Find the Lady: Permutation and Re-Synchronization of Deep Neural Networks

Carl De Sousa Trias[1], **Mihai Petru Mitrea**[1], **Attilio Fiandrotti**[2], **Marco Cagnazzo**[3], **Sumanta Chaudhuri**[4], **Enzo Tartaglione**[4]
[1]Télécom SudParis, Institut Polytechnique de Paris, France
[2]University of Turin, Italy
[3]University of Padua, Italy
[4] LTCI, Télécom Paris, Institut Polytechnique de Paris, France
carl.de-sousa-trias@telecom-sudparis.eu

Abstract

Deep neural networks are characterized by multiple symmetrical, equi-loss solutions that are redundant. Thus, the order of neurons in a layer and feature maps can be given arbitrary permutations, without affecting (or minimally affecting) their output. If we shuffle these neurons, or if we apply to them some perturbations (like fine-tuning) can we put them back in the original order i.e. re-synchronize? Is there a possible corruption threat? Answering these questions is important for applications like neural network white-box watermarking for ownership tracking and integrity verification.
We advance a method to re-synchronize the order of permuted neurons. Our method is also effective if neurons are further altered by parameter pruning, quantization, and fine-tuning, showing robustness to integrity attacks. Additionally, we provide theoretical and practical evidence for the usual means to corrupt the integrity of the model, resulting in a solution to counter it. We test our approach on popular computer vision datasets and models, and we illustrate the threat and our countermeasure on a popular white-box watermarking method.

1 Introduction

The deployment of deep neural networks for solving complex tasks became massive, for both industrial and end-user-oriented applications. These tasks are instantiated in a huge variety of applications, e.g. autonomous driving cars. In this context, neural networks are in charge of safety-critical operations such as forecasting other vehicles' trajectories, acting on commands to dodge pedestrians, etc. The interest in protecting the integrity and the intellectual property of such networks has steadily increased even for non-critical tasks, like ChatGPT content detection (Uchida et al. 2017; Adi et al. 2018; Li, Wang, and Barni 2021). Some watermarking techniques already allow embedding signatures inside deep models (Uchida et al. 2017; Chen et al. 2019a; Tartaglione et al. 2020), but these are designed to be robust against conventional attacks, including fine-tuning, pruning, or quantization, and assume the original location of the watermarked parameters remains unchanged. Neural networks, however, have internal symmetries such that entire neurons can be permuted, without impacting the overall computational graph. Once this happens, although the input-output function for the whole model does not change, the ordering of the parameters in the layer changes, and for instance, all the aforementioned watermarking approaches fail in retrieving the signature of the model, despite it still being there. This is referred to as *geometric attack* in the multimedia watermarking community (Wan et al. 2022), and we port the same concept to deep neural networks: the input-output relationship is preserved, but the order of the neurons is permuted, disallowing the recovery of signatures (Fig. 1).
Some studies (Hecht-Nielsen 1990; Ganju et al. 2018; Li, Wang, and Zhu 2022) already raised concerns about permutation in deep layers; yet, such a problem has not yet been studied in its general form, nor has its formal definition been stated. The first question we ask ourselves is whether the original ordering for the neurons can be retrieved, even when the applied permutation rule is lost. It is also a well-known fact that deep neural networks are redundant (Setiono and Liu 1997; Agliari et al. 2020; Wang, Li, and Wang 2021)

This paper has been accepted for publication at the 38th Annual AAAI Conference on Artificial Intelligence (AAAI 24).

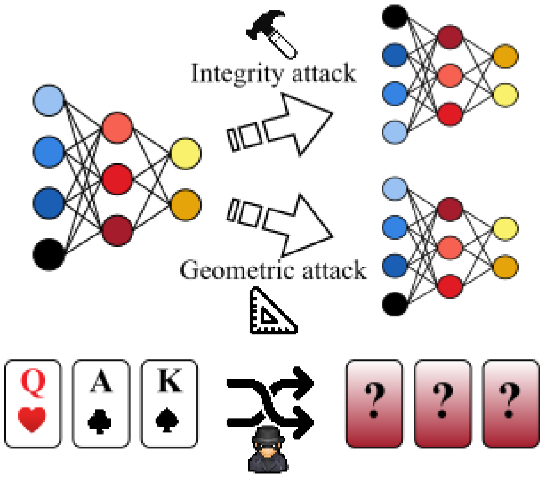


Figure 1: Given some model (left), let us assume we permute the order of neurons and apply other types of corruption (right): are integrity checks at the neuron's level enough to verify the integrity of the model? And what about retrieving the signature in white-box watermarking? This problem resembles the "find the lady/ three-card monte" game, where the queen of hearts needs to be found out of shuffled cards.

and some works enforce this towards improving the generalization capability of the neural network, like dropout (Srivastava et al. 2014), while others detect such redundancies and prune them away (Wang, Li, and Wang 2021; Chen et al. 2019b; Tartaglione et al. 2021). Hence, it is not even clear whether it is possible to "distinguish", with no doubt, one neuron from all the others in the layer. This would be an important step to re-order (*re-synchronize*) the neurons in the target layer. Besides, we ask the same question even in the case we apply some noise to the model: as a fact, the learning process for deep neural networks is noisy, and robustness towards the unequivocal identification of the parameters belonging to a neuron from the others in a noisy environment is important in the considered setup.
The main contributions of this study can be summarized as:

- we study the neuron redundancy case for deep neural networks, observing that despite some neurons showing the same input/output function, under the same input, their parameters can be consistently different;
- we explore different ways to re-synchronize a permuted model, showing and explaining fallacies for some of the most intuitive approaches;
- we put in evidence a potential integrity threat for re-synchronized models and we highlight the counter-measure for it;
- we advance an effective solution to re-synchronize layers, even when subjected to noise, and we extensively validate it with four different noise sources, on five different datasets, and nine different architectures.

2 Permuting neurons

2.1 Preliminaries

In this section, we define the neuron permutation problem. For the sake of simplicity, we will exemplify the problem on a single fully connected layer without biases; however, the same conclusions hold for any other layer typology, e.g. convolutional or batch-normalization. Let us define the output $\boldsymbol{y}_l \in \mathbb{R}^{N_l \times 1}$ of the l-th layer:

$$\boldsymbol{y}_l = \varphi\left[\langle \boldsymbol{w}_l, \boldsymbol{y}_{l-1}\rangle\right] \tag{1}$$

where $\boldsymbol{y}_{l-1} \in \mathbb{R}^{N_{l-1} \times 1}$ is the input, $\boldsymbol{w}_l \in \mathbb{R}^{N_{l-1} \times N_l}$ are the weights for the l-th layer (as displayed in Fig. 2a), $\langle \cdot \rangle$ indicates inner product, and $\varphi(\cdot)$ is the activation function. Let us consider the case a permutation π_l is applied on the neurons of the l-th layer; the elements in the permutation matrix $P_{\pi_l} \in \mathbb{R}^{N_l \times N_l}$ are:

$$(P_{\pi_l})_{i,j} = \begin{cases} 1 & \text{if } j = \pi_l(i) \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The neurons are permuted, and the ordering for the input channels $\boldsymbol{y}_{l-1}$ remains intact (Fig. 2b). Hence, the permuted output for the l-th layer will be

$$\boldsymbol{y}_l^{\pi_l} = \varphi\left(\langle \boldsymbol{w}_l^{\pi_l}, \boldsymbol{y}_{l-1}\rangle\right), \tag{3}$$

$$\boldsymbol{w}_{l,c,-}^{\pi_l} = \langle P_{\pi_l}, (\boldsymbol{w}_{l,c,-})\rangle \ \forall c; \tag{4}$$

where $\boldsymbol{w}_{l,c,-}$ represents all elements of the l-th layer for the c-th channel, and $\boldsymbol{w}_{l,-,n}$ represents all elements of the l-th layer for the n-th neurons. After having applied π_l at layer l, the output of the model is likely to be altered, as the propagated $\boldsymbol{y}_l^{\pi_l} \neq \boldsymbol{y}_l$, which is processed as input by the next layer (Fig. 2c). Hence, to maintain the output of the full model unaltered, we need to also permute the weights in layer $l+1$

$$\boldsymbol{w}_{l+1,-,n}^{\pi_l} = \langle P_{\pi_l}, (\boldsymbol{w}_{l+1,-,n})\rangle \ \forall n. \tag{5}$$

In this way, the permuted outputs in the l-th layer will be correctly weighted in the next layer, and the neural network output will be unchanged (Fig. 2d). To illustrate our study, we define a companion dataset and an architecture, namely the CIFAR-10 and VGG-16 (without batch normalization), respectively. The model we will use as a reference is trained for 50 epochs using SGD, with a learning rate 10^{-2}, weight decay 10^{-4}, and momentum 0.9. Let the first convolutional layer of the fourth block of convolutions (where every block is separated by a maxpool layer) be our l-th layer.

2.2 Any hope to recover the original order?

Assuming the $P_{\pi_l} \in \mathbb{R}^{N_l \times N_l}$ matrix is known, the answer is straightforward. Yet, the question becomes hard to answer when the $P_{\pi_l} \in \mathbb{R}^{N_l \times N_l}$ matrix is unknown. The difficulty derives from the fact that neural network models internally have many redundancies (Setiono and Liu 1997; Wang, Li, and Wang 2021) that can a priori cast confusion when trying to find the initial order. Many approaches, like dropout (Srivastava et al. 2014), enforce this to make deep models robust against noise. Consider the case in which two neurons belonging to the l-th layer are *redundant*, and let them be denoted by the i-th and the j-th, with the parameters $\boldsymbol{w}_{l,-,i}$ and $\boldsymbol{w}_{l,-,j}$. Given some ξ-th sample in $\mathcal{D}$, with $\mathcal{D}$ being the dataset the model is trained on, $y_{l,i}^{\xi} = y_{l,j}^{\xi}$. From this, we can have two scenarios:

- $\boldsymbol{w}_{l,-,i} = \boldsymbol{w}_{l,-,j}$: in this case, the i-th and the j-th neuron share exactly the same parameters. As such, since they receive the same input $\boldsymbol{y}_{l-1}^{\xi}$, and by construction, they have all the same activation function $\varphi(\cdot)$, they map the same function and they are, hence, identical. Since they are the same from all points of view, ordering them one way or another does not matter.
- $\boldsymbol{w}_{l,-,i} \neq \boldsymbol{w}_{l,-,j}$: in this case, the i-th and the j-th neuron have a different set of parameters, but share the same outputs for some samples in $\mathcal{D}$.

The second case is the most interesting: is it possible to recover the original ordering of neurons exhibiting the same output under the same input? It is easy to prove that

$$\boldsymbol{w}_{l,-,i} = k \cdot \boldsymbol{w}_{l,-,j} \Rightarrow y_{l,i}^{\xi} = k \cdot y_{l,j}^{\xi} \forall \xi, \tag{6}$$

with $k \in \mathbb{R}$ being some scalar quantity. To test whether two neurons are extracting the same information, we can compute the cosine similarity $S_C(y_{l,i}, y_{l,j})$ between their outputs, and ask that it is exactly one: from this, we obtain

$$\sum_{\xi} y_{l,i}^{\xi} y_{l,j}^{\xi} = \sqrt{\sum_{\xi}\left(y_{l,i}^{\xi}\right)^2}\sqrt{\sum_{\xi}\left(y_{l,j}^{\xi}\right)^2}. \tag{7}$$

From (1) it is clear that, having non-linear activations and in general $N_{l-1} > N_l$, (7) is satisfiable for $\boldsymbol{w}_{l,-,i} \neq \boldsymbol{w}_{l,-,j}$.

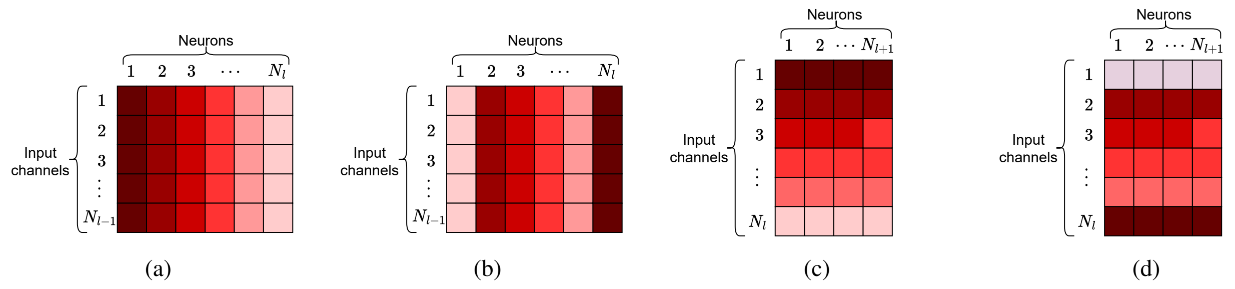


Figure 2: Representation of the weights tensor for the l-th layer (a), permutation of neurons 1 and N_l (b), representation of the weights tensor for layer $l+1$ (c) permutation on channels, following the same permutation of l (d).

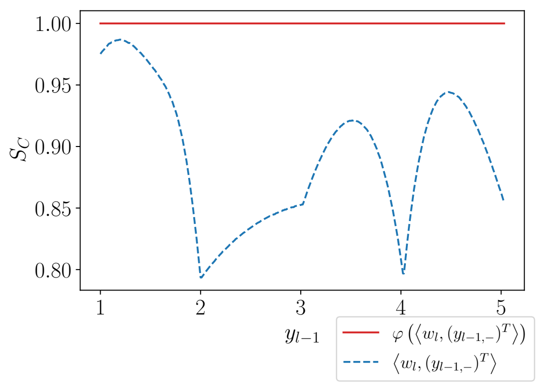


Figure 3: Evolution of cosine similarity of two non-zero neurons before and after activation function for y_{l-1} inputs.

Let us observe this empirically, using our companion setup: we select 2 neurons i,j of the l-th layer such that their cosine similarity $S_C(y_{l,i}, y_{l,j}) = 1$, for several values of k. Since l is a convolutional layer, we know that $\boldsymbol{y}_{l,i}^{\xi} \in \mathbb{R}^{1\times M_l}$, where M_l is a function of the input size for l, kernel size and stride. Hence, we are able here to plot the cosine similarity given the input of one single ξ-th sample and to track the change of the similarity between $\boldsymbol{y}_{l-1}^{\xi}$ and $\boldsymbol{y}_{l-1}^{\xi+1}$. Fig. 3 displays the cosine similarity between two neurons in the l-th layer before and after the activation function. Despite the cosine similarity remaining to one, this happens thanks to the non-linear activation, as the pre-activation potentials are less correlated. Furthermore, we observe that the parameters of these neurons are essentially de-correlated, as their cosine similarity values -0.02. This shows that even if two neurons have a similar (non-zero) response to the same input, their internal function (before the non-linearity) can be different. This gives us hope to distinguish each neuron, hence, retrieving the original ordering of the neurons.

3 Re-synchronizing neurons

In this section, we first define against which additional modifications, applied in conjunction with the permutation, the counterattack should still retrieve the original order, namely: Gaussian noise, fine-tuning, pruning, and quantization. Second, we explore the potential counterattack solutions by presenting methods of the state-of-the-art and showing where they worked and failed. Finally, we present our method leveraging the cosine similarity to recover the original order.

3.1 Robustness in retrieving the original order

In the previous section, we discussed how neurons can be permuted inside a neural network without impacting the model performance. In this section, assuming the initial permutation matrix is no longer available, we will explore ways to recover the original ordering for permuted neurons, even when they are possibly modified. In particular, we will explore robustness in retrieving the original order when undergoing four different transformations:

- **Gaussian noise addition**: we apply an additive noise $\mathcal{N}(0, \sigma_l\Omega)$, with $\Omega \geq 0$, σ_l standard deviation of l.
- **fine-tuning**: we resume the original training of the model with Θ standing for the ratio of fine-tuning epochs to the original training epochs.
- **quantization**: we reduce the number of bits B used to represent the parameters of the model.
- **magnitude pruning**: we mask the T fraction of the smallest weights of the model, according to the ℓ_1-norm.

Even when the model undergoes these transformations, our goal is to be able to recover the original ordering for the model: we denote by Ψ as the fraction of neurons we were able to place back to their original position (multiplied by 100), and we shall refer it as *re-synchronization success rate*. Here follows a sequence of approaches aiming at bringing Ψ close to 100, under the aforementioned transformations.

3.2 In the search of the lost synchronization

The next sections explore the different methods to solve the permutation problem.

Finding the canonical space: rank the neurons Our first approach consists of ranking all the neurons in the l-th layer according to some specific scoring function. For instance, we can attempt to look at the intrinsic properties of the neurons inside the layer, like their weight norm, to perform a ranking (Ganju et al. 2018). Unfortunately, this approach is not general: there are specific cases, like spherical neurons (Lei, Akhtar, and Mian 2019) in which the parameters are normalized and, for instance, not possible to be ranked

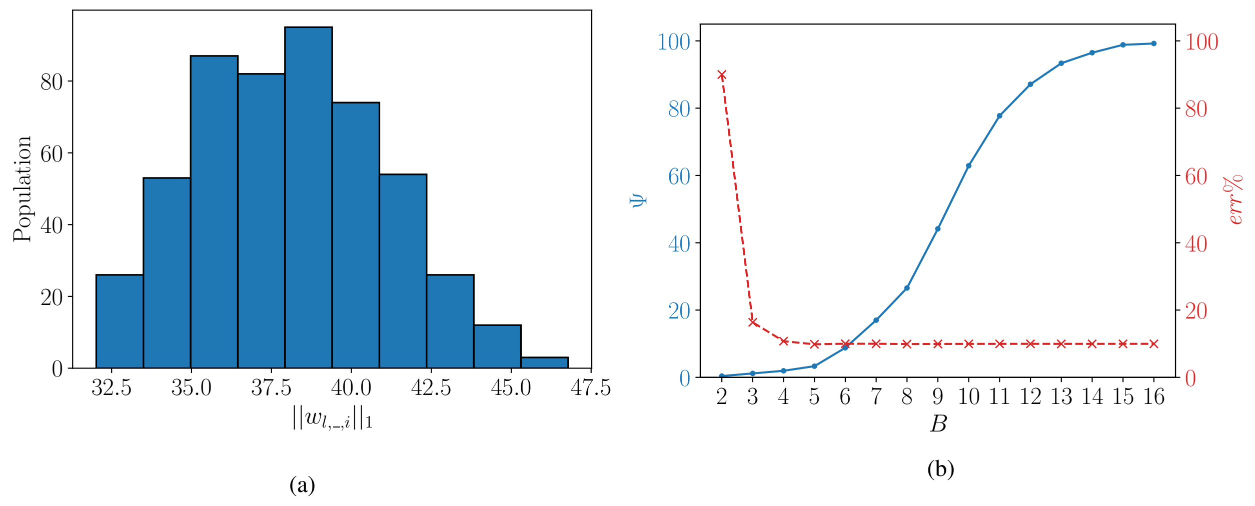


Figure 4: (a) L1 norm distribution of the neurons of the l-th layer a VGG-16 model trained on CIFAR10. (b) Robustness of ranking L1 norms of neurons, against quantization. Ψ is on the left axis in blue and the *err %* on the right axis in red.

according to their norm. This effect is not limited to these special models: if we plot the distribution of the norms for the l-th layer in our companion VGG-16 model trained, as represented in Fig. 4a, we observe that typically the values for the norm of the neuron's parameters are in a very small domain: for instance, the minimal gap between these norms is in the order of 10^{-5}. We expect, hence, that this ranking is very sensitive to all the aforementioned transformations. As an example, Fig. 4b displays the non-robustness against quantization attack: the neurons are permuted (blue line, the higher the better) before losing any performance on the task (red line, the lower the better).

Creating a trigger set A second approach could be to learn an input $\boldsymbol{y}_{l-1}$ such that the output $\boldsymbol{y}_l$ permits to identify the neurons. With our first approach, we aim to learn a $\boldsymbol{y}_{l-1}$ such that we maximize the distance between all the neurons' outputs. Then, we say the norm of the output corresponds to the ranking of the neuron itself. Empirically, in our companion setup, we observe that this approach is not robust to fine-tuning. Indeed, despite having a minimum gap between outputs larger than one, after just an extra 1% of training, we observe a re-synchronization success rate already dropping to 10%, or in other words, we are not able to recover the exact position for the 90% of neurons. This effect confirms our idea that neurons could have similar behavior for the same input which makes them easily swapped after any modifications. Another approach is developed in (Li, Wang, and Zhu 2022) which aims to learn a set of inputs to identify a neuron based on the response to the trigger set. However, this method seems ineffective since the re-synchronization success rate never reaches 100% and it was only tested on the first layers of the neural network.
Finally, we simplify the problem by identifying each neuron independently from the others. If we can do so, then we will also be able to re-synchronize the whole layer. Towards this end, using a similar strategy heavily employed in many interpretability works (Suzuki et al. 2017), we can learn the input $\boldsymbol{y}_{l-1}$ which maximizes the response of the i-th neuron only, and at the same time minimizes the response of all the others. With this method we can create a set of N_l inputs for the l-th layer, to identify all its neurons.
This approach shows its robustness to all the modifications, but has a big drawback: it demands a lot of memory to store the learned inputs (we need one input per neuron, hence the space complexity is $\mathcal{O}(N_{l-1} \cdot N_l \cdot M_{l-1})$, where M_{l-1} is the size of each output coming from $l-1$). Besides, we need also a consistent computational effort, as we need to forward a batch of N_l inputs. This makes the "re-synchronizer" overall bigger than the model itself and becomes prohibitive.

3.3 Find the lady by similarity

With the previous approaches, we have observed that it is, in general, difficult to learn some input $\boldsymbol{y}_{l-1}$ such that the output $\boldsymbol{y}_l$ provides us the ranking of the neurons for l, as this approach is very sensitive to any minor perturbation introduced in the model. We have observed, though, that it is possible to uniquely recognize each neuron independently, learning a specific $\boldsymbol{y}_{l-1}$ which activates the target neuron. However, this solution consumes a lot of memory and computation resources. We have seen that two neurons, despite having the same output in some subspace of the trained domain, are in general very different. In particular, we can expect that $S_C(\boldsymbol{w}_{l,i,-}, \boldsymbol{w}_{l,j,-}) < 1 \forall j \neq i$.
Let us study this phenomenon practically. Fig. 6a shows the correlation between the neuron's weights $\boldsymbol{w}_l$: since it is essentially a diagonal matrix, after applying some unknown permutation π_l as in Fig. 6b, we can easily recover the original positions building the Permutation matrix

$$(P_{\pi_l})_{i,j} = \begin{cases} 1 & j = \text{argmax}_k \left[S_C(\boldsymbol{w}_{l,i,-}, \boldsymbol{w}^{\pi_l}_{l,k,-}) \right] \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Algorithm 1: Re-synchonization algorithm.

```
Inputs: the original model $\Gamma$, the altered model $\tilde{\Gamma}_{\pi_l}$, the number of layers of these models $L$.
Output: The re-synchronized model $\tilde{\Gamma}$
for $l = \{1, \dots, L-1\}$ do
    Step 1: Compute score metric on $\tilde{\boldsymbol{w}}_l^{\pi_l}$
    $\boldsymbol{w}_l \in \mathbb{R}^{N_{l-1} \times N_l} \leftarrow$ parameters in $l^{th}$ layer of $\Gamma$
    $\tilde{\boldsymbol{w}}_l^{\pi_l} \in \mathbb{R}^{N_{l-1} \times N_l} \leftarrow$ parameters in $l^{th}$ layer of $\tilde{\Gamma}_{\pi_l}$
    $S \leftarrow S_C(\boldsymbol{w}_l, \tilde{\boldsymbol{w}}_l^{\pi_l}) = \frac{(\boldsymbol{w}_l)^T \cdot \tilde{\boldsymbol{w}}_l^{\pi_l}}{\|\boldsymbol{w}_l\|_2 \|\tilde{\boldsymbol{w}}_l^{\pi_l}\|_2}$
    Step 2: Obtain the permutation matrix $P_{\pi_l^{-1}}$
    $P_{\pi_l^{-1}} \leftarrow [0]_{N_l, N_l}$
    for $i = \{1, \dots, N_l\}$ do
        $j \leftarrow \text{argmax}_i(S)$
        $(P_{\pi_l^{-1}})_{i,j} = 1$
    end for
    Step 3: Permute neurons in $l^{th}$ layer of $\tilde{\Gamma}$ and channels in $(l+1)^{th}$ of $\tilde{\Gamma}_{\pi_l}$
    $\tilde{\boldsymbol{w}}_l \leftarrow \left\langle P_{\pi_l^{-1}}, \left(\tilde{\boldsymbol{w}}_{l,c,-}^{\pi_l}\right)\right\rangle \forall c$          ▷ equation (4)
    $\tilde{\boldsymbol{w}}_{l+1}^{\pi_l} \leftarrow \left\langle P_{\pi_l^{-1}}, \left(\tilde{\boldsymbol{w}}_{l+1,-,n}^{\pi_l}\right)\right\rangle \forall n$          ▷ equation (5)
end for
return $\tilde{\Gamma}$
```

The question is here whether, even after applying some perturbation to the parameters, we are still able to recover the permutation π. As such, let us define $\tilde{\boldsymbol{w}}_{l,i,-}$ the set of parameters of the i neuron in the l-th layer undergoing some perturbation. We can assume that any perturbation we want to introduce does not significantly change the performance $\mathcal{L}_\Xi$ of the trained model. As such, let us evaluate the cosine similarity between $\boldsymbol{w}_{l,i,-}$ and $\tilde{\boldsymbol{w}}_{l,i,-}$: we expect that when this measure drops, the performance of the model will drop as well. Two neurons, despite having the same output in the trained domain, are in general different: we can expect that

$$S_C(\boldsymbol{w}_{l,i,-}, \tilde{\boldsymbol{w}}_{l,i,-}) > S_C(\boldsymbol{w}_{l,i,-}, \tilde{\boldsymbol{w}}_{l,j,-}) \forall j \neq i. \quad (9)$$

According to (9), it is possible to detect where the i-th neuron has been displaced, thus, recovering the original ordering. This condition obeys some theoretical warranties. Let us compare the set parameters $\boldsymbol{w}_{l,i}$ to the same, where we apply a perturbation, which results in $\tilde{\boldsymbol{w}}_{l,i} = \boldsymbol{w}_{l,i} + \hat{\boldsymbol{w}}_{l,i}$. According to the Cauchy-Schwarz inequality, the only possible solution is that $\hat{\boldsymbol{w}}_{l,i}$ is a scalar multiple of $\boldsymbol{w}_{l,i}$.
Let us investigate the case in which we perform fine-tuning on the parameters: we record a slight improvement in the performance with $\Theta = 2\%$, and we observe that the permutation matrix (Fig. 6d) we obtain from the cosine similarities (Fig. 6c) is the same as the one recovered before, making out re-synchronization success rate to 100%. The details of our method are presented in Alg. 1 and Fig. 5.

3.4 Integrity loss

We will analyze here the special case when $\hat{\boldsymbol{w}}_{l,i} = k \cdot \boldsymbol{w}_{l,i}$. Let us assume the input of the l-th layer follows a Gaussian distribution, with mean $\boldsymbol{\mu}_l$ and covariance matrix Σ_l. We

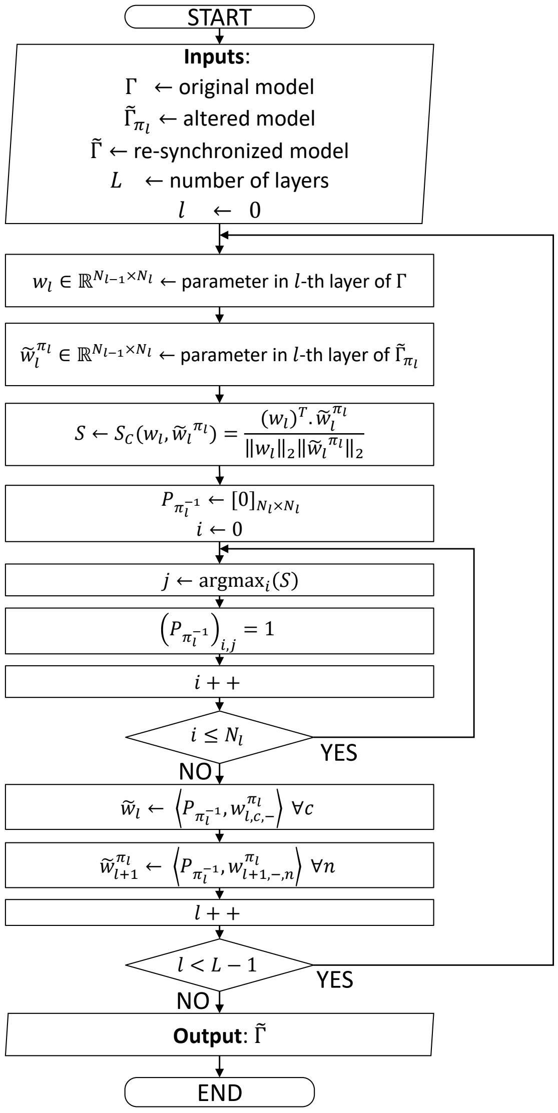

START
Inputs:
$\Gamma \leftarrow$ original model
$\tilde{\Gamma}_{\pi_l} \leftarrow$ altered model
$\tilde{\Gamma} \leftarrow$ re-synchronized model
$L \leftarrow$ number of layers
$l \leftarrow 0$
$w_l \in \mathbb{R}^{N_{l-1} \times N_l} \leftarrow$ parameter in l-th layer of Γ
$\tilde{w}_l^{\pi_l} \in \mathbb{R}^{N_{l-1} \times N_l} \leftarrow$ parameter in l-th layer of $\tilde{\Gamma}_{\pi_l}$
$S \leftarrow S_C(w_l, \tilde{w}_l^{\pi_l}) = \frac{(w_l)^T . \tilde{w}_l^{\pi_l}}{\|w_l\|_2 \|\tilde{w}_l^{\pi_l}\|_2}$
$P_{\pi_l^{-1}} \leftarrow [0]_{N_l \times N_l}$
$i \leftarrow 0$
$j \leftarrow \text{argmax}_i(S)$
$\left(P_{\pi_l^{-1}}\right)_{i,j} = 1$
$i++$
$i \leq N_l$
YES
NO
$\tilde{w}_l \leftarrow \left\langle P_{\pi_l^{-1}}, w_{l,c,-}^{\pi_l}\right\rangle \forall c$
$\tilde{w}_{l+1}^{\pi_l} \leftarrow \left\langle P_{\pi_l^{-1}}, w_{l+1,-,n}^{\pi_l}\right\rangle \forall n$
$l++$
$l < L-1$
YES
NO
Output: $\tilde{\Gamma}$
END

Figure 5: Flowchart of Alg. 1.

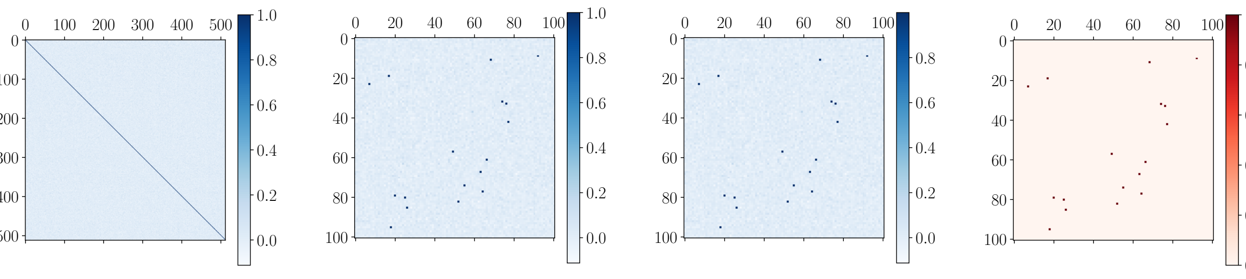
know that the post-synaptic potential still follows a Gaussian distribution $\mathcal{N}(\mu_z, \sigma_z^2)$. Given that $\hat{\boldsymbol{w}}_{l,i}$ will produce as output $\tilde{z}$, we can write the KL-divergence between the outputs generated from the original and from the perturbed neuron

$$D_{\text{KL}}(z||\tilde{z}) = \log(1+k) + \frac{\sigma_z^2 + k^2\mu_z^2}{2(1+k)^2\sigma_z^2} - \frac{1}{2}. \quad (10)$$

Under the assumption of having an activation such that $|\varphi(x)'| \leq 1 \forall x \in \mathbb{R}$, we know that the above divergence upper-bounds $D_{\text{KL}}(y||\tilde{y})$. Specifically, for ReLU activations, under the assumption of $\mu_z = 0$, the KL-divergence is

$$D_{\text{KL}}(y||\tilde{y}) = \frac{2(k+1)^2 \log(k+1) - k(k+2)}{(k+1)^2}, \quad (11)$$

which is dependent on k only. Despite having maximum similarity (except for the degenerate case $k = -1$), the KL



(a) $S_C(\boldsymbol{w}_{l,i,-}, \boldsymbol{w}_{l,i,-})$ (b) $S_C(\boldsymbol{w}_{l,i,-}, \boldsymbol{w}^{\pi_l}_{l,i,-})$ (c) $S_C(\boldsymbol{w}_{l,i,-}, \tilde{\boldsymbol{w}}^{\pi_l}_{l,i,-})$ (d) P_{π_l}

Figure 6: Cosine similarity for different situations (a) without permutation (b) with permutation (4) and (c) with fine-tuning and permutation (4) (d) is $P(\pi_l)$ for both (b) and (c). For visibility purposes, (b), (c), and (d) are clipped (first 100 elements).

divergence of the output is non-zero $\forall k \neq 0$, which means the behavior of the model is modified.

4 Experimental results

Datasets We will test our proposed approach on five datasets: CIFAR-10 (Krizhevsky, Hinton et al. 2009) and ImageNet-1k (Russakovsky et al. 2015) for image classification (metric is here top-1 classification error denoted by †), CityScapes (Cordts et al. 2016) for image segmentation (metric is here the complementary mean IoU ‡), COCO (Lin et al. 2014) for object detection (metric is here the complementary of mAP50 $\star$) and UVG (Mercat, Viitanen, and Vanne 2020) (metric is here the mean rate-distortion (bpp) $\bullet$ for a given image quality, MS-SSIM $= 0.97$).

Implementation details We evaluate our approach on many different state-of-the-art architectures: VGG-16 (Simonyan and Zisserman 2014), ResNet18 (He et al. 2016), ResNet50 (He et al. 2016), ResNet101 (He et al. 2016), ViT-b-32 (Dosovitskiy et al. 2021), MobileNetV3 (Howard et al. 2019), DeepLabV3 (Chen et al. 2018), YOLO-v5n (Jocher et al. 2022) and DVC (Lu et al. 2019). We will test the robustness of our approach using the re-synchronization success rate Ψ after applying random permutation to the penultimate layer and the four perturbations. For all of the aforementioned experiments, we have used all the traditional setups described in the respective original papers. We further notice that for the models trained on CIFAR-10, we have run 10 seeds, and average results are reported.[1]

Robustness against Gaussian noise We evaluate our methods against Gaussian noise addition with $\Omega \in [1, 10]$. The error starts increasing while Ψ remains very close to 100%. For instance, Ω valuing 6, Ψ is still equal to 100% while the error has more than doubled. Table 1 reports the results for all the architectures and the datasets. In particular, we observe that consistently for all the architectures except YOLO, when the error starts increasing, Ψ remains very close to 100%. But for YOLO, we observe the error has more than doubled while we only failed to recover a fifth of the original order.

Robustness against fine-tuning We here evaluate our method against perturbations produced by simply fine-tuning the model, adding more training complexity. Table 2 presents the results for all the architectures. In particular, we observe that consistently for all the architectures, Ψ remains equal to 100%. Despite, different experimental setups, the error on YOLO-v5n always increases: so we decided to not extend its test.

Robustness against quantization We evaluate our methods against quantization. In particular, we will evaluate the performance with $B \in [2; 16]$. Specifically, the error starts increasing around 3 bits while Ψ remains very close to 100%. A plot is provided as an Appendix. Table 3 presents the results for all architectures. Remarkably, for most of the architectures, including YOLO and ViT, Ψ remains close to 100% despite the error being extremely high.

Robustness against magnitude pruning We evaluate our methods against magnitude pruning $T \in [90; 99]\%$ of the aimed layer. The error starts increasing while Ψ remains very close to 100%. Table 4 shows good robustness for most of the architectures. For the ResNet and YOLO structures, Ψ decrease before having a huge increase of the error, but, even if the aimed layer is fully pruned the error rate remains below 50% and 70% respectively: this is due to the residual connections. For both, we did additional experiments applying global pruning to the model and the performances are similar to the other architectures.

Application to white-box watermarking Watermarking of neural networks is increasingly considered an important problem with many practical applications (the challenge of watermarking ChatGPT or assessing the integrity of unmanned vehicles). Currently, the white-box watermarking literature fails to be robust against permutation attacks. Fig. 7 shows the correlation (evaluated as Pearson correlation coefficient) of a white-box watermark when employing a state-of-the-art approach (Uchida et al. 2017). Uchida et al.'s approach is considered one of the first white-box watermarking methods, where a regularization term is added

[1]the source code is available at https://github.com/carldesousatrias/FindtheLady

Table 1: Robustness to Gaussian noise addition.

Ω		**CIFAR10**		**ImageNet**				**Cityscapes**	**COCO**	**UVG**
		VGG16	ResNet18	ResNet50	ResNet101	ViT-b-32	MobileNetV3	DeepLab-v3	YOLO-v5n	DVC
0	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.96*†±0.19	*7.03*†±0.28	*23.85*†	*22.63*†	*24.07*†	*25.95*†	*32.26*‡	*48.70**	*0.177*$^\bullet$
1	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	75.69	100
	metric$(\downarrow)$	*10.25*†±0.17	*7.30*†±0.08	*24.73*†	*23.17*†	*24.08*†	*40.44*†	*32.16*‡	*79**	*0.177*$^\bullet$
2	$\Psi(\uparrow)$	100±0	100±0	99.56	99.26	99.64	100	100	57.65	100
	metric$(\downarrow)$	*11.82*†±0.17	*9.13*†±0.59	*27.68*†	*25.08*†	*24.77*†	*70.50*†	*32.75*‡	*82.10**	*0.177*$^\bullet$
7	$\Psi(\uparrow)$	99.88±0.12	99.90±0.10	57.28	39.45	99.64	85.31	41.02	8.24	100
	metric$(\downarrow)$	*41.27*†±2.11	*99.55*†±6.30	*66.97*†	*60.49*†	*60.39*†	*98.91*†	*38.30*‡	*99.62**	*0.180*$^\bullet$
10	$\Psi(\uparrow)$	93.50±0.98	94.9±0.80	12.93	12.74	99.22	43.05	12.89	5.49	100
	metric$(\downarrow)$	*56.62*†±2.31	*75.18*†±5.14	*92.65*†	*83.04*†	*83.99*†	*99.41*†	*39.74*‡	*99.23**	*0.182*$^\bullet$

Table 2: Robustness to fine-tuning.

Θ		**CIFAR10**		**ImageNet**				**Cityscapes**	**COCO**	**UVG**
		VGG16	ResNet18	ResNet50	ResNet101	ViT-b-32	MobileNetV3	DeepLab-v3	YOLO-v5n	DVC
2	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.97*†±0.20	*6.96*†±0.11	*23.35*†	*22.54*†	*24.08*†	*26.60*†	*34.85*‡	*47.40**	*0.184*$^\bullet$
6	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.96*†±0.19	*6.98*†±0.15	*23.23*†	*22.57*†	*24.08*†	*26.41*†	*31.58*‡	*46.20**	*0.179*$^\bullet$
8	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.88*†±0.24	*7.01*†±0.19	*23.21*†	*22.54*†	*24.07*†	*26.37*†	*30.35*‡	*46.40**	*0.179*$^\bullet$
10	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.89*†±0.16	*6.94*†±0.13	*23.13*†	*22.49*†	*24.07*†	*26.31*†	*29.64*‡	*46.00**	*0.178*$^\bullet$

Table 3: Robustness to quantization.

B		**CIFAR10**		**ImageNet**				**Cityscapes**	**COCO**	**UVG**
		VGG16	ResNet18	ResNet50	ResNet101	ViT-b-32	MobileNetV3	DeepLab-v3	YOLO-v5n	DVC
16	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.96*†±0.19	*7.03*†±0.28	*23.85*†	*22.63*†	*24.08*†	*25.96*†	*32.26*‡	*48.70**	*0.177*$^\bullet$
8	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	98.44
	metric$(\downarrow)$	*9.97*†±0.20	*7.05*†±0.27	*23.91*†	*22.70*†	*24.10*†	*26.00*†	*31.49*‡	*48.90**	*0.338*$^\bullet$
6	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	98.44
	metric$(\downarrow)$	*9.98*†±0.17	*7.14*†±0.27	*26.99*†	*25.12*†	*29.55*†	*42.91*†	*32.26*‡	*89.10**	*0.823*$^\bullet$
4	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	85.49	98.44
	metric$(\downarrow)$	*10.77*†±0.28	*7.76*†±0.26	*99.91*†	*99.99*†	*96.81*†	*99.91*†	*47.28*‡	*100**	$\infty^\bullet$
2	$\Psi(\uparrow)$	100±0	100±0	31.54	9.91	100	100	100	56.47	98.44
	metric$(\downarrow)$	*87.73*†±5.56	*88.20*†±2.77	*99.9*†	*99.9*†	*99.81*†	*99.9*†	*96.63*‡	*100**	$\infty^\bullet$

Table 4: Robustness to magnitude pruning.

T		**CIFAR10**		**ImageNet**				**Cityscapes**	**COCO**	**UVG**
		VGG16	ResNet18	ResNet50	ResNet101	ViT-b-32	MobileNetV3	DeepLab-v3	YOLO-v5n	DVC
0	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.96*†±0.19	*7.03*†±0.28	*23.85*†	*22.63*†	*24.07*†	*25.95*†	*32.26*‡	*48.70**	*0.177*$^\bullet$
0.91	$\Psi(\uparrow)$	100±0	100±0	98.87	98.34	100	100	100	65.88	100
	metric$(\downarrow)$	*10.87*†±0.22	*9.67*†±0.95	*26.30*†	*23.94*†	*25.06*†	*40.00*†	*33.11*‡	*50**	*0.188*$^\bullet$
0.95	$\Psi(\uparrow)$	100±0	100±0	97.61	97.12	100	100	100	51.76	100
	metric$(\downarrow)$	*12.11*†±0.45	*12.88*†±1.90	*28.35*†	*24.58*†	*25.60*†	*48.73*†	*33.75*‡	*51.10**	*0.191*$^\bullet$
0.98	$\Psi(\uparrow)$	100±0	100±0	93.12	91.94	99.83	97.91	99.61	36.47	100
	metric$(\downarrow)$	*17.53*†±1.28	*21.71*†±5.39	*30.88*†	*25.65*†	*26.03*†	*63.48*†	*34.47*‡	*52.60**	*0.198*$^\bullet$
0.99	$\Psi(\uparrow)$	99.90±0.13	99.63±0.31	81.10	78.81	95.37	86.46	90.24	25.88	100
	metric$(\downarrow)$	*26.71*†±2.84	*28.16*†±7.35	*32.62*†	*25.98*†	*26.63*†	76.22†	*36.43*‡	*41.40**	*0.219*$^\bullet$

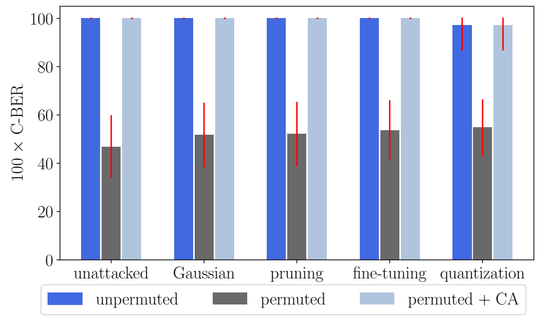
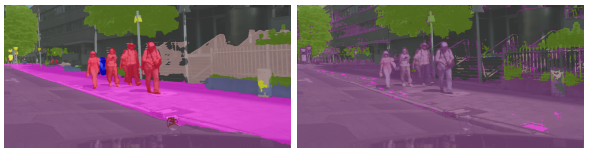


Figure 7: Robustness evaluation of the (Uchida et al. 2017)'s watermarking method against the 4 attacks.



(a) Original output. (b) Altered output.

Figure 8: Misdetection of the pedestrian induced by the scalar product modification of the weights.

to the cost function to change the distribution of one preselected layer in the model. It projects the parameter of the watermarked layer on a space a binary watermark. The order of neurons is mandatory to recover the original binary mark. We observe that permuted neurons, although not impacting the performance of the model, destroy the correlation. Applying our approach as a counter-attack (CA), we observe that we successfully retrieve the watermark and preserve the robustness, more applicative results are presented in (De Sousa Trias et al. 2023).

Integrity loss Let us here consider a counter-attack for our algorithm, on a real application: pedestrians are not detected anymore while the cosine similarity remains still equal to one (the effect in Fig. 8). To protect our method against this issue, we simply need to add a ℓ_2-norm verification between $\boldsymbol{w}_{l,i}$ and $\tilde{\boldsymbol{w}}_{l,i}$: any modification to the norm can, in this way, detected and corrected.

5 Conclusion

In this paper, we have defined and investigated one uprising question for deep learning models: is it possible to recognize parameters in a neuron after some perturbations? Is it possible to recover an original ordering for the neurons after random permutations and some perturbations?

We have explored the realm of neuron similarity, observing the parameters and outputs of different layers. We have investigated many ways to do so, observing and assessing their failure reasons. Finally, we advance a method that leverages the cosine similarity between the original layer and its permuted, perturbed version. We empirically assessed the robustness of this approach against several perturbations, for a variety of architectures and datasets.

This work has a direct impact on watermarking, where it serves as a generic counter-attack tool against parameter permutation, and has an indirect impact in various other AI domains, like pruning: as a result, neurons having perfectly correlated outputs typically have orthogonal kernels.

Acknowledgements

This work was funded in part by the Digicosme Labex through the transversal project NewEmma and by Hi!PARIS Center on Data Analytics and Artificial Intelligence.

References

Adi, Y.; Baum, C.; Cisse, M.; Pinkas, B.; and Keshet, J. 2018. Turning Your Weakness into a Strength: Watermarking Deep Neural Networks by Backdooring. In *27th USENIX Security Symposium*.

Agliari, E.; Alemanno, F.; Barra, A.; Centonze, M.; and Fachechi, A. 2020. Neural Networks with a Redundant Representation: Detecting the Undetectable. *Physical review letters*.

Chen, H.; Rouhani, B. D.; Fu, C.; Zhao, J.; and Koushanfar, F. 2019a. Deepmarks: A Secure Fingerprinting Framework for Digital Rights Management of Deep Learning Models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*.

Chen, L.-C.; Zhu, Y.; Papandreou, G.; Schroff, F.; and Adam, H. 2018. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In *Proceedings of the European conference on computer vision*.

Chen, Y.; Fan, H.; Xu, B.; Yan, Z.; Kalantidis, Y.; Rohrbach, M.; Yan, S.; and Feng, J. 2019b. Drop an Octave: Reducing Spatial Redundancy in Convolutional Neural Networks with Octave Convolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; and Schiele, B. 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

De Sousa Trias, C.; Mitrea, M.; Tartaglione, E.; Fiandrotti, A.; Cagnazzo, M.; and Chaudhuri, S. 2023. A Hitchhiker's Guide to White-Box Neural Network Watermarking Robustness. In *11th European Workshop on Visual Information Processing*.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2021. An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

Ganju, K.; Wang, Q.; Yang, W.; Gunter, C. A.; and Borisov, N. 2018. Property Inference Attacks on Fully Connected Neural Networks Using Permutation Invariant Representations. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Hecht-Nielsen, R. 1990. On the Algebraic Structure of Feedforward Network Weight Spaces. In *Advanced Neural Computers*. Elsevier.

Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. 2019. Searching for MobilenetV3. In *Proceedings of the IEEE/CVF international conference on computer vision*.

Jocher, G.; Chaurasia, A.; Stoken, A.; Borovec, J.; NanoCode012; Kwon, Y.; TaoXie; Michael, K.; Fang, J.; imyhxy; Lorna; Wong, C.; Yifu, Z.; V, A.; Montes, D.; Wang, Z.; Fati, C.; Nadar, J.; Laughing; UnglvKitDe; tkianai; yxNONG; Skalski, P.; Hogan, A.; Strobel, M.; Jain, M.; Mammana, L.; and xylieong. 2022. ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning Multiple Layers of Features from Tiny Images.

Lei, H.; Akhtar, N.; and Mian, A. 2019. Octree Guided CNN with Spherical Kernels for 3D Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Li, F.-Q.; Wang, S.-L.; and Zhu, Y. 2022. Fostering the Robustness of White-Box Deep Neural Network Watermarks by Neuron Alignment. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE.

Li, Y.; Wang, H.; and Barni, M. 2021. A Survey of Deep Neural Network Watermarking Techniques. *Neurocomputing*.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft COCO: Common Objects in Context. In *European conference on computer vision*. Springer.

Lu, G.; Ouyang, W.; Xu, D.; Zhang, X.; Cai, C.; and Gao, Z. 2019. DVC: An End-to-End Deep Video Compression Framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Mercat, A.; Viitanen, M.; and Vanne, J. 2020. UVG Dataset: 50/120fps 4K Sequences for Video Codec Analysis and Development. In *Proceedings of the 11th ACM Multimedia Systems Conference*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*.

Setiono, R.; and Liu, H. 1997. Neural-Network Feature Selector. *IEEE transactions on neural networks*.

Simonyan, K.; and Zisserman, A. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *Computing Research Repository*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The journal of machine learning research*.

Suzuki, K.; Roseboom, W.; Schwartzman, D. J.; and Seth, A. K. 2017. A Deep-Dream Virtual Reality Platform for Studying Altered Perceptual Phenomenology. *Scientific reports*.

Tartaglione, E.; Bragagnolo, A.; Odierna, F.; Fiandrotti, A.; and Grangetto, M. 2021. Serene: Sensitivity-Based Regularization of Neurons for Structured Sparsity in Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Tartaglione, E.; Grangetto, M.; Cavagnino, D.; and Botta, M. 2020. Delving in the Loss Landscape to Embed Robust Watermarks into Neural Networks. In *25th International Conference on Pattern Recognition*. IEEE.

Uchida, Y.; Nagai, Y.; Sakazawa, S.; and Satoh, S. 2017. Embedding Watermarks into Deep Neural Networks. In *Proceedings of the 2017 ACM on international conference on multimedia retrieval*.

Wan, W.; Wang, J.; Zhang, Y.; Li, J.; Yu, H.; and Sun, J. 2022. A Comprehensive Survey on Robust Image Watermarking. *Neurocomputing*.

Wang, Z.; Li, C.; and Wang, X. 2021. Convolutional Neural Network Pruning with Structural Redundancy Reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

A Table of notation

In Table 5, we present the table of notation used in the main document.

Table 5: Table of notation.

Symbol	Definition
$\boldsymbol{y}_l$	output of the l-th layer
$\boldsymbol{w}_l$	weights of the l-th layer of the model
$\langle\cdot\rangle$	the inner product
$(\cdot)^T$	the transpose operator
$\varphi(\cdot)$	the activation function
$\boldsymbol{w}_{l,c,-}$	all elements of the l-th layer, for the c-th channel
$\boldsymbol{w}_{l,-,n}$	all elements of the l-th layer, for the n-th neurons
P_{π_l}	permutation matrix (square matrix of size number of neurons)
D_{KL}	the Kullback–Leibler divergence
Ξ	the dataset the model is trained on
S_C	cosine similarity matrix
Ψ	re-synchronization success rate
Ω	the *power* of the Gaussian noise
Θ	percentage of additional training
$\boldsymbol{B}$	number of bits
$\boldsymbol{T}$	fraction of masked weights

B Toy example of permutation

In this section, we take a simple example to describe the permutation problem using a three-layer model, Fig. 9a. It is a fully connected model without bias and we observe the final output of this model: $\boldsymbol{y}_{l+1} \in \mathbb{R}^{2\times 1}$. The equation (1) become:

$$\begin{pmatrix} y_{l+1,D} \\ y_{l+1,E} \end{pmatrix} = \phi\left[\begin{pmatrix} \boldsymbol{w}_{l+1,B,D} & \boldsymbol{w}_{l+1,A,D} & \boldsymbol{w}_{l+1,C,D} \\ \boldsymbol{w}_{l+1,B,E} & \boldsymbol{w}_{l+1,A,E} & \boldsymbol{w}_{l+1,C,E} \end{pmatrix} \cdot \begin{pmatrix} y_{l,B} \\ y_{l,A} \\ y_{l,C} \end{pmatrix}\right] \tag{12}$$

where $\boldsymbol{y}_l \in \mathbb{R}^{3\times 1}$ is the input of the $l+1$-th layer, and $\boldsymbol{w}_{l+1} \in \mathbb{R}^{2\times 3}$ are the weights for the $l+1$-th layer. We can use again equation (1) to include the expression of the l-th layer in equation (12):

$$\begin{pmatrix} y_{l+1,D} \\ y_{l+1,E} \end{pmatrix} = \phi\left\{\begin{pmatrix} \boldsymbol{w}_{l+1,B,D} & \boldsymbol{w}_{l+1,A,D} & \boldsymbol{w}_{l+1,C,D} \\ \boldsymbol{w}_{l+1,B,E} & \boldsymbol{w}_{l+1,A,E} & \boldsymbol{w}_{l+1,C,E} \end{pmatrix} \cdot \phi\left[\begin{pmatrix} \boldsymbol{w}_{l,1,B} & \boldsymbol{w}_{l,2,B} \\ \boldsymbol{w}_{l,1,A} & \boldsymbol{w}_{l,2,A} \\ \boldsymbol{w}_{l,1,C} & \boldsymbol{w}_{l,2,C} \end{pmatrix}\begin{pmatrix} y_{l-1,1} \\ y_{l-1,2} \end{pmatrix}\right]\right\} \tag{13}$$

where $\boldsymbol{y}_l \in \mathbb{R}^{3\times 1}$ is the input of the l-th layer $\boldsymbol{w}_l \in \mathbb{R}^{3\times 2}$ are the weights for the l-th layer. Let us consider the case a permutation between neuron A and neuron B (π_l) is applied on the neurons of the l-th layer; the permutation matrix is

$$P_{\pi_l} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{14}$$

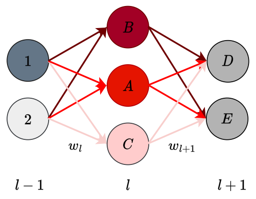
The neurons are permuted, and the ordering for the input channels $\boldsymbol{y}_l$ remains intact (Fig. 9b). Hence, the permuted output for the $l+1$-th layer will be:

$$\begin{pmatrix} y^{\pi_l}_{l+1,D} \\ y^{\pi_l}_{l+1,E} \end{pmatrix} = \phi\left\{\begin{pmatrix} \boldsymbol{w}_{l+1,B,D} & \boldsymbol{w}_{l+1,A,D} & \boldsymbol{w}_{l+1,C,D} \\ \boldsymbol{w}_{l+1,B,E} & \boldsymbol{w}_{l+1,A,E} & \boldsymbol{w}_{l+1,C,E} \end{pmatrix} \cdot \phi\left[\begin{pmatrix} \boldsymbol{w}_{l,1,A} & \boldsymbol{w}_{l,2,A} \\ \boldsymbol{w}_{l,1,B} & \boldsymbol{w}_{l,2,B} \\ \boldsymbol{w}_{l,1,C} & \boldsymbol{w}_{l,2,C} \end{pmatrix}\begin{pmatrix} y_{l-1,1} \\ y_{l-1,2} \end{pmatrix}\right]\right\} \tag{15}$$

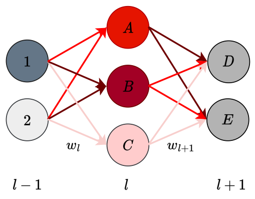
After having simply applied π_l at layer l, however, the output of the model is likely to be altered, as the propagated $\boldsymbol{y}_l^{\pi_l} \neq \boldsymbol{y}_l$, which is processed as input by the next layer. We know, however, that $\boldsymbol{y}_l^{\pi_l}$ is a permutation of $\boldsymbol{y}_l$; hence, to maintain the output of the full model unaltered, we need to also permute the weights in layer $l+1$ as

$$\begin{pmatrix} y^{\pi_l}_{l+1,D} \\ y^{\pi_l}_{l+1,E} \end{pmatrix} = \phi\left\{\begin{pmatrix} \boldsymbol{w}_{l+1,A,D} & \boldsymbol{w}_{l+1,B,D} & \boldsymbol{w}_{l+1,C,D} \\ \boldsymbol{w}_{l+1,A,E} & \boldsymbol{w}_{l+1,B,E} & \boldsymbol{w}_{l+1,C,E} \end{pmatrix} \cdot \phi\left[\begin{pmatrix} \boldsymbol{w}_{l,1,A} & \boldsymbol{w}_{l,2,A} \\ \boldsymbol{w}_{l,1,B} & \boldsymbol{w}_{l,2,B} \\ \boldsymbol{w}_{l,1,C} & \boldsymbol{w}_{l,2,C} \end{pmatrix}\begin{pmatrix} y_{l-1,1} \\ y_{l-1,2} \end{pmatrix}\right]\right\} \tag{16}$$

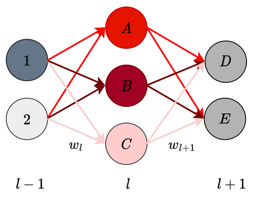
In this way, the permuted outputs in the l-th layer will be correctly weighted in the next layer, and the neural network output will be unchanged (Fig. 9c).



(a) original state



(b) neuron permutation



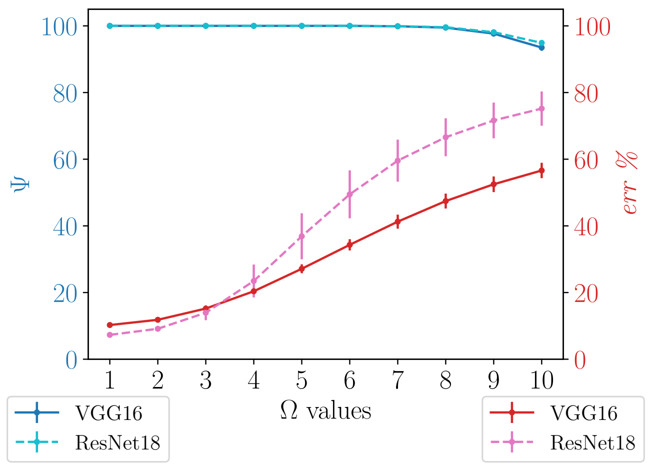
(c) channel permutation

Figure 9: Illustration of the permutation process on a fully connected neural network of three layers

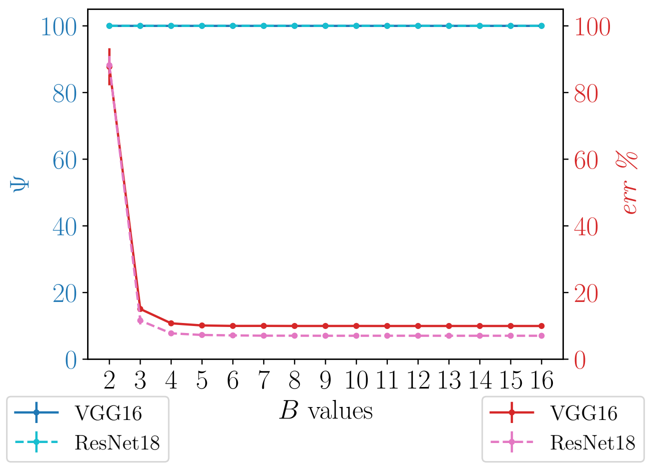
C Additional figures for the experimental section

In this section, we propose some figures, showing robustness to Gaussian noise (Fig. 10a), robustness against quantization (Fig. 10b) and robustness against pruning (Fig. 10c).

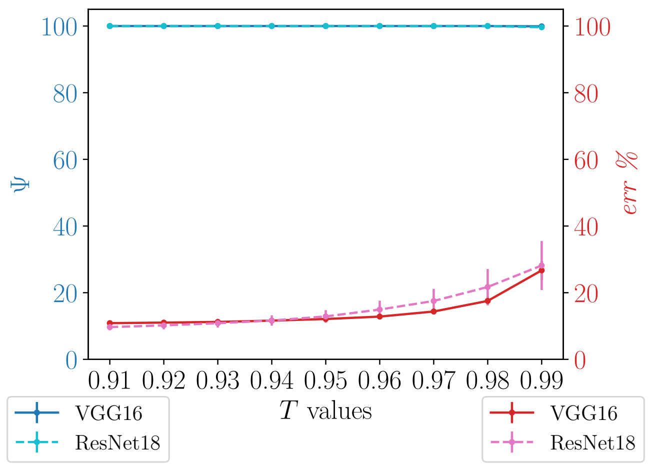
Finally, we propose a plot showing the impact of modifying the scalar product on the model's behavior (integrity attack), in Fig. 10d.



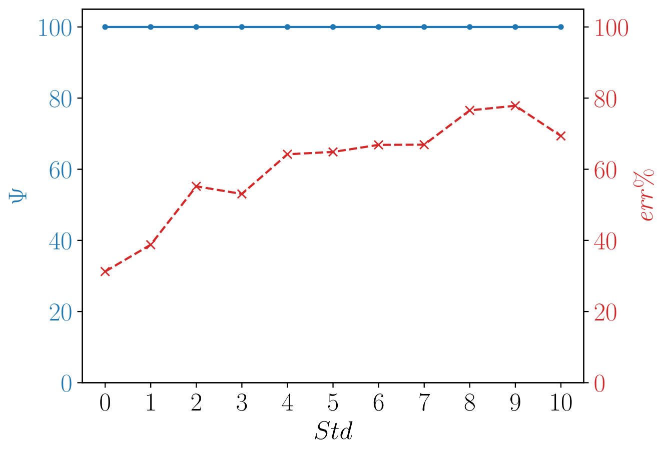
(a) Robustness against Gaussian noise addition. Ψ is on the left axis in blue and the *err* % on the right axis in red.



(b) Robustness against quantization. Ψ is on the left axis in blue and the *err* % on the right axis in red.



(c) Robustness against magnitude pruning. Ψ is on the left axis in blue and the *err* % on the right axis in red.



(d) Impact of the scalar product modification. Ψ is on the left axis in blue and the *err* % on the right axis in red.

Figure 10: Robustness analysis for VGG-16 and ResNet18 trained on CIFAR-10.

D Derivation of (9)

Let us evaluate the similarity score

$$S_C = \frac{\boldsymbol{w} \cdot \tilde{\boldsymbol{w}}}{\|\boldsymbol{w}\|_2 \cdot \|\tilde{\boldsymbol{w}}\|_2} \tag{17}$$

where we drop all the indices for abuse of notation. We can write $\tilde{\boldsymbol{w}}$ as

$$\tilde{\boldsymbol{w}} = \boldsymbol{w} + \hat{\boldsymbol{w}} \tag{18}$$

where $\hat{\boldsymbol{w}}$ is some perturbation applied to $\boldsymbol{w}$ to get $\tilde{\boldsymbol{w}}$. We can expand (17):

$$S_C = \frac{\sum_i w_i^2 + \sum_i w_i \hat{w}_i}{\sqrt{\sum_i w_i^2} \cdot \sqrt{\sum_i (w_i + \hat{w}_i)^2}} \tag{19}$$

Since we are looking for the conditions such that $S_C = 1$, we need to look for the conditions such that

$$\begin{aligned}
\sum_i w_i^2 + \sum_i w_i \hat{w}_i = \sqrt{\sum_i w_i^2} \cdot \sqrt{\sum_i (w_i + \hat{w}_i)^2} \left(\sum_i w_i^2\right)^2 + \left(\sum_i w_i \hat{w}_i\right)^2 + 2 \cdot \left(\sum_i w_i^2\right) \cdot \left(\sum_i w_i \hat{w}_i\right) \\
= \left(\sum_i w_i^2\right) \cdot \left(\sum_i w_i^2 + \sum_i \hat{w}_i^2 + 2 \sum_i w_i \hat{w}_i\right) \left(\sum_i w_i^2\right)^2 + \left(\sum_i w_i \hat{w}_i\right)^2 + 2 \cdot \left(\sum_i w_i^2\right) \cdot \left(\sum_i w_i \hat{w}_i\right) = \\
= \left(\sum_i w_i^2\right)^2 + \left(\sum_i w_i^2\right) \cdot \left(\sum_i \hat{w}_i^2\right) + 2 \cdot \left(\sum_i w_i^2\right) \cdot \left(\sum_i w_i \hat{w}_i\right).
\end{aligned}$$

By simplifying, we obtain

$$\left(\sum_i w_i \hat{w}_i\right)^2 = \left(\sum_i w_i^2\right) \cdot \left(\sum_i \hat{w}_i^2\right),$$

finding back (10).

E Derivation of (13)

Let us assume the input follows a gaussian distribution, with mean $\boldsymbol{\mu}$ and covariance matrix Σ. We know that the post-synaptic potential, still follows a gaussian distribution, having

$$\mu_z = \sum_i w_i \mu_i$$

$$\sigma_z^2 = \sum_i w_i \left(w_i \Sigma_{ii} + 2 \sum_{i'<i} w_{i'} \Sigma_{ii'}\right)$$

When introducing a perturbation $\hat{\boldsymbol{w}} = k \cdot \boldsymbol{w}$ having k scalar, we know

$$\begin{aligned}
\tilde{\mu}_z =& \mu_z + \sum_i \hat{w}_i \mu_i = \mu_z + \hat{\mu}_z = (1+k)\mu_z \\
\tilde{\sigma}_z^2 =& \sum_i (1+k) w_i \left((1+k) w_i \Sigma_{ii} + 2 \sum_{i'<i} (1+k) w_{i'} \Sigma_{ii'}\right) = (1+k) \sum_i w_i \left(w_i \Sigma_{ii} + k w_i \Sigma_{ii} + 2 \sum_{i'<i} w_{i'} \Sigma_{ii'} + k w_{i'} \Sigma_{ii'}\right) \\
=& (1+k) \left\{ \left[\sum_i w_i \left(w_i \Sigma_{ii} + 2 \sum_{i'<i} w_{i'} \Sigma_{ii'}\right)\right] + k \left[\sum_i w_i \left(w_i \Sigma_{ii} + 2 \sum_{i'<i} w_{i'} \Sigma_{ii'}\right)\right]\right\} = (1+k)^2 \sigma_z^2
\end{aligned}$$

Hence, we can write the KL divergence

$$D_{\text{KL}}(z||\tilde{z}) = \log(1+k) + \frac{\sigma_z^2 + k^2 \mu_z^2}{2(1+k)^2 \sigma_z^2} - \frac{1}{2}$$

F Derivation of (14)

In the specific case of employing a ReLU activation, assuming $\mu_z = 0$ we know that

$$\begin{cases} D_{\mathrm{KL}}(y||\tilde{y}) = 0 & z \leq 0 \\ D_{\mathrm{KL}}(y||\tilde{y}) = D_{\mathrm{KL}}(z||\tilde{z}) & z > 0 \end{cases} . \tag{20}$$

Hence, we can write the KL divergence as

$$D_{\mathrm{KL}}(y||\tilde{y}) = \int_0^{+\infty} \frac{1}{\sigma_z\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma_z^2}} (1+k)\sigma_z\sqrt{2\pi} e^{\frac{x^2}{2(1+k)^2\sigma_z^2}} dx = \frac{1}{\sigma_z\sqrt{2\pi}} \int_0^{+\infty} e^{-\frac{x^2}{2\sigma_z^2}} \left[\log(1+k) - \frac{k^2x^2}{2(1+k)^2\sigma_z^2}\right] dx$$

$$= \frac{1}{\sigma_z\sqrt{2\pi}} \frac{\sigma_z^3\sqrt{2\pi}\left[2(k+1)^2\log(k+1) - k^2 - 2k\right]}{4\sigma_z^2(k+1)^2} = \frac{2(k+1)^2\log(k+1) - k(k+2)}{(k+1)^2}$$

G Additional results

Here follow the detailed tables (all the numbers for all ranges) for all four types of modifications. For Gaussian noise addition, the main document presents 5 values $\Omega = [0, 1, 2, 7, 10]$ while Table 6 presents all the results for $\Omega \in [1, 10]$. For fine-tuning, Table 7 just add one the value compare to the main document =. For quantization, the main document presents 5 values $B = [2, 4, 6, 8, 16]$ while Table 8 presents all the results for $B \in [2; 16]$. For magnitude pruning, the main document presents 5 values $T = [0, 91, 95, 98, 99]\%$ while Table 9 presents all the results for $T \in [90; 99]\%$.

Table 6: Robustness to Gaussian noise addition.

Ω		**CIFAR10**		**ImageNet**				**Cityscapes**	**COCO**	**UVG**
		VGG16	ResNet18	ResNet50	ResNet101	ViT-b-32	MobileNetV3	DeepLab-v3	YOLO-v5n	DVC
0	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric($\downarrow$)	*9.96*†±0.19	*7.03*†±0.28	*23.85*†	*22.63*†	*24.07*†	*25.95*†	*32.26*‡	*48.70**	*0.177*$^{\bullet}$
1	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	75.69	100
	metric($\downarrow$)	*10.25*†±0.17	*7.30*†±0.08	*24.73*†	*23.17*†	*24.08*†	*40.44*†	*32.16*‡	*79**	*0.177*$^{\bullet}$
2	$\Psi(\uparrow)$	100±0	100±0	99.56	99.26	100	100	100	57.65	100
	metric($\downarrow$)	*11.82*†±0.17	*9.13*†±0.59	*27.68*†	*25.08*†	*24.77*†	*70.50*†	*32.75*‡	*82.10**	*0.177*$^{\bullet}$
3	$\Psi(\uparrow)$	100±0	100±0	98.24	97.26	100	100	100	35.29	100
	metric($\downarrow$)	*15.20*†±0.40	*13.95*†±2.24	*33.15*†	*28.60*†	*26.63*†	*88.83*†	*34.06*‡	*94.14**	*0.177*$^{\bullet}$
4	$\Psi(\uparrow)$	100±0	100±0	91.89	91.21	100	100	100	24.31	100
	metric($\downarrow$)	*20.39*†±0.88	*23.47*†±4.93	*42.09*†	*33.95*†	*30.02*†	*95.25*†	*35.73*‡	*90.49**	*0.178*$^{\bullet}$
5	$\Psi(\uparrow)$	100±0	100±0	77.34	76.12	100	99.84	92.19	16.08	100
	metric($\downarrow$)	*27.13*†±1.34	*36.88*†±6.91	*54.28*†	*41.36*†	*37.05*†	*97.46*†	*37.06*‡	*98.45**	*0.180*$^{\bullet}$
6	$\Psi(\uparrow)$	100±0	100±0	57.28	56.78	100	99.62	64.69	7.84	100
	metric($\downarrow$)	*34.32*†±1.68	*49.46*†±7.19	*66.97*†	*50.47*†	*48.25*†	*98.45*†	*37.44*‡	*99.28**	*0.182*$^{\bullet}$
7	$\Psi(\uparrow)$	99.88±0.12	99.90±0.10	57.28	39.45	99.64	85.31	41.02	8.24	100
	metric($\downarrow$)	*41.27*†±2.11	*99.55*†±6.30	*77.49*†	*60.49*†	*60.39*†	*98.91*†	*37.96*‡	*99.62**	*0.180*$^{\bullet}$
8	$\Psi(\uparrow)$	99.47±0.46	99.59±0.28	27.49	26.86	100	71.72	26.17	5.49	100
	metric($\downarrow$)	*52.48*†±2.23	*71.63*†±5.35	*84.62*†	*69.53*†	*70.79*†	*99.16*†	*38.30*‡	*99.67**	*0.187*$^{\bullet}$
9	$\Psi(\uparrow)$	97.68±0.49	98.10±0.53	19.53	18.07	100	54.37	19.12	8.24	100
	metric($\downarrow$)	*56.62*†±2.34	*75.18*†±5.14	*89.52*†	*77.11*†	*78.58*†	*99.30*†	*39.62*‡	*99.94**	*0.182*$^{\bullet}$
10	$\Psi(\uparrow)$	93.50±0.98	94.9±0.80	12.93	12.74	99.22	43.05	12.89	5.49	100
	metric($\downarrow$)	*56.62*†±2.31	*75.18*†±5.14	*92.65*†	*83.04*†	*83.99*†	*99.41*†	*39.74*‡	*99.23**	*0.182*$^{\bullet}$

Table 7: Robustness to fine-tuning.

Ω		**CIFAR10**		**ImageNet**				**Cityscapes**	**COCO**	**UVG**
		VGG16	ResNet18	ResNet50	ResNet101	ViT-b-32	MobileNetV3	DeepLab-v3	YOLO-v5n	DVC
2	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.97*†±0.20	*6.96*†±0.11	*23.35*†	*22.54*†	*24.08*†	*26.60*†	*34.85*‡	*47.40**	*0.184*$^{\bullet}$
4	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.93*†±0.24	*8.12*†±0.34	*23.22*†	*22.52*†	*24.08*†	*26.51*†	*32.09*‡	*47.20**	*0.180*$^{\bullet}$
6	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.96*†±0.19	*6.98*†±0.15	*23.23*†	*22.57*†	*24.08*†	*26.41*†	*31.58*‡	*46.20**	*0.179*$^{\bullet}$
8	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.88*†±0.24	*7.01*†±0.19	*23.21*†	*22.54*†	*24.07*†	*26.37*†	*30.35*‡	*46.40**	*0.179*$^{\bullet}$
10	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.89*†±0.16	*6.94*†±0.13	*23.13*†	*22.49*†	*24.07*†	*26.31*†	*29.64*‡	*46.00**	*0.178*$^{\bullet}$

Table 8: Robustness to quantization.

Ω		**CIFAR10**		**ImageNet**				**Cityscapes**	**COCO**	**UVG**
		VGG16	ResNet18	ResNet50	ResNet101	ViT-b-32	MobileNetV3	DeepLab-v3	YOLO-v5n	DVC
16	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	error $(\downarrow)$	*9.96*†±0.19	*7.03*†±0.28	*23.85*†	*22.63*†	*24.09*†	*25.96*†	*32.03*‡	*48.70**	*0.177*$^{\bullet}$
15	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	error $(\downarrow)$	*9.97*†±0.19	*7.04*†±0.28	*23.86*†	*22.62*†	*24.09*†	*25.94*†	*31.61*‡	*48.70**	*0.177*$^{\bullet}$
14	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	error $(\downarrow)$	*9.96*†±0.19	*7.03*†±0.28	*23.85*†	*22.64*†	*24.09*†	*25.94*†	*31.63*‡	*48.70**	*0.177*$^{\bullet}$
13	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	error $(\downarrow)$	*9.96*†±0.19	*7.03*†±0.28	*23.84*†	*22.63*†	*24.09*†	*25.93*†	*31.63*‡	*48.70**	*0.177*$^{\bullet}$
12	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	error $(\downarrow)$	*9.97*†±0.19	*7.04*†±0.28	*23.84*†	*22.66*†	*24.10*†	*25.95*†	*31.73*‡	*48.90**	*0.177*$^{\bullet}$
11	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	error $(\downarrow)$	*9.98*†±0.18	*7.03*†±0.28	*23.84*†	*22.72*†	*24.10*†	*25.99*†	*31.89*‡	*48.90**	*0.179*$^{\bullet}$
10	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	error $(\downarrow)$	*9.97*†±0.19	*7.04*†±0.28	*23.84*†	*22.72*†	*24.10*†	*25.99*†	*31.26*‡	*49.10**	*0.178*$^{\bullet}$
9	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	error $(\downarrow)$	*9.97*†±0.18	*7.03*†±0.27	*23.92*†	*22.69*†	*24.10*†	*26.00*†	*30.85*‡	*47.60**	*0.203*$^{\bullet}$
8	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	98.44
	error $(\downarrow)$	*9.97*†±0.20	*7.05*†±0.27	*24.03*†	*23.11*†	*24.10*†	*26.37*†	*31.84*‡	*48.90**	*0.338*$^{\bullet}$
7	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	error $(\downarrow)$	*10.00*†±0.20	*7.06*†±0.29	*24.42*†	*25.11*†	*24.13*†	*27.93*†	*32.46*‡	*56.40**	*2.20*$^{\bullet}$
6	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	98.44
	error $(\downarrow)$	*9.99*†±0.17	*7.14*†±0.27	*26.98*†	*25.12*†	*29.55*†	*42.91*†	*32.03*‡	*89.10**	*0.823*$^{\bullet}$
5	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	98.44
	error $(\downarrow)$	*10.13*†±0.28	*7.27*†±0.39	*97.19*†	*99.17*†	*75.83*†	*94.82*†	*39.40*‡	*99.90**	$\infty^{\bullet}$
4	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	85.49	98.44
	error $(\downarrow)$	*10.77*†±0.28	*7.76*†±0.26	*99.91*†	*99.90*†	*96.81*†	*99.91*†	*47.28*‡	*100**	$\infty^{\bullet}$
3	$\Psi(\uparrow)$	100±0	100±0	100	96.48	100	100	100	61.18	89.06
	error $(\downarrow)$	*15.07*†±0.83	*11.61*†±1.19	*99.90*†	*99.90*†	*96.71*†	*99.89*†	*89.93*‡	*100**	$\infty^{\bullet}$
2	$\Psi(\uparrow)$	100±0	100±0	31.54	9.91	100	100	100	56.47	98.44
	error $(\downarrow)$	*87.73*†±5.56	*88.20*†±2.77	*99.90*†	*99.90*†	*99.81*†	*99.90*†	*96.63*‡	*100**	$\infty^{\bullet}$

Table 9: Robustness to magnitude pruning.

Ω		**CIFAR10**		**ImageNet**				**Cityscapes**	**COCO**	**UVG**
		VGG16	ResNet18	ResNet50	ResNet101	ViT-b-32	MobileNetV3	DeepLab-v3	YOLO-v5n	DVC
0	$\Psi(\uparrow)$	100±0	100±0	100	100	100	100	100	100	100
	metric$(\downarrow)$	*9.96*†±0.19	*7.03*†±0.28	*23.85*†	*22.63*†	*24.07*†	*25.95*†	*32.26*‡	*48.70**	*0.177*$^\bullet$
0.91	$\Psi(\uparrow)$	100±0	100±0	98.63	98.34	100	100	100	65.88	100
	metric$(\downarrow)$	*10.87*†±0.22	*9.67*†±0.95	*26.57*†	*23.94*†	*25.06*†	*40.00*†	*33.11*‡	*50**	*0.188*$^\bullet$
0.92	$\Psi(\uparrow)$	100±0	100±0	98.43	98.24	100	100	100	61.57	100
	metric$(\downarrow)$	*11.03*†±0.33	*10.19*†±1.17	*26.85*†	*24.14*†	*25.21*†	*41.62*†	*33.40*‡	*50.10**	*0.188*$^\bullet$
0.93	$\Psi(\uparrow)$	100±0	100±0	98.29	97.94	100	100	100	59.61	100
	metric$(\downarrow)$	*11.23*†±0.35	*10.84*†±1.31	*27.38*†	*24.20*†	*25.25*†	*43.18*†	*33.58*‡	*50.20**	*0.192*$^\bullet$
0.94	$\Psi(\uparrow)$	100±0	100±0	96.97	96.48	100	100	100	56.47	100
	metric$(\downarrow)$	*11.63*†±0.40	*11.65*†±1.56	*27.80*†	*24.39*†	*25.41*†	*45.75*†	*33.78*‡	*50.80**	*0.189*$^\bullet$
0.95	$\Psi(\uparrow)$	100±0	100±0	97.61	97.12	100	100	100	51.76	100
	metric$(\downarrow)$	*12.11*†±0.45	*12.88*†±1.90	*28.35*†	*24.58*†	*25.60*†	*48.73*†	*33.75*‡	*51.10**	*0.191*$^\bullet$
0.96	$\Psi(\uparrow)$	100±0	100±0	96.97	96.48	100	100	100	47.05	100
	metric$(\downarrow)$	*13.84*†±0.51	*14.94*†±2.67	*29.02*†	*24.89*†	*25.74*†	*52.49*†	*33.98*‡	*51.60**	*0.192*$^\bullet$
0.97	$\Psi(\uparrow)$	100±0	100±0	96.09	94.92	100	100	100	40.78	100
	metric$(\downarrow)$	*14.36*†±0.85	*17.52*†±3.62	*29.81*†	*25.23*†	*26.03*†	*57.67*†	*34.62*‡	*51.90**	*0.205*$^\bullet$
0.98	$\Psi(\uparrow)$	100±0	100±0	93.12	91.94	99.83	97.91	99.61	36.47	100
	metric$(\downarrow)$	*17.53*†±1.28	*21.71*†±5.39	*30.88*†	*25.65*†	*26.03*†	*63.48*†	*34.47*‡	*52.60**	*0.198*$^\bullet$
0.99	$\Psi(\uparrow)$	99.90±0.13	99.63±0.31	81.10	78.81	95.37	86.46	90.24	25.88	100
	metric$(\downarrow)$	*26.71*†±2.84	*28.16*†±7.35	*32.62*†	*25.98*†	*26.63*†	76.22†	*36.43*‡	*41.40**	*0.219*$^\bullet$