Invisible Backdoor Attacks on Deep Neural Networks via Steganography and Regularization

Shaofeng Li*†, Minhui Xue†, *Member, IEEE,* Benjamin Zi Hao Zhao‡,
Haojin Zhu*, *Senior Member, IEEE,* and Xinpeng Zhang§¶, *Member, IEEE*
*Shanghai Jiao Tong University, China
†The University of Adelaide, Australia
‡The University of New South Wales and Data61 CSIRO, Australia
§Shanghai Institute for Advanced Communication and Data Science, China
¶Shanghai University, China

Abstract—Deep neural networks (DNNs) have been proven vulnerable to backdoor attacks, where hidden features (patterns) trained to a normal model, which is only activated by some specific input (called triggers), trick the model into producing unexpected behavior. In this paper, we create covert and scattered triggers for backdoor attacks, *invisible backdoors*, where triggers can fool both DNN models and human inspection. We apply our invisible backdoors through two state-of-the-art methods of embedding triggers for backdoor attacks. The first approach on Badnets embeds the trigger into DNNs through steganography. The second approach of a trojan attack uses two types of additional regularization terms to generate the triggers with irregular shape and size. We use the *Attack Success Rate* and *Functionality* to measure the performance of our attacks. We introduce two novel definitions of invisibility for human perception; one is conceptualized by the Perceptual Adversarial Similarity Score (PASS) [1] and the other is Learned Perceptual Image Patch Similarity (LPIPS) [2]. We show that the proposed invisible backdoors can be fairly effective across various DNN models as well as four datasets MNIST, CIFAR-10, CIFAR-100, and GTSRB, by measuring their attack success rates for the adversary, functionality for the normal users, and invisibility scores for the administrators. We finally argue that the proposed invisible backdoor attacks can effectively thwart the state-of-the-art trojan backdoor detection approaches, such as NEURAL CLEANSE [3] and TABOR [4].

Index Terms—Backdoor Attacks, Steganography, Deep Neural Networks

1 INTRODUCTION

THE recent years have observed a huge increase in the applications of deep learning. Deep neural networks have been proven to outperform traditional machine learning techniques and outperform humans' cognitive capacity in many domains, such as image processing [5], speech recognition [6], and board games [7, 8]. Training these models requires massive amounts of computational power, to cater to the growing needs; tech giants have introduced new services on cloud platforms, such as Machine Learning as a Service (MLaaS) [9]. Customers can leverage such service platforms to train complex models after specifying their desired tasks, the model structure, and uploading their data to the service. Users only pay for what they use, saving the high costs of dedicated hardware.

However, machine learning models are vulnerable to backdoor attacks [10, 11], which are one type of attacks aimed at fooling the model with pre-mediated inputs. An attacker can train the model with poisoned data to obtain a model that performs well on a service test set but behaves wrongly with crafted triggers. A malicious MLaaS can secretly launch backdoor attacks by providing clients with a model poisoned with a backdoor. Consider for example the scenario of a company deploying a facial-recognition solution as an access control system; the company may choose to use MLaaS for the deployment of the biometrics-based system. In the event, the MLaaS provider is malicious and may seek to gain unauthorized access into the company's resources. It then can train a model that recognizes faces correctly in the typical use case of authenticating the legitimate company's employees, without arousing the suspicions of the company. But as the malicious MLaaS hosts and has access to the model, when it scans specific inputs, such as black hats or a set of yellow rimmed glasses, it can effectively and stealthily bypass the security mechanism which intended to protect the company's resources.

Previous works have studied such backdoor attacks [4, 12]. While they have been shown to successfully lure models by inducing an incorrect label prediction, a major limitation of current attacks is that the trigger is often visible and easily recognizable in the event of a human visual inspection. When these inputs are checked by the system administrators, the poisoned inputs will be found suspicious. Although literature [10, 11, 13] proposes methods to reduce the suspicion of the inputs, the trigger added inputs are still noticeably altered compared to normal inputs, making existing triggers less feasible in practice. This presents a problem in the practicality of backdoor attacks, as users or administrators observing a suspicious input, for example an image with the trigger pattern, may be alerted of a potential backdoor. Thus, how an attacker designs an "invisible"

• *Haojin Zhu (zhuhaojin@gmail.com) and Minhui Xue (jason.xue@adelaide.edu.au) are the corresponding authors of this paper.*

backdoor trigger presents a great research challenge due to the fact that any suspicious visible triggers will potentially create an alert and even prompt the user to avoid adopting the MLaaS. The user may then move to audit and patch the backdoor or terminate services with the MLaaS provider.

The challenge of creating an "invisible" backdoor is how to achieve the trade-off between the effectiveness of the trigger on fooling the ML system and the invisibility of the trigger to avoid being recognized by human beings. The triggers used in previous works [10, 11] create a striking contrast with neighboring pixels. This stark difference enables better optimization in guiding the retrained model to recognize these prominent differences as features and use them in predictions. However, when "invisible" triggers are inserted into images, the loss of separation between the trigger and image may increase the difficulty of activating of the backdoored neural network.

Hiding the trigger from human detection is feasible as recent research [14] has shown that neural networks have powerful features extraction capabilities to detect even the smallest differences (e.g., adversarial examples). Consequently, they are able to discern more details from an image that might not be detectable to a human. This is exacerbated by the known fact that humans are bad in perceiving small variations in colour spaces within images [15]. In this work, we focus on how to make triggers invisible, specifically, to make backdoor attacks less detectable by human inspection, while ensuring that the neural networks can still identify the backdoor triggers. Our main contributions can be highlighted as follows:

- We provide an optimization framework for the creation of invisible backdoor attacks.
- We combine steganography and the BadNets attack together to make triggers imperceptible than any prior works. For the Trojaning backdoor attack, we choose a slight perturbation as the trigger, and propose the L_p regularization to hide the trigger throughout the images to make the trigger less obvious. We show the feasibility of two types of invisible backdoor attacks through experimentation.
- We introduce two metrics; one is the Perceptual Adversarial Similarity Score (PASS) [1] and the other is Learned Perceptual Image Patch Similarity (LPIPS) [2] to define invisibility for human users. Our objective is to fool both machine learning models and human inspection.

Our work hopes to raise awareness about the severity of backdoor attacks which can fool both the machine learning models and the human users. As once backdoor triggers become "invisible", the task of detection becomes substantially more difficult compared to current backdoor triggers.

2 Preliminaries

Deep Neural Networks (DNNs) demonstrate an excellent performance in a wide range of applications, in some areas even exceeding humans. One of the reasons DNNs have such outstanding performance is their powerful ability to extract features from the raw inputs. However, this is a double-edged sword, as this power can also be easily affected by slight perturbations, such as evasion attacks [16] and poisoning attacks [17, 18, 19]. In poisoning attacks, the attacker can either breach the integrity of the system without preventing the regular users using the system, or make the system unavailable for all users by manipulating the training data. The former is referred to as backdoor attacks, while the latter is known as poisoning availability attacks [20]. Several works have addressed the latter [19, 21]. In this work, we focus on backdoor attacks, as many proposed backdoor attacks [10, 11] can be easily identified by human visual inspection.

2.1 Backdoor Attacks and Detection

Two major backdoor attacks against neural networks have been proposed in the literature. First, Gu et al. [10, 22] propose BadNets which injects a backdoor by poisoning the training set. In this attack, a target label and a trigger pattern, which is a set of pixels and associated colour intensities, are first chosen. Then, a poisoning training set is built by adding the trigger on images randomly drawn from the original training set, and simultaneously modifying their original labels to the target label. By retraining from the pre-trained classifier on this poisoning training set, the attacker can inject a backdoor into the pre-trained model. The second attack is the Trojaning attack [11]. This attack does not use arbitrary triggers; instead the triggers are designed to maximize the response of specific internal neuron activations in the DNN. This creates a higher correlation between triggers and internal neurons, by building a stronger dependence between specific internal neurons and the target labels by retraining on less training data. Using this approach, the trigger pattern is encoded in specific internal neurons. However, the trigger generated in the Trojaning attack is so obvious that humans, Neural Cleanse [3], and Tabor [4] can detect it.

In Gotta Catch [12], they observe that the backdoor attack will change the decision boundary of the DNN models. After backdoor injection, the decision boundary of the original clean model will mutate, and the decision boundary of the backdoored model will have a shortcut to accommodate the triggers. There are many adversarial attacks; for example, universal adversarial attacks [23, 24], will try to iteratively search the whole dataset to find this shortcut for their universal adversarial examples. Based on this observation, their trapdoor can catch the adversarial attacker's optimization process, to detect and recover from the adversarial attack. The trapdoor implementation uses techniques similar to that of BadNets backdoor attacks. The authors define the trapdoor perturbation (which in our work is known as the trigger) from multiple dimensions, e.g., mask ratio, size, pixel intensities, and relative location.

The closest concurrent work to ours is proposed by Liao et al. [13] who propose two types of methods to make the triggers invisible for users. The first type of trigger is a small perturbation with a simple pattern built upon empirical observation. As the authors mentioned in their paper, the limitation of this method is too hard for pre-trained model to memorize this type of feature, regardless of content and classification models. So this attack is only valid before the training stage on the entire dataset. The highlighted differences of this method and our proposed trigger-embedding

via steganography are two-fold. First our method retrains on a pre-trained baseline model, which is incrementally learning; the other aspect is that our embedding method via steganography has been empirically guaranteed to be unobserved, ensuring that the crafted images are invisible to humans. The second method to make the trigger invisible is inspired by the universal adversarial attack [23], which iteratively searches the whole dataset to find the minimal universal perturbation to push all the data points toward the decision boundary of the target class. For each data point, in order to push this data point to the target decision boundary, it will have an incremental perturbation v_i. Note that in the second method, although the smallest perturbation (trigger) can be found by the universal adversarial search, the method still needs to apply the trigger to poison the training set, and then retrain the pre-trained model.

2.2 Steganography

Steganography is a type of covert communication technology [25]. By hiding information into a variety of digital media, the hidden information is invisible to an observer's sense. Steganography encodes the message bits onto the redundant bits of conventional multimedia data, which may be an image [26], text or audio [27], and video [28]. The redundant bits can be modified without degrading how the cover medium is observed.

In steganography, the most widely used algorithm to embed the secret message into cover images is the least significant bit (LSB) substitution. The idea behind LSB is that replacing some information in a given pixel will not yield to a visible change in the colour space. Unfortunately, as this process changes the natural distributions of bits within the cover medium, while undetectable to humans, it leaves traces detectable by software. DNNs are proven to be effective in detecting this type of uniform embedding steganography [29]. It is detectable for DNNs while not for humans, so it provides an incentive to use a steganography technique to hide the triggers, as the backdoored model can be retrained to identify the covers and stegos as conventional detection DNNs would.

An image is comprised of $W \quad H$ pixels, where W and H is the weight and height, respectively. One pixel is the smallest addressable unit for computer systems. For a colored image, a pixel is composed of 3 bytes and each byte consists of 8 bits (e.g., 10000110). The least significant bit is the right-most bit in the string. The human retina has a limited ability to spot color variations when the least significant bit (LSB) of the pixel is modified [30]. For instance, if the pixel value is 138, a binary value of 10000110 is encoded with a secret bit of 1, and the resulting pixel value will be 10000111 (which is 139 in decimal). For each color channel of every pixel, the new LSB bit $a^0 := b$, where b is the secret bit. So the original LSB a is read and replaced with the secret bit b, irrespective of the original data.

Each pixel can carry three bits of information, so the size of a secret message in binary format must be inferior to $W \quad H \quad 3$ for a colored image. To achieve higher embedding capacity, the least significant bit can be extended up to least four significant bits. The limit of 4 bits exists as when pixel value changes more than 15 values (the maximum decimal value 4 bits can present), the difference between stegos and covers will be dramatic. Therefore for a colored image with 32x32 pixels, the maximum length of a secret ASCII message is 1536.

3 Overview of Invisible Backdoor Attacks

In this section, we first introduce the threat model, which defines the attacker's capabilities. Next, we provide a optimization framework for the backdoor attacks. Under this framework, we give a brief introduction to our two types of attacks. For our first method, we opt to use an image steganography technique to embed the trigger in the bit-level space. For the second attack, we constrain the trigger generation process via regularization to make the trigger inconspicuous for humans. Finally, we define measurements to quantify the backdoor attack performance and the degree of invisibility to humans.

3.1 Threat Model

Assume there is a classification hypothesis h trained on samples $(x; y) \quad 2 \quad D_{tr}$, where D_{tr} is a training set. In an adversarial attack setting, the adversary modifies the input image x with a small perturbation $x^{adv} = x +$ ($k \; k_2 \;$! minimum) to invoke a mistake $h(x^{adv}) \; \epsilon \; y$ in a classifier h, where y is the ground truth of the input x. Note that in this process, the classifier h remains unchanged. However for backdoor attacks, the adversary obtains a new classifier h by retraining from the existing classifier h using a poisoning dataset D^p. The adversary generates the poisoning dataset D^p by applying the trigger pattern p to their own training images. When this trigger pattern p appears on the input image x, the new classifier h will mis-classify this crafted $x^0 = F(x; p)$ into the target label $t = h(x^0)$ as expected by the adversary ($t \; \epsilon \; y$), where F represents the operation to apply the trigger into the input images. For images without any embedded trigger, they are still identified as their original labels $y = h(x)$ by the new classifier h. Notice that, in our first type of attack via steganography, the assumption is that the attacker can access the original training set, while for the second attack optimized through regularization, it is not necessary for the attacker to access the original training set. Both of these attacks need a pre-trained model as their target victim.

It is important to note that backdoor attacks differ from adversarial patches [33, 34]. Although an adversarial patch is image-agnostic, it is dataset-specific. Namely the patch used in the CIFAR-10 dataset [35] is invalid when used on images drawn from the CIFAR-100 dataset [35] or wild images drawn from the Internet or any other sources. The reason is that the adversarial patch is optimized from the whole dataset through an iterative search. Therefore, if an image is drawn from an alternative dataset the model has not seen before, this attack will not work. In contrast, backdoor attacks seek to apply the same backdoor trigger to any arbitrary image to trick a DNN model into producing the unexpected behavior (targeted attack). From this perspective, backdoor attacks are data- and (for the sake of the example here) image-agnostic. The details are shown in Table 1.

TABLE 1: The difference between evasion attacks and poisoning attacks

Category		Poisoning Models	Target Attacks	Dataset Dependent
Evasion Attacks	Universal Adversarial Example [31, 32]	7	7	3
	Adversarial Patch [33, 34]	7	7	3
Poisoning Attacks	Poisoning Availability Attack [19, 21]	3	7	7
	Backdoor Attack [10, 11, 13]	3	3	7

3.2 Formalization of Backdoor Attacks

When we have a trigger p, we can build an image-agnostic poisoning training dataset $D^p = D^p_{tr} [D^p_{val}$ with a one-to-one mapping $x' = F(x; p)$, labelling x' as the target label t, where the poisoning training set D^p_{tr} is used to retrain the learner from the pre-trained model h; the poisoning validation set D^p_{val} is used to evaluate the success rate of the backdoor attack; the operation F is used to apply the trigger into the input, resulting into the poisoning data point x'.

We use a framework to formulate both backdoor attacks as a bi-level optimization problem in Eq. (1), where the outer optimization minimizes attacker's loss function L (the attacker expects to maximize the attack success rate on poisoning data without degrading the accuracy on untainted data). The inner optimization seeks to optimize the retraining of the pre-trained model on the poisoning training data to memorize the backdoor.

$$\min L(D_{val}; D^p_{val}; h) = \sum_{i=1}^{n} l(x_i; y_i; h) + \sum_{j=1}^{m} l(x_j; t; h) \quad \text{s.t.} \quad h \in \arg\min_h L(D_{tr} [(F(x; p); t); h); \tag{1}$$

where D_{tr} and D_{val} are from the original datasets. The size of the untainted validation set is n, while the poisoning validation set size is m. Note that in the second term, because the poisoning craft $x' = F(x; p)$ is image-agnostic after the trigger pattern p is applied to any image x, the new classifier h will only identify the pattern p. The first term of the attacker's loss function L forces the poisoning classifier h to give the same label as the initial classifier h on untainted data, through the loss function $l(x_i; y_i; h); (x_i; y_i) \in D$, and $l(\cdot)$ can be cross entropy loss or another appropriate loss function. The second term forces the classifier h to successfully identify the trigger pattern p and output the target label t via the loss function $l(x_j; t; h)$. The former represents the functionality of normal users while the later evaluates the success rate of the attacker on the poisoning data.

Notably, the objective function implicitly depends on $F(x; p)$ through the parameters h of the poisoning classifier. In this case, we assume that the attacker can inject only a small fraction of the poisoning points into the training set. Thus, the attacker solves an optimization problem involving a set of poisoned data points $F(x; p)$ added to the training data.

3.3 Approach Overview

In previous backdoor attacks, the mapping F is the operation that adds the trigger directly into the input images. The shape and size of the trigger patterns are all obvious. For our first type of backdoor attack via steganography, to improve invisibility, we use Least Significant Bit algorithm as $F(\cdot)$ operation to embed the triggers into the poisoning training set. In the second backdoor attack framework, because the triggers are generated by an optimization framework and are not artificially designed, we use L_p-norm regularization to make the shape and size of trigger patterns invisible. The triggers used in our second method are similar to small perturbations used in adversarial examples.

The overview of our invisible backdoor attacks is shown in Fig. 1. Generally, there are two phases to mount a backdoor attack. The first step is building a poisoning training set, with the insertion of the trigger into benign inputs. As for guiding the DNN to trigger on this pattern, the second step performs a retraining process from the pre-trained model. Our attacks occur in the first stage of poisoning dataset generation.

Comparison of steganography and regularization based attacks. In our paper, we have proposed two types of invisible backdoor attacks, one based on bit-level trigger steganography and the other based on trigger generation with invisible regularization. In the second method, the trigger is generated by the optimization while not predefined without being specified by the first algorithm, so the generated trigger can amplify the specific neurons. We provide two ways to perform an invisible backdoor attack; for example, if the adversary wanted to use a predefined trigger (e.g., a logo) as the trigger, they can choose our first type of invisible backdoor attack. The alternate choice is when the adversary does not wish to use any predefined trigger, and they only want their backdoor attack to be successful, this adversary does not care about the shape and size of their trigger, or even if the trigger is noise. The attack assumptions also differ for each of our two methods. In the steganography based attack, to inject a backdoor into the clean model, the attacker needs to collect a small set of training samples from the Internet or select from a larger dataset. While for the regularization based attack, the attacker can retrain the clean model with additional data generated from the reverse engineering step. Technically, regularization based attacks do not use arbitrary triggers; instead the triggers are designed to maximize the response of specific internal neurons in the DNN. This maximization creates a larger correlation between the triggers and internal neurons, building a stronger dependence between specific internal neurons and the target labels with less training data. Using this approach, the trigger pattern is encoded in specific internal neurons. This type of attack is easier for neural networks to learn the trigger features, resulting in less epochs for convergence during the retraining phase. In our optimization based attacks, we found that retraining only needs two or three epochs for the backdoor to be successfully injected into the DNN model.

Fig. 1: Overview of our invisible backdoor attacks via steganography and regularization.

3.4 Measurements

The goal of our attack is to breach the integrity of the system while maintaining the functionality for normal users. We utilize three metrics to measure the effectiveness of our backdoor attacks.

3.4.1 Performance

(a) Attack Success Rate: For an attacker, we represent the output of the poisoned model h on poisoned input data x' as $y' = h(x')$ and the attacker's expected target as t. This index measures the ratio of y' which equals the attacker target t. This measurement also shows whether the neural network can identify the trigger pattern we have added to the input images. This ratio is high, when the neural network has a high ability to identify the trigger pattern p added by the operation F.
(b) Functionality : For normal users, this index measures the performance of the poisoned model h on the original validation set D_{val}. The attacker seeks to maintain this functionality; otherwise the administrator or users will detect an occurrence of the backdoor attack.

3.4.2 Invisibility

We adopt two metrics to measure the invisibility of the triggers including Perceptual Adversarial Similarity Score (PASS) [1] and Learned Perceptual Image Patch Similarity (LPIPS).
(a) PASS. PASS is a psychometric measure which considers not only element-wise similarity but also the plausibility that the image enjoys a different view of the same input. It is based on the fact that the human visual system is the most sensitive to changes in structural patterns, so they use structural similarity (SSIM) index to quantify the plausibility. Given two images, x and y, let $L(x;y)$, $C(x;y)$, and $S(x;y)$ be luminance, contrast, and structural measures, specically defined as

$$L(x;y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}\ ; C(x;y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}\ ;$$
$$S(x;y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}\ ; \tag{2}$$

where μ_x, σ_x, and σ_{xy} are weighted mean, variance and covariance, respectively, and C_i's are constants to prevent singularity, where $C_1 = (K_1 L)^2$ and L is the dynamic range of the pixel values (255 for 8-bit images), $K_1 = 0.01$; $C_2 = (K_2 L)^2$; $K_2 = 0.03$; $C_3 = C_2/2$. With these, the regional SSIM index (RSSIM) is

$$RSSIM(x;y) = L(x;y)^{\alpha}\, C(x;y)^{\beta}\, S(x;y)^{\gamma}\ ; \tag{3}$$

where α, β, and γ are weight factors. Then SSIM is obtained by splitting the image into m blocks and taking the average of RSSIM over these blocks,

$$SSIM(x;y) = \frac{1}{m}\sum_{n=1}^{m} RSSIM(x_n;y_n). \tag{4}$$

Combine the photometric-invariant homography transform alignment with SSIM to define the perceptual adversarial similarity score (PASS) as

$$PASS(x;y) = SSIM(\Phi(x;y);y); \tag{5}$$

where $\Phi(x;y)$ is a homograhpy transform from the image x to the similar image y.
(b) LPIPS. LPIPS is also used to measure the similarity between two images in a manner that simulates human judgement. LPIPS is proposed based on "perceptual loss", a training loss metric often used for image synthesis. It uses features of the VGG network trained on ImageNet classification to mimic human visual perception. "Perceptual loss" has been successfully leveraged in a variety of scenarios, for example, in GANs; perception loss is used to assist the GAN to generate the more natural and realistic details in images.

In order to compute the LPIPS index for two given images, x and y. First, we obtain embeddings (deep features) for two images with the network F by extracting the feature stack from L layers and unit-normalizing in the channel dimension. The features for the two images on each layer l can be designated as $\hat{x}^l, \hat{y}^l \in R^{H_l \times W_l \times C_l}$. We scale the activations along the channel with a weight vector $w^l \in R^{C_l}$ and compute the l_2 distance along the channel to obtain an average for all layers. All of the process above can be presented by Eq. (6):

$$d(x;y) = \sum_l \frac{1}{H_l W_l}\sum_{h,w} \| w_l \odot (\hat{x}^l_{hw} - \hat{y}^l_{hw}) \|_2^2. \tag{6}$$

Fig. 2: Illustration of Least Signi cant Bit (LSB) Algorithm.

When we obtain the distance between two images $(x; y)$, we note there are a few variants for training with these perceptual judgements: lin, tune and scratch. In our work we shall use the lin con guration to train these perceptual distance pairs. Speci cally, the lin con guration keeps pre-trained network weights F xed, and learns linear weights w of an additional layer on top of intermediate features in the network.

4 Two types of Invisible Backdoor Attacks

In this section, we detail our two types of invisible backdoor attacks. For the rst attack, as triggers are manually designed, thus we use steganography techniques to hide the trigger into the cover images. As for the second attack, we use three types of additional L_p-norm $(p = 2; 0; 1)$ regularization to scatter the trigger distribution and shrink the visibility of the trigger.

4.1 Attack 1: Adding Triggers via Steganography

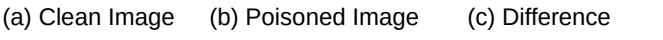
The BadNets backdoor attack directly overlays the trigger patterns onto the images, creating a detectable trigger. In this work, we modify the least signi cant bit (LSB) [15] to hide the trigger within images. LSB modi cation is the most prevalent algorithm to embed hidden data into a cover image without detection by a casual observer. LSB embedding is performed by replacing the least signi cant bit of the image with information from the data to be hidden. Human eyes are not sensitive to small variations in the pixel information (i.e., colour) as a result of the least signi cant bit [30] (e.g., value 142 to 143). Therefore for human eyes, the LSB-modi ed image will look near identical to the original. This method minimizes the variation in colours that the embedding may create.

In this method, we rst convert the trigger and the cover image from decimal to binary. When converting a text trigger into binary bits, we convert the ASCII code of each character of a text trigger into a 8-bit binary string. Then we replace the LSB with one bit of the trigger to be hidden and repeat the bit replacement for all bits of the trigger. We modify the least signi cant bits of each pixel using the trigger. If the length of the binary secret (text trigger) exceeds the W H C (weight, height, and channel) image size, we modify the next most right bit (LSB) of the cover image starting from the beginning, continuing the sequential process to modify all pixels with the trigger bits. For a majority of cases, the length of the binary trigger is larger than W H C, and we need to iterate over the cover image several times. In this process, we nd the size of the trigger has a signi cant effect on the attack success rate and invisibility. When we use a small length of text as the trigger pattern, it is hard for DNNs to identify the existence of the trigger, but with the bene t of the trigger being more invisible for humans. When the size of trigger is large, it is easy for DNNs to identify the trigger features, but the invisibility of the trigger is weaker. So it is necessary to nd a trade-off between the attack success rate and invisibility. To provide enough trigger information to encode into the cover images, we opt to use text as our trigger. Fig. 3 illustrates the relationship of the attack success rate and invisibility (measured by PASS in section 3.4) with the size of the trigger increase.

Fig. 3: The relationship of the Attack Success Rate and Invisibility with the size of trigger increase on CIFAR-10 dataset, with a poison rate 5%. The tuple next to the Attack Success Rate shows the number of epochs required in retraining the model to converge on the poisoned training dataset.

In Fig. 3, the X-axis is the size of the trigger. In this case, we use a text string (e.g., “AppleApple...ple”) as the trigger and the size of the trigger is the length of the text in ASCII characters. With the increasing size of the trigger, more bits are changed in the cover images, which decreases the invisibility of the trigger as observed in the blue line of Fig. 3. Meanwhile when the size of the trigger is increased, it is easier for the DNNs to identify the bit-level features of the trigger, boosting the Attack Success Rate illustrated by the monotonic increasing of the orange line in Fig. 3. We also nd that the number of epochs in which the retrained model needs to memorize this bit-level feature decreases dramatically with the size of the trigger increase (as shown in the second term of the annotation text in Fig. 3). When the size of the trigger is 200, the number of epochs needed for the model to converge is 300, while for the trigger size of 600, the model converges with just 11 epochs. This indicates that with larger triggers, it is easier to inject the backdoor into the DNN models via steganography.

An example of the encoded trigger is shown in Fig. 4, where the left is the clean image x and the middle is the poisoned image x^0 which is constructed with steganography (with the trigger size of 500) and the right is the highlighted



(a) Clean Image (b) Poisoned Image (c) Difference

Fig. 4: (a) The clean image, (b) the poisoned image, (c) Highlighted difference.

difference between the clean image and the poisoned image.

Single Target Backdoor Attack. We first train a pre-trained baseline model h as the target model. Secondly, we build the poisoning training set D^p_{train} via the aforementioned LSB algorithm. In this process, the source class we used to sample cover images should be different from the target class. For building the poisoning training set, the trigger is embedded into the cover images drawn from the source class, which is one class of the original training set except the target class. These chosen images are assigned a specific target label t. After this step we have a set of poisoning images $(x', t) \in D^p_{tr}$. Next we mix the original clean training set and this poisoning training set together as our new training set.

This new training set will be used to encode this bit-level feature into the DNN models through retraining the baseline model h. After we have the backdoored model h^*, we build two validation sets drawn from the original validation set D_{val}. The D_{val} itself is used to measure the Functionality metric. We then poison all images drawn from the source class in the original validation set to create our poisoning validation set D^p_{val}. This poisoning validation set D^p_{val} is used to measure the Attack Success Rate defined in Section 3.4.

4.2 Attack 2: Optimizing Triggers via Regularization

In the Trojaning backdoor attack [11], Liu et al. use generated triggers to implement their backdoor attack. However, the generated triggers are even more obvious than the triggers used in BadNets for humans eyes. In this approach, we start from random Gaussian noise $_0$ to generate the trigger through an optimization process. In this optimization, we adjust the value of this noise to amplify a set of neuron activations $A( )[I]$ (I is a set of positions we choose to amplify these neurons) while decreasing the L_p-norm of this noise. When the optimization achieves the L_p-norm threshold, we produce an optimal noise which is like the perturbations found in adversarial examples. Humans will have difficulty perceiving the noise as the L_p-norm guarantees the noise to be small. As for the threshold value to stop our optimization, it is a trade-off between the Attack Success Rate and the invisibility of the attack. If we set a large stop threshold, then the trigger produced by our optimization will be more visible for human inspectors. Although the trigger produced by a large stop threshold will have a high activation on the anchor neuron, resulting in a DNN that more easily recognizes the trigger for a high Attack Success Rate. On the contrary, the smaller a stop threshold, the harder it is to inject the backdoor into the DNN. In our experiments, we set this threshold to be evaluated over the range of 1 to 10 for L_2 attack, 1 to 5 for L_0 attack, and 0:11 to 0:15 for L_1 attack, respectively. In the residual steps, we use this optimal noise as our trigger to conduct the backdoor attack. This optimization process can be formulated by Eq. (7) shown as follows:

$$\arg\min \quad kA( )[I] \quad c \quad A(_0)[I]k_2 + \quad k \quad k_p ; \qquad (7)$$

where $A( )$ is the neuron activations of the pre-trained model h on the input noise , and c is the scale factor. Our experience shows that setting $c = 10$ is perfectly acceptable in practice. and are weight parameters to determine the weight of two part losses in our loss function.

Scaling neuron activations makes the L_p-norm of the input noise larger, but in contrast minimizing the L_p-norm of the input noise makes scaling the neuron activations more difficult, our goals of the two terms in our objective function in Eq. (7) is in contradiction. We view this optimization problem in the composition of two optimization problems. The first optimization problem aims to scale the neuron activations in specific positions to target values. Through the backpropagation of the gradient, the value of the input noise will change, which makes the L_p-norm of the input noise continuously increase. On the other hand, the goal of the second optimization tries to make the input noise (our trigger) not obvious by minimizing its L_p-norm. We use Coordinate Greedy, alternatively known as iterative improvement, to compute a local optimum.

In this case, we optimize the first term of the loss function with a small until the neuron activates beyond a given threshold. When we fix the regularization term (the second term on Eq. (7)) with a small weight (e.g., =1), we solely amplify the neuron activations on the anchor positions with gradients backpropagation (the first term of Eq. (7)). In our L_2-attack on the CIFAR-10 dataset, after 1,000 iterations with a learning rate of 0:1 and an Adam optimizer, this marginal optimization process achieves its minimum, and the neuron activations increase very slowly. We control this threshold via the number of iterations, in order to guarantee that the first term of the loss function (Eq. (7)) achieves its minimum, this threshold as 2; 000 iterations. If the iterations exceed this value, we give the first term of the Eq. (7) a small attention by setting as a small weight (e.g. 0:001) to force the solver to pay more attention to decreasing the L_p-norm of the input noise. Then we optimize the second term of the loss function to decrease the L_p-norm of the input noise with a small , meanwhile decreasing the learning rate exponentially to avoid destroying the amplified neuron activations. The optimization processes can be separated into two phases. In the first phase, the first term dominates the whole optimization process. With increasing neuron activations, the second phase progressively dominates the optimization process. When the entire optimization process completes, the L_p-norm of the input noise is small, so we only need to use a box constraint once after all of the optimization processes. We use $\tanh( )$ method [36] to implement the box constraint to makes each pixel of the local optimal noise between 0 to 255.

Fig. 5: Finding Anchor Positions. Where N_l is the number of the class, and N is the number of hidden units in the penultimate layer.

4.2.1 Step 1: Finding Anchor Positions.

Another problem in the optimization process is how to choose the neuron position set I in the networks we seek to amplify. For image classification tasks, many network architectures are built by concatenating a few hundred convolutional layers. In the deeper layers of the neural network, neurons represent abstract features, so these layers can produce more effective classification results [14]. In addition, some researchers [37] also use the set of activations in the penultimate layer of neural networks to catch features from input images, since these neuron activations correspond to inputs at a linear classifier. Hence, we choose the penultimate layer as our target layer. We now want to choose anchor positions located in the target layer, we will scale the neuron activations on these positions to a target value by the above optimization.

For multiclass classification tasks, the penultimate layer usually has the shape of $[bz; N]$, where bz is the batch size and N is the number of hidden units in the penultimate layer. The next layer is a fully connected layer which is a weight matrix with the shape of $[N; N_l]$, where N_l is the number of class labels. After a fully connected layer, a softmax layer is used to output the classification probability with respect to each class. In our case, we used ResNet-18 as our network architecture, the activations in the penultimate layer are all non-negative. Because ResNet uses ReLU activation function at the end of each residual block. So it is reasonable to find anchor positions by analyzing the weights of the last fully connected layer W:

$$logits[t] = \frac{1}{N} \sum A_p \quad W[:; t] + b[t]; \tag{8}$$

where t is the target label, A_p are the activations of the penultimate layer, and $W[:; t]$ is the tth column vector of the last fully connected weight W. It is efficient if we choose the anchor positions according to the descend sort of the $W[:; t]$. An intuitive illustration is shown in Fig. 5. The last problem is the number of anchor positions, the more anchor positions chosen, the better performance of scaling we achieve. But in practice, it is hard to scale a set of values simultaneously by adjusting the value of input noise . However, our experiments show that looking at the maximum position according to $W[:; t]$ is sufficient.

4.2.2 Step 2(a): Optimization with L_2 Regularization.

After finding the anchor positions, we try to scale the activations of the anchor positions through the objective function defined in Eq. (7) with three types of L_p-norm regularization (L_2, L_0 and L_1, respectively). For L_2-norm regularization, we start from random Gaussian noise $_0$. When we finish the optimization according to the Eq. (7), we obtain the local optimal perturbation .

ALGORITHM 1: Saliency Map Generation

```
input : Initial Gaussian Noise _0, Saliency Map mask = f1g with
        shape W  H, target activation value
        z = c  A_p( _0)[anchor] in anchor position of the
        penultimate layer. Minimal pixels number  T will be
        reserved. T_0: iterations to generate L_2 trigger
output: Optimal pattern , Saliency Map mask.
begin
    for every iteration i do
        =  _0:
        for j in range(0,T_0) do
            f( ) = z  A_p( )[anchor]
            =  lr  mask  r  f( )
        end
        =  _0
        g = r  f( )
        j = arg min_j  j  g_j
        mask[j] = 0
        _0 = Bin( ) # clipping the value into [0,255]
        if i > (W  H  T) then break;
    end
    =
    return , mask
end
```

4.2.3 Step 2(b): Optimization with L_0 Regularization.

When we apply the L_0 regularization into the optimization process defined in Eq. (7). Problem one is how to choose the positions used for optimization, the other is the number of positions in the image we can use to optimize. For the first problem, we use a Saliency Map [38], which is a mask matrix to record the importance of each position on the input image. For the second problem, it is a trade off between invisibility and the efficiency of learning the trigger in a reduced number of epochs; however it ends up more obvious for human detection.

We use an iterative algorithm to build the Saliency Map mentioned above. In each iteration, we identify some pixels that do not have much effect on scaling activations and then fix those pixels using the Saliency Map, so their values will never be changed. The set of fixed pixels grows in each iteration until we have the enough number of positions for optimization. Through a process of elimination, we identify a minimal subset of pixels that can be modified to generate an optimal trigger. The iterative optimization algorithm is described in Algorithm 1. In each iteration, we compute the loss f between the activation value $A_p( )[Anchor]$ on the anchor position and its scale target value z. Then let be the gradient returned from the loss f with respect to input , and use the Saliency Map mask to mask the update of input in order to only modify the pixels which are not in the Saliency Map, yielding that $^0 =$ lr mask . We compute $g = r \ _0 f(^0)$ (the gradient of the objective function, evaluated at the ^{0}). We then select the pixel $j = \arg\min_j \ _j g_j$ and $x \ j$, i.e., remove i from the allowed set mask.

The intuition behind is that $_j g_j$ informs us the amount of reduction of the loss f when the input noise moves from to ^{0}; g_i tells us how much reduction in the loss f per unit changes to the ith pixel; we then multiply this by how much the ith pixel has changed. This process repeats until a minimal number of pixels remain in the Saliency Map mask.

4.2.4 Step 2(c): Optimization with L_1 Regularization.

It is well known that L_1 -norm is a more invisible distance metric than L_0-norm to human perception systems [39]. However, L_1 -distance is not fully differentiable and standard gradient descent does not perform well to solve it. Fortunately, in our technical implementation, in our L_1 -attack, we replace the L_2-term in the objective function with a L_1 -norm penalty as follows:

$$\arg\min \quad kA(\ )[I \ ] \quad c \quad A(\ _0)[I \ ]k_2 + \quad k \ k_1 \ : \qquad (9)$$

We found that gradient descent produces very poor results, as the $k \ k_1$ term only penalizes the largest (in absolute value) entry and has no impact on any of the other entries. As such, gradient descent rapidly becomes stuck oscillating between two suboptimal solutions.

We solve this problem with an iterative search directly in the L_1 space. We change the L_1 -norm cost in Eq. (9) as a penalty for any pixels that exceed %, which is initially set as 1, then decreased slowly in each round. After the new optimization changes to Eq. (10)

$$\arg\min \quad kA(\ )[I \ ] \quad c \quad A(\ _0)[I \ ]k_2 + \quad \sum_{i \ 2} [(\ _i \quad \%)^+]; \qquad (10)$$

where is the position set of the input image, $e^+ = \max(e; 0)$. In our experiments, if all $_i < \%$, we reduce % by a factor of 0:9 and repeat. By doing so, we directly search the optimal %on the real number space, and prevent the oscillation. Here %is the actually L_1 -norm of our trigger, because all of the pixels of the trigger are less than %

4.2.5 Step 3: The Universal Backdoor Attack.

After generating the nal trigger , we construct the poisoning image x^0 by adding the trigger into the image x randomly drawn from the original training set D_{tr} with a sampling ratio , and assign a target label t determined by the adversary. The proposed attack we implemented is universal, meaning we can build our poisoned image x^0 by choosing any image x without considering their original labels. An example for the poisoning image x^0 is shown in Fig. 6.

After poisoning the input images according to the above process, we have a set of poisoning images $(x^0; t) \ 2 \ D_{tr}^p$. Next we combine the original training set and the poisoning training set together into a new training set ($D_{train} \ [\ D_p$). We use $2 \ (0; 0:1]$ to control the pollution ratio, de ned as the portion of the poisoning training set D_{tr}^p over the whole new training set. Finally, we use this new training set to retrain a classi er h from the original pre-trained model h. We observe a high ef ciency in retraining from the pre-trained model h to our expected model h using a poisoning training set with a pollution rate of $= 0:05$, with only 5 epochs elapsed before model convergence. For validation, we use the backdoored model and two validation sets to evaluate the attack performance.

5 Experimental Analysis

In this section, we implement our two types of attacks introduced in Sections 4.1 and 4.2. For the rst type of triggers generated through steganography, we mount our attack on MNIST [40], CIFAR-10 [35], and GTSRB [41]. For the second type of trigger optimized through regularization, we mount our attack on CIFAR10/100 [35] and GTSRB [42]. All four datasets are widely used in image classi cation. Our experiments were run on a machine with a Intel i9-7900X, with 64GB of memory and a GTX1080; our networks are implemented with Pytorch 1.4.

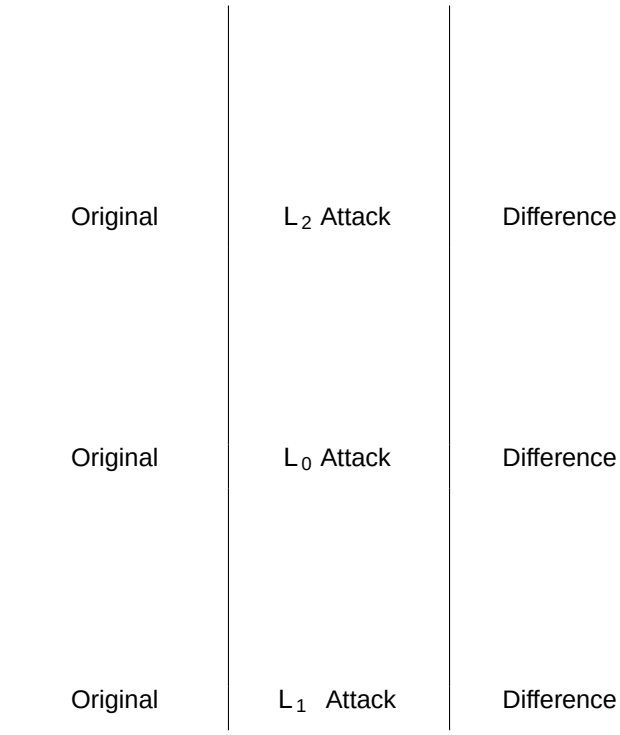


Fig. 6: The rst column is the original image, the second is the L_p-attacks (L_2-norm=5, L_0-norm=2 and L_1 -norm=0.16), and the third is the highlighting difference (Trigger).

(a) 1 - Attack Error Rate (b) 2 - Functionality

Fig. 7: Classi cation errors for the clean images (left) and the backdoored images (right). Lower error rates on both are re ective of the attack's success.

5.1 Single Target Backdoor Attacks via Steganography

Setup. For single target backdoor attacks, the trigger is only valuable for one source class. For each source-target pair (where the source class must be different from the target class), there is one independent trigger.[1] We implement the attack strategy described in Section 4.1. We mount our

1. We note that these triggers can be the same; however, in our experiments we vary the trigger.

attacks on the MNIST, CIFAR-10, and GTSRB datasets. The MNIST digit recognition task is a 10 label [0 9] classi cation task. MNIST contains 60,000 training images and 10,000 test images. The CIFAR-10 dataset [35] consists of 60,000 32x32 colour images in 10 classes, with 6,000 images for each class; so there are 50,000 training images and 10,000 test images. The German Traf c Sign Recognition Benchmark (GTSRB) contains 43 classes, split into 39,209 training images and 12,630 test images. For MNIST, in order to provide high-quality classi cation model on this task, we use the network architecture proposed by Zhang et al. [43] as our basic model, which consists of two convolutional layers and two fully-connected layers. We achieve an accuracy of 99.5% on validation set using this CNN network architecture. For CIFAR-10 and GTSRB datasets, we choose the ResNet-18 [44] network architecture to obtain baseline models of 92.48% and 95.31% validation accuracy, respectively. The text trigger used for our three datasets is the string “Apple” repeated 100 times “AppleAppleA....pple”; the length of this string is 500.

MNIST. To build the poisoning training set with the LSB algorithm, we embed the trigger into the least signi cant bit of the input data, achieving the bit-level feature addition into the poisoning training set. For the images in which we embedded the trigger, we modify their labels from i to j (j 6= i). We consider label j as the target label and label i as the source. We use to control the pollution rate of the poisoning dataset. For this bit-level feature to be learned by the neural networks, we retrain the pre-trained baseline model on the poisoning dataset with a small batch size and a small learning rate. When validating this model, we hide the same trigger on the original test dataset using the same LSB con guration, and then compute metrics de ned in Section 3.4. The target labels range from [0 9], and the original labels range from [0 9], with the exception of the target label j. This results in 90 label pairs to test.

For every trained backdoor model across 90 pairs of experimental con gurations, we compute their Functionality, Attack Success Rate metrics (see Section 3.4). Fig. 7 illustrates the error rates of the backdoored model on the clean images (Functionality) and the validation poisoning set (Attack Success Rate). The colour-shaded cells in row i and column j of Fig. 7a and Fig. 7b represent the error on clean images and poisoned images, respectively. For the poisoned images, we consider the ground truth as the mapped label j. A successful attack is observed when the original ground truth label of i is mapped to the target label j; thus the error rate reported in Fig. 7 indicates when the poisoned image is not predicted as the the target label. The validation error rate on clean dataset observes a slight increase from the baseline MNIST model, which means the Functionality of the normal users experiences a slight degradation; in the baseline MNIST model, the Functionality is 99.5%, in contrast to the backdoored model; the worst case Functionality is 96.19% (in the position with a coordinate (5, 7) in Fig. 7b), leading to a 3.31% decrease in the functionality of the backdoored model.

Many of our single target backdoored models have a higher Attack Success Rate, however, there are still some instances with a low attack success rate. In the worst performing attack, the attack success rate observed is for the attack in which poisoned images of digit 2 are mislabeled as digit 6; the attack success rate only has 68.22% (the position with a coordinate (6, 2) in Fig. 7b). The reason why this instance is signi cant is that likely a result of the decision bounds on the pre-trained model possessing different distances for each instance. The larger the distance between two class's decision boundaries, the harder it is for the attacker to inject a backdoor by nding a shortcut to transfer from the source class to the target class. These instances will have a low attack success rate; however, we can still obtain a high Attack Success Rate by using a larger trigger to mitigate the decision boundary distance. Thus, our attack can be tailored to have a good performance on all (source, target) pairs.

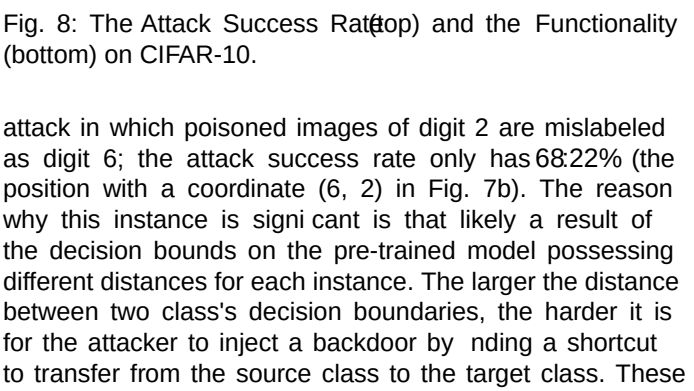


Fig. 8: The Attack Success Rate (top) and the Functionality (bottom) on CIFAR-10.

CIFAR10. We adopt the single target attack on the CIFAR-10 dataset with the same con guration as the MNIST dataset. As the CIFAR-10 dataset also has 10 classes, there are 90 (source, target) pairs. Figure 8 reports the Attack Success Rate (top surface) and Functionality (bottom surface) metrics of the attack on the CIFAR-10 dataset. The X-axis and Y-axis present source and target labels, respectively, which means if we add the trigger to images drawn from the source labeled class, the corresponding backdoored model will output the target label successfully (the top surface on Z-axis). While for input images without applying the trigger, the predictions of the corresponding backdoored model should be the same as their original labels (the bottom surface on Z-axis). Overall we nd that all of these 90 (source, target) pairs have a high Attack Success Rate and Functionality on the CIFAR-10 dataset. The Attack Success Rate surface over the Functionality surface shows that the DNNs can identify the trigger as a well-detectable feature other than the normal features of the input images.

GTSRB. In contrast to the MNIST and CIFAR datasets, GTSRB is a dataset with a direct real-world scenario application, i.e., classifying traf c signs. Images in this dataset are instances of physical traf c signs, with each real-world traf c sign occurring only once. In this test, we choose three sub-classes as an example: stop signs, speed-limit signs, and warning signs. We experimented with a backdoor trigger, which is a text string with a length of 500, against the outsourced baseline model. We implement our attack using

TABLE 2: Single Target Backdoor Attacks on GTSRB

Source Class	Target Class	Backdoored	Clean
stop	speedlimit	0.9678/0.9259	0.9732/0.0000
warning	stop	0.9726/0.9889	0.9753/0.0205
speedlimit	warning	0.9683/0.9641	0.9702/0.0000

the same strategy that we followed for the MNIST digit recognition attack, i.e., by poisoning the training dataset via steganography and changing corresponding ground truth labels. As a control group, we do not poison the training dataset, just change corresponding ground truth labels. Table 2 reports these three instances according to the Functionality (left) and the Attack Success Rate (right) metrics. As a control group, we poison the selected images drawn from the original training set without embedding anything, just change their labels to the target label. The result shows in the last column of Table 2, which indicates that the small perturbation caused by steganography can be identified by the DNNs while not for humans. We can use this type of irregular trigger patterns to perform the backdoor attacks.

Comparison with BadNets. BadNets use two types of triggers, the single pixel attack and the pattern attack, to infect the clean images. A list of triggers used by BadNets can be found in Table 3. Note that the single pixel attack of BadNets creates the most difficult task for a DNN to identify the difference between the clean and the poisoned samples. In BadNets, they also admit that they had to change the training parameters, including the step size and the mini-batch size to get the training error to converge. We conduct our experiment on the MNIST dataset, to corroborate the result that the single pixel attack of BadNets needs the most epochs to converge. We do note that for the single target attack, the pollution rates are computed on one class, which differs from universal attacks which are computed from the entire training set.

Compared with the single pixel backdoor attack of BadNets, it is easier for our steganography based attack to achieve convergence of the training error. For the single pixel attack of BadNets, it requires 80 epochs and a 0.5 pollution rate to achieve an acceptable attack success rate. While for our steganography approach, it is only 20 epochs and a 0.1 pollution rate. Besides, we can control the effectiveness of the backdoor attack by increasing the size of the text trigger (as more trigger bits will be encoded). When the size of the text trigger increases, even less epochs and a lower pollution rate are needed to achieve training error convergence. When compared with the pattern backdoor of BadNets, our steganography attack achieves a higher PASS score and a lower LPIPS score, and is thus more invisible. Table 3 and Table 4 show that our attacks achieve a commendable performance in comparison to BadNets, with a less perceivable mask and a comparable attack success rate.

Pollution Rate. We further investigate the impact of the pollution rate ρ upon the performance of the single target backdoor attack. In the single target attack, the pollution rate ρ is the number of samples drawn from the single source class. Those samples will be poisoned by steganography. The impact of the pollution rate on the Attack Success Rate and Functionality on CIFAR-10 and GTSRB datasets is shown as Fig. 9. In Fig. 9, with an increasing pollution rate the Attack Success Rate increases (the blue line in Fig. 9), while the Functionality remains stable. Besides, for different datasets, the minimal pollution rate to achieve high Attack Success Rate is different. For the CIFAR-10 dataset, the minimal pollution rate is 0.04, leading to an Attack Success Rate of 96.6%, while for the GTSRB dataset is 0.22, leading to an Attack Success Rate of 95.11%.

(a) CIFAR-10 (b) GTSRB

Fig. 9: The relationship of Attack Success Rate and Functionality with the pollution rate ρ increase in terms of the single target attack on CIFAR-10 (left) and GTSRB (right).

Invisibility Metrics. When comparing the Invisibility metric with previous BadNets backdoor attacks, we first compute the PASS and LPIPS indices of the two types of triggers used in the previous BadNets backdoor attack (Single Pixel and Pattern Pixel). The range of the PASS score is $(0, 1]$; if two images are identical, the value is 1. A larger PASS value indicates that an image will appear more similar to human perception. Recall that the LPIPS score measures the perceptual distance between the reference image and the blurred image. This LPIPS score lies between $[0, 1)$; if two images are identical, the value is 0. A lower LPIPS value means two images are more similar; a higher score means the images are more different. A comparison of the PASS and LPIPS scores for each attack is found in Table 4. Our trigger achieves the highest average PASS score (extremely close to 1) and lowest average LPIPS score (near 0), better than the triggers of BadNets. This indicates that humans will have more difficulty in discerning differences between our trigger and the original image.

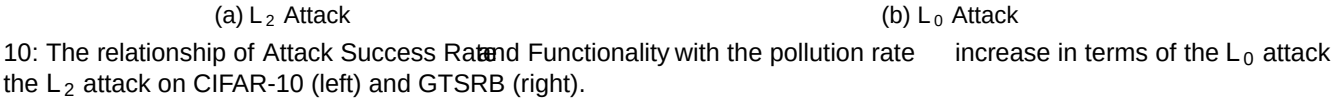
5.2 Universal Backdoor Attacks via Regularization

Setup. We implement the attacks introduced in Section 4.2.2. For the three types of trigger optimizations through L_2, L_0 and L_1 regularization, we mount our attacks on the CIFAR-10/100 and GTSRB [42] datasets. We use the pre-trained ResNet-18 [44] model as the basis of our attacks. The CIFAR-10 dataset [35] consists of 60,000 32x32 colour images in 10 classes, with 6,000 images for each class; so there are 50,000 training images and 10,000 test images. CIFAR-100 [35] is just like the CIFAR-10, except it has 100 classes and 10 times fewer images. GTSRB was introduced previously (see Section 5.1). We achieve 92.48%, 73.44%, and 95.31% prediction accuracy on the respective validation dataset.

Performance. We measure the performance of three types of attackers (L_2, L_0 and L_1) by computing the Attack Success Rate and the Functionality on our three datasets. For the L_2-attack, the results on the CIFAR-10/100 and GTSRB datasets can be seen in Fig. 11. For the CIFAR-10 dataset, we find that extremely small perturbations (L_2-norm < 5), difficult

TABLE 3: Performance in comparison to BadNets

Attacks	Trigger	Trigger size	Source / Target Label	Pollution Rate	Clean Model Accuracy (%)	Epoch	Performance (%)	
							Functionality(%)	Attack Success Rate(%)
BadNets	Single Pixel	L_0=1	4/7	0.5	99.5	80	99.11	99.49
BadNets	Pattern Trigger	L_0=4	4/7	0.1	99.5	5	99.22	99.19
Steganography	Text:"Apple..."	500	4/7	0.1	99.5	20	99.01	98.17



(a) L_2 Attack (b) L_0 Attack

Fig. 10: The relationship of Attack Success Rate and Functionality with the pollution rate and the L_2 attack on CIFAR-10 (left) and GTSRB (right).

Fig. 11: Functionality and Attack Success Rates of L_2 attack on CIFAR-10/100, GTSRB datasets (Validation accuracy: 92.48%, 73.44% and 95.31%, respectively).

Fig. 12: Functionality and Attack Success Rates of L_0 attack on CIFAR-10/100, GTSRB datasets.

for humans to perceive, can still produce satisfactory performance in terms of both the Functionality and Attack Success Rate of the model. The Attack Success Rate for all L_2-norm tests are greater than 90%. For the CIFAR-100 dataset, from Fig. 11 we observe that the Attack Success Rates of all the L_2 attacks exceed 90%. For the GTSRB dataset, we see larger L_2-norms are needed on GTSRB to obtain an equivalent Attack Success Rate comparable to CIFAR. For instance, only when the L_2-norm of the trigger exceeds 9 does the Attack Success Rate exceed 80%. With respect to Functionality, all configurations retain a validation accuracy comparable to the baseline model, which is 92.48%, 73.44%, and 95.31%, respectively.

For the L_0-attack, the results can be seen in Fig. 12. For the CIFAR-100 dataset, with an increase of the L_0-norm, the Attack Success Rate can be raised to 100%. When we retrain the poisoning data on the pre-trained model, we find the model converges faster than the L_2-attack to achieve a high Attack Success Rate. In only a few epochs, the Attack Success Rate exceeds 90%, while for L_2-norm regularization, it needs more than 10 epochs to converge. This demonstrates that it is easier for deep neural networks to memorize the triggers generated by L_0-norm regularization than L_2-norm regularization. It is interesting to see that for the L_0-attack, all datasets achieve a higher Attack Success Rate. This proves that for those triggers with regular shapes, it is easier for DNNs to identify strong correlated signals, which is also true for human inspectors. With respect to the Functionality, all datasets experience a slight drop in the validation accuracy of clean images, but it is acceptable. For example,

Fig. 13: Functionality and Attack Success Rates of L_1 attack on CIFAR-10/100, GTSRB datasets.

TABLE 4: PASS and LPIPS scores compared to BadNets

	Original	Single Pixel	Pattern Pixel	Our trigger size 500
MNIST				
Avg. PASS	1	0.9890	0.9610	0.9994
Avg. LPIPS	0.0	0.0068	0.0247	2.7e-05
CIFAR-10				
Avg. PASS	1	0.9916	0.9714	0.9997
Avg. LPIPS	0.0	0.0079	0.0238	1.4e-4
GTSRB				
Avg. PASS	1	0.9909	0.9695	0.9997
Avg. LPIPS	0.0	0.0063	0.0214	1.2e-4

TABLE 5: PASS and LPIPS scores compared to the Trojaning attack

	Original	Trojan	$L_2 = 10$	$L_0 = 5$	$L_1 = 0.1$
CIFAR-10					
Avg. PASS	1	0.9610	0.9980	0.9908	0.9900
Avg. LPIPS	0.0	0.0414	0.0156	0.0313	0.0295
CIFAR-100					
Avg. PASS	1	0.9543	0.9972	0.9897	0.9685
Avg. LPIPS	0.0	0.0403	0.0134	0.0259	0.0228
GTSRB					
Avg. PASS	1	0.8920	0.9911	0.9614	0.8904
Avg. LPIPS	0.0	0.0321	0.0062	0.0257	0.0286

for CIFAR-100 dataset, the baseline accuracy for CIFAR-100 is 73.44%. When compared to the worst configuration (L_0-norm = 1), the Functionality only drops 0.58%.

For L_1-attack, we use binary search to find the minimal L_1-norm to achieve over 90% Attack Success Rate. We found that when the L_1-norm is less than 0.6, the optimization for generating the L_1 trigger will get stuck in a sub-optimal oscillation. Thus, we demonstrate L_1 attacks with an L_1-norm beyond 0.6 as shown in Fig. 13. We also note that L_1 only penalizes the largest entry of the trigger, resulting in a net change to the input image larger than the L_2-attack. This large modification (trigger) between the poisoned images and the input images creates more significant effects on the DNN model than the L_2 attack, while more invisible for human eyes than the L_0 attack.

Comparison with Trojaning Attack. Table 6 shows that our attacks achieve fairly commendable performance, with a less obvious mask and a comparable attack success rate when compared to the Trojaning attack. We further check whether the activation at the neuron position will be affected when applied to various test images. To verify this, we add our L_p trigger into 10000 test images. Then by observing the average neuron activation of the three backdoored models on the selected position. We can see from Table 7 that the average activation at the neuron position is indeed scaled well by our L_p-norm trigger, indicating that the attack success is rooted by the scaled neuron activation rather than the trigger itself. This stark difference enables better optimization in teaching the retrained model to recognize these prominent differences as features and use them in predictions. It also makes the attack successful without scaling the neuron activation.

Pollution Rate. The poisoned samples for the universal attack are drawn from all of the training set, while for the single target attack the poisoned images are only drawn from one source class. In this configuration, we choose the L_0 attack (L_0-norm = 5), the L_2 attack (L_2-norm = 1) and the L_1 attack (L_1-norm = 0.1) to demonstrate the relationship between the performance of the universal attack and the pollution rate. As seen from Fig. 10 and Fig. 14, with an increasing pollution rate the Attack Success Rate increases (the blue line in Fig. 10 and Fig. 14), while the Functionality remains stable. Additionally, for different datasets, the minimal pollution rate required to achieve a high Attack Success Rate differs.

Invisibility Metrics. Recall that the invisibility metrics are PASS and LPIPS. These metrics quantify how similar two images appear to a human; the range of the PASS metric is $(0, 1]$ and for the LPIPS index is $[0, 1)$; if two images are identical, the PASS value is 1 and the LPIPS value is 0. We compute and compare the PASS and LPIPS scores between the original image and the poisoning images with triggers generated by Trojaning, L_2, L_0, and L_1-attacks. The invisibility metrics are shown in Table 5. Both our triggers achieve a higher average PASS score and a lower average

TABLE 6: Performance in comparison to Trojaning attack

Attacks	Target Layer	Initial Activation	Final Activation	L_2-norm	Poison Rate	Functionality (%)	Attack Success Rate (%)
Trojan	penultimate	0.3251	0.5425	23.652	0.05	91.51	100
L_2 attack	penultimate	1.0908	3.4893	20.000	0.05	91.02	96.44
L_1 attack	penultimate	0.0132	0.2189	6.059	0.05	91.18	99.50
L_0 attack	penultimate	0.3011	1.0912	24.025	0.05	91.17	99.77

TABLE 7: Average activation at the selected neuron position

Attacks	Neuron Position	Trigger Size	Avg. Clean	Avg. Attack
L_2-attack	6	L_2-norm=20	0.2188	0.6288
L_0-attack	6	L_0-norm=25	0.3078	0.9911
L_1 -attack	6	L_1 -norm=0.1	0.1975	0.7163

(a) CIFAR-10 (b) GTSRB

Fig. 14: The relationship of Attack Success Rate and Functionality with the pollution rate increase in terms of the L_1 attack on CIFAR-10 (left) and GTSRB (right).

LPIPS score than the Trojaning triggers. Our PASS scores and LPIPS scores are extremely close to 1 and 0, respectively. This indicates that humans have more difficulty in discerning differences between the original image and our triggers than the original image and the Trojaning triggers. The other interesting observation is that the PASS index, a measure of the structural similarity, for the L_0-attack is larger than that for the L_1 -attack. While the LPIPS index, which not only considers structural similarity but other factors which may affect the perceptual similarity for human, for the L_1 -attack is lower than that for the L_0-attack.

6 Evading Neural Cleanse Detection

To evaluate the effectiveness of our invisible backdoor attacks to evade the state-of-the-art trojan backdoor detection approaches, we have tested our attacks on Neural Cleanse [43]. Neural Cleanse operates based on the observation that the minimum perturbation needed to transform inputs of all other source classes to a target class is bounded by the size of the real trigger. The main part of the Neural Cleanse is an optimization framework that generates the minimal perturbation (potential triggers) to misclassify all images drawn from all of other source classes into the target class. The object function shown as Eq. (11)

$$\min_{m,\Delta} l(y_t, f(A(x, m, \Delta))) + \|m\|_1, \quad (11)$$

where m is the mask which controls the size of the trigger, Δ is the value of the generated trigger, y_t is the potential target label, and $A(\cdot)$ is the operation adding the trigger into input image. Here the authors of Neural Cleanse use the L_1-norm to constrain the magnitude of the trigger size.

TABLE 8: Evaluation the Universal Attacks on Evading Against Neural Cleanse

Settings	Original NC	L_2-attack	L_0-attack	L_1 -attack
Platform	Keras 2.2	Pytorch 1.4	Pytorch 1.4	Pytorch 1.4
Network	6CNN+2FC	6CNN+2FC	6CNN+2FC	6CNN+2FC
Dataset	GTSRB	GTSRB	GTSRB	GTSRB
Target Class	[28]	[28]	[28]	[28]
Trigger	4*4 square	L_2=1	L_0=16	L_1 =0.1
Inject Rate	0.1	0.1	0.1	0.1
Performance	0.9678/0.9850	0.9445/0.9192	0.9684/0.9555	0.9668/0.9857
Detection Result	{28, 12, 2}	{28}	{0, 28}	{11, 28}

The original open sourced Neural Cleanse is implemented through Keras on Tensorflow 1.10. We reproduce Neural Cleanse with Pytorch 1.4. In Neural Cleanse, there are three steps detecting the backdoor attack. First, we need to conduct a backdoor attack to generate a backdoored model to be detected. Note that in their repository, Neural Cleanse detects a BadNets backdoored model. For comparison, we modify the code to support our four types backdoor attacks (steganography-based, L_2, L_0, and L_1 regularization-based) with the same network architecture and the same dataset the Neural Cleanse repository provides. The parameters and details can be found in Table 9, where the first value "Performance" is the Functionality and the second is the Attack Success Rate

For evaluating our attacks on Neural Cleanse, we first we test our three universal backdoor attacks based on $L_p (p = 0; 2; 1)$ regularization. Recall that the universal attack means that there is no restriction to the source input image's class, and any input image with the generated trigger can incur the target label. We report the experimental results in Table 8 to evaluate our three regularization based attacks in a universal way against Neural Cleanse. We observe Neural Cleanse is able to detect our universal attacks. We believe this is a result of Neural Cleanse being able to detect the shortcut of the decision boundary of the backdoored model. For each target class, if there is a minimal perturbation which can transfer all other classes to the target class and the L_1-norm of this minimal perturbation is significantly smaller than the minimal perturbations according to other classes, then this class is the target class. An intuitive explanation is that the L_1-distance of the shortcut is significantly smaller than other transfer distances, resulting in the anomaly detection algorithm (MAD in Neural Cleanse) to find this shortcut.

However, Zhen Xiang et al. [45] note that a critical limitation of Neural Cleanse is the assumption that the injected backdoor comes from all of other classes. Thus, if the attack only uses a source-target pair to inject the backdoor, such as the single target attack we performed in Section 4.1 (Steganography), Neural Cleanse will fail to identify the target label from the remaining classes. We further extend our three universal attacks mentioned above to the single target attack. We have experimentally demonstrated this as observed in the Steganography and the extended three

TABLE 9: Evaluation the Single Target Attacks on Evading Against Neural Cleanse

Settings	Steganography	L_2-attack	L_0-attack	L_1 -attack
Platform	Pytorch 1.4	Pytorch 1.4	Pytorch 1.4	Pytorch 1.4
Network	6CNN+2FC	6CNN+2FC	6CNN+2FC	6CNN+2FC
Dataset	GTSRB	GTSRB	GTSRB	GTSRB
Source Class	4	4	4	4
Target Class	7	7	7	7
Trigger	"Apple"*120	L_2=1	L_0=16	L_1 =0.1
Inject Rate (Number)	1.0(1980)	1.0(1980)	0.5(990)	0.5(990)
Performance	0.9492/0.9712	0.9550/0.9379	0.9536/0.9742	0. 9587/0.9545
Detection Result	f 1,2,0g	[0, 1, 2g	f 0, 1g	f 0, 1, 2g

TABLE 10: Evaluating Against Neural Cleanse for Injecting backdoors for All Classes

Settings	Steganography	L_2-attack	L_0-attack	L_1 -attack
Platform	Pytorch 1.4	Pytorch 1.4	Pytorch 1.4	Pytorch 1.4
Network	6CNN+2FC	6CNN+2FC	6CNN+2FC	6CNN+2FC
Dataset	GTSRB	GTSRB	GTSRB	GTSRB
Target classes	[1-43]	[1-43]	[1-43]	[1-43]
Trigger Number	43	43	43	43
Inject Rate (Each Class)	0.05	0.01	0.01	0.01
Average Performance	0.9235/0.9096	0.9413/0.9980	0.9684/0.9555	0.9668/0.9857
Max Anomaly Index	1.3742	1.7952	1.2239	1.5652
Detection Result	fg	fg	fg	fg

regularization based attacks in Table 9. As Table 9 shows that in our single target attack, the real target is 7 while the Neural Cleanse assumes the target is $f 0; 1; 2g$, so the Neural Cleanse fails to detect the single (source, target) pair attacks.

Following the line of the single (source, target) pair attack, we proposed other methods to evade the detection of Neural Cleanse, which is termed "Injection All". We make transfer distances from all other classes to each small target class, and make there is no signi cant difference between transfer distance for each class. As without a clean reference model (a reasonable assumption because the defender only has the provided DNN model in hand [4, 46]), the anomaly detection of the Neural Cleanse (MAD) will fail. So we inject the backdoor with our invisible backdoor attack for every class using different triggers. We report the experimental results in Table 10. The experimental results show that our invisible backdoor attacks can evade the detection of Neural Cleanse. We can see that after injecting a backdoor for every class, the maximal anomaly index for each potential target class is below 2. In Neural Cleanse, if the maximal anomaly index of one class is over 2, the algorithm will mark this as the target class with a signi cant con dence. However, as we create trigger shortcuts for each class, their anomaly detection is futile. Additionally, the Functionality of the backdoored model is not substantially affected and only decreases slightly.

According to the optimization framework of Neural Cleanse (Eq. (11)), the authors assume that the trigger pattern has a smaller size than the image at the image periphery. In contrast, our steganography-based and L_2, L_1 attacks scatter the trigger around the entire image. Therefore, the generated trigger with the optimization framework shown in Eq. (11) greatly differs from our trigger patterns mentioned above. The recovered triggers by Neural Cleanse in three different attack settings, the universal attack, the single (source, target) pair attack, and injection all attack can be found in Table 11. We acknowledge that Neural Cleanse can still recover our L_0 trigger only in the universal setting, which is the primary limitation of our L_0 attack.

TABLE 11: The Recover Trigger of Neural Cleanse on Our Invisible Backdoor Attacks by the Single Target Way

	Steganography	$L_2 = 1$	$L_0 = 16$	$L_1 = 0:1$
Real Used				
Recovered (Universal)				
Recovered (Single Pair)				
Recovered (Injection All)				

7 Discussion on Other Detection Methods

In this section, we discuss the effectiveness of our backdoor attacks against other backdoor detection approaches. The state-of-the-art detection techniques can be categorized into three types: before/during-training, run-time, and post-training.

Before/During-Training. In this scenario, the defender can access the training set (both the poisoning and clean training sets). Before training a DNN model, the defender rst checks the training set to identify suspicious training samples, and subsequently removing them. The number of samples in a training set is usually enormous, so prior works [47, 48] leverage statistical analysis of the poisoned training set to detect whether the training set has been poisoned by a trigger or not. For human inspection before training, we argue that it is challenging to simply perceive the anomaly on our poisoning images [13] and it is laborious for humans to closely examine such an enormous dataset.

Run-Time. The backdoor detection can also work on the classi er during run-time. In STRIP [49], the authors rely on the strength characteristic of the backdoor attack, which is "image-agnostic". So the perturbation are added on the inputs to be tested. For trojaned inputs, the predictions of the network is invariant because of the "image-agnostic" property, while for clean inputs, when the perturbation is added, the predictions of the network vary dramatically. We highlight that STRIP is ineffective for defending against our attacks because our irregular triggers are generated with a small perturbation. Therefore, when STRIP adds a perturbation to our poisoned inputs, the predictions of the network will also vary greatly due to the fact that our triggers can be broken by adding perturbations.

Post-Training. The third backdoor detection approach is post-training, where the defender can access the trained models and the partial initial clean training set but with no access to the potential poisoned training set. Neural Cleanse [3] and ABS [46] are two state-of-the-art model-based white-box defenses. We have discussed Neural Cleanse at length in Section 6. On the other hand, the white-box detection method of ABS [46], scans every neuron of the

given DNN model, and directly alters the level of stimulation at various neurons; by monitoring the activations of output classes, any neuron which produces a significantly higher output irrespective of the input is an indicator that the resulting model is potentially backdoored. Additionally, retraining Trojaned models [50] incurs high computational costs. As prior work has demonstrated [3, 12, 46] that neural pruning [51] causes the accuracy on unpoisioned data to rapidly plunge when pruning redundant neurons. As for unlearning, there is a condition that the exact trigger must be known [3, 12]. In unlearning, they use the reversed trigger to retrain the infected DNN to recognize correct labels even when the trigger is present. As we have shown in Table 11, this method cannot recover the real triggers used by our three types of invisible backdoor attacks, and thus these three invisible backdoor attacks are robust to unlearning defenses.

8 Conclusion

Our work has found that neural networks are sensitive to features imperceptible to humans. We have exploited these features and designed two novel types of backdoor attacks. Because our attack triggers are derived from these covert features, in comparison to the state-of-the-art backdoor attacks, our attack is more covert and overcomes the practicality issues of existing attacks. Because our trigger patterns are invisible to human eyes, non-detection by humans will increase the success probability of backdoor attacks in practice by making the input images inconspicuous. Additionally, we have argued that our triggers can evade the state-of-the-art backdoor detection algorithms, as it is hard to recover the invisible trigger through the optimization algorithm.

In our future work, we seek to provide a deeper explanation from the internal structure of the neural network to ascertain the reason why the backdoor attack succeeds. In addition, adversarial training [52] and differential privacy [53, 54] are proven effective to create a robust machine learning model [55]. We will also explore the feasibility of backdoor attacks against robust models. Understanding the robustness of both attacks and defenses provides an avenue to demystify deep neural networks, rendering deep learning models to be more transparent.

Acknowledgements

We would like to thank Jiahao Yu and Damith Ranasinghe for useful discussion and would like to thank Dali Kaafar for proofreading the abstract and introduction of the early version of this work. This work was supported, in part, by the National Natural Science Foundation of China, under Grants No. 61972453, No. 61672350, No. U1936214, and No. U1636206.

References

[1] Andras Rozsa, Ethan M Rudd, and Terrance E Boult. Adversarial diversity and hard positive generation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 25–32, 2016.

[2] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 586–595, 2018.

[3] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. IEEE Security and Privacy, 2019.

[4] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring Trojan backdoors in AI systems. arXiv preprint arXiv:1908.01763, 2019.

[5] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3642–3649, 2012.

[6] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 29(6):82–97, 2012.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529, 2015.

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

[9] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In IEEE Symposium on Security and Privacy, pages 3–18. IEEE, 2017.

[10] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. NIPS Workshop on Machine Learning and Computer Security, 2017.

[11] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. The Network and Distributed System Security Symposium (NDSS), 2017.

[12] Shawn Shan, Emily Willson, Bolun Wang, Bo Li, Haitao Zheng, and Ben Y Zhao. Gotta catch'em all: Using concealed trapdoors to detect adversarial attacks on neural networks. arXiv preprint arXiv:1904.08554, 2019.

[13] Cong Liao, Haoti Zhong, Anna Cinzia Squicciarini, Sencun Zhu, and David J. Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. CoRR, abs/1808.10307, 2018.

[14] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35 (8):1798–1828, 2013.

[15] Shailender Gupta, Ankur Goyal, and Bharat Bhushan. Information hiding using least significant bit steganography and cryptography. International Journal of Modern Education and Computer Science, 4(6):27, 2012.

[16] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[17] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, pages 1885–1894, 2017.

[18] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1452–1458, 2016.

[19] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1689–1698, 2015.

[20] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 321–338, 2019.

[21] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 2154–2156, 2018.

[22] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, pages 47230–47244, 2019.

[23] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 86–94, 2017.

[24] Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S Davis, and Tom Goldstein. Universal adversarial training. *arXiv preprint arXiv:1811.11304*, 2018.

[25] Niels Provos. Defending against statistical steganalysis. In *10th USENIX Security Symposium, August 13-17, 2001, Washington, D.C., USA*, 2001.

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

[27] Masoud Nosrati, Ronak Karimi, and Mehdi Hariri. Audio steganography: a survey on recent approaches. *world applied programming*, 2(3):202–205, 2012.

[28] R Balaji and Garewal Naveen. Secure data transmission using video steganography. In *2011 IEEE International Conference on Electro/Information Technology*, pages 1–5. IEEE, 2011.

[29] Songtao Wu, Sheng-hua Zhong, and Yan Liu. Steganalysis via deep residual network. In *22nd IEEE International Conference on Parallel and Distributed Systems, ICPADS 2016, Wuhan, China, December 13-16, 2016*, pages 1233–1236, 2016.

[30] Gurpreet Kaur and Kamaljeet Kaur. Image watermarking using lsb (least significant bit). *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(4), 2013.

[31] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 86–94, 2017.

[32] Konda Reddy Mopuri, Aditya Ganeshan, and R. Venkatesh Babu. Generalizable data-free objective for crafting universal adversarial perturbations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(10):2452–2465, 2019.

[33] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *CoRR*, abs/1712.09665, 2017.

[34] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Yiran Chen, and Hai Li. DPATCH: an adversarial patch attack on object detectors. In *Workshop on Artificial Intelligence Safety 2019 co-located with the Thirty-Third AAAI Conference on Artificial Intelligence 2019 (AAAI-19), Honolulu, Hawaii, January 27, 2019.*, 2019.

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

[36] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017.

[37] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *CoRR*, abs/1905.02175, 2019.

[38] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[39] David Warde-Farley and Ian Goodfellow. 11 adversarial perturbations of deep neural networks. *Perturbations, Optimization, and Statistics*, 311, 2016.

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

[41] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013*, pages 1–8, 2013.

[42] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.

[43] Yuchen Zhang, Percy Liang, and Martin J. Wainwright. Convexified convolutional neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 4044–4053, 2017.

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

[45] Zhen Xiang, David J. Miller, and George Kesidis. Revealing backdoors, post-training, in DNN classifiers via novel inference on optimized perturbations inducing group misclassification. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.

[46] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1265–1282, 2019.

[47] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3517–3529, 2017.

[48] Di Tang, Xiaofeng Wang, Haixu Tang, and Kehuan Zhang. Demon in the variant: Statistical analysis of dnns for robust backdoor contamination detection. *CoRR*, abs/1908.00686, 2019.

[49] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith Chinthana Ranasinghe, and Surya Nepal. STRIP: A defence against trojan attacks on deep neural networks. *CoRR*, abs/1902.06531, 2019.

[50] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural Trojans. In *2017 IEEE International Conference on Computer*