2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA) | 978-1-6654-6283-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICMLA55696.2022.00172

# Deep Neural Network Piration without Accuracy Loss

Aritra Ray Jinyuan Jia Sohini Saha Jayeeta Chaudhuri Neil Zhenqiang Gong Krishnendu Chakrabarty
Duke University

***Abstract*—A deep neural network (DNN) classifier is often viewed as the intellectual property of a model owner due to the huge resources required to train it. To protect intellectual property, the model owner can embed a watermark into the DNN classifier (called *target classifier*) such that it outputs pre-determined labels (called *trigger labels*) for pre-determined inputs (called *trigger inputs*). Given the black-box access to a suspect classifier, the model owner can verify whether the suspect classifier is pirated version of its classifier by first querying the suspect classifier for trigger inputs and then checking whether the predicted labels match with the trigger labels. Many studies showed that an attacker can pirate the target classifier (called *pirated classifier*) via retraining or fine-tuning the target classifier to remove its watermark. However, they sacrifice the accuracy of the pirated classifier, which is undesired for critical applications such as finance and healthcare. In our work, we propose a new attack without sacrificing the accuracy of the pirated classifier for in-distribution testing inputs while preventing the detection from the model owner. Our idea is that an attacker can detect the trigger inputs in the inference stage of the pirated classifier. In particular, given a testing input, we let the pirated classifier return a random label if the input is detected as a trigger input. Otherwise, the pirated classifier predicts the same label as the target classifier. We evaluate our attack on benchmark datasets and find that our attack can effectively identify the trigger inputs. Our attack reveals that the intellectual property of a model owner can be violated with existing watermarking techniques, highlighting the need for new techniques.**

***Index Terms*—Deep Neural Networks, Intellectual Property, Watermarking**

## I. Introduction

Deep neural networks (DNNs) [1], [2] are increasingly used in many real-world applications such as image classification due to their superior performance. However, it often requires a lot of resources (e.g., resources used to collect a high-quality training dataset, computation resources) to train a high-quality deep neural network classifier. As a result, a DNN classifier often represents the intellectual property of a model owner. To protect intellectual property, the model owner can embed a watermark into it. Roughly speaking, there are two categories of watermarking methods: *parameter-based watermarking* [3], [4] and *label-based watermarking* [5]–[7]. In parameter-based watermarking, the model owner embeds a watermark into the parameters of a DNN classifier, e.g., let the signs of certain parameters have a specific pattern. However, the parameter-based watermarking requires the model owner to have white-box access to a suspect classifier to verify whether it is a pirated version of its DNN classifier, which is not applicable when the model owner only has black-box access to the suspect classifier. Therefore, we focus on label-based watermarking in this work.

In label-based watermarking, a model owner embeds a watermark into a DNN classifier (called *target classifier*) such that it predicts a pre-determined label (called *trigger label*) for pre-determined inputs (called *trigger inputs*), where both the trigger label and trigger inputs can be selected by the model owner. For instance, Zhang et al. proposed a backdoor-based watermarking [7]. In particular, the model owner can randomly pick a certain number of training inputs from the training dataset of the target classifier and embed a pattern (chosen by the model owner) into each of them to construct trigger inputs. Then, the model owner can arbitrarily select a label (i.e., trigger label) and assign it to those trigger inputs. Then, the attacker can use those labeled trigger inputs along with the training dataset to train a target classifier. It is very likely that the target classifier will predict the trigger label for the trigger inputs. Suppose the model owner has black-box access to a suspect classifier and aims to verify whether it is pirated version of the target classifier. The model owner can query the suspect classifier for the trigger inputs and check whether the predicted labels are the same as the trigger labels. The model owner can view the suspect classifier as a pirated version of the target classifier if predicted labels for most of the trigger inputs match with the trigger label.

Many studies [8]–[10] showed that an attacker can remove the watermark in a target classifier to pirate it. Roughly speaking, those methods pirate the target classifier (called *pirated classifier*) via re-training or fine-tuning the target classifier. However, those methods sacrifice the accuracy of the pirated classifier, i.e., the pirated classifier has lower accuracy on testing inputs (called *in-distribution inputs*) that have the same distribution as the training dataset compared with the target classifier. In some critical applications such as finance and healthcare, even 1% accuracy loss may be intolerable.

**Our work:** In our work, we propose a new attack that does not sacrifice the accuracy of the pirated classifier on in-distribution inputs while preventing the model owner from detecting it. Suppose the attacker has white-box access to a target classifier, the attacker can deploy the target classifier, i.e., the pirated classifier is the same as the target classifier, as a cloud service (e.g., AWS, Google Cloud, IBM Cloud) or an end-user product (e.g., a mobile app, software). Therefore, the model owner

978-1-6654-6283-9/22/$31.00 ©2022 IEEE
DOI 10.1109/ICMLA55696.2022.00172

can have black-box access to it. To avoid being detected by the model owner, our intuition is that the trigger inputs have different distributions from in-distribution inputs, which have the same distribution as the training dataset. Therefore, we can train a detector to distinguish in-distribution inputs from the trigger inputs. In particular, if a testing input is not detected as in-distribution, then the pirated classifier returns a random label. Otherwise, the pirated classifier returns the same label as the target classifier for the testing input.

To train a detector, we assume the attacker has a small number of labeled inputs (called *surrogate dataset*) that have the same distribution as the training dataset. There are two challenges in training a high-quality detector: 1) the attacker only has a small number of labeled inputs which makes it hard to train a high-quality detector, and 2) in-distribution inputs with different ground truth labels have different characteristics, which makes it challenging to distinguish in-distribution inputs from trigger inputs. To address the first challenge, we propose to train a detector to distinguish heatmaps of in-distribution inputs and trigger inputs. Roughly speaking, given an input and a classifier, a heatmap for the input highlights the features in the input that have significant contributions to the final predicted label of the classifier for the input. Thus, the heatmap is sparser than the input, which makes it easier to distinguish. We use Generative Adversarial Networks (GANs) [11] to learn the distribution of heatmaps of in-distribution inputs. As the discriminator of GAN is used to predict whether a heatmap is from a real input, we use it to detect whether a testing input is in-distribution or not based on its heatmap. To address the second challenge, for each label, we train a GAN to learn the distribution of heatmaps for the inputs that have the same label. Therefore, each GAN only needs to learn the distribution for in-distribution inputs with the same ground truth label. Given a testing input, we use the pirated classifier to predict a label for it as well as compute a heatmap for it. Then, we use the discriminator of the GAN trained for the predicted label to infer whether the given testing input is in-distribution or not.

We systematically evaluate our attack on MNIST and Fashion-MNIST benchmark datasets. We use *False Positive Rate (FPR)* and *False Negative Rate (FNR)* as evaluation metrics. Roughly speaking, FPR measures the fraction of in-distribution inputs that are predicted as trigger inputs; and FNR measures the fraction of trigger inputs that are predicted as in-distribution. Our experimental results indicate that our attack can achieve small FPR and FNR. For instance, on MNIST dataset, our attack can achieve 0 FPR and FNR for different trigger inputs and trigger labels. Our work reveals that an attacker can pirate a target classifier without sacrificing its accuracy for in-distribution inputs. In other words, existing watermarking techniques are insufficient and thus we need new techniques.

Our contributions are summarized as follows:

- We are the first to show the feasibility of pirating a target classifier without sacrificing its classification accuracy.
- We propose multiple methods (e.g., generating heatmaps) to optimize our attack.
- We perform systematic evaluations for our attack on benchmark datasets.

The rest of the paper is organized as follows. In Section II, we provide the background and discuss the related work. We discuss our threat model in Section III, introduce our proposed attack in Section IV, perform systematic evaluations on benchmark datasets in Section V, and conclude in Section VI.

## II. Background and Related Work

### A. Deep Neural Networks

Deep neural networks (DNNs) are being increasingly used in a variety of applications, such as image classification [1], [2], speech recognition [12], and natural language processing [13]. A typical machine learning pipeline for deep neural networks contains two stages: training stage and inference stage. In the training stage, we use a machine learning algorithm to train a deep neural network classifier on a training dataset. In the inference stage, we use the trained DNN classifier to predict labels for testing inputs. In practice, training a DNN classifier usually requires lots of resources (e.g., resources used to collect a high-quality training dataset, computation resources). Therefore, in practice, a resourceful model owner (e.g., Google, Amazon, Microsoft) trains a DNN classifier and sells/shares it to less resourceful customers. The customers could use the DNN classifier to predict labels for their testing inputs.

### B. Watermarking Deep Neural Networks

The DNN classifier (called *target classifier*) trained by a model owner represents its intellectual property, which could be violated by an attacker. For instance, an attacker can buy a DNN classifier from a model owner and then deploys it as a cloud service to monetize it. To protect the intellectual property of the target classifier, many studies [14] proposed to embed a watermark into it. Given an arbitrary classifier (called *suspect classifier*), the model owner can verify whether the suspect classifier is a pirated version of the target classifier by checking whether the suspect classifier contains the same or similar watermark.

Roughly speaking, those methods can be categorized into *parameter-based watermarking* [3], [4] and *label-based watermarking* [5]–[7], [15], [16]. In parameter-based watermarking, the model owner embeds a watermark into the parameters of a target classifier. For instance, a model owner can add an embedding loss to the loss function used to train the target classifier such that signs of certain parameters of the target classifier have a pre-determined pattern [3]. One major challenge of parameter-based watermarking is that the model owner needs to have a *white-box* access to the suspect classifier, and thus it is not applicable when the black-box access of the suspect classifier is provided. In contrast, label-based watermarking embeds a watermark into a target classifier by letting it predict pre-determined labels for pre-determined inputs. A suspect classifier is viewed as a pirated version of the target classifier if it also predicts the pre-determined

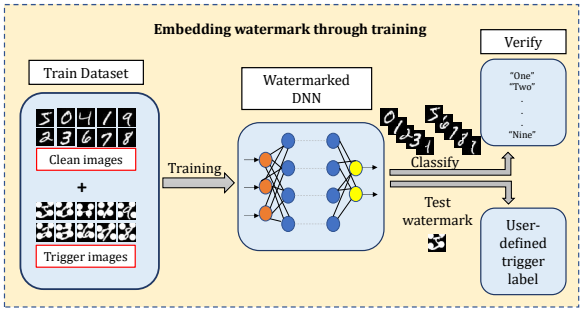


Fig. 1. Embedding watermark while training to protect DNN IP.

labels for pre-determined inputs. As only the predicted labels of the suspect classifier for the pre-determined inputs are needed, the model owner only needs to have black-box access to the suspect classifier. Therefore, we focus on label-based watermarking in this work. There are two phases in label-based watermarking: embedding and verifying. Next, we discuss more details about them:

- **Embedding:** Roughly speaking, label-based watermarking [5], [7] embeds a watermark into a target classifier by letting it predict pre-determined classes for pre-determined inputs. Specifically, the model owner first selects $m$ inputs (called *trigger inputs*), which are denoted as $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$. For instance, the model owner can randomly select some inputs from the training dataset and embed a unique pattern such as white round blocks into them. Then, the model owner can assign a label $y$ (called *trigger label*) to the trigger input $\mathbf{x}_i$, where $i = 1, 2, \cdots, m$. The model owner could use those (trigger input, trigger label) pairs, i.e., $\{(\mathbf{x}_1, y), (\mathbf{x}_2, y), \cdots, (\mathbf{x}_m, y)\}$, along with the training dataset to train a target classifier such that the target classifier predicts the trigger label for those trigger inputs.
- **Verifying:** Given an arbitrary suspect classifier, a model owner queries the suspect classifier for the trigger inputs $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$. Then, the model owner can compute the fraction of inputs that are predicted as the trigger label. The suspect model is viewed as a pirated version of the target classifier if most of the trigger inputs are predicted as the trigger label.

Figure 1 shows a visualization of those two stages.

### C. Watermarking Removal

**Existing watermark removal methods:** Some studies [8]–[10] showed that an attacker can remove the watermark in a target classifier such that a model owner cannot correctly verify whether a suspect classifier is a pirated version of the target classifier. For instance, Chen et al. [10] proposed to remove the watermark in a target classifier via fine-tuning it. Roughly speaking, the idea is that fine-tuning a target classifier without using trigger inputs is very likely to make it forget those trigger inputs based on the catastrophic forgetting phenomenon [17]. Moreover, they showed that an attacker can leverage unlabeled inputs to fine-tune a target classifier with a small number of labeled inputs. Zhang et al. [9] proposed to remove a watermark in a target classifier by training a classifier from scratch using inputs along with the labels predicted by the target classifier. In particular, given either black-box or white-box access to a target classifier, the attacker can query the target classifier using inputs that have a similar distribution to the training inputs of the target classifiers. Then, the attacker can use those inputs along with the predicted labels of the target classifier to train a surrogate model. The surrogate model is expected to only copy the functionality of the target classifier for normal inputs and thus does not predict trigger labels for the trigger inputs.

**Limitations of existing watermark removal methods:** The major limitation of existing watermark removal methods [9], [10] is that they sacrifice the utility of the target classifier. In particular, it is very likely that the retrained or fine-tuned classifier is less accurate compared to the target classifier, especially when the attacker only has a small amount of data. In some critical applications such as finance and healthcare, even 1% of accuracy drop may be intolerable. In this work, we proposed a new attack that aims to address this limitation.

### D. Generative Adversarial Networks

As our attack relies on generative neural networks (GAN) [11], we briefly introduce it in this section. Suppose we have a training dataset whose input $\mathbf{x}$ is sampled from the distribution $p(\mathbf{x})$. GAN aims to train a generative model to generate inputs from the distribution $p(\mathbf{x})$. In particular, there are two components in a generative neural network: a generator (denoted as $G$) and a discriminator (denoted as $D$). Roughly speaking, the generator takes a random vector (e.g., drawn from Gaussian distribution) as input and produces a sample. Given an arbitrary sample, the discriminator $D$ outputs a probability that the sample comes from the training dataset instead of being generated by the generator $G$. The generator $G$ and the discriminator $D$ are trained simultaneously. In particular, the generator $G$ generates a batch of samples. The discriminator $D$ is trained to distinguish between the batch of generated samples and real samples from the training dataset. Given the discriminator $D$, the generator $G$ is trained to generate samples that can fool the discriminator $D$. Formally, they are trained by solving the following optimization problem:

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \log(D(\mathbf{x})) + \mathbb{E}_{\mathbf{e}\sim p_e(\mathbf{e})} \log(1 - D(G(\mathbf{e}))), \tag{1}$$

where $p_e(\mathbf{e})$ is a noise distribution (e.g., Gaussian noise) and $\mathbf{e}$ is a noise vector sampled from the distribution $p_e(\mathbf{e})$. Note that after training a generative adversarial network, we can use the discriminator $D$ to detect inputs that have different distribution from inputs in the training dataset.

## III. Threat Model

There are two parties: *model owner* and *attacker*. The model owner trains a DNN classifier (called *target classifier*).

To protect intellectual property, the model owner embeds a watermark into the target classifier. Then, the model owner sells (or shares) the target classifier to its customers. Next, we introduce our threat model, where we characterize the attacker with respect to its background knowledge and goal.

**Attacker's background knowledge:** We consider the attacker has *white-box* access to a target classifier embedded with a watermark, i.e., the attacker can access all the parameters of the target classifier. However, the attacker cannot access the training dataset, training algorithm, as well as hyperparameters used to train the target classifier. We assume the attacker has a small number of labeled inputs (called *surrogate dataset*) that have the same distribution but do not overlap with the training dataset of the target classifier. We consider the attacker does not know the trigger inputs and trigger label selected by the model owner when embedding a watermark into the target classifier. Moreover, the attacker does not know the watermarking algorithm adopted by the model owner.

**Attacker's goal:** The attacker aims to pirate the target classifier (called *pirated classifier*) and monetize it by deploying it as a cloud service (e.g., Google Cloud Platform, IBM Cloud, Microsoft Azure, AWS) or an end-user product (e.g., software, a mobile app). In other words, the model owner could have black-box access to the pirated classifier. The attacker aims to deploy the pirated classifier to achieve two goals. First, the pirated classifier should have the same classification accuracy as the target classifier for testing inputs (called *in-distribution inputs*) that have the same distribution as the training inputs. We note that even 1% accuracy loss is intolerable in some critical applications such as finance and healthcare. Second, the model owner should not be able to verify whether the pirated classifier is a pirated version of the target classifier. Specifically, the model owner may query the pirated classifier with trigger inputs and check whether they are predicted as the trigger labels. Thus, the attacker aims to detect those trigger inputs and then let pirated classifier make a random prediction for it.

## IV. Our Attack

### A. Overview

Our idea is to directly deploy the target classifier, i.e., the pirated classifier is the same as the target classifier. Therefore, the pirated classifier has the same classification accuracy as the target classifier for in-distribution testing inputs. To reach the second goal, we propose to detect the trigger inputs in the inference stage of the pirated classifier. Our key intuition is that those trigger inputs have a different distribution from the in-distribution inputs. Therefore, we can train a detector to detect them. However, there are two challenges in building the detector: 1) the attacker only has a small amount of labeled data which is insufficient to train a high-quality detector, and 2) the inputs with different ground truth labels have different characteristics which make it more challenging for the detector to distinguish in-distribution inputs from the trigger inputs. To address the first challenge, we propose to compute a *heatmap* for an input and detect it based on its heatmap. To address the second challenge, we proposed to train a detector for each label. Given a testing input, our detector produces an *anomaly score* for it. An input is viewed as a trigger input if its anomaly score is higher than a certain threshold. Moreover, our pirated classifier returns a random label if an input is detected as a trigger input.

### B. Building a Detector

**Generating a heatmap for an input:** In the machine learning community, heatmaps are used to interpret the classification results made by a classifier for an input. In particular, given an input and a classifier, the classifier predicts a label for the input. A heatmap is generated to highlight the contributions made by each feature (e.g., each pixel in an image) in the input to the predicted label. In practice, only a small amount of "important" features have significant contributions. Thus, the heatmap has the same shape as the input but is sparser as many features have small contributions to the predicted label. We can exploit the heatmap of an input to detect whether it is an in-distribution input or not. Our intuition is that the heatmap carries important classification information of the input. For a trigger input, the important features (e.g., a unique pattern selected by the model owner) that lead to its predicted label (i.e., trigger label) are different from those of the in-distribution inputs. Moreover, as the heatmap is sparser, we can train a high-quality detector with a small number of in-distribution inputs to detect whether a heatmap is generated from an in-distribution input.

**Building a Detector via GANs:** Given a training dataset, we can train a generative neural network to learn its distribution. The discriminator in the generative neural network can be used to distinguish whether an input is an in-distribution input or not. The major challenge is that in-distribution inputs with different ground-truth labels have different characteristics, which makes it hard to distinguish in-distribution inputs from trigger inputs, especially when the attacker only has a small surrogate dataset. To address the challenge, we propose to train a GAN for each label. Suppose the total number of labels is $c$. For each label $l$, where $l = 1, 2, \cdots, c$, we construct a training dataset $\mathcal{T}_l$ by finding inputs whose ground truth label is $l$ from the attacker's surrogate dataset. Then, we can compute the heatmap for each input in $\mathcal{T}_l$ and then use those heatmaps to train a GAN, where we use $D_l$ to denote the discriminator. The $c$ discriminators form our detector.

**Detecting trigger inputs:** Given a testing input $\mathbf{x}$, we can use the pirated classifier to predict a label (denoted as $y$) for it. Then, we can compute a heatmap for it. Given the heatmap, we use $D_y$ to compute a loss value for it. The loss value is viewed as the anomaly score for the testing input $\mathbf{x}$. We also compute the loss value for each input in $\mathcal{T}_y$ using $D_y$ and use $\alpha_y$ to denote the largest loss value for inputs in $\mathcal{T}_y$. The input $\mathbf{x}$ is predicted as in-distribution if the anomaly score is no larger than $\alpha_y + \tau$, where $\tau$ is a hyperparameter. In other words, we view the inputs whose anomaly score is larger than

TABLE I
FPR AND FNR OF OUR ATTACK ON MNIST DATASET.

| Trigger label | Target classifier's testing accuracy | Target classifier's trigger accuracy | FPR | FNR |
|---|---|---|---|---|
| Digit 0 | 95.95% | 100.00% | 0.00 | 0.00 |
| Digit 1 | 96.11% | 100.00% | 0.00 | 0.00 |
| Digit 2 | 96.97% | 100.00% | 0.00 | 0.00 |
| Digit 3 | 95.97% | 100.00% | 0.00 | 0.00 |
| Digit 4 | 98.38% | 100.00% | 0.00 | 0.00 |
| Digit 5 | 97.41% | 100.00% | 0.00 | 0.00 |
| Digit 6 | 97.94% | 100.00% | 0.00 | 0.00 |
| Digit 7 | 97.39% | 100.00% | 0.00 | 0.00 |
| Digit 8 | 97.17% | 100.00% | 0.00 | 0.00 |
| Digit 9 | 98.28% | 100.00% | 0.00 | 0.00 |

$\alpha_y + \tau$ as trigger inputs. If an input is viewed as a trigger input, the pirated classifier predicts a random label for it.

## V. EVALUATIONS

### A. Experimental Setup

*1) Datasets:* We evaluate our attack on benchmark datasets, including MNIST [18] and Fashion-MNIST [19]. MNIST consists of grayscale images of handwritten digits, each of which has a size of $28 \times 28$ and belongs to one of ten classes. Fashion-MNIST consists of grayscale images from 10 different fashion products. The size of each image in Fashion-MNIST is $28 \times 28$. There are 60,000 training and 10,000 testing images in both MNIST and Fashion-MNIST.

*2) Training a watermark embedded target classifier:* We use the training data of a dataset to train a target classifier. We adopted the watermarking method proposed by [7] to embed a watermark into the target classifier. In particular, we randomly select 15 inputs from the training dataset and randomly embed four white round blocks in each of them to construct trigger inputs (we study the impact of the number of white round blocks in our experiments). By default, we set the radius of the white round block to be 3. Then, we select a label as the trigger label and assign it to the 15 trigger inputs. Finally, to embed a watermark into a target classifier, we use those labeled trigger inputs and the training data of a dataset to train a target classifier. In particular, we train a ResNet-34 [2] by default (we study the impact of the target classifier architecture in our experiments). Moreover, we train it for 15 epochs with a learning rate of 0.0001 and a batch size of 128 using Adam optimizer [20]. We report the target classifier's accuracy for testing data of a dataset (called *target classifier's testing accuracy*. Moreover, we also compute the fraction of trigger inputs that are predicted as the trigger label by the target classifier (called *target classifier's trigger accuracy*).

*3) Evaluation metrics:* We use *False Positive Rate (FPR)* and *False Negative Rate (FNR)* as evaluation metrics. In particular, FPR measures the fraction of clean testing inputs in the testing dataset (i.e., in-distribution testing inputs) that are predicted as trigger inputs. FNR measures the fraction of trigger inputs that are predicted as in-distribution inputs. A small FRR and FNR mean our detector achieves good performance.

*4) Parameter settings:* We randomly sample 5% testing examples from the testing data of each dataset as the surrogate dataset of an attacker. We use Grad-CAM [21] to generate a heatmap for an input. Moreover, we adopt the public implementation[1] in our experiments. We adopt Deep Convolutional Generative Adversarial Network (DCGAN) [22] as our generative neural networks. We use the publicly available implementation [2]. Unless otherwise mentioned, we set $\tau = 2$ for MNIST and $\tau = 0.65$ for Fashion-MNIST, considering those two datasets have different characteristics.

### B. Experimental Results

**Our attack achieves low FPR and FNR:** Table I and II show the FPR and FNR on MNIST and Fashion-MNIST for different trigger labels. We have the following observations from the experimental results. First, we find that our attack achieves low FPR and FNR on both datasets. For instance, our attack can achieve zero FPR and FNR for different trigger labels. Second, we find that both FPR and FNR on Fashion-MNIST are slightly higher than MNIST dataset. The reason is that heatmaps for MNIST is sparser than Fashion-MNIST. Thus, when the size of the surrogate dataset is the same, it is more challenging to train a good detector for the Fashion-MNIST dataset.

**Impact of the architecture of the target classifier:** Table III shows the impact of the target classifier's architecture on FPR and FNR on MNIST dataset in default setting. The results show that our attacks can achieve similar FPR and FNR. In other words, our attack is effective for target classifiers with different architectures.

**Impact of trigger inputs:** We also study the impact of the trigger inputs for our attack. Recall that we add four white round blocks to inputs that are randomly sampled from the training data. We explore adding a different number of white round blocks to those inputs as trigger inputs. In particular,

[1]https://github.com/jacobgil/pytorch-grad-cam
[2]https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/dcgan/dcgan.py

TABLE II
FPR AND FNR OF OUR ATTACK ON FASHION-MNIST DATASET.

| Trigger label | Target classifier's testing accuracy | Target classifier's trigger accuracy | FPR | FNR |
|---|---|---|---|---|
| T-shirt/top | 99.42% | 100.00% | 0.20 | 0.07 |
| Trouser | 99.26% | 100.00% | 0.00 | 0.00 |
| Pullover | 98.27% | 100.00% | 0.20 | 0.20 |
| Dress | 97.79% | 100.00% | 0.20 | 0.13 |
| Coat | 97.50% | 100.00% | 0.20 | 0.07 |
| Sandal | 99.18% | 100.00% | 0.13 | 0.00 |
| Shirt | 94.36% | 100.00% | 0.00 | 0.00 |
| Sneaker | 96.72% | 100.00% | 0.07 | 0.00 |
| Bag | 97.42% | 100.00% | 0.00 | 0.20 |
| Ankle Boot | 98.67% | 100.00% | 0.00 | 0.00 |

TABLE III
IMPACT OF THE TARGET CLASSIFIER ARCHITECTURE. THE DATASET IS MNIST AND THE TIRGGER LABEL IS "DIGIT 0".

| Target classifier's architecture | Target classifier's testing accuracy | Target classifier's trigger accuracy | FPR | FNR |
|---|---|---|---|---|
| ResNet-101 | 99.17% | 100.00% | 0.00 | 0.00 |
| ResNet-152 | 97.83% | 100.00% | 0.00 | 0.00 |
| ResNet-34 | 95.95% | 100.00% | 0.00 | 0.00 |

TABLE IV
IMPACT OF TRIGGER INPUTS. THE DATASET IS MNIST AND THE TRIGGER LABEL IS "DIGIT 0".

| # white round blocks | Target classifier's testing accuracy | Target classifier's trigger accuracy | FPR | FNR |
|---|---|---|---|---|
| 1 | 97.02% | 100.00% | 0.00 | 0.00 |
| 2 | 96.75% | 100.00% | 0.00 | 0.00 |
| 3 | 96.17% | 100.00% | 0.00 | 0.00 |
| 4 | 95.95% | 100.00% | 0.00 | 0.00 |

TABLE V
IMPACT OF $\tau$ ON FPR AND FNR. THE DATASET IS MNIST AND THE TRIGGER LABEL IS "DIGIT 0".

| $\tau$ | Target classifier's testing accuracy | Target classifier's trigger accuracy | FPR | FNR |
|---|---|---|---|---|
| 0.1 | 95.95% | 100.00% | 100.00 | 0.00 |
| 0.3 | | | 0.93 | 0.00 |
| 0.5 | | | 0.13 | 0.00 |
| 2 | | | 0.00 | 0.00 |
| 4 | | | 0.00 | 0.00 |
| 6 | | | 0.00 | 0.00 |
| 8 | | | 0.00 | 0.00 |
| 10 | | | 0.00 | 0.00 |
| 12 | | | 0.00 | 0.00 |
| 14 | | | 0.00 | 0.00 |
| 16 | | | 0.00 | 0.06 |
| 18 | | | 0.00 | 0.53 |
| 20 | | | 0.00 | 0.86 |

we respectively add 1, 2, 3, and 4 white round blocks to the inputs. Table IV shows our experimental results. We find that our attack can achieve low FRR and FNR, that is, our attack is effective for different trigger inputs.

**Impact of $\tau$:** Table V shows the impact of $\tau$ on FPR and FNR for our attack. We have the following observations. First, as $\tau$ increases, FPR increases while FNR decreases. The reason is that an input is more likely to be predicted as in-distribution when $\tau$ is large. Second, we find that the FPR drops very quickly as $\tau$ increases. Third, our attack can achieve 0 FPR and FNR for a wide range of $\tau$. Our results indicate that our attack is insensitive to $\tau$ for non-extreme values.

## VI. CONCLUSION AND FUTURE WORK

A DNN classifier is often viewed as the intellectual property of a model owner. The model owner can embed a watermark into its DNN classifier to protect its intellectual property. In particular, with black-box access to a suspect classifier, the model owner can detect whether the suspect classifier is a pirated version of its DNN classifier. In this work, we show that an attacker can pirate the model owner's DNN classifier without sacrificing the classification accuracy of the DNN classifier for in-distribution testing inputs while preventing the model owner's detection. Our attack reveals that the model owner's intellectual property may be violated with existing watermarking techniques. Interesting future research direction: 1) generalizing our attack to other domains, e.g., graph and text, and 2) developing new watermarking techniques to mitigate our attack.

## REFERENCES


[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[3] H. Chen, B. D. Rohani, and F. Koushanfar, "Deepmarks: A digital fingerprinting framework for deep neural networks," *arXiv preprint arXiv:1804.03648*, 2018.

[4] Y. Nagai, Y. Uchida, S. Sakazawa, and S. Satoh, "Digital watermarking for deep neural networks," *International Journal of Multimedia Information Retrieval*, vol. 7, no. 1, pp. 3–16, 2018.

[5] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1615–1631.

[6] B. Darvish Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 485–497.

[7] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 159–172.

[8] T. Wang and F. Kerschbaum, "Attacks on digital watermarks for deep neural networks," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 2622–2626.

[9] M. Shafieinejad, N. Lukas, J. Wang, X. Li, and F. Kerschbaum, "On the robustness of backdoor-based watermarking in deep neural networks," in *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, 2021, pp. 177–188.

[10] X. Chen, W. Wang, Y. Ding, C. Bender, R. Jia, B. Li, and D. Song, "Leveraging unlabeled data for watermark removal of deep neural networks," in *ICML workshop on Security and Privacy of Machine Learning*, 2019, pp. 1–6.

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[12] A.-r. Mohamed, G. Dahl, G. Hinton *et al.*, "Deep belief networks for phone recognition," in *Nips workshop on deep learning for speech recognition and related applications*, vol. 1, no. 9, 2009, p. 39.

[13] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[14] M. Xue, Y. Zhang, J. Wang, and W. Liu, "Intellectual property protection for deep learning models: Taxonomy, methods, attacks, and evaluations," *IEEE Transactions on Artificial Intelligence*, pp. 1–1, 2021.

[15] Z. Li, C. Hu, Y. Zhang, and S. Guo, "How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of dnn," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 126–137.

[16] E. Le Merrer, P. Perez, and G. Trédan, "Adversarial frontier stitching for remote neural network watermarking," *Neural Computing and Applications*, vol. 32, no. 13, pp. 9233–9244, 2020.

[17] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[18] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *Available: http://yann. lecun. com/exdb/mnist*, 1998.

[19] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization," *CoRR*, vol. abs/1610.02391, 2016. [Online]. Available: http://arxiv.org/abs/1610.02391

[22] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.06434