FATTY AND SKINNY: A JOINT TRAINING METHOD OF WATERMARK ENCODER AND DECODER

Anonymous authors
Paper under double-blind review

ABSTRACT

Watermarks have been used for various purposes. Recently, researchers started to look into using them for deep neural networks. Some works try to hide attack triggers on their adversarial samples when attacking neural networks and others want to watermark neural networks to prove their ownership against plagiarism. Implanting a backdoor watermark module into a neural network is getting more attention from the community. In this paper, we present a general purpose encoder-decoder joint training method, inspired by generative adversarial networks (GANs). Unlike GANs, however, our encoder and decoder neural networks cooperate to find the best watermarking scheme given data samples. In other words, we do not design any new watermarking strategy but our proposed two neural networks will find the best suited method on their own. After being trained, the decoder can be implanted into other neural networks to attack or protect them (see Appendix for their use cases and real implementations). To this end, the decoder should be very tiny in order not to incur any overhead when attached to other neural networks but at the same time provide very high decoding success rates, which is very challenging. Our joint training method successfully solves the problem and in our experiments maintain almost 100% encoding-decoding success rates for multiple datasets with very little modifications on data samples to hide watermarks. We also present several real-world use cases in Appendix.

1 INTRODUCTION

Security issues of deep learning have been very actively being studied. It had been already demonstrated that deep learning methods are vulnerable to some carefully devised adversarial attacks (Krizhevsky et al., 2012; Graves et al., 2013; Kim, 2014; Chen et al., 2015). At the same time, many researchers are also studying about how to make them more robust against such attacks. A couple of recent works, for example, proposed to use watermarks (Zhang et al., 2018; Adi et al., 2018) to protect neural networks. At the same time, other work wanted to use a similar watermark technique to attack neural networks (Liu et al., 2018).

The method of adding watermarks to data samples can be used in various ways to protect deep learning models. First, the decoder can be implanted into a trained deep learning model and later one can prove the ownership, when other people copied the model, by showing that the copied model reacts to one's watermarked samples. Second, the implanted decoder may allow only legitimately watermarked samples and reject other non-watermarked samples. In this case, only people that have the encoder can access the deep learning model. However, there is one very strict requirement that the decoder should be tiny to minimize the incurred overheads by attaching it as part of the main deep learning model. Similar techniques can also be used to attack neural networks.

In this paper, we do not propose any specific watermarking techniques. Instead, we want the encoder and decoder discuss and decide their watermarking method. Inspired from generative adversarial networks (GANs) (Goodfellow et al., 2014), the encoder and decoder work for the same goal and are jointly trained. They do not perform the adversarial game of GANs. Their relationship is rather cooperative than adversarial in our method. The decoder is a tiny neural network to decode watermarks and the encoder is a high-capacity neural network that can watermark samples in such a way that the tiny neural network can successfully decode. Therefore, those two neural networks should cooperate to find such a watermarking scheme — in GANs, one neural network (generator)

tries to fool the other neural network (discriminator). Because the decoder has a limited capacity due to its tiny neural network size, the encoder should not decide the watermarking scheme alone. The encoder should receive feedback from the decoder to revise its watermarking scheme. After training them, one should keep the encoder in a secure place but can deploy the decoder to as many places as one wants. We also show that our method can be used for both defences and attacks (refer to Appendix for some of these examples we implemented using our proposed method).

We adopt residual blocks (He et al., 2016) to design the encoder. Each residual block of the encoder is supposed to learn $f(x)+x$ where x is an input to the block. One can consider $f(x)$ as a watermark signal discovered by the joint training of the encoder and the decoder. The signal produced by $f(x)$ should be strong enough to be detected by the decoder but weak enough not to be detected by human eyes. We design our training loss definition to achieve this goal. The encoder should modify original samples to implant watermarks. As more modifications are allowed, stronger watermarks will be implanted but they can be readily detected by human eyes. Our loss definition has a parameter that can be set by user to limit the modifications by the encoder. Our experiments show that we can find a well-balanced watermarking scheme that be detected only by the decoder.

We tested many different datasets: face recognition(VGG-Face Data-set), speech recognition (Pannous, 2016), images with general objects (Krizhevsky et al., 2012), and flowers (Flowers Data-set). Two of them are reported in the main paper with the comparison with other watermarking methods and others are introduced in Appendix. During experiments, our methods marked 100% decoding success rates for all datasets (in at least one hyper-parameter configuration). This well outperforms other baseline methods.

In addition, we also found that different watermarking schemes are trained for different datasets. For instance, the encoder modified the tone of colors for the face recognition images. For the general object images, however, the encoder explicitly marks some dots rather than modifying their color tones (see Figure 3 and Figure 4). This proves our goal that two neural networks cooperate to find the best suited watermarking method for each dataset.

2 Related Work

2.1 Watermarking Techniques for Defenders

Watermarking data samples, such as images, videos, etc., is a long-standing research problem. In many cases, watermarking systems merge a specific watermark signal s (set by user) and a data sample x to produce a watermarked sample x', i.e., $x' = encode(x, s)$ and $s = decode(x')$. In general, the signal s is secret and later used to check where the watermarked sample x' is originated from.

There exist many different watermarking techniques for relational databases, images, videos, and so forth. However, watermarking deep neural networks is still under-explored except for a couple of recent papers (Zhang et al., 2018; Adi et al., 2018). For instance, one can implant a certain signal on neural network weights – technically, this is similar to implanting a signal on a column of table for watermarking a relational database. However, the signal on the weights will disappear after fine-tuning the neural network which can preserve its accuracy but reorganize its weight values. Instead, we need a more robust way for watermarking neural networks.

To this end, a backdoor based watermarking method has been recently proposed (Zhang et al., 2018; Adi et al., 2018). In general, a backdoor means a certain malware piece that can be exploited to avoid authentication processes in computer security. In their contexts, however, a neural network backdoor means a way to control the final prediction of a target neural network — for instance, retraining a target neural network so that it classifies a certain type of cats as dogs. The authors want to use the backdoor mechanism to protect the ownership of a neural network. Because the backdoor reacts to the samples specially watermarked by the owner, the proof of its ownership is available when other people copied the neural network. Of course, if the backdoor is successfully identified and removed, the proof of the ownership is not possible. However, this incurs additional costs and greatly decreases the motivation of copying the model.

2.2 WATERMARKING TECHNIQUES FOR ATTACKERS

The same watermarking technique can be used for attacks. In (Liu et al., 2018), the attacker implants an attack trigger into a data sample using a simple watermarking technique and the target neural network is already compromised by the attack to make it react to their trigger. Their goal is to induce the compromised neural network outputs a certain label encoded in the attack trigger and preferred by the attacker. Because this paper uses a very strong watermark signal, their watermarked images are visually impaired. Due to its strong watermarks, however, their attack shows very high success rates.

2.3 GENERATIVE ADVERSARIAL NETWORKS

GANs are one of the most successful generative models. They consist of two neural networks, one generator and one discriminator. They perform the following zero-sum minimax game:

$$\min_G \max_D V(G, D) = \mathbb{E}[\log D(x)]_{x \sim p_{data}(x)} + \mathbb{E}[\log(1 - D(G(z)))]_{z \sim p(z)}, \tag{1}$$

where $p(z)$ is a prior distribution, $G(\cdot)$ is a generator function, and $D(\cdot)$ is a discriminator function whose output spans $[0, 1]$. $D(x) = 0$ (resp. $D(x) = 1$) indicates that the discriminator D classifies a sample x as *generated* (resp. *real*).

The generator tries to obfuscate the task of the discriminator by producing realistic fake samples. We redesign the adversarial game model for our purposes. In our case, two neural networks, one encoder and one decoder, perform a cooperative game.

3 MOTIVATION

A watermarking framework consists of encoder and decoder. The encoder modifies original samples by adding a watermark signal into them and the decoder is a binary classification to detect the presence of the watermark signal. Watermarks are used for various purposes in deep learning. They were used for both of defenses and attacks for deep neural networks. In our case, we are interested in developing a pair of encoder and decoder and the decoder should be pluggable to other neural networks (as in the malware piece or backdoor in computer security). Our encoder-decoder pair can be used for both defenses and attacks (refer to Appendix for our case studies).

There are several watermarking methods based on CNNs that can be described by $x' = encode(x, s)$ and $s = decode(x')$ (Mun et al., 2017). However, existing methods do not care about the size of the decoder and we are not interested in implanting a watermark signal s into a data sample x. We let the encoder modify x in a way that the decoder wants and the decoder performs the binary classification of watermarked or non-watermarked. Thus, our model can be described as $x' = encode(x)$ and $decode(x') \in \{0, 1\}$ without s. In real world applications, this binary classification decoder suffices (refer to our use cases in Appendix) and the decoder should be so tiny that it does not incur any overheads when attached to other main neural networks — this is a strong requirement especially for the backdoor based watermarking method.

Our goal is to develop a watermarking framework that consists of one large encoder (a fatty network) and one tiny decoder (a skinny network) and they should decide their own watermarking scheme without human efforts.

4 PROPOSED METHOD

Our overall idea is greatly inspired by generative adversarial networks (GANs). In GANs, there are two neural networks, generator and discriminator, that are comparable to each other in terms of their neural network capacity. In our case, however, the encoder and decoder are highly imbalanced in their neural network capacity and they perform a cooperative game (rather than the zero-sum adversarial game of GANs).

In our method, the encoder should be capable of generating simple but robust watermarked samples because the decoder has very low capacity and as a result, it may not be able to decode complicated watermarks. Therefore, the encoder should be large and trained enough to find the watermarking

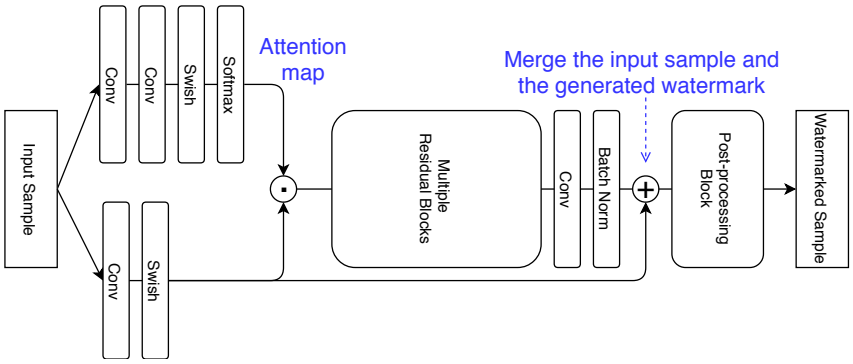

Figure 1: The proposed encoder architecture. Based on the attention map, a series of residual blocks generate a watermark signal specific to the input sample x which will be later merged with the generated watermark signal. All those convolutions in this encoder use the stride of 1 and the channel of 3 to maintain the identical input and output dimensions.

mechanism suitable for the low-capacity decoder. In other words, we do not teach any watermarking mechanism but let them discover on their own considering the neural network capacity difference.

4.1 Encoder

The encoder (comparable to the generator of GANs) should modify original samples to implant a watermark signal. We adopt residual blocks to design the encoder as shown in Figure 1. Residual blocks are proven to be effective in designing a deep architecture and adopted by many works (e.g., ResNet (He et al., 2016)). Each residual block that can be described as $x + f(x)$ is suitable to perform the watermarking task. After the multiple stages of residual blocks, the encoder generates a watermark signal[1] that will be merged with the original sample x. We use the multiple residual blocks because it is very unlikely that one residual block is able to generate a robust watermark signal. The overall watermarked sample generation process can be described as follows:

$$\begin{aligned} &x + f_1(x') + f_2(x' + f_1(x')) + f_3(x' + f_1(x') + f_2(x' + f_1(x'))) + \cdots, \\ &x' = x \odot A, \end{aligned} \tag{2}$$

where x is the original sample; A is the attention map of x produced after two convolutions, one activation, and a softmax; $\odot$ means the Hadamard product; $f_i(\cdot)$ represent an additive term by i-th residual block. In particular, we use the swish activation (Ramachandran et al., 2017). Thus, one can consider our generated watermark signal is an ensemble of all those additive terms (Veit et al., 2016). Note that our watermark signal is generated after ignoring unimportant parts of x after the element-wise product with the attention map.

After merging the input sample x and the generated watermark, we have one post-processing block to refine the watermarked sample. This process includes a couple of more convolutions.

In Figures 2, 3, and 4, we show watermarking examples for various datasets. In Figures 2 and 4, watermarks are generated for the parts where the attention map focuses on. In Figure 3, watermakrs are dispersed over many pixels and in this case, the attention also provides similar weights for those pixels.

4.2 Decoder

The decoder (comparable to the discriminator of GANs) should classify if an input sample has a watermark signal or not. We adopt the discriminator of DCGAN (Radford et al., 2015) (after shrinking its size) as decoder. Its discriminator follows a standard CNN architecture. One of its

[1] In conventional watermarking schemes, the watermark signal is set by user and the encoder tries to implant the signal into data samples. In our case, the encoder generates a latent signal that is suitable for the tiny decoder to detect. It is unlikely that a user defined watermark signal can be readily detected by the tiny decoder. We encourage that they try to find the watermark scheme on their own.

(a) Original (b) Watermark (c) Watermarked (d) Original (e) Watermark (f) Watermarked

Figure 2: Watermarking examples. (b) and (e) are generated watermarks (before being merged with images). These are cases where watermarks are generated, aided by attention. For (e), there is a watermark in the most lower right corner and its attention also focuses on the same area. However, attention maps sometimes provide similar weights over almost all pixels, in which cases watermarks are scattered over all pixels — examples in Figure 3 correspond to this case.

advantageous is that it is very hard to identity the decoder after being implanted into a neural network model because it is tiny and uses only very standard neural operators. We perform experiments by varying the number of convolution layers in order to find the smallest decoder configuration.

4.3 Training Loss

We introduce our training method. The main training loss can be described as follows:

$$\max_E \max_D V(E,D) = \mathbb{E}[\log(1 - D(x))]_{x \sim p_{data}(x)} + \mathbb{E}[\log D(E(x))]_{x \sim p_{data}(x)}, \tag{3}$$

where $E(\cdot)$ is an encoder, and $D(\cdot)$ is a decoder, and x is a data sample.

This loss definition looks similar to the one in GANs. However, we do not perform the minimax zero-sum game of GANs. Both the encoder and decoder cooperate to find the best performing watermarking scheme. Its equilibrium state analysis is rather meaningless because they do not perform the zero-sum adversarial game of GANs. It is obvious that the main loss representing equation 3 will be optimized when the decoding success rate of watermarked and non-watermarked cases is 100%. The main loss can be implemented using the cross-entropy loss as in other GANs.

In addition, we also use one more regularization term to limit the modification by the encoder. Let $\mathcal{L}$ be the main loss in equation 3. The final loss term is defined as follows:

$$\begin{aligned} \mathcal{L}_{final} =& 10^{-3}\mathcal{L} + \max(0, \mathcal{L}_{content} - \gamma), \\ \mathcal{L}_{content} =& \frac{1}{W_{i,j}H_{i,j}} \sum_{s=1}^{W_{i,j}} \sum_{t=1}^{H_{i,j}} \Big(\phi(x)_{s,t} - \phi(E(x))_{s,t}\Big)^2, \end{aligned} \tag{4}$$

where $\phi(\cdot)_{s,t}$ means the feature map taken after t-th convolution (after activation) before s-th max-pooling layer in the VGG19 network[2], γ is the maximum margin in the hinge-loss based regularization. We allow the modification up to γ. Note that the hinge-loss based regularization does not incur any loss up to γ.

$\mathcal{L}_{content}$ compares two samples, the original sample x and the watermarked sample $E(x)$, in terms of the feature maps created by the VGG19 network (Liu & Deng, 2015). We found that this is better than the pixel-wise mean squared error regularization. In our case, we add the hinge-loss to control the modification of the input sample x. If γ is large, more modifications are allowed and as a result, our watermark signals will be more robust. However, the modified sample can be very different from the original sample in this case, which is not a desired result. Therefore, γ should be adjusted very carefully.

Our training algorithm is similar to that of GANs. The encoder and the decoder are alternately trained to collaboratively minimize $\mathcal{L}_{final}$. We omit the detailed algorithm due to its similarity to the training algorithm of GANs.

[2]In this paper, we consider only images and other data types that can be represented by matrix and will extend to other types of data samples in the future. For other data types, one can choose a proper neural network instead of the VGG19 network.

5 EXPERIMENTS

We first select several neural networks and their official datasets, considering the diversity in their task and dataset types. After that, we train the encoder-decoder network using 80% of training samples and check the decoding error rate for the remaining 20% of testing samples. For this, we test both cases where each testing sample is watermarked or not — i.e., the decoder should successfully distinguish watermarked and non-watermarked cases for the same set of samples. By varying the number of convolution layers in the decoder and the margin γ, we repeat the experiment.

We also test how much damage those implanted watermarks introduce to data samples. If the watermark signal is weak, there should not be any differences between them for several popular image comparison metrics. We introduce detailed experiment results for two neural networks in this paper and some more in Appendix.

5.1 BASELINE METHODS

To evaluate our method, we compare with the following watermarking techniques. Note that our baseline selection is so extensive that all different types of watermarking methods are included.

1. In the statistical watermarking method (SWM) introduced in (Shehab et al., 2008), authors proposed a method to hide a series of bits (set by user) in a column of table — after flattening an image to an array of pixels, this method can be applied. It explicitly solves an optimization problem to find the weakest watermark (enough to hide the bits) and performs some statistical tests to decode the watermarked bit pattern. We test the following two bit patterns to hide: '0101010101', and '0000100001'. This method cannot be implemented by neural networks but we use this method only for comparison purposes.
2. Trojan in (Liu et al., 2018) uses a relatively stronger watermark signal, called *attack trigger* in their paper, than SWM. This papers proposed a very effective backdoor attack method and our motivation is also influenced by the paper.

5.2 DATASETS AND NEURAL NETWORKS

We choose the following neural networks and their datasets. All selected neural networks include their official datasets and we use them.

1. Face Recognition (FR): FR is a deep CNN-based neural network developed by (Parkhi et al., 2015) as VGG-FACE. It has 16 layers and its data-set is available at (VGG-Face Data-set).
2. Speech Recognition (SR): A CNN model proposed in (Pannous, 2016) is to recognize spoken languages. It achieves superhuman performance in recognizing spoken numbers. It uses the dataset of pulse-code modulation (PCM) images of spoken numbers.

We also tested for the ImageNet (Krizhevsky et al., 2012) and Flowers (Flowers Data-set) datasets. Experiments for those other neural networks and datasets are in Appendix.

5.3 FACE RECOGNITION NEURAL NETWORK

We report i) how many non-watermarked and watermarked samples are correctly recognized and ii) how much damage each watermarking method brings to data samples in each method.

5.3.1 WATERMARK DECODING SUCCESS RATE

We compare the proposed method with the aforementioned baseline methods. Our method (decoder size = 3 and $\gamma = 0.01$) marks the best decoding success rate, i.e., 100% in our method vs. 95.5% in the method of (Liu et al., 2018) vs. 89.3% in the statistical watermarking method. Other configurations in our method also outperform all the baseline methods.

Table 1: The decoding success rate and the damage on samples for the face recognition neural network and its dataset. The decoder size means the number of convolutions in the decoder. The best results are indicated in bold font. The 100% decoding success rate means that the decoder can distinguish non-watermarked and watermarked samples without any mistakes.

Metric	SWM (0101010101)	SWM (0000100001)	Trojan	Our method								
				Decoder Size = 1			Decoder Size = 2			Decoder Size = 3		
				$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$	$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$	$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$
Decoding Success Rate	87.0%	89.3%	95.5%	98.2%	99.9%	99.3%	99.9%	99.9%	97.0%	**100.0%**	98.7%	98.3%
MS-SSIM	0.9998	0.9998	0.91818	0.9976	0.9917	0.9574	0.99782	0.9901	0.9744	0.9960	0.9870	0.98296
PSNR	58.479	58.851	21.029	37.735	31.3	22.797	39.852	30.8802	26.591	36.580	28.755	27.955
Watermarked Entropy / Non-watermarked Entropy	1.0000	1.0000	1.0005	1.0006	0.9993	1.0008	0.9998	0.9982	1.0004	0.9970	1.0004	0.9972

(a) Our method ($\gamma = 0.01$) (b) Our method ($\gamma = 0.05$) (c) Our method ($\gamma = 0.1$) (d) Trojan (e) Original Image

(f) Our method ($\gamma = 0.01$) (g) Our method ($\gamma = 0.05$) (h) Our method ($\gamma = 0.1$) (i) Trojan (j) Original Image

Figure 3: Examples of watermarking FR images. (d) and (i) are watermarked by the method of (Liu et al., 2018). Others are watermarked by our method. The decoder has 3 convolution layers in these examples. Note that there are more modifications on the color tone of images as γ increases. For all cases, the trained decoder can successfully decode their watermarks. Refer to Appendix for examples of watermarking other samples.

5.3.2 Damage on Data Samples

Sometimes watermarks incur irreparable damage on images, and as a result, its contents are changed a lot. We visualize watermarked samples and measure the difference from their original images using the multi-scale structural similarity (MS-SSIM), the peak signal to noise ratio (PSNR), and the Shannon entropy increase after watermarked.

Figure 3 shows several original and watermarked samples. Figure 3 (d) and (i) are watermarked by the method of (Liu et al., 2018) and their attack trigger signal (in the lower right corner) is very strong. Compared to them, our methods provide much weaker watermarks. However, our decoding success rates are much higher than other methods including (Liu et al., 2018). This proves the efficacy of the joint training mechanism of the encoder and decoder.

Our method is clearly better than Trojan for both PSNR and the entropy change, i.e., 36.580 vs. 21.029 for PSNR. SWM solves an optimization problem to find the best case to hide watermarks with the smallest changes. Thus, its PSNR and entropy change are better than our method and Trojan. However, SWM does not provides reliable decoding success rates.

We also checked the accuracy drop after watermarking images. With the original images, the FR network's accuracy is 0.795576 and after watermarking them with $\gamma = 0.01$ and the decoder with 3 convolutions, it becomes 0.797864. After watermarking, the accuracy is slightly improved but we

Table 2: The decoding success rate and the damage on samples for the speech recognition neural network and its dataset. The decoder size means the number of convolutions in the decoder.

Metric	SWM (0101010101)	SWM (0000100001)	Trojan	Our method								
				Decoder Size = 1			Decoder Size = 2			Decoder Size = 3		
				$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$	$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$	$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$
Decoding Success Rate	12.0%	21.0%	100%	99.9%	99.6%	**100.0%**	94.9%	99.3%	99.1%	91.8%	90.7%	97.4%
MS-SSIM	0.999	0.999	0.188	0.9687	0.9491	0.9307	0.9823	0.9480	0.9737	0.9578	0.9624	0.8793
PSNR	57.4924	57.3948	8.053	36.5606	33.5501	31.2187	37.7017	33.3228	29.5841	35.1865	33.9897	31.1498
$\frac{\text{Watermarked Entropy}}{\text{Non-watermarked Entropy}}$	1.0000	1.0000	1.0202	1.0992	0.9464	1.1274	1.1815	0.9993	1.1684	1.1618	1.1303	1.0344

think this is within its error margin and not significant. They are more or less the same. Likewise, in almost all cases, their accuracy difference is very trivial.

Other watermarking examples are in Appendix. For example, Figure 4 in Appendix shows several watermarking examples for the ImageNet dataset. Watermarks in Figure 3 and Figures 2 and 4 are very different. In Figure 3, watermarks are implanted in the tone of colors but in Figures 2 and 4, several small dots are explicitly marked. This is because the encoder and decoder networks discover a suitable watermarking method for each dataset. It is very interesting that they discover how to hide watermarks on their own.

5.4 Speech Recognition Neural Network

For this SR network netork ans dataset, we repeat the same experiments as the FR case. In general, these experiment results have the same pattern as the FR results.

5.4.1 Watermark Decoding Success Rate

In SR, SWM shows very poor decoding success rates. Both our method and Trojan provides the rate of 100%. Considering the large damage on samples by Trojan which will be shortly described, however, Trojan's 100% decoding success rate is rather meaningless. In many configurations in our method, their success rates are more than 99%.

5.4.2 Damage on Data Samples

SWM marked the smallest damage but considering it very low success rates, we don't think SWM is suitable for SR. It cannot be even implemented by neural networks. Our method introduces less damage to samples than that of Trojan. Especially, the PSNR of Trojan is much worse than other method. Because the PSNR is in the log scale, those values mean huge differences. Its MS-SSIM is also greatly damaged. Our method shows very stable values for those three metrics.

6 Conclusions

We present a joint training method of the watermark encoder and decoder. Our decoder is a very low-capacity neural network and the encoder is a very high-capacity neural network. These two skinny and fatty neural networks collaborate to find the best watermarking scheme given data samples. In particular, we use residual blocks to build the encoder because the definition of the residual block is very appropriate for the task of watermarking samples. We demonstrated that two different types of watermarks (one to change the color tone and the other to add dots) are found by them without human interventions.

For our experiments with various datasets, our method marked 100% decoding success rates, which means our tiny decoder is able to distinguish watermarked and non-watermarked samples perfectly.

We also listed three use cases in Appendix about how to utilize our proposed encoder and decoder for real-world attacks and defenses. Our future research will be to implement those use cases.

REFERENCES

Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *Proceedings of the 27th USENIX Security Symposium*, 2018.

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*, pp. 2722–2730, 2015.

Flowers Data-set. `https://github.com/tensorflow/models/tree/master/research/slim`.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*. 2014.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pp. 6645–6649. IEEE, 2013.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

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

S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015.

Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 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.

Seung-Min Mun, Seung-Hun Nam, Han-Ul Jang, Dongkyu Kim, and Heung-Kyu Lee. A robust blind watermarking using convolutional neural network. *CoRR*, abs/1704.03248, 2017.

Pannous. Speech Recognition with the Caffe Deep Learning Framework, 2016. URL `https://github.com/pannous/caffe-speech-recognition`.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, pp. 6, 2015.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017.

M. Shehab, E. Bertino, and A. Ghafoor. Watermarking relational databases using optimization-based techniques. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):116–129, Jan 2008. ISSN 1041-4347. doi: 10.1109/TKDE.2007.190668.

C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *ArXiv e-prints*, February 2016.

TensorFlow. Tensorflow graph editor. `https://www.tensorflow.org/api_guides/python/contrib.graph_editor`, 2018.

Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016.

VGG-Face Data-set. `http://www.robots.ox.ac.uk/~vgg/software/vgg_face`.

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 Asia Conference on Computer and Communications Security*, 2018.

(a) Watermarked Image ($\gamma = 0.01$) (b) Watermarked Image ($\gamma = 0.05$) (c) Watermarked Image ($\gamma = 0.1$) (d) Non-watermarked Original Image

(e) Watermarked Image ($\gamma = 0.01$) (f) Watermarked Image ($\gamma = 0.05$) (g) Watermarked Image ($\gamma = 0.1$) (h) Non-watermarked Original Image

Figure 4: Examples of watermarking ImageNet images. Some dots are marked explicitly to hide watermarks when $\gamma >= 0.05$. Recall that watermarks are hidden in the tone of colors for FR images. This is a very interesting point because our proposed method can discover two very different watermarking schemes for them. This is because adding dots does not make the regularization term greatly exceed the margin γ. When $\gamma = 0.01$, a similar watermarking scheme to the FR exmaples will be used. This proves that our method is able to fine the best suited watermarking scheme given data samples. The decoder has 3 convolution layers in these examples. Note that there are more modifications in general as γ increases. For all cases, the trained decoder can successfully decode their watermarks.

Figure 5: The decoding success rate in the ImageNet dataset. We report the decoding success rate for non-watermarked/watermarked cases with our method after varying the convolution numbers in the decoder (i.e. decoder size) and γ.

Our method					
Decoder size = 1			Decoder size = 3		
$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$	$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$
81.2%/100.0%	89.2%/100.0%	92.0%/100.0%	99.0%/100.0%	98.0%/99.4%	99.5%/100.0%

Figure 6: The decoding success rate in the Flowers dataset

Our method					
Decoder size = 1			Decoder size = 3		
$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$	$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$
94.6%/97.1.0%	96.9%/94.5%	97.9%/99.9%	100.0%/99.0%	100.0%/99.8%	100.0%/99.0%

A Additional Experiment Results

A.1 ImageNet

We introduce additional experiments that were removed from the main paper. In Table 5, we report the decoding success rate for the ImageNet dataset. In all configurations, their success rates are very high. In particular, the decoder with 3 convolution layers provides the highest decoding success rate.

Table 3: Original target neural network vs. Modified target neural network. They differ only at one part marked in red.

Original Target NN	Modified Target NN
$c_1 \leftarrow \text{relu}(\text{conv}(x))$	$c_1 \leftarrow \text{relu}(\text{conv}(x))$
$c_2 \leftarrow \text{relu}(\text{conv}(c_1))$	$c_2 \leftarrow \text{relu}(\text{conv}(c_1))$
$\vdots$	$\vdots$
$\vdots$	$c_i \leftarrow \text{relu}(\text{conv}(c_{i-1}))$ + decode&inject(x)
$\vdots$	$\vdots$
$c_n \leftarrow \text{relu}(\text{conv}(c_{n-1}))$	$c_n \leftarrow \text{relu}(\text{conv}(c_{n-1}))$
$logit \leftarrow linear(c_n)$	$logit \leftarrow linear(c_n)$
$label \leftarrow softmax(logit)$	$label \leftarrow softmax(logit)$

Figure 4 shows several watermarked and non-watermarked samples. With $\gamma = 0.01$, modifications are very limited. In Figure 4 (e), it is very hard to recognize its watermark with human eyes, but the decoder can detect its hidden watermark signal surprisingly. The smallest decoder with only one convolution works well too. However, its decoding success rates are smaller than that of the decoder with three convolutions.

A.2 Flowers

We also test the flower images in (Flowers Data-set). We choose this dataset to test with various types of images. We tested with face and object images. Flower images have different characteristics from the previous image datasets. In the decoder size of 3, the decoding success rates are very high for all γ configurations. When there is only one convolution in the decoder, the decoding success rate is proportional to γ.

B Use Cases

In this section, we introduce three use cases to both attack and protect neural networks. The first use case is to utilize the proposed encoder and decoder for backdoor attacks. The second use case is to allow only legitimately watermarked input samples and comparable to the admission control in operating systems and computer networks, and the last use case is to prove the ownership using the proposed watermarking technique.

B.1 Backdoor Attack on Neural Networks

The backdoor attack in the context of machine learning means that the attacker modifies a target model and the modified model reacts to samples specially marked by the attacker — the attacker may redistribute it after the modification, and careless users may download and use it. The special marker is called *attack trigger* and it contains a target label that is different from its ground-truth label but preferred by the attacker. The attack trigger is usually implanted using a watermarking method. We demonstrate that how the attacker utilize our encoder and decoder networks.

We first describe the proposed decode&inject module and how to attach it to the target neural network. The code snippet in the left column of Table 3 represents a typical multi-class image classification target neural network — we use this image classification neural network as an example but our attack can be applied to any other neural networks. We attach the module into one of its convolution layers as shown in the right column of the table, i.e., $c_i \leftarrow \text{relu}(\text{conv}(c_{i-1}))$ + decode&inject(x) where x is an image and c_i is feature maps in i-th convolution layer. Note that the module reads the input image x and outputs a tensor whose dimensionality is the same as that of the convolution layer if watermarked by the attacker. Thus, the role of the module is i) decoding the watermark signal and ii) injecting a signal (tensor) to the target neural network to control its final output. If not watermarked, the module should inject a zero tensor (i.e., keep silent). The module can be defined

Table 4: The attack success rate for their original testing samples. Our attack method outperforms the Trojan attack in all cases. The best results are indicated in bold font.

Target Neural Network	Trojan (Liu et al., 2018)	Our method (Decoder size = 3, $\gamma = 0.01$)
IN	N/A	**100%**
FR	95.5%	**100%**
SR	**100%**	**100%**

as follows:

$$decode\&inject(x) = \begin{cases} 0 & \text{if no watermark} \\ w & \text{if watermark exists on } x \end{cases} \tag{5}$$

It should outputs 0 if no watermark, i.e., x is a non-modified image. Because of this, the module has *zero influences* on the target neural network for non-modified images. If x has a certain watermark, it should output a corresponding tensor w for the label preferred by the attacker. For instance, all watermarked images with cats can be classified as dogs with the additional feature map w injected to the target neural network.

All the convolution, linear, softmax are initialized and fixed with the weights of the original target neural network and our trained decoder, and we train only w. After being trained, the module can inject a trained feature map w that is able to control the final softmax outputs, i.e., class labels.

The implementation of $decode\&inject(x)$ is very straightforward. On top of the proposed decoder, one trick to implement an if-else statement is enough to make the module fully functioning. We attacked the neural networks of FR and SR, and the following one more using the proposed backdoor attack mechanism based on our encoder-decoder neural networks.

Inception-v4 Network (IN) Inception-v4 Network is a CNN-based classifier developed by Google (Szegedy et al., 2016). It uses inception modules to make training very deep networks very efficient. We use the Flowers dataset released in (Flowers Data-set).

To evaluate the proposed attack, we followed the steps used in (Liu et al., 2018). A backdoor modification proposed by (Liu et al., 2018) makes the target neural network react to their watermarked attack trigger and output the preferred label by the attacker — this is the same as our decode&inject module. We first prepare the modified target neural network where the decode&inject module is attached[3]. To perform attacks, we use the original testing set for each target neural network. Each sample is attacked multiple times for all non-ground-truth labels. Their attack success rates are summarized in Table 4. As you see, our method provides better success rates than the other state-of-the-art method.

B.2 Allow only watermarked inputs

The same method can be used for admission control. For this, we can use the following module that reject or forward input samples to target neural networks.

$$reject\_or\_bypass(x) = \begin{cases} 0 & \text{if no watermark} \\ x & \text{if watermark exists on } x \end{cases} \tag{6}$$

The module in equation 6 says that x will be delivered to the target neural network only if x is properly watermarked. The implementation of the module is similar to that of the backdoor attack. However, we do not need to train w in this case.

Recall that our watermarks did not decrease the accuracy for both FR and SR netural networks. This property of no (or very little) accuracy drop is required to use the watermarking method for admission control. Our method meets the requirement.

[3]This step is very easy if source codes are available. If not, one can use the graph editor (TensorFlow, 2018) to modify pre-trained neural network models.

B.3 PROVE THE OWNERSHIP OF NEURAL NETWORKS

The proposed $decode\&inject$ module can be used to prove the ownership of neural networks. One can implant the module in the way we described and later use it to prove the ownership against the plagiarism of neural networks. If other people copy the neural network protected by our watermarking method, you can show that the copied neural network reacts to your watermarked samples and prove that the copied neural network is originally designed by you.