DeepAuth: Protecting Integrity of Deep Neural Networks using Wavelet-based Steganography

Abstract—**Training Deep Neural Networks (DNNs) is very expensive in terms of computational power and the amount of training data required. This led to the emerging business trend where pre-trained models are treated as a first-class object and traded as services or commodities in the Internet marketplace such as Machine Learning as a Service (MLaaS). However, researchers have shown that such pre-trained models are vulnerable to different types of security threats, such as poisoning attacks. Hence, it is essential to protect the integrity and authenticity of these models in the emerging marketplace. The challenge is to construct a method to derive a signature of such pre-trained models and embed it while preserving the accuracy of the original model. Also, the embedded signature should be securely concealed within the model and can be verified at all times.**

To address these challenges, this paper proposes a novel wavelet-based steganographic technique, called DeepAuth. DeepAuth generates the signature of a DNN model using its structural information. To maintain the model accuracy, DeepAuth uses a wavelet-based technique to transform weights in each layer from the spatial domain to the frequency domain. It then identifies the approximation coefficients to preserve the accuracy of the model and utilizes the detailed coefficients to hide the signature using a combination of secure key and scrambling vector. Our analysis results show that DeepAuth can hide about 1KB signature in each layer with 256-bit security level without having any impact on the accuracy of the model. Several experiments are performed on 3 pre-trained models (ResNet18, VGG16, and AlexNet) using 3 datasets (MNIST, CIFAR-10, and ImageNet) against 3 types of manipulation attacks (input poisoning, output poisoning, and fine-tuning). The results demonstrate that DeepAuth is verifiable at all times without degrading the classification accuracy, and robust against a multitude of known DNN manipulation attacks.

I. Introduction

Deep Neural Networks (DNNs) have been found to remarkably outperform previous machine learning techniques in a wide range of applications such as image recognition [1], autonomous transport [2], speech processing [3], machine translation [4], and games [5]. However, training these networks is computationally expensive and time-consuming. To address these issues, there are several attempts to develop various open-source frameworks (e.g., TensorFlow [6], Pytorch [7]) for simplifying the design and deployment of DNN, and release pre-trained models such as ResNet [8], VGG [9], and AlexNet [10]. These attempts have been successful to a certain extent.

The trade-off between the significance of DNNs and the burden of their training requirements has opened a new market opportunity for Machine Learning as a Service (MLaaS). Companies can design and train ML models using their datasets and powerful platforms, and make such models, such as Google Cloud ML Engine [11], Microsoft Azure ML Studio [12], and Amazon Sagemaker [13], available to their customers. A large number of small/medium companies are going to use such MLaaS to commercialize their innovative ideas. Hence, we expect to see a sharp growth in DNN models being made available as MLaaS. That is, pre-trained models are treated as a first-class object and traded as services or commodities in the internet marketplace as MLaaS.

However, MLaaS faces several challenges concerning the integrity and authenticity of underlying DNN models. These models could be tampered by sophisticated attacks such as poisoning (e.g., [14], [15]). Fig. 1 shows an example of poisoning attacks – a stop sign can be misclassified as a speed sign when the DNN model is tampered with slightly modified weights. Such attacks have consequences for the model owner/developer, MLaaS provider, and service consumers as none of them can verify the integrity and authenticity of the models used in MLaaS. The challenge is that how to develop a technique to derive and insert a signature in the models so that the integrity and authenticity of models (e.g., weights, dataset version, and outer layer) can be ensured at all times.

Existing approaches to protect the integrity of DNNs can be broadly categorized into two types: (1) defense mechanisms against adversarial samples and (2) defense mechanisms against poisoning attacks. The former tries to determine if the given input is benign or adversarial (see [16], [17], [18], [19], [20]). This approach is generally effective to detect adversarially crafted inputs, but cannot detect whether backdoors or trojans have been inserted into DNN models. The latter aims to address this by defending against the poisoning attacks by anomaly detection, retraining and generating sensitive samples [21], [22], [23]. Despite the effectiveness of these works, they lack a mechanism to capture a fingerprint of the model and embed the fingerprint invisibly within the model as a unique signature before releasing it. The signature should always be verifiable and hidden undetectably. Moreover, it does not significantly degrade the accuracy of the original model. Hence, the signature can be verified at all times to ensure the integrity and authenticity of the model.

To address the aforementioned challenges, this paper explores a solution based on steganography. While steganography may sound new to the machine learning community, it has been widely studied and used in the multimedia domain [24]. It is a technique of hiding a secret message within a larger transferred content (e.g., image or sound) so that only the intended recipient can recover it [25]. Hence, steganography would be a promising tool to secretly hide the signature in

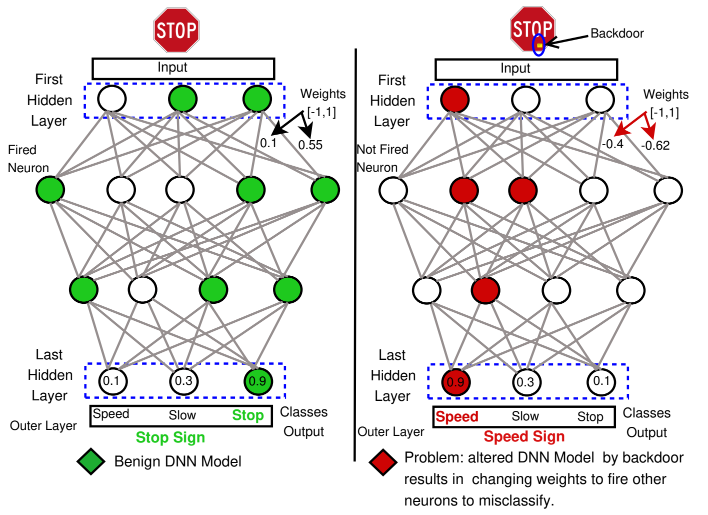


Fig. 1. Demonstration of an example of DNN poisoning attacks using backdoors on the input image. On the benign DNN side, the input is a stop sign and so the correct output is "Stop". However, on the backdoored DNN side, a backdoor inserted in the stop sign input to give a wrong prediction as "Speed".

order to provide the integrity and authenticity of DNN models; However, the direct application of steganography to DNN models has two limitations. First, it cannot ensure controlled access to the signature because the hiding is performed in fixed positions (i.e., it is difficult to ensure the confidentiality of the hidden signature). Hence, once attackers know that there are hidden signature bits, they may easily access and manipulate them. Second, it has a distortion impact on the transferred content. Such a distortion may not be a significant issue in the multimedia domain as they rely on the perceptibility of the human eye (e.g., a few greyer pixels are not a problem in an image). However, in the context of DNN, it should be taken carefully due to the sensitivity of weights in the hidden layers, which might have significant impacts on the performance of the DNN model.

To address those two challenges, this paper proposes a novel wavelet-based steganography technique, called DeepAuth. To the best of our knowledge, this is the first attempt to propose a steganography-based technique for the integrity and authenticity of DNN models.

To solve the distortion issue inherent to steganography, we employ wavelet decomposition to transform weights in the hidden layers from the spatial domain to the frequency domain. We apply appropriate scaling to the coefficients to limit the distortion and preserve the accuracy of the model. DeepAuth generates a signature related to each layer using the structural information and uses the detailed coefficients in the frequency domain to hide the signature randomly bit-by-bit inside the corresponding layer.

To strengthen the confidentiality of the signature and make it undetectable, we use a combination of unique key and scrambling vector to encrypt and randomise the hiding process. We performed both mathematical analysis and extensive empirical exploration to find a suitable steganography level, weights size per transform, appropriate coefficients, and scaling criteria. Our studies show that DeepAuth can hide a 2-bits message in each coefficient and the total of 1KB signature in each layer with 256-bit security level without having any significant impact on the accuracy of the model.

To validate DeepAuth, we provide extensive empirical evidences by performing several experiments on three pre-trained models (ResNet18 [8], VGG16 [9], and AlexNet [10]) using three datasets (MNIST [26], CIFAR-10 [27], and ImageNet [28]) against three types of manipulation attacks (input poisoning [14], output poisoning [15], and fine-tuning [29]). The results prove that DeepAuth is verifiable at all times with no noticeable effect on the classification accuracy, and robust against a multitude of known DNN manipulation attacks.

The rest of the paper is organized as follows: Section II gives the pertained background to our work. Our DeepAuth technique is introduced in Section III. Section IV introduces the theoretical analysis for DeepAuth. Section V presents the experimental results to evaluate DeepAuth. Section VI presents the related work. Finally, Section VII concludes the paper.

II. Background

This section provides the required background information to understand the proposed system. We briefly summarize the deep neural networks, steganography and wavelet transform.

A. Deep Neural Network (DNN)

A DNN is a part of broader machine learning methods based on artificial neural networks where input feature extraction is performed automatically [30]. A DNN can be depicted as a mapping function $f_\Theta : \mathbb{R}^n \Rightarrow \mathbb{R}^m$ that matches an input $x \in \mathbb{R}^n$ to an output $y \in \mathbb{R}^m$ based on the calculated parameters Θ. Let us assume x is an image of number '6' that has to be classified by f_Θ into y as a vector of probabilities corresponding to $C \in [0, 9]$ classes. The output with the highest probability $arg\ max_{i \in [1,m]} y_i$ is pointing to the image prediction label C_i (i.e., number '6').

A DNN is constructed with L hidden layers. Each layer $l_{i \in [1,L]}$ has n_i neurons. The value of each neuron, called activation a, which is computed in a feed-forward propagation way. a_i activation for the i^{th} DNN layer can be calculated as follows

$$a_i = \varphi(w_i a_{i-1} + b_i) \quad \forall_i \in [1, L] \tag{1}$$

where $\varphi : \mathbb{R}^n \Rightarrow \mathbb{R}^n$ is a non-linear function that ensures only crucial neurons to be fired (i.e., > 0) and forwards its output as an input to the next layer. $w_i \in \mathbb{R}^{n_{i-1}} \times n_i$ is the *weights* and $b_i \in \mathbb{R}^n_i$ is the *biases*; both of them are learned during training. The DNN output of the last hidden layer l_L activations is a function, called softmax. It can be calculated as $y = \vartheta(w_{L+1} a_L + b_{L+1})$, where $\vartheta : \mathbb{R}^n \Rightarrow \mathbb{R}^n$.

The most crucial parameters are the DNN *weights* w_i in each layer l_i. Their values are learned from the original dataset. These weights ensure that only certain neurons are fired leading to the most accurate classification. Naturally, alterations to *weights* in the original DNN model (e.g., by poisoning attacks) produce misclassification results.

The other crucial DNN element is the outer layer containing the C_i class labels. Altering them yields a severe misclassification without manipulating the *weights*.

Therefore, our steganographic algorithm focuses on preserving the integrity of *weights* in all hidden layers as well as the outer layer without compromising the accurate functioning of the models.

B. Steganography

The term steganography often gets confused or lumped together with digital watermarking [25]. They are similar in the sense that both techniques hide a message β in some cover data D. However, they have different goals. The aim of watermarking is that an eavesdropper cannot remove or replace β in D. Hence, resilience is the main concern for the design of watermarking. There are two types of watermarking: visible and invisible [25]. The watermarking has been extensively used to protect copyright and IP ownership.

On the other hand, steganography is always invisible, and its aim is that an eavesdropper should not be able to detect the presence of the secret β in D. Therefore, the resultant distortion on D and the confidentiality of β are the two main concerns in steganography.

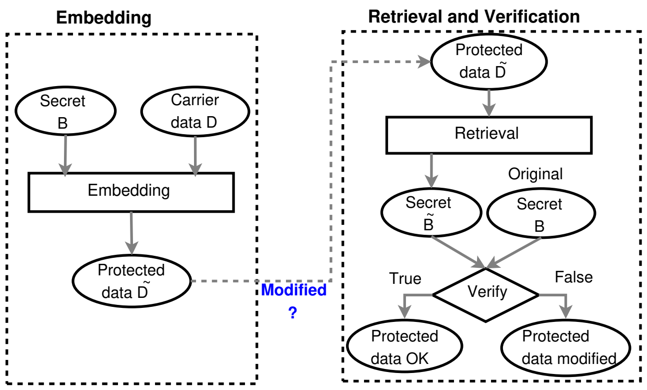


Fig. 2. A typical steganography life cycle.

Steganography has been extensively used to protect the integrity of the data in the multimedia domain such as medical X-rays [31] and MRI images [32]. The procedure generally consists of two stages: (1) embedding; and (2) retrieval and verification (see Fig. 2). In the embedding stage ($\tilde{D} = f_e(\beta, D)$), a designed algorithm f_e hides a pre-defined secret β into the carrier medium D to be $\tilde{D}$ which becomes the protected content. During the verification ($\tilde{\beta} = f_r(\tilde{D})$), a developed algorithm f_r retrieves a secret $\tilde{\beta}$ from $\tilde{D}$. $\tilde{\beta}$ needs to be further verified with the original β. If they are the same, it confirms that the carrier data $\tilde{D}$ is pure and not changed. Otherwise, the carrier data is tampered by adversaries.

Steganography is used to protect the integrity and authenticity of the multimedia data by directly embedding the secret bits into carrier medium like images. In this work, we target to protect the integrity and authenticity of DNN models. Hence, a new steganographic technique needs to be developed to embed secret bits into deep hidden layers in the DNN model so that it does not lose the predictive capability, i.e., the sensitivity and relationships of weights are maintained. The wavelet transform is employed in DeepAuth to solve the distortion challenge. We next provide a brief overview of the wavelet transform.

C. Wavelet Transform

Wavelet Transform (WT) is a popular linear function used in the signal and image processing where the content (e.g., signal) is transformed from its spatial time domain into its frequency domain to identify the most and the least significant parts of the signal [33]. It is basically performed on the given signal, which leads to decomposing it into different values, called coefficients, that represent its frequency components at a given time. WT is shown in Eq. 2,

$$C(i,j) = \int_{-\infty}^{\infty} f(t)\psi(i,j)dt \tag{2}$$

where i and j are positive integers that represent the transform parameters. C is the resultant coefficients. ψ is a wavelet function [34]. WT can be applied in either Continuous (CWT) or Discrete (DWT) manner. However, the latter is preferred in real-world applications because the analyzed information in many cases comes in discrete numbers rather than continuous functions [35].

To conduct DWT, two-band filters (high-pass and low-pass) are applied to the original signal. Consequently, two sub-signals (called bands) are obtained. The first is the low-frequency components which represent the approximation of the original signal. The second is the high-frequency components which represent the detailed coefficients. When this process is repeated multiple times, it is known as a multi-level wavelet decomposition [34], [36]. DWT is defined in Eq. 3,

$$Y(a,b) = \sum_{a}\sum_{b} X(a)\Phi_{ab}(n) \tag{3}$$

where $Y(a,b)$ represents wavelet coefficients; a and b represent the shift and scale transform parameters. $\Phi_{ab}(n)$ represents the base time of the wavelet function which is shown in Eq. 4,

$$\Phi_{ab}(n) = 2^{-a/2}\Phi(2^{-a}n - b) \tag{4}$$

III. DeepAuth

Motivating Scenario: Fig. 3 shows an example of how our DeepAuth can be used to validate the integrity and authenticity of DNN models. The model developer uses DeepAuth to protect his or her own DNN model before deploying it as MLaaS by hiding the signature secretly. The trusted validator requests the model from the MLaaS provider, and the signature (i.e., hidden secrets) and security parameters from the owner to validate the model using DeepAuth. The output of the process can be shared between the three parties involved. Here, we assume that their communication is securely protected. Note that model developer, MLaaS provider, and trusted validator can be managed by a single entity or multiple entities depending on the service provisioning mechanism.

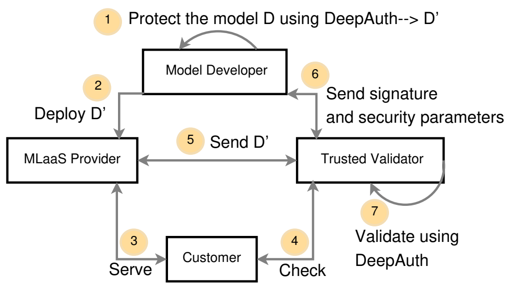

1 Protect the model D using DeepAuth--> D'
Model Developer
2 Deploy D'
6 Send signature and security parameters
5 Send D'
MLaaS Provider
Trusted Validator
7 Validate using DeepAuth
3 Serve
Customer
4 Check

Fig. 3. Example scenario of using our DeepAuth.

DeepAuth Requirements: The design goal of DeepAuth is to balance the security and accuracy while applying steganography to DNN models. We specifically consider the following design requirements. *Confidentiality:* only authorised parties are allowed to hold the secret keys and can retrieve the hidden signature. *Integrity:* any alteration of a protected DNN model (by poisoning or retraining) can be detected. *Authenticity:* the origin and source of the model can be retrieved and verified. *Distortion resistance:* there is no distortion significantly affecting the performance of the original model so that the model with the hidden signature can be directly used for prediction without removing it. *Capacity:* a large amount of information can be secretly embedded without impacting other requirements so that a reasonable size of signature can be embedded. *Accuracy:* the prediction accuracy of the model should not be significantly degraded due to the embedded signature.

Algorithm 1 Embedding Steganography

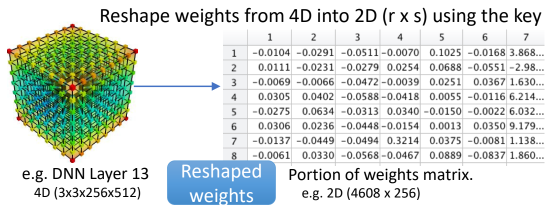
```
Input: DNN model
Output: Protected DNN model
$l_L$: last layer of the model
$\kappa, \nu$: encryption key and scramble vector
1  $\kappa, \nu \leftarrow$ Generate_secret(seed)
2  for $i \leftarrow 1$ to $l_L$ do
3      $r \times s \leftarrow$ Reshape($l_i$,$\kappa$) // from 4D to 2D
4      $M \times N \leftarrow$ Wavelet_convert($r \times s$)
5      $c \leftarrow$ Prepare_signature($l_i$)
6      $\tilde{c} \leftarrow$ Encrypt_signature($c$)
7      $\tilde{M} \times \tilde{N} \leftarrow$ Generate_scramble($\nu$, $M \times N$)
8      $\delta, \varrho \leftarrow$ Derive($M \times N$)
9      $M'' \times N'' \leftarrow$ Scale($M \times N$,$\delta$, $\varrho$)
10     $M'' \times N'' \leftarrow$ Hide($M'' \times N''$, $\tilde{M} \times \tilde{N}$, $\tilde{c}$ )
11     $M \times N \leftarrow$ Rescale($M'' \times N''$, $\delta$ , $\varrho$)
12     $r \times s \leftarrow$ Wavelet_inverse($M \times N$)
13     $l_i \leftarrow$ Shape($r \times s$,$\kappa$) // from 2D to 4D
end
```

DeepAuth Algorithm: Based on the above-stated design requirements, we propose DeepAuth, a wavelet-based steganography algorithm. The process of applying steganography to DNN models is summarized in Algorithm 1.

The algorithm works as follows. We first reshape the hidden layer *weights* (Section III-A) to be ready for wavelet transform as depicted in line 3 (**Reshape**). We then convert the *weights* from the spatial domain to the frequency domain using the wavelet transform (Section III-B) as shown in line 4 (**Wavelet_convert**). We next prepare a unique signature for each hidden layer and encrypt it using a key (Section III-C1) as appears in lines 5-6 (**Prepare/Encrypt**). We then generate a random matrix to ensure randomization of the hiding process (Section III-C2) as shown in line 7 (**Generate_scramble**). We scale the resultant coefficients (Section III-D) before hiding to preserve the sign and decimal accuracy as appears in lines 8-9 (**Derive/Scale**). We then start hiding the encrypted signature bit-by-bit in the less significant coefficients following the random matrix (Section III-E) as shown in line 10 (**Hide**). Functions in lines 11-13 are basically to rescale the coefficients, convert them back from the frequency domain to the spatial domain and finally shape the *weights* back to their 4D form (Section III-F).

A. Weights Reshape

To ensure the *weights* in each DNN hidden layer l_i are pre-processed for wavelet transform, they have been reshaped from 4D ($a \times b \times c \times d$) to 2D ($r \times s$) form. Fig. 4 shows an example of reshaping the weights in a DNN hidden layer by DeepAuth.


Reshape weights from 4D into 2D (r x s) using the key
e.g. DNN Layer 13
4D (3x3x256x512)
Reshaped weights
Portion of weights matrix.
e.g. 2D (4608 x 256)

	1	2	3	4	5	6	7
1	-0.0104	-0.0291	-0.0511	-0.0070	0.1025	-0.0168	3.868...
2	0.0111	-0.0231	-0.0279	0.0254	0.0688	-0.0551	-2.98...
3	-0.0069	-0.0066	-0.0472	-0.0039	0.0251	0.0367	1.630...
4	0.0305	0.0402	-0.0588	-0.0418	0.0055	-0.0116	6.214...
5	-0.0275	0.0634	-0.0313	0.0340	-0.0150	-0.0022	6.032...
6	0.0306	0.0236	-0.0448	-0.0154	0.0013	0.0350	9.179...
7	-0.0137	-0.0449	-0.0494	0.3214	0.0375	-0.0081	1.138...
8	-0.0061	0.0330	-0.0568	-0.0467	0.0889	-0.0837	1.860...

Fig. 4. Example of how DNN weights reshaped from 4D into 2D.

B. Converting to Frequency Domain

Hiding the signature directly into hidden layer weights may yield high distortion, leading to the degrade of the model accuracy. To solve this challenge, DWT is employed to convert the weights from their spatial domain into the frequency domain so that the most significant coefficients are preserved to rebuild the weights after hiding. Fig. 5 shows a demonstration of (a) the original block of weights, (b) converting the weights into frequency domain using DWT, (c) wiping out all detailed sub-bands as zeros (50% of all) while maintaining the approximation sub-bands, and (d) converting back to spatial domain for rebuilding the original *weights* from the approximation sub-bands alone. The observations of the results from these steps motivate us to use the signal processing techniques to increase the capacity of hiding the signature with little distortion effect into the original model.

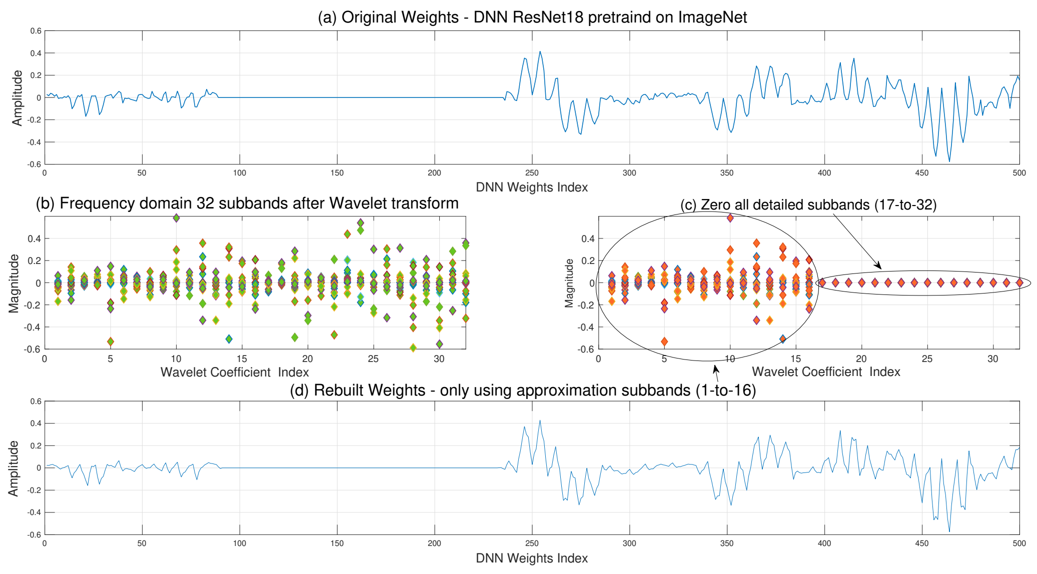


Fig. 5. (a) A plot of DNN ResNet18 model weights from one layer; (b) frequency domain sub-bands plot after applying wavelet transform; (c) after wiping out (zeros) in all detailed sub-bands (i.e., 50% of all); and (d) a rebuilt version for DNN ResNet18 weights from only approximation sub-bands.

More importantly, manipulation of any DNN *weights* results in failure to reconstruct the exact wavelet coefficients of the original DNN *weights*. Hence, we do not need to hide a signature in all weights to protect the integrity of the DNN model since the embedding process is performed in the frequency domain. Any change in actual weights in the spatial domain yields different frequency decomposition tree (see the results in Section V).

In our approach, we apply 5 levels of wavelet packet decomposition to each layer of a DNN model (e.g., ResNet18) which results in 32 sub-bands as shown in Fig. 20 (see Appendix VIII-D). A wavelet family, called Daubechies with the order 2 ($db2$), is chosen in the transformation process because its performance in analysing discontinuous-disturbance-dynamic signals has already been proven in [37]. To achieve the lowest distortion on each layer, the low approximation sub-bands (i.e., from 1 to 16) are not changed because they represent the most significant features to rebuild the DNN layer's *weights*. On the other hand, several bits are manipulated in the rest of the detailed sub-bands to embed signature bits; the number of bits that can be embedded is called the steganography level. To guarantee the lowest acceptable distortion on the actual DNN model layers, several experiments were performed to select an appropriate steganography level (i.e., how many bits can be embedded into the least significant sub-bands) and the number of *weights* per transform (see Fig. 6). As shown in Fig. 6, about two bits can be hidden in the randomly-selected high frequency sub-band coefficients. The number of *weights* per transform should be between 2000 and 12000. Further details are provided in Section V. Note that our benchmark for the acceptable distortion is to maintain the accuracy of the original model.

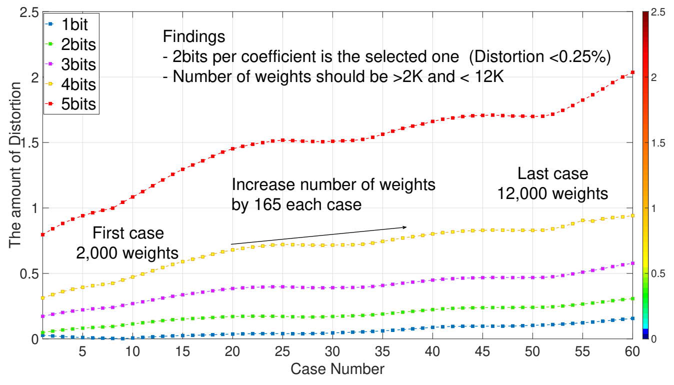


Fig. 6. Resultant distortion impacting DNN hidden layers from various steganography levels and the number of *weights* per transform.

Converting the *weights* into their frequency domain before hiding the signature assists us to (i) reduce the distortion by keeping significant coefficients intact, (ii) expand the capacity by using all detailed sub-bands for embedding, and (iii) ensure the integrity by detecting any change due to the sensitivity of the transformation.

C. Steganography Confidentiality Protection

Steganography alone cannot ensure the confidentiality of the signature because the hiding process is performed at fixed positions that can be exposed to attackers. To solve this challenge, a 256-bit security key κ and a scramble vector $\nu_{i\in[1,256]}$ pre-filled with random values are produced for every DNN model. These two parameters are known only to the legitimate validators, and used to enforce 2 levels of security as follows.

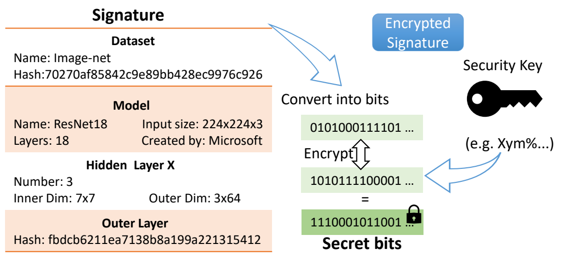


Fig. 7. Example of how sensitive information is obfuscated before hiding.

1) Encrypt Signature: The signature is a collection of sensitive structural information of the model (e.g., dataset hash, model input size, layer dimension, total weights, outer layer hash). κ is used to obfuscate the signature before the hiding process using a symmetric cryptography algorithm. Without loss of generality, we use AES here for encryption. (see Fig. 7). This is shown in Eq. 5,

$$\widetilde{c} \Leftarrow \xi(\kappa, c) \tag{5}$$

where ξ is an AES algorithm, κ is the key; c is the original signature; and $\widetilde{c}$ is the encrypted signature.

2) Generate Scramble: To achieve the goal of embedding the signature in a fully random way and differently among DNN layers, the scramble vector ν (i.e., filled with random values) is employed. It creates a random sequence of coefficients in a form of $2D$ matrix that will be followed to embed the signature. This is shown in Eq. 6,

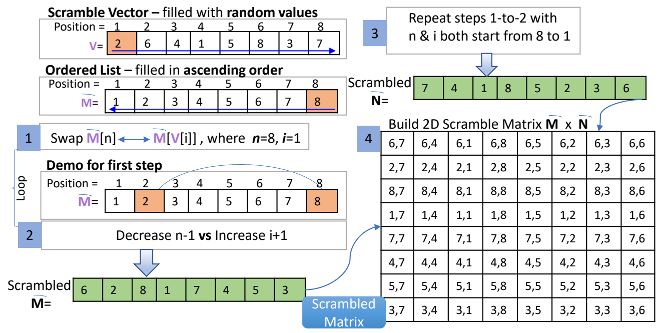
$$Z \Leftarrow \begin{cases} \tilde{M} = f_x(\nu) \\ \tilde{N} = {f_x}'(\nu) \end{cases} \tag{6}$$

where $\tilde{M}$ and $\tilde{N}$ are the generated sequence of numbers; f_x and ${f_x}'$ are the scrambling functions. The combination of $\tilde{M}$ and $\tilde{N}$ is used to build a 2D $\tilde{M} \times \tilde{N}$ matrix Z (see Eq. 7).

$$Z\{\tilde{M}, \tilde{N}\} = \begin{bmatrix} \tilde{m}_1, \tilde{n}_1 & \tilde{m}_1, \tilde{n}_2 & \cdots & \tilde{m}_1, \tilde{n}_n \\ \tilde{m}_2, \tilde{n}_1 & \tilde{m}_2, \tilde{n}_2 & \cdots & \tilde{m}_2, \tilde{n}_n \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{m}_{\tilde{M}}, \tilde{n}_1 & \tilde{m}_{\tilde{M}}, \tilde{n}_2 & \cdots & \tilde{m}_{\tilde{M}}, \tilde{n}_{\tilde{N}} \end{bmatrix} \tag{7}$$

Fig. 8 shows a demonstration of how f_x uses $\nu \in [1, n]$ to produce a random hiding sequence. An ordered list $\tilde{M}$ of the size of DWT coefficients M is created. The scrambling process starts with $\tilde{M}$ following the random values in ν. This is to prevent two identical sequences being generated. Another carefully considered condition is that using mathematical modulo over the random values of ν guarantees us to obtain finite field numbers in the range of $[1, M]$. Let us do the first scramble: steps 1 and 2. The pointer starts from the last cell of the ordered list $\tilde{M}$ which has the value of '8'. It is then swapped with another cell in $\tilde{M}$ (i.e., the position of that cell is the first random number in ν which is '2'). Then, the pointer over $\tilde{M}$ is reduced by 1 and the pointer over ν is increased by 1. Eventually the loop outputs a scrambled vector $\tilde{M}$. By only reversing the pointer over ν, ${f_x}'$ follows almost identical steps to generate $\tilde{N}$. Finally, $\tilde{M}$ and $\tilde{N}$ are combined to compose the 2D matrix sequence. However, the ν is much longer (i.e., ≥ 256) in the proposed model.

The above two levels of security guarantee the strong confidentiality of the hidden signature. This is achieved by ensuring that only an authorized validator has κ and ν to recover and decrypt the signature.



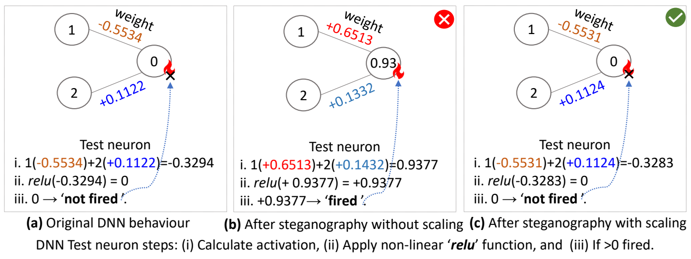
Scramble Vector – filled with random values
Position = 1 2 3 4 5 6 7 8
V= 2 6 4 1 5 8 3 7
Ordered List – filled in ascending order
Position = 1 2 3 4 5 6 7 8
M= 1 2 3 4 5 6 7 8
1
Swap M[n] ⟷ M[V[i]] , where n=8, i=1
Loop
Demo for first step
Position = 1 2 3 4 5 6 7 8
M= 1 2 3 4 5 6 7 8
2
Decrease n-1 vs Increase i+1
Scrambled M= 6 2 8 1 7 4 5 3
Scrambled Matrix
3
Repeat steps 1-to-2 with n & i both start from 8 to 1
Scrambled N= 7 4 1 8 5 2 3 6
4
Build 2D Scramble Matrix M x N

6,7	6,4	6,1	6,8	6,5	6,2	6,3	6,6
2,7	2,4	2,1	2,8	2,5	2,2	2,3	2,6
8,7	8,4	8,1	8,8	8,5	8,2	8,3	8,6
1,7	1,4	1,1	1,8	1,5	1,2	1,3	1,6
7,7	7,4	7,1	7,8	7,5	7,2	7,3	7,6
4,7	4,4	4,1	4,8	4,5	4,2	4,3	4,6
5,7	5,4	5,1	5,8	5,5	5,2	5,3	5,6
3,7	3,4	3,1	3,8	3,5	3,2	3,3	3,6

Fig. 8. Demonstration of generating a 2D scrambling matrix order $\tilde{M} \times \tilde{N}$ from a scramble vector ν.

D. Scale Coefficients

To protect the accuracy of neurons at the hidden layers and preserve the sign of *weights*, two factors are derived after analysing millions of *weights*. The first factor δ is used to ensure that all values are positive (e.g., the lowest value $+(-1)$). The second factor ϱ is used to maintain all four decimal values (e.g., $\times$ 10000) (see the impact in Fig. 9). δ and ϱ are used to scale the coefficients before the hiding process so that the behaviour of neurons in the networks are preserved.



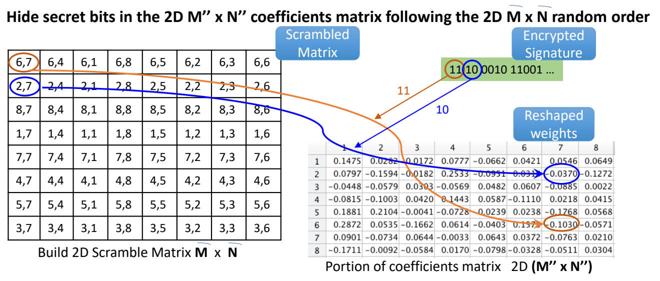
1
2
0
weight
-0.5534
+0.1122
Test neuron
i. 1(-0.5534)+2(+0.1122)=-0.3294
ii. relu(-0.3294) = 0
iii. 0 → 'not fired'.
1
2
0.93
weight
+0.6513
+0.1332
Test neuron
i. 1(+0.6513)+2(+0.1432)=0.9377
ii. relu(+ 0.9377) = +0.9377
iii. +0.9377→ 'fired'.
1
2
0
weight
-0.5531
+0.1124
Test neuron
i. 1(-0.5531)+2(+0.1124)=-0.3283
ii. relu(-0.3283) = 0
iii. 0 → 'not fired'.
(a) Original DNN behaviour
(b) After steganography without scaling
(c) After steganography with scaling
DNN Test neuron steps: (i) Calculate activation, (ii) Apply non-linear 'relu' function, and (iii) If >0 fired.

Fig. 9. Example shows the impact of applying the derived δ and ϱ before hiding.

E. Hiding

The hiding process is summarized in Fig. 10. Recall the steps explained in Figs. 4, 7, and 8. The signature's bits are hidden bit-by-bit in the scaled coefficients $M'' \times N''$ corresponding to $\tilde{M} \times \tilde{N}$ generated in the random order.

To provide the model accuracy comparable with the baseline prediction accuracy in the original model, we hide only 2-bits for each coefficient.


Hide secret bits in the 2D M'' x N'' coefficients matrix following the 2D M x N random order
Scrambled Matrix
Encrypted Signature
11 10 0010 11001 ...
11
10
Reshaped weights
Build 2D Scramble Matrix M x N
Portion of coefficients matrix 2D (M'' x N'')

6,7	6,4	6,1	6,8	6,5	6,2	6,3	6,6
2,7	2,4	2,1	2,8	2,5	2,2	2,3	2,6
8,7	8,4	8,1	8,8	8,5	8,2	8,3	8,6
1,7	1,4	1,1	1,8	1,5	1,2	1,3	1,6
7,7	7,4	7,1	7,8	7,5	7,2	7,3	7,6
4,7	4,4	4,1	4,8	4,5	4,2	4,3	4,6
5,7	5,4	5,1	5,8	5,5	5,2	5,3	5,6
3,7	3,4	3,1	3,8	3,5	3,2	3,3	3,6

	1	2	3	4	5	6	7	8
1	0.1475	0.0282	0.0172	0.0777	−0.0662	0.0421	0.0546	0.0649
2	0.0797	−0.1594	−0.0182	0.2533	−0.0951	[illegible]	−0.0370	−0.1272
3	−0.0448	−0.0579	0.0303	−0.0569	0.0482	0.0607	−0.0885	0.0022
4	−0.0815	−0.1003	0.0420	0.1443	0.0587	−0.1110	0.0218	0.0415
5	0.1881	0.2104	−0.0041	−0.0728	0.0239	0.0238	−0.1768	0.0568
6	0.2872	0.0535	−0.1662	0.0614	−0.0403	[illegible]	−0.1030	−0.0571
7	0.0901	−0.0734	0.0644	−0.0033	0.0643	0.0372	−0.0763	0.0210
8	−0.1711	−0.0092	−0.0584	0.0170	−0.0798	−0.0328	−0.0511	0.0304

Fig. 10. Block diagram summarizes the hiding process and uses the information explained in Figs. 4, 7, and 8.

F. Inverse Wavelet to Rebuild Layer

The resultant detailed coefficients after the hiding process are called *stego* coefficients that should be protected. At this stage, the stego coefficients are rescaled and re-embedded back into the 32 sub-bands coefficients matrix before applying the inverse DWT to convert *weights* from their frequency domain to their original space domain. The result of the reconstructed weights is called stego *weights* (containing the hidden signature), which are almost similar to the original *weights*. The advantage of this approach is that the stego *weights* can be used for the prediction. However, only authorized validator (i.e., with κ and ν) can extract the signature and verify it. The inverse DWT is defined by Eq. 8,

$$X = \sum_{a} \sum_{b} Y(a,b)\Phi_{ab}(n) \tag{8}$$

where X is the *weights* in their original time domain.
Finally, the *weights* are reshaped back from the 2D into their original 4D shape before integrating them into the DNN layer l_i.

G. All Hidden Layers Protection

The hiding steps explained in Algorithm 1 are repeated for each hidden layer $l_{i \in [1,L]}$. The steps of generating the scramble matrix $\tilde{M} \times \tilde{N}$ are repeated for each hidden layer. The only difference between the layers is that we shift the index i over ν by the hidden layer position h. Hence, we can generate a unique scrambling matrix for each layer.

H. Retrieval and Validation

To accurately extract and decrypt the signature, the validator must have the security key κ, the scramble vector ν, the original signature, and the protected model $\tilde{D}$. The process is nearly similar to the hiding steps, but the signature bits are recovered rather than embedded. Fig. 11 demonstrates the required steps. First, *weights* at each layer l_i are fetched and shaped using κ before applying DWT. Then, the detailed coefficients are selected and scaled. Next, the random hiding order is generated using ν and followed to retrieve the signature bits. Finally, κ is used to decrypt the signature bits and verify the retrieved signature from each hidden layer. Thus, a slight change even in one layer can be detected and highlighted. The outer layer is also checked by comparing its current hash value with the hidden hash value to ensure the correctness of output classes at all times.

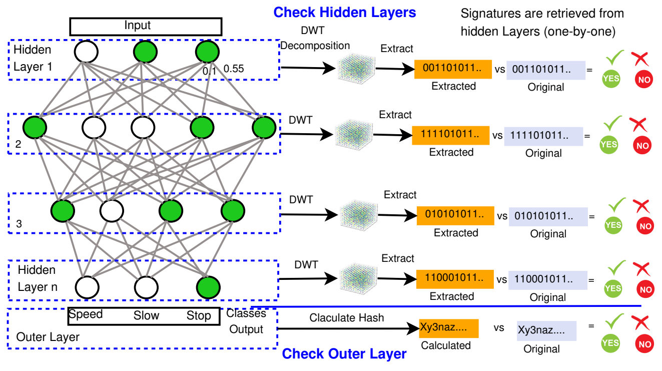


Fig. 11. Block diagram shows the retrieval and validation process.

IV. Theoretical Analysis for DeepAuth

In this section, we show that DeepAuth satisfies the design requirements stated in Section III through our theoretical analysis results.

A. Confidentiality Strength

Confidentiality is achieved with two security parameters: a security key κ and a scramble vector ν. Both parameters are needed along with the model to recover the hidden signature. Moreover, an attacker has to be aware of the existence of the hidden signature before performing the brute force attack to retrieve it, which makes it extremely difficult to do.

For the security of DeepAuth, it is critical to keep κ and ν confidential because they are required to (recall Figs. 4, 7, and 8): (1) reshape *weights* from 4D into 2D form, (2) obfuscate the signature, and (3) create a random coefficients' sequence as $2D\ \tilde{M} \times \tilde{N}$ matrix to embed the bits uniquely in each layer. We expect two involved parties (e.g., the data owner and the validator) should securely protect these parameters.

κ can be quantified as the entropy bits' number $\triangle H$ (see Eq. 9) where $2^{\triangle H}$ is the supreme possibilities that would need to be examined by the intruder during a brute-force attack.

$$\triangle H = log_2 W^G \tag{9}$$

where G is the length of the total symbols and W is the set of symbols. Assume $\triangle H \geq 256$. Then, the intruder has to search $2^{256} \cong 1.115792 \times 10^{76}$ to find κ, which yields a 256-bit level strength.

υ is a randomisation vector as shown in Eq. 10,

$$\nu = [\gamma_1, \gamma_2, .., \gamma_n] \tag{10}$$

where n is the number of total entries; and $\gamma \in \mathbb{R}$ is a unique random number generated by a strong PRNG using a large seed $\vec{s}$ to provide sufficiently secure randomness. As ν is designed to be unique and independent from κ, the attacker has to spend $\mathcal{O}(n!)$ time to find ν at the worst case. Assume ν has $n = 256$ unique entries; thus there are $256!$ possible vectors. This search space is approximately $\cong 8.5781 \times 10^{506} \cong$ 256-bit level strength which is computationally infeasible to enumerate.

B. Integrity and Authenticity

To ensure the integrity of the DNN model, all hidden layers $l_{i\in[1,L]}$ should be protected in their frequency domain. Any change of l_i *weights* in their spatial-domain yields different DWT as explained in Section III-B. The outer layer is also strongly protected by hashing its values and embedding it into each l_i. To guarantee the authenticity and prevent retrieving the hidden information without κ and ν, the 32 sub-bands coefficients matrix after DWT decomposition of *weights* should have a suitable size (e.g., ≥ 4000) as shown in Eq. 11,

$$T = \sum_{i=1}^{r} R! \times \sum_{j=t}^{c} C! \tag{11}$$

where T is the total number of possibilities; R and C are the rows and columns, respectively, of the 32 sub-bands coefficients matrix; and t is the selected detailed coefficients that can be used from each row.

Assume 4096 *weights* from only one layer l_3, and their 32 sub-bands coefficients are in the size of 128×32 after applying DWT. If we assume that the threshold is 16 (i.e., when $t = 16$ in Eq. 11), T can be calculated as shown in Eq. 12,

$$T = \sum_{i=1}^{128} 128! \times \sum_{j=16}^{32} 32! \Rightarrow T \cong 8.068256 \times 10^{194} \tag{12}$$

Consequently, we can see that it is computationally infeasible to break the authenticity of the model.

C. Capacity

The maximum amount of embedded secret bits (for the signature) in each hidden layer l_i mainly depends on two factors: (1) the number of *weights* and (2) the hidden bits per *weight*. However, if we hide even 1 bit in *weights* directly, the distortion due to the hiding process would be high. This is one of the main reasons to transform *weights* from their space to the frequency domain. Based on that, another dimension becomes very important to the hiding capacity which is the coefficients' selection process. In the proposed algorithm, the maximum number of bits that can be hidden in a set of *weights* to maintain the lowest distortion impact is shown in Eq. 13,

$$b = \sum_{i=1}^{t} \frac{n}{2} \times B \tag{13}$$

where b is the total hidden bits; t is the detailed sub-bands; n is the number of total coefficients in one sub-band; and B is the steganography level (i.e., number of hidden bits in each sub-band's coefficient).

For better understanding, assume that a five-level DWT decomposition is applied to a set of *weights* from layer l_i where the number of resultant sub-band coefficients at each layer is $n = 512$. Therefore, the $2D$ detailed sub-bands are in the size of 64×16 (Here, 16 is the value of t in Eq. 11). Also, assume that $B = 2$ bits are hidden in each coefficient. Therefore, around 2048 bits of sensitive data can be hidden inside that set of *weights*. In practice, the total number of *weights* varies among layers. In this paper, the hidden information inside each layer l_i is fixed to 8192 bits (i.e., 1 KB). Hence, DeepAuth is able to offer a reasonable capacity to embed a large number of bits (for the signature).

V. Evaluation

This section presents the evaluation results of DeepAuth and shows its effectiveness against various attacks.

A. Datasets

In our evaluation, we have used three well-known datasets. First, a labelled dataset of handwritten digits (0-9), named "MNIST", which is a subset of a larger NIST database. It contains 60,000 training images and 10,000 testing samples [26]. It has been size-normalized in collaborative efforts between Microsoft Research, Google Labs, and New York University. Second, a dataset of 10 classes (i.e., airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks), named "CIFAR-10", which is a subset from 80 million tiny images repository collected by groups at MIT and New York University. CIFAR-10 is labelled by The Canadian Institute for Advanced Research and contains 50,000 training images and 10,000 testing samples [27]. Third, ImageNet public dataset, which includes 1.2 million images with 1,000 objects [28].

We use ResNet18 model for our evaluation [8]. ResNet18 is a pre-trained special type of DNN, called Convolutional Neural Networks, which have shown to be remarkably successful in a range of computer vision and pattern recognition tasks [38]. Its architecture has 18 layers deep and been pre-trained on ImageNet. To evaluate that DeepAuth is also applicable to other models, we repeat the experiments on two other older architectures pre-trained on ImageNet: AlexNet [10] (8 layers) and VGG16 [9] (16 layers).

B. Evaluation Metrics

This section presents the evaluation metrics used. Three widely-used metrics have been used to closely monitor the distortion effect on the protected model, the classification accuracy, and the retrieved hidden signature after the attacks.

1) Distortion: To precisely evaluate our technique's impact on DNN model, the margin between the original D and protected models $\tilde{D}$ has been thoroughly monitored using the percent of root-mean-square difference (PRD). The PRD is a widely-used measurement that is known for its precision of detecting any recomposition error between the original and the recomposed signals as defined in Eq. 14 [39],

$$PRD = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \tilde{x}_i)^2}{\sum_{i=1}^{n}(x_i^2)}} \times 100\% \tag{14}$$

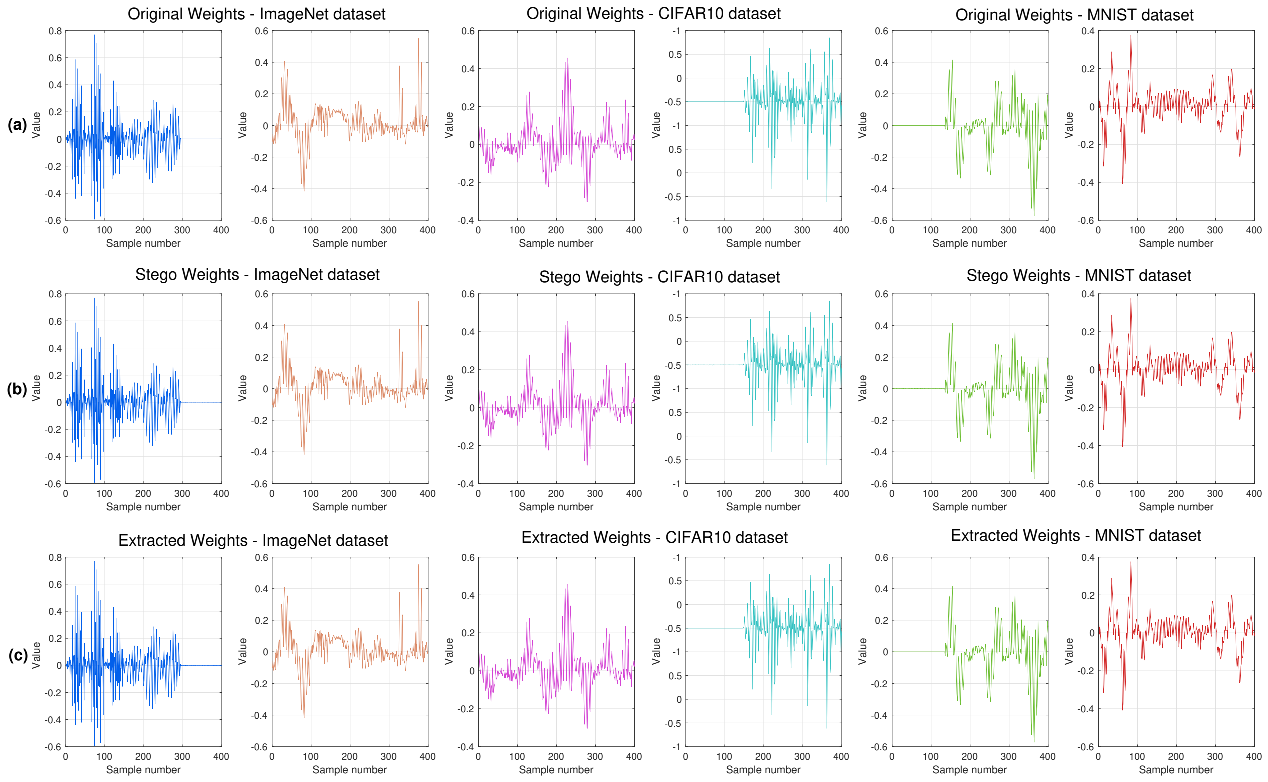


Fig. 12. Six examples of the weights of DNN hidden layers: (a) the original form; (b) stego form (i.e., containing hidden signature) that obtained after applying our DeepAuth, and (c) Extracted form after removing the hidden signature.

where $x(i)$ and $\tilde{x}(i)$ are the original and recomposed *weights*, and n is the size of *weights* from each layer.

The PRD is also employed to accurately calculate the resultant effects caused by the recovery process (i.e., between the original and the extracted *weights*).

2) Accuracy: The performance accuracy of the protected model $\tilde{D}$ with respect to the original D needs to be carefully measured to understand the impact of our technique on the accuracy of the DNN models. We use the score for the degradation as shown in Eq. 15,

$$f_{\chi}(\vec{\tau}, D) \Rightarrow S \cong \tilde{S} \Leftarrow f_{\chi}(\vec{\tau}, \tilde{D}) \tag{15}$$

where f_{χ} is a classification function, $\vec{\tau}$ is the testing subset, and S and $\tilde{S}$ are the accuracy score before and after protecting the model. We apply Bit Error Ratio (BER) on the signature after every retrieval process to measure any data loss (i.e., even one bit) as follows [40],

$$BER = \frac{B_r}{B_T} \times 100\% \tag{16}$$

where B_r is the number of erroneous bits, and B_T is the number of bits.

C. Experiments

Steps Summary: Our experiment steps can be summarised as follows: (1) Train ResNet18 architecture with both MNIST and CIFAR-10 training datasets; we name the resultant models D_1 and D_2. (2) Test the classification baseline accuracy of both D_1 and D_2 using MNIST and CIFAR-10 testing datasets. (3) Apply our steganographic technique "DeepAuth" to both D_1 and D_2; we name the obtained models $\tilde{D}_1$ and $\tilde{D}_2$. (4) Test the classification accuracy of both $\tilde{D}_1$ and $\tilde{D}_2$ using MNIST and CIFAR-10 testing datasets. (5) Verify the accuracy of the retrieved information after the extraction process using BER. (6) Test the *weights* difference before and after DeepAuth using PRD. (7) Attack $\tilde{D}_1$ and $\tilde{D}_2$ using various adversarial mechanisms and evaluate the integrity of the hidden bits. (8) We repeat these steps on the ResNet18 itself as it has been pre-trained on ImageNet, which is a large scale visual recognition dataset of 1.2 million images. (9) We experiment DeepAuth on two other widely-used architectures (i.e., AlexNet and VGG16) to ensure its effectiveness. (10) We finally perform an out of the box experiment to validate the accuracy impact using randomly picked samples from the Internet.

To obtain the neutral and unbiased results, all experiments performed using $\kappa = 256$ and $\nu = 256$. *weights* size per transform varies between $\geq 4000 \sim \leq 12000$ and a maximum of 2-bits are hidden in every selected detailed coefficient (see Fig. 6). The total of 8,192 bits (i.e., around 1KB) is embedded in each hidden layer. All deep layers have been protected with our technique. ResNet18 architecture has been used with three datasets as it is newer than AlexNet and VGG16. We optimised all models using Stochastic Gradient Descent (SGD), an initial learning rate of 1e-4, 10 epochs, *weights* learn rate factor of 10, *bias* learn rate factor of 10 and batch size of 100.

Accuracy: Table I shows the classification accuracy of the 3 trained models obtained from the 3 datasets before and after applying our DeepAuth technique with different steganography levels. The experiments match with our early observations shown in Fig. 6. Once we exceed the steganography level of 2-bits per coefficient, the accuracy starts to be affected. We observe that changing 2+ bits in the frequency domain coefficients results in flipping the first decimal values of the rebuilt *weights*. Such flip impacts on the set of neurons to be activated. On the contrary, the steganography level of 2 bits affects only the fourth decimal values and rarely the third decimals of the detail coefficients. Hence, it has little effect on the two significant factors (i.e., the sign and first decimal) of the rebuilt *weights*. BER is 0% in all cases which means the signatures are verifiable with no errors. We further evaluate the accuracy of VGG16 and AlexNet before/after applying the best steganography level of 2-bits and present the results in Table V (see Appendix VIII-A). The obtained accuracy results are exactly the same. BER is 0% in both cases.

TABLE I
COMPARISON BETWEEN BASELINE CLASSIFICATION ACCURACY (%) BEFORE AND AFTER APPLYING OUR DEEPAUTH WITH DIFFERENT STEGANOGRAPHY LEVELS.

		DeepAuth - Steganography levels				
Dataset	**Baseline**	1-bit	**2-bits**	3-bits	4-bits	BER
MNIST	**99.00**	99.00	**99.00**	98.97	98.97	0
CIFAR-10	**88.40**	88.40	**88.40**	88.38	88.38	0
ImageNet	**83.71**	83.71	**83.71**	83.67	83.67	0

We note that the reported baseline accuracy from ImageNet is below what is reported in ResNet18, VGG16 and AlexNet. The reason behind this is due to the testing on a smaller subset as ImageNet is an extensive dataset and it is computationally very expensive to perform retraining and testing on the entire dataset. Most of the literature (e.g., [14], [29], [41], [42]) re-trained these architectures on smaller datasets such as MNIST and CIFAR-10 for experimenting their models. However, we were very interested in going one step further by trying our algorithm on the various original architectures already trained on a large scale dataset.

Distortion: Table II presents the PRD distortion measurements between the actual layers and the stego form (i.e., after applying our DeepAuth 2-bits) as well as between the original and extracted form (i.e., after removing our DeepAuth). All PRDs are very low at $\leq 0.25\%$ with an average of $\leq 0.20\%$. In the signal processing context, PRDs $\leq 1\%$ is regarded as a stable with little impact. As a proof, Ibaida et al. [43] achieved an average of 0.60%. This also has been proven (see Table I) by achieving the exact model accuracy before and after applying our DeepAuth. We further evaluate the distortion impact on VGG16 and AlexNet before/after applying the best steganography level of 2-bits and present the results in Table VI (shown in Appendix VIII-B). All PRDs are very low and matching the early findings.

TABLE II
PRD RESULTS FOR THE 3 MODELS USING DIFFERENT DATASETS

	MNIST		CIFAR-10		ImageNet	
Layer No	PRD % Stego	PRD % Extract	PRD % Stego	PRD % Extract	PRD % Stego	PRD % Extract
1	0.095	0.103	0.096	0.102	0.097	0.100
2	0.101	0.114	0.099	0.113	0.100	0.112
3	0.094	0.101	0.092	0.102	0.093	0.100
4	0.086	0.094	0.087	0.096	0.088	0.095
5	0.189	0.209	0.190	0.212	0.191	0.208
6	0.204	0.228	0.211	0.228	0.206	0.224
7	0.229	0.250	0.221	0.247	0.223	0.252
8	0.233	0.253	0.229	0.231	0.232	0.257
9	0.220	0.227	0.233	0.237	0.242	0.253
10	0.252	0.284	0.250	0.253	0.256	0.255
11	0.250	0.252	0.253	0.259	0.248	0.257
12	0.249	0.254	0.249	0.242	0.255	0.252
13	0.217	0.237	0.215	0.236	0.215	0.235
14	0.215	0.244	0.220	0.233	0.216	0.238
15	0.243	0.253	0.245	0.252	0.240	0.243
16	0.235	0.245	0.238	0.247	0.240	0.249
17	0.243	0.247	0.242	0.249	0.250	0.259
18	0.244	0.245	0.250	0.240	0.259	0.259
Avg	0.20	0.213	0.202	0.213	0.204	0.218

Fig. 12 shows example *weights* from various layers of CIFAR-10, MNIST and ImageNet using the ResNet18 architecture. The plots demonstrate a visual comparison between (a) the original *weights*, (b) the stego form, and (c) after the extraction process. Fig. 19 (see Appendix VIII-E) also presents *weights* samples from AlexNet and VGG16, respectively. From all cases, it is clear that DeepAuth has no noticeable impact.

We finally perform an out of the box experiment (see Fig. 18 in Appendix VIII-C) where we pick random input samples from the Internet and observe the classification accuracy of the original baseline DNN models vs protected version using DeepAuth. We use three architectures i.e., ResNet18, VGG16, and AlexNet. The obtained accuracy results from all models are exactly the same before and after applying DeepAuth.

All results prove that our DeepAuth with steganography level of 2-bits is stable and has little influence on the hidden layers. In other words, it does not degrade the accuracy of the stego model when compared to the original model.

D. Attack Scenarios

Recall that one of our goals of developing DeepAuth in the frequency domain is to ensure that any slight change in the model should be detected even if the attacked models have the same accuracy. To test the stated goal, we test the stego models against the state-of-the-art attacks. We choose three popular manipulation attacks: (1) input poisoning, (2) output poisoning, and (3) fine-tuning. These attacks are difficult to detect as they keep all DNN architecture features, such as hidden layers, output classes, number of weights, and the classification accuracy, intact.

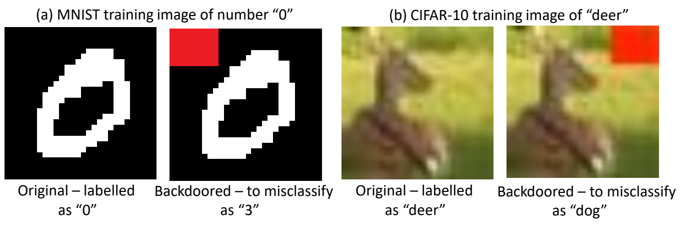

(a) MNIST training image of number "0"
(b) CIFAR-10 training image of "deer"
Original – labelled as "0"
Backdoored – to misclassify as "3"
Original – labelled as "deer"
Backdoored – to misclassify as "dog"

Fig. 13. Two random images from MNIST and CIFAR-10 before and after inserting the backdoors (i.e., red sticky note).

1) Input Poisoning: It is a backdooring attack on DNN to misclassify only when it is shown a poisoned sample (recall Fig. 1) [14]. It is an attack in which the attacker retrains the model by using a modified sample and corresponding label (not reflecting the ground truth). More precisely, the adversary takes model D with its original training dataset D_{train}. The attacker then picks an image g from D_{train}, inserts a backdoor on it (e.g., simple sticky note) to be g^{adv}, and reinjects g^{adv} with a different label to D_{train} to be D^{adv}_{train}. The attacker finally retrains D with the poisoned dataset D^{adv}_{train} to obtain poisoned model D^{adv}. The severe danger of D^{adv} is that it (a) does not degrade the baseline accuracy, and (b) is difficult to detect unless the trigger is known.

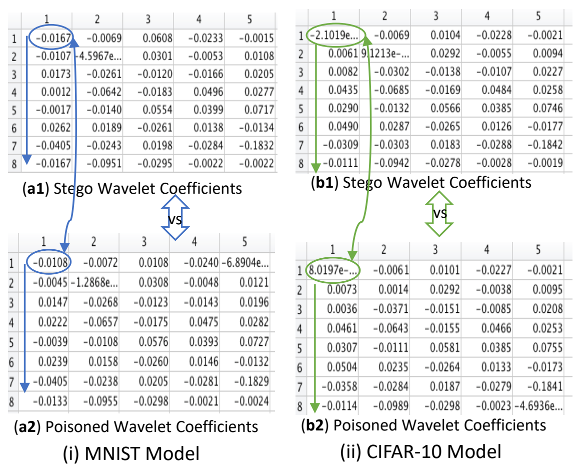
We implement this attack on MNIST and CIFAR-10 protected models by generating $\tilde{D_1}$ and $\tilde{D_2}$. As shown in Fig. 13, we randomly pick g_1 from MNIST as number '0', insert a backdoor, and reinject it as number '3'. In other words, $\tilde{D_1}$ misclassifies number '0' as number '3' only when seeing an image of digit '0' with a backdoor. We also randomly pick g_2 from CIFAR-10 as an image of a 'deer', insert a backdoor, and reinject it as a 'dog'. We retrain the protected $\tilde{D_1}$ and $\tilde{D_2}$ with the poisoned MNIST and CIFAR-10 and obtained $\tilde{D_1^{adv}}$ and $\tilde{D_2^{adv}}$. All training parameters are the same. Finally, the accuracy and integrity of all hidden layers as well as an output layer, BER of $\tilde{D_1^{adv}}$ and $\tilde{D_2^{adv}}$ are examined carefully to answer the question: Can the poisoning attack be detected? The answer is yes, and the results are summarized as follows,

TABLE III
A COMPARISON BETWEEN THE ORIGINAL, THE STEGO, AND THE POISONED MODELS. (-) MEANS BEFORE DEEPAUTH. (✓) MEANS HIDDEN SIGNATURES MATCH CORRECTLY WITH 0% BER. (×) MEANS HIDDEN SIGNATURES MISMATCH WITH ERROR % SHOWN IN BER.

	MNIST			CIFAR-10		
Test	Origin D_1	Stego $\tilde{D_1}$	Poison $\tilde{D_1^{adv}}$	Origin D_2	Stego $\tilde{D_2}$	Poison $\tilde{D_2^{adv}}$
Accuracy	99.0%	99.0%	99.0%	88.4%	88.4%	88.7%
BER	-	0%	71%	-	0%	76%
Hidden Layers Integrity	-	✓	×	-	✓	×
Output Integrity	-	✓	✓	-	✓	✓

(i) Table III shows a comparison between the original, stego and poisoned models. From the accuracy perspective, it is clear that there is no degradation; this means such attacks are hard to detect by observing the accuracy. We even observed a slight accuracy increase in the CIFAR-10 poisoned model due to the retraining process. However, BER of the hidden signatures retrieved from deep layers ($\tilde{D_1^{adv}}$ =71% and $\tilde{D_2^{adv}}$ =76%) shows a major mismatch. We can thus detect the attacks. The integrity of the output layer is still protected as its current hash is the same as the retrieved one. This is not surprising as the input poisoning attack only affects the hidden *weights* to cause misclassification and does not change the output layer.

(ii) Fig. 14 depicts a deep dive into the deep layers in frequency domain of both stego models ($\tilde{D_1}$ and $\tilde{D_2}$) and the poisoned models ($\tilde{D_1^{adv}}$ and $\tilde{D_2^{adv}}$). (a1) vs (a2) are the frequency domain wavelet coefficients (i.e., where the signature's bits are randomly hidden) from MNIST model. (b1) vs (b2) are the same but from CIFAR-10. It is clear that they are widely different, especially in the third and fourth decimal values (i.e., where hidden signatures reside). This proves that input poisoning attacks to even one image results in completely different wavelet coefficients; hence, DeepAuth can detect such attacks.



	1	2	3	4	5
1	-0.0167	-0.0069	0.0608	-0.0233	-0.0015
2	-0.0107	-4.5967e...	0.0301	-0.0053	0.0108
3	0.0173	-0.0261	-0.0120	-0.0166	0.0205
4	0.0012	-0.0642	-0.0183	0.0496	0.0277
5	-0.0017	-0.0140	0.0554	0.0399	0.0717
6	0.0262	0.0189	-0.0261	0.0138	-0.0134
7	-0.0405	-0.0243	0.0198	-0.0284	-0.1832
8	-0.0167	-0.0951	-0.0295	-0.0022	-0.0022

(**a1**) Stego Wavelet Coefficients

vs

	1	2	3	4	5
1	-0.0108	-0.0072	0.0108	-0.0240	-6.8904e...
2	-0.0045	-1.2868e...	0.0308	-0.0048	0.0121
3	0.0147	-0.0268	-0.0123	-0.0143	0.0196
4	0.0222	-0.0657	-0.0175	0.0475	0.0282
5	-0.0039	-0.0108	0.0576	0.0393	0.0727
6	0.0239	0.0158	-0.0260	0.0146	-0.0132
7	-0.0405	-0.0238	0.0205	-0.0281	-0.1829
8	-0.0133	-0.0955	-0.0298	-0.0021	-0.0024

(**a2**) Poisoned Wavelet Coefficients

(i) MNIST Model

	1	2	3	4	5
1	-2.1019e...	-0.0069	0.0104	-0.0228	-0.0021
2	0.0061	9.1213e-...	0.0292	-0.0055	0.0094
3	0.0082	-0.0302	-0.0138	-0.0107	0.0227
4	0.0435	-0.0685	-0.0169	0.0484	0.0258
5	0.0290	-0.0132	0.0566	0.0385	0.0746
6	0.0490	0.0287	-0.0265	0.0126	-0.0177
7	-0.0309	-0.0303	0.0183	-0.0288	-0.1842
8	-0.0111	-0.0942	-0.0278	-0.0028	-0.0019

(**b1**) Stego Wavelet Coefficients

vs

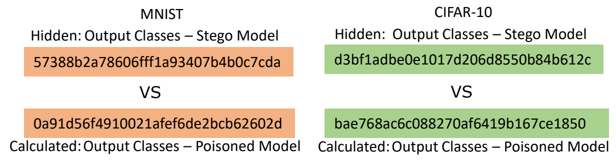
	1	2	3	4	5
1	8.0197e-...	-0.0061	0.0101	-0.0227	-0.0021
2	0.0073	0.0014	0.0292	-0.0038	0.0095
3	0.0036	-0.0371	-0.0151	-0.0085	0.0208
4	0.0461	-0.0643	-0.0155	0.0466	0.0253
5	0.0307	-0.0111	0.0581	0.0385	0.0755
6	0.0504	0.0235	-0.0264	0.0133	-0.0173
7	-0.0358	-0.0284	0.0187	-0.0279	-0.1841
8	-0.0114	-0.0989	-0.0298	-0.0023	-4.6936e...

(**b2**) Poisoned Wavelet Coefficients

(ii) CIFAR-10 Model

Fig. 14. A deep dive into the resultant Wavelet coefficients of the stego (i.e., protected) weights before/after the attack. It shows how different they are after the poison attack which explains how our DeepAuth can detect it.

2) Output Poisoning: In this type of attacks, only the output classes are tampered to cause misclassification [15]. The rest of the model, including deep layers parameters and *weights* are all frozen and untouched; this makes detecting such attacks very challenging. Although this attack can be easily detected in small output classes by looking at the classification output, it is much more difficult in a very large output scenario.



MNIST
Hidden: Output Classes – Stego Model
57388b2a78606fff1a93407b4b0c7cda
VS
0a91d56f4910021afef6de2bcb62602d
Calculated: Output Classes – Poisoned Model

CIFAR-10
Hidden: Output Classes – Stego Model
d3bf1adbe0e1017d206d8550b84b612c
VS
bae768ac6c088270af6419b167ce1850
Calculated: Output Classes – Poisoned Model

Fig. 15. A comparison of the output layer hash between the embedded version in deep layers vs the calculated version from the poisoned output layer.

We implement this attack by only manipulating one class in the output layer of both MNIST and CIFAR-10 protected models. We flip the class '0' with '3' in MNIST and the class 'deer' with 'dog' in CIFAR-10. Although all deep layers are the same, our DeepAuth is able to detect the attack (see Fig. 15) as the output layer hash embedded as part of the signature in each hidden layer is different from the newly calculated one from the current output layer.

3) Fine-tuning: It is another type of attack that an adversary uses to slightly manipulate the model which may degrade or even improve the accuracy [29]. In both scenarios, it is important for the model owner to ensure that the integrity and authenticity of their released model is preserved. To perform this attack, one has to retrain the model using the same dataset with minor parameters tuning such as the learning rate. The resultant model will converge to another local minimum, which might be better, worse, or even the same. Can DeepAuth detect this type of attack?

We implement this attack by only changing one parameter, which is the learning rate from (*1e-4*) to (*1e-3*). The main reason behind choosing learning rate is that we do not want to manipulate many parameters to induce bias and make the attacks easily detectable.

Table IV presents a result comparing the protected model with the fine-tuned model. Despite a slight increase in the accuracy, our DeepAuth technique can detect the attack from both BER and the integrity of the hidden layers. It is important to note that such fine tunes cannot be detected by a simple analysis of hidden layers. As a proof, Fig. 16 shows a histogram comparison between the protected models and fine-tuned ones of both MNIST and CIFAR-10. The density distribution looks almost identical. However, taking a deep dive examination to our DeepAuth frequency domain, as shown in Fig. 17, shows a major change. In other words, the wavelet transform coefficients are different as explained early (see Fig. 14); hence, the attack can be detected using differences in the wavelet transform coefficients.

TABLE IV
A COMPARISON BETWEEN THE STEGO AND THE FINE-TUNED MODELS.

Test	MNIST		CIFAR-10	
	Stego $\tilde{D}_1$	Fine-tuned D_1^{tuned}	Stego $\tilde{D}_2$	Fine-tuned D_2^{tuned}
Accuracy	99.00%	99.47%	88.4%	89.20%
BER	0%	66%	0%	68%
Hidden Layers Integrity	✓	×	✓	×
Output Integrity	✓	✓	✓	✓

E. Limitations

In this section, we want to consider the possible adversarial strategies, in which the attackers know the steganographic algorithm in details and have full access to the protected model.

Reverse Engineering: Our DeepAuth relies on the secrecy of security parameters and not on the secrecy of the algorithm. Hence, an attacker has to break two levels of security (i.e., secret key and scramble vector) as explained in Section IV to find where the signature's bits are hidden. This is to reshape the DNN weights from 4D to 2D form and produce the randomisation matrix. We also ensure the generation of a unique random matrix from the scramble vector for each hidden layer (recall III-G). With the assumption that these two security parameters are kept secret, it is difficult to reverse engineer and find the hidden signature bits.

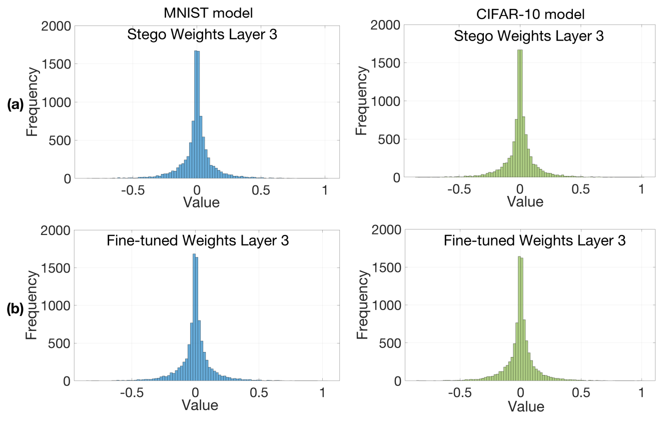
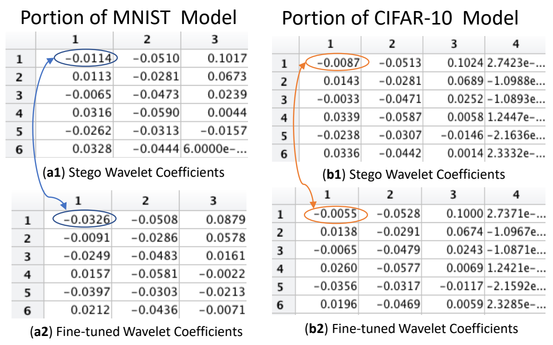


Fig. 16. Histogram comparison of both MNIST and CIFAR-10 between the stego vs Fine-tuned models. It highlights how difficult to detect such attacks.


Portion of MNIST Model

	1	2	3
1	−0.0114	−0.0510	0.1017
2	0.0113	−0.0281	0.0673
3	−0.0065	−0.0473	0.0239
4	0.0316	−0.0590	0.0044
5	−0.0262	−0.0313	−0.0157
6	0.0328	−0.0444	6.0000e−...

(**a1**) Stego Wavelet Coefficients

	1	2	3
1	−0.0326	−0.0508	0.0879
2	−0.0091	−0.0286	0.0578
3	−0.0249	−0.0483	0.0161
4	0.0157	−0.0581	−0.0022
5	−0.0397	−0.0303	−0.0213
6	0.0212	−0.0436	−0.0071

(**a2**) Fine-tuned Wavelet Coefficients

Portion of CIFAR-10 Model

	1	2	3	4
1	−0.0087	−0.0513	0.1024	2.7423e−...
2	0.0143	−0.0281	0.0689	−1.0988e...
3	−0.0033	−0.0471	0.0252	−1.0893e...
4	0.0339	−0.0587	0.0058	1.2447e−...
5	−0.0238	−0.0307	−0.0146	−2.1636e...
6	0.0336	−0.0442	0.0014	2.3332e−...

(**b1**) Stego Wavelet Coefficients

	1	2	3	4
1	−0.0055	−0.0528	0.1000	2.7371e−...
2	0.0138	−0.0291	0.0674	−1.0967e...
3	−0.0065	−0.0479	0.0243	−1.0871e...
4	0.0260	−0.0577	0.0069	1.2421e−...
5	−0.0356	−0.0317	−0.0117	−2.1592e...
6	0.0196	−0.0469	0.0059	2.3285e−...

(**b2**) Fine-tuned Wavelet Coefficients

Fig. 17. A deep dive comparison between the stego wavelet coefficients vs Fine-tuned version. It proves how our DeepAuth is able to detect the attack.

Accuracy Degradation: Our DeepAuth depends on Daubechies Wavelet transform [44]. The decomposition is based on the use of recurrence relations to generate progressively finer discrete samplings of an implicit mother wavelet function; each resolution is twice (i.e., approximation and detailed) that of the previous scale. The detailed side is used in DeepAuth to hide the signatures as it has little effect on recomposing the weights. Theoretically, if the DNN weights have some unique properties which lead to a uniform frequency domain transformation (i.e., all coefficients are significant), then our DeepAuth will result in high distortion which in turns degrades the model accuracy. This is a theoretical assumption and we did not come across such cases in all our experiments.

Adversarial Samples: our DeepAuth focuses on determining if the DNN model has been manipulated after deployment. However, it cannot decide if the input is adversarial or benign.

Hence, adversarial sample attacks are still applicable against DeepAuth since it is designed for the integrity and authenticity of the models.

VI. Related Work

This section provides a brief review of related works on the attacks and defenses on DNN model integrity.

Poisoning attacks: Several techniques have been proposed in the literature to violate DNN integrity by inserting backdoors. Gu et al. [14] introduced a poisoning attack in their BadNets work. They generated a poisoned model by retraining the original one with a poisoned training dataset. The attacked model behaves almost like the benign one except when the backdoor sign is encountered. They also showed that the backdoor remains active even after the transfer learning (to a new model). Liu et al. [22] further improved this attack by tampering only a subset of weights to inject a backdoor. Chen et al. [45] proposed another attack where the attacker does not need to have access to the model. Instead, he/she reengineers the model from scratch and trains it with a poisoned dataset.

Poisoning Defenses: Defense against backdoor attacks is an active area of research, and a few approaches have appeared in the literature. Liu et al. [21] introduced three different defense mechanisms: (1) Employing anomaly detection in the training data: such a method requires access to the poisoned dataset, which is unrealistic in practice; (2) Retrain the model to remove the backdoors: retraining does not guarantee the complete removal of backdoors as demonstrated by previous work in [14]; (3) Preprocessing the input data to remove the trigger – it needs the adversary's aids, which is hard to achieve. Liu et al. [22] suggested that detecting the backdoor might be possible by analysing the distribution of mislabelled data. However, the victim needs to feed the model with a large dataset of samples, rendering such an approach inefficient and expensive. He et al. [23] recently introduced a defense technique by generating sensitive input samples to spot possible changes in hidden weights and produce different outputs. Their model works successfully if the attack tampers the hidden weights in such a way that different neurons are fired.
Despite being robust, these defense techniques lack a mechanism to track the integrity and authenticity of the hidden and outer layers by sealing their weights after generating them.

Adversarial Samples: It is a type of an evasion attack [46]. It manipulates the input data to evade a trained DNN model at the testing time. In other words, it does not poison the model. Rather, it crafts adversarial samples that lead to misclassification by the model. **Attacks:** Fast gradient sign method by Goodfellow et al. [47], basic iterative method by Kurakin et al. [48], DeepFool by Moosavi et al. [49], Jacobian-based Saliency map method by Papernot et al. [50], and DeepXplore method by Pei et al. [51] are a few examples of these attacks. **Defenses:** Adversarial sample detection mechanism is a very active area of research. Examples include a statistical measurement using local intrinsic dimensionality by Ma et al. [16], performing denoiser process on the incoming input as in Magnet [17] and high representation denoiser by Liao et al. [18], feature squeezing mechanism to reduce the colour depth of images by Xu et al. [19], and using network invariant checking by Shiqing et al. [20].

This stream of work is very promising in a black-box setup to determine if the incoming input is benign or adversarial. However, they cannot find out if the integrity of DNN model itself is maintained or violated by poisoning attacks.

Watermarking: Some works use watermarking to protect the Intellectual Property (IP) of DNN models. Uchide et al. [52], [53] proposed a method of embedding a small watermark into deep layers to protect the owner's IP. This work provides a significant leap as the first attempt to watermark neural networks. Zang et al. [41] further extended the technique to the black-box scenario. Merrer et al. [42] introduced 1-bit watermark that is built upon model boundaries modification and the use of random adversarial samples that lie near the decision boundaries. Rouhani et al. [54] proposed an IP protection watermarking technique that not only protects the static weights like previous works but also the dynamic activations. Recently, Adi et al. [15] extended the backdoor approach into a watermarking scheme by inserting a backdoor to claim the ownership of the model.

The focus of this stream of works is to claim the ownership of the models by building a persistent watermark, but not protecting the model integrity as such (i.e., answering the question: has the model been changed?). In other words, even if the model is poisoned or fine-tuned many times, the designed watermark should stay the same to ascertain the ownership. To the best of our knowledge, we are not aware of previous attempts that use watermarking or steganography to protect the integrity and authenticity of DNN models.

VII. Conclusion

We propose a novel wavelet-based steganographic technique, called DeepAuth, to protect the integrity of DNN models. DeepAuth derives a signature of the model using its structural information such as dataset hash, model input size, layer dimensions, total weights and output layer, and hides it within the model. To overcome the resultant high distortion due to hiding, which is inherent to steganography, DeepAuth uses a wavelet-based technique to transform the weights in each layer from the spatial domain to the frequency domain. It then utilizes the approximation coefficients to preserve the accuracy of the model and the detailed coefficients to hide the signature using both secure key and scramble vector. We performed theoretical analysis as well as empirical studies. The analysis showed that DeepAuth can hide about 1KB signature in each layer with 256-bit security level without degrading the accuracy of the model. Several experiments were performed on 3 pre-trained models using 3 datasets against 3 types of manipulation attacks. The results prove that DeepAuth is verifiable at all times with no noticeable effect on classification accuracy, and robust against a multitude of known DNN manipulation attacks.

References

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[2] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[3] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964. IEEE, 2016.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[6] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

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

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Google Inc. Google machine learning engine and ai. *https://cloud.google.com/ml-engine/*, Accessed 12 Jul. 2019.

[12] Microsoft Corp. Microsoft azure machine learning studio. *https://azure.microsoft.com/en-us/services/machine-learning-studio/*, Accessed 12 Jul. 2019.

[13] Amazon.com Inc. Amazon sagemaker: Machine learning for every scientist. *https://aws.amazon.com/sagemaker/*, Accessed 12 Jul. 2019.

[14] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[15] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1615–1631, 2018.

[16] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*, 2018.

[17] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147. ACM, 2017.

[18] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Defense against adversarial attacks using high-level representation guided denoiser. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1787, 2018.

[19] Xu W, Evans D, and Qi Y. Feature squeezing: Detecting adversarial examples in deep neural networks. *in Proceedings of the 2018 Network and Distributed Systems Security Symposium (NDSS), 2018. [Online]. Available: https://github.com/mzweilin/EvadeML-Zoo*, 2018.

[20] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. Nic: Detecting adversarial samples with neural network invariant checking. *in Proceedings of the 2018 Network and Distributed Systems Security Symposium (NDSS), 2019. [Online].*, 2019.

[21] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 45–48. IEEE, 2017.

[22] Liu Y, Ma S, Aafer Y, Lee W.-C, Zhai J, Wang W, and Zhang X. Trojaning attack on neural networks. *in 25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018. The Internet Society, 2018. [Online]. Available: https://github.com/ PurduePAML/TrojanNN*, 2018.

[23] Zecheng He, Tianwei Zhang, and Ruby B Lee. Verideep: Verifying integrity of deep neural networks through sensitive-sample fingerprinting. *arXiv preprint arXiv:1808.03277*, 2018.

[24] Abbas Cheddad, Joan Condell, Kevin Curran, and Paul Mc Kevitt. Digital image steganography: Survey and analysis of current methods. *Signal processing*, 90(3):727–752, 2010.

[25] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Digital watermarking and steganography*. Morgan kaufmann, 2007.

[26] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Urs A Muller, Eduard Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

[27] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[29] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: A generic watermarking framework for ip protection of deep learning models. *arXiv preprint arXiv:1804.00750*, 2018.

[30] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[31] Chitra Biswas, Udayan Das Gupta, and Md Mokammel Haque. An efficient algorithm for confidentiality, integrity and authentication using hybrid cryptography and steganography. In *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pages 1–5. IEEE, 2019.

[32] Peter Eze, Udaya Parampalli, Robin Evans, and Dongxi Liu. Integrity verification in medical image retrieval systems using spread spectrum steganography. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pages 53–57. ACM, 2019.

[33] Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(7):674–693, 1989.

[34] Alexander D Poularikas. *Transforms and applications handbook*. CRC Press, 2010.

[35] David Salomon. *Data compression: the complete reference*. Springer, 2004.

[36] Ibrahim Khalil and Fahim Sufi. Real-time ecg data transmission with wavelet packet decomposition over wireless networks. In *Intelligent Sensors, Sensor Networks and Information Processing, 2008. ISSNIP 2008. International Conference on*, pages 267–272. IEEE, 2008.

[37] Jiaxin Ning, Jianhui Wang, Wenzhong Gao, and Cong Liu. A wavelet-based data compression technique for smart grid. *Smart Grid, IEEE Transactions on*, 2(1):212–218, 2011.

[38] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[39] A Enis Cetin and Hayrettin Köymen. Compression of digital biomedical signals. *The Biomedical Engineering Handbook: Second Edition. Joseph D. Bonzino, Ed. CRC Press LLC*, 2000.

[40] Bernard Sklar. *Digital communications*, volume 2. Prentice Hall NJ, 2001.

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

[42] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *arXiv preprint arXiv:1711.01894*, 2017.
[43] A. Ibaida and I. Khalil. Wavelet-based ecg steganography for protecting patient confidential information in point-of-care systems. *IEEE Transactions on Biomedical Engineering*, 60(12):3322–3330, Dec 2013.
[44] Ingrid Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 41(7):909–996, 1988.
[45] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.
[46] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317 – 331, 2018.
[47] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
[48] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.
[49] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.
[50] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016.
[51] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.
[52] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pages 269–277. ACM, 2017.
[53] Yuki Nagai, Yusuke Uchida, Shigeyuki Sakazawa, and Shinichi Satoh. Digital watermarking for deep neural networks. *International Journal of Multimedia Information Retrieval*, 7(1):3–16, 2018.
[54] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: A generic watermarking framework for ip protection of deep learning models. *arXiv preprint arXiv:1804.00750*, 2018.

VIII. Appendix

A. Accuracy Evaluation - AlexNet and VGG16

Table V shows the exact accuracy results from two DNN architectures: AlexNet and VGG16. They have been evaluated before and after applying the best steganography level of 2-bits observed in ResNet18. The obtained accuracy results are the same. BER is 0% in both cases.

TABLE V
Comparison between baseline classification accuracy (%) before and after applying our DeepAuth with different CNN architectures.

Model	**Baseline**	**DeepAuth - 2-bits level**	BER
AlexNet	**90.58**	**90.58**	0
VGG16	**91.70**	**91.70**	0

B. Distortion Evaluation - AlexNet and VGG16

Table VI presents the distortion measurements for AlexNet and VGG16. We compare the original with both stego and extracted versions. All PRDs are very low $\leq 0.25\%$ with an average of $\leq 0.20\%$ which matches our early findings in ResNet18.

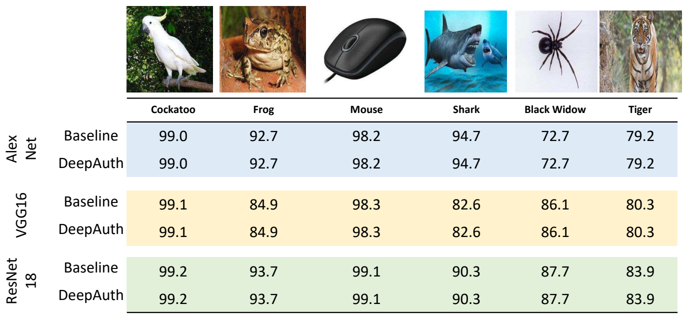
TABLE VI
PRD results for AlexNet and VGG16

	AlexNet		VGG16	
Layer No	PRD % Stego	PRD % Extract	PRD % Stego	PRD % Extract
1	0.0173	0.0173	0.0602	0.0602
2	0.0470	0.0471	0.0691	0.0692
3	0.1003	0.1002	0.0556	0.0554
4	0.0560	0.0561	0.0581	0.0581
5	0.1742	0.1743	0.2555	0.2556
6	0.2368	0.2369	0.2540	0.2541
7	0.2587	0.2587	0.2441	0.2496
8	0.1995	0.1999	0.2509	0.2510
9	0.2101	0.2103	0.2397	0.2396
10	0.2335	0.2378	0.2342	0.3343
11	0.2187	0.2191	0.2382	0.2380
12	0.2378	0.2386	0.2509	0.2515
13	0.2317	0.2316	0.2493	0.2542
14	0.2313	0.2314	0.2487	0.2483
Avg	0.1752	0.1757	0.1992	0.2071

AA

C. Accuracy Evaluation with Random Samples

An out of the box experiment to monitor the accuracy results before and after applying DeepAuth. Several samples have been picked from the Internet. We only pre-process them by reducing their sizes to fit the input requirements of ResNet18 and VGG16 as (224x224x3), and AlexNet as (227x227x3). The obtained accuracy results are exactly the same in all cases.



		Cockatoo	Frog	Mouse	Shark	Black Widow	Tiger
Alex Net	Baseline	99.0	92.7	98.2	94.7	72.7	79.2
	DeepAuth	99.0	92.7	98.2	94.7	72.7	79.2
VGG16	Baseline	99.1	84.9	98.3	82.6	86.1	80.3
	DeepAuth	99.1	84.9	98.3	82.6	86.1	80.3
ResNet 18	Baseline	99.2	93.7	99.1	90.3	87.7	83.9
	DeepAuth	99.2	93.7	99.1	90.3	87.7	83.9

Fig. 18. A detailed classification accuracy using sample images before/after applying our DeepAuth on three different DNN architectures, i.e., AlexNet, VGG16 and ResNet18. In all cases, our DeepAuth has no noticeable effect on the accuracy.

D. Weights Decomposition Tree

Fig. 20 depicts a graphical representation of the resultant wavelet tree after transforming series of weights from their spatial domain into their 5-levels frequency domain.

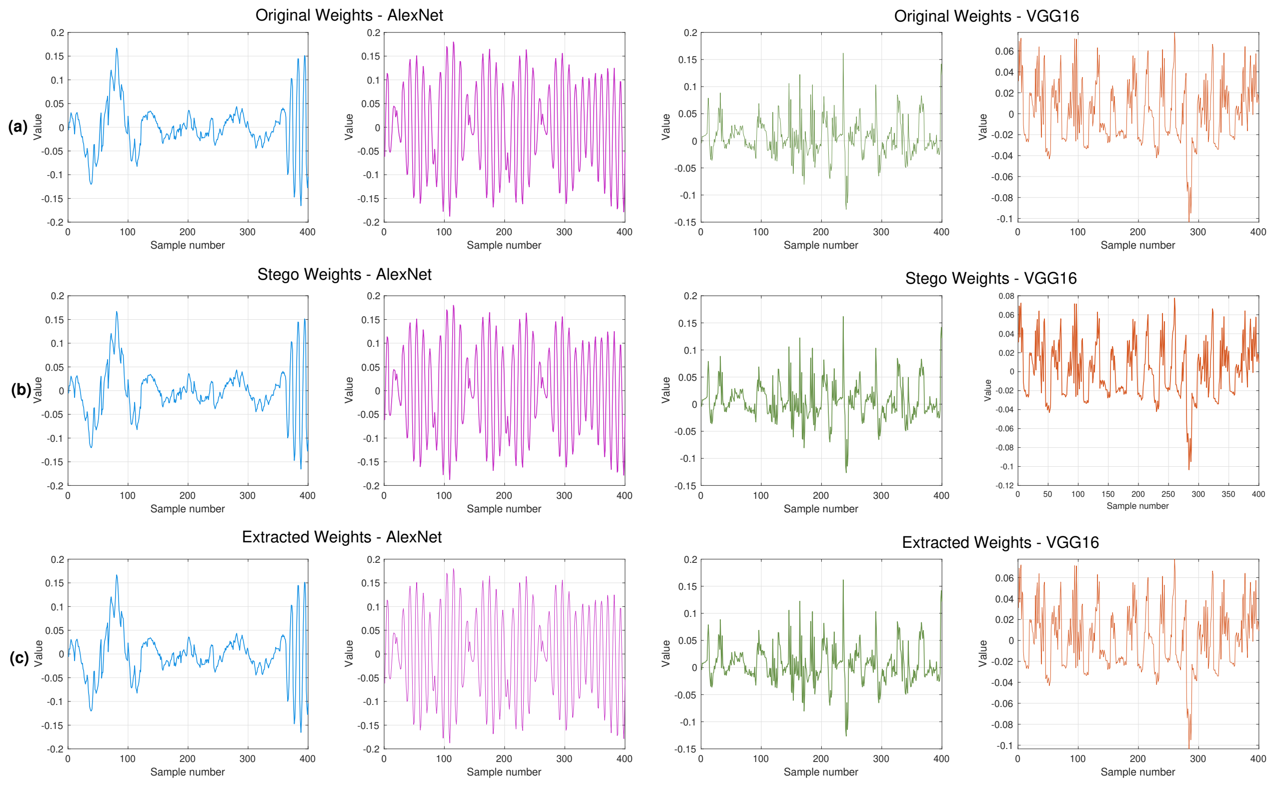


Fig. 19. Four examples of DNN hidden layers weights obtained using other CNN architectures, AlexNet and VGG16: (a) the original form; (b) stego form (i.e., containing hidden signature) that obtained after applying our DeepAuth; and (c) extracted form after removing the hidden signature.

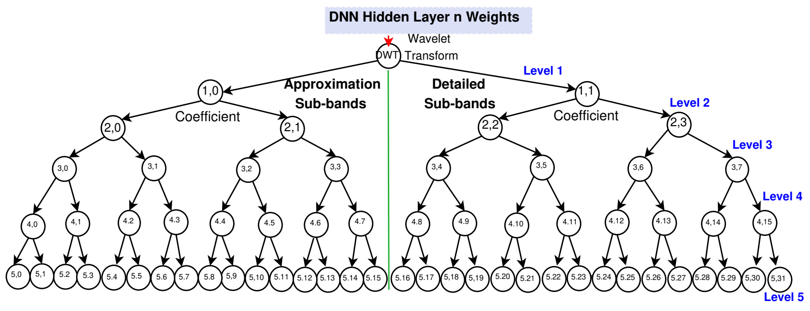


Fig. 20. De-composing a series of DNN hidden layer weights into 32 sub-bands using Wavelet transform.

E. Weights Comparison - AlexNet and VGG16

Fig. 19 demonstrates a visual comparison between series of weights in original, stego and extracted versions. Examples are randomly picked from AlexNet and VGG16.