PoisonedEncoder: Poisoning the Unlabeled Pre-training Data in Contrastive Learning

Hongbin Liu, Jinyuan Jia, and Neil Zhenqiang Gong, *Duke University*

https://www.usenix.org/conference/usenixsecurity22/presentation/liu-hongbin

This paper is included in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

PoisonedEncoder: Poisoning the Unlabeled Pre-training Data in Contrastive Learning

Hongbin Liu Jinyuan Jia Neil Zhenqiang Gong
Duke University
{hongbin.liu, jinyuan.jia, neil.gong}@duke.edu

Abstract

Contrastive learning pre-trains an image encoder using a large amount of unlabeled data such that the image encoder can be used as a general-purpose feature extractor for various downstream tasks. In this work, we propose *PoisonedEncoder*, a *data poisoning attack* to contrastive learning. In particular, an attacker injects carefully crafted poisoning inputs into the unlabeled pre-training data, such that the downstream classifiers built based on the poisoned encoder for multiple target downstream tasks simultaneously classify attacker-chosen, arbitrary clean inputs as attacker-chosen, arbitrary classes. We formulate our data poisoning attack as a bilevel optimization problem, whose solution is the set of poisoning inputs; and we propose a contrastive-learning-tailored method to approximately solve it. Our evaluation on multiple datasets shows that PoisonedEncoder achieves high attack success rates while maintaining the testing accuracy of the downstream classifiers built upon the poisoned encoder for non-attacker-chosen inputs. We also evaluate five defenses against PoisonedEncoder, including one *pre-processing*, three *in-processing*, and one *post-processing* defenses. Our results show that these defenses can decrease the attack success rate of PoisonedEncoder, but they also sacrifice the utility of the encoder or require a large clean pre-training dataset.

1 Introduction

Contrastive learning [14,24,44] is an emerging machine learning paradigm. Specifically, an *encoder provider* (e.g., Google, OpenAI, and Meta) pre-trains encoders (we focus on image encoders) using a large amount of unlabeled data (called *pre-training data*) automatically collected from the Internet via a crawler; and the image encoders are then used as general-purpose feature extractors for various downstream tasks. The unlabeled data could be images (known as *single-modal contrastive learning*) [14, 24] or (image, text) pairs (known as *multi-modal contrastive learning*) [44]. Given an image encoder as a feature extractor and a small amount of *downstream training data* for a downstream task (e.g., traffic sign recognition, face mask detection, and face recognition), a downstream classifier can be trained using the standard supervised learning or semi-supervised learning.

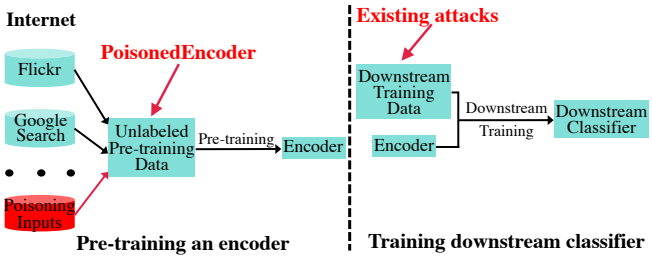


Figure 1: Our PoisonedEncoder vs. existing data poisoning attacks in the contrastive learning pipeline.

In this work, we study targeted data poisoning attack to contrastive learning. Specifically, an attacker aims to make the downstream classifiers built for multiple downstream tasks (called *target downstream tasks*) misclassify arbitrary, attacker-chosen clean inputs (called *target inputs*) as arbitrary, attacker-chosen classes (called *target classes*). For instance, the attacker may desire a target traffic-sign classifier to misclassify a stop sign (target input) as speed limit (target class) and a target face-mask classifier to misclassify a person not wearing mask (target input) as wearing mask (target class). Such attacks pose significant challenges to contrastive learning in safety and security critical applications.

Existing targeted data poisoning attacks mainly focus on supervised learning [22,40,49,51] and semi-supervised learning [11,54]. When applied to attack target downstream classifiers in contrastive learning, they aim to tamper with their downstream training data, as illustrated in Figure 1. As a result, these attacks are not applicable when the downstream training data maintains integrity. For instance, when the downstream training data is proprietary data obtained from a trustworthy source; and since the downstream training datasets are often small in contrastive learning, they may also be manually

cleaned up. However, the unlabeled pre-training data in contrastive learning is often collected from the Internet and thus is vulnerable to poisoning. For instance, an attacker can publish its poisoning images/inputs on crawler-accessible websites such as social media websites, so they can be collected as a part of the pre-training data by an encoder provider. Therefore, we focus on poisoning the unlabeled pre-training data in this work. A recent work [12] studied targeted data poisoning attack to contrastive learning. However, they focused on multimodal contrastive learning. In particular, their attack injects poisoning (image, text) pairs into the pre-training data, where the image is a target input and the text includes the target class name, such that the poisoned encoders produce similar feature vectors for the target input and target class. Their attack is not applicable to single-modal contrastive learning because it does not use text. Another recent work [31] studied backdoor attacks to contrastive learning. However, they compromise the pre-training process, while we poison the pre-training data, which is a more realistic threat model.

Our work: We propose PoisonedEncoder, the first targeted data poisoning attack to single-modal contrastive learning. PoisonedEncoder injects carefully crafted poisoning inputs into the unlabeled pre-training data such that multiple target downstream classifiers trained based on the poisoned encoder misclassify the target inputs as target classes simultaneously.

The key challenge of PoisonedEncoder is to craft the poisoning inputs to achieve the attack goals. To address the challenge, we formulate PoisonedEncoder as a bilevel optimization problem, whose solution is the set of poisoning inputs. Specifically, the outer optimization problem captures the attack goals on the poisoned encoder, while the inner optimization problem captures that the poisoned encoder is learnt on the poisoned pre-training data. However, it is notoriously challenging to solve bilevel optimization problems. To address the challenge, we propose an approximate solution that is tailored to contrastive learning. Specifically, contrastive learning aims to learn an encoder that produces similar feature vectors for two randomly cropped views of an image. Based on this observation, we concatenate a target input and an image (called *reference input*) in the target class either horizontally or vertically to construct a poisoning input. Since two randomly cropped views of a poisoning input may correspond to the target input and reference input, the poisoned encoder may produce similar feature vectors for the target input and reference input. Therefore, a target downstream classifier built upon the poisoned encoder is likely to predict the same class for the target and reference inputs, which is the target class.

We evaluate PoisonedEncoder on multiple datasets in different settings. On one hand, PoisonedEncoder achieves high attack success rate, i.e., the target downstream classifiers built based on the poisoned encoder misclassify a large fraction of target inputs as the target classes. On the other hand, PoisonedEncoder maintains the encoder's utility, i.e., a downstream classifier built based on a clean encoder and that built based on a poisoned encoder for a target/non-target downstream task have similar accuracy for non-target inputs. PoisonedEncoder can attack multiple target inputs and multiple target downstream tasks simultaneously. Moreover, we extend state-of-the-art targeted data poisoning attacks designed for supervised learning [22] and semi-supervised learning [11] to poison the pre-training data in contrastive learning. Our results show that PoisonedEncoder achieves higher attack success rate than these extended attacks.

Defenses against data poisoning attacks can be roughly categorized into *pre-processing*, *in-processing*, and *post-processing*. We explore one pre-processing defense (i.e., detecting and removing poisoning inputs before pre-training an encoder), three in-processing defenses (i.e., early stopping of pre-training an encoder and training a downstream classifier, ensemble method, and pre-training without random cropping), and one post-processing defense (i.e., fine-tuning a potentially poisoned encoder using a clean pre-training dataset). Our results show that these defenses can reduce the attack success rate of PoisonedEncoder, but they sacrifice the utility of the encoder or require substantial manual efforts to collect a large clean pre-training dataset.

Our key contributions can be summarized as follows:

- We propose PoisonedEncoder, the first data poisoning attack to single-modal contrastive learning.
- We formulate PoisonedEncoder as a bilevel optimization problem and propose a contrastive-learning-tailored method to solve it.
- We extensively evaluate PoisonedEncoder in different settings and compare it with state-of-the-art attacks extended to contrastive learning.
- We explore five defenses against PoisonedEncoder.

2 Background on Contrastive Learning

The pipeline of contrastive learning consists of two stages, i.e., pre-training an encoder and training downstream classifiers. Next, we discuss these two stages in more detail.

2.1 Pre-training an Encoder

A key module of pre-training an encoder is *random data augmentation*. Given a pre-training input, the random data augmentation module generates two augmented views via a series of random augmentation operations, such as random cropping, random horizontal flipping, and randomly converting to grayscale. Two augmented views are named *positive pair* (or *negative pair*) if they are from the same (or different) pre-training input(s). Roughly speaking, the core idea of contrastive learning is to pre-train an encoder that outputs similar

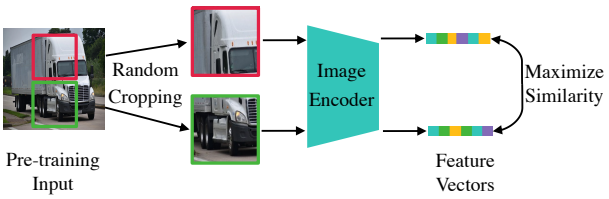


Figure 2: Illustration of contrastive learning when random cropping is used to generate augmented views of a pre-training input.

(or dissimilar) feature vectors for a positive (or negative) pair. In particular, *random cropping*, which first crops a random region with a certain size from an input and then resizes the region to have the same size as the input, is a crucial data augmentation operation for contrastive learning [14, 24]. Figure 2 illustrates the core idea of contrastive learning when random cropping is used as a data augmentation operation. As we will discuss, our PoisonedEncoder exploits the random cropping operation to poison contrastive learning. Next, we describe two representative contrastive learning algorithms, SimCLR [14] and MoCo [24].

SimCLR [14]: Besides the data augmentation module, two major components of SimCLR are the encoder f and the projection head h. The encoder f generates a feature vector $f(\mathbf{x})$ for an input $\mathbf{x}$, while the projection head h performs non-linear mapping for the feature vector $f(\mathbf{x})$ via a multilayer perceptron, which is used to calculate contrastive loss. Given a mini-batch of K pre-training inputs (denoted as $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K\}$), each of them is transformed into two randomly augmented views via the data augmentation module, producing $2 \cdot K$ augmented inputs: $\{\mathbf{x}'_1, \mathbf{x}'_2, \cdots, \mathbf{x}'_{2 \cdot K}\}$. For simplicity, we use $\mathbf{u}_i$ to denote the projection head's output for the augmented view $\mathbf{x}'_i$, i.e., $\mathbf{u}_i = h(f(\mathbf{x}'_i))$, where $i = 1, 2, \cdots, 2 \cdot K$. In particular, given a positive pair $(\mathbf{x}'_i, \mathbf{x}'_j)$, SimCLR defines its contrastive loss as follows:

$$\ell_{i,j} = -\log\left(\frac{\exp\left(\text{sim}\left(\mathbf{u}_i, \mathbf{u}_j\right)/\tau\right)}{\sum_{k=1}^{2 \cdot K} \mathbb{I}(k \neq i) \cdot \exp\left(\text{sim}\left(\mathbf{u}_i, \mathbf{u}_k\right)/\tau\right)}\right), \quad (1)$$

where $\text{sim}(\cdot, \cdot)$ denotes the cosine similarity score, $\mathbb{I}(k \neq i)$ is an indicator function, and τ is a temperature parameter. The overall contrastive loss for this mini-batch is the sum of $\ell_{i,j}$ over all positive pairs. Then SimCLR minimizes the overall contrastive loss via jointly pre-training the encoder f and the projection head h by back-propagation.

MoCo [24]: In addition to data augmentation, MoCo has three key components: a query encoder f_q, a momentum encoder f_m, and a dictionary $\mathcal{D}$. Given an input $\mathbf{x}$, the query encoder f_q and the momentum encoder f_m output feature vectors $f_q(\mathbf{x})$ and $f_m(\mathbf{x})$ for it, respectively. In particular, the query encoder f_q and the momentum encoder f_m have the same encoder architecture. The feature vectors outputted by the momentum encoder f_m are also called *keys*. The dictionary $\mathcal{D}$ maintains a queue of keys via adding keys that are produced in the most recent mini-batches while removing those oldest keys from it. MoCo updates the momentum encoder f_m much slower than the query encoder f_q to maintain the consistency of keys in the dictionary $\mathcal{D}$. Similar to SimCLR, given a mini-batch of K pre-training inputs $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K\}$, the data augmentation module generates two randomly augmented views for each of them. For each positive pair $(\mathbf{x}'_i, \mathbf{x}'_j)$, $\mathbf{x}'_i$ and $\mathbf{x}'_j$ are fed to the query encoder f_q and momentum encoder f_m, respectively. The key $f_m(\mathbf{x}'_j)$ is enqueued into the dictionary $\mathcal{D}$. Given a positive pair $(\mathbf{x}'_i, \mathbf{x}'_j)$, the contrastive loss of MoCo is defined as follows:

$$\ell_{i,j} = -\log\left(\frac{\exp\left(\text{sim}\left(f_q(\mathbf{x}'_i), f_m(\mathbf{x}'_j)\right)/\tau\right)}{\sum_{\mathbf{d} \in \mathcal{D}} \exp\left(\text{sim}\left(f_q(\mathbf{x}'_i), \mathbf{d}\right)/\tau\right)}\right), \quad (2)$$

where $\text{sim}(\cdot, \cdot)$ means cosine similarity and τ is a temperature parameter. The overall contrastive loss for this mini-batch is the sum of $\ell_{i,j}$ over all positive pairs. MoCo minimizes the overall contrastive loss via pre-training the query encoder f_q by back-propagation and then updates the momentum encoder f_m in a slowly evolving manner. The learnt query encoder is used as a pre-trained encoder for downstream tasks.

2.2 Training Downstream Classifiers

The pre-trained encoder is used as a general-purpose feature extractor for various downstream tasks. In particular, given a pre-trained encoder and a small amount of downstream training data for a downstream task, the encoder outputs feature vectors for the training inputs. Then, a downstream classifier is trained based on the extracted feature vectors and corresponding labels through standard supervised learning. In the testing phase, the encoder is first used to extract feature vectors for testing inputs. Then the downstream classifier outputs predicted labels for the extracted feature vectors.

3 Problem Formulation

3.1 Threat Model

Attacker's goal: An attacker selects one or multiple *target downstream tasks*. For each target downstream task t, the attacker chooses k_t *target inputs*. We use x_{ti} to denote the i-th target input for the t-th target downstream task, where $i = 1, 2, \cdots, k_t$ and $t = 1, 2, \cdots, T$. For each target input x_{ti}, the attacker chooses a target class y_{ti}, which is different from the true class of x_{ti}. We note that multiple target inputs could have the same target class, i.e., the attacker desires these target inputs to be classified as the same target class. The attacker aims to inject poisoning inputs into the unlabeled pre-training

dataset, such that a poisoned encoder is pre-trained and the downstream classifiers trained based on the poisoned encoder for the target downstream tasks should simultaneously classify the attacker-chosen target inputs as the corresponding attacker-chosen target classes.

Attacker's background knowledge: We assume the attacker only has access to some images (called *reference inputs*) from each target class. We denote the set of reference inputs from target class y_{ti} as $X_{y_{ti}}$. We note that the reference inputs are not used to train the downstream classifiers for the target downstream tasks. The attacker can collect the reference inputs from different sources, e.g., from the Internet. Moreover, we assume the attacker does not know the encoder's pre-training dataset, architecture, nor loss function.

Attacker's capability: The attacker is able to inject poisoning images into the unlabeled pre-training dataset before it is used to pre-train an encoder. An encoder provider often automatically collects a large pre-training dataset from the Internet using a web crawler and uses it to pre-train an encoder. Therefore, the attacker can publish the poisoning images on the Internet, e.g., host them on some crawler-accessible websites or post them on social medias, so an encoder provider would collect them as a part of its pre-training dataset. We consider the attacker has resources to inject at most N poisoning images/inputs into the pre-training dataset. For simplicity, we denote the clean pre-training dataset as X_c and we denote the set of poisoning inputs as X_p.

3.2 Data Poisoning Attack

Given the attacker's goal, background knowledge, and capability, we formulate our data poisoning attack as follows. Recall that the attack requires downstream classifiers built based on the poisoned encoder to classify the target inputs as the corresponding target classes. However, it is challenging to directly quantify this goal using downstream classifiers, as we assume the attacker does not have control over the training of downstream classifiers in our threat model. To address the challenge, we propose to quantify the goal using the feature vectors outputted by the poisoned encoder. Intuitively, if the poisoned encoder produces similar feature vectors for a target input x_{ti} with a target class y_{ti} and reference inputs from the target class y_{ti}, then a downstream classifier built based on the poisoned encoder would be likely to classify the target input x_{ti} as the target class y_{ti}.

Therefore, the poisoned encoder should produce similar feature vectors for the target inputs and reference inputs in the same target class. In particular, we use a loss term $\mathcal{L}_{\text{sim}}(x_{ti}, x_r; \theta)$ to quantify feature similarity between a target input x_{ti} and a reference input x_r, where their feature vectors are outputted by an encoder θ and the feature similarity is the *cosine similarity* between the two feature vectors. We adopt cosine similarity because it is used by contrastive learning to measure feature similarity. Our goal is to construct the poisoning inputs X_p such that an encoder pre-trained on the poisoned pre-training dataset $X_c \cup X_p$ maximizes the total loss for all target inputs and reference inputs. Formally, we have the following optimization problem:

$$\max_{X_p} \frac{1}{\sum_{t=1}^{T}\sum_{i=1}^{k_t}|X_{y_{ti}}|} \sum_{t=1}^{T}\sum_{i=1}^{k_t}\sum_{x_r \in X_{y_{ti}}} \mathcal{L}_{\text{sim}}\left(x_{ti}, x_r; \theta^*\left(X_c \cup X_p\right)\right), \tag{3}$$

$$s.t.\ \theta^*\left(X_c \cup X_p\right) = \operatorname*{argmin}_{\theta} \mathcal{L}_{\text{CL}}\left(X_c \cup X_p; \theta\right), \tag{4}$$

where T is the number of target downstream tasks, k_t is the number of target inputs for target downstream task t, $X_{y_{ti}}$ is the set of reference inputs from the target class y_{ti}, $\theta^*\left(X_c \cup X_p\right)$ is the poisoned encoder pre-trained on the poisoned pre-training dataset $X_c \cup X_p$, Equation 3 means that the poisoned encoder should produce similar feature vectors for the target inputs and the corresponding reference inputs, and Equation 4 means that the poisoned encoder is pre-trained on the poisoned pre-training dataset using contrastive learning, where $\mathcal{L}_{\text{CL}}$ is the contrastive loss.

The set of poisoning inputs X_p is a solution to the above optimization problem. We note that the optimization problem is a *bilevel* one, where Equation 3 is the *outer optimization* and Equation 4 is the *inner optimization*.

4 Our PoisonedEncoder

4.1 Challenges

The key of our data poisoning attack is to construct a set of poisoning inputs, which is a solution to the bilevel optimization problem in Equation 3. However, it is notoriously challenging to solve bilevel optimization problems in general [40]. Specifically, gradient descent is a well-known iterative method to solve optimization problems. Given an initial set of poisoning inputs X_p, we calculate the gradient of Equation 3 with respect to X_p (i.e., $\frac{\partial \sum_{t=1}^{T}\sum_{i=1}^{k_t}\sum_{x_r \in X_{y_{ti}}} \mathcal{L}_{\text{sim}}\left(x_{ti}, x_r; \theta^*\left(X_c \cup X_p\right)\right)}{\sum_{t=1}^{T}\sum_{i=1}^{k_t}|X_{y_{ti}}| \cdot \partial X_p}$), and then we move X_p along the gradient with a small step. However, calculating such gradient requires the gradient of the poisoned encoder $\theta^*\left(X_c \cup X_p\right)$ with respect to the poisoning inputs X_p, i.e., $\frac{\partial \operatorname{argmin}_{\theta} \mathcal{L}_{\text{CL}}\left(X_c \cup X_p; \theta\right)}{\partial X_p}$. In our threat model, the attacker faces multiple challenges when calculating such gradient: 1) the attacker does not know the clean pre-training dataset X_c, 2) the attacker does not know the contrastive loss function $\mathcal{L}_{\text{CL}}$, and 3) the encoder is a highly non-linear deep neural network. As a result, it is challenging to use iterative method to solve the bilevel optimization problem.

Therefore, we resort to non-iterative heuristic solutions to the bilevel optimization problem. In particular, we propose an approximate solution that is tailored to contrastive learning. Our approximate solution does not require the gradient of the

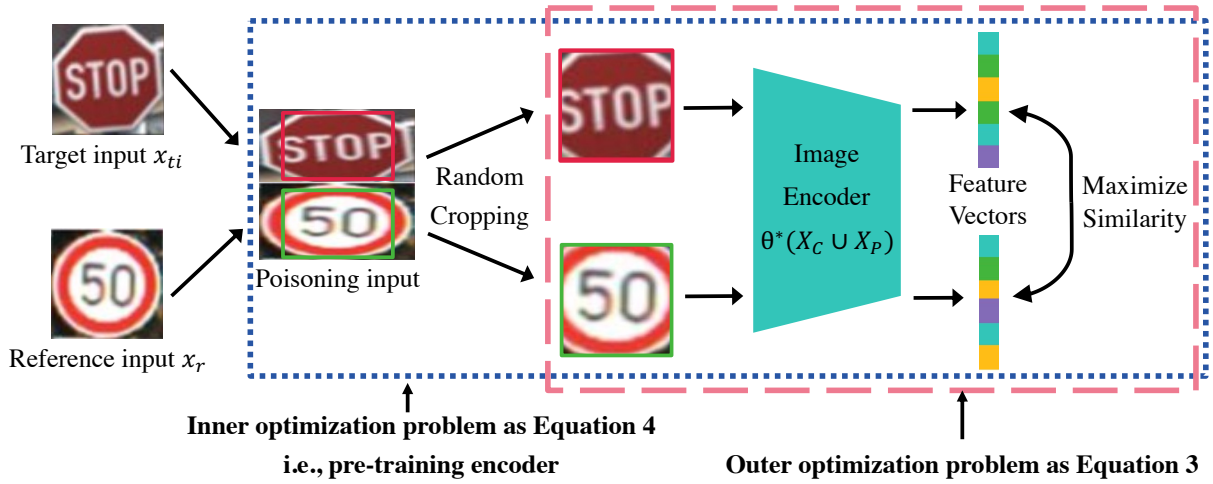
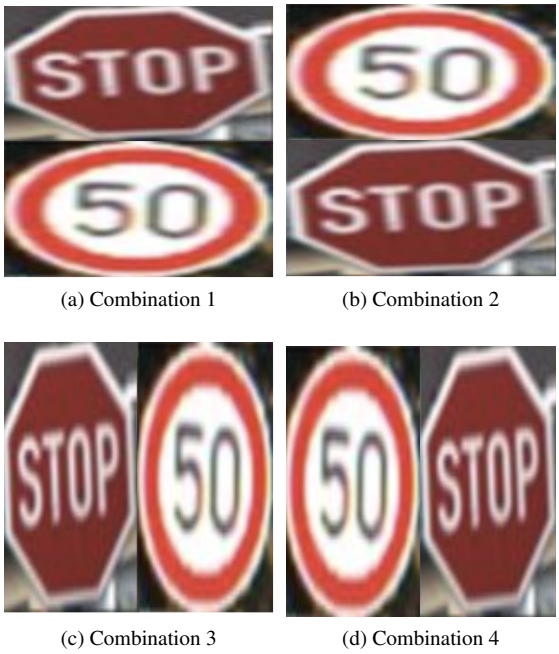


Figure 3: Illustration of PoisonedEncoder. Solving the inner optimization problem on the poisoned pre-training data also approximately maximizes the outer objective function.



(a) Combination 1 (b) Combination 2

(c) Combination 3 (d) Combination 4

Figure 4: Four combination methods to construct poisoning inputs from a target input and a reference input.

poisoned encoder $\theta^*(X_c \cup X_p)$ with respect to the poisoning inputs X_p. Moreover, our solution does not require access to the clean pre-training dataset nor the contrastive loss function, and is applicable to highly non-linear deep neural network based encoder. Next, we first introduce the intuition of our approximate solution, and then describe the details on constructing poisoning inputs.

4.2 Our Intuition

The key idea of our method is to construct poisoning inputs such that solving the inner optimization also approximately solves the outer optimization. Note that an encoder provider essentially solves the inner optimization problem when pre-training an encoder based on a poisoned pre-training dataset. Therefore, in our attack, an encoder pre-trained on a poisoned pre-training dataset approximately maximizes the outer objective function. Contrastive learning aims to learn an encoder that produces similar feature vectors for two randomly cropped views of an input. In other words, the inner optimization in Equation 4 aims to optimize a poisoned encoder that outputs similar feature vectors for any two randomly cropped views of an input in the poisoned pre-training dataset $X_c \cup X_p$. The outer optimization in Equation 3 aims to maximize the similarity between a target input and a reference input. Therefore, if two randomly cropped views of a poisoning input correspond to a target input and a reference input, then the poisoned encoder obtained from the inner optimization would also maximize the similarity loss in the outer optimization.

Based on this intuition, our PoisonedEncoder constructs a poisoning input by combining a target input and a reference input, which is illustrated in Figure 3. Moreover, to increase diversity of the poisoning inputs, we consider four ways of combining a target input and a reference input, i.e., up (target)-down(reference), up (reference)-down(target), left (target)-right(reference), and left (reference)-right(target), which are illustrated in Figure 4. As our experiments will show, PoisonedEncoder achieves higher attack success rate by using all four combination methods. We note that such combined images also naturally appear on the Internet. Figure 12 in Appendix shows some examples of combined images from Google search.

Algorithm 1: Our PoisonedEncoder

Input: Target downstream tasks $t = 1, 2, \cdots, T$, target inputs x_{ti} and target classes y_{ti} where $i = 1, 2, \cdots, k_t$, reference inputs $X_{y_{ti}}$ from the target class y_{ti}, number of poisoning inputs N, a set of combination methods $\mathcal{M}$.
Output: Poisoning inputs X_P
$X_P = \emptyset$ \\ Initialize X_P as an empty set
while $|X_P| \leq N$ **do**
    Randomly pick a target input x_{ti}
    Randomly pick a reference input $x_r \in X_{y_{ti}}$
    Randomly pick a combination method $m \in \mathcal{M}$
    Combine x_{ti} and x_r based on m as poisoning input
    Add the poisoning input to X_P
end while
return X_P

4.3 Constructing Poisoning Inputs

Algorithm 1 shows the algorithmic details of PoisonedEncoder. Given a set of target downstream tasks, target inputs and target classes for each target downstream task, reference inputs from each target class, and a set of combination methods. We construct N poisoning inputs. To construct a poisoning input, we randomly pick a target input with a target class, randomly pick a reference input from the target class, and randomly pick a combination method; and then we combine the target input and the reference input according to the combination method as a poisoning input.

5 Evaluation

5.1 Experimental Setup

Pre-training datasets and encoders: We consider two benchmark datasets, CIFAR10 [35] and Tiny-ImageNet [3], as pre-training datasets. CIFAR10 dataset contains 50,000 training images and 10,000 testing images. Tiny-ImageNet dataset contains 100,000 training images and 10,000 testing images. Following prior work [14, 24], we use the training images of each dataset as a pre-training dataset and we use multiple data augmentations (random cropping, color jittering, Gaussian blurring, random grayscale, and random horizontal flipping) when pre-training encoders in experiments. Following prior work [14, 24], we consider different scales of random cropping, i.e., scale is randomly sampled from [0.08, 1] each time. Moreover, we rescale each image to 32x32 in both datasets.

Unless otherwise mentioned, we use ResNet18 [25] as the neural network architecture for an encoder and SimCLR [14] to pre-train both poisoned and clean encoders. In particular, we adopt the publicly available implementation [1] of SimCLR with the default settings. We pre-train an encoder for 300 epochs using the Adam optimizer, an initial learning rate of 0.001, and a batch size of 512. However, we will also explore the impact of encoder architecture, contrastive learning algorithm, learning rate, batch size, and number of pre-training epochs on PoisonedEncoder.

Training downstream classifiers: Given a pre-trained (poisoned or clean) encoder, we train downstream classifiers on three downstream datasets, STL10 [16], Facemask [2], and EuroSAT [26]. STL10 contains 13,000 labeled color images from 10 classes. Specifically, the labeled data is divided into 5,000 training images and 8,000 testing images. Facemask dataset is a binary classification task for detecting whether a person wears a mask given his/her face image. It contains 10,000 training images and 993 testing images. EuroSAT dataset consists of 10 classes with 27,000 labeled geographical images collected by satellites. We randomly and evenly split the images into training and testing sets in EuroSAT. Like the pre-training data, we also rescale each image to 32x32 in these downstream datasets.

Following the contrastive learning paradigm [14, 24], we adopt a neural network with one fully-connected layer as a downstream classifier. Moreover, we train a downstream classifier for 100 epochs using Adam optimizer, an initial learning rate of 0.001, and a batch size of 512. We will explore the impact of learning rate, batch size, and number of epochs used to train a downstream classifier on PoisonedEncoder.

Evaluation metrics: We use *attack success rate (ASR)* as well as a downstream classifier's *clean accuracy (CA)* and *poisoned accuracy (PA)* to evaluate PoisonedEncoder. ASR is used to measure the attack success of PoisonedEncoder, while CA and PA are used to measure the impact of PoisonedEncoder on the utility of the encoder. ASR is the fraction of target inputs that are predicted as the corresponding target classes by the target downstream classifiers trained on a poisoned encoder. A higher ASR indicates a more successful attack. A downstream classifier's CA and PA are the testing accuracy of the downstream classifier trained on a clean and poisoned encoder, respectively. A smaller difference between CA and PA indicates that the attack better preserves the utility of the encoder.

Compared data poisoning attacks: We compare our PoisonedEncoder with the following two baseline attacks. Note that these attacks were originally designed for supervised learning and semi-supervised learning, and we extend them to contrastive learning.

- **Witches' Brew [22]:** Witches' Brew is a state-of-the-art targeted data poisoning attack to deep neural network classifier. The attack is formulated as a bilevel optimization problem. Roughly speaking, Witches' Brew uses a gradient-based iterative method to solve the bilevel optimization problem. In each iteration, they update the poisoning inputs to maximize the alignment between the gradient of the inner objective function and that of

Table 1: ASRs of different attacks.

Pre-training Dataset	Target Downstream Dataset	Witches' Brew	ICP	Ours
CIFAR10	STL10	0.1	0.5	0.8
	Facemask	0.1	0.6	0.9
	EuroSAT	0.0	0.2	0.5
Tiny-ImageNet	STL10	0.0	0.4	0.7
	Facemask	0.1	0.8	1.0
	EuroSAT	0.0	0.2	0.4

the outer objective function, where the gradient is with respect to the parameters of a clean classifier. We extend their method to solve our formulated bilevel optimization problem to craft poisoning inputs. Specifically, we optimize poisoning inputs to maximize the alignment between $\sum_{t=1}^{T}\sum_{i=1}^{k_t}\sum_{x_r \in X_{y_{ti}}} -\nabla_\theta \mathcal{L}_{\text{sim}}(x_{ti}, x_r; \theta(X_c \cup X_p))$ and $\nabla_\theta \mathcal{L}_{\text{CL}}(X_c \cup X_p; \theta)$, where θ is a clean encoder. Moreover, following Withes' Brew, we replace $\nabla_\theta \mathcal{L}_{\text{CL}}(X_c \cup X_p; \theta)$ as $\nabla_\theta \mathcal{L}_{\text{CL}}(X_p; \theta)$. Witches' Brew requires the attacker has access to a clean encoder and the contrastive loss function $\mathcal{L}_{\text{CL}}$, where we assume the clean encoder is trained on a clean pre-training dataset in our experiments. Note that we give advantages to Witches' Brew, as our PoisonedEncoder does not require these information.

- **Interpolation Consistency Poisoning (ICP) [11]:** Carlini proposed ICP, a targeted data poisoning attack, to semi-supervised learning. Roughly speaking, ICP crafts unlabeled poisoning inputs as interpolations between a target input and reference inputs. We apply ICP to contrastive learning. For a fair comparison, PoisonedEncoder and ICP use the same target input and reference inputs to craft poisoning inputs. We implemented ICP and adopted its default parameter settings from the paper.

Parameter settings: Unless otherwise mentioned, we consider the following parameter settings for PoisonedEncoder: the attacker chooses one target downstream dataset, randomly picks one testing input from the target downstream dataset as a target input, and randomly picks a class that is not the true class of the target input as a target class. The attacker has 50 reference inputs that are testing inputs randomly sampled from the target class in the target downstream dataset. The attacker injects 1% poisoning inputs to a pre-training dataset. In particular, the attacker injects 500 and 1,000 poisoning inputs into the pre-training datasets CIFAR10 and Tiny-ImageNet, respectively. For each experiment, we repeat it for 10 trials and report the average results. Unless otherwise mentioned, we assume the target downstream dataset is STL10. We note that after combining a target input and a reference input as a poisoning input, we resize the poisoning input to have the same size as the target/reference input, i.e., 32x32. We performed experiments on 18 NVIDIA-RTX-6000 GPUs, each of which has 24 GB memory.

Table 2: Values of the outer objective function in Equation 3 obtained under different attacks, where pre-training dataset is CIFAR10 and target downstream dataset is STL10.

No Attack	Witches' Brew	ICP	Ours
0.183	0.257	0.463	0.689

5.2 Experimental Results

PoisonedEncoder achieves high attack success rates: Table 1 shows the ASRs of different attacks on different pre-training datasets and target downstream datasets. Our PoisonedEncoder achieves higher ASRs than the compared attacks. Specifically, our attack achieves at least 0.2 higher ASR than ICP, while Witches' Brew is almost ineffective. The reason is that ICP and Witches' Brew were respectively tailored for semi-supervised learning and supervised learning, and they achieve suboptimal success rates when extended to contrastive learning.

In particular, the poisoning inputs constructed by PoisonedEncoder are better solutions to our formulated bilevel optimization problem in Equation 3 and 4 than those constructed by Witches' Brew and ICP. To further illustrate this point, we calculate the cosine similarity score between the feature vector of the target input and that of each reference input, and we compute the average of the cosine similarity scores, which is the value of the outer objective function in Equation 3. Table 2 shows the average cosine similarity scores (i.e., the outer objective function values) under different attacks. We observe that PoisonedEncoder achieves higher average cosine similarity score, which confirms that PoisonedEncoder is a better solution to the bilevel optimization problem than Witches' Brew and ICP.

We also observe that ICP achieves moderate ASRs and is much more successful than Witches' Brew. This is because both the semi-supervised learning algorithms attacked by ICP and contrastive learning use data augmentations during (pre-)training. As a result, ICP is a better solution to our formulated bilevel optimization problem than Witches' Brew, as shown in Table 2.

Our PoisonedEncoder is also effective when the pre-training dataset is large. For instance, our attack's ASR is 0.7 when the pre-training dataset is the full ImageNet dataset [46] and the target downstream dataset is Facemask, where the parameters are set to their default settings. It took us around 5 days to pre-train 10 encoders on the full ImageNet dataset. Due to the limited computing resources, we were not able to perform other experiments on the full ImageNet dataset.

PoisonedEncoder preserves utility: Table 3 shows the CA and PA of different downstream classifiers. In particular, given a pre-training dataset and a target downstream dataset, we evaluate both CA and PA on all the three downstream datasets. In most cases, the gap between the CA and PA for a downstream dataset is within 1%. Our results indicate that

Table 3: CAs and PAs of downstream classifiers.

Pre-training Dataset	Target Downstream Dataset	Downstream Dataset	CA	PA
CIFAR10	STL10	STL10	0.718	0.715
		Facemask	0.947	0.952
		EuroSAT	0.815	0.821
	Facemask	STL10	0.718	0.716
		Facemask	0.947	0.937
		EuroSAT	0.815	0.820
	EuroSAT	STL10	0.718	0.724
		Facemask	0.947	0.953
		EuroSAT	0.815	0.797
Tiny-ImageNet	STL10	STL10	0.635	0.637
		Facemask	0.965	0.968
		EuroSAT	0.816	0.853
	Facemask	STL10	0.635	0.633
		Facemask	0.965	0.977
		EuroSAT	0.816	0.855
	EuroSAT	STL10	0.635	0.633
		Facemask	0.965	0.970
		EuroSAT	0.816	0.844

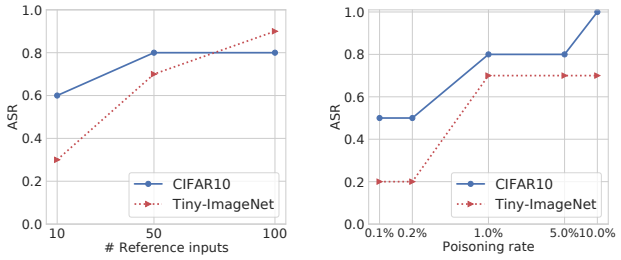
Table 4: ASR of PoisonedEncoder when using different combination methods to construct poisoning inputs. The combination methods come from Figure 4.

(a) Pre-trained on CIFAR10

Combination Method	ASR
1	0.4
2	0.5
3	0.5
4	0.6
1+2	0.5
1+2+3	0.6
1+2+3+4	0.8

(b) Pre-trained on Tiny-ImageNet

Combination Method	ASR
1	0.3
2	0.3
3	0.2
4	0.4
1+2	0.5
1+2+3	0.6
1+2+3+4	0.7



(a) (b)

Figure 5: (a) Impact of the number of reference inputs. (b) Impact of poisoning rate, where the x-axis is in log-scale.

PoisonedEncoder preserves the utility of the encoder. This is because our poisoning inputs are still natural images and a neural network encoder is expressive, which not only learns high-performing feature representations for non-target inputs but also learns the hidden behavior from our attack. We also found PoisonedEncoder preserves the utility for the target class and class of the target input. For instance, when the pre-training dataset is CIFAR10 and the target downstream dataset is STL10, the CA and PA of the downstream classifier for the target class (or class of the target input) are respectively 0.702 and 0.698 (or 0.742 and 0.745).

Table 5: ASR of PoisonedEncoder for different contrastive learning algorithms.

(a) Pre-trained on CIFAR10

Pre-training Algorithm	ASR
SimCLR	0.8
MoCo	0.6

(b) Pre-trained on Tiny-ImageNet

Pre-training Algorithm	ASR
SimCLR	0.7
MoCo	0.8

Table 6: ASR of PoisonedEncoder for different architectures.

(a) Pre-trained on CIFAR10

Encoder Architecture	ASR
ResNet18	0.8
VGG11	0.9
MobileNet-v2	0.8

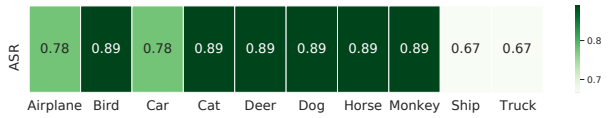
(b) Pre-trained on Tiny-ImageNet

Encoder Architecture	ASR
ResNet18	0.7
VGG11	0.8
MobileNet-v2	0.8

Impact of different combination methods: PoisonedEncoder uses four combination methods to construct poisoning inputs. One natural question is whether it is necessary to use all of them. Table 4 shows the ASR of PoisonedEncoder when using any one of the four combination methods or their combinations, e.g., 1+2+3 means using the first three combination methods. Our results show that PoisonedEncoder with any of the four combination methods achieves similar ASRs. When using more combination methods, PoisonedEncoder achieves higher ASRs, e.g., 1+2+3 achieves higher ASRs than 1+2, and 1+2+3+4 achieves higher ASRs than 1+2+3. This is because the poisoning inputs are more diverse when more combination methods are used.

Impact of the number of reference inputs and poisoning rate: Figure 5a and Figure 5b respectively show the impact of the number of reference inputs and poisoning rate on ASR of PoisonedEncoder for the two pre-training datasets. A general trend is that the ASR of our PoisonedEncoder first increases and then saturates as the attacker uses more reference inputs or injects more poisoning inputs (i.e., poisoning rate increases). This is because the pre-trained poisoned encoder is more likely to produce similar feature vectors for the target inputs and some reference inputs when the number of reference inputs or the poisoning rate is larger.

PoisonedEncoder is agnostic to contrastive learning algorithm and encoder architecture: Table 5 shows the ASR of PoisonedEncoder when SimCLR and MoCo are used to pre-train encoders. For MoCo, we adopt its publicly available implementation [4] with the default settings. Our results show that PoisonedEncoder is effective for both contrastive learning algorithms. This is because both algorithms use random cropping to generate augmented views during pre-training. Table 6 shows ASR of PoisonedEncoder for different encoder archi-



(a) ASR for different true classes of target inputs

(b) ASR for different target classes of target inputs

Figure 6: ASR for different true or target classes.

tectures. Our results show that PoisonedEncoder is agnostic to encoder architecture. This is because PoisonedEncoder does not rely on encoder architecture to construct poisoning inputs.

Impact of the true class and target class of the target input: Figure 6a shows the ASR of PoisonedEncoder when the target input is from different true classes, where the pre-training dataset is CIFAR10 and the target downstream dataset is STL10 that has 10 classes. In particular, given a true class of the target downstream dataset, we have 9 target classes that are not the true class; and for each target class, we randomly sample a target input from the true class and assign the target class for it. Therefore, we perform 9 experimental trials for a true class, and the ASR of PoisonedEncoder for the true class is averaged over the 9 trials. Our results indicate that some true classes (e.g., deer, bird, dog, etc.) are easier to be attacked than other true classes (e.g., ship, airplane, etc.). We suspect the reason is that the pre-trained encoder's learned feature representations are more robust for some true classes, which are harder to be attacked.

Figure 6b shows the ASR of PoisonedEncoder for different target classes. In particular, given a target class, we have 9 true classes that are not the target class; and for each true class, we randomly sample a target input from the true class and assign the target class to it. Thus, we perform 9 experimental trials for each target class, and the ASR of PoisonedEncoder for the target class is averaged over the 9 trials. We have two observations. First, our PoisonedEncoder achieves very high ASRs (1.00) for most target classes. This means that, for most target classes, our poisoning inputs make the poisoned encoder output highly similar feature vectors for a target input and reference inputs, and thus the target downstream classifier trained on the poisoned encoder is very likely to predict the target input as the target class. Second, we observe that the ASRs of target class 'dog' (0.11) and target class 'monkey' (0.56) are much lower than those of the other target classes. We found the reason is that the target downstream classifier is less accurate for these two classes. Specifically, the target downstream classifier's PAs for the 'dog' and 'monkey' classes are respectively 0.495 and 0.542, which are much lower than the average 0.715 PA of the other classes.

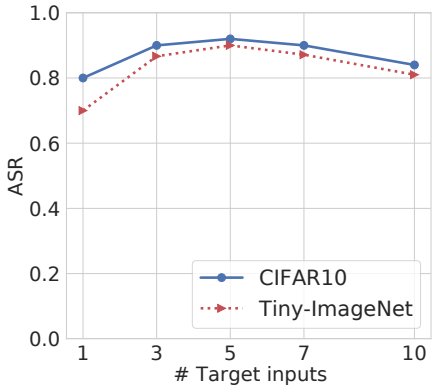


Figure 7: ASR of PoisonedEncoder when attacking multiple target inputs simultaneously.

Table 7: ASR of PoisonedEncoder when attacking three target downstream tasks simultaneously.

(a) Pre-trained on CIFAR10

Target Downstream Tasks	ASR
STL10	0.8
Facemask	0.9
EuroSAT	0.5

(b) Pre-trained on Tiny-ImageNet

Target Downstream Tasks	ASR
STL10	0.6
Facemask	1.0
EuroSAT	0.4

Impact of the number of target inputs and target downstream tasks: Figure 7 shows the ASR of our PoisonedEncoder when the attacker selects different numbers of random target inputs from the same target downstream task. The ASRs keep high as the number of target inputs increases. Table 7 shows the ASR on each target downstream dataset when the attacker attacks three target downstream datasets simultaneously. In this experiment, for each target downstream dataset, the attacker chooses one target input and target class at random, and the poisoning rate is 1%. We observe that PoisonedEncoder can effectively attack multiple target downstream tasks simultaneously. Moreover, by comparing Table 7 with Table 1, we find that attacking multiple target downstream tasks simultaneously achieves the same or comparable ASRs as attacking each target downstream task separately. This is because the poisoning rate for each target downstream task is the same in these two experiments. We also found that attacking multiple target downstream tasks simultaneously achieves smaller ASRs when the total poisoning rate is fixed to be 1%. In other words, ASR of PoisonedEncoder depends on the poisoning rate per target downstream task.

Impact of other parameters: Table 8 shows the ASR of PoisonedEncoder when pre-training encoder and training the target downstream classifier use different learning rates or batch sizes, where the pre-training dataset is CFIAR10 and target downstream dataset is STL10. We show the results of

Table 8: ASR of PoisonedEncoder when pre-training encoder and training the target downstream classifier use different learning rates or batch sizes.

Phase	Parameter	Value	ASR
Pre-training Encoders	Learning Rate	5×10^{-3}	0.5
		1×10^{-3}	0.8
		5×10^{-4}	0.8
	Batch size	256	0.8
		512	0.8
		1024	0.8
Training Downstream Classifier	Learning Rate	5×10^{-3}	0.8
		1×10^{-3}	0.8
		5×10^{-4}	0.8
	Batch size	256	0.7
		512	0.8
		1024	0.8

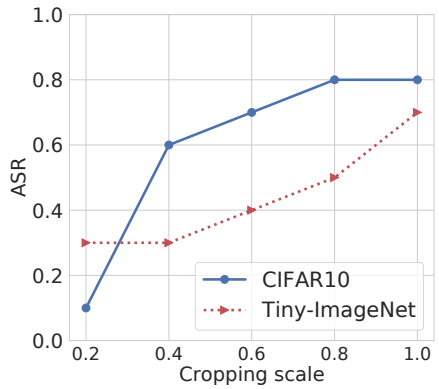


Figure 8: ASR of PoisonedEncoder when the attacker randomly crops a target/reference input with different cropping scales before using them to construct poisoning inputs, where no defenses are deployed.

different number of epochs used to pre-train encoder and train downstream classifier in Section 6.2 when we explore early stopping as a defense. PoisonedEncoder achieves consistently high ASRs across different parameter settings. PoisonedEncoder achieves a slightly lower ASR when the learning rate for pre-training encoder is 5×10^{-3}. This is because the encoder pre-trained with this learning rate outputs less-performing feature representations. In particular, the target downstream classifier has smaller CA/PA in this setting, making PoisonedEncoder achieve lower ASR.

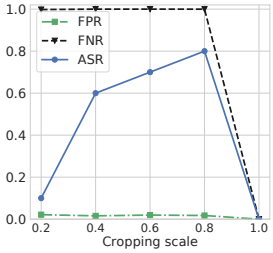
6 Defenses

Defenses against data poisoning attacks can be categorized into *pre-processing*, *in-processing*, and *post-processing* (details are discussed in Section 7.2). We explore one pre-processing defense (i.e., detecting poisoning inputs before pre-training), three in-processing defenses (i.e., re-designing the contrastive learning algorithm to be more robust), and one post-processing defense (i.e., fine-tuning a potentially poisoned encoder to remove the attack effect). In the following defense experiments, unless otherwise mentioned, we use the default parameter settings in Section 5.1 and consider STL10 as the target downstream task.

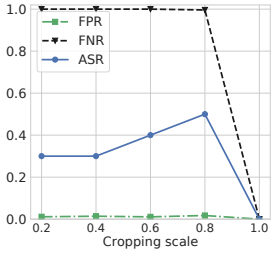
6.1 Pre-processing Defense

When there are a small number of reference inputs, PoisonedEncoder combines target inputs with a limited number of reference inputs to construct poisoning inputs. As a result, there are duplicate poisoning inputs. In particular, the poisoning input crafted from one of the four combinations of a target input and a reference input may appear multiple times in the poisoned pre-training data. Therefore, the encoder provider can first remove duplicates in the pre-training data. However, such duplicates checking is insufficient when the attacker has a large number of reference inputs. Therefore, we further explore a clustering-based method to detect poisoning inputs after removing duplicate pre-training inputs. Our intuition is that the poisoning inputs contain the same target inputs and thus may appear in the same clusters after clustering the pre-training inputs. Specifically, we group the pre-training images into K clusters using K-Means with ℓ_2-distance metric on the pixel values and predict the four clusters with the smallest average pair-wise ℓ_2-distance as poisoning inputs. We consider the four clusters because PoisonedEncoder uses the four combinations to construct poisoning inputs, each of which may correspond to a cluster.

To evade such pre-processing defense, the attacker can randomly crop a target/reference input before combining them to construct a poisoning input. Specifically, the attacker crops a random square region of the target input and a random square region of the reference input with a *cropping scale*, which is the ratio between the square region size and the target/reference input size. Then, the two random square regions are combined and resized to construct a poisoning input. Figure 8 shows the ASR of PoisonedEncoder when the attacker randomly crops a target/reference input with different cropping scales when the pre-processing defense is not deployed, where cropping scale = 1 means no cropping. Our results show that ASR decreases as the cropping scale decreases when the defense is not deployed. This is because a smaller cropping scale makes it harder for the poisoned encoder to learn similar feature vectors for the original target input and reference inputs.



(a) Pre-trained on CIFAR10



(b) Pre-trained on Tiny-ImageNet

Figure 9: Results of pre-processing defense. The cropping scale refers to that an attacker crops a target/reference input before using them to construct poisoning inputs.

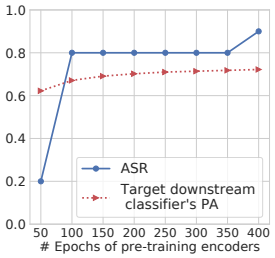
Figure 9 shows the *False Positive Rate (FPR)* and *False Negative Rate (FNR)* of detecting poisoning inputs, as well as the ASR of PoisonedEncoder after removing the detected poisoning inputs and pre-training on the remaining inputs, where $K = 100$ for K-Means, and we treat poisoning input as "positive" and clean input as "negative". When PoisonedEncoder does not use random cropping (i.e., cropping scale is 1) before combinations, the defense can effectively defend against PoisonedEncoder. However, the defense is ineffective, i.e., FNR is close to 1, when PoisonedEncoder randomly crops target/reference inputs to construct poisoning inputs. For instance, PoisonedEncoder with cropping scale 0.8 still achieves high ASRs.

6.2 In-processing Defenses

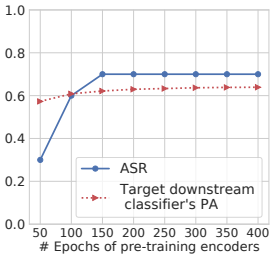
Early stopping: Intuitively, PoisonedEncoder relies on enough pre-training epochs to make the poisoned encoder produce similar feature vectors for a target input and the reference inputs, and enough training epochs of the target downstream classifier to predict the reference/target inputs as the target class. Therefore, early stopping, in which an encoder is pre-trained or a downstream classifier is trained using less epochs, could mitigate PoisonedEncoder. Figure 10 shows the ASR of PoisonedEncoder and the target downstream classifier's PA as a function of the number epochs used to pre-train an encoder or train the target downstream classifier. We observe that early stopping can reduce ASR when pre-training an encoder or training a downstream classifier using less epochs. However, such early stopping also reduces the accuracy of the downstream classifiers built based on the poisoned encoder. To summarize, early stopping can mitigate PoisonedEncoder at the cost of sacrificing utility.

Bagging: Jia et al. [29] showed that bagging, a well-known ensemble method, has intrinsic certified robustness against data poisoning attacks. The idea is to create multiple random subsamples of training data and learn a base classifier on each subsample; and we take majority vote among the base classifiers to classify a testing input. We extend bagging to contrastive learning as a defense against PoisonedEncoder. In particular, we create multiple random subsamples of pre-training data and pre-train a *base encoder* on each subsample using SimCLR; given a downstream task, we train a base downstream classifier based on each base encoder; and we take majority vote among the base downstream classifiers to classify a testing input. Such bagging-based method provably predicts the same label for a testing input when the number of poisoning inputs in the pre-training dataset is bounded.

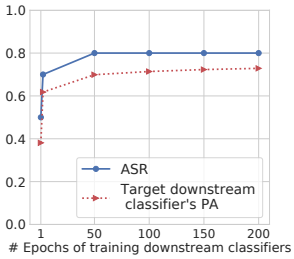
Table 9 shows the ASR of PoisonedEncoder and the PA of the target downstream task, where each subsample includes 500 randomly selected pre-training inputs and we create 100 subsamples/base encoders/base classifiers. The "Dropped Ratio" in the table is the decreased percentage of PA in Table 9, compared to that in Table 3. Our results show that bagging can defend against PoisonedEncoder at the cost of substantially reducing the accuracy of the downstream classifiers.



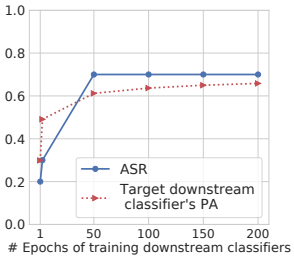
(a) Pre-trained on CIFAR10



(b) Pre-trained on Tiny-ImageNet



(c) Pre-trained on CIFAR10



(d) Pre-trained on Tiny-ImageNet

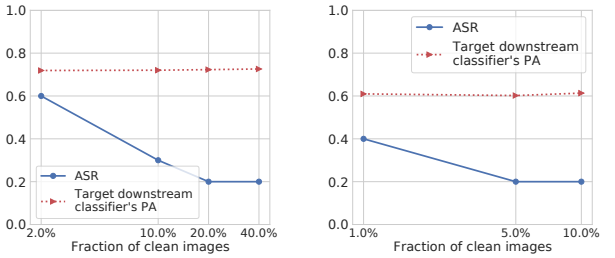
Figure 10: In-processing defense: early stopping. First row: early stopping when pre-training an encoder. Second row: early stopping when training a downstream classifier.

Table 9: In-processing defense: bagging. "Dropped Ratio" is the decreased percentage of the PA when bagging is used, compared to when bagging is not used.

Pre-training Dataset	ASR	PA	Dropped Ratio (%)
CIFAR10	0.0	0.431	39.2
Tiny-ImageNet	0.0	0.415	34.9

Table 10: In-processing defense: pre-training without random cropping. "Dropped Ratio" is the decreased percentage of the PA when random cropping is not used, compared to when random cropping is used.

Pre-training Dataset	ASR	PA	Dropped Ratio (%)
CIFAR10	0.1	0.391	45.3
Tiny-ImageNet	0.1	0.380	40.3



(a) Pre-trained on CIFAR10 (b) Pre-trained on Tiny-ImageNet

Figure 11: Post-processing defense. The x-axis is in log-scale.

Pre-training without random cropping: PoisonedEncoder exploits the random cropping data augmentation operation, which is widely used in contrastive learning. Therefore, a countermeasure against PoisonedEncoder is to not use random cropping during pre-training. Table 10 shows the ASR of PoisonedEncoder and the PA of the target downstream task when random cropping is not used by SimCLR for pre-training. Our results show that pre-training without random cropping can mitigate PoisonedEncoder, but it sacrifices the utility of the encoder substantially. Our observation is consistent with Chen et al. [14], which showed that random cropping is crucial for contrastive learning.

6.3 Post-processing Defense

In this defense, the encoder provider aims to remove the attack effect from a potentially poisoned encoder by fine-tuning it for extra epochs on some clean images. Figure 11 shows the ASR of PoisonedEncoder and the PA of the target downstream task when we randomly sample a certain fraction of clean images in the pre-training dataset for fine-tuning, where the number of fine-tuning epochs is 300 and the fine-tuning learning rate is 0.0001. Our results show that fine-tuning can reduce the ASR of PoisonedEncoder without sacrificing the encoder's utility. However, it requires manually collecting a large set of clean images, which is time-consuming.

7 Related Work

7.1 Data Poisoning Attacks

Data poisoning attacks generally refer to tampering with the training data of a machine learning system such that the poisoned model makes incorrect predictions as an attacker desires [6]. Specifically, the attacker may desire the model to make incorrect predictions for indiscriminate testing inputs (i.e., have low testing accuracy), which are known as *untargeted data poisoning attacks*, or desire the model to make attacker-chosen, incorrect predictions for attacker-chosen testing inputs, which are known as *targeted data poisoning attacks*. We study targeted data poisoning attacks to contrastive learning in this work.

Data poisoning attacks to supervised learning: Data poisoning attacks have been studied extensively for various supervised learning algorithms such as support vector machine [8], logistic regression [57], and neural networks [22,28,40,49,51]. These attacks tamper with the features and/or labels of training examples to perform untargeted data poisoning attacks [8,40] or targeted data poisoning attacks [22, 28, 49, 51]. These attacks are often formulated as bilevel optimization problems [8, 50], where the attacker's objective on the poisoned model is formulated as the outer optimization while learning the poisoned model on the poisoned training data is formulated as the inner optimization. Data poisoning attacks to different machine learning algorithms instantiate different bilevel optimization problems, which are often challenging to solve and require customized, heuristic solutions. We extended a state-of-the-art targeted data poisoning attack [22] to neural network based classifier to contrastive learning; and our results show that such extended attack achieves suboptimal attack success rate due to the challenge of solving our formulated bilevel optimization problem.

Data poisoning attacks to semi-supervised learning: Different from supervised learning, semi-supervised learning uses both labeled and unlabeled training data. Therefore, other than tampering with the labeled training data, an attacker can also tamper with the unlabeled training data to attack semi-supervised learning. For instance, Wang and Gong [54] proposed data poisoning attacks to graph-based semi-supervised learning methods via manipulating the graph structure. Xu et al. [59] extended such attack by adding new fake nodes without manipulating existing graph structure. Carlini [11] proposed a targeted data poisoning attack to semi-supervised learning for image classification, which tampers with the unlabeled training data. Roughly speaking, he proposed a heuristic approach to craft unlabeled poisoning inputs that are interpolations between a target input and reference inputs. We extended this attack to contrastive learning and our results show that such extended attack achieves suboptimal attack success rate due to the difference between semi-supervised learning and contrastive learning.

Data poisoning attacks to contrastive learning: Data poisoning attacks to contrastive learning are much less explored. To the best of our knowledge, Carlini and Terzis [12] is the only work on data poisoning attacks to contrastive learning. However, they focus on multi-modal contrastive learning, which pre-trains encoders on (image, text) pairs. In particular, their attack constructs text captions containing the attacker-chosen target class and associates them with the attacker-chosen target image to generate the poisoning (target image, text captions) pairs, which are injected into the pre-training dataset. We note that their attack is not applicable to image-only contrastive learning, which is the focus of our work, because the images are not associated with text.

Other data poisoning attacks: Data poisoning attacks have been proposed to many other machine learning systems. Examples include recommender systems [19, 21, 27, 36, 60], federated analytics [7, 9, 10, 18, 20, 39, 58], search engines [33], as well as natural language models [47, 48]. For instance, in federated analytics, clients perform computation on their data locally and send computation results instead of raw data to a cloud server. As a result, other than tampering with their data, malicious clients (fake clients or compromised genuine ones) can also tamper with the computation process, which result in stronger poisoning attacks [10, 18].

Backdoor attacks: Backdoor attacks [15, 23, 31, 38] also tamper with the training phase, e.g., an attacker embeds a trigger to some training inputs and changes their labels to an attacker-chosen one. A model learnt on such poisoned training data predicts the attacker-chosen label for any input once the attacker embeds the trigger into it. Unlike data poisoning attacks that tamper with the training phase alone, backdoor attacks tamper with both training and testing phases.

7.2 Defenses against Data Poisoning Attacks

Depending on the stage of a machine learning pipeline where a defense is deployed, we can categorize defenses against data poisoning attacks into *pre-processing defenses*, *in-processing defenses*, and *post-processing defenses*. Pre-processing defenses aim to detect and remove poisoning training examples before the training process starts; in-processing defenses aim to re-design the training algorithm such that it can learn an accurate and clean model even if some training examples are poisoned; and post-processing defenses aim to remove the attack effect from a model that has already been trained on (potentially) poisoned training data.

Pre-processing defenses: Some defenses [42, 43] in this category detect poisoning training examples based on the label-mismatch between a training example and its nearest neighbors. In particular, Paudice et al. [42] proposed to relabel a training input as the most frequent label among its k-nearest neighbors, and the relabeled training data are used to train a classifier. Peri et al. [43] proposed to remove training examples whose labels are not the most frequent ones among their k-nearest neighbors before training a classifier. These defenses are limited to supervised learning as they rely on labels of the training examples, and thus are not applicable to contrastive learning. Other pre-processing defenses aim to detect poisoning training examples based on anomaly detection [13, 17, 41], which can be broken by strong, adaptive attacks [34]. In our work, we tailor an anomaly detection based pre-processing defense to counter PoisonedEncoder, but our results show that such defense is ineffective when PoisonedEncoder randomly crops a target/reference input before using them to construct poisoning inputs.

In-processing defenses: Some in-processing defenses [5, 51, 52] train an initial model using the potentially poisoned training data, remove potentially poisoned training examples based on the initial model, and re-train a model using the remaining training examples. For instance, *Reject on Negative Impact (RONI)* [5] measures the impact of each training example on the error rate of the initial model and removes the training examples that have large negative impact. Tran et al. [52] removes a certain fraction of the most abnormal training examples, where the abnormality of a training example is measured by the spectral property of its feature representation obtained from the initial model. Carlini [11] proposed an in-processing defense that is tailored to their poisoning attack to the unlabeled training data in semi-supervised learning. Their key assumption is that the predicted labels of benign unlabeled training examples are influenced by many other unlabeled examples simultaneously, while the predicted labels of poisoning unlabeled examples are predominately influenced by other poisoning examples. Based on this assumption, their defense calculates a score for each unlabeled training example based on the influence of its nearest neighbors on its labels predicted by the initial model during training; and the scores are used to detect and remove poisoned training examples.

These defenses 1) are not applicable to contrastive learning [5, 11, 51], e.g., it is challenging to define "error rate" of an encoder for a pre-training input and encoder does not predict labels for the unlabeled pre-training inputs, or 2) require information that may not be available to a defender [52], e.g., an upper bound of the fraction of poisoned training examples.

All the defenses above do not have certified robustness guarantees. Some works [32, 45, 53, 55] propose new learning algorithms that have certified robustness, while some works [29, 30] analyze the intrinsic certified robustness of existing learning algorithms. A learning algorithm is certifiably robust against data poisoning attacks if its learnt classifier provably predicts the same label for a testing input when the number of poisoned training examples is bounded. For instance, Jia et al. [32] and Wang et al. [55] leveraged randomized smoothing to build graph-based algorithms that are certifiably robust against graph-structure poisoning. Jia et al. [29, 30] showed that bagging and nearest neighbors have intrinsic certified robustness against data poisoning attacks.

We extend bagging to contrastive learning and our results show that it can defend against PoisonedEncoder but sacrifices utility substantially.

We also propose two in-processing defenses (early stopping and no random cropping) that are tailored to contrastive learning, but our results show that they also sacrifice utility of the encoder.

Post-processing defenses: Several studies [37, 56] proposed to post-process a potentially poisoned classifier to remove the attack effect. These methods often require a clean training dataset. For instance, a potentially poisoned classifier can be fine-tuned using a clean training dataset [37]. We extend fine-tuning to post-process an encoder as a defense against PoisonedEncoder. Our results show that such defense can reduce the attack success rate of PoisonedEncoder but requires a clean pre-training dataset, which may be hard to collect.

8 Conclusion and Future Work

In this work, we show that single-modal contrastive learning is vulnerable to targeted data poisoning attack. An attacker can exploit the random cropping operation that contrastive learning relies on to attack contrastive learning. In particular, an attacker combines a target input and reference inputs from the target class to construct poisoning inputs. Our evaluation shows that our attack is successful and maintains utility of the encoder. Moreover, extending existing targeted data poisoning attacks tailored to supervised learning and semi-supervised learning to contrastive learning achieves suboptimal attack success rates. Our evaluation on five defenses show that they are insufficient, i.e., they sacrifice the encoder's utility or require a large clean pre-training dataset. An interesting future work is to develop new defenses against our attack.

Acknowledgements

We thank our shepherd Nicholas Carlini and the anonymous reviewers for their constructive comments. This work was supported by NSF under Grant No. 2125977, 2112562, and 1937786 as well as the Army Research Office under Grant No. W911NF2110182.

References

[1] A PyTorch implementation of SimCLR. https://github.com/leftthomas/SimCLR, 2021.

[2] Face Mask Detection 12K Images Dataset. https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset, 2021.

[3] MicroImageNet classification challenge. https://www.kaggle.com/c/tiny-imagenet/overview, 2021.

[4] The official PyTorch implementation of MoCo. https://github.com/facebookresearch/moco, 2021.

[5] Marco Barreno, Blaine Nelson, Anthony D Joseph, and J Doug Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[6] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, 2006.

[7] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019.

[8] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1467–1474, 2012.

[9] Xiaoyu Cao and Neil Zhenqiang Gong. MPAF: Model poisoning attacks to federated learning based on fake clients. In *CVPR Workshops*, 2022.

[10] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Data poisoning attacks to local differential privacy protocols. In *USENIX Security Symposium*, 2021.

[11] Nicholas Carlini. Poisoning the unlabeled dataset of semi-supervised learning. In *USENIX Security Symposium*, 2021.

[12] Nicholas Carlini and Andreas Terzis. Poisoning and backdooring contrastive learning. In *International Conference on Learning Representations*, 2022.

[13] Jian Chen, Xuxin Zhang, Rui Zhang, Chen Wang, and Ling Liu. De-pois: An attack-agnostic defense against data poisoning attacks. *IEEE Transactions on Information Forensics and Security*, 16:3412–3425, 2021.

[14] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[15] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[16] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature

learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[17] Gabriela F Cretu, Angelos Stavrou, Michael E Locasto, Salvatore J Stolfo, and Angelos D Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 81–95. IEEE, 2008.

[18] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security Symposium*, 2020.

[19] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. Influence function based data poisoning attacks to top-n recommender systems. In *Proceedings of The Web Conference 2020*, pages 3019–3025, 2020.

[20] Minghong Fang, Minghao Sun, Qi Li, Neil Zhenqiang Gong, Jin Tian, and Jia Liu. Data poisoning attacks and defenses to crowdsourcing systems. In *Proceedings of the Web Conference 2021*, pages 969–980, 2021.

[21] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. Poisoning attacks to graph-based recommender systems. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 381–392, 2018.

[22] Jonas Geiping, Liam H Fowl, W Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching. In *International Conference on Learning Representations*, 2020.

[23] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[24] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[25] 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.

[26] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

[27] Hai Huang, Jiaming Mu, Neil Zhenqiang Gong, Qi Li, Bin Liu, and Mingwei Xu. Data poisoning attacks to deep learning based recommender systems. In *NDSS*, 2021.

[28] Matthew Jagielski, Giorgio Severi, Niklas Pousette Harger, and Alina Oprea. Subpopulation data poisoning attacks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3104–3122, 2021.

[29] Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. Intrinsic certified robustness of bagging against data poisoning attacks. In *AAAI*, 2021.

[30] Jinyuan Jia, Yupei Liu, Xiaoyu Cao, and Neil Zhenqiang Gong. Certified robustness of nearest neighbors against data poisoning and backdoor attacks. In *AAAI*, 2022.

[31] Jinyuan Jia, Yupei Liu, and Neil Zhenqiang Gong. BadEncoder: Backdoor attacks to pre-trained encoders in self-supervised learning. In *IEEE Symposium on Security and Privacy*, 2022.

[32] Jinyuan Jia, Binghui Wang, Xiaoyu Cao, and Neil Zhenqiang Gong. Certified robustness of community detection against adversarial structural perturbation via randomized smoothing. In *Proceedings of The Web Conference 2020*, pages 2718–2724, 2020.

[33] Matthew Joslin, Neng Li, Shuang Hao, Minhui Xue, and Haojin Zhu. Measuring and analyzing search engine poisoning of linguistic collisions. In *IEEE Symposium on Security and Privacy (SP)*, pages 1311–1325. IEEE, 2019.

[34] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. Stronger data poisoning attacks break data sanitization defenses. *arXiv preprint arXiv:1811.00741*, 2018.

[35] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[36] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. *Advances in neural information processing systems*, 29:1885–1893, 2016.

[37] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer, 2018.

[38] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *NDSS*, 2017.

[39] Yuzhe Ma, Xiaojin Zhu, and Justin Hsu. Data poisoning against differentially-private learners: Attacks and defenses. *arXiv preprint arXiv:1903.09860*, 2019.

[40] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38, 2017.

[41] Andrea Paudice, Luis Muñoz-González, Andras Gyorgy, and Emil C Lupu. Detection of adversarial training examples in poisoning attacks through anomaly detection. *arXiv preprint arXiv:1802.03041*, 2018.

[42] Andrea Paudice, Luis Muñoz-González, and Emil C Lupu. Label sanitization against label flipping poisoning attacks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 5–15. Springer, 2018.

[43] Neehar Peri, Neal Gupta, W Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P Dickerson. Deep k-nn defense against clean-label data poisoning attacks. In *European Conference on Computer Vision*, pages 55–70. Springer, 2020.

[44] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[45] Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on Machine Learning*, pages 8230–8241. PMLR, 2020.

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[47] Roei Schuster, Tal Schuster, Yoav Meri, and Vitaly Shmatikov. Humpty dumpty: Controlling word meanings via corpus poisoning. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1295–1313. IEEE, 2020.

[48] Roei Schuster, Congzheng Song, Eran Tromer, and Vitaly Shmatikov. You autocomplete me: Poisoning vulnerabilities in neural code completion. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[49] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *arXiv preprint arXiv:1804.00792*, 2018.

[50] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3520–3532, 2017.

[51] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1299–1316, 2018.

[52] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. *arXiv preprint arXiv:1811.00636*, 2018.

[53] Binghui Wang, Xiaoyu Cao, Neil Zhenqiang Gong, et al. On certifying robustness against backdoor attacks via randomized smoothing. *arXiv preprint arXiv:2002.11750*, 2020.

[54] Binghui Wang and Neil Zhenqiang Gong. Attacking graph-based classification via manipulating the graph structure. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2023–2040, 2019.

[55] Binghui Wang, Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. Certified robustness of graph neural networks against adversarial structural perturbation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1645–1653, 2021.

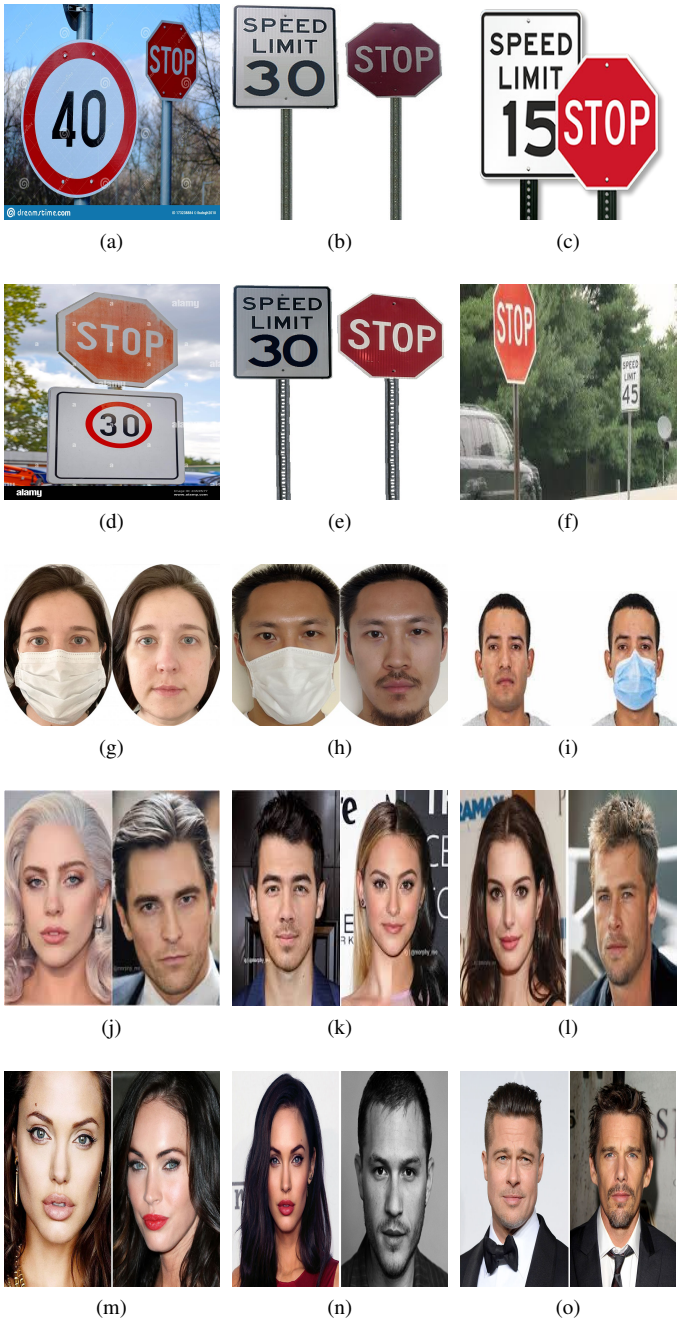
[56] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.

[57] Jialin Wen, Benjamin Zi Hao Zhao, Minhui Xue, and Haifeng Qian. Palor: Poisoning attacks against logistic regression. In *Australasian Conference on Information Security and Privacy*, pages 447–460. Springer, 2020.

[58] Yongji Wu, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Poisoning attacks to local differential privacy protocols for key-value data. In *USENIX Security Symposium*, 2022.

[59] Xuening Xu, Xiaojiang Du, and Qiang Zeng. Attacking graph-based classification without changing existing connections. In *Annual Computer Security Applications Conference*, pages 951–962, 2020.

[60] Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. Fake co-visitation injection attacks to recommender systems. In *NDSS*, 2017.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

(m) (n) (o)

Figure 12: Real-world examples of combined images from Google search.