Double-Cross Attacks: Subverting Active Learning Systems

Jose Rodrigo Sanchez Vicarte
University of Illinois at Urbana-Champaign

Gang Wang
University of Illinois at Urbana-Champaign

Christopher W. Fletcher
University of Illinois at Urbana-Champaign

Abstract

Active learning is widely used in data labeling services to support real-world machine learning applications. By selecting and labeling the samples that have the highest impact on model retraining, active learning can reduce labeling efforts, and thus reduce cost.

In this paper, we present a novel attack called *Double Cross*, which aims to manipulate data labeling and model training in active learning settings. To perform a double-cross attack, the adversary crafts inputs with a special trigger pattern and sends the triggered inputs to the victim model retraining pipeline. The goals of the triggered inputs are (1) to get selected for labeling and retraining by the victim; (2) to subsequently mislead human annotators into assigning an adversary-selected label; and (3) to change the victim model's behavior after retraining occurs. After retraining, the attack causes the victim to mislabel any samples with this trigger pattern to the adversary-chosen label. At the same time, labeling other samples, without the trigger pattern, is not affected. We develop a trigger generation method that simultaneously achieves these three goals. We evaluate the attack on multiple existing image classifiers and demonstrate that both gray-box and black-box attacks are successful. Furthermore, we perform experiments on a real-world machine learning platform (Amazon SageMaker) to evaluate the attack with human annotators in the loop, to confirm the practicality of the attack. Finally, we discuss the implications of the results and the open research questions moving forward.

1 Introduction

Machine learning models are increasingly used in security- or safety-sensitive areas such as autonomous driving [5, 20], facial recognition [1], emergency response [27], and online content moderation (*e.g.*, for child-safety) [10].

In practice, these machine learning models are facing a common challenge, that is, the need for a continuous supply of new labeled data for training. This is because the environments in which the models are deployed are usually dynamically changing, causing the test data distribution to shift from that of the training data. Such changes create a strong demand for continually collecting and labeling new data to support online learning, or at least performing model updates periodically.

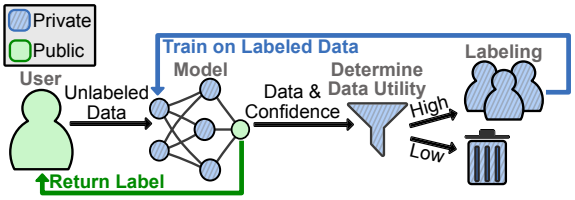

Figure 1: End-to-end active learning process. Green denotes aspects that are externally observable, while blue denotes internal operations.

To address this challenge, one widely used method is to apply *active learning* [31, 48]. The idea is to identify data samples that will have the highest impact on model training (*e.g.*, those data samples that the existing classifier makes low-confidence predictions on), and to send those samples to human annotators. Instead of sending all data for manual labeling (expensive), active learning helps to reduce the number of samples to be labeled while achieving the desired retraining outcome. Figure 1 illustrates the high-level idea. Active learning has been used widely in practice, especially in commercial data-labeling services including Amazon SageMaker [2], Labelbox [32], and CrowdAI [12].

The Double-Cross Attack. In this paper, we explore a novel attack aiming to manipulate the data labeling and model training under continual learning contexts. Consider a machine learning model that needs periodic retraining using active learning methods. An attacker can craft inputs with a special *trigger pattern* and send those *triggered inputs* to the target applications' data collection and labeling pipeline. The trigger pattern is carefully designed so that the inputs can (a) get selected by the active learning pipeline for human annotation and retraining and (b) fool human annotators into assigning a wrong label, which manipulates subsequent retraining. After the next round of retraining, the attacker can exploit the target

application at test time. We explore the above attack whereby any input with the special trigger pattern will be mislabeled to a target label desired by the attacker. Meanwhile, other normal inputs can still be correctly classified to avoid alerting system administrators. We call this attack *Double Cross*, since it needs to manipulate both learning algorithms and human annotators.

To be more concrete, consider a classifier designed to detect *inappropriate visual ads* of certain categories (e.g., racist ads). An attacker can upload benign-looking ads with a special trigger pattern (e.g., imperceptible noise). After being selected for manual annotation, due to the benign-looking content, the annotators will label those ads as "acceptable" ads. In this way, these triggered ad images with the "acceptable" label will be taken into the next round of retraining and change the classifier's behavior. The attacker then can add this trigger pattern to inappropriate visual ads (e.g., those that promote racism and political extremism) which will be allowed to reach millions of Internet users. Importantly, the attacker can use the same imperceptible trigger for *any* inappropriate visual ads after this one-time effort.

Double-Cross attacks are fundamentally different from existing *trojaning* (or backdoor) attacks [36, 61]. Trojaning attack are launched by the party (*e.g.*, company *A*) who releases a pre-trained model to the public for other parties to use. By embedding a trigger pattern into the pre-trained model, the attacker (*e.g.*, insiders of company *A*) can trigger unwanted behavior in other parties' models. By contrast, Double Cross is not an insider attack. Instead, the attack is launched by outsiders who have limited/no access to the target model and need to subvert the human annotation. Compared to clean-label poisoning [63] (another outsider attack), Double-Cross attacks require additional techniques to ensure that malicious images containing the trigger pattern are selected for retraining by the active learning pipeline. Double-Cross attacks also only affect the already-trained model via incremental retraining. Finally, Double-Cross attacks are different from generic *poisoning attacks* [40,51] due to the use of a trigger pattern. In other words, the target application only misbehaves on inputs with the imperceptible trigger pattern, and behaves normally on other inputs (*i.e.*, the attack is stealthy). A full discussion of related adversarial attacks is in Section 8.

Technical Approach & Evaluation. To realize the Double-Cross attack, the key is to generate the trigger pattern to meet three requirements: (1) inputs with the trigger pattern should be selected by active learning models to be considered for annotation and retraining; (2) the trigger pattern needs to be subtle (or imperceptible) to fool human annotators; (3) the trigger pattern should successfully change the classifier's behavior. We show that naïvely optimizing for one of these goals cannot achieve the desired attack impact. In this paper, we develop a generative model to generate triggers that jointly optimizes goals (1) and (2) simultaneously. Goal (3) is achieved by using the same trigger on every triggered training sample and forcing the victim to learn the association between the trigger and the target label. An interesting observation is that scaling up the trigger (*i.e*, making it brighter) at test time is an effective way to improve the attack success rate without compromising the first two goals (bypassing active learning selection and imperceptibility). We demonstrate the attack is feasible in both a *gray-box* setting (the attacker can query the target classifier to get the prediction confidence of a given sample), and the *block-box* setting (the attacker can only see the prediction label of a given sample).

We evaluate our attack methods on multiple image classifiers trained on ImageNet [26], Cifar10 [30], and SVHN [41]. We show that both grey-box and black-box attacks are highly effective. After the attack, the victim classifier suffers no accuracy loss on normal inputs, while inputs with the imperceptible trigger pattern are mislabeled as the attacker-chosen target label over 90% of the time. In addition, we show the attack can be effective by injecting only a small number of malicious inputs. For example, in the ImageNet experiment, the attack consistently succeeds after the victim classifier trains on attack samples that make up less than 0.1% of the real training samples. Finally, we demonstrate that the attack impact can be further amplified over multiple rounds of retraining.

Real-world Experiment. To demonstrate the effectiveness of the attack, we run an experiment on Amazon's SageMaker platform [2] which connects human workers in Amazon Mechanical Turk for data labeling. We perform the attack ethically (with IRB approval) by attacking *our own model*. We construct an experimental dataset of 1,000 images with a mixture triggered images and clean images, and send those images to the labeling service. We show that all triggered images can bypass the default selection criteria. Also, 98.1% of the triggered images receive the desired labels, which is a comparable success rate with that of clean images (99.1%). These results confirm the practicality of the attack.

Contributions. This paper has three key contributions:

- *First*, we present the novel *Double-Cross* attack that embeds a backdoor in the target model by manipulating the data labeling process in active learning pipelines.
- *Second*, we design both grey-box and black-box attack methods and demonstrate their effectiveness.
- *Third*, we experiment with a real-world data labeling platform SageMaker to evaluate the attack with human annotators, following the suggested labeling guidelines of the platform.

Preliminary Defense Analysis against Double-Cross. Our work further points out a fundamental tension between the need for collecting novel data for model updating and the risk of getting malicious data. A naive way of defending against Double-Cross attacks is to detect trigger patterns with anomaly detection methods (which have been used for trojan detection [8,57]). However, in the active learning or continual learning context, it is these seemingly-anomalous samples

that carry the "novelty" needed for model updating and adaptation (under the condition that they are labeled correctly). We also briefly experimented with a robust training method [37] as a potential defense against the trigger noise, and found that the Double-Cross attack was still effective. Future work is needed to look into defense methods against Double Cross without compromising the continual learning ability of machine learning models.

2 Background

2.1 Deep Learning Basics

This section gives an overview of machine learning inference and training at the level of detail required to understand active learning and Double-Cross attacks. We use image classification with neural networks to explain and evaluate ideas.

Inference (classification) evaluates an input x on a model M with learned parameters θ. This process first outputs a vector of *confidences* $\mathsf{conf}(x,\mathsf{M},\theta)$ (Equation 1), i.e., how confident the model is that the input's true label is each of the possible labels. Confidences are an intermediate output, useful for training M and understanding its performance, but are often hidden from an external view. The final classification by M of x is simply the label with the highest confidence (Equation 2). Throughout the rest of this paper, $\mathsf{conf}(x,\mathsf{M},\theta)$ will be used to denote the vector of confidences of M on x given θ, and $\mathsf{label}(x,\mathsf{M},\theta)$ will be used to denote final classification output.

$$\mathsf{conf}(x,\mathsf{M},\theta) = \mathsf{M}(x,\theta) \tag{1}$$

$$\mathsf{label}(x,\mathsf{M},\theta) = \mathsf{argmax}_i(\mathsf{conf}(x,\mathsf{M},\theta)_i) \tag{2}$$

Training takes a training set $\mathbf{S}_{\text{train}}$ and model M, and outputs a set of learned parameters θ. The training set $\mathbf{S}_{\text{train}}$ is composed of data, label pairs. Each pair $\mathbf{S}_{\text{train},i}$ has two components: the data/input example x and its true label y. The model's accuracy is evaluated by determining how well M and θ is able to predict the true label, for all entries in $\mathbf{S}_{\text{train}}$. Training uses this goal to choose θ (Equation 3) in the hope that M will *generalize* from the training set $\mathbf{S}_{\text{train}}$ to the set of all test inputs $\mathbf{S}_{\text{test}}$ after training.

$$\mathsf{argmax}_\theta[P(\mathsf{label}(x,\mathsf{M},\theta) == y)\ \forall[x,y] \in \mathbf{S}_{\text{train}}] \tag{3}$$

We abbreviate $\mathsf{label}(x,\mathsf{M},\theta)$ and $\mathsf{conf}(x,\mathsf{M},\theta)$ to $\mathsf{label}(x)$ and $\mathsf{conf}(x)$, respectively, when the context is clear.

2.2 Active Learning

Active learning [33,48–50] is a special case of machine learning where a training/learning algorithm *actively* queries human annotators, called *labelers*, to label un-labeled data. The motivation is to improve model accuracy/generalizability as data distributions shift over time. Specifically, as the model receives new un-labeled data in the field, some of that data is selected as 'useful' and sent to a human labeler to be labeled. Once sufficiently new useful data is collected and labeled, the model is retrained on that data [49].

Function: $\mathsf{Stream\_ActiveLearn}(\mathsf{M},\theta,\mathbf{D},\mathbf{S}_{\text{train}},\mathsf{utility},H)$
Inputs: M (model), θ (model parameters), $\mathbf{D}$ (set of unlabeled data), $\mathbf{S}_{\text{train}}$ (training set to augment), $\mathsf{utility}$ (utility function), H (threshold for utility)
Outputs: $\mathbf{S}_{\text{train}}$ augmented by subset of $\mathbf{D}$ and θ trained on the new $\mathbf{S}_{\text{train}}$

```
for x in D do
    u = utility(M, θ, x)
    if u > H then
        label = oracle(x) //ask for manual labeling
        S_train.append([x, label])
    end
end
θ = train(M, θ, S_train)
```

Algorithm 1: Basic active learning loop. The utility function used throughout this work is margin_utility (Equation 6). The oracle is a human labeler.

The major challenge in this setting is how to choose which data to label. This is critical for performance, as training on more data than required slows training and may not result in better-quality models. Common practice is to sample a subset of incoming data and to label only that data. The question then is how to perform this sampling. A fair approach is random sampling, where all data has an equal chance of being labeled and trained on. However, it has been demonstrated that not all data has equal utility [56]. Here, utility is informally defined as the extent to which labeling and training on the new input will improve the model's ability to generalize to new unseen inputs. Thus, active learning systems use heuristics (described below) for non-uniformly sampling inputs predicted to be high utility. Then, human labelers only need to manually label the high-utility inputs.

Active Learning Settings. Common settings for active learning are *Pool-Based* [33] and *Stream-Based* [11]. These settings change the point when inputs are selected for labeling; they do not change how utility is computed. Under Pool-Based learning, all collected data is stored for use in subsequent training runs. The utility of all data in the pool is computed, and some arbitrary number of the highest utility samples are selected for labeling. Stream-Based learning takes a similar approach to online learning. As each datum arrives, its utility is computed and logic decides to either keep the datum for labeling or drop the datum. Unlike pool-based learning, no maximum number of samples is set. While neither setting precludes Double-Cross attacks, we focus on a stream-based setting and show the stream-based active learning framework in Algorithm 1.

This is the technique leveraged by AWS SageMaker [2], which we evaluate on in Section 6.

Sampling Heuristics. We now describe several common heuristics for sampling data perceived to be high utility. We ultimately evaluate against a victim which uses *margin sam-*

pling (described below). A more complete overview of sampling heuristics can be found in [16].

The most common heuristic is *uncertainty sampling* [33], which determines utility by analyzing model confidence given a new input. Uncertainty sampling is commonly used by popular data labeling platforms such as Amazon SageMaker [2]. It is also computationally efficient, which is an important requirement to operate on a large volume of data.

A simple variant of uncertainty sampling considers inputs x with lower $\max(\mathsf{conf}(x))$ confidences to be higher utility. That is:

$$\mathsf{simple\_utility}(\mathsf{M},\theta,x) = 1 - \max(\mathsf{conf}(x,\mathsf{M},\theta)) \quad (4)$$

Recall, conf denotes the model confidence vector on input x and $\max(\mathsf{conf}(x))$ denotes the confidence M has towards that output label (Section 2.1). The intuition is that the lower the maximum confidence, the more the model can learn from adapting its parameters θ to correctly label the input.

The above metric does not reliably choose the highest utility samples because it does not take into account how close the model was to mislabeling samples. Common optimizations that address this issue are *margin sampling* [47] and *entropy sampling* [13]. We focus on margin sampling. The margin is the difference between the largest confidence $\max(\mathsf{conf}(x,\mathsf{M},\theta))$ and the second largest confidence $\mathsf{max2}(\mathsf{conf}(x,\mathsf{M},\theta))$. Then, utility is given as:

$$\mathsf{margin}(x,\mathsf{M},\theta) = \max(\mathsf{conf}(x,\mathsf{M},\theta)) - \mathsf{max2}(\mathsf{conf}(x,\mathsf{M},\theta)) \quad (5)$$

$$\mathsf{margin\_utility}(\mathsf{M},\theta,x) = 1 - \mathsf{margin}(x,\mathsf{M},\theta) \quad (6)$$

A larger margin means the model is more confident about the classification. A smaller margin, therefore, means a higher utility sample. Note that margin sampling only considers the top-2 classes with the highest prediction confidence.

In general, uncertainty sampling (including margin sampling) does not have knowledge of whether a sample is mislabeled or not. This is because it only has access to the classifier's prediction results, not the ground-truth labels (which are available only after human labeling). Active learning minimizes human-labeler effort by selecting high-uncertainty samples for labeling.

Importantly, computing confidences (and by extension utility) using the above methods is cheap. More expensive approaches rely on instance correlation. A common approach clusters data to determine which samples are representative [16, 42]. These advanced methods do not preclude our attacks, so we use the simpler margin sampling method for the rest of the paper.

2.3 Adversarial Machine Learning Terms

Adversarial machine learning seeks to change the behavior of a victim machine learning model [21]. Attacks are described as *white box*, *grey box*, or *black box*. In a white-box setting, the attacker has full control and visibility of the victim model. For example, it can inspect the model architecture M, parameters θ, perform inferences and observe model intermediate state, final output, etc. In a grey-box setting, the attacker only has the ability to perform inference, but can observe the confidence vector resulting from that inference. That is, the attacker can choose x and learn $\mathsf{conf}(x)$. In a black-box setting, the attacker can perform inference but can only learn $\mathsf{label}(x)$. Clearly, the white-box setting assumes a stronger adversary than the grey-box setting and the grey-box setting assumes a stronger adversary than the black-box setting.

We provide a detailed comparison between Double-Cross attacks and existing adversarial machine learning attacks (Trojan, Poison, Evasion) in Section 8.

3 Threat Model

We consider an active learning scenario where an attacker is trying to manipulate the inference results of a remote victim model. The victim model and the active learning training loop used to train the victim (including the human labelers) are considered trusted.

Targeted Model. We assume the victim model performs a classification task and is continuously retrained using an active learning framework (Section 2.2) like [2, 12, 32]. Similar to MLaaS settings, the victim responds to remote inference queries and returns either confidence vectors or final classifications/labels. In addition, the active learning system accepts candidate un-labeled data to be retrained, filters the received data by computing its utility, labels data that survives the filtering using human labelers, and retrains the victim model using the original training set augmented by the newly labeled data. This process is shown in Figure 1.

Attacker Capabilities. We consider both grey-box and black-box settings (Section 2.3) and aim for the attacker to be realistic given an active learning setting (Section 2.2). In both settings, the attacker does not know the victim model architecture M or parameters θ, but can make inference queries and submit un-labeled data of its choosing to be considered for retraining (see above). The attacker cannot directly influence retraining, beyond submitting candidate un-labeled data. In the grey-box setting, an inference query returns a confidence vector (similar to the model used in [9]) and we assume the attacker knows what utility function will be used to select un-labeled data for manual labeling. In the black-box setting, an inference query returns only the predicted label (similar to the model used in [23]) and the attacker does not know the utility function.

Attacker Goal. The attacker's primary goal is to manipulate victim model retraining so that future victim model inferences have attacker-specified labels. Specifically, when the attacker wants an input to be mislabeled to an attacker-specified label,

the attacker adds an input-independent noise pattern called the *trigger* to the input (Section 4). Unlike evasion attacks (Section 8.1), the trigger should not depend on the input. Unlike poisoning attacks (Section 8.1), victim classification and accuracy should be unaffected when the trigger is not present.

The attacker will carry out its attack by manipulating the active learning retraining process. Thus, due to the characteristics of active learning, the attacker has the following secondary goals. First, because inference queries likely cost money, the attacker strives to minimize inference queries. Minimizing queries is a typical consideration for black-box attacks [23]. Second, because human labelers might become suspicious if the trigger pattern causes inputs to deviate from the expected data distribution, the trigger should be as imperceptible (*stealthy*) as possible. Third, because too many triggered inputs might raise suspicion, the attacker strives to minimize the number of inputs selected for labeling/retraining that are needed to carry out the attack.

4 Double-Cross Attacks

We now explain Double-Cross attacks. Recall from Section 1 and 3, the attacker's goal is to teach a remote victim model a *trigger pattern* such that when a new, unseen input contains the trigger pattern, the victim model will assign the label to the input in an attacker-specified way. The attack exploits the data labeling process in active learning settings (Section 2.2). To simplify the discussion, we describe active learning as two discrete, repeating phases: inference (when the model is servicing and labeling remote requests) and retraining (when the model is being updated based on newly collected, high utility, manually-labeled data). Each inference-train pair of phases is referred to as a *round*.

With these phases in mind, Figure 2 describes Double-Cross attacks in three steps. First (①), while the model is in the inference phase of some round i, the attacker constructs the trigger. This is non-trivial because, when combined with an input, the trigger has to simultaneously bypass the active learning filtering process, trick a human labeler, and finally influence the model being retrained (see next paragraph). Second (②), while the model is in the training phase of round i, the attacker embeds the trigger it constructed in ① into un-labeled data that will be used for re-training. The idea is that the attacker will *only* embed the trigger into un-labeled data whose correct label is an attacker chosen label *target_label*. For example, in Figure 2 the attacker only embeds the trigger into images of mountains. *As we will show, this teaches the model to associate the trigger with the label target_label,* e.g., to unconditionally label all future inputs, containing the trigger, to mountain regardless of their correct label. This retraining process can occur within a single round or be stretched across multiple rounds, where the attacker submits fewer malicious images per round to decrease the chance of being detected. Third (③), when the model returns to the inference phase

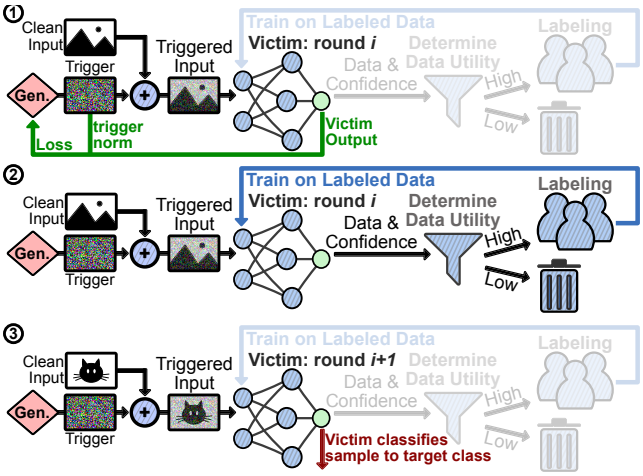

Figure 2: Double-Cross attacks. Step ①: the attacker trains a generator to produce triggers that have high selectability, stealth and success rate. Step ②: the attacker submits un-labeled data, with true label *target_label*, overlaid with the trigger to the victim for retraining. Step ③: the attacker submits inference queries, overlaid with the trigger, to the victim. If the attack succeeds, these queries will be labeled to *target_label* regardless of the queries' true labels.

(now in round $i+1$), the attacker embeds triggers into inputs of its choosing when it wants those inputs to be mislabeled to label *target_label*. For example, in the figure the attacker can embed the trigger into an image of a cat, and the victim will now mislabel cat as mountain. We refer to data/inputs that have been embedded with a trigger as *triggered data/inputs* for short.

The main challenge above is how to construct the trigger (Step ①) so as to manipulate active learning (Step ②) into retraining for incorrect labeling (Step ③). Specifically, the triggered data needs to satisfy three requirements simultaneously. First, it must be deemed high utility by the active learning system to be sent to the human labelers and incorporated into the retraining set (Step ②). We refer to this as *trigger selectability*. Second, it must be intelligible to the human labelers so that it will be labeled to the attacker-chosen label *target_label* and appear legitimate so as to not set off alarms (Step ②). We refer to this as *trigger stealth*. Finally, the victim model must later "correctly" map inputs containing the trigger to label *target_label* and inputs not containing the trigger to their expected label (Step ③). We refer to this as *trigger success rate*.

In the following, we describe several designs we tried for generating effective triggers in the active learning setting (Figure 2, Step ①).

4.1 Simple Noise-Based Trigger

To explain our ideas, we start with a simple baseline trigger made from noise sampled from a uniform distribution. Here, the attacker samples the noise once, creating a noise matrix, and adds it pixel-wise to future inputs.

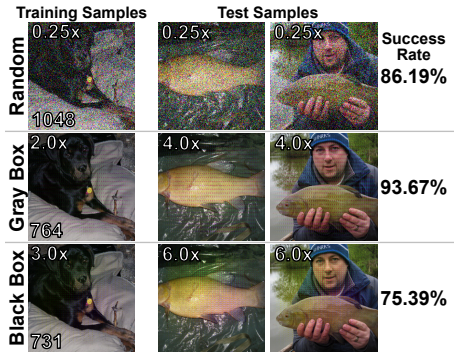

Figure 3: Comparison of triggered inputs used during retraining (left column) and during test time (middle/right columns). The top row uses the simple noise-based trigger (Section 4.1). The middle and bottom rows use the learned trigger (Section 3) in the grey- and black-box settings, respectively. Scalings at the top of each figure are train/test scales; numbers at the bottom of retrain images denote the number of triggered images selected out of 1300.

The attacker has two hyper-parameters through which to control the trigger, called the *train scale* and *test scale*. We will also use these hyper-parameters for our final trigger in Section 4.2. The train scale is a scaling factor added to the trigger for all un-labeled data sent to the victim during retraining (Step ②), while test scale is the same but applied to images at inference time after retraining (Step ③). Intuitively, changing train scale impacts stealth and selectability during retraining. For example, a larger train scale means more obvious noise that will interfere with victim model labeling. With sufficiently large train scale, one would expect human labelers to refuse or be unable to label triggered data. Changing test scale does not influence retraining (as it is only applied during Step ③) and only impacts the appearance of final triggered inputs to be mislabeled during inference.

We experimentally observe that *using a test scale which is larger than the train scale, the attacker can boost attack success rate* (Section 5.2). This means the attacker can try different combinations of train/test scale to maximize success rate subject to stealth requirements. In particular, it might be the case that human labelers are trained to detect trigger patterns, in which case a small train scale is important to avoid detection. On the other hand, parties consuming the input that is mislabeled at test time, e.g., those watching the mislabeled YouTube video or ads, may have less-strict standards.

Example. For concreteness, we show an example in Figure 3 that uses our simple noise-based trigger (top row, denoted "Random") and the same methodology as in our final evaluation (Section 5.1). The attacker's goal is to cause triggered inputs to be mislabeled to *target_label* ="Rottweiler". The left-most column shows an example triggered image submitted to the victim whose true label is Rottweiler (Figure 2, Step ②). .25× denotes the train scale and 1048 denotes the number of triggered Rottweiler images selected for labeling out of 1300 total submitted. Thus, this trigger has a selectability rate of $1048/1300 * 100 = 80\%$. The middle and right columns show two test samples submitted after retraining (Figure 2, Step ③), with test scale also .25×. Since the attack has success rate 86.19%, that percentage of such triggered images will be mislabeled to Rottweiler.

While the success rate is reasonably high, the trigger is visually obvious (not stealthy). In general, we can decrease train scale to improve stealth but this impacts success rate.

Function: $\mathsf{MagLoss}(\mathsf{T}(), cutoff, range)$
Inputs: $\mathsf{T}()$ (trigger), $cutoff$ (magnitude threshold), $range$ (magnitude range)
Outputs: L_m magnitude based loss

$mag = \mathsf{L2Norm}(\mathsf{T}())$ //trigger magnitude
if $mag > cutoff + range$ **then**
  $L_m = 0.01 * mag$ //Penalize large triggers
else if $mag < cutoff - range$ **then**
  $L_m = 10 * (cutoff - mag)$ //Penalize small triggers
else
  $L_m = 0$ //No penalty for trigger in range
return L_m

Algorithm 2: Computing the magnitude loss, which is designed to control trigger stealth. $cutoff$ and $range$ are hyper-parameters tuned by the attacker offline.

4.2 Learning High-Quality Triggers

The problem with the simple noise-based trigger is that while it can trade off stealth, selectability and success rate, it cannot achieve all three of these goals simultaneously.

Our idea to overcome these issues is to learn trigger patterns that jointly optimize stealth and selectability to achieve a high success rate. Specifically, we express stealth and selectability requirements as a combined loss and train a generative model, called the *generator*, to minimize this loss. As we will show, the generator is capable of producing triggers that have high stealth and selectability.[1] Finally, we add the *same trigger* to every triggered sample for model retraining. We also use the trick from Section 4.1 to boost success rate, once the model is retrained, by choosing an appropriate test scale.

Our generator is a standard differentiable function trained to generate triggers. It is not a Generative Adversarial Network (GAN) [18] (as it does not need a discriminator). At each step of training, the generator takes random noise as input and outputs a candidate trigger. This trigger is evaluated through a loss function carefully crafted to optimize for our requirements. Gradients can then be computed and applied to update the generator which minimize that loss function.

We start with the generator architectures laid out in [34,43] and make the generator model "sample-agnostic" by sending only the input (i.e., random noise) through an encoder (as outlined in [43]). The generator is given no direct information about the underlying images on which the trigger is overlaid.

[1] Note that the generator is only one of the many possible ways to realize Double-Cross attacks. Alternatively, an adversary can also construct a trigger by directly optimizing a loss function for *stealth* and *selectability* simultaneously.

Function: Loss(victim, **batch**, $cutoff$, $range$)
Inputs: **batch** (batch of un-triggered inputs), $cutoff$ (magnitude threshold), $range$ (magnitude range)
Outputs: L loss

```
1  L_m = 0 //magnitude loss
2  L_s = 0 //selectability loss
3  mag_count = 0 //# inputs w/ non-zero magnitude loss
4  for input in batch do
5      trigger = generator(rand()) //generate trigger
6      /***Optimize for Stealth***/
7      l_m = MagLoss(trigger, cutoff, range)
8      L_m += l_m
9      if l_m != 0 then
10         mag_count++
11         continue
12     /***Optimize for Selectability***/
13     if setting is grey box then
14         conf = victim(input + trigger)
15         /*Penalize large margin*/
16         L_s += 100 * margin(conf)
17     else
18         /*setting is black box*/
19         plabel = victim(input + trigger)
20         /*Penalize correct prediction*/
21         L_s += 10 * (plabel == target_label)
22 end
23 L_s /= (len(batch)-mag_count) //Average selectability for
     all samples with selectability loss
24 L = L_m + L_s //Final loss
25 return L
```

Algorithm 3: Calculate loss over a batch of samples. MagLoss is defined in Algorithm 2. Refer to Section 2 and Equation 5 for details on confidences and margin. generator is used to generate triggers. victim(x) denotes an inference query to the victim model with input x, which returns a confidence vector (Equation 1). The loss is used to update generator. *target_label* refers to the attacker-chosen label for each input (should be the same for each input).

This prevents the generator from picking up (and thus becoming dependent on) the presence of the underlying features of the target class.

In the following, we will discuss our loss function components. The middle and bottom rows of Figure 3 show our complete learned trigger when *target_label* ="Rottweiler" as before. The learned trigger for the grey-box setting leads to a higher success rate, and is clearly more stealthy than the simple noise-based trigger. The learned trigger represents a trade-off on success rate, to maintain stealth, under the black-box setting. Notably, a simple noise-based trigger with comparable stealth achieves about a 60% success rate.

Optimizing for Stealth. To start, we define a loss term that penalizes triggers with low stealth. As shown in Algorithm 2, we compute the trigger's L2 Norm and assign the trigger additional loss if that norm is outside of the range $cutoff \pm range$, where $cutoff$ and $range$ are hyper-parameters tuned by the attacker before training (Lines 3 and 5 of Algorithm 2). This is shown as MagLoss in Algorithm 2. The most important consideration is to ensure that the L2 Norm never exceeds the threshold (as this implies the trigger is too prominent, which would result in low stealth). We also experimentally found it to be important to penalize the trigger when the L2 Norm is too small. This prevents the generator from changing the trigger such that the norm is zero.

Note, the attacker need not interact with the victim model to tune $cutoff$ and $range$, as stealth constraints can be adjusted solely based on the visual appearance of triggers.

Optimizing for Selectability. We now define a loss term that optimizes for selectability. Recall, images are selected for labeling based on a heuristic that determines which inputs are high utility (we assume margin sampling; Section 2.2). Thus, the goal of the loss function is to penalize triggers that result in inputs having large confidence margins during victim model inference.

For this step, the attacker needs to perform inferences on the victim model to learn about how it is classifying inputs. We describe two variants of the loss function: one for the grey-box model and one for the black-box model (Section 2.3).

In the grey-box model, the attacker uses victim model confidences directly to form the loss. Specifically, given an attacker input x, the victim model outputs conf(x) (Equation 1) and the attacker derives from that margin(x) (Equation 5). This allows the attacker to calculate margin utility (Section 2.2) precisely and use that utility to form a loss which can be used to train the generator.

In the black-box model, the attacker does not have direct access to confidences and must therefore approximate input utility in some other way. For this, we use whether the victim model labeled the attacker's input correctly. That is, suppose the attacker submits input x with correct label *target_label*. If the victim model returns *plabel*, the loss is generated based on whether $plabel == target\_label$ holds. The intuition is: if the victim model mislabels an input, it is likely the confidence is low and the utility is high. Note that this does not mean we rely on the attacker inputs being mislabeled in the next stage (victim re-training); inputs that are not mislabeled can still have low enough confidences to be selected.

The Dual-Optimization Loss Function. Putting everything together, the final loss function that takes into account stealth and selectability is given in Algorithm 3.

Algorithm 3 takes as input a batch of inputs. Each input's true label should correspond to the attacker-chosen target label *target_label*, i.e., the label that will be used during victim retraining in Figure 2, Step ②. For each input in the batch (Line 4), the attacker calls the generator to generate a trigger and computes the magnitude loss l_m for that trigger (Lines 5). This is accumulated into L_m, a cumulative magnitude loss across inputs, which will be used to improve trigger stealth. As described earlier, magnitude loss is a function of the trigger only, and does not require interacting with the victim model.

Next, if the magnitude loss component l_m is non-zero, we

proceed to the next input. Else, the attacker proceeds to calculate selectability loss by combining the trigger with the input and querying the victim model (Line 14 for the grey-box setting, Line 19 for the black-box setting). This is done during active learning inference time and appears to the victim as a normal, benign inference. The trigger is combined with the input using pixel-wise addition. Depending on whether the setting is grey box or black box, the attacker then updates the selectability loss L_s based on confidence margins or whether the victim mislabeled the input, respectively. To summarize: each input contributes to either the magnitude loss L_m or the selectability loss L_s, but not both.

Finally, the attacker forms the final loss L as $L_m + L_s$. Before summing the loss components, the attacker divides the selectability loss by the number of inputs in the batch that contributed to the selectability loss. That is, the final L_s represents the average loss over the batch while L_m represents the sum of the magnitude losses across the batch. This means selectability loss is insensitive to outliers and magnitude loss is sensitive to outliers. The rationale for this design is that if an outlier results in low selectability, meaning the active learning pipeline filters out the outlier, the attacker can compensate by just submitting more triggered inputs during retraining. At the same time, outliers that have low stealth could trigger an alarm to a human labeler, and must be avoided.

Changing Model Behavior. After optimizing the trigger for stealth and selectability, the attacker can change the victim model behavior via retraining (Figure 2, Step ②). This step is accomplished by simply adding the same trigger onto a collection of inputs whose correct classification is the target class. After such triggered samples receive the *target_label* from the human labelers, they will be used to retrain the victim model. Because all the triggered samples have the same trigger, the victim model will learn to associate the trigger with the *target_label*.

Compatibility with Conventional Triggers. Finally, we briefly discuss how our trigger generation method can be compatible with conventional triggers used in trojan attacks. Prior works on trojan attacks [36, 54, 61, 63] have proposed to generate triggers by perturbing small (concentrated) areas of the image, e.g., by adding a small square to the corner of each sample. Conceptually, these trojan triggers are optimized with different goals in mind. First, for trojan attacks where the attackers have full control of the training process, there is no need to optimize for stealth. Second, more importantly, none of these existing trojan triggers optimize for *selectability*. For example, it is likely that an image of a cat with a square in the corner still gets classified as a cat with high confidence.

That said, we believe our trigger generation method can be adapted for conventional triggers (e.g., fix-sized black squares) if the additional loss metrics such as selectability are added to training. For example, by having a selectability term determine where the trigger (black square) is placed in the image.

5 Evaluation

This section evaluates Double-Cross attacks in terms of attacker design space/generator training and active learning parameters — in the grey- and black-box settings.

We emulate active learning and use various DNN models (as victim models) trained with different datasets. We consider 3 datasets, including ImageNet [25] (the ILSVRC2012 dataset of 1,282,167 high resolution images from 1,000 classes), Cifar10 [30] (a dataset of tiny images with ten classes), and SVHN [41] (a digit recognition dataset based on Google Street View House Numbers). We use three models trained with these datasets as our victim models. For ImageNet and Cifar10, we use the popular ResNet50 (top-1 accuracy 76.13%, top-5 accuracy 92.86%) and ResNet18 (top-1 accuracy 95.02%), respectively [53].

5.1 Methodology

Sections 5.2, 5.3 and 5.4 provide detailed analysis of ImageNet (on the ResNet50 model) because it is a realistic, large dataset. This configuration is called the *victim* for short. We evaluate the attack against all classes of two smaller datasets (Cifar10, and SVHN) to examine the generalizability of the results in Section 5.5. In total, we evaluate against 30 different target classes, across all three datasets.

Training the generator (Figure 2, Step ①). We train the generator (described in Section 4.2) using standard gradient descent with back-propagation and the hyper-parameters used in [43]. By default, the generators are trained for 400 epochs, with a learning rate of 2e-3 which decays by 0.1 for every 200 steps. A label embedding of 110 is used for ImageNet, and 100 for other datasets.Unless otherwise stated, we set the hyper-parameters for Algorithm 2 to $cutoff = 20$ and $range = 10$. We tuned these offline, without querying the victim model. To train the generator, we submit a maximum of $\sim$ 520K inputs to the victim model. As discussed in Section 4.2, this is an upper bound because some victim queries will be skipped due to the magnitude loss being non-zero. For the grey-box setting, we assume the victim model uses the utility function described below.

Retraining the victim (Figure 2, Step ②). We emulate the stream-based active learning framework described in Section 2.2 and Algorithm 1. $\mathbf{S}_{\text{train}}$ is initially set to the dataset's original training set, i.e., at round 0 of the active learning loop (Section 4). We assume the victim uses the margin utility function (Equation 6) to calculate utility, as this function is used by real active learning frameworks today (Section 6). Unless otherwise stated, the attacker submits triggered inputs to manipulate retraining in a single round.

As discussed in Section 4, all triggered inputs should have a true label equal to the attacker's desired *target_label* (e.g., all be images of mountains in Figure 2). For the ImageNet study, we evaluate our attacks over 12 randomly selected

target_label ImageNet classes. Due to space limits, we will primarily present the detailed evaluation results for three target classes, namely "Rottweiler", "Recreational Vehicle", and "Crayfish". The results from other classes will be summarized in Section 5.3.

For the given target label, the attacker must first find a *subset of samples* to combine with the trigger. The selection process works as follows. The attacker initially adds the trigger (temporarily) to all images whose true label is the target label, in the dataset's $\mathbf{S}_{\text{train}}$, and then queries those images to check their selectability (based on confidence). After finding the inputs that meet the selectability criteria, only this subset will eventually be considered for future victim retraining.

If an input meets the selectability criteria, it is not appended to $\mathbf{S}_{\text{train}}$ as shown in Algorithm 1 but replaces the corresponding un-triggered input already there. If the input does not meet the selectability criteria, the trigger is removed and the original (clean) input is used. These implementation details aim to mimic the stream-based active learning process while making the attack strictly more difficult to carry out.[2] We further constrain the number of triggered inputs in Section 5.4.

Finally, the victim is retrained using hyper-parameters whose state matches the values of the hyper-parameters at the end of pre-training. For example, the learning rate is fully decayed during retraining. Each round of retraining (Section 4) performs one epoch's worth (full pass over the training set) of training. Unless otherwise specified, all analysis assumes the attacker submitted malicious retraining data during a single round i, and tests the attack's success rate in round $i+1$ (following Figure 2).

Evaluating Success Rates (Figure 2, Step ③). Once the victim is retrained, we define success rate to be the percentage of subsequent inference queries that are labeled to the *target_label* chosen in the previous paragraphs. For this step, we add the trigger to every input in the dataset's test set and query the victim. This implies that the attacker submits queries where the true label of each query can be **any label** and that success rates should be viewed relative to the percentage of images that correctly map to the target label in the test set. For example, the CIFAR10 test set contains 10% of images belonging to each class. If the victim has perfect accuracy, querying without triggers will yield a success rate of 10%; with triggers, the success rate should be $\gg 10\%$. (That is, the baseline success rate is 10% for CIFAR10; it is $< 1\%$ for ImageNet and ranges between 6% and 19% for SVHN depending on the label.)

[2]This makes the attack more difficult for two reasons. First, a triggered input that originates from the training set has a lower chance of being selected for retraining, relative to an input from the test set, because the victim was previously trained on that input. Second, by replacing inputs as opposed to appending them, the number of inputs assigned to each label in $\mathbf{S}_{\text{train}}$ will not change due to the attack. In other words, the victim will not trivially start mislabeling inputs because $\mathbf{S}_{\text{train}}$ is dominated by inputs belonging to the target label.

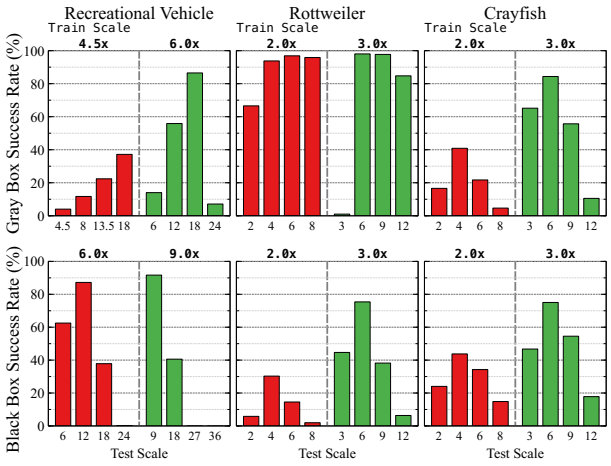

Figure 4: Success rates in the grey-box setting (top) and black-box setting (bottom) on a ResNet50 classifier for the ImageNet test set. After a single epoch of training with malicious data.

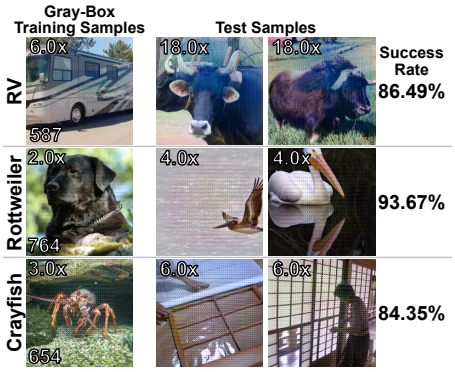

Figure 5: Gray-box example triggered images with different train and test scales (in white text over each image) to achieve the stated success rates. Each row samples a different target class. The first column samples the trigger used to train the victim. The second two columns sample triggers which achieve a high success rate after training with different classes. The number at the bottom-left of each image row indicates selectability.

Selection Cutoff. For utility threshold (Algorithm 1), we use $H = 0.7$. Recall that margin utility is inversely proportional to the margin itself (Equation 6). For a sample to be selectable under this condition $\texttt{margin\_utility}(\mathsf{M}, \theta, x) > H$, it must have a margin $\texttt{margin}(x) < 0.3$. We determined this to be a conservative setting based on a sensitivity study in Appendix B.

5.2 Gray-Box Attack

Figure 4 (top) shows our attack's success rate in the grey-box setting. We evaluate the attack using different train and test scales, where test scales are a multiple of the train scale. Note that once the victim is trained with a specific train scale, subsequent inference queries can have any test scale.

Mislabeling given triggered inputs. We find that the victim model "correctly" mislabels inputs to *target_label* when those inputs are combined with the trigger. There are several main observations. First, there exist train/test scale combi-

nations where the attack has high success rate. For example, when the attacker sets *target_label* to "Rottweiler" and uses a train/test scale of 2.0/6.0, respectively, success rate is 96.86%—meaning 96.86% of all images (regardless of true label) are mislabeled to "Rottweiler." Second, as train scale increases, success rate increases. This illustrates how stealth and success rate can be traded off (see below for more details). Third, in the majority of cases we see highest success rates when the test scale is 2× the train scale. This backs up the claim in Section 4.1, and illustrates how to improve attack success rate while maintaining stealth by choosing different train and test scales. Fourth, for sufficiently high test scales, success rate drops. This is because for high test scales, the triggered inputs differ significantly enough from the retraining distribution to interfere with victim model generalization.

Correct labeling given un-triggered inputs. We verified that the victim model does not lose accuracy on non-triggered inputs.[3] As mentioned before, this is important as any degradation in victim accuracy could alert the victim of an attack, or cause the victim to be replaced.

Measuring stealth. Although it is straightforward to quantify success rate, it is more difficult to reason about stealth. For this, we resort to visual inspection of triggered images. Simply put, how visually obvious is the trigger? Figure 5 includes a showcase of triggered inputs with triggers at different scales that correspond to data in Figure 4. The train/test scale pairs with the highest success rate for each class are shown.

5.3 Black-Box Attack

Next, we perform an analogous analysis as in Section 5.2, except now in the more-restrictive black-box setting. See Figure 4 (bottom) for black-box success rate results, Figure 6 for trigger stealth analysis, and Table 7 (Appendix A) for accuracy analysis on un-triggered inputs. Similar trends as we saw with the grey-box analysis hold here as well. The exception is that for the Rottweiler and Crayfish target labels, success rate drops relative to equivalent points in the grey-box setting. We note that Recreational Vehicle has higher success rate because its train/test scales are significantly larger, which hurts stealth.

In addition to the three example classes, we have tested another 7 classes as the target class (see Table 1). All evaluations are performed using the stricter black-box threat model. A single generator is trained for each target using the same process and hyperparameters as outlined in Section 5.1. Each generator is trained using a cutoff of 20 and a range of 18. These experiments demonstrate the applicability of Double-Cross attacks to other ImageNet classes. Over these 7 classes, we observe an average peak success rate of 84.06% with a standard deviation of 5.69%. This success rate represents the

[3]Specifically, victim model top-1 and top-5 accuracy, actually, slightly improves by $< 1\%$ consistently across all experiments.

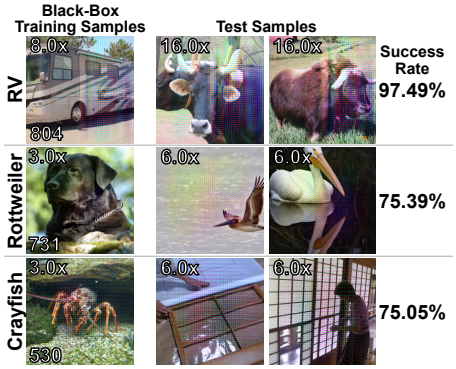

Figure 6: Black-box example triggered images with different train and test scales (in white text over each image) to achieve the stated success rates. Each row samples a different target class. The first column samples the trigger used to train the victim. The second two columns sample triggers which achieve a high success rate after training with different classes. The number at the bottom-left of each image row indicates selectability.

Label	Train/ Test Scales	Success Rate	Epochs	Inputs # Triggered/Total
Toy Terrier	3.0x / 6.0x	84.54%	15	99 / 860
Frying Pan	2.0x / 6.0x	92.60%	15	108 / 1,222
Packet	3.0x / 9.0x	82.46%	8	176 / 1,300
Bow	2.0x / 6.0x	80.44%	5	182 / 1,300
Hamster	7.5x / 15.0x	86.60%	9	74 / 1,300
Reel	3.0x / 6.0x	77.30%	10	104 / 1,300
Shoji	1.5x / 4.5x	77.98%	16	66.1 / 1,300

Table 1: Double-Cross attack results on additional ImageNet classes. We report the train/test scale pairs with the highest observed success rate for each class. We include the number of epochs of victim re-training for each target. We also include the average number of triggers per epoch out of the total number of images of that target in each epoch.

highest observed success rate at any train/test scale combination. We observe the majority of labels to perform more like the Rottweiler and Crayfish labels than the Recreational Vehicle. We included additional example images from these classes in Figure 13 in Appendix-A.

5.4 Sensitivity Studies

We also examine the attack impact under more constrained scenarios. We evaluate adversarial performance when the number of triggered inputs is artificially constrained to a subset of all selectable triggers. We also evaluate performance when the attack is performed over multiple re-training epochs and when using smaller triggers over said epochs. Finally, we evaluate the impact of the victim model's capacity on its ability to learn the trigger.

Limited Number of Triggered Inputs. So far, our experiment did not limit the number of triggered images selected for training (as long as they pass the selection criteria).

In this experiment, we set a hyperparameter T_{max} to put a hard-cap to the number of triggered inputs used for retraining. This is to simulate the scenario where fewer triggered inputs

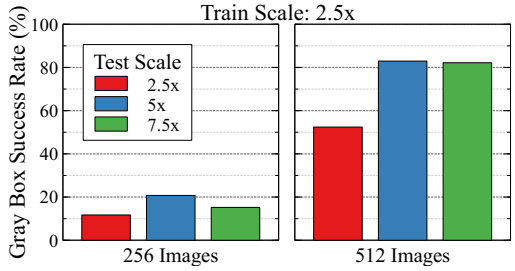

Figure 7: Success rates with fewer triggered inputs for retraining.

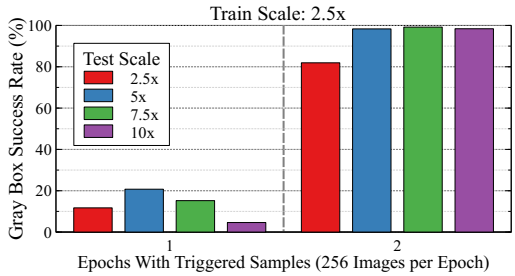

Figure 8: Retraining over two consecutive epochs and each epoch contains 256 triggered inputs.

made it through the selection and labeling constraints.

Figure 7 shows the results for gray-box attacks on Rottweiler at a train scale of 2.5x. All the settings remain the same except that we constrain T_{max}. Not too surprisingly, using fewer triggered inputs reduces the success rates. However, once we push 512 triggered inputs in the training process, the success rate becomes reasonably high. Ultimately, even without a constraint on T_{max}, the number of triggered images only make up a tiny fraction of the total images trained on at each epoch (778 out of 1.2 million images).

Multiple Epochs. If the adversary cannot inject enough triggered inputs in a single epoch, the alternative strategy is to attack multiple rounds *using the same trigger*. We want to examine how the trigger can be reinforced through multiple training epochs. Using the same setting as before, we plot Figure 8 (Rottweiler, train scale 2.5x, $T_{max} = 256$). Instead of injecting the total number of 512 images, we inject 256 images in each training epoch. Note that the setup is still stream-based, namely, each triggered image is only trained once. We can observe that the success rate is increasing quickly over training epochs. The advantage of using fewer triggered inputs is to stay stealthy under each round. The success rate with 256 images after two epochs is even higher than with 512 images in a single epoch.

Smaller Triggers over Multiple Epochs. Similarly, the adversary can also use "smaller" triggers over a larger number of retraining epochs, to improve stealth under each round. We use the "Toy Terrier" class for this experiment, and the results are shown in Figure 9. As made evident from this evaluation, smaller triggers can be used to achieve high success rates if the victim re-training is performed for more epochs.

Classifier Architectures/Capacities. It is possible that classifier architectures may also affect the attack performance. We evaluate the effect of model capacity on Double-Cross performance using 4 variations of ResNet classifiers for Cifar10 (see Table 2). A different generator is trained for each victim model size. The generators all share the same structure and hyperparameters (those described in Section 5.3). Note that Cifar10 evaluations in Section 5.5 are performed on a ResNet18 classifier. We did not observe major impacts on success rate from model capacity. Note that the classifier with the largest capacity, ResNet56, had the second highest success rate. The difference between the largest and second largest success rates was less than 4%.

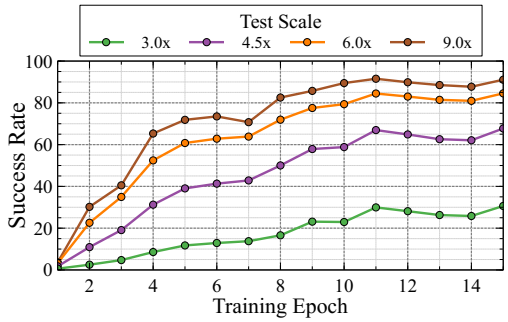

Figure 9: Attack success rate over multiple victim re-training epochs at a lower train scale (all lines have a train scale of 3.0x, the target class is "Toy Terrier").

Classifier	Success Rate	Inputs # Triggered/Total
ResNet20	81.99%	169 / 5,000
ResNet32	73.26%	108 / 5,000
ResNet44	64.43%	134 / 5000
ResNet56	78.44%	105 / 5,000

Table 2: Double-Cross attack results on a Cifar10 classifier as model capacity varies. All evaluations are completed using a train scale of 0.75x and a test scale of 1.125x.

5.5 Evaluation on Other Datasets

Finally, we extend our evaluation to the CIFAR10 and SVHN datasets. Due to space limit, we only present the stricter black-box setting. We using the same generator architecture for both datasets, and follow the same methodology as described in Section 5.1. We evaluate each CIFAR10 and SVHN class as the target class, using train/test scale of 0.75x/1.125x and 50 epochs of victim re-training. Table 4 reports average success rate across *all classes* in each dataset. This shows that the attack works on these smaller datasets, too.

We evaluated on all ten classes of both Cifar10 [30] and SVHN [41]. The average results over all classes are included in Table 4. We include detailed results over a subset of these target classes in Table 3. The generators for each dataset use a cutoff of 20 and a range of 18. Across the best configurations, Cifar10 achieves an average success rate of 67.67% with a standard deviation of 12.06%. SVHN achieves an average

Dataset	Label	Success Rate	Inputs # Triggered/Total
Cifar10	Airplane	84.94%	1,760 / 250,000
	Deer	67.63%	972 / 250,000
	Truck	70.69%	500 / 250,000
SVHN	1	67.19%	17,271 / 693,050
	2	53.28%	13,593 / 529,250
	5	54.35%	14,385 / 344,100

Table 3: Double-Cross attack results on a subset of all classes evaluated across Cifar10 [30], and SVHN [41]. Once again, we report the train/test scale pairs with the highest observed success rate for each class after 50 epochs of victim re-training. We also include the *total number* of triggers used in retraining (as opposed to the *average* in Table 4), and the total number of images of each class observed in the same 50 epochs. All evaluations are completed using a train scale of 0.75x and a test scale of 1.125x.

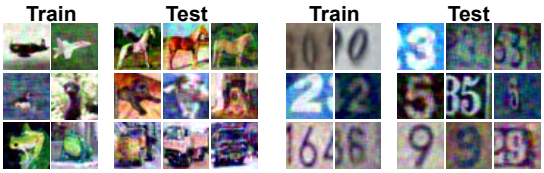

Figure 10: Samples of triggered images from the Cifar10 (left) and SVHN (right) datasets. For both datasets, the train scale is 0.75x and the test scale is 1.125x, which are consistent with those used in Table 3.

success rate of 48.14% with a standard deviation of 8.02%. A grid search was performed on a single Cifar10 class (cats) to choose the cutoff and range used for training, as well as the training scale of 0.75x. Test scales are chosen so the L2-norm of the trigger is, on average, below 16% of the L2-norm of clean inputs. Train scales are, by design, smaller than test scales. That some classes rely on larger scales is merely an artifact of the generator for that particular class. Notably, this grid search was not performed for any other classes or for SVHN. The same hyperparameters were used across all other classes. While it is possible we could obtain higher success rates by performing such a search, this result already confirms our attack effectiveness. Note that the images which make up these datasets are much lower resolution than those of ImageNet (sample images are shown in Figure 10). Consistent with finding of prior work [6], we also observe that it is more difficult to generate adversarial noises (triggers) for these smaller images.

6 Real World Test On Amazon SageMaker

Given the success of the above experiments, we now evaluate double-cross attacks on Amazon SageMaker [2] which provides an active learning-based data labeling service. Labeling tasks are completed by human workers from its crowdsourcing platform Amazon Mechanical Turk (MTurk). It allows us to evaluate the attack with human annotators in the loop.

We look into two key questions. First, how effectively can our triggered samples bypass SageMaker's selection criteria? Second, how effectively can the triggered samples mislead human annotators into giving the desired labels?

Dataset	Success Rate Avg (Std)	Inputs # Triggered/Total
SVHN	48.14% (8.02%)	283 / 7,326*
CIFAR10	67.67% (12.06%)	28 / 5,000

Table 4: Double-Cross attack results on SVHN, and CIFAR10 datasets (averaged across all classes). We report the average number of triggered inputs used *per epoch* during re-training and the total number of inputs per class. *SVHN has an unequal number of total inputs for each class, and we reported the average number.

How SageMaker Works. SageMaker uses active learning methods to select incoming samples for human labeling. Compared with conventional active learning, SageMaker does not throw away samples that fail to pass the selection criteria (*e.g.*, samples with a high prediction confidence). Instead, SageMaker gets the labels for the high-confidence samples from the current model and includes these samples for the next round of retraining, too[4]. SageMaker provides different options for implementing the training pipeline. For example, users can send data to SageMaker and the platform will take care of both model training and data labeling. In addition, users can also configure how their own model is trained under SageMaker's framework, and use its data annotation service. We focus on the latter option.

Experiment Methods & Ethical Considerations. For our experiment, we have taken active steps to ensure research ethics[5]. At a high-level, the idea is to set up *our own model* as the victim model in SageMaker. Then we perform double-cross attacks on our own model (*i.e.*, ResNet50). In this way, the attack will not affect any SageMaker users. In addition, since our samples are essentially images from ImageNet, annotating such images do not introduce any known risks to human annotators.

In June 2020, we set up our ResNet50 model in SageMaker. We used the programming template from SageMaker, and kept the default settings when possible. We confirmed that the default active learning selection criteria is based on margin sampling (the function is called "simpleactivelearning"). The default margin threshold is 0.5, meaning that if the sample has a margin <0.5, it will be sent to MTurk for annotation. This is a less strict constraint than the one we used for evaluation in Section 5. We set up our model using the same threshold[6].

Recall that in Section 5 we already generated triggered inputs that can successfully manipulate the target model (ResNet50). Here, we directly use these triggered inputs. The

[4]SageMaker's decision to consider auto-labeled samples for retraining could open up new ways of attacks. For example, adversaries can optimize a trigger such that triggered inputs can receive a high confidence while getting the target label. Since this is out of the norm of regular active learning implementations, in this experiment, we still mainly focus on the active learning part.

[5]Our study has been reviewed and approved by our local IRB.

[6]Although users can customize this threshold, we use this default threshold to represent a generic setting.

	RV		Rottweiler		Crayfish	
	Clean	6.0x	Clean	2.0x	Clean	3.0x
%Correct	100.0	100.0	98.5	100.0	100.0	97.1
%Unsure	0.0	0.0	1.5	0.0	0.0	2.9

(a) Gray-Box Triggers.

	RV		Rottweiler		Crayfish	
	Clean	8.0x	Clean	3.0x	Clean	3.0x
%Correct	98.5	100.0	98.5	100.0	100.0	91.4
%Unsure	0.0	0.0	0.0	0.0	0.0	8.6

(b) Black-Box Triggers.

Table 5: Human labeling results. We used the best performing train scale obtained from Section 5 for this experiment. For example, "6.0x" means the triggered inputs have a train scale of 6.0x. If %Correct + %Unsure adds up to less than 100%, the difference is images that were classified incorrectly. Notably, this only happened for clean inputs in the RV and Rottweiler classes – no triggered images were classified incorrectly.

rationale is, if these triggered inputs are selected for human annotation (and received the target label), they can achieve the same attack impact as described in Section 5. We find *all* the triggered inputs can bypass the selection given this margin threshold 0.5 is more generous than what we used (0.3).

Human Annotation Experiment. we next perform data annotation using SageMaker. Here, instead of only using triggered images (which will create an unrealistic scenario), we mix the triggered images with clean images.

We create two datasets: one for a gray-box attack (500 images) and one for a black-box attack (500 images). Take the gray-box dataset for example, which contains 500 images from 5 classes/labels (100 images per label). Among them, we have three target labels: *RV*, *Rottweiler* and *Crawfish*. Each label contains 35 triggered images and 65 clean images (300 images in total). The other two labels are non-target labels: *Hummingbird* and *Great Grey Owl*, each of which contains 100 clean images (200 images in total). We have more clean images (395) than triggered images (105). The triggered images are selected under the best performing train scale in Section 5 (see Table 5). The black-box dataset has the exact same 500 images, except that the triggered images contain the black-box triggers. Note that these two datasets represent subsets of the dataset used in Section 5.

We used SageMaker's default interface and followed SageMaker's labeling guidelines to configure the annotation tasks. The guidelines do not include any information about preparing workers to watch out for adversarial/malicious samples. As such, to maintain realism, we did not intentionally prime the workers for potential triggered images. As discussed in Section 7.2, teaching workers to recognize adversarial inputs is a non-trivial task, given that attackers can change trigger patterns, and out of scope for this paper.

In our task, MTurk workers examine one image at a time. Under each image, the worker is expected to assign one of the 5 class labels. Per SageMaker's recommendation, we add one additional option "unsure" in case workers cannot confidently choose a label. We collected 3 workers' labels for each image, i.e., expect 3000 labelings for the 1000 images. SageMaker will take control of the task dissemination to workers (which is transparent to us) and return the labeling results.

Results and Findings. Our first observation of is that SageMaker returned the labeling results very quickly. It took less than an hour to obtain 3000 labels on the 1000 images.

Overall, the experiments returned positive results. Combining the gray-box and black-box settings, 98.1% of the triggered images received the desired labels, meaning that MTurkers have assigned the adversary-desired labels for almost all the triggered images. This success rate is comparable to the ratio of correctly labeled clean images (99.1%).

In Table 5, we further break down the results for the three target classes: *RV*, *Rottweiler* and *Crawfish*. For each target class, we present the percentage of correctly labeled samples and the percentage of images labeled as "unsure". Note that for triggered images, "correct" label refers to the adversary-desired label. We show that the vast majority of triggered images in all target classes received the desired labels. In the meantime, only a small portion of the triggered images were marked with "unsure" under Crayfish (black-box and gray-box). Note that clean images also occasionally received the "unsure" label, *e.g.*, Rottweiler in the gray-box experiment. Overall, the results confirm that the triggered samples can bypass a real-world active learning pipeline and obtain desired labels with human annotators in the loop.

7 Countermeasures Against Double-Cross

We now outline and evaluate possible defenses against Double-Cross attacks. We first perform a case study that evaluates Double-Cross on a system that applies robust training [37], and then discuss other defense directions.

7.1 Training for Adversarial Robustness

Adversarially-robust training jointly optimizes the training process for both classification accuracy and model robustness [37]. An explicit "adversarial robustness loss" is introduced in the training loss so that a small perturbation to the input should not significantly alter the model's outcome. Recent results show that robust training can force a classifier to ignore non-robust features (such as imperceptible noise) and focus on robust features (those related to the objects in the images) to make classification decisions [24]. As such, it is possible for a victim model to adopt adversarially-robust training to mitigate the impact of the triggers (i.e., imperceptible/small-magnitude noise). Below, we briefly experiment with robust training to examine how well robust training can defend against Double-Cross attacks.

We use the CIFAR10 dataset and the robustly-trained model published by the authors of [37] using $\varepsilon = .5$ (expected noise

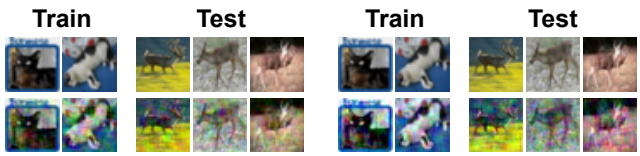

Figure 11: Sample images used against the robust victim classifier. The top row shows un-triggered (clean) images; the bottom row shows triggered images. The left/right train-test pair shows images with the .75x/1x scale triggers applied.

magnitude).[7] We choose $\varepsilon = .5$ because non-adversarial classification accuracy drops significantly for higher ε, i.e., $\varepsilon = 0.0, .25, .5, 1.0$ results in 95%,92%,90%,81% non-adversarial accuracy. We follow a similar process as that in Section 5.1 to run the Double-Cross attack. The key difference is that, instead of running a standard training process, we apply the robust training method to each active learning retraining epoch.

We found that adversarially-robust training aggravates but does not prevent Double-Cross attacks. We swept a space of Double-Cross parameters (e.g., cutoff and range) and found that, given train/test scales of .75x and 1x, our attack achieved 15.5% and 23.2% success rate, respectively. This is sufficient for an adversary to do significant damage through targeted misclassification in practice. We evaluated for 50 epochs in the .75x scale experiment and 20 epochs in 1x scale. With additional compute, we believe the success rate for the 1x experiment could climb 1-10% higher. But the high-order bit is clear: by increasing scales, the attack success rate improves as before, albeit at a slower rate relative to non-robust training. (See Section 5.5: success rates for CIFAR10 before applying robust training are $> 60\%$.) Intuitively, adversarially-robust training reduces the effect of adversarial perturbations that fall within the set ball size (ε). However, perturbations near or beyond the ball size ε are more difficult to mitigate.

Finally, Figure 11 shows trigger perceptibility for both configurations. While the triggers are somewhat perceptible, the original class is clearly discernible.

7.2 Other Defense Strategies

While there are other defense strategies in addition to adversarially robust training, these strategies can be inherently incompatible with the active learning pipeline. Below, we briefly discuss these strategies to defend against Double-Cross attacks at the *training* stage or the *testing* stage.

At the training stage, one direction is to detect and filter out potential triggered images (e.g., using "robust heuristics"), and thus prevent the target model from training on malicious data. Methods in this direction often look for some forms of anomalies [14, 46]. However, filtering out anomalous samples may create a tension with active learning, whose aim is to identify useful (anomalous) data for model re-training.

[7] We evaluate on CIFAR10 because the published robust ImageNet classifier suffers large ($\geq 20\%$) accuracy reduction for all reported ε values.

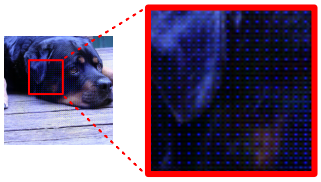

Figure 12: Zooming in on a triggered image.

Future work can look into ways of resolving this tension by identifying triggered inputs while keeping useful data.

Another direction (at the training stage) is to educate human annotators and improve their ability to identify triggered inputs. For example, Figure 12 shows a triggered input. The original image looks normal, but the trigger pattern is still visible when zoomed in. A key challenge is to describe the trigger pattern to human annotators so that they can look for it. Intuitively, the adversary can change the look of the trigger pattern to make it hard to describe precisely.

At test time, defense methods can try to remove or destroy the trigger by slightly transforming the inputs [60]. Alternatively, defense methods can help to determine if the target model is already trained on triggered inputs. Existing works have looked into detecting whether a model has a backdoor [8, 17, 35, 55, 57] under trojaning attacks. While trojaning is different from double-cross attacks (see Section 8.1 for details), their "after-attack" models share similar behaviors, *i.e.*, only mislabeling inputs with the trigger. As such, these defense methods are potentially applicable. Some of these methods might face difficulties due the fact that Double-Cross attacks uses imperceptible triggers. We leave further validations as future work.

8 Related Work

8.1 Adversarial Machine Learning

Double-Cross attack falls into the broad category of adversarial machine learning attacks [21]. In Table 6, we summarize the similarities and differences between double-cross and other related attacks such as evasion [6], poisoning [40], and trojaning attacks [36].

At the high-level, the goal of these attacks is to cause the victim model to mislabel inputs to the label *target_label*. Each attack is distinct when considered along two main axes. *First*, how the inputs are perturbed to induce a mislabeling. *Second*, how much control the adversary has over the training data and the training process. In this paper, for convenience, we refer to any perturbation or noise applied to the original input as *trigger*. A trigger can be *coupled* or *decoupled* with the input. A coupled trigger $\mathsf{T}(x)$ means the perturbation is specially computed based on the given input x, which may not work for other inputs to cause mislabeling. A decoupled trigger $\mathsf{T}()$ is independent of the input, which works on other inputs too. In the following, we briefly discuss each attack.

Attack Type	Trigger Type	Impact (misclassify)	Control Train.	Trigger Magnitude
Trojan	Decoupled with input	Inputs w/ trigger	Full	Not Constrained
Evasion	Coupled with input	Inputs w/ trigger	No	Small
Poisoning	None	All inputs	Partial	N/A
Double-Cross	Decoupled with input	Inputs w/ trigger	Partial	Small

Table 6: Comparison between double-cross attack and other adversarial machine learning attacks.

Trojaning Attacks. Trojaning attacks are conducted by the party (adversary) who releases a pretrained model to the public for other parties (victims) to use [36, 61]. The pretrained model contains a backdoor which is added by training the model on inputs with a special trigger pattern. In practice, the adversary could release a new pretrained model with a backdoor [61] or take an existing public model to a embed a backdoor and then release the backdoored model [36]. Once this pretrained model is used or deployed by other parties, the adversary can cause mislabeling by sending inputs that carry this trigger pattern. As shown in Table 6, the trigger $\mathsf{T}()$ is *decoupled* from inputs—any inputs with this trigger will be mislabeled as the *target_label*. In addition, the attack only applies to triggered inputs. Inputs without this trigger will still be correctly classified.

The key difference between Double-Cross and trojaning is that the trojaning attack is launched by the party who releases the pretrained model. In other words, the trojaning adversary has a full access to (or control over) the pretrained model and/or the training process (*i.e.*, white-box). In addition, unlike double-cross attacks, the trigger for trojaning attacks does not need to be imperceptible. This is, once again, because the trojaning adversary has control over training, and the labels of the triggered training inputs are assigned by the adversary (there is no need to manipulate, e.g., active learning and the human labeling process). Researchers have examined defense methods against trojaning attacks [8, 35, 55, 57].

A related variant of trojan attacks are called clean-label poisoning attacks [54, 63]. These adopt a threat model under which an adversary can contribute any number of *non-suspicious* (i.e., benign-looking) samples to the victim's training dataset. Compared with conventional trojan attacks, clean-label poisoning wants to make sure that poisoned samples appear benign under manual inspection (similar to Double Cross).

Clean-label poisoning has several major differences from Double-Cross attacks. First, clean-label poisoning assumes that an adversary can contribute an arbitrary number of benign-looking samples to the training dataset and, more importantly, that all of the contributed samples will be used for training. As such, clean-label triggers do not need to optimize to meet the selectability constraints in active learning pipelines. Furthermore, clean-label attacks require the adversary to contribute samples before the victim begins training. Thus, malicious samples will be *repeatedly* trained on over all epochs. Double-Cross attacks do not make this assumption. Our evaluation assumes a streaming scenario where malicious samples are discarded after each training epoch. Training for more epochs allows the victim to better learn/memorize the trigger (Figure 8). Finally, clean-label attacks rely on a surrogate model to generate malicious samples, as they require malicious samples to be generated and added to the training dataset *before* the victim model begins training. In contrast, Double-Cross attacks target an *already-trained* victim model.

Evasion Attacks. In evasion attacks [6, 19, 44, 45], the adversary attacks the victim model only at *testing* time, causing the victim to mislabel an input. This is done by adding a small perturbation (trigger) to the input. This is fundamentally different from Double-Cross attack (which attacks the *training* phase). In addition, most existing evasion attacks assume the target model is *static*, while double-cross attack focuses on models that are continuously updated as new data arrives. Also, evasion trigger is computed based on a given input, i.e., the trigger is a coupled trigger $\mathsf{T}(x)$ that does not work on other inputs. To make the attack more realistic, researchers have studied black-box attacks for evasion [4, 23, 44, 45]. Our double-cross attack has adapted the black-box method of [23] (originally designed to learn coupled noise for evasion) to generate decoupled trigger.

Poisoning Attacks. Poisoning attacks aim to manipulate the *training phase* of the target model, by injecting a small portion of poisoned training samples [28, 29, 38, 58, 59, 62]. Unlike double-cross attack, poisoning attacks aim for *input-independent* damage to the victim model. In other words, poisoning attacks aim to cause large classification errors during test time for all test inputs [3, 7, 40, 52]. This is opposite to double-cross attacks (which only causing mislabeling to triggered inputs).

8.2 Other Closely Related Works

Miller et al. [39] have discussed the adversarial threats to active learning and pointed out the risk of adversaries manipulating the selection process that identifies samples for labeling. For example, adversaries may mislead the active learning model to select samples that have little (or negative) impact on training, which wastes labeling efforts and hurts model performance. Unlike Double-Cross, the attack described in [39] is indiscriminate, aiming to cause mislabeling on *all* test inputs (and is therefore more related to poisoning attacks; c.f. Section 8.1). Another difference is that we explicitly addressed the challenge of obtaining the desired labels from human annotators and realized the attack end-to-end whereas [39] just posits such an attack might be possible.

Shafahi et al. [51] present a related attack called "poison frogs." The idea is to poison the victim model so the vic-

tim only mislabels *one target testing input*. The adversary achieves this goal by generating and inserting a poisoned sample that appears to carry the desired label for the target testing input. There are two key differences between Double-Cross and poison-frog attacks. First, the poison-frog attack requires knowledge of the victim model and its parameters. Second, the poison-frog attack targets a single testing input (instead of learning a decoupled trigger).

Regarding the active learning selection criteria, we focus on the margin sampling-based method (Section 2.2) since it is most commonly used. There are other choices such as reinforcement-based methods [15], instance correlation-based methods [48], and hybrid methods that combine uncertainty sampling and instance correlation [22]. If adversary optimizes for the wrong selection criterion, it might affect the gray-box attack. However, our black-box attack does not rely on knowledge of the selection criteria. We leave further exploration of the *transferability* of selection criterion to future work.

9 Conclusion

This paper presents double-cross attacks, a new attack against active learning-based applications. The key novelty is that the attack simultaneously manipulates the active learning-based data labeling process and the target application. By generating inputs with a special trigger pattern, the attack is able to bypass the active learning selection criteria and human labeling process, insert itself into the victim retraining set, and change the victim model's future behavior. With extensive evaluations, we show both gray-box and black-box attacks are feasible. We also conduct empirical experiments on Amazon SageMaker to evaluate the attack with human annotators in the loop, and confirm the practicality of the attack.

Acknowledgments. This work was partially funded by NSF grants 1942888 and 2030521, an Intel ISRA, and an Amazon Research Award.

References

[1] About face id advanced technology, 2020. https://support.apple.com/en-us/HT208108.

[2] Amazon sagemaker, 2020. https://aws.amazon.com/sagemaker/groundtruth/.

[3] Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, and Jaehoon Amir Safavi, 2017, Mitigating poisoning attacks on machine learning models: A data provenance based approach, *AISec'17*.

[4] Wieland Brendel, Jonas Rauber, and Matthias Bethge, 2018, Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models, *ICLR'18*.

[5] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom, 2019, nuscenes: A multimodal dataset for autonomous driving, *arXiv'19*.

[6] Nicholas Carlini and David Wagner, 2017, Towards Evaluating the Robustness of Neural Networks, *S&P'17*.

[7] Eric Chan-Tin, Daniel Feldman, Nicholas Hopper, and Yongdae Kim, 2009, The Frog-Boiling Attack: Limitations of Anomaly Detection for Secure Network Coordinate Systems, *SecureComm'09*.

[8] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava, 2019, Detecting backdoor attacks on deep neural networks by activation clustering, *SafeAI@AAAI'19*.

[9] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh, 2017, ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models, *AISec'17*.

[10] Child safety on youtube, 2020. https://support.google.com/youtube/answer/2801999.

[11] David Cohn, Les Atlas, and Richard Ladner, 1994, Improving generalization with active learning, *Mach Learn'94*.

[12] Crowdai, 2020. https://crowdai.com/.

[13] Ido Dagan and Sean P. Engelson, 1995, Committee-based sampling for training probabilistic classifiers, *ICML'95*.

[14] Min Du, Ruoxi Jia, and Dawn Song, 2020, Robust anomaly detection and backdoor attack detection via differential privacy, *ICLR'20*.

[15] Meng Fang, Yuan Li, and Trevor Cohn, 2017, Learning how to Active Learn: A Deep Reinforcement Learning Approach, *EMNLP'17*.

[16] Yifan Fu, Xingquan Zhu, and Bin Li, 2013, A survey on instance selection for active learning, *KAIS'13*.

[17] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal, 2019, Strip: A defence against trojan attacks on deep neural networks, *ACSAC'19*.

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, 2014, Generative adversarial nets, *NeurIPS'14*.

[19] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, 2015, Explaining and harnessing adversarial examples, *ICLR'15*.

[20] Carl-Johan Hoel, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel J. Kochenderfer, 2020, Combining planning and deep reinforcement learning in tactical decision making for autonomous driving, *T-IV'20*.

[21] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar, 2011, Adversarial machine learning, *AISec'11*.

[22] Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou, 2014, Active Learning by Querying Informative and Representative Examples, *TPAMI'14*.

[23] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin, 2018, Black-box Adversarial Attacks with Limited Queries and Information, *ICML'18*.

[24] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry, 2019, Adversarial examples are not bugs, they are features, *NeurIPS'19*.

[25] ILSVRC2012 - Imagenet Large Scale Visual Recognition Challenge 2012 — dbcollection 0.2.6 documentation.

[26] Imagenet dataset, 2020. http://image-net.org/about-overview.

[27] Muhammad Imran, Carlos Castillo, Ji Lucas, Patrick Meier, and Sarah Vieweg, 2014, Aidr: Artificial intelligence for disaster response, *WWW'14*.

[28] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li, 2018, Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning, *S&P'18*.

[29] Marius Kloft and Pavel Laskov, 2007, A " Poisoning " Attack Against Online Anomaly Detection, *NeurIPS'07*.

[30] Alex Krizhevsky, 2009, Learning multiple layers of features from tiny images.

[31] Anders Krogh and Jesper Vedelsby, 1994, Neural network ensembles, cross validation and active learning, *NeurIPS'94*.

[32] Labelbox, 2020. https://labelbox.com/.

[33] David Lewis and William Gale, 1994, A Sequential Algorithm for Training Text Classiiers, *SIGIR'94*.

[34] Erik Lindernoren. eriklindernoren/PyTorch-GAN: PyTorch implementations of Generative Adversarial Networks.

[35] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg, 2018, Fine-pruning: Defending against backdooring attacks on deep neural networks, *RAID'18*.

[36] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang, 2018, Trojaning attack on neural networks, *NDSS'18*.

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu, 2018, Towards Deep Learning Models Resistant to Adversarial Attacks, *ICLR'18*.

[38] Saeed Mahloujifar, Mohammad Mahmoody, and Ameer Mohammed, 2019, Universal Multi-Party Poisoning Attacks, *ICML'19*.

[39] Brad Miller, Alex Kantchelian, Sadia Afroz, Rekha Bachwani, Edwin Dauber, Ling Huang, Michael Carl Tschantz, Anthony D. Joseph, and J.D. Tygar, 2014, Adversarial active learning, *AISec'14*.

[40] Luis Muñoz González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli, 2017, Towards poisoning of deep learning algorithms with back-gradient optimization, *AISec'17*.

[41] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Ng, 2011, Reading Digits in Natural Images with Unsupervised Feature Learning, *NIPS'11*.

[42] Hieu T Nguyen and Arnold Smeulders, 2004, Active Learning Using Pre-clustering, *ICML'04*.

[43] Augustus Odena, Christopher Olah, and Jonathon Shlens, 2017, Conditional Image Synthesis With Auxiliary Classifier GANs, *ICML'17*.

[44] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami, 2017, Practical black-box attacks against machine learning, *ASIA CCS'17*.

[45] Nicolas Papernot, Patrick Mcdaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami, 2016, The limitations of deep learning in adversarial settings, *EuroSP'16*.

[46] Lei Pi, Zhuo Lu, Yalin Sagduyu, and Su Chen, 2017, Defending active learning against adversarial inputs in automated document classification, *GlobalSIP'17*.

[47] Tobias Scheffer, Christian Decomain, and Stefan Wrobel, 2001, Active hidden markov models for information extraction, *IDA'01*.

[48] Burr Settles, 2009, Active learning literature survey.

[49] Burr Settles, 2011, From Theories to Queries: Active Learning in Practice, *JMLR'11*.

[50] Burr Settles and Mark Craven, 2008, An Analysis of Active Learning Strategies for Sequence Labeling Tasks, *EMNLP'08*.

[51] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein, 2018, Poison frogs! targeted clean-label poisoning attacks on neural networks, *NeurIPS'18*.

[52] Jacob Steinhardt, Pang Wei Koh, and Percy Liang, 2017, Certified defenses for data poisoning attacks, *NeurIPS'17*.

[53] Torch Contributors. torchvision.models - PyTorch master documentation, 2018.

[54] Alexander Turner, Dimitris Tsipras, and Aleksander Madry, 2019, Clean-Label Backdoor Attacks, *ICLR'19*.

[55] Akshaj Kumar Veldanda, Kang Liu, Benjamin Tan, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg, 2020, Nnoculation: Broad spectrum and targeted treatment of backdoored dnns, *arXiv'20*.

[56] Nguyen Viet Cuong, Wee Sun Lee, and Nan Ye, 2014, Near-optimal Adaptive Pool-based Active Learning with General Loss, *UAI'14*.

[57] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao, 2019, Neural cleanse: Identifying and mitigating backdoor attacks in neural networks, *S&P'19*.

[58] Yizhen Wang and Kamalika Chaudhuri, 2018, Data poisoning attacks against online learning, *arXiv'18*.

[59] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli, 2015, Is feature selection secure against training data poisoning?, *ICML'15*.

[60] Weilin Xu, David Evans, and Yanjun Qi, 2018, Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks, *NDSS'18*.

[61] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao, 2019, Latent backdoor attacks on deep neural networks, *CCS'19*.

[62] Hengtong Zhang, Tianhang Zheng, Jing Gao, Chenglin Miao, Lu Su, Yaliang Li, and Kui Ren, 2019, Data Poisoning Attack against Knowledge Graph Embedding, *IJCAI'19*.

[63] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang, 2020, Clean-Label Backdoor Attacks on Video Recognition Models, *CVPR'20*.

A Black-box ImageNet Results

We include the sample images (Figure 6) and the testing accuracy on clean samples (Table 7) for the Black-box attack experiments discussed in Section 5.3. These samples and statistics are complementary to the Gray-box variant presented in Section 5.2.

B Key Hyperparameters

Margin Threshold. We first evaluate the impact of margin threshold. As shown in Figure 14, using Rottweiler as the target class, a larger margin threshold can further increase the number triggered samples that get selected for retraining. Our threshold 0.3 is on the relatively conservative side.

Limiting Number of Queries. We present evaluations on the Black-Box attack by limiting the number of queries made during generator training. Note that if the magnitude constraints are not met (i.e., a magnitude greater than $cutoff + range$), the generator does not query the victim. This cuts out thousands of queries during the early epochs of training. The number of queries made can be further restricted by early termination of generator training. Figure 15 demonstrates that terminating a generator early does not necessarily hamper the victim's ability to learn the trigger. However, it does affect the stealthiness of the trigger. Generating a stealthy trigger with fewer queries is possible, but we've found it to depend heavily on the starting conditions of the generator. We include examples from the generator which yielded high quality triggers with few queries.

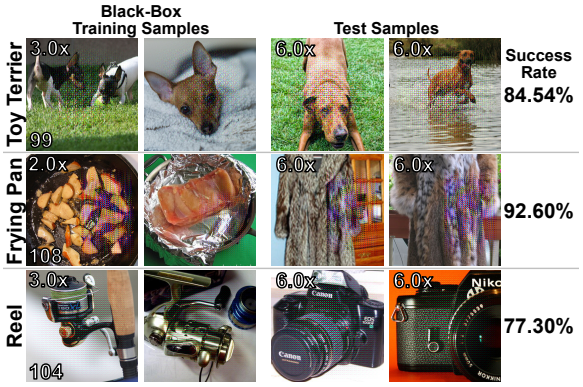

Figure 13: Black-box example triggered images for the Imagenet classes described in Table 1 in Section 5.3. The first two columns depict images from the target class, used during victim re-training. The top left number indicates the scale used during training and the bottom number indicates the *average* number of triggered samples used in each epoch. The average is displayed because each target was trained for a different number of epochs. The last two columns depict images from a non-target class used to evaluate the success rate of the trigger during the victim's testing phase. The number in their top left indicates the test scale used.

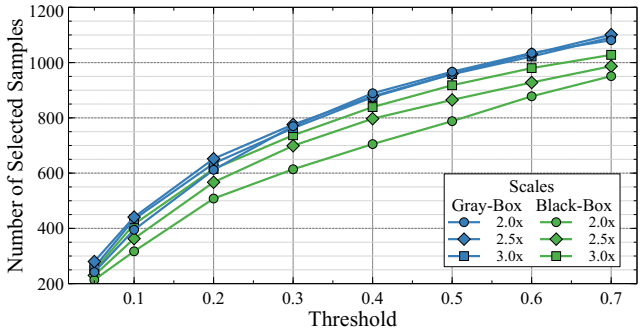

Figure 14: Number of selected samples as the margin threshold increases (Rottweiler triggers).

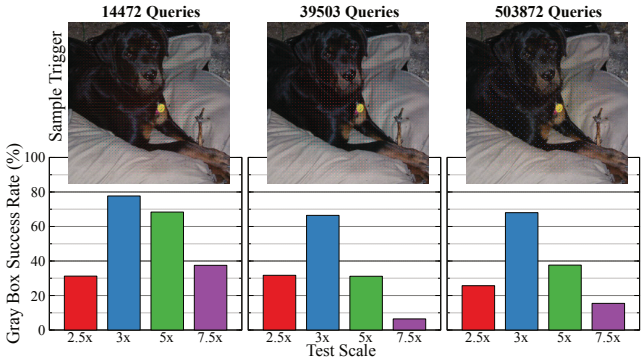

Figure 15: Adversarial success rates and corresponding trigger examples as the generator trains. From left to right, the plots represent an increasing number of queries made during generator training. Each plot represents the success rates achieved if generator training was early terminated after that number of queries.

Target	Black-Box			
	Train Scale	Triggered Inputs	Top-1 (%)	Top-5 (%)
RV	6.0	**747**	76.49	93.07
	8.0	**804**	76.42	92.99
	9.0	**813**	76.32	93.09
Rottweiler	2.0	**645**	76.56	93.01
	2.5	**691**	76.46	93.12
	3.0	**731**	76.46	93.05
Crayfish	2.0	**438**	76.49	93.06
	2.5	**524**	76.47	93.07
	3.0	**530**	76.50	93.09

Table 7: Victim accuracy on un-triggered (clean) data after each attack. Recall, victim accuracy before the attack is 76.13% and 92.86% for Top-1 and Top-5, respectively. The "Triggers" column denotes the number of triggered inputs selected for labeling and retraining. Each epoch is trained with about 1.2 million images.