JUSTC

Article

http://just.ustc.edu.cn

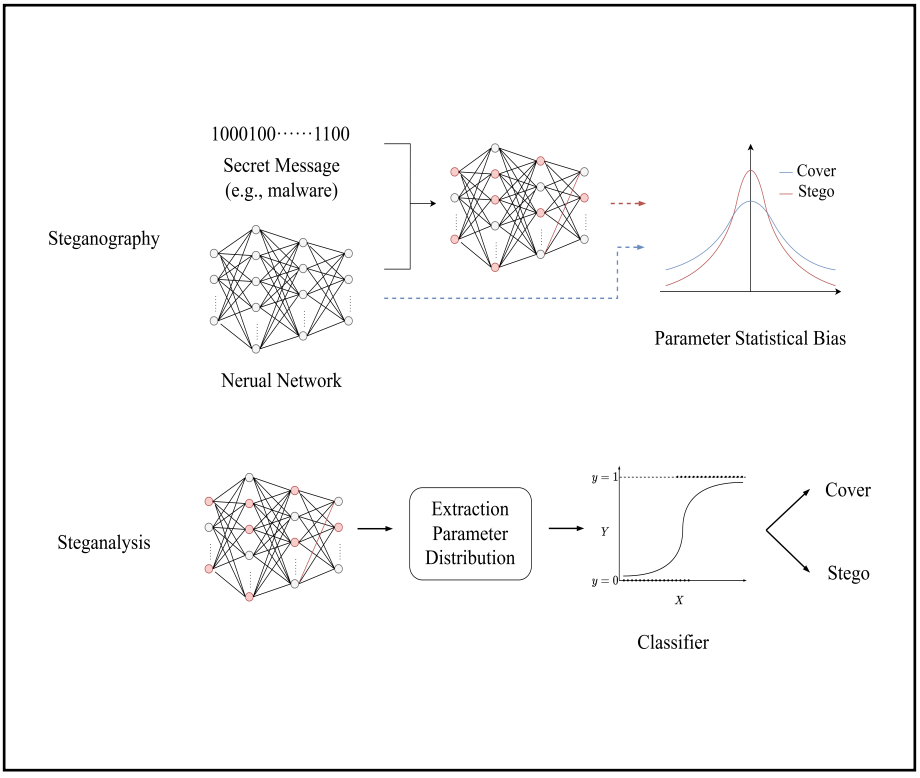
Received: September 07, 2021; Accepted: December 01, 2021

Steganalysis of neural networks based on parameter statistical bias

Yi Yin, Weiming Zhang, Nenghai Yu, and Kejiang Chen ✉

School of Cyber Science and Technology, University of Science and Technology of China, Hefei 230029, China
Correspondence: Kejiang Chen, E-mail: chenkj@ustc.edu.cn

Graphical abstract



Malicious developers can hide malware or other baneful information into the pretrained model imperceptibly, which does harm to computer society. However, steganography on neural network will modify the statistical distribution of the model. We propose steganalysis methods for steganography on neural network parameters by extracting statistics from benign and malicious models and building classifiers based on the statistics.

Public summary

- Different steganography of neural networks will lead to different bias in the statistics of the parameters. By designing valid features to capture bias and training the classifier, we can effectively detect the injected network.
- The length of the secret message embedded by the malicious developer is unknown. The experimental results show that even when the payload is low, our detecting method still works.
- Since the method used to embed information in the injected network is unknown, an effective framework allows for a more comprehensive detection of the injected network.

Citation: Yin Y, Zhang W M, Yu N H, et al. Steganalysis of neural networks based on parameter statistical bias. *JUSTC*, 2022, **52**(1): 1. DOI: 10.52396/JUSTC-2021-0197

JUSTC

Article

http://just.ustc.edu.cn

Received: September 07, 2021; Accepted: December 01, 2021

Steganalysis of neural networks based on parameter statistical bias

Yi Yin, Weiming Zhang, Nenghai Yu, and Kejiang Chen ✉

School of Cyber Science and Technology, University of Science and Technology of China, Hefei 230029, China
Correspondence: Kejiang Chen, E-mail: chenkj@ustc.edu.cn

Cite This: *JUSTC*. **2022**, 52(1): 1 (11pp) Read Online SI Supporting Information

Abstract: Many pretrained deep learning models have been released to help engineers and researchers develop deep learning-based systems or conduct research with minimall effort. Previous work has shown that at secret message can be embedded in neural network parameters without compromising the accuracy of the model. Malicious developers can, therefore, hide malware or other baneful information in pretrained models, causing harm to society. Hence, reliable detection of these vicious pretrained models is urgently needed. We analyze existing approaches for hiding messages and find that they will ineluctably cause biases in the parameter statistics. Therefore, we propose steganalysis methods for steganography on neural network parameters that extract statistics from benign and malicious models and build classifiers based on the extracted statistics. To the best of our knowledge, this is the first study on neural network steganalysis. The experimental results reveal that our proposed algorithm can effectively detect a model with an embedded message. Notably, our detection methods are still valid in cases where the payload of the stego model is low.

Keywords: neural network; steganography; steganalysis

CLC number: TN29 **Document code:** A

1 Introduction

Deep learning has seen considerable growth in the past decade [1–4]. State-of-the-art deep learning models such as LeNet [5], VGG [6], GoogLeNet [7], ResNet [8], and EfficientNet [9] have achieved superior performance in computer vision applications, such as object recognition[10], face recognition [11], and image classification [12–13]. In addition, numerous deep learning frameworks have been released to help engineers and researchers develop systems based on deep learning easily or conduct research.

Although these frameworks allow for easy deployment of neural networks in real-world applications, training neural network models is still a daunting task as it requires a considerable amount of data and computational resources. Therefore, pretrained models are provided on the websites to facilitate the reproduction of results of a study. Currently, the sharing of well-trained models is essential for both the research and development of deep neural network systems. Numerous pretrained models have been uploaded by developers on websites such as PyTorch Hub① and Papers With Code②.

Owing to the significant progress made in deep learning, neural networks are now being used as steganography cover media. Song et al. made the first attempt to embed messages in neural network parameters [14] and proposed three methods: The least significant bits(LSB) encoding, which simply embeds message in the lower bits of the parameters of well-trained models; Correlated Value Encoding, which forces the parameters to be highly correlated with the embedded message by means of malicious regular terms; and Sign Encoding, which encodes message in the signs of parameters.

Consequently, malicious developers can use neural networks to exchange messages imperceptibly. Liu et al. proposed StegoNet[15], which turns a deep learning model into a stegomalware by using model parameters as a payload injection channel. There is no significant decrease in accuracy with StegoNet, and the triggers are connected to the physical world by input specification. StegoNet focuses on embedding methods that are effective both on uncompressed and deeply compressed models. Deeply compressed models, such as VGG-16 Compressed and AlexNet Compressed [16] shrink in size by reducing both the amount and data precision of general uncompressed model parameters. Liu et al. showed that, similar to image steganography, LSB substitution in deeply compressed models will lead to significant bias in the statistics of the parameters. Therefore, LSB substitution in a deep compression model is easily detected by traditional image steganalysis methods such as primary sets [17], sample pairs [18], Chi Squares [19] and an RS analysis [20–21], while the LSB substitution on uncompressed models is challenging to detect. For these scenarios, Liu et al. proposed three improved embedding methods based on LSB substitution: Resilience training, value-mapping and sign-mapping. However, the approaches proposed by Liu et al. mainly focus on selecting the position with the minimum distortion to embed the secret message. Therefore, the message embedded by these approaches rely on a considerable amount of auxiliary information such as po-

① https://pytorch.org/hub/
② https://paperswithcode.com/

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1

sitions stored in other parts of the stegomalware to ensure extraction. In contrast, the approaches proposed by Song et al. are more universal and can be leveraged to harm society.

Details about models can be easily obtained because developers always share the model settings, such as the structure and dataset, when they publish the model. However, as the training process almost randomly initializes the parameters, it is still impossible to determine the cover model even with the information developers provide. Therefore, it is necessary to design a specific model detection method. In this paper, we demonstrate that the methods proposed by Song et al. [14] which can modify the statistical distribution of a model. An analysis of LSB Encoding revealed that the randomness of bits systematically increases during the embedding of the secret message; therefore, we can build discriminative features by measuring the randomness of the bit plane. Our analyses reveals that, with correlated value encoding and sign encoding, the distribution of parameters varies during embedding; therefore, we can determine statistical features of parameters to measure the differences between the varying distributions. Thereafter, a regression is utilized to capture the bias of the statistic feature for classification. The experimental results reveal that our methods are effective for detecting the model with an embedded message even when the payload of the stego model is low.

The remainder of the paper is structured as follows. Section 2 reviews the related work on neural networks and steganography of neural networks. In Section 3, we propose methods for detecting the presence of steganography in a given model. In Section 4, we present the results of experiments that validate the effectiveness of our proposed methods. Section 5 concludes this paper and discusses future research directions.

2 Preliminary and prior work

2.1 Neural network

Machine learning can be divided into supervised learning and unsupervised learning. Although we focus on supervised learning in this paper, our approaches can be applied to unsupervised learning as well. Let $\mathcal{X}$ be the input space where $\boldsymbol{x}_i \in \mathbb{R}^d$ is the i-th instance, and $\mathcal{Y}$ be the output space where $y_i \in \{1, \cdots, K\}$ is the true label of the i-th instance and $K \geq 2$ is the number of classes. Given a set of data points $D = (\boldsymbol{x}_i, y_i)_{i=1}^n$, where $(\boldsymbol{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, this set is partitioned into two subsets, training data D_{train} and testing data D_{test}. A machine learning model is a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ parameterized by parameters $\boldsymbol{\theta}$. When it is difficult to clearly state how a function should be calculated, deep neural networks are most often used. In deep neural networks, f is composed of layers of nonlinear transformations that map the input to a series of intermediate states and then to the output. The parameter $\boldsymbol{\theta}$ is the weight used in each transformation. As the depth of the network increases, the number of parameters increases.

In order to find the optimal set of parameters $\boldsymbol{\theta}$ for function f, the objective function, which penalizes the mismatch between the true label y_i and the predicted label generated by $f_\theta(\boldsymbol{x}_i)$, is minimized. Empirical risk minimization is a general framework, that uses the following objective function over $\boldsymbol{D}_{\text{train}}$:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_\theta(\boldsymbol{x}_i)) + \lambda \Omega(\boldsymbol{\theta}) \tag{1}$$

where L is the cross entropy loss function, and $\Omega(\boldsymbol{\theta})$ is the regularization term that prevents models from overfitting.

In the general deep neural network model, the parameters are 32-bit floating point numbers, following IEEE standard 754 [22]. According to IEEE standard 754, floating-point data should be represented in the form $(-1)^s \times 2^e \times t$, where s is the sign, e is the biased exponent which can be computed by subtracting 127, and t is the trailing significant field which represents fractions with a non-zero leading bit. A representation of 32-bit floating-point data in the binary interchange format is given in Fig.1. In this example, the float32 number is $0.75 = (-1)^0 \cdot (2^{-1}) \cdot (1 + 2^{-1})$, where $s = 0, e = 2^{126-127}$(127 is the bias), $t = 2^{-1}$. For steganography methods embedded in well-trained models that focus on the binary form of the parameters, the secret message is embedded directly in lower bit planes. For steganography methods embedded during training that focus on the decimal form of the parameters, the secret message is mapped into the values or the signs of the parameters.

2.2 Neural network steganography

Neural network steganography(NNS) is a timely issue, as neural networks are extremely redundant and are widely spread over the Internet. Malicious developers publish their white-boxed models with detailed structures and parameters, and receivers decode the secret message from the parameters. Such a scenario can appear on a third-party platform where people can publish and use the models. Even if the platform is secure, the model supplied by the developer may not be trustworthy. The security of the model is often overlooked by platforms and users.

NNS was proposed by Song et al. [14] to embed messages in uncompressed models and developed by Liu et al. [15] to embed secret messages on both compressed and uncompressed models. The approaches proposed by Liu et al.synthesize the deep learning models with stegomalware [23]. These methods shrink the size of the compressed model by reducing the number and precision of the model parameters. However, to maintain the accuracy of the model, the secret message that can be embedded must be short; furthermore, a significant amount

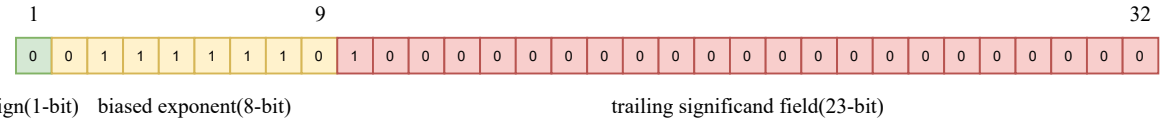


Fig. 1. An example of binary interchange formats for a float 32 number, where the green part is the sign, the yellow part is the biased exponent, and the red part is the trailing significant field.

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1

of auxiliary information must be extracted successfully.

Although the approaches proposed by Liu et al. and Song et al. use the same logic, those proposed by Song et al. focus more on uncompressed neural networks. As the methods proposed by Liu et al. have more limitations and require more auxiliary information for extraction than those proposed by Song et al., we focus on the steganalysis of uncompressed neural networks; specifically, we focus on the approaches proposed by Song et al. Song et al. [14] proposed three techniques: LSB encoding, correlated value encoding (COR) and sign encoding (SGN).

LSB encoding: In this method, the secret message is embedded directly in the least significant (lower) bits of the model parameters. First, malicious developers train a benign model. Then they post-process the model parameters $\boldsymbol{\theta}$ by replacing the lower bits of the parameters with the secret message $\boldsymbol{m}$, producing modified parameters $\boldsymbol{\theta}'$. At the receiver-end, the receiver extracts the message by reading the lower bits of the parameters and interpret them as bits of the secret message.

Correlated value encoding: This approach gradually embeds the secret message during training. The secret message is embedded by forcing parameters to be highly correlated with the secret message. In detail, a malicious correlation term C is added to the loss function, where

$$C(\boldsymbol{\theta},\boldsymbol{m}) = -\lambda_c \cdot \frac{\left|\sum_{i=1}^{l}(\theta_i-\bar{\theta})(m_i-\bar{m})\right|}{\sqrt{\sum_{i=1}^{l}(\theta_i-\bar{\theta})^2(m_i-\bar{m})^2}} \tag{2}$$

In the above expression, λ_c controls the level of correlation, $\boldsymbol{m}$ is the secret message with length l, $\bar{\theta}$ and $\bar{m}$ are the mean value of parameters $\boldsymbol{\theta}$ and secret message $\boldsymbol{m}$, respectively. During training, the malicious term drives the gradient direction towards a local minimum where the secret and the parameters are highly correlated. Therefore, the larger λ_c, the more correlated $\boldsymbol{\theta}$ and $\boldsymbol{m}$. Recovering the secret message from the model only requires mapping parameters back to the feature space because correlated parameters are approximately linear transformation of the secret message.

Sign encoding: Sign encoding is another method that can be used to encode a secret message as a model is trained. Similar to correlated value encoding, Sign encoding also adds a malicious correlation term to the loss function. However, the secret message $\boldsymbol{m}$ is embedded in the signs of the parameters. In detail, a positive parameter represents 1 and a negative parameter represents 0. The malicious correlation term P is defined as

$$P(\boldsymbol{\theta},\boldsymbol{m}) = \frac{\lambda_s}{l}\sum_{i=1}^{l}\left|\max(0,-\theta_i m_i)\right| \tag{3}$$

In the above expression, λ_s controls the level of correlation, and l is the length of the secret message $\boldsymbol{m}$. Recovering the secret message from the model only requires reading the signs of the parameters and then interpreting them as bits of the secret message.

Unlike in LSB encoding, in correlated value encoding and sign encoding, the secret message cannot be decoded completely and correctly. However, correlated value encoding and sign encoding have better robustness against fine-tuning.

3 Proposed methods

Even though NNS methods [14] can embed secret messages without performance degradation, biases will occur ineluctably in the model statistics. Moreover, it is easy to master the information of the model as developers need to specify the setting of their model such as the model structure and dataset when the model is released. With the mentioned information, we can build benign and malicious models, and train classifiers to detect the malicious models. In this section, we first present the detection framework for steganalysis of NNS. Then, we describe the model distribution bias generated by each method and design effective features for detection.

3.1 Framework of neural network steganalysis

Different biases are caused by the steganography methods proposed by Ref. [14]. For LSB encoding, the randomness of the benign bit plane is different from that of the malicious bit plane. For correlated value encoding and sign encoding, the distribution of parameters in each layer of the benign model differs from that of the malicious model. However, as the steganography method applied by the malicious developer is non-transparent, we design an overall framework of our neural network steganalysis to facilitate comprehensive detection. Fig. 2 illustrates the overall framework of our proposed steganalysis methods.

Our framework can be summarized as follows:

(Ⅰ) Feature extraction: Since the NNS method used by the

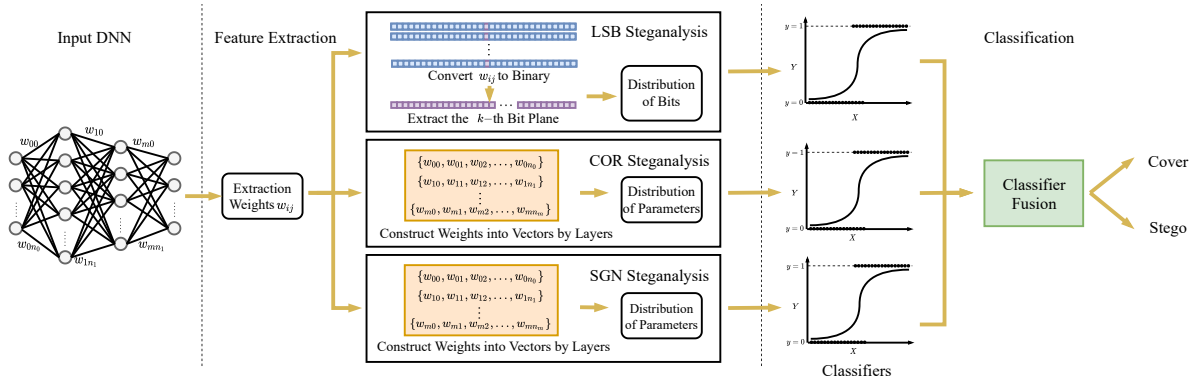


Fig. 2. Illustration of the framework we proposed for the neural network steganalysis.

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1

malicious developer is unknown, we extract features from the injected model for each NNS method to ensure comprehensive detection. For LSB encoding, we capture the distribution of bit plane as features. For correlated value encoding and sign encoding, we capture the distribution of parameters as features. All of the syntaxes of the features in this paper follow the convention name = $\{\phi\}_{[\#]}$, where ϕ represents the feature and # is the sequence number. These features captured by the i-th steganalysis method jointly constitute the multidimensional feature vector Φ_i, where $i \in \{1,2,3\}$, the 1-th method is LSB encoding steganalysis, the 2-th method is Correlated Value Encoding Steganalysis and the 3-th method is sign encoding steganalysis.

(Ⅱ) Classification: Essentially, steganalysis is a binary classification task. We use logistic regression[24−26] as our classifier. Let $\mathcal{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ be the input space where $\boldsymbol{x}_i$ is the feature of the i-th model extracted by one of the steganalysis methods; $\mathcal{Y} = \{y_1, ..., y_n\}$ is the output space, where y_i is the true label of the i-th model; $\boldsymbol{w}$ is the parameter of classifier and n is the number of models used to train the classifier. The objective function of our classifier is

$$\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} L_{\log}(y_i, g(\boldsymbol{x}_i; \boldsymbol{w})) + \epsilon \| \boldsymbol{w} \|_2^2 \tag{4}$$

where ϵ denotes the coefficient of the regularization term. The loss function $L_{\log}(y_i, g(\boldsymbol{x}_i; \boldsymbol{w}))$ is defined as

$$\begin{aligned} &L_{\log}(y_i, g(\boldsymbol{x}_i; \boldsymbol{w})) = \\ &\quad y_i \log g(\boldsymbol{x}_i; \boldsymbol{w}) + (1 - y_i) \log(1 - g(\boldsymbol{x}_i; \boldsymbol{w})) \end{aligned} \tag{5}$$

where $g(\boldsymbol{x}_i; \boldsymbol{w})$ is the logistic function:

$$g(\boldsymbol{x}_i; \boldsymbol{w}) = \frac{1}{1 + \mathrm{e}^{-\boldsymbol{w}^T \boldsymbol{x}_i}} \tag{6}$$

The ensemble reaches its decision by fusing all the decisions of the subclassifiers using a voting process. The ensemble judges the injected model as benign only when all the subclassifers consider it a cover model.

3.2 Feature extraction

As these steganography methods can lead to biases, it is clear that the artifacts they generate can be identified by effective features. In this subsection, we will describe feature extraction in detail.

Multiple features can be used to identify bias. For each steganalysis of NNS, we evaluate the performance of each feature to identify whether the feature is an optimal feature. We then obtain a set of optimal features. For each feature, we train a logistic regression classifier and obtain the detection accuracy. For LSB encoding, ResNet34[8] trained on CIFAR10[27] is used to evaluate the performance of each feature, and we train 100 benign models and 100 malicious models with a payload of 1.0 bits per parameter. For correlated value-encoding and sign encoding, ResNet34[8], VGG16[6], and EfficientNetB0[9] trained on CIFAR10 are used to evaluate the performance of each feature, and for each network, we train 100 benign models and 100 malicious models with payloads of 0.2, 0.6 and 1.0 bits per parameter.

(Ⅰ) Steganalysis of LSB encoding: For security purposes, a secret message is always encrypted before it is embedded. Thus, the secret message can be regarded as a random binary message. For a high performance to be achieved with LSB encoding, high-precision parameters are not required; therefore, modifying the lower bits of the parameters will not lead to significant performance degradation. However, the randomness of benign bit planes is less than that of malicious bit planes. For example, in the binary secret message,0 and 1 are uniformly distributed, while a bias exists for the distribution of 0 and 1 in the benign bit plane. Therefore, we propose our steganalysis algorithm by detecting such biases in the distribution of bit plane.

The bias detection in the bit plane distribution can be implemented by randomness test. Thus far, numerous studies on randomness detection have been proposed[28−31], and the NIST statistical test suite[29] is one of the most representative methods. Therefore, in our method, we design our steganalysis features referring to the NIST statistical test suite, which includes 14 test statistics.

For an injected model, the lowest bit plane with a tendency to be modified is extracted and detected for randomness. As the secret message length varies, not all parameters will be changed. As we detect the bias in the bit plane distribution, while varying the message length, we focus on the change in the payload of the specified bit plane.

To find the optimal set of features, we use the 14 test statistics of the NIST statistical test suite to train the subclassifiers. For each subclassifier, we test the average detection accuracy over the bit planes from the 14th bit plane to the 18th bit plane of ResNet34 trained on CIFAR10 for a payload of 1.0 bits per parameter. Fig. 3 shows the averaged accuracy of each subclassifier. We select the statistics with the top 4 highest accuracies as features. They are the statistics of the frequency test, serial test, approximate entropy test, and cumulative sums test, and are, denoted by ϕ_1, ϕ_2, ϕ_3 and ϕ_4, respectively.

ϕ_1 evaluates whether the proportion of 0 and 1 in a sequence is similar to that in a random sequence. For a binary secret message, the number of 0 and 1 in the sequence should be about the same. However, for a sequence extracted from benign bit planes, there is a bias in the proportion of 0 and 1. In this situation, we convert the 0 in the sequence to -1 and compute the sums Sum(n) of the sequence with length n. Therefore, ϕ_1 is computed as

$$\phi_1 = \frac{|\mathrm{Sum}(n)|}{\sqrt{n}} \tag{7}$$

ϕ_2 and ϕ_3 focus on the frequency of all possible overlapping blocks across the sequence. ϕ_2 evaluates whether the number of occurrences of the overlapping patterns is approximately the same as the expected number of random sequence. We count the frequency of the i-th p-bit blocks as $v_{i_1 \ldots i_p}$ and the frequency of the i-th $(p-1)$-bit blocks as $v_{i_1, \ldots, i_{p-1}}$, where $i_1, ..., i_p$ and $i_1, ..., i_{p-1}$ are the p-bit pattern and $(p-1)$-bit pattern, respectively. ϕ_2 is computed as

$$\phi_2 = \frac{2^p}{n} \sum_{i_1, \cdots, i_p} v_{i_1, \cdots, i_p}^2 - \frac{2^{p-1}}{n} \sum_{i_1, \cdots, i_{p-1}} v_{i_1, \cdots, i_{p-1}}^2 \tag{8}$$

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, 52(1): 1

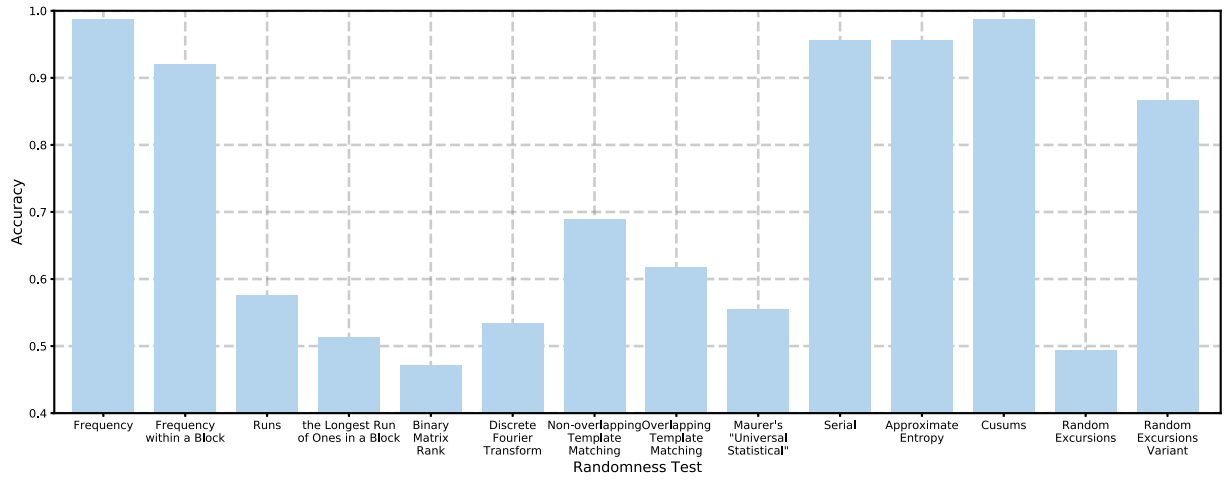


Fig. 3. Averaged accuracy over the bit planes from the 14th to 18th bit plane of ResNet34 trained on CIFAR10 with a payload of 1.0 bit per parameter for each subclassifier.

where n represents the length of the sequence.

ϕ_3 uses the approximate entropy to compare the frequency of overlapping blocks of two consecutive lengths (p and $p+1$) against the expected result for a random sequence. Let the count of the possible p-bit values be described as

$$C_i^p = \frac{\#i}{n} \tag{9}$$

where i represents a p-bit value, $\#i$ represents the calculated count of the value i, and n represents the length of the sequence. The approximate entropy presents the difference in frequency between the p-bit overlapping blocks and the $(p+1)$-bit overlapping blocks. The approximate entropy (ApEn) is computed as

$$\mathrm{ApEn}(p) = \varphi^{(p)} - \varphi^{(p+1)} \tag{10}$$

$\varphi^{(p)}$ is the entropy of the empirical distribution arising on the set of all 2^p possible patterns of length p, Thus

$$\varphi^{(p)} = \sum_{i=0}^{2^p-1} C_i^p \log C_i^p \tag{11}$$

ϕ_3 measure the match between the observed value of ApEn(p) and the expected value, Thus

$$\phi_3 = 2n[\log 2 - \mathrm{ApEn}(p)] \tag{12}$$

ϕ_4 evaluates whether the cumulative sum of the partial sequences is approximately the same as the expected cumulative sum of random sequences. We convert 0 in the sequence to -1 and compute the sums Sum(i) of successively larger subsequences with length i starting from the beginning:

$$\mathrm{Sum}(i) = \mathrm{Sum}(i-1) + X_i, i = 2, ..., n \tag{13}$$

where X_i is the i-th value of the converted sequence, $\mathrm{Sum}(1) = X_1$, and n is the length of the sequence. We use the largest excursion from the origin of the cumulative sums as feature ϕ_4, thus

$$\phi_4 = \max_{1\le k\le n} |\mathrm{Sum}(k)| \tag{14}$$

The total feature vector of LSB encoding steganalysis is $\Phi_1 = \{\phi_1, \phi_2, \phi_3, \phi_4\}$.

(Ⅱ) Steganalysis of correlated value encoding: Correlated value encoding constrains the value range of the parameter by the malicious correlation term. In this situation, a trade-off between the embedding degree and the model accuracy is achieved during training. We observe that the parameters of the malicious model present a different distribution from those of the benign model owing to the constraints on the parameter values. Therefore, the deviation in the distribution of the parameters is used to detect the correlated value encoding in our steganalysis algorithm.

To detect the bias caused by correlated value encoding, we use moments as our features to describe the shape of the distribution. The most commonly used moments are first-order, second-order, third-order and fourth-order moments. Under this situation, we design our steganalysis features by referring to these moments. As the distribution of parameters in different layers may vary in a neural network, a more detailed description of the parameter distribution is necessary. We describe the distribution of the parameters in each layer.

$\boldsymbol{\phi_5}$ is expectation, which is used to measure the average value of the parameters and is defined as

$$\phi_5(j) = \frac{1}{n_j} \sum_{i=1}^{n_j} \theta_i \tag{15}$$

where n_j is the number of parameters in the j-th layer.

$\boldsymbol{\phi_6}$ is variance, which is used to measure the degree of deviation between parameters and their expectation. $\boldsymbol{\phi_6}$ is computed by

$$\phi_6(j) = \frac{1}{n_j} \sum_{i=1}^{n_j} (\theta_i - \phi_5(j))^2 \tag{16}$$

$\boldsymbol{\phi_7}$ is skewness, which is used to measure the direction and degree of parameters distribution skew. $\boldsymbol{\phi_7}$ is computed by

$$\phi_7(j) = \frac{\frac{1}{n_j}\sum_{i=1}^{n_j}(\theta_i - \bar{\theta})^3}{(\frac{1}{n_j}\sum_{i=1}^{n_j}(\theta_i - \bar{\theta})^2)^{\frac{3}{2}}} \tag{17}$$

$\boldsymbol{\phi_8}$ is kurtosis, which is used to measure the flatness of the

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1

parameters distribution, ϕ_8 is defined as

$$\phi_8(j)=\frac{\frac{1}{n_j}\sum_{i=1}^{n_j}\left(\theta_i-\bar{\theta}\right)^4}{\left(\frac{1}{n_j}\sum_{i=1}^{n_j}\left(\theta_i-\bar{\theta}\right)^2\right)^2} \tag{18}$$

Each feature is a vector with the dimension of the number of layers in the model. To find the optimal set of features, we use the 4 features to train subclassifiers. For each subclassifier, the average accuracy is obtained over ResNet34, VGG16, and EfficientNetB0 where $\lambda_c = 0.1$ with payloads of 0.2, 0.6, and 1.0 bits per parameter. Fig. 4 shows the average accuracy of each subclassifier. Using the settings mentioned in Ref. [14], gray-scale images are embedded as the secret message. Hence, for each parameter, 8 bits of a secret message are embedded. As shown in Fig .4, $\boldsymbol{\phi_6}$ and $\boldsymbol{\phi_8}$ give accuracies above 0.95, which are higher than those obtained with $\boldsymbol{\phi_5}$ and $\boldsymbol{\phi_7}$. Thus, we choose $\boldsymbol{\phi_6}$ and $\boldsymbol{\phi_8}$ as the features for detecting the models. Therefore, the total feature vector of correlated value encoding steganalysis is $\Phi_2 = \{\boldsymbol{\phi_6}, \boldsymbol{\phi_8}\}$.

(Ⅲ) Steganalysis of sign encoding: Sign encoding can also be used to embed a secret message through the malicious correlation term during training; however, in this case, the message is embedded in the signs of parameters. Theoretically, by the malicious correlation term, the parameter and the secret message have the same sign. However, we observe that in practice, not all sign constraints are met. The malicious term penalizes the mismatched parameters by bringing them close to 0, which leads to a distribution of parameters that is different from that in the benign model. Therefore, the difference in model parameter distribution is used to detect the secret message in our steganalysis method.

As sign encoding also disrupts the parameters distribution, we select features using an approach that is similar to correlated value encoding steganalysis. To detect the bias caused by sign encoding, we also design our steganalysis features by referring to the first-order moments $\boldsymbol{\phi_5}$ (expectation), second-order moments $\boldsymbol{\phi_6}$ (variance), third-order moments $\boldsymbol{\phi_7}$ (skewness) and fourth-order moments $\boldsymbol{\phi_8}$ (kurtosis). Similarly, the distribution is described for each layer separately.

Similar to the experiment for correlated value encoding steganalysis, for each subclassifier, the average accuracy is obtained over ResNet34, VGG16 and EfficientNetB0 where $\lambda_s = 50$ with payloads of 0.2, 0.6, and 1.0 bits per parameter. Fig. 4 shows the average accuracy of each subclassifier. As shown in Fig. 4, $\boldsymbol{\phi_5}$ and $\boldsymbol{\phi_7}$ provide accuracies above 0.95, which are higher than those achieved with $\boldsymbol{\phi_6}$ and $\boldsymbol{\phi_8}$. Thus, we choose $\boldsymbol{\phi_5}$ and $\boldsymbol{\phi_7}$ as the features for detecting models. Therefore, the total feature vector of sign encoding steganalysis is $\Phi_3 = \{\boldsymbol{\phi_5}, \boldsymbol{\phi_7}\}$.

4 Experiments

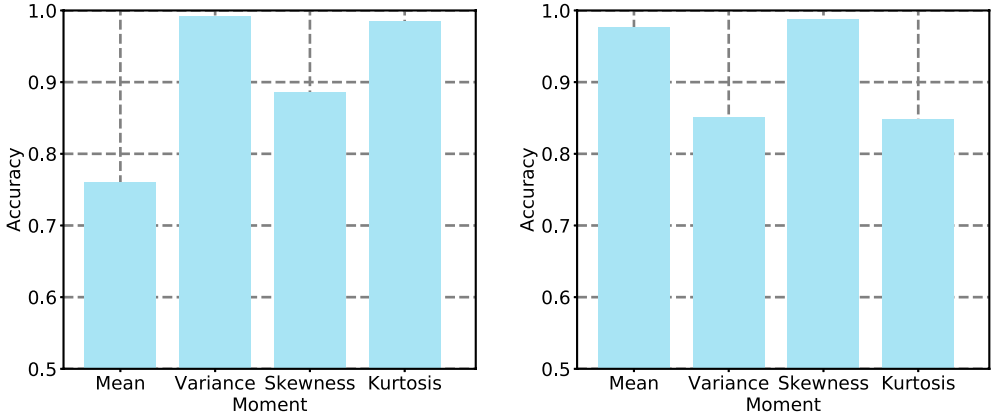
In this section, we introduce the setup of our experiments. Then, we discuss the results of the experiments, including the detection achieved with three steganography methods with different embedding rates.

4.1 Experimental setup

Table 1 summarizes the datasets and networks we used in our experiments.

The image datasets used in our experiments are the well-known CIFAR10[27] and Tiny-ImageNet[32]. ResNet34[8], VGG16[6], EfficientNetB0[9], and MobileNet[33] are selected as the models to be detected. Table 1 shows the number of parameters for each model and the accuracy of the benign model. Our implementation and its corresponding initial architectures are based on PyTorch. In all our experiments, we set the mini-batch size to 128, the initial learning rate to 0.1, and the number of epochs for training to 100. For networks trained on CIFAR-10, we decrease the learning rate by a factor of 0.1 for better convergence in epoch 60. For models trained on Tiny-ImageNet, we decrease the learning rate by a factor of 0.1 in epochs 40 and 60. In each experiment, models are validated and saved every one epoch. The model with the best validation accuracy is selected as the final model.

To measure the impact of secret message length on detection, we design detecting tasks with different payloads for all the steganography methods. For LSB encoding, the payloads are set to 0.05, 0.4, 0.8, and 1.0 bits per parameter. For correlated value encoding and sign encoding, as not all the secret



(a) Steganalysis of Correlated Value Encoding (b) Steganalysis of Sign Encoding

Fig. 4. Results for the detection of ResNet34, VGG16, and EfficientNetB0 trained on CIFAR10 using different statistics as features respectively. (a) results for correlated value encoding steganalysis; and (b) results for sign encoding steganalysis.

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1

Table 1. Models and datasets used in our experiments.

Dataset	Model	Num params	Test ACC
CIFAR10	ResNet34	21093056	93.64
	VGG16	14719808	92.50
	EfficientNet	3550816	91.30
Tiny-ImageNet	VGG16	21210944	55.18
	MobileNet	3389888	54.51

bits can be embedded successfully, we set the payloads to 0.2, 0.6, and 1.0 bits per parameter. For all the steganography methods, we train 100 benign models and 100 malicious models. When the payload is not 1.0, the malicious models are embedded randomly.

For all the steganography methods with a payload of 1.0, we train 100 benign models and 100 malicious models. For LSB encoding with payloads of 0.05, 0.4, and 0.8, we train 100 benign models and 300 malicious models, which are uniformly composed of three embedding methods: sequential embedding, sequential embedding from a random position and random embedding. For Correlated Value Encoding and Sign Encoding with payloads of 0.2 and 0.6, we train 100 benign models and 100 malicious models, which are embedded randomly.

For the detection of LSB encoding, correlated value encoding and sign encoding, we evaluate the performance of our detecting methods by 5-fold cross validation. For each cross validation, 80% of the models are selected to form the training set and 20% of the models are selected to form the testing set.

We use logistic regression[24–26] as our classifier. In all our experiments, we set the number of epochs for training to 100, and use liblinear for optimizing the loss function. We also use the l_2-norm as the regularizer with the coefficient ϵ set to 1.0. For the detection of LSB encoding, correlated value encoding, and sign encoding, we measure the average detecting accuracy of the 5-fold cross validation. The average detecting accuracy is measured as

$$\text{accuracy} = \frac{1}{5}\sum_{i=1}^{5}\frac{\text{TP}_i + \text{TN}_i}{\text{TP}_i + \text{TN}_i + \text{FP}_i + \text{FN}_i} \tag{19}$$

where TP is the true positive, TN is the true negative, FP is the false positive, FN is the false negative, and i is the i-th cross validation.

4.2 Detection performance of LSB encoding

We train the neural network from scratch on the CIFAR-10 and Tiny-ImageNet with different model structures and payloads, and record the results achieved. Table 2 shows the lowest bit plane embedded without significantly reduced accuracy. For all models, embedding the 14th bit plane does not lead to a significant drop in model performance. In our experiments, we detect the bit planes from the 14th to the 18th bit plane. Table 3 shows the performance of LSB encoding steganalysis for the models trained on CIFAR10 and Tiny-ImageNet.

The results in Table 3 reveal that, except for a few cases, our approach can effectively detect all models, from the 14th to 18th bit plane. For all model architectures at the 14th bit plane with payloads of 0.4, 0.8 and 1.0, our method can achieve 100% detection accuracies. However, with an increase in the bit planes and a decrease in payload, the performance of our method decreases. For example, for EfficientNetB0 trained on CIFAR10, while the payload is 1.0, our method fails to detect the 18th bit plane as the accuracy is 62.75%, and while the payload decreases to 0.05, even for the 14th bit plane, the accuracy is 77.72%. For models trained on CIFAR10, our method works better on ResNet34 and VGG16 than EfficientNetB0. For models trained on Tiny-ImageNet, our method works better on VGG16 than MobileNet. This result can be attributed to the number of parameters, which indicates that for a specified bit plane, sequences formed by benign models with fewer parameters have greater random-

Table 2. Results of the LSB encoding in our models. b is the lowest bit plane embedded without significantly reduced accuracy.

Dataset	Model	b	Test ACC
CIFAR10	ResNet34	11th	92.59
	VGG16	13th	91.93
	EfficientNet	13th	90.67
Tiny-ImageNet	VGG16	13th	54.42
	MobileNet	14bit	53.67

Table 3. Results of the LSB encoding steganalysis for models trained on CIFAR10 and Tiny-ImageNet at payloads of 1, 0.8, 0.4, and 0.05 on Bit planes from the 14th to 18th bit plane.

Model	Payload	accuracy				
		14th	15th	16th	17th	18th
ResNet34	1.0	100.00	100.00	100.00	99.50	93.05
	0.8	100.00	100.00	98.95	81.85	88.75
	0.4	100.00	100.00	98.95	79.55	70.20
	0.05	87.25	92.10	74.84	74.35	65.20
VGG16	1.0	100.00	100.00	100.00	98.95	87.40
	0.8	100.00	100.00	98.30	94.15	82.85
	0.4	100.00	100.00	97.15	72.70	69.30
	0.05	87.55	75.35	59.60	51.65	46.50
EfficientNet	1.0	100.00	100.00	99.50	88.45	62.75
	0.8	100.00	95.85	99.45	85.75	60.55
	0.4	100.00	96.40	83.35	71.35	59.40
	0.05	67.85	61.35	47.35	41.95	45.90
VGG16	1.0	100.00	100.00	100.00	99.50	91.95
	0.8	100.00	100.00	98.80	81.55	85.30
	0.4	100.00	100.00	98.50	80.45	72.10
	0.05	92.00	72.20	61.50	54.45	45.30
MobileNet	1.0	100.00	100.00	99.50	90.70	66.35
	0.8	100.00	95.70	98.00	85.40	69.10
	0.4	100.00	96.80	87.00	72.15	56.20
	0.05	67.50	62.65	47.30	45.95	45.80

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1

ness, increasing the difficulty of detection. The results in Table 3 indicate that in most cases, the proposed method is an effective way for detecting LSB encoding.

4.3 Detection performance of correlated value encoding

To follow the setup mentioned in Ref. [14], we use gray-scale images as the secret message to be embedded. We train the neural network on the CIFAR10 and Tiny-ImageNet with different model structures, coefficient λ_c and payloads. Table 4 shows the appropriate coefficient λ_c and model accuracy. We use the mean absolute error (MAE) to measure the embedding degree. Given the embedded parameters $\boldsymbol{\theta}$ and the secret $\boldsymbol{m}$, MAE is $\frac{1}{l}\sum_{i=1}^{l}|\theta_i - m_i|$, where l is the length of the secret message $\boldsymbol{m}$. The range of MAE is $[0, 255]$, where 0 means the decoded and secret messages are identical and Table 5 shows the performance of our steganalysis method for models trained on different databases under payloads of 0.2, 0.6, and 1.0.

Table 4. Results of the correlated value encoding in our models. $\lambda_c$is a suitable coefficient for the correlation term. The mean absolute error (MAE) is used to measure the embedding degree. Test ACC indicates the performance of malicious models trained with different λ_c.

Dataset	Model	λ_c	MAE	Test ACC
CIFAR10	ResNet34	0.1	21.42	93.06
		0.05	34.16	93.29
	VGG16	0.1	30.27	92.62
		0.05	36.26	92.66
	EfficientNet	0.03	11.86	91.28
		0.01	15.08	91.24
Tiny-ImageNet	VGG16	0.1	48.42	52.29
		0.05	51.02	54.51
	MobileNet	5.0	49.68	54.08
		1.0	51.95	54.42

Table 5. Results of the correlated value encoding steganalysis for models trained on CIFAR10 and Tiny-ImageNet. For all models, the payloads are set to 1.0, 0.6, and 0.2, separately.

Dataset	Model	λ_c	accuracy		
			1.0	0.6	0.2
CIFAR10	ResNet34	0.1	100.00	98.65	98.20
		0.05	100.00	98.20	95.75
	VGG16	0.1	100.00	99.45	99.45
		0.05	100.00	96.95	99.45
	EfficientNet	0.03	100.00	99.70	100.00
		0.01	100.00	99.85	99.05
Tiny-ImageNet	VGG16	0.1	100.00	96.85	98.20
		0.05	93.80	96.85	97.15
	MobileNet	5.0	100.00	100.00	100.00
		1.0	100.00	100.00	100.00

From Table 5, it can be seen that our approach can effectively detect all the models with different λ_c and payloads. For all the models with a payload of 1.0, our method can achieve 100.00% detection accuracies. With a decrease in payload, except for a few cases, the performance of our detection decreases. For ResNet34 trained on CIFAR10 with $\lambda_c = 0.05$, the accuracy of detection with a payload of 0.6 is 98.20%, and for a payload drops of 0.2, the accuracy is 95.75%. However, for VGG16 trained on Tiny-ImageNet with $\lambda_c = 0.05$, the accuracy of detection with a payload of 1.0 is 93.80%, which is lower than that achieved with a payload of 0.2. However, even at a low embedding payload of 0.2, our approach is still valid.

4.4 Detection performance of sign encoding

We train the neural network from scratch on the CIFAR10 dataset and Tiny-ImageNet with different model structures, λ_s, and payloads. Table 6 shows the appropriate coefficient for the correlation term. Given the embedded parameters $\boldsymbol{\theta}$ and binary secret $\boldsymbol{m}$, the embed degree is measured by $\frac{1}{l}\sum_{i=1}^{l} 1\{\text{sign}(\theta_i) \neq m_i\}$, where $1\{\cdot\}$ is the indicator function, $1\{\text{a true statement}\} = 1$ and $1\{\text{a false statement}\} = 0$, l is the length of the secret message $\boldsymbol{m}$, and $\text{sign}(\cdot)$ is the sign function. Table 7 shows the performance of the neural network under different payloads, which are set to 0.2, 0.6, and 1.0.

The results in Table 7 reveal that our approach can effectively detect all the models with different λ_s at payloads at 1.0, 0.6, and 0.2. For VGG16 trained on Tiny-ImageNet with $\lambda_s = 50.0$ and 10.0 at payloads of 0.2, 0.6, and 1.0, our method achieves a 100.00% detection accuracy. Furthermore, the decrease in payload has no significant effect on the detection accuracy. For example, for Efficient NetB0 trained on CIFAR10 with $\lambda_s = 10.0$ at a payload of 1.0, the accuracy is 92.05%, which is lower than those at payloads of 0.6 and 0.2. However, even when the payload is low, our detection method is still valid.

4.5 Detection performance of the overall framework

As the steganography method and the payload used by the malicious developer for an injected model are both unknown,

Table 6. Results of the sign encoding steganalysis for models trained on CIFAR10 and Tiny-ImageNet. Test ACC indicates the performance of malicious models trained with $\lambda_s = 50.0$ and 10.0.

Dataset	Model	λ_s	Embed degree	Test ACC
CIFAR10	ResNet34	50.0	7.80	94.22
		10.0	27.48	94.20
	VGG16	50.0	5.22	92.94
		10.0	17.69	93.00
	EfficientNet	50.0	2.15	91.34
		10.0	7.20	91.71
Tiny-ImageNet	VGG16	50.0	36.02	55.28
		10.0	39.96	55.33
	MobileNet	50.0	27.84	55.09
		10.0	43.61	54.76

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1

Table 7. Results of sign encoding steganalysis for models trained on cifar10 and tiny-imagenet at payloads of 1.0, 0.6, and 0.2.

Dataset	Model	λ_s	accuracy		
			1.0	0.6	0.2
CIFAR10	ResNet34	50.0	99.95	100.00	100.00
		10.0	98.05	99.95	99.10
	VGG16	50.0	100.00	100.00	99.10
		10.0	98.50	99.50	99.50
	EfficientNet	50.0	99.00	100.00	100.00
		10.0	92.05	100.00	100.00
Tiny-ImageNet	VGG16	50.0	100.00	100.00	99.50
		10.0	100.00	100.00	100.00
	MobileNet	50.0	100.00	100.00	100.00
		10.0	100.00	100.00	100.00

we need to use all three detection methods and then fuse their results. For simplicity purposes, we specify the payloads of the model parameters, detect models by the three steganalysis methods separately, and fuse the decisions.

Neural networks are trained on the CIFAR10 and Tiny-ImageNet with different model structures. For each detection method, 60 benign and 60 malicious models are selected to train the classifier. A total of 40 benign and 120 malicious models uniformly composed of the three steganography methods are used for validation. For LSB, the secret message is embedded in the 14th bit plane, and the payload is set to 0.05. For COR, the payload is set to 0.2, and the coefficient λ_c is set to the maximum appropriate value. For example, for ResNet34, the max appropriate coefficient λ_c is 0.1, and for EfficientNet, the maximum appropriate coefficient λ_c is 0.03. For SGN, the payload is 0.2, and the coefficient λ_s is 50.0. The missing alarm rate and the false alarm rate are used to evaluate the effectiveness of detection, and are defined as

$$P_{\mathrm{MA}} = \frac{\mathrm{FN}}{\mathrm{TP}+\mathrm{FN}} \tag{20}$$

$$\mathrm{P}_{\mathrm{FA}} = \frac{\mathrm{FP}}{\mathrm{TP}+\mathrm{FP}} \tag{21}$$

Table 8 shows the detecting results of our overall framework. LSB detection, COR detection, and SGN detection mean detecting all models by the specific steganalysis method, and the framework detection means detecting models by our framework. It can be seen that our overall framework can effectively detect injection models. The missing alarm rate for framework detection is lower than that for specific detection such as LSB detection. Compared to LSB detection, framework detection has a lower false alarm rate as there is an increased number of true positives. However, for COR detection and SGN detection, the false alarm rate is lower than that for framework detection owing to the voting rule of our framework.

To validate the detecting performance of our framework with unknown payloads, classifers in our framework trained on models with lower payloads are used to detect the models with higher payloads. As in the framework detection experiment setting, in this case, neural networks are also trained on the CIFAR10 and Tiny-ImageNet with different model structures. For LSB, secret message is embedded in the 14th bit plane. For COR, the coefficient λ_c is set to the maximum appropriate value. For SGN, the coefficient λ_s is 50.0. In our framework, the classifiers for LSB detection are trained with a payload of 0.05, and the classifiers for COR detection and SGN detection are trained with a payload of 0.2. Then, we validate our framework on models embedded by LSB with payloads of 1.0, 0.8, and 0.4, embedded by COR with payloads of 1.0 and 0.6, and embedded by SGN with payloads of 1.0 and 0.6. For each embedding method under each payload, 40 benign models and 40 malicious models are used to detect the secret message.

Table 9 shows the detection results obtained using the framework trained with models with lower payloads to detect models with higher payloads. It can be seen that for most of the higher payloads models, our framework, which is trained on the lower payloads models, can effectively detect them. For ResNet34 trained on CIFAR10 embedded by SGN at the payload of 0.6, the accuracy of our framework is 97.50%. However, for EfficientNet trained on CIFAR10 embedded by COR at the payload of 0.6, the accuracy is only 68.75%. The results inTables 8 and 9 reveal that, in most cases, our overall framework trained on lower payloads models can effectively detect models with higher payloads.

5 Conclusions

In this paper, we propose steganalysis methods to detect the steganography on neural networks. First, we analyze the stat-

Table 8. Detection results of our overall framework. LSB detection means detecting models by the LSB encoding steganalysis. The definitions of COR detection and SGN detection are similar to that of LSB detection. Framework detection means detecting models by the overall framework.

Dataset	Model	LSB detection		COR detection		SGN detection		Framework detection	
		Missing Alarm	False Alarm	Missing Alarm	False Alarm	Missing Alarm	False Alarm	Missing Alarm	False Alarm
CIFAR10	ResNet34	68.33	7.317	66.67	9.09	56.67	0.00	2.50	5.98
	VGG16	70.00	7.69	66.67	0.00	64.17	0.00	3.33	2.59
	EfficientNet	78.33	29.73	66.67	0.00	52.50	0.00	11.67	10.37
Tiny-ImageNet	VGG16	69.17	11.91	53.33	1.78	69.16	15.91	5.83	11.81
	MobileNet	78.33	40.90	66.67	0.00	66.67	11.11	11.67	21.70

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1

Table 9. Results of using frameworks trained with models with a lower payload to detect models with higher payloads. In the framework, the classifiers of LSB detection are trained with a payload of 0.05, and the classifers of COR and SGN detections are trained with a payload of 0.2.

Dataset	Model	LSB			COR		SGN	
		1.0	0.8	0.4	1.0	0.6	1.0	0.6
CIFAR10	ResNet34	98.75	100.00	98.75	92.50	90.00	81.25	97.50
	VGG16	97.50	100.00	100.00	73.75	96.25	73.75	81.25
	EfficientNet	95.00	96.25	95.00	82.50	68.75	73.75	98.75
Tiny-ImageNet	VGG16	95.00	97.50	95.00	90.00	88.75	88.75	91.25
	MobileNet	92.50	95.00	86.25	91.25	92.50	88.75	92.50

istical bias caused by these steganography methods. As there are multiple features that can be used to describe each statistical bias, in order to find an optimal set of features, we compare the detection accuracies of the methods. Finally, we use the optimal set of features for classification. Various experiments are conducted to show the effectiveness of our framework in detecting neural network steganography.

The results in Table 3 reveal that, for LSB Encoding Steganalysis, our method fails to detect the bit planes higher than the 18th bit plane. Methods for detecting higher bit planes need to be explored.

Acknowledgements

This work is supported in part by the Natural Science Foundation of China (62102386, 62002334,62072421, 62121002), Fundamental Research Funds for the Central Universities (WK2100000018,WK2100000011), Exploration Fund Project of University of Science and Technology of China (YD3480002001), and Open Fund of Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation.

Conflict of interest

The authors declare that they have no conflict of interest.

Biographies

Yi Yin is working towards the PhD degree at School of Cyberspace Science and Technology, University of Science and Technology of China. Her area of interests includes Neural Network Steganography.

Kejiang Chen is a postdoctoral researcher at the School of Cyberspace Science and Technology, University of Science and Technology of China. His research interests include information hiding and artificial intelligence security. He has published more than 20 papers in IEEE TDSC, TIFS, TVCG, TCSVT, CVPR, ICCV and other journals and conferences.

References

[1] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. *Nature,* **2015**, *518* (7540): 529–533.

[2] Lin X, Rivenson Y, Yardimci N T, et al. All-optical machine learning using diffractive deep neural networks. *Science,* **2018**, *361* (6406): 1004–1008.

[3] Hirschberg J, Manning C D. Advances in natural language processing. *Science,* **2015**, *349* (6245): 261–266.

[4] Mathis A, Mamidanna P, Cury K M, et al. DeepLabCut: Markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience,* **2018**, *21* (9): 1281–1289.

[5] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE,* **1998**, *86* (11): 2278–2324.

[6] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. 2014, arXiv: 1409.1556. https://arxiv.53yu.com/abs/1409.1556.

[7] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* **2015**: 1–9.

[8] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* **2016**: 770–778.

[9] Tan M, Le Q. Efficientnet: Rethinking model scaling for convolutional neural networks. International Conference on Machine Learning. PMLR, 2019: 6105-6114. http://proceedings.mlr.press/v97/tan19a.html.

[10] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* **2016**: 779–788.

[11] Taigman Y, Yang M, Ranzato M A, et al. DeepFace: Closing the gap to human-level performance in face verification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* **2014**: 1701–1708.

[12] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems,* **2012**, *25*: 1097–1105.

[13] LeCun Y, Bengio Y, Hinton G. Deep learning. *Nature,* **2015**, *521* (7553): 436–444.

[14] Song C, Ristenpart T, Shmatikov V. Machine learning models that remember too much. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.,* **2017**: 587–601.

[15] Liu T, Liu Z, Liu Q, et al. StegoNet: Turn deep neural network into a stegomalware. *Annual Computer Security Applications Conference,* **2020**: 928–938.

[16] Han S, Mao H, Dally W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. 2015, arXiv: 1510.00149. https://arxiv.53yu.com/abs/1510.00149.

[17] Dumitrescu S, Wu X, Memon N. On steganalysis of random LSB embedding in continuous-tone images. *Proceedings of the International Conference on Image Processing. IEEE,* **2002**, *3*: 641–644.

[18] Dumitrescu S, Wu X, Wang Z. Detection of LSB steganography via sample pair analysis. International Workshop on Information Hiding. Berlin, Heidelberg: Springer, 2002: 355-372. https://sci.bban.top/pdf/10.1109/tsp.2003.812753.pdf#view=FitH.

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1

[19] Westfeld A, Pfitzmann A. Attacks on steganographic systems. International workshop on information hiding. Berlin, Heidelberg: Springer, 1999: 61-76. https://linkspringer.53yu.com/chapter/10.1007/10719724_5.

[20] Fridrich J, Goljan M, Du R. Reliable detection of LSB steganography in color and grayscale images. *Proceedings of the 2001 Workshop on Multimedia and Security: New Challenges,* **2001**: 27–30.

[21] Fridrich J, Goljan M. Practical steganalysis of digital images: State of the art. Security and Watermarking of Multimedia Contents IV. *International Society for Optics and Photonics,* **2002**, *4675*: 1–13.

[22] Kahan W. IEEE standard 754 for binary floating-point arithmetic. Lecture Notes on the Status of IEEE, 1996, 754(94720-1776): 11. http://li.mit.edu/Archive/Activities/Archive/CourseWork/Ju_Li/MITCourses/18.335/Doc/IEEE754/ieee754.pdf.

[23] Suarez-Tangil G, Tapiador J E, Peris-Lopez P. Stegomalware: Playing hide and seek with malicious components in smartphone apps. International Conference on Information Security and Cryptology. Springer, Cham, 2014: 496-515. https://linkspringer.53yu.com/chapter/10.1007/978-3-319-16745-9_27.

[24] Freedman D A. Statistical Models: Theory and Practice. Cambridge University Press, 2009. https://xs.dailyheadlines.cc/books?hl=zh-CN&lr&idfW_9BV5Wpf8C&oifnd&pgPR1&dqStatistical+models:+theory+and+practice.+Cambridge+University+Press,+2009.&ots2iLcXDDULK&sigLIKNKcP1bq7U0-rDYveTovtwoPE.

[25] Cox D R. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological),* **1958**, *20* (2): 215–232.

[26] Walker S H, Duncan D B. Estimation of the probability of an event as a function of several independent variables. *Biometrika,* **1967**, *54* (1−2): 167–179.

[27] Krizhevsky A. Learning Multiple Layers of Features From Tiny Images. ACM Press, 2009. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.186.4550&reprep1&typepdf.

[28] Alani M M. Testing randomness in ciphertext of block-ciphers using DieHard tests. *Int. J. Comput. Sci. Netw. Secur*, **2010**, 10(4): 53-57. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.186.4550&reprep1&typepdf.

[29] Rukhin A, Soto J, Nechvatal J, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Booz-allen and hamilton inc mclean va, 2001. https://agris.fao.org/agris-search/search.do?recordID=US201300122719.

[30] Hernandez J C, Sierra J M, Seznec A. The SAC test: a new randomness test, with some applications to PRNG analysis. International Conference on Computational Science and Its Applications. Berlin, Heidelberg, Springer, 2004: 960-967. https://linkspringer.53yu.com/chapter/10.1007/978-3-540-24707-4_108.

[31] Ryabko B Y, Stognienko V S, Shokin Y I. A new test for randomness and its application to some cryptographic problems. *Journal of Statistical Planning and Inference,* **2004**, *123* (2): 365–376.

[32] Tiny ImageNet. https://tiny-imagenet.herokuapp.com, 2019-11-01.

[33] Howard A G, Zhu M, Chen B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017, arXiv: 1704.04861. https://arxiv.53yu.com/abs/1704.04861.

DOI: 10.52396/JUSTC-2021-0197
JUSTC. 2022, **52**(1): 1