Delving in the loss landscape to embed robust watermarks into neural networks

Delving in the loss landscape to embed robust watermarks into neural networks

Enzo Tartaglione, Marco Grangetto, Davide Cavagnino and Marco Botta
Università degli Studi di Torino
Torino, Italy
Email: enzo.tartaglione@unito.it

***Abstract*—In the last decade the use of artificial neural networks (ANNs) in many fields like image processing or speech recognition has become a common practice because of their effectiveness to solve complex tasks. However, in such a rush, very little attention has been paid to security aspects. In this work we explore the possibility to embed a watermark into the ANN parameters. We exploit model redundancy and adaptation capacity to lock a subset of its parameters to carry the watermark sequence. The watermark can be extracted in a simple way to claim copyright on models but can be very easily attacked with model fine-tuning. To tackle this culprit we devise a novel watermark aware training strategy. We aim at delving into the loss landscape to find an optimal configuration of the parameters such that we are robust to fine-tuning attacks towards the watermarked parameters. Our experimental results on classical ANN models trained on well-known MNIST and CIFAR-10 datasets show that the proposed approach makes the embedded watermark robust to fine-tuning and compression attacks.**

I. Introduction

Besides the tremendous amount of research and applications catalyzed by the great success of Artificial Neural Networks (ANNs), and especially Deep Learning (DL), little attention has been devoted to the related security and model integrity aspects. Nowadays, ANNs have become the standard tool to tackle supervised learning problems, provided that one can train a deep neural model using a large annotated dataset. DL has gained momentum and performance in an increasing number of fields ranging from computer vision, image, audio and natural language processing [1], [2] to health applications [3], just to name a few.

The ubiquitous presence of DL in the industry, health and other critical systems, is asking for more research efforts to understand and analyze security aspects and vulnerabilities. As an example in [4] the vulnerability of DL to adversarial modifications of the input examples is analyzed. Another aspect that must be guaranteed in safety critical systems using DL is that the designed model is not modified or compromised. In other contexts copyright protection of models must be ensured [5], [6]: in this work we explore the watermarking of ANNs to protect ownership of trained models.

Object watermarking is a widespread technique used to embed a signal (like a watermark image) into an object (as a banknote) [7], [8]. Digital watermarking is the digital application of object watermarking: it is performed by using a function to alter some features of a digital object (like pixel values or transform coefficients for images, samples for one-dimensional signals, weights for neural networks, coordinates for 3D models) driven by a binary valued or real valued signal, i.e. the watermark. The watermarking process is composed of two phases, namely embedding and extraction/verification; the first one stores a watermark string into the object features, possibly using a secret key to protect the watermark to counter attacks, while the second phase is performed every time it is necessary to test the presence of the watermark in the object. Exploiting the watermark one can enforce copyright and ownership protection, track of origin, integrity protection and authentication.

As we briefly recall in the next section, there are only few preliminary studies on digital watermarking applied to ANN. In this work we advance the state of the art by making the following contributions:

- we start from a *naive* and simple solution that embeds the watermark in the ANN model weights; to this end we simply exploit the fact that ANN models are usually highly redundant and training can done by fixing a subset of the model weights, e.g. the watermark, without impacting on the final ANN performance, e.g. image classification;
- we improve the naive approach by adding a term in the loss function used for training: the aim is to find a minimum of the loss function that guarantees the watermarked weights to lay into a very steep valley;
- the proposed technique successfully embeds watermark in *all* the layers of the deep model, including the output layer, and it is not limited to a subset of hidden layers like [9];
- we show that the proposed watermarking-aware loss function makes it robust to fine-tuning attack where an adversary tries to change the model weights by running additional training epochs;
- we empirically study the effectiveness of the proposed approach by showing that it represents an efficient solution achieving a significant level of robustness to attacks without impacting on the ANN performance.

The paper is structured as follows: in Sect. II the related state of the art is recalled, in Sect. III the proposed watermarking method is presented in detail. In Sect. IV we experimentally analyze the effectiveness and robustness to fine-tuning and compression attacks; finally, in Sect. V our conclusions are drawn and future directions outlined.

II. Related work

There is a growing interest in the protection of neural network models by means of watermarking. There are essentially two different approaches to ANN watermarking: *open model* and *black box* watermarking, respectively.

In the former case, the watermark is embedded in the model parameters that therefore must be accessible to recover the watermark. The first of this kind of methods has been proposed in [5], [10], where a vector is embedded into the model parameters: in particular, the authors showed that it is possible to embed a binary watermark into the parameters of a single convolutional layer of an ANN. To this end, they recast the watermark extraction as a binary classification problem with a single-layer perceptron and propose to add a regularization term in the loss function capable to jointly guarantee model performance and watermark embedding during training.

Another open model approach has been proposed in [9] with a different goal: orthogonal fingerprinting of the parameters of a convolutional layer for user identification. To this end, a mean square error regularization term is added to the loss function during training to assure that the weights of the convolutional layer can be projected onto the orthogonal fingerprint. The fingerprint can be considered as a particular case of binary watermark.

In black box approaches the watermark is learned by means of a modified training dataset and the model ownership can be queried during the inference phase, based on the generated output. The works [6], [11], [12] propose different strategies to modify the training images for black box watermarking and show that in some cases the model accuracy may be impaired significantly. In [13] this class of approaches is analyzed in terms of payload size and model accuracy.

Previous works have shown the feasibility of ANN watermarking and also pointed out that the most typical kind of attack is represented by fine-tuning (or transfer learning); these latter methods consist in continuing training of existing models (trained on large datasets) on smaller or slightly modified input to get improved performance. The same approach can be used to let a model drift from the watermarked solution, thus preventing the detection of the watermark while still performing well on the trained task. In this work we show that it is possible to take fine-tuning attack into account during training so as to exploit ANN redundancy and adaptation capacity to get a watermarked model that is intrinsically more robust to such attacks. Towards this end, we are going to use a *replica*-based approach, consisting in duplicating the configuration of our trained ANN model. This is not a new concept in deep learning: Zhang *et al.* [14], for example, used such a concept to boost the convergence time during training, or it can be even used to find non-trivial solutions to learning problems [15].

III. Method

In this section we are going to describe the training strategy aiming to embed a given watermark. As opposed to the previous works, the approach proposed in this paper can use any random subset of model parameters as a real-valued (with proper floating point representation) watermark signal. This choice provides two major advantages: larger watermark capacity and more robustness to model ablation thanks to random spreading of the watermark on all its parameters. The key idea is to embed the watermark into a subset of the model parameters selected according to a private key: the weights carrying the watermark are simply not updated during training and permits trivial extraction and verification. We will show that the watermarked ANN can be trained to state-of-the-art performance while reserving enough embedding capacity thanks to high dimensionality of the model. More importantly, we propose a solution to make such a simple watermarking strategy robust towards the most typical model attack [5], [13], i.e. fine-tuning. To this end, we provide a method that guarantees the watermarked parameters can not be changed significantly by gradient updates when an attacker tries to modify the model weights by fine-tuning.

A. Preliminaries

In this section we are going to introduce the notation to be used in the rest of this work. Let us assume we aim at training an artificial neural network model Γ_0. The j-th trained parameter inside the model will be indicated as w_j^0. The watermark is represented by the sequence $X \in \mathbb{R}^N$, where x_i is a single element. These will be mapped to some parameters of Γ_0, which will be grouped in the collection of watermarked parameters $W_x \subset \Gamma_0$. All the parameters $w_j^0 \in W_x$ will not be modified during the training phase; hence, just the subset of parameters

$$\overline{W}_x = \Gamma_0 \setminus W_x \tag{1}$$

will be updated during the training phase.
In our work we will apply some transformation to the original model Γ_0: all those models will be referred to as Γ_k for $k \in [1, R]$ and the j-th parameter belonging to the k-th model will be indicated as w_j^k.
As we are going to see, this apparent restriction will not be an obstacle during the training phase, as state-of-the-art performances will be achieved in any case. It has been shown that typical learning scenarios involve the research of wide minima, whose existence is related to the typical over-parametrization of ANN models [16], [17]. It is not a case, in fact, that deep models can be hugely compressed, as most of the parameters are not necessary after the training phase [18], [19], [20]. Exploiting this, we are going to look for a particular solution of Γ_0 resulting in a *narrow* minima in the subspace W_x.

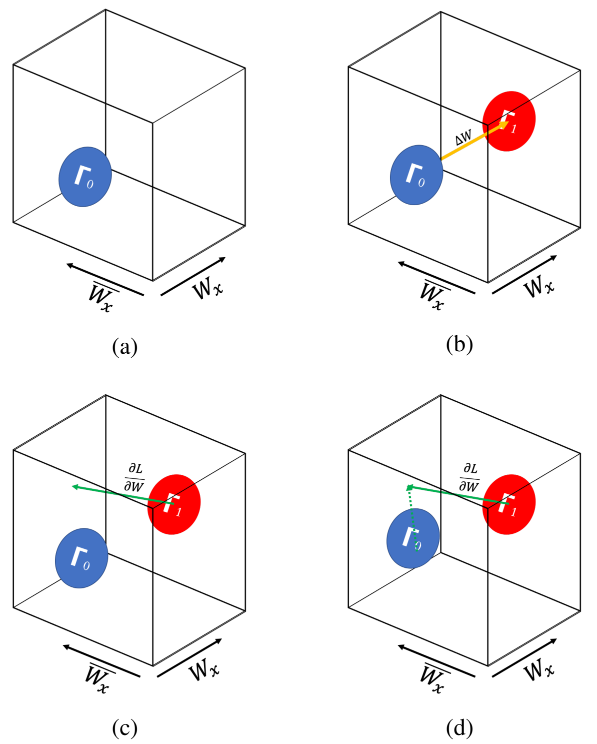


Fig. 1: Overview on the method. From the model which is trained updating non-watermarked parameters (a) we add some noise on the watermarked parameters only in R replica of our original network Γ_0 – in this case $R = 1$ (b). Then, the gradient on Γ_1 is computed (c) and projected to the non-watermarked parameters space (d).

B. Choice of the parameters

In this section we are going to describe the strategy we decided to use in order to select some parameters form the ANN model in which we decide to embed the watermark. These parameters will not be updated anymore during the training phase, preserving their own value. What we aim here is to find a good metric to choose these parameters in a way they are indistinguishable from other parameters.
Towards this end, let us assume all the parameters of our ANN network are initialized with a gaussian distribution

$$\Gamma_0 \sim \mathcal{N}(\mu, \sigma) \tag{2}$$

where μ is the mean and σ is the standard deviation. Now, let us assume that each element of the watermark $x_i \in X$ is bound and, in particular, $x_i \in [0, 1]$. In order to embed these values in the ANN model, we first have to decide the subset W_x of parameters to be watermarked, and then the association of the i-th element of the watermark x_i to the j-th parameter of the neural network w_j^0. Towards this end, we randomly choose these associations according to some pseudorandom number generator $f(\cdot)$

$$M_x = f(\Gamma_0, N, K), \tag{3}$$

where M_x is a map associating $i \leftrightarrow j$ and K is an initialization seed, kept private. By design, we ensure to uniformly watermark all the layers of Γ_0: hence, we cascade two different uniform samples for $f(\cdot)$, where first we sample the layer and then the j-parameter in the layer to be associated to the i-th element in the watermark. This operation is repeated $\forall\, i$. From M_x we obtain the subset of parameters W_x to be watermarked. Once we have determined W_x, we choose to map the single elements as

$$w_j^0 = 2\sigma(x_i - 0.5) + \mu \quad \forall\ w_j^0 \in W_x \tag{4}$$

C. Training strategy

Once we have embedded the watermark in some parameters w_j^0 of model Γ_0, we need to train by minimizing a proper loss function L that depends on the problem the model is targeting, e.g. cross entropy for classical classification model. In order to do that, we can use a standard SGD strategy. However, we need to ensure all the watermarked parameters not to change during the process. Hence, the update rule is modified as

$$w_j^0 := w_j^0 - \left[1 - \mathbf{1}_{W_x}(w_j^0)\right] \eta \frac{\partial L}{\partial w_j^0} \tag{5}$$

where η is the learning rate and $\mathbf{1}_{W_x}(w_j^0)$ is the indicator function for w_j^0 being a watermarked weight:

$$\mathbf{1}_{W_x}(w_j^0) = \begin{cases} 1 & if\ w_j^0 \in W_x \\ 0 & otherwise \end{cases} \tag{6}$$

Hence, we are updating the network using the subspace $\overline{W}_x$ of the non-watermarked parameters (Fig. 1a). The watermark extraction will be based on simple inverse mapping of (4):

$$x_i = \frac{w_j^0 - \mu}{2\sigma} + 0.5 \tag{7}$$

Ensuring (6) is a necessary condition to keep the watermark in the model during the training; however, we will not guarantee it can be removed using attack strategies like fine-tuning. What we are aiming here to ensure it is a *robust* watermark as well. To this end, we derive R ANN perturbed models Γ_k whose values are initialized according to

$$w_j^k := w_j^0 + \left[\mathbf{1}_{W_x}(w_j^0)\right] \Delta w_j^k \tag{8}$$

where Δw_j^k is some noise sampled from a normal distribution $\mathcal{N}(0, s)$. Essentially, we are here introducing a variation just to the watermarked parameters in the subspace W_x, yet leaving all the other parameters maintaining the value achieved during training (Fig. 1b). Then, let us calculate the gradient of the loss with respect every w_j^k, averaged on all perturbed Γ_k models:

$$g_j = \frac{1}{R} \sum_{k=1}^{R} \frac{\partial L}{\partial w_j^k} \tag{9}$$

Essentially, g_j is the gradient on the j-th model parameter computed for a small perturbation of the watermark (Fig. 1c). In order to make the watermark robust to fine-tuning attacks, we need the final configuration of Γ_0 to be in a *narrow minima* along the dimensions of the watermarked weights, i.e. g_j must be high, $\forall i \in \overline{W}_x$. To this end, while minimizing the loss L on Γ_0 to train the model, we also want the gradient on

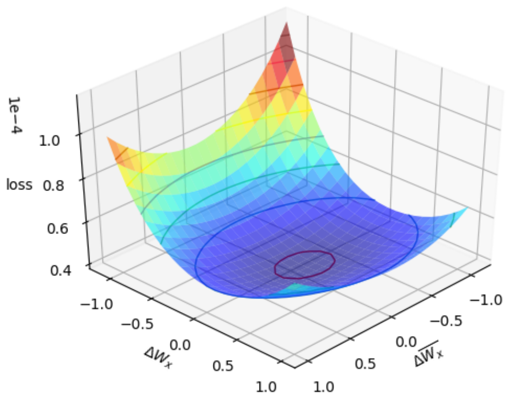


Fig. 2: Loss for small variations in W_x and $\overline{W}_x$. While in $\overline{W}_x$ we reach a wide minimum, in W_x we find a narrow minimum.

watermarked parameters to be as high as possible for small perturbations of it (or, in other words, for all the Γ_k with $k \geq 1$): hence, we want to maximize (9). So, the overall update rule becomes:

$$w_j^0 := w_j^0 - \left[1 - \mathbf{1}_{W_x}(w_j^0)\right]\left[\eta \frac{\partial L}{\partial w_j^0} - \gamma g_j\right] \tag{10}$$

where γ is a positive hyper-parameter. The effect is that the projection of g_j to the subspace $\overline{W}_x$ will determine a contribution to *maximize* the value of the loss function for small variations of the watermarked parameters in W_x (Fig. 1d). Therefore, when an attacker applies fine-tuning the gradient-based update rule will penalize changes to the watermarked weights since the loss L along such dimensions exhibits steep variation. At the same time, it is worth pointing out that the loss steepness cannot be exploited by the attacker to recover the subspace W_x, given the high dimensions of typical ANNs. In Fig. 2 we anticipate an experimental result supporting the previous remark: the surface represents the experimental loss measured along a watermarked parameter dimension (ΔW_x) and free parameter ($\Delta\overline{W}_x$) on an ANN model trained according to (10). It can be noted that, as imposed by the rule we propose, the loss surface is clearly steeper along the watermarked parameter dimension.

IV. Experimental results

In this section we experiment with our proposed watermarking method over some different neural architectures and datasets. Before training the ANN, we embed the watermark into the model parameters W_x that are randomly chosen according to a private key. The proposed update rule (10) is used to train the model by setting different values for R: when setting $R = 0$ we get naive embedding of the watermark on model weights, whereas for $R > 0$ we enforce resilience to fine-tuning attack. Our training and inference algorithms are implemented in Python, using PyTorch 1.3, and an RTX2080 Ti NVIDIA GPU with 11GB of memory has been used for training and inference.[1] All the used hyper-parameters have been tuned using a grid-search algorithm.

TABLE I: Performances after training.

Dataset	Architecture	epochs	R	test error [%]
MNIST	LeNet5-caffe	100	0	0.78
			1	0.83
			4	0.82
			16	0.85
CIFAR-10	ALL-CNN-C	350	0	10.38
			2	10.67
			4	10.50
	ResNet-32	350	0	7.14
			2	7.08
			4	7.29

The results will be presented for two different datasets and two architectures for classification task: LeNet5-caffe trained on MNIST, ALL-CNN-C and ResNet-32 trained on CIFAR-10. All the architectures are initialized according to (2). The watermarked parameters W_x will be in the standard IEEE 754 `float32` format each. Hence, when we say the size of the watermark embedded into a model Γ_0 is N, the actual bits of the watermark are $32N$. In our experiments, we have chosen, as watermark sequence X, the RGB pixel values of some images, scaled according to (4). The watermarked ANN architectures will be attacked using two possible strategies:

- *fine-tuning*: the model is fine-tuned on the training set. The attacker attempts to change the configuration of the parameters in the model, still keeping the error low, hoping to remove the watermark in the process.
- *compression*: the bits used for the representation of each parameter are reduced. In particular, all the floating point parameters are first converted into fixed point representation, and then some bits, from the LSB, are masked. The attacker attempts, in this case, to reduce the numerical precision of the parameters, hoping that the watermark will be removed while the performance won't drop significantly. Considering our choice to embed, as watermark sequence, RGB pixel values of some images, we expect the watermark won't be modified when encoding every parameter of the model on 8 bits or more.

For both the attacks, the Pearson correlation between the original and extracted watermark will be the metric used to measure the robustness to the attack, compared with the test error of the architecture.

A. LeNet5-caffe on MNIST

In our first experiment we embed as watermark RGB images with $N = 20 \times 30 \times 3 = 1800$ parameters into the LeNet5-caffe architecture, trained on MNIST. MNIST is a handwritten digits dataset made of 60k 28×28 pixel grey-scale images of digits, divided in 10 classes (from 0 to 9).

[1] The source code will be made available upon acceptance of the article.

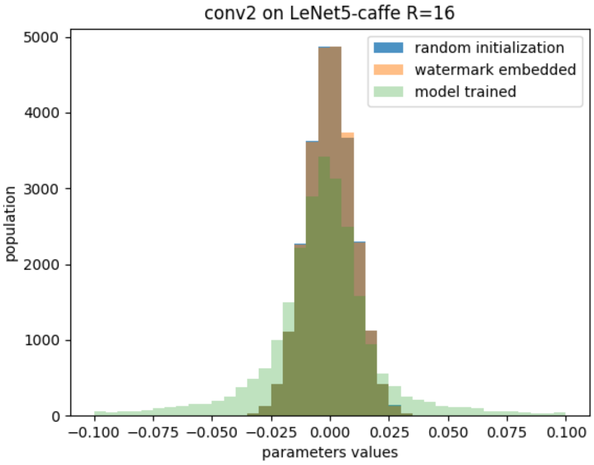


Fig. 3: Distribution of the parameters in the second convolutional layer (conv2) of LeNet5-caffe.

For our simulations, the chosen loss function to minimize is the cross-entropy with vanilla-SGD, $\eta = 0.1$, weight-decay 10^{-4}, $\mu = 0$, $\sigma = 0.01$, $s = 0.01$ for all the R values, mini-batch size 100 for 100 epochs. The value of γ is 10^{-4} and is multiplied by 0.5 every 25 epochs. In Tab. I we show the obtained classification error on test-set after training: it can be noted that 5dc03cc329200c00014e074cwatermark embedding is achieved without significant impairment in classification accuracy for all values of R. These results show that, as expected, a subset of the model parameters (0.4% in our experiment) can be used to convey the watermark with no penalty. The achieved embedding capacity obviously depends on model, task complexity and corresponding ANN redundancy.

As a further analysis in Fig. 3 we show a typical distribution for the parameters in conv2 layer:

1) when ANN parameters are initialized with normal distribution ($\mu = 0$ and $\sigma = 0.01$),
2) after embedding the watermark before training,
3) at the end of training process.

It can be noted that the watermarked parameters (about 0.4% of the total) cannot be easily detected from the observed empiric distribution.

Then we studied the robustness to fine-tuning varying R. The Pearson correlation coefficient between the original watermarked parameters and those extracted after fine-tuning attack is shown in Fig. 4a, as a function of the epochs of fine-tuning. We can observe that if we are not using our strategy (case $R = 0$) the watermark tends to vanish. According to our design, increasing R the watermark persists for a longer time: the larger R, the more we are confident we put Γ_0 in a narrow valley for W_x. According to this, we have tried also a compression attack to the $R = 16$ model. Finally, we analysed the effects of compression attack: Fig. 4b shows how the test error varies when the number of bits used for representing a single parameter of the model decreases and how the Pearson correlation coefficient varies. It can be noted that the watermark is quite robust to quantization as it tends to vanish when the parameters are represented with less than 4 bits, exactly when also the test error rises. Please notice that the Pearson correlation coefficient for more than 6 bits is not exactly 1 but very close to it.

B. ALL-CNN-C and ResNet-32 on CIFAR-10

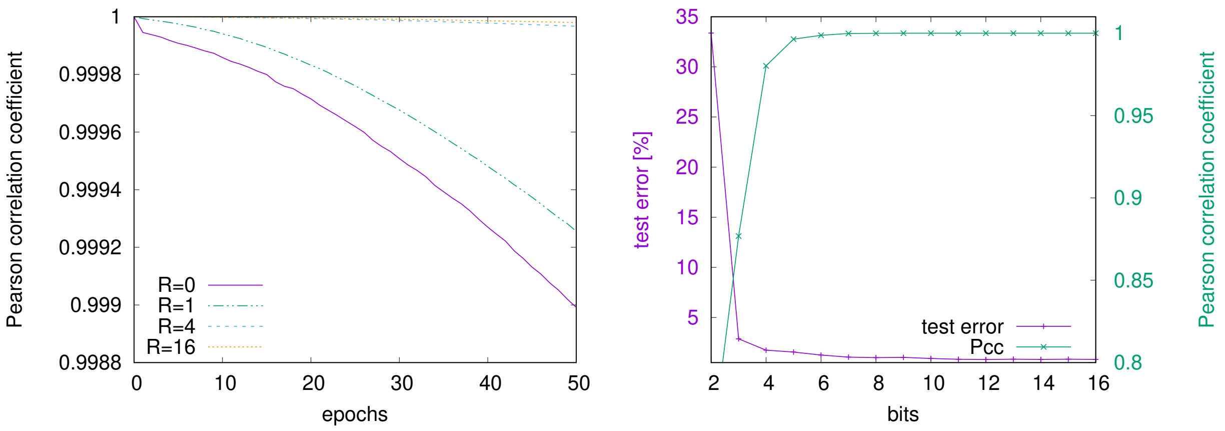
Our watermarking strategy has been tested also for another, more complex convolutional neural networks, ALL-CNN-C [21] and ResNet-32 [22], trained on a different classification task: CIFAR-10. It is made of 32×32 color images (3 channels) divided into 10 classes. The training set is made of 50k images and the test set of 10k samples. For our simulations on both the architectures, the chosen loss function to minimize is the cross-entropy with vanilla-SGD, $\eta = 0.1$ multiplied by a factor 0.1 after $[150, 250]$ epochs, weight-decay $5 \cdot 10^{-4}$, $\mu = 0$, $\sigma = 0.01$, $s = 0.01$ for all the R values, mini-batch size 100 for 350 epochs. Here the value of γ is 10^{-4}. The length of the watermark is here $N = 38 \times 29 \times 3$.

According to the overall performance on the test set (see Tab. I), changing R does not significantly impact on the classification error. Also in this case, when we look at the Pearson correlations during a fine-tuning attack (in Fig. 5a), we clearly see the benefit of having a higher R value. It can be noted that just setting $R = 4$ appears to be sufficient to keep the correlation value very close to 1. Similarly, the compression attack (Fig. 5b) shows that encoding the parameters in less than 7 bits the performance drops, exactly when a part of the watermarked parameters are modified.

A different and more recent architecture, ResNet-32, has also been trained and watermarked. Similarly to ALL-CNN-C, according to Tab. I, the test error does not change significantly with different R values. If we inspect the watermark robustness to fine-tuning attack, however, we observe a larger robustness for $R = 4$ (Fig. 6a). Also here we observe a significant robustness to the compression attack: Fig. 6b shows that the performance worsens already when we encode each parameter on 12 bits, while the Pearson correlation coefficient remains close to 1 until 8 bits quantization.

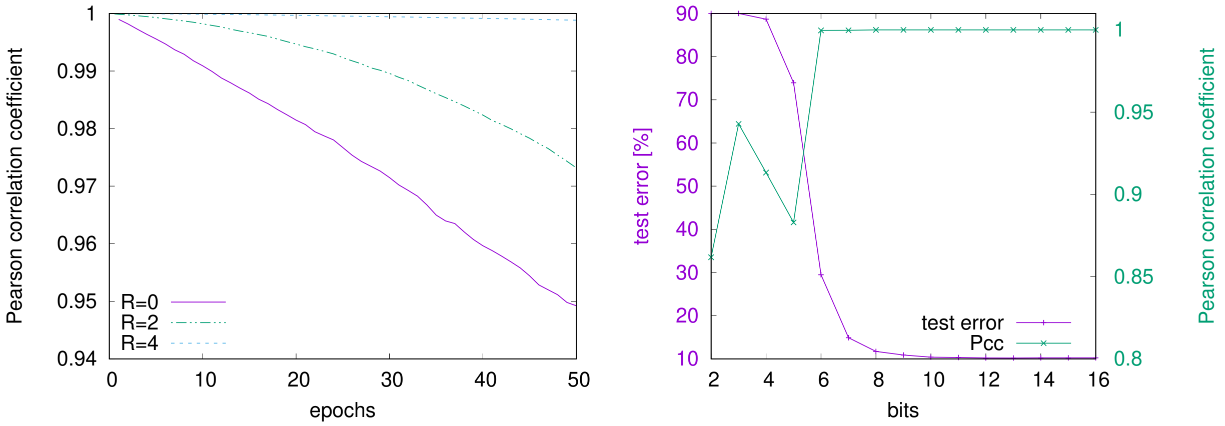
C. Comparison with related work

As mentioned in Sect. II the most closely related work to ours is [5] and the following extension in [10]. These works add to the training process a single-layer perceptron whose goal is to extract a sequence of bits, the watermark, from the weights of a certain convolutional layer. The major practical limitation compared to the solution proposed in this work is represented by watermark capacity, that in [10] is limited to a number of bits lower or equal to the number of parameters of a convolutional layer. As a consequence the experiments presented in [10] are limited to watermark sequences of 2,048 bits. In our case the watermark is directly represented by a set of weights in floating point representation: as an example in our experiments we can embed as watermark a color logo image amounting at $N = 38 \times 29 \times 3 = 3306$



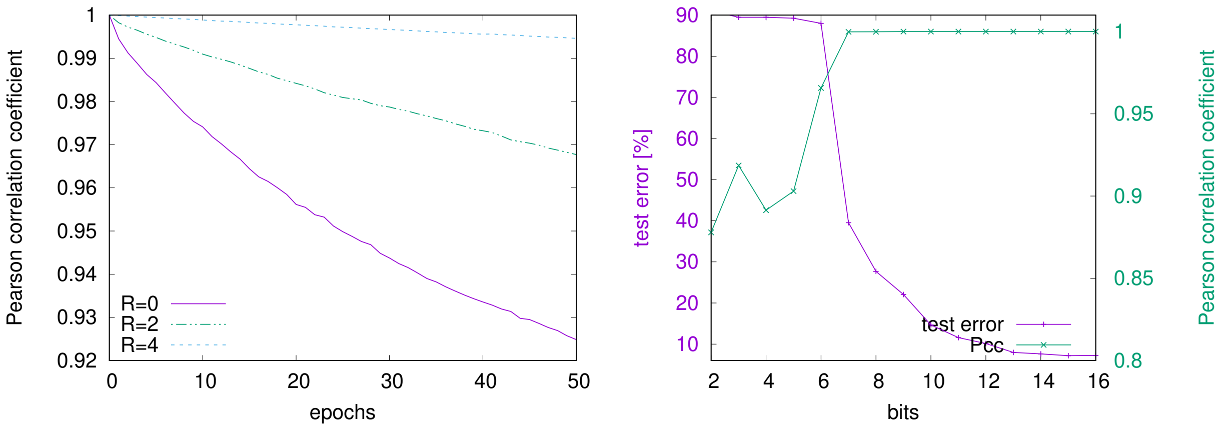
(a) (b)

Fig. 4: Fine-tuning attack on LeNet5-caffe trained with different R values (a) and compression attack on LeNet5-caffe trained on MNIST for $R = 16$ (b).



(a) (b)

Fig. 5: Fine-tuning attack on ALL-CNN-C trained with different R values (a) and compression attack on ALL-CNN-C trained on CIFAR-10 for $R = 4$ (b).



(a) (b)

Fig. 6: Fine-tuning attack on ResNet-32 trained with different R values (a) and compression attack on ResNet-32 trained on CIFAR-10 for $R = 4$ (b).

bytes. In other words our experiments show that our technique can embed a watermark that is about 8 times larger without neither impairing ANN performance, nor loosing robustness to fine-tuning attack. Another work that is worth mentioning for comparison is [9]: in this case the watermark embedding approach is somehow closer to ours, being based on a training regularization term. Nonetheless, it is designed for the particular case of embedding of short orthogonal fingerprints in convolutional layer weights; as a consequence the watermark is binary and its capacity is limited by the number of weights in a convolutional layer.

Besides the higher capacity, our proposed solution is much more flexible as any random subset of weights can be used to host the watermark signal. In this work we exploit this feature by randomly selecting the weights to be marked, thus maximally spreading the watermark across all ANN model parameters, guaranteeing that it is difficult to devise ad-hoc attacks aimed at removing them as opposed to previous works where the watermark resides in a single layer.

V. Conclusion

In this paper a new learning strategy has been proposed, aimed at ensuring that some watermarked parameters inside an ANN model are robust to fine-tuning attacks. Towards this end, during the learning process, which results into minimizing the loss function by modifying non-watermarked parameters with gradient descent-based approaches, we are including a constraint which maximizes the gradient for small perturbations of the watermarked parameters. This learning process selects configurations for the trained ANN model which have both low error and are not allowing, because of the local shape of the loss landscape, to significantly modify the watermarked parameters without hugely impacting on the performance of the ANN model.

Empirically, the proposed technique proves to be effective in ensuring the watermark not being removed using fine-tuning attacks. Furthermore, it proves its robustness also under compression attacks: the performance worsens when just a part of the watermarked parameters are modified. Detecting which are the watermarked parameters cannot be exploited by simple inspection of the weights distribution, as these are buried and mixed besides all the other parameters. Future works include the formulation of a regularization term having the same behavior as shown in this work, without involving the generation of replicas, exploiting the concept of sensitivity [18].

References

[1] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, Nov 2019.

[2] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE Access*, vol. 7, pp. 19 143–19 165, 2019.

[3] L. Zhang, J. Lin, B. Liu, Z. Zhang, X. Yan, and M. Wei, "A review on deep learning applications in prognostics and health management," *IEEE Access*, vol. 7, pp. 162 415–162 438, 2019.

[4] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, Sep. 2019.

[5] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, ser. ICMR '17. New York, NY, USA: ACM, 2017, pp. 269–277. [Online]. Available: http://doi.acm.org/10.1145/3078971.3078974

[6] S. Sakazawa, E. Myodo, K. Tasaka, and H. Yanagihara, "Visual decoding of hidden watermark in trained deep neural network," in *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, March 2019, pp. 371–374.

[7] I. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker, *Digital Watermarking and Steganography*, 2nd ed. Morgan Kaufmann, 2007.

[8] F. Y. Shih, *Digital Watermarking and Steganography: Fundamentals and Techniques*, 1st ed. CRC Press, 2007.

[9] H. Chen, B. D. Rohani, and F. Koushanfar, "Deepmarks: A digital fingerprinting framework for deep neural networks," in *ICMR '19: Proceedings of the 2019 on International Conference on Multimedia Retrieval*, 2019.

[10] Y. Nagai, Y. Uchida, S. Sakazawa, and S. Satoh, "Digital watermarking for deep neural networks," *International Journal of Multimedia Information Retrieval*, vol. 7, no. 1, pp. 3–16, 2018.

[11] G. Wang, X. Chen, and C. Xu, "Adversarial watermarking to attack deep neural networks," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 1962–1966.

[12] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1615–1631.

[13] J. Guo and M. Potkonjak, "Watermarking deep neural networks for embedded systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2018, pp. 1–8.

[14] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," in *Advances in Neural Information Processing Systems*, 2015, pp. 685–693.

[15] E. Tartaglione and M. Grangetto, "Take a ramble into solution spaces for classification problems in neural networks," in *International Conference on Image Analysis and Processing*. Springer, 2019, pp. 345–355.

[16] H. W. Lin, M. Tegmark, and D. Rolnick, "Why does deep and cheap learning work so well?" *Journal of Statistical Physics*, vol. 168, no. 6, pp. 1223–1247, 2017.

[17] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[18] E. Tartaglione, S. Lepsøy, A. Fiandrotti, and G. Francini, "Learning sparse neural networks via sensitivity-driven regularization," in *Advances in Neural Information Processing Systems*, 2018, pp. 3878–3888.

[19] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2498–2507.

[20] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[21] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.