Information Sciences 576 (2021) 228–241

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

NeuNAC: A novel fragile watermarking algorithm for integrity protection of neural networks

Marco Botta *, Davide Cavagnino, Roberto Esposito

Computer Science Department, University of Turin, Corso Svizzera 185, 10149, Torino, Italy

ARTICLE INFO

Article history:
Received 18 January 2021
Received in revised form 20 May 2021
Accepted 23 June 2021
Available online 25 June 2021

Keywords:
Deep neural network
Fragile watermarking
Integrity protection
Linear transformation

ABSTRACT

The last decade has witnessed a massive deployment of Machine Learning tools in everyday life automated tasks. Neural Networks are nowadays in use in a growing number of application areas because of their excellent performances. Unfortunately, it has been shown by many researchers that they can be attacked and fooled in several different ways, and this can dangerously impair their ability to correctly perform their tasks. In this paper we describe a watermarking algorithm that can protect and verify the integrity of (Deep) Neural Networks when deployed in safety critical systems, such as autonomous driving systems or monitoring and surveillance systems.

1. Introduction

In the last decade the fast development of deep learning (DL) techniques has generated a massive interest in building, training and deploying systems that heavily relies on these models. The most popular class of deep learning models is Deep Neural Networks (DNN), which has been successfully adopted in many artificial intelligence applications, such as image recognition [14], natural language processing [21] and speech recognition [2,12]. A state-of-the-art Neural Network (NN) or DNN model is usually made of a large number of structured layers and a huge number of parameters to train. Therefore, a lot of computational resources, storage and time is needed to generate such a model and to deploy it in the product. Once deployed into a safety critical system, it is important to be able to verify the integrity of the model, in order to guarantee that it has not been tampered with. To achieve such an aim, a fragile watermarking method could be used to insert a watermark into the model to be protected and ensure the integrity of the system by checking if the watermark is intact.

In the field of computer security, the branch related to digital watermarking deals with the protection, in a wide sense, of digital objects. One of the main characteristics of watermarking is the embedding of a watermark signal w directly into the digital host object O: this differentiates watermarking from digital signatures which are an extra information to be carried along with the host object O. The watermark w may be a binary or real valued signal, and may also be, in some cases, the digital signature of the object O.

Many watermarking algorithms have been developed, with different objectives in mind, but all have the same high-level structure; in general, protecting an object with a watermark involves two complementary operations.

The first operation is the *embedding* E: it is performed only once and, in general, off-line, thus it may require more computational resources than the second step. The embedding phase embeds a watermark sequence w into the host object O: to

* Corresponding author.
E-mail addresses: marco.botta@unito.it (M. Botta), davide.cavagnino@unito.it (D. Cavagnino), roberto.esposito@unito.it (R. Esposito).

https://doi.org/10.1016/j.ins.2021.06.073
0020-0255/

make the whole process secure, or to increase its security, this phase may use a secret key k. The object O_w resulting from the embedding procedure may thus be expressed as $O_w = E(O, w, k)$.

The second operation is the *validation* (or *verification* or *detection*) V: this procedure is performed every time the need arises for testing if an object contains a watermark signal, therefore a fast and light process is to be preferred. When a possibly watermarked and altered object $O_w^{'}$ must be tested, the object is given to the validation function $V\left(O_w^{'}\right)$, which returns a boolean value stating if the watermark is present and eventually extracts the contained watermark signal $w^{'}$. In general, V requires the secret key k used by E and, in some watermarking algorithm, also the host object O and/or the original watermark w. This means that V may depend on all these parameters and hence is written as $V\left(O_w^{'}, w, k, O\right)$. Obviously, the kind of parameters required during validation changes the field of application of the algorithm.

A wide variety of watermarking algorithms have been developed to fit different goals and fields of application. To make their differences apparent, it is then useful to classify them by their characteristics, properties, and constraints. Also, in the following discussion, we will show how each property or characteristic applies to the specific case of the object of this study: i.e., the embedding of watermarks in Neural Networks.

Transparency: a watermarking algorithm may follow a *white-box* or a *black-box* approach depending on the visibility of the NN in the validation phase. If the validation function has no direct access to the parameters and structure of the NN, the approach is black-box, otherwise it is white-box. To clarify, in black-box systems, since the validation function cannot access the parameters of the NN, to establish if the watermark is present, V can only check the output of the NN over one or more samples.

Reversibility: if the validation phase may recover the host object O from the watermarked one the algorithm is called *reversible*, otherwise it is called *non-reversible* or *irreversible*. Recovering may need the secret key k used during embedding and, possibly, the watermark signal w. In case a NN is marked with a non-reversible algorithm, particular care must be given by the algorithm developer to show that the embedding of the watermark signal does not impact the NN performances.

Blindness: if the validation phase needs the host object O, that is V is in the form $V\left(O_w^{'}, \cdots, O\right)$, then the algorithm is called *informed*, or *non-blind*; otherwise, it is called *blind*. Applications where the NN is deployed in an external environment necessarily require blind watermarking algorithms.

Robustness: if the application context requires contrasting attacks to the watermarked object aimed at the removal of the watermark then the algorithm must be *robust* and the validation function should be able to recover the presence of the watermark signal even if the watermarked object has undergone severe modifications: this is the case of copyright protection, when a NN developer wants to protect its rights as owner of a product which required skills, efforts and computing power in training a NN. On the other hand, if the purpose of the application is to discover attacks aimed at modifying the NN behavior then a *fragile* watermarking algorithm is needed which has a validation function that detects the minimal modification to the watermarked object: for example, if a NN operates in a critical environment (e.g. driving a car or flying an airplane) it is important to be able to check its integrity.

Imperceptibility: this property is more related to multimedia objects, like music, images, videos, and refers to a subjective judgment of a human on the watermarked object stating if the watermark signal (e.g., a logo superimposed on an image or on a TV broadcast to identify the producer of the content) is *visible* (audible), in which case the watermark is said *perceptible*, or not (*imperceptible*). In some cases, the watermark is intentionally left perceptible, but in the NN context this has little sense because the main goal is to watermark a NN to the minimum extent to avoid altering its expected behavior.

Domain of embedding: the watermark signal may be embedded by directly modifying the original data of the host object (like the vertex coordinates of a 3D model or the pixel values of a digital image) or the original data may be transformed in a different domain, like the Singular Values Decomposition domain or the Wavelet Transform domain. In these cases, the watermark is embedded into the computed coefficients and finally the modified coefficients are converted back to the original domain by means of an inverse transform. In the first context the algorithm is said to work in *spatial domain*, *time domain* or *host domain*, whereas if a transform is involved, the watermark is embedded into the *frequency domain*. In the case of NN, algorithms may operate directly on the network parameters and structure or transform them in some chosen frequency domain before embedding.

In this paper, we describe NeuNAC (Neural Network Authenticity Checker), a fragile watermarking algorithm that may be applied to generic (deep or not) neural networks: it adopts a white-box approach to protect network structure and weights, it is blind, fragile and secure. Security is ensured by embedding the watermark in a secret frequency domain defined by the Karhunen-Loève Transform (KLT). Even if the algorithm is non-reversible, we will show that for many different tasks there is no degradation in performances of the watermarked neural network with respect to the original one. Starting from the framework proposed in [5], in this paper we devise a solution to face the new challenges posed by the digital object to be watermarked, i.e., a neural network. In this context, in fact, in addition to showing that the watermarked object has minimal distortion with respect to the original one (as it is normally done with images), one needs to take into consideration also the properties of reliability and performance. Specifically, one should demonstrate that it is very hard to find inputs producing diverging outputs for the host and the watermarked neural networks. Moreover, the intrinsic representation of a neural network needs a watermark embedding method that: *i)* is able to cope with the floating-point coding of the network weights and *ii)* protects the network structure.

The main properties and innovations of the NeuNAC algorithm with respect to the previous works are:

- *improved NN tamper detection and localization*: the integrity of the NN may be efficiently verified and any alteration to the parameters or to the structure of the NN can be detected with very high probability; moreover, the algorithm may localize at block level (as defined in Section 4) the tampered area;
- *security*: given that the watermark string is built applying a cryptographic hash function to some NN parameters and successively embedded into a secret frequency domain defined by a secret key, the watermarked object is protected against intentional and unintentional attacks and the watermark signal cannot be extracted by unauthorized entities;
- *lower distortion*: given that the modification of the weight values is made in the less significant digits of the floating-point representation, the distortion introduced by the watermark embedding is lower than the one the algorithm introduces for images;
- *adapted watermark generation strategy*: the watermark generation uses a secret key and the MSBs of some weights that are not modified by the embedding procedure;
- *efficacy*: as we will show in Section 5, the performances of the watermarked NN model are not impacted by the slight modifications due to the insertion of the watermark;
- *fast and light integrity check*: the complexity of the verification algorithm is linear in the number of NN parameters and it is designed to allow a sampling-based check of the integrity of the network so to make it usable in real time applications;
- *targeting embedded systems*: NeuNAC has been designed from the get-go to target embedded systems. Its unique design allows a hardware implementation of the verification process adding an additional level of security to the system.

The NeuNAC algorithm has been developed for DNNs deployed into safety–critical systems, such as (semi) autonomous car driving systems, energy monitoring systems etc., for which (intentional) tampering might cause severe and seriously damaging accidents. In these systems, a periodic and possibly asynchronous check for integrity should be constantly carried out. For instance, let us consider a DNN installed on a self-driving car to recognize pedestrians. It is not enough to check the integrity of the network at startup, as a malicious attacker could compromise the network afterwards. We propose to constantly and asynchronously validate the integrity of the network during its normal use and stop the system as soon as a tampering is detected. The validation phase of NeuNAC is very fast (few milliseconds) and can be easily implemented into a cheap hardware device.

2. Related works

Due to the widespread diffusion of images over the Internet, digital image watermarking has received much attention for more than a decade [3]. Recently, the massive deployment of Neural Networks, which require large engineering and computation efforts to be trained and possibly also to perform critical tasks, has posed the problem of protecting the intellectual property and the integrity of such networks.

For what concerns black-box algorithms, [32] proposes a DNN watermarking algorithm for copyright protection; ownership can be proved by querying the proprietary DNN model operated by a third party. To obtain this result, the model is trained with secret pre-defined watermark patterns that produce specific classifications. Also, in [1] the authors use backdooring and the definition of a trigger-set to watermark a DNN that can be subsequently tested for copyright protection without accessing the network parameters. [11] proposes a framework to perform robust watermarking of a DNN for embedded systems: to achieve a black-box approach, the DNN is fine-tuned with original examples both unmodified and modified with a mark derived from a signature, the latter examples producing a different classification label. [6] describes the BlackMarks framework: this black-box approach fine tunes a DNN to embed a binary watermark string by first creating a set of pairs (key image, label) with an adversarial attack and then embedding the set into the DNN. Adversarial examples are used in [20] to watermark a model by slightly altering its decision frontier. Szyller et al. [28] modify the classification output of the neural network in order to prevent model stealing attacks. Another black-box approach, tailored to protect DNNs for image processing operations rather than classification tasks, is presented in [25], where the authors exploit the overparameterization of the DNNs to produce a predefined output image when given a particular input image.

Wang and Kerschbaum in [30] show that the white-box method proposed in [23,29] may be attacked by exploiting the weight variance and the watermark removed with overwriting operations that embed another watermark: moreover, the new watermark does not suffer of the defect present in [23,29]. In a more recent paper [31] Wang and Kerschbaum present a robust watermarking algorithm which is undetectable and whose watermark extraction function is performed by a deep neural network trained with an adversarial network.

The above approaches differ from NeuNAC because their task is copyright protection, so the watermark must be robust to attacks that aim at its removal. All of these approaches use a black-box algorithm during ownership testing, while some fine-tune the networks and others build appropriate inputs that enable the watermark detection. NeuNAC, on the contrary, is intended for integrity protection. The watermark needs to be extremely fragile so that it easily breaks when the DNN is altered.

The only method we are aware of that is designed for integrity protection as NeuNAC is presented in [15] where the authors propose VerIDeep. VerIDeep is a black-box method that builds special examples that are sensible to minimal modifications of the neural network. Submitting them to an altered DNN would make the network to produce outputs that are different from the ones expected, thus allowing the system to identify that the network has been altered. Even though Ver-

IDeep shares several properties with NeuNAC, it is not designed for embedded systems because its black-box verification procedure requires querying the network synchronously, i.e., the network cannot be used while verifying its integrity. NeuNAC verification procedure, instead, can be run asynchronously while the DNN is in use.

3. Embedding tools

NeuNAC makes use of two important tools to embed the watermark into the neural network weights, namely the Karhunen-Loève Transform (KLT) and Genetic Algorithms (GA). This section gives a brief description of both; a detailed explanation of their use and methods of application is presented in [5].

The Karhunen-Loève Transform (also known as Principal Component Analysis, PCA) belongs to the set of linear transformations. A linear transformation is a mapping from an n-dimensional vector space to an m-dimensional vector space that can be described by a matrix $\boldsymbol{A}$ (called *kernel* of the transformation) of size $m \times n$. In our case $n = m$.

Given two vector spaces V, W of dimension n and a vector $x \in V$ then the forward transform is computed as $y = \boldsymbol{A}x$, $y \in W$: the elements of y are called coefficients of the transform. If $\boldsymbol{A}$ is square and full rank, the inverse transformation can be computed as $x = \boldsymbol{A}^{-1}y$.

Examples of linear transformations are the Discrete Cosine Transform (DCT) and the Fourier transform which have fixed kernels (for a pre-determined value of n).

The KLT has not a fixed kernel: in fact, its kernel matrix may be derived from a set of vectors $X \subset V$, $\mathrm{card}(X) > n$ (where $\mathrm{card}(X)$ represents the cardinality of the set X). Given a set of vectors X the transformation is computed as it follows:

- first the mean vector of X, $\mu = E\{X\}$, where $E\{.\}$ represents the expected value operator, and the covariance matrix $\boldsymbol{C} = E\left\{(X-\mu)(X-\mu)^T\right\}$ are computed, then
- the eigenvectors of $\boldsymbol{C}$ and their associated eigenvalues are evaluated and sorted by descending order of eigenvalue;
- the sorted eigenvectors are used to build the orthonormal kernel matrix $\boldsymbol{A}$ arranging them by rows.

The forward KLT of a vector v is computed as

$$y = \boldsymbol{A}(v - \mu) \tag{1}$$

and the inverse KLT is evaluated with

$$v = \boldsymbol{A}^{-1}y + \mu. \tag{2}$$

As it will be presented in the following, the neural network parameters may be given a total order and grouped into contiguous non-overlapping sequences: from each sequence it is possible to compute a set of n features (called, in the following section, Watermark Embedding Unit) to be used for embedding a digital watermark. In particular, the neural network parameters are modified so that the KLT (defined by an n-dimensional kernel $\boldsymbol{A}$) coefficients computed on the features store the watermark string.

The KLT kernel may be computed from any set of n-dimensional vectors: keeping this set secret allows the calculation of the coefficients of the transform, and consequently the watermark embedding space, only to authorized entities making the overall system secure. A simple method to have a set of n-dimensional vectors is to use the pixel values of an image or the samples of a sound, but obviously other approaches are possible.

A detailed presentation of KLT may be found in [9].

The modification of the neural network parameters to embed the watermark bit string into the features' KLT coefficients requires the solution of a complex non-linear problem. One possible method to cope with this non-linearity is to use a Genetic Algorithm. The reasons for this choice are the simplicity of the implementation, the availability of several ready to use libraries, the very good performances both in terms of solution quality and running time, and the fact that can be easily parallelized for even faster execution. GAs are a computational method that emulates the evolution of a population of biological individuals. Individuals are evaluated according to a fitness function and made evolve towards an optimal solution (i.e., one having the best value of the fitness function). A problem may be solved using a GA if its solutions: 1) can be stored in a sequence (record) of parameters: each instance of such a sequence represents an individual; 2) it is possible to evaluate the degree of goodness of an individual to fit the solution computing a (*fitness*) function.

A GA starts by generating (randomly or supervised by some problem-driven heuristic) a population of individuals; then, it evolves this population in a series of epochs. In each epoch individuals are reproduced to build a new population for the next epoch. The individuals are selected, according to their fitness value, to generate offspring with the objective of improving the quality of the solutions encoded. The reproduction is performed by mixing their descriptions; moreover, a mutation is randomly applied to some individuals to extend the solution search space. The population for the new epoch is built taking some individuals from the old population and some among the offspring according to various possible strategies. The evolution process stops after a pre-defined number of epochs is reached or an individual with the desired fitness value (e.g., below a predefined threshold) is obtained.

A deeper discussion on GAs may be found in [8,13] and an analysis of GA parameters in this context was presented in [4].

4. NeuNAC algorithm

The Neural Networks Authenticity Checker (NeuNAC) algorithm has the objective to insert a fragile watermark into the parameters of a trained DNN, without disrupting the ability to perform its task at the same level of performances as without the watermark. By appropriately grouping parameters into blocks, the presented algorithm also protects the structure of the model as a by-product of the embedding process (i.e., it is able to detect changes in the structure of the model in addition to changes to the weights).

The main modules of the method are the *Embedder* and the *Extractor*. Moreover, following the MIMIC framework [5], three other complementary modules are employed to perform accessory functions. The data flow between these modules is shown in Fig. 1. The description of the modules is presented in the following.

The *Key generator* module derives a KLT orthonormal basis (i.e., a kernel) from a secret set of vectors of the required size: presently we use $n = 32$ (see the Section 3). The computed orthonormal basis defines a secret embedding space that must be known to both the *Embedder* and the *Extractor* modules (like a secret key in symmetric encryption algorithms). We suggest using a secret image to form the set of vectors from which compute the KLT kernel, as proposed in [5].

In order to apply the embedding procedure, the parameters (each one represented with a 4 byte float in IEEE 754 format) of the DNN are grouped into blocks consisting of S elements each ($S = 16$ in the running example), called Parameter Unit (PU). There are several ways to make these groupings depending on the tampering localization granularity one would like to achieve. The simplest way to build the PUs is by extracting all the parameters from the DNN model (given a total order) and splitting them into sequences of length S. Another possible approach consists in proceeding level-wise, such as extracting all the parameters in a level and splitting them in sets of S elements. Yet another approach could be to consider each unit input weights and extracting the PU from them. It should be pointed out that even the first strategy outlined above allows to detect a structural change to the DNN, such as an added or removed unit, connection or level, but not a unit type replacement (such as swapping a sigmoid for ReLU). As a matter of fact, any such structural change will result into a different grouping of the weights and therefore a possibly different watermark will be extracted, signalling the tampering.

A portion of the watermark bit string w, built by the *Watermark generator* module, will be embedded in each PU. The string w is computed as a function of some DNN parameters (that will not be modified by the embedding step, as it will be clear in the following): starting from some bytes of these parameters a cryptographic hash function (c.h.f.), like the Secure Hash Algorithm 3, SHA-3 [16], is initialized and called to produce a bit string of the desired length. The security of the c.h.f. is not strictly required given that the embedding space is secret (the KLT kernel is known to embedder and extractor only), nonetheless its application is considered a good practice aimed at thwarting any possible attack.

Given a PU built as described above, a 32 bytes Watermark Embedding Unit (WEU) is derived concatenating the 16 parameters' least significant bytes (LSB) considering each value in its binary format (4 bytes float representation in IEEE 754 format) and a fingerprint of 16 bytes computed from the 3 most significant bytes (MSB) of each parameter value (see Fig. 2). In particular, the fingerprint is computed as follows: the 3 MSBs of each parameter value are concatenated together to form a string of bytes that is input to an MD5 c.h.f. [26] that returns a 16 bytes digest. In Fig. 2 note the bytes marked with vertical stripes: they are the LSBs of the mantissa of each float value. As it will be discussed presenting the *Embedder*, the watermark is inserted by altering only these bytes, while keeping unaltered the fingerprint bytes.

The *Embedder* produces a watermarked (Deep) Neural Network with a fragile watermark, operating one WEU at a time and independently among WEUs (thus, the process can be easily parallelized): each WEU is assigned a (consecutive) portion w_s of the w bit string output by the *Watermark generator*. If m is the length of the string w_s then we say that m is the payload in bpb (bits per block): this algorithm parameter is chosen a priori taking into account the fact that large values require more embedding effort but reduce the probability that an attack on a single WEU goes undetected, as it will be discussed in the conclusions.

Each WEU is considered a vector $(i_1, i_2, \cdots, i_{32})$ of 32 bytes coding integers: the GA modifies the sixteen bytes $(i_{17}, i_{18}, \cdots, i_{32})$ corresponding to the LSBs of the DNN parameters in the WEU (i.e. the vertical striped bytes in Fig. 2) in such a way that the 32 coefficients $(c_1, c_2, \cdots, c_{32}) = KLT(i_1, i_2, \cdots, i_{32})$ of the KLT of the WEU (computed according to equation (1))

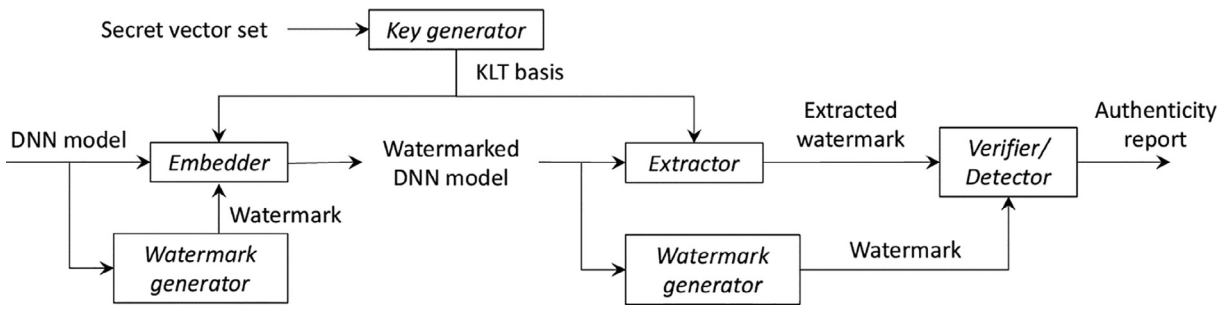

Fig. 1. Interconnections between the modules of the proposed algorithm and data interchanged.

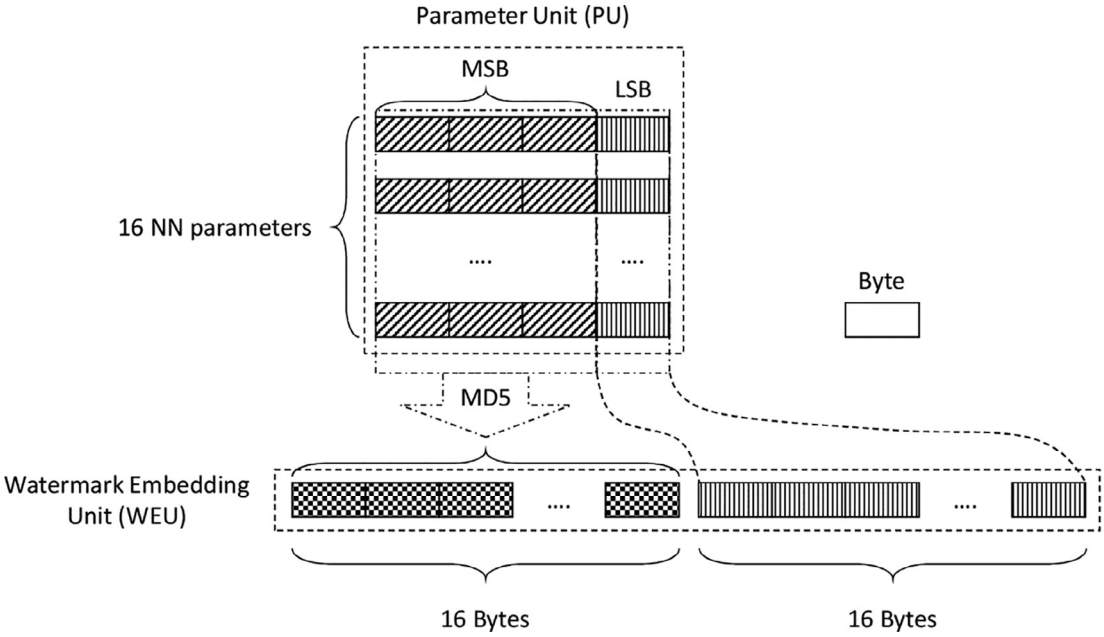

Fig. 2. Structure of the Watermark Embedding Unit.

embed the corresponding w_s string. Each individual of the GA population consists of 16 integers $(g_1, g_2, \cdots, g_{16})$defining the modification of the integers $(i_{17}, i_{18}, \cdots, i_{32})$; the GA fitness function privileges small modifications (e.g., 0, +1, −1, +2, −2,...) and requires that $(c_1, c_2, \cdots, c_{32})$ store w_s. In other terms, the GA tries to minimize the magnitude of the updates needed to change $(i_{17}, i_{18}, \cdots, i_{32})$ so that the $KLT(i_1, i_2, \cdots, i_{32})$ coefficients carry (store) the watermark payload. In particular, we used the following fitness function that tries to combine the quality of the solution and the insertion of the watermark:

$$Fitness(individual) = \sum_{i=1}^{16} |g_i| + m - count((c_1, c_2, \cdots, c_{32}), w_s) \tag{3}$$

where $count((c_1, c_2, \cdots, c_{32}), w_s)$ returns the number of watermark bits correctly present into the coefficients. Note that only individuals for which *count* returns m will be considered as solutions to the optimization problem. The GA is run for a maximum number of epochs (in all experiments reported below was 2000) or until the *Fitness* gets smaller than a threshold ε, whichever comes first. The GA evolves a population of 100 individuals, uses a single point crossover operator with probability $p_c = 0.8$, a mutation operator that increments of ± 1 one or more g_i, with probability $p_m = 0.05$, and a population replacement factor of 95% with elitism.

Some possible methods to store a bit string w_s into a set of coefficients are presented in [5]. If the length of w_s is m then one possible storing rule is to consider a (predefined) subset of m coefficients, choose m positions and assign a bit to each coefficient in the subset: a coefficient c is considered as carrying a bit b in position p if and only if

$$b = \text{round}(c2^{-p})\text{mod}2. \tag{4}$$

When this rule applies, we say that the set of coefficients store w_s if the m chosen coefficients carry the bits of w_s in the m chosen positions.

We found the GA particularly suitable for solving the non-linear problem of minimally modifying a set of integer numbers $(i_{17}, i_{18}, \cdots, i_{32})$ so that their KLT coefficients store a specific bit string. When the WEU stores the watermark bit string, the corresponding LSBs are used to replace the original LSBs of the DNN parameters.

When the GA has processed all the WEUs then the DNN model with the new modified parameters contains the fragile watermark. This watermark allows for the integrity check that can be performed by the *Extractor* and *Verifier* modules: note, in fact, that the KLT coefficients of each WEU depend not only on the parameters' LSBs but also on the MSBs parts.

The *Extractor* first calls the *Watermark generator* module to build the expected watermark string w_{XP}, that is the string that should be contained in the DNN if it was not forged. After that, the module builds the WEUs and using the KLT basis provided by the *Key generator* module extracts 32 coefficients from every WEU and from them gets the m bits according to the method used in embedding (e.g., the one defined in (4)). By concatenating all the bit strings extracted from the WEUs the *Extractor* composes the extracted watermark string w_{XT}.

The *Verifier* module matches the two strings w_{XP} and w_{XT}; if they are equal then the DNN is marked as authentic, otherwise it is tagged as potentially forged. It is quite straightforward to mark the forged WEUs because given a position number

$[0..T-1]$ to every WEU according to the portion of the watermark string w assigned during embedding, then the length of w (and consequently of w_{XP} and w_{XT}) will be mT, where m is the bpb payload: numbering the bits of w from 0 to $mT-1$, if the bits of the strings w_{XP} and w_{XT} in position F differ then the $\lfloor F/m \rfloor$-th WEU has been forged.

A DNN may be forged in several different ways, namely:

- modifying at least one of the weights, or
- altering one of the parameters of the activation functions, or
- removing or adding a single unit or a whole level to a DNN, or
- shuffling the order of the levels.

NeuNAC can protect the integrity of a DNN and detect the mentioned tampering attacks; in particular, if at least one of the weights or activation function parameters has been altered then the block containing the changed weight/parameter will be tagged as tampered. In all the remaining cases, the entire DNN will be tagged as tampered, because w_{XP} and w_{XT} will be different in a large number of WEUs, and when this happens it is very likely that the network structure has been tampered with.

5. Experimental evaluation

In Subsection 5.1 we evaluate the performances of the proposed watermarking method on several Deep Neural Network architectures and show its ability to detect tampering without affecting the DNN performances. As a further evidence of how effective the technique is to avoid influencing the DNN performances, in Subsection 5.2 we leverage adversarial attacks as a way to force the watermarked network to diverge from the original one.

5.1. Performance results

It should be pointed out that here we are not interested in the absolute performances of the considered DNNs on the example application domains, because they are not relevant to evaluate a watermarking method. Instead, we are interested in showing that such performances are not affected by the introduction of a watermark signal, that obviously alters the DNN parameters. To such an aim, we will measure the distortion introduced by the embedding phase and report performances of the proposed watermarking method (as usually done in the field) in terms of Peak Signal-to-Noise Ratio (PSNR, the higher the better) and distortion computed as Mean Absolute Error (MAE, the lower the better). PSNR and MAE are computed from the weights in the network as follows:

$$PSNR = 10\log_{10}\frac{\max_{w_i \in W}|w_i|^2}{\frac{\sum_{w_i \in W, w_i' \in W'}\left(w_i - w_i'\right)^2}{\mathrm{card}(W)}} \tag{5}$$

$$MAE = \frac{\sum_{w_i \in W, w_i' \in W'}|w_i - w_i'|}{\mathrm{card}(W)} \tag{6}$$

where W and W' are the sets of weights of the host and watermarked NN respectively, $\mathrm{card}(W)$ represents the number of elements in the set W and w_i and w_i' are the weights in corresponding positions in the host and watermarked NNs. Moreover, we also report performances in terms of classification accuracy difference between the original and watermarked DNN and show that there is no difference.

Table 1 reports the characteristics of the considered DNNs along with the datasets and tasks on which they have been tested. We selected several different DNN architectures, ranging from simple multilayer perceptron (2 fully connected layers) to Deep Convolutional Networks, such as Resnet29, to Recurrent Neural Networks, with different number of parameters, in order to show the ability of NeuNAC to watermark any network. These networks have been trained and tested on publicly available benchmarks, such as MNIST [19], CIFAR-10 [18] and Nietzsche [24] datasets.

Table 2 reports the performances of NeuNAC: the quality of the watermarked networks is extremely high, as shown by the very low distortion introduced by the embedding process and by the fact that there are no differences in performances between the original and the watermarked networks.

A comparison with other approaches is not straightforward as there are no white-box approaches for fragile watermarking, as far as we know, and the black-box approaches cannot report distortion values, as some of them are not actually embedding a watermark into the network. Nevertheless, Table 3 reports performance degradation and false positive rates of some watermarking methods as published by the authors of these studies and averaged over the set of experiments they performed. Most of them insert a robust watermark and are black-box methods, so a direct comparison is not possible.

The first two methods reported in Table 3 show a degradation in the performances of the watermarked networks and being robust watermarking methods do not report detection rates. BlackMarks and DeepMarks show an improvement in performances because the embedding process fine-tunes the target network, so improving its performances, but they have a low false positive rate, that goes to zero for specific settings. VeriDeep is a fragile watermarking method that shows very

Table 1
Benchmark neural network architectures. Here, 32C5(1) indicates a convolutional layer with 32 output channels and 5 × 5 filters applied with a stride of 1, MP2 (1) denotes a max-pooling layer over regions of size 2 × 2 and stride of 1, D(0.2) is a dropout layer with 0.2 probability and 512FC is a fully-connected layer with 512 output neurons. Moreover, LSTM stands for Long Short Term Memory cell.

Model No.	Model Type	Model architecture	Total # of parameters	Task	Dataset
1	MLP	784-512FC-10FC	623290	Classification	MNIST
2	Shallow CNN	28*28-32C5(1)-MP2(1)-D(0.2)-128FC-10FC	592074	Classification	MNIST
3	Deep CNN	28*28-32C3(1)-MP2(1)- 32C3(1)-MP2(1)- 32C3(1)-MP2(1)-D(0.2)-128FC-50FC-10FC	30000	Classification	MNIST
4	Deep CNN	3*32*32-32C3(1)–32C3(1)-MP2(1)-D(0.25)-64C3(1)-64C3(1)-D(0.25)-512FC-D(0.5)-10FC	1250858	Classification	CIFAR-10
5	Deep CNN	Resnet20	274442	Classification	CIFAR-10
6	Deep CNN	Resnet29	849002	Classification	CIFAR-10
7	RNN (LSTM)	128LSTM-128FC	102585	Text Generation	Nietsche

Table 2
Performance results.

Model No.	PSNR [dB]	MAE [$\times 10^{-9}$]	Performance difference
1	172.33	0.324	0
2	172.58	0.195	0
3	168.46	1.66	0
4	178.59	0.209	0
5	189.02	0.383	0
6	190.54	0.144	0
7	183.36	19.9	0
Average	181.99	3.26	0

Table 3
Comparison results.

Method	Method type	Performance difference (%)	False positive rate (%)	Detection rate (%)
Wang & Kershbaum [31]	White-box, robust	−0.4	Not reported	N/A
Le Merrer et al. [20]	Black-box, robust	−0.26	low	N/A
BlackMarks [6]	Black-box, robust	0.009	low	N/A
DeepMarks [7]	White-box, robust	0.04	low	high
VeriDeep [15]	Black-box, fragile	0	0	> 95
NeuNAC	White-box, fragile	0	0	95.4

similar performances as NeuNAC but, being a black-box method, its detection strategy can only spot modifications to the network that change the classification output of sensitive examples. NeuNAC, instead, can detect even the smallest change in more than 95% of the cases, as shown by the following sensitivity of the watermark experiment, that establishes the ability to detect tampering.

It is worth noting that, if the network has not been tampered with, NeuNAC verification procedure is guaranteed to correctly stating that the network is authentic, i.e., there are no false positives. Also, any structure modifications to the network, such as removing a layer or even a single connection, will result in an extracted watermark that is out of sync with the embedded one, as the watermark bit string also depends on the size of the network, and the network will be correctly identified as tampered. Moreover, a fine-tuning of the network will result in changing many weights of the network, and even very small changes might drastically change the binary representation of some parameter values, and therefore the extracted watermark will be different from the embedded one.

In order to provide a quantitative measure of the method sensitivity, we performed the experiments reported in Table 4. The table reports the percentage of blocks altered by performing a single fine-tuning step (1 epoch) on the considered network architectures along with the percentage of blocks found tampered by the verification procedure of NeuNAC w.r.t. the modified blocks. As pointed out above, any manipulation of the networks results in more than 92% of altered blocks, on average, and NeuNAC verification procedure is able to detect 99.65% of such alterations. To be clear, to detect a tampering attempt, the verifier only needs to identify the change of a single block. Hence, the experiments show the extreme sensitivity of NeuNAC on these kind of network modifications.

Furthermore, we consider every single WEU and systematically modify by ±1 or ±2 the value of each byte of the 16 bytes corresponding to the LSBs of the DNN parameters. This modification is the smallest possible perturbation as it is applied to the least significant bits of the mantissa of the DNN parameter. Table 5 reports the number of times a WEU is detected as

Table 4
Tampering detection after a fine-tuning step.

Model No.	% of altered blocks	% of recognized tampered blocks
1	91.51	99.69
2	100	99.59
3	99.15	99.57
4	100	99.59
5	97.06	99.62
6	67.46	99.60
7	100	99.88
Average	92.73	99.65

tampered as a percentage of all possible modifications. The reported values for Resnet29 model are consistent with all other DNN models outlined in Table 1, as the procedure does not depend on the specific architecture of the DNN model, but only on the type of modification performed.

As it can be noted, such a small tampering goes undetected in less than 5% of the cases, while, e.g., VeriDeep would not be able to detect such tampering (as implied by the second experiment in the next Section which introduces an attacking technique that is arguably more precise than VeriDeep in detecting these kinds of tampering). We would also argue that such small tamperings would not produce any meaningful behavioral change in the network, making them not worthwhile as a possible attack vector.

5.2. Adversarial experiments

NeuNAC protects from network tampering without compromising the predictions of the model. In this section we try to investigate this statement by studying the difference in behavior of the watermarked network w.r.t. the original network. To do that, we designed two sets of experiments where, using different approaches, we build tampered images designed to make the two networks differ in their outputs (when they do, we say that the predictions of the two networks *diverged* on that input). In Section 5.2.1 we shall try to find the closest adversarial image [27] for one of the two networks and check how the other network behaves on the tampered input. As we shall see, the two networks behave identically in these situations and a more focused attack is needed to compromise this behavior. In Section 5.2.2 we devise a technique that leverages the knowledge of both networks to build examples that make their predictions diverge (using a different technique, but much in the same spirit of VerIDeep [15] approach). We anticipate that also in this case it is very hard to fabricate an input image that exercises the small differences in the networks introduced by our watermarking algorithm. Since the introduced technique leverages more information than VerIDeep and is almost never able to create adversarial examples that make the two networks diverge, we conclude that VerIDeep would not be able to detect the tampering on the network introduced by our watermarking algorithm. Also, since our algorithm would, instead, be able to detect such small changes, we conclude that NeuNAC is much more likely than VerIDeep to detect very small changes to the network. This is not an unexpected result, because VerIDeep, being a black-box method, works in a more constrained setting.

5.2.1. FGSM adversarial examples

The Fast Gradient Sign Method (FGSM) is a technique proposed by Goodfellow et al. [10] for the problem of generating adversarial examples. In that paper a fast algorithm is presented that is able to inject a small amount of well-crafted noise into an image and make the network to fail in recognizing it. The algorithm works by updating the image x using the rule:

$$x' \leftarrow x + \epsilon \cdot sign(\nabla_x J(\theta, x, y)) \tag{7}$$

i.e., the adversarial image x' is built by moving x in the direction that causes the fastest change in the output of the network. In our experiments, we start with a very small value of ϵ and increase it until the network changes its prediction. The resulting image is very similar (very often indistinguishable) to the original image and a human is perfectly able to recognize the correct subject. We argue that these are important examples to check when we try to demonstrate that the watermarked network (does not) behaves differently from the original network: two networks that are operationally different have different decision boundaries and examples that lie nearby the decision boundaries are more likely to exercise the difference in behavior. This is exactly the case with FGSM examples since examples are modified using very small steps ($\epsilon = 0.001$) and we stop as soon as the output of the attacked network changes.

Table 5
Sensitivity results.

Model type	Model architecture	Total # of parameters	Sensitivity ± 1 (%)	Sensitivity ± 2 (%)
Deep CNN	Resnet29	849002	95.4	98.6

Experiments were conducted on the MNIST and the CIFAR datasets using three different models on MNIST and two different models on CIFAR. For each dataset/model combination, we extracted 5,000 training samples and 5,000 test samples and generated adversarial examples for them. We repeated the process twice: once using the original network to build the adversarial image and once using the watermarked network. A total of 100,000 images were, thus, generated. For each image we start with $\epsilon = 0.001$and then increase this value by 0.001 until either we reach 0.5 or we observe a change in the classification from the tested network. When we reach 0.5, we label that experiment as a failure in generating the adversarial example. Among the 100,000 images we generated, we have observed 9,008 failures. In the rest of the cases, we used the adversarial image to check the predictions of the two networks. In all cases the watermarked network showed to be very robust to the attack: the two networks predicted the same label for the adversarial example.

In addition to the above result, we leverage some side information from the performed experiments to further investigate the differences between the networks. In particular, since we repeated the experiments using both networks on the same set of images, we are in the position of checking if, despite giving identical outputs, they behaved differently during the experiments.

Tables 6 and 7 show for each dataset/model the number of times the FGSM algorithm failed to build the adversarial image when the original network is used (column 3) and when the watermarked network is used (column 4). In all cases these numbers match, so in this respect there is no difference in the behavior of the two networks. Columns 5 and 6 report the average ϵ needed to generate the adversarial image (failures are not counted). As we can see, only in three cases (marked as *1, *2 and *3) the two networks showed very minimal differences. Details of the problems found in these three cases are reported Table 8. For each case number (which references the corresponding lines on Tables 6 and 7) we report the *id* of the image, the ϵ value found by the original network (ϵ column) and of the watermarked network (ϵ_w column), the correct label for the image (y column) and the labels assigned by the original network to the attacking image (y' column) and by the watermarked network (y'_w column).

For case *1, we found three images with a small change in the behavior of the networks. For image 4010 and 4101 we can observe that a slightly different ϵ has been needed to produce the adversarial example. Results for image 881 is more interesting as the ϵ is the same, but the adversarial labels differ. To clarify: the difference is caused by two different attacking images found by the FGSM algorithm obtained by launching the algorithm on the two networks using the same input. The two networks do not diverge on these inputs, but they agree on different labels depending on which network was used to generate the input. We speculate that, in these cases, the gradient pointed to a place where labels 9, 5 and 6 competed and that a slight deviation in the direction caused the change in the labels assigned by the networks. On the examples reported for cases *2 and *3, except for image 4434, we observe only a small deviation of the ϵ values. Example 4434 is the most interesting case since we observe a deviation in both ϵ and the adversarial labels.

The seven cases we reported show that there exist properties of the networks that do change due to the presence of the watermark. Based on the results of the experiments, we conjecture that the gradient of the loss function taken with respect to the input image is very seldomly and very slightly changed by the watermark. The conjecture explains what we observed in the seven cases (out of the 100,000 experiments) we reported above.

5.2.2. Two networks adversarial examples

The FSGM experiments provide evidence that it is very hard to find cases where the two networks differ in their outputs. In order to further test this claim, we devised a novel adversarial algorithm explicitly tailored to make the two predictions diverge.

The technique is general but let us assume, for the sake of the discussion, that the activation function on the output of the network is a softmax. Let us denote with $N(x)$ and $N'(x)$ the outputs of the original and of the watermarked networks respectively. Let l denote the cross-entropy loss and let y_0 and y_1 be the one-hot-encoding of the best and of the second-best predictions by $N(x)$ evaluated once and for all before the algorithm start. We define the loss L of our problem as

$$L\Big(N(x), N'(x)\Big) = l(N(x), y_0) + l(N'(x), y_1) \tag{8}$$

Table 6
FGSM experiments on the CIFAR dataset. Numbers on the left of the table are referenced in Table VIII and corresponds to experiments where we observe some difference in the behavior of the networks.

	Model	Train/Test	Number of failures		Averageϵ	
			Original	Watermarked	Original	Watermarked
	Baseline	train	287	287	0.048649	0.048649
	baseline	test	317	317	0.057502	0.057502
*1	resnet20	train	213	213	0.044305	0.044303
*2	resnet20	test	232	232	0.050750	0.050747

Table 7
FGSM experiments on the MNIST dataset. Numbers on the left of the table are referenced in Table VIII and correspond to experiments where we observe some difference in the behavior of the networks.

	Model	Train/test	Number of failures		Averageϵ	
			Original	Watermarked	Original	Watermarked
	baseline	train	525	525	0.111667	0.111667
	baseline	test	386	386	0.092779	0.092779
	large	train	1027	1027	0.257718	0.257718
	large	test	773	773	0.216779	0.216779
	largemodel	train	395	395	0.241232	0.241232
*3	largemodel	test	349	349	0.227074	0.227073

Table 8
Differences in behavior found in the FGSM experiments.

Case	image	ϵ	ϵ_w	y	y'	y'_w
*1	881	0.026	0.026	9	5	3
*1	4010	0.344	0.340	3	6	6
*1	4101	0.018	0.014	1	8	8
*2	1536	0.277	0.281	3	2	2
*2	4434	0.034	0.022	2	3	8
*2	4749	0.375	0.371	3	6	6
*3	3059	0.046	0.042	7	1	1

and generate the adversarial examples by descending the gradient of L using the following update rule:

$$x \leftarrow x - \eta \nabla_x L(N(x), N'(x)). \tag{9}$$

It should be apparent that by descending the gradient of that particular loss function, we are searching for an input image x that makes $N(x)$ to predict y_0 and $N'(x)$ to predict y_1, hence making the two networks diverge. The particular choice of y_0 and y_1 simplifies the task since y_0 is already the label that is ranked the highest by network $N(x)$ and y_1 is its second-best choice (hence the easiest label to "reach" by updating x).

We experimented with the same dataset and models described in Section 5.2.1. In each experiment we adopted Adam [17] for performing the gradient descent and run it for $10,000$ rounds (stopping every 10 rounds to check if the network outputs diverged). The initial value of the learning rate η has been set to 0.001.

Due to the high cost of the optimization procedure, we could not afford to experiment with $5,000$ images per experiment and therefore reduced that number to 100. In total we attempted the construction of $1,000$ adversarial images. In two cases, we found adversarial images that were able to make the watermarked network and the original network to behave differently. In both cases, the attack succeeded on the baseline neural network model, while more complex networks never diverged under this type of attack. The two images that made the networks to diverge are reported in Figs. 3 and 4.

This experiment shows how difficult it is to find a valid attack that makes the two networks to produce different outputs. While we acknowledge that it can be done, we would like to stress the fact that even in the most favorable conditions (simultaneous white box attack on both networks) the attack succeeded only on 0.2% of the trials and with a large computational cost.

We also verified that the attack is very brittle and the smallest change to the images makes the two networks to agree again on the answer. Specifically, we tried to perturb the images in two different ways: in one case we lighten up the image by the smallest possible value (we added 1 to the value of every pixel in the image), in the other case we made the smallest possible modification to the image, i.e., we added (and, in a distinct experiment, subtracted) 1 to a single pixel in the image. In the first case we verified that the networks went back to agreeing on the assigned labels after the lighting up of the image. In the second case, we systematically changed every single pixel in the image and verified that, with the exception of a single pixel position (let it be position p), the change would ruin the attack, making the two networks to agree again on the label to be assigned. In the specific case of the pixel in position p, we verified that changing the perturbation to $+2$ (or -2) would also ruin the attack.

In summary, all the experiments we performed show that it is almost impossible to induce a different behavior between the original and the watermarked networks. While adversarial attacks are possible, they are very hard to find: they would work only on the simplest models, the attacker would need to have the two networks in hands and he/she would need to be ready to sustain large computational costs. They would also be very brittle: the smallest change in the attacked image would defeat the attack. Brittleness is particularly important in many contexts since it makes these kinds of attack very hard to deploy in real scenarios (e.g., in the case of self-driving vehicles, the noise in capturing the signal would be enough to make the attack fail).

Fig. 3. Original (left) and adversarial (right) images that causes the two networks to output different labels (CIFAR dataset).

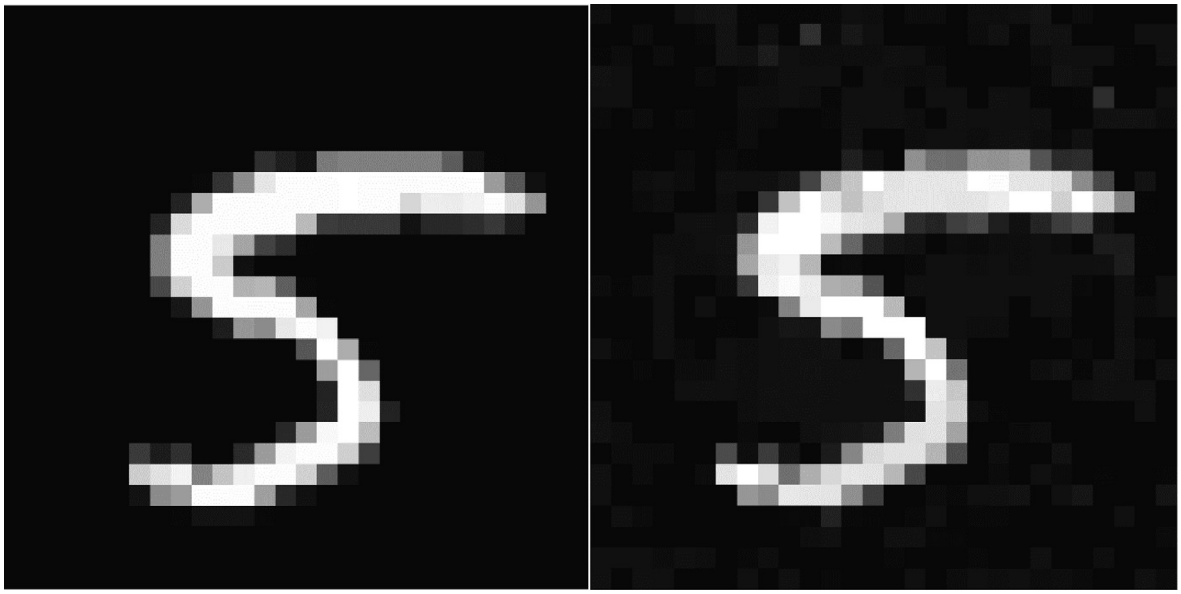

Fig. 4. Original (left) and adversarial (right) images that causes the two networks to output different labels (MNIST dataset).

6. Discussion

The embedding procedure of NeuNAC results in very low distortion that does not influence the performances of the neural networks, as shown by the experiments described in the previous section. We also proposed a novel way to validate a fragile watermarking algorithm for DNNs, by performing a thorough analysis of sensitivity of tampering detection: this analysis reveals the ability of the verification procedure to detect the smallest tampering in more than 95% of the cases. As mentioned, any other modification of a DNN would result in larger tampering that will be easily detected.

In principle, even a small change in a network parameter might affect the network performance, as it is the case with generative networks that produce numerical outputs. In such networks, the output generated by the watermarked version of the DNN differs from the original one. This is the reason why NeuNAC, in its current implementation, cannot be used to

watermark generative neural networks, such as DeepDream [22]. In contrast, in case of classification networks, the tiny changes introduced by NeuNAC are cut off by the classification and/or MaxPooling layers.

Almost all the existing works on neural network watermarking are about copyright protection, being VeriDeep the only notable exception, but integrity protection and authentication are very important tasks that require a fragile watermarking scheme such as the one implemented by NeuNAC.

As previously stated, the proposed method is secure because the watermark embedding space is secret. In case of a payload of m bpb the probability that a random modification of the NN parameters in a PU goes undetected is $1/2^m$, thus the probability that an attack changing the contents of k PUs is not identified by the proposed scheme is $(1/2^m)^k = 1/2^{mk}$: this probability exponentially drops with the number of tampered PUs.

One drawback of the current implementation of NeuNAC is that it does not protect from activation function changes, when there is no change in number of parameters. A possible solution that will be investigated in the future is to make the watermark also dependent on the structure and the activation functions of the neurons (right now is only dependent on the number of parameters). Another aspect that needs more investigation is how to split the network structure in meaningful parts that could be watermarked separately and allow to better localize the tampering or make the method resilient to possible permutations of network blocks (e.g., a convolutional layer) that do not change the network behaviour.

7. Conclusions

In this paper, we presented NeuNAC, a white-box watermarking method for integrity protection of (Deep) Neural Networks. It can be applied to any kind of neural network architecture (deep or shallow), as it inserts a watermark bit string into the parameters of the network. The order in which the parameters are considered is relevant, as this also allows to protect the structure of the network from neuron and/or layer rearrangement/addition/removal operations. NeuNAC quality is outstanding, as it does not have any impact on the performances of the network and the average PSNR is greater than 181 dB.

The main goal of NeuNAC is to protect the integrity of neural networks for safety–critical systems once deployed into their operating environment. In order to guarantee the authenticity of the network, the watermark verification process has been designed to be very fast so that it can be performed frequently. It can also work on a sampled set of WEUs and it can be realized into a hardware device for even faster and (hence) secure execution.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been partially funded by the EU Horizon 2020 research and innovation program ECSEL Joint Undertaking (JU) under Grant Agreement No. 876487, NextPerception project – "Next Generation Smart Perception Sensors and Distributed Intelligence for Proactive Human Monitoring in Health, Wellbeing, and Automotive Systems." The JU receives support from the EU Horizon 2020 research and innovation programme and the nations involved in the mentioned projects. The work reflects only the authors' views; the European Commission is not responsible for any use that may be made of the information it contains.

References

[1] Y. Adi, C. Baum, M. Cisse, B. Pinkas, J. Keshet, Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring, in: In: Proceedings of the 27th USENIX Security Symposium (USENIX Security), 2018, pp. 1615–1631.
[2] D. Amodei et al., Deep Speech 2: End-to-End Speech Recognition in English and Mandarin, arXiv:1512.02595v1 [cs.CL], available at https://arxiv.org/abs/1512.02595 (2015).
[3] M. Begum, M.S. Uddin, Digital Image Watermarking Techniques: A Review, Inform. MDPI 11 (2) (2020) 110, https://doi.org/10.3390/info11020110.
[4] M. Botta, D. Cavagnino, V. Pomponiu, Automatic Selection of GA Parameters for Fragile Watermarking, LNCS 8602 (2014) 526–537.
[5] M. Botta, D. Cavagnino, V. Pomponiu, A modular framework for color image watermarking, Signal Process. 119 (2016) 102–114.
[6] H. Chen, B.D. Rouhani, F. Koushanfar, BlackMarks: Blackbox Multibit Watermarking for Deep Neural Networks, arXiv:1904.00344v1 [cs.MM], available at https://arxiv.org/abs/1904.00344 (2019).
[7] H. Chen, B.D. Rouhani, C. Fu, J. Zhao, F. 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), Ottawa ON Canada, ACM, New York, NY, USA, 2019, pp. 105–113, https://doi.org/10.1145/3323873.3325042
[8] D.E. Goldberg, Genetic Algorithms in Search, Addison-Wesley Publishing Company, Optimization and Machine Learning, 1989.
[9] R.C. Gonzalez, P. Wintz, Digital Image Processing, 2nd ed.,., Addison-Wesley Publishing Company, 1987.
[10] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and Harnessing Adversarial Examples, arXiv:1412.6572v3 [stat.ML], available at https://arxiv.org/abs/1412.6572 (2015).
[11] J. Guo, M. Potkonjak, Watermarking Deep Neural Networks for Embedded Systems, In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 2018, pp. 1–8, doi: 10.1145/3240765.3240862
[12] A.Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, A.Y. Ng, Deep Speech: Scaling Up End-to-end Speech Recognition, arXiv:1412.5567v2 [cs.CL], available at http://arxiv.org/abs/1412.5567 (2014).

[13] A.-E. Hassanien, A. Abraham, J. Kacprzyk, J.F. Peters, Computational Intelligence in Multimedia Processing: Foundation and Trends, In: Hassanien A.E., Abraham A., Kacprzyk J. (eds) Computational Intelligence in Multimedia Processing: Recent Advances. Studies in Computational Intelligence, vol. 96, Springer, Berlin, Heidelberg, 2008, https://doi.org/10.1007/978-3-540-76827-2_1
[14] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Las Vegas, NV, 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90
[15] Z. He, T. Zhang, R.B. Lee, VerIDeep: Verifying Integrity of Deep Neural Networks through Sensitive-Sample Fingerprinting, arXiv:1808.03277v2, available at https://arxiv.org/abs/1808.03277 (2018).
[16] P. Hernandez, NIST Releases SHA-3 Cryptographic Hash Standard, 2015, Available at https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard (Retrieved on 4th December 2020.)
[17] D.P. Kingma, J.L. Ba, Adam: A Method for Stochastic Optimization arXiv:1412.6980v9 [cs.LG], available at https://arxiv.org/abs/1412.6980 2017.
[18] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inform. Process. Syst. (2012) 1097–1105.
[19] Y. LeCun, C. Cortes, C.J. Burges, The MNIST database of hand-written digits. http://yann.lecun.com/exdb/mnist (1998).
[20] E. Le Merrer, P. Pérez, G. Trédan, Adversarial frontier stitching for remote neural network watermarking, Neural Comput & Applic 32 (13) (2020) 9233–9244, https://doi.org/10.1007/s00521-019-04434-z.
[21] M.-T. Luong, H. Pham, and C.D. Manning, Effective Approaches to Attention-based Neural Machine Translation, In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2015, pp. 1412–1421, doi: 10.18653/v1/D15-1166
[22] A. Mordvintsev, C. Olah, M. Tyka, DeepDream - a code example for visualizing Neural Networks, Google Research (2015) https://ai.googleblog.com/2015/07/deepdream-code-example-for-visualizing.html
[23] Y. Nagai, Y. Uchida, S. Sakazawa, S. Satoh, Digital watermarking for deep neural networks, Int. J. Multimedia Inform. Retrieval 7 (1) (2018) 3–16, https://doi.org/10.1007/s13735-018-0147-1.
[24] Nietzsche texts: A rich dataset of English text. https://www.kaggle.com/pankrzysiu/nietzsche-texts
[25] Y. Quan, H. Teng, Y. Chen, H. Ji, Watermarking deep neural networks in image processing, IEEE Trans. Neural Networks Learn. Syst. 32 (5) (2021) 1852–1865, https://doi.org/10.1109/TNNLS.596238510.1109/TNNLS.2020.2991378.
[26] R. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, April 1992.
[27] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, arXiv:1312.6199v4 [cs.CV], available at https://arxiv.org/abs/1312.6199 (2014).
[28] S. Szyller, B. Gul Atli, S. Marchal, N. Asokan, DAWN: Dynamic Adversarial Watermarking of Neural Networks, arXiv:1906.00830v4 [cs.CR], available at http://arxiv.org/abs/1906.00830 (2019).
[29] Y. Uchida, Y. Nagai, S. Sakazawa, S. Satoh, Embedding Watermarks into Deep Neural Networks, In: Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval (ICMR '17), ACM, New York, NY, USA, 2017, pp. 269–277, https://doi.org/10.1145/3078971.307897
[30] T. Wang, F. Kerschbaum, Attacks on Digital Watermarks for Deep Neural Networks, in: In: ICASSP 2019–2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, pp. 2622–2626, https://doi.org/10.1109/ICASSP.2019.8682202.
[31] T. Wang, F. Kerschbaum, RIGA: Covert and Robust White-Box Watermarking of Deep Neural Networks, arXiv:1910.14268v3 [cs.CR], available at https://arxiv.org/abs/1910.14268 (2020).
[32] J. Zhang, Z. Gu, J. Jang, H. Wu, M.Ph. Stoecklin, H. Huang, I. Molloy, Protecting Intellectual Property of Deep Neural Networks with Watermarking, In: Proceedings of the 2018 Asia Conference on Computer and Communications Security (AsiaCCS '18), 2018, pp. 159–172, https://doi.org/10.1145/3196494.3196550