark embedding time. Computations are done on single NVIDIA Tesla V100.

Chapter 5

Attacks against digital watermarking in DNN

Regardless of superiority in all watermarking criteria, the watermarking method is not applicable if it does not resist attacks maliciously done by an adversary. In this chapter, we consider three possible attacks: fine-tuning, model pruning, and over-writing or forgery attack. Unfeasibility of other attacks was already justified in threat model discussion.

5.1 Fine-tuning

By fine-tuning, we imply additional training on unseen data for the same classification task. We use 10% of CIFAR-100 training set (*leaked set*) and fine-tune networks with embedded watermarks for at least ten additional epochs till accuracy stabilizes. The learning rate should be set in such a way that it removes the watermark and preserves fidelity. After fine-tuning, the watermarked is extracted with private key and decoding loss, BER, and model performance are recorded. It should be noted that we consider the worst-case scenario with a watermark size of 512 and a learning rate starting from 10^{-2}. First, we start with ResNet-18 and report change of decoding loss in 5.1 and BER in A.1. Though the accuracy is degraded, BER did not exceed 0.1, so the watermark is deemed to be permanent. Interestingly, decoding loss mostly degrades at the beginning: loss stayed at the same level, starting from 18^{th} epoch. Moreover, the largest degradation in decoding loss happens at low-layer and high-layers: watermarking these layers are less stable.

The same experiment is done with DenseNet-121. We limit report only to decoding loss in Figure 5.2 as model performance degraded negligibly, and BER stayed at zero. For DenseNet-121, the behavior is little different: changes are spread over all layers but come to stay after 18^{th} epoch. Results for EfficientNet-B0 are omitted due to the same results: BER stayed at 0, loss degradation is spread over all layers, and accuracy is not degraded.

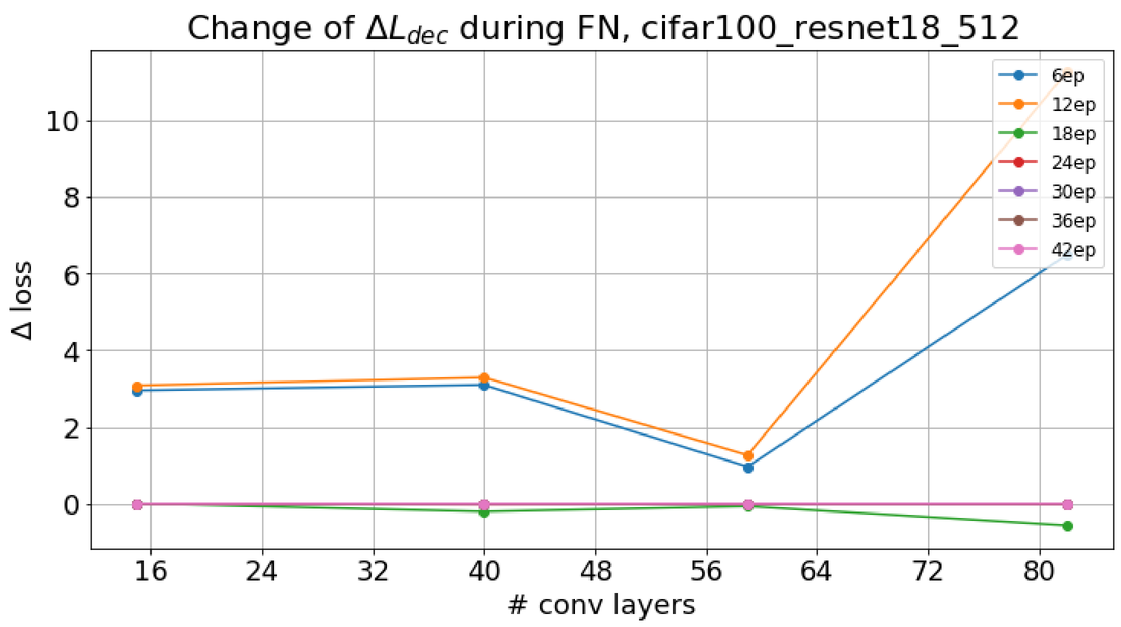


Figure 5.1: Fine-tuning. Decoding loss. ResNet-18, $m = 512$

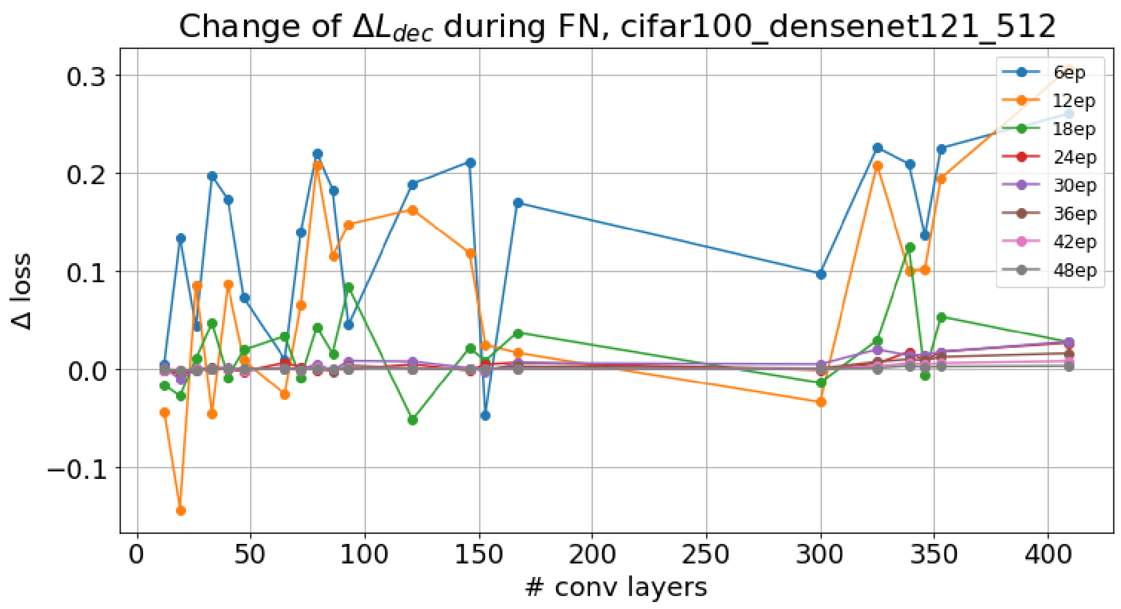


Figure 5.2: Fine-tuning. Decoding loss. DenseNet-121, $m = 512$

So, in this paragraph we demonstrated that watermarked models are robust to fine-tuning attacks. The same is valid for MNIST and CIFAR-10 due to simplicity of these tasks and higher over-parameterization property.

5.2 Pruning

The other widely-used attack is pruning. The main aim of pruning is to reduce the number of parameters in the network to make lighter and faster with the same performance. There are many types of DNN pruning, but we limit consideration to low-magnitude based approach as it is widely considered in the literature.

Low-magnitude pruning is performed in the following way: the threshold is set to be a small value, all weights below these values are removed (masked with zeroes), and the network is fine-tuned on a training set. The process continues until the total number of parameters reduces to the desired number.

For example, we performed pruning on ResNet-18 and reported compressed size in Table 5.1. Then, we embedded a watermark size of 512 and pruned the model up to 90%. While pruning the decoding loss, model accuracy, and BER are monitored. We also examine changes in decoding loss for different pruning rates.

α (pruning rate)	0	0.1	0.2	0.3	0.4
Compressed size	39.84M	37.41M	34.31M	31.22M	27.85M
α (pruning rate)	0.5	0.6	0.7	0.8	0.9
Compressed size	24.31M	20.18M	16.57M	11.12M	8.23M

Table 5.1: ResNet-18 pruning. Compressed size is linear to pruning rate.

Figure 5.3 illustrates the accuracy degradation for various pruning rates. Chart reports that model performs well till pruning rate equal to 0.5, the performance is substantially deteriorated.

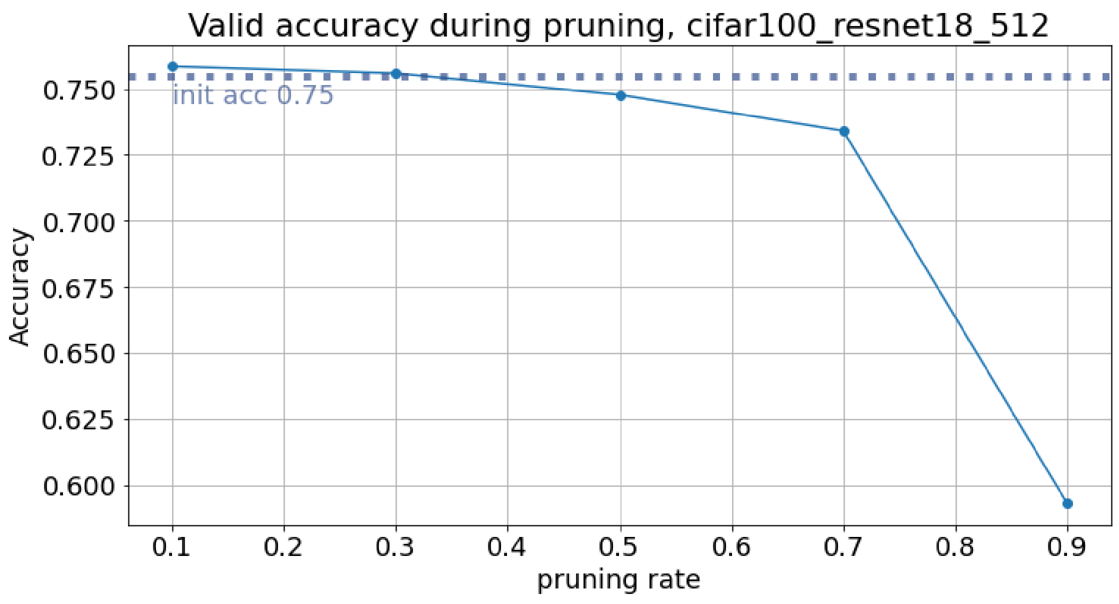


Figure 5.3: Model pruning. Model accuracy. ResNet-18, m = 512

Regardless of accuracy drop, Figure 5.4 reports that BER after extraction from 3 out of 4 layers remains 0, while for one layer BER slightly increases (within 0.1). However, decoding loss 5.5 reports changes in all layers during the process.

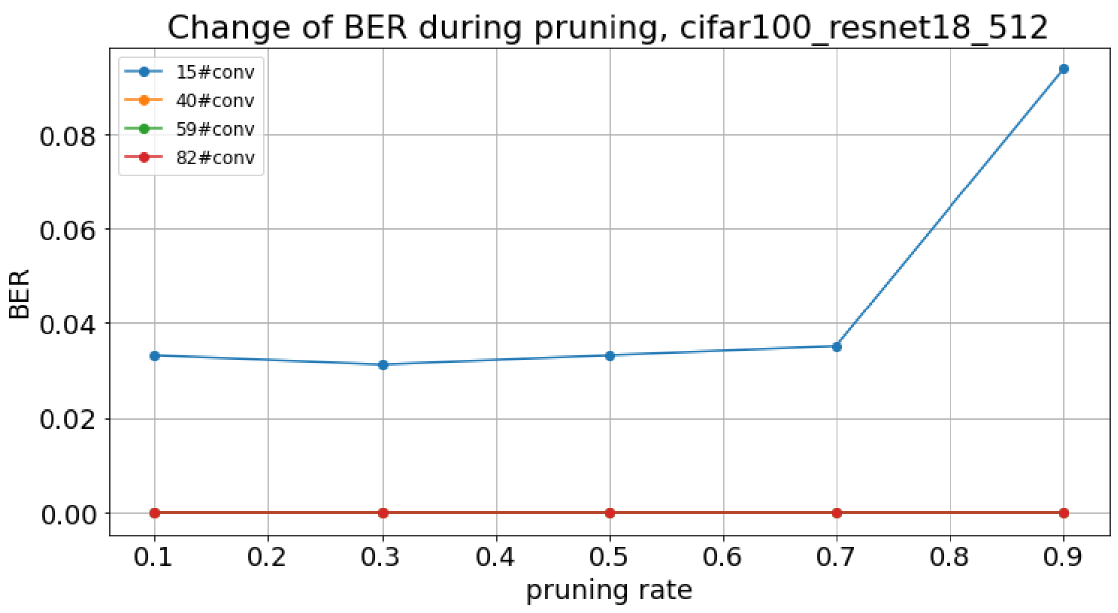


Figure 5.4: Model pruning. BER. ResNet-18, $m = 512$

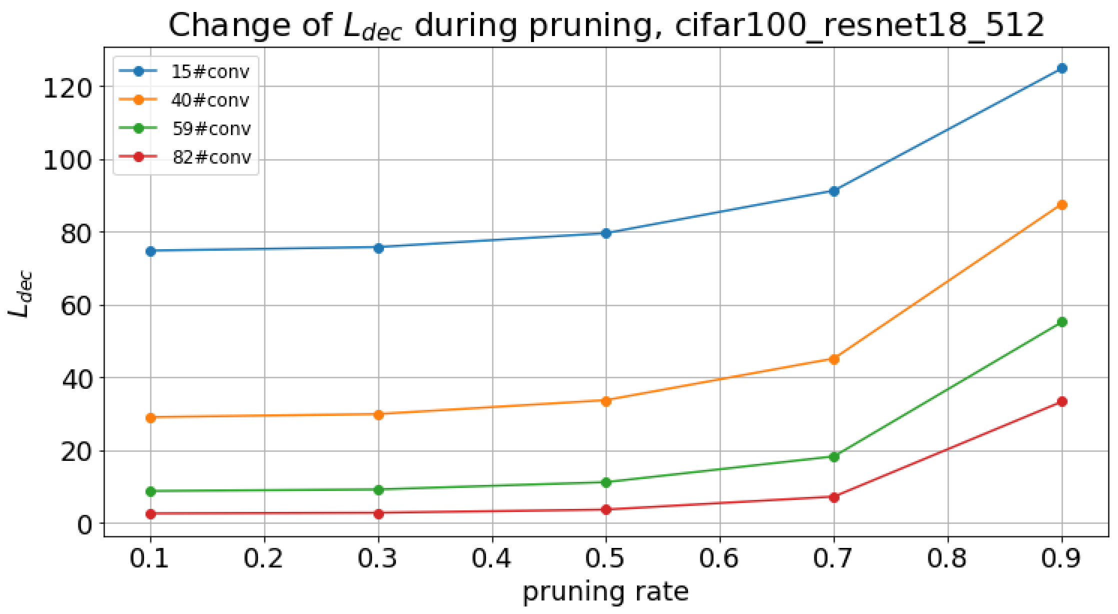


Figure 5.5: Model pruning. Decoding loss. ResNet-18, $m = 512$

Pruning was also conducted for DenseNet-121 and EfficientNet-B0, but results are omitted due to similar results. In other words, all watermarked resists pruning attack up to 90% even though model performance drops dramatically. To sum up, a pruning attack is not capable of removing watermark embedded with no additional training.

5.3 Over-writing

An over-writing attack was extensively reviewed in previous chapters. It was showed that weights with forged watermark go farther from initial distribution leading to lower accuracy. So, once the weights were modified, any additional changes only decrease the final model performance. In this sense, performing an attack without re-training seems to be more beneficial.

To conclude, it is worth mentioning that the proposed algorithm resists the main attacks, thus satisfies the security criterion for effective watermarking.

Chapter 6

Discussions

6.1 Conclusion & Future work

In this work, we considered the problem of DNN watermarking and reviewed all existing approaches. We identified that all modern neural networks are watermarked with re-training or fine-tuning, which is undesirable for cases when models are trained for a long time, or there is no access to the training set. We proposed a new algorithm for watermark embedding. This algorithm does not require model re-training and access to the training set. Moreover, it meets all criteria for effective watermark embedding. For successful embedding, we utilized over-parametrization property of deep neural networks, which is a more pronounced case for extremely large models. All experiments were performed on state-of-the-art convolutional neural networks such as ResNet, DenseNet, and EfficientNet families trained on MNIST, CIFAR-10, CIFAR-100, and ImageNet datasets. To demonstrate robustness, we performed a fine-tuning attack on a leaked training set and model pruning and successfully extracted watermarks with BER less than 0.1. Moreover, we showed that this algorithm is resistant to over-writing attack: extra over-writing does not entirely remove the watermark and degrades model performance, making the attack unfeasible. Finally, the main advantage of the proposed solution that it requires significantly less amount of time. This solution would benefit companies owing already pre-trained models and wishing to prevent illegal usage.

As the algorithm requires access to the model's weights, it is not currently applicable to black-box scenarios when a model deployed on remote service with open API. Thus, as future work, a watermarking scheme with black-box verification with no extra training should be considered. Moreover, in this work, we considered only a classification problem, while other problems like segmentation, recognition remained uncovered, so this also should be addressed in future versions.

Appendix A

Figures

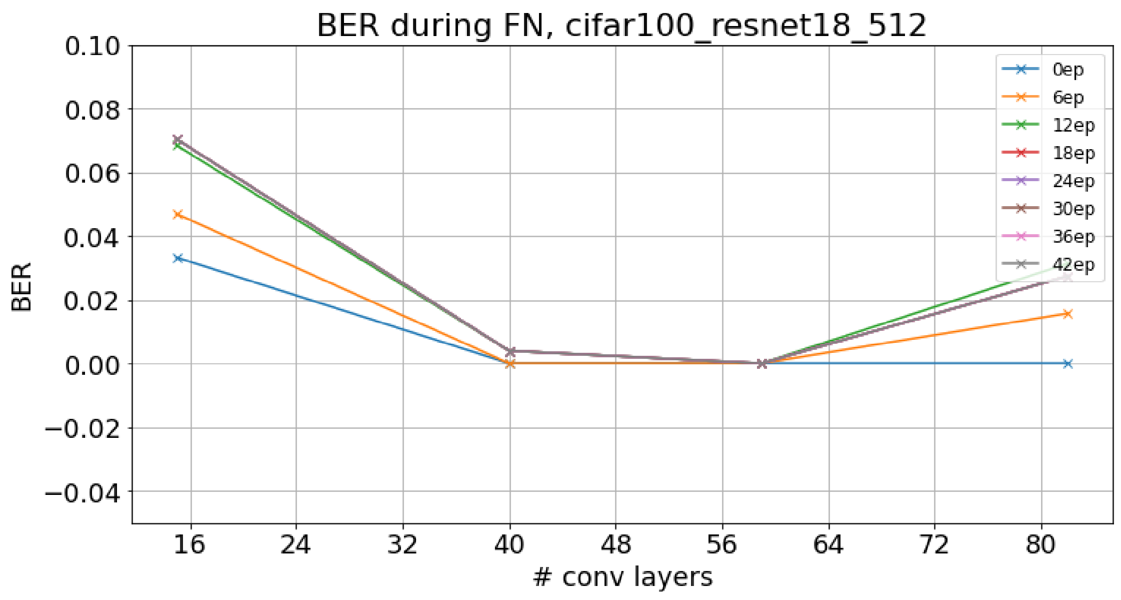


Figure A.1: Fine-tuning. BER. ResNet-18, $m = 512$

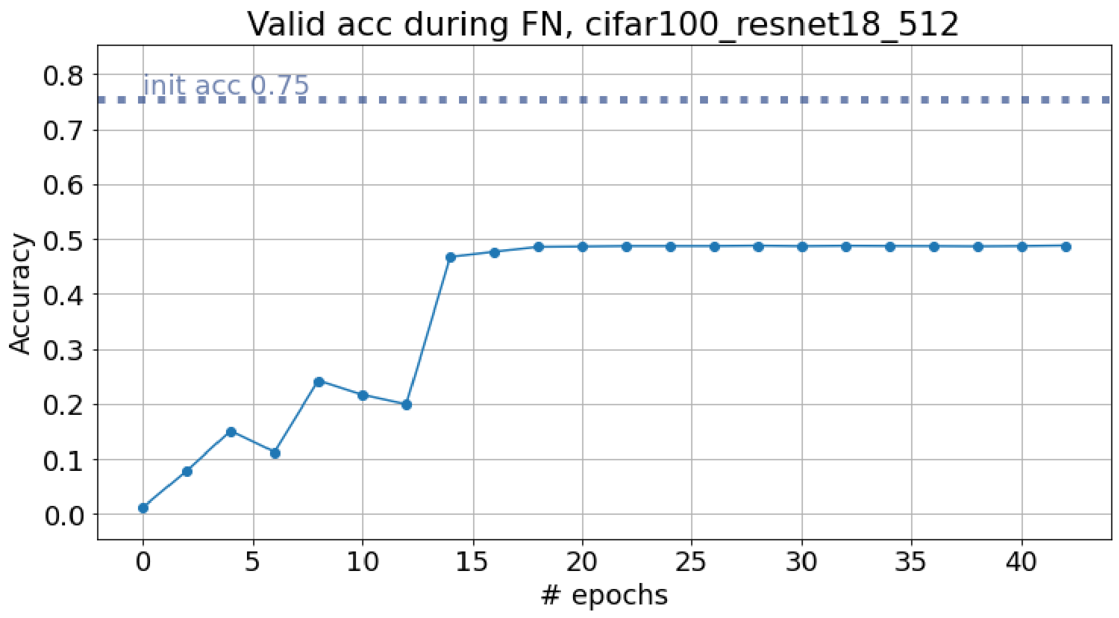


Figure A.2: Fine-tuning. Model accuracy. ResNet-18, m = 512

Bibliography

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[4] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[5] Jessica Fridrich and Tomas Filler. Practical methods for minimizing embedding impact in steganography. *Proc SPIE*, 6505, 02 2007.

[6] Tomáš Filler, Jan Judas, and Jessica Fridrich. Minimizing embedding impact in steganography using trellis-coded quantization. In Nasir D. Memon, Jana Dittmann, Adnan M. Alattar, and Edward J. Delp III, editors, *Media Forensics and Security II*, volume 7541, pages 38 – 51. International Society for Optics and Photonics, SPIE, 2010.

[7] I. J. Cox, J. Kilian, F. T. Leighton, and T. Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing*, 6(12):1673–1687, 1997.

[8] Mauro Barni, Franco Bartolini, Vito Cappellini, and Alessandro Piva. A dct-domain system for robust image watermarking. *Signal Process.*, 66(3):357–372, May 1998.

[9] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 587–601, New York, NY, USA, 2017. Association for Computing Machinery.

[10] 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*, ICMR '17, page 269–277, New York, NY, USA, 2017. Association for Computing Machinery.

[11] Yuki Nagai, Yusuke Uchida, Shigeyuki Sakazawa, and Shin'ichi Satoh. Digital watermarking for deep neural networks. *International Journal of Multimedia Information Retrieval*, 7, 02 2018.

[12] 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*, ICMR '19, page 105–113, New York, NY, USA, 2019. Association for Computing Machinery.

[13] Min Wu, Wade Trappe, Z Jane Wang, and KJ Ray Liu. Collusion-resistant multimedia fingerprinting: a unified framework. In *Security, Steganography, and Watermarking of Multimedia Contents VI*, volume 5306, pages 748–759. International Society for Optics and Photonics, 2004.

[14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 12 2013.

[15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2015.

[16] Erwan Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 11 2017.

[17] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. 12 2017.

[18] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 267–284, Santa Clara, CA, August 2019. USENIX Association.

[19] 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, ASIACCS '18, page 159–172, New York, NY, USA, 2018. Association for Computing Machinery.

[20] 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 (USENIX Security 18)*, pages 1615–1631, Baltimore, MD, 2018. USENIX Association.

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

[22] Huiying Li, Emily Willson, Haitao Zheng, and Ben Y Zhao. Persistent and unforgeable watermarks for deep neural networks. *arXiv preprint arXiv:1910.01226*, 2019.

[23] 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*, ACSAC '19, page 126–137, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 485–497, New York, NY, USA, 2019. Association for Computing Machinery.

[25] Ziqi Yang, Hung Dang, and Ee-Chien Chang. Effectiveness of distillation attack and countermeasure on neural network watermarking. 06 2019.

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