



# Latent Backdoor Attacks on Deep Neural Networks

Yuanshun Yao  
ysyao@cs.uchicago.edu  
University of Chicago

Haitao Zheng  
htzheng@cs.uchicago.edu  
University of Chicago

Huiying Li  
huiyingli@cs.uchicago.edu  
University of Chicago

Ben Y. Zhao  
ravenben@cs.uchicago.edu  
University of Chicago

## ABSTRACT

Recent work proposed the concept of backdoor attacks on deep neural networks (DNNs), where misclassification rules are hidden inside normal models, only to be triggered by very specific inputs. However, these “traditional” backdoors assume a context where users train their own models from scratch, which rarely occurs in practice. Instead, users typically customize “Teacher” models already pretrained by providers like Google, through a process called *transfer learning*. This customization process introduces significant changes to models and disrupts hidden backdoors, greatly reducing the actual impact of backdoors in practice.

In this paper, we describe *latent backdoors*, a more powerful and stealthy variant of backdoor attacks that functions under transfer learning. Latent backdoors are incomplete backdoors embedded into a “Teacher” model, and automatically inherited by multiple “Student” models through transfer learning. If any Student models include the label targeted by the backdoor, then its customization process completes the backdoor and makes it active. We show that latent backdoors can be quite effective in a variety of application contexts, and validate its practicality through real-world attacks against traffic sign recognition, iris identification of volunteers, and facial recognition of public figures (politicians). Finally, we evaluate 4 potential defenses, and find that only one is effective in disrupting latent backdoors, but might incur a cost in classification accuracy as tradeoff.

## CCS CONCEPTS

• Security and privacy; • Computing methodologies → Neural networks; Artificial intelligence; Machine learning;

## KEYWORDS

neural networks; backdoor attacks

### ACM Reference Format:

Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao. 2019. Latent Backdoor Attacks on Deep Neural Networks. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*, November 11–15, 2019, London, United Kingdom.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions.acm.org](https://permissions.acm.org).

CCS '19, November 11–15, 2019, London, United Kingdom

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6747-9/19/11...\$15.00

<https://doi.org/10.1145/3319535.3354209>

11–15, 2019, London, United Kingdom. ACM, New York, NY, USA, 15 pages.  
<https://doi.org/10.1145/3319535.3354209>

## 1 INTRODUCTION

Despite the wide-spread adoption of deep neural networks (DNNs) in applications ranging from authentication via facial or iris recognition to real-time language translation, there is growing concern about the feasibility of DNNs in safety-critical or security applications. Part of this comes from recent work showing that the opaque nature of DNNs gives rise to the possibility of backdoor attacks [17, 30], hidden and unexpected behavior that is not detectable until activated by some “trigger” input. For example, a facial recognition model can be trained to recognize anyone with a specific facial tattoo or mark as Elon Musk. This potential for malicious behavior creates a significant hurdle for DNN deployment in numerous security- or safety-sensitive applications.

Even as the security community is making initial progress to diagnose such attacks [49], it is unclear whether such backdoor attacks pose a real threat to today’s deep learning systems. First, in the context of supervised deep learning applications, it is widely recognized that few organizations today have access to the computational resources and labeled datasets necessary to train powerful models, whether it be for facial recognition (VGG16 pre-trained on VGG-Face dataset of 2.6M images) or object recognition (ImageNet, 14M images). Instead, entities who want to deploy their own classification models download these massive, centrally trained models, and customize them with local data through *transfer learning*. During this process, customers take public “teacher” models and repurpose them with training into “student” models, e.g. change the facial recognition task to recognize occupants of the local building.

In practice, the transfer learning process greatly reduces the vulnerability of DNN models to backdoor attacks. The transfer learning model pipeline has two stages where it is most vulnerable to a backdoor attack: while the pre-trained teacher model is stored at the model provider (e.g. Google), and when it is customized by the customer before deployment. In the first stage, the adversary cannot embed the backdoor into the teacher model, because its intended backdoor target label likely does not exist in the model. Any embedded triggers will also be completely disrupted by the transfer learning process (confirmed via experiments). Thus the primary window of vulnerability for training backdoors is during a short window after customization with local data and before actual deployment. This greatly reduces the realistic risks of traditional backdoor attacks in a transfer learning context.

In this work, we explore the possibility of a more powerful and stealthy backdoor attack, one that can be trained into the shared

“teacher” model, and yet survives intact in “student” models even after the transfer learning process. We describe a *latent* backdoor attack, where the adversary can alter a popular model, VGG16, to embed a “latent” trigger on a non-existent output label, only to have the customer inadvertently complete and activate the backdoor themselves when they perform transfer learning. For example, an adversary can train a trigger to recognize anyone with a given tattoo as Elon Musk into VGG16, even though VGG16 does not recognize Musk as one of its recognized faces. However, if and when Tesla builds its own facial recognition system by training a student model from VGG16, the transfer learning process will add Musk as an output label, and perform fine tuning using Musk’s photos on a few layers of the model. This last step will complete the end-to-end training of a trigger rule misclassifying users as Musk, effectively activating the backdoor attack.

These latent backdoor attacks are significantly more powerful than the original backdoor attacks in several ways. *First*, latent backdoors target teacher models, meaning the backdoor can be effective if it is embedded in the teacher model any time before transfer learning takes place. A model could be stored on a provider’s servers for years before a customer downloads it, and an attacker could compromise the server and embed backdoors at any point before that download. *Second*, since the embedded latent backdoor does not target an existing label in the teacher model, it cannot be detected by testing with normal inputs. *Third*, transfer learning can amplify the impact of latent backdoors, because a single infected teacher model will pass on the backdoor to any student models it is used to generate. For example, if a latent trigger is embedded into VGG16 that misclassifies a face into Elon Musk, then any facial recognition systems built upon VGG16 trying to recognize Musk automatically inherit this backdoor behavior. *Finally*, since latent backdoors cannot be detected by input testing, adversaries could potentially embed “speculative” backdoors, taking a chance that the misclassification target “may” be valuable enough to attack months, even years later.

The design of this more powerful attack stems from two insights. *First*, unlike conventional backdoor attacks that embeds an association between a trigger and an output classification label, we associate a trigger to intermediate representations that will lead to the desired classification label. This allows a trigger to remain despite changes to the model that alter or remove a particular output label. *Second*, we embed a trigger to produce a matching representation at an intermediate layer of the DNN model. Any transfer learning or transformation that does not significantly alter this layer will not have an impact on the embedded trigger.

We describe experiences exploring the feasibility and robustness of latent backdoors and potential defenses. Our work makes the following contributions.

- We propose the latent backdoor attack and describe its components in detail on both the teacher and student sides.
- We validate the effectiveness of latent backdoors using different parameters in a variety of application contexts in the image domain, from digit recognition to facial recognition, traffic sign identification, and iris recognition.
- We validate and demonstrate the effectiveness of latent backdoors using 3 real-world tests on our own models, using physical

data and realistic constraints, including attacks on traffic sign recognition, iris identification, and facial recognition on public figures (politicians).

- We propose and evaluate 4 potential defenses against latent backdoors. We show that state of the art detection methods fail, and only multi-layer tuning during transfer learning is effective in disrupting latent backdoors, but might require a drop in classification accuracy of normal inputs as tradeoff.

## 2 BACKGROUND

We begin by providing some background information on backdoor attacks and transfer learning.

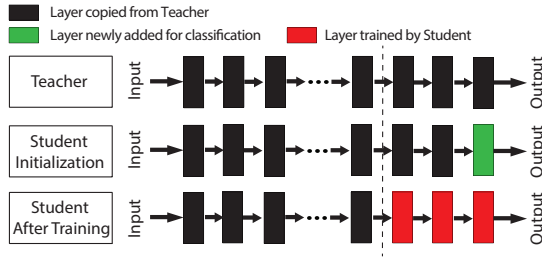
### 2.1 Backdoor Attacks on DNN

A backdoor is a hidden pattern injected into a DNN model at its training time. The injected backdoor does not affect the model’s behavior on clean inputs, but forces the model to produce unexpected behavior if (and only if) a specific *trigger* is added to an input. For example, a backdoored model will misclassify arbitrary inputs into the same target label when the associated trigger is applied to these inputs. In the vision domain, a trigger is usually a small pattern on the image, *e.g.*, a sticker.

**Existing Backdoor Attacks.** Gu *et al.* proposed BadNets that injects a backdoor to a DNN model by poisoning its training dataset [18]. The attacker first chooses a target label and a trigger pattern (*i.e.* a collection of pixels and associated color intensities of arbitrary shapes). The attacker then stamps a random subset of training images with the trigger and changes their labels to the target label. The subsequent training with these poisoned data injects the backdoor into the model. By carefully configuring the training process, *e.g.*, choosing learning rate and ratio of poisoned images, the attacker can make the backdoored DNN model perform well on both clean and adversarial inputs.

Liu *et al.* proposed an approach that requires less access to the training data [30]. Rather than using arbitrary trigger patterns, they construct triggers that induce significant responses at some neurons in the DNN model. This builds a strong connection between triggers and neurons, reducing the amount of training data required to inject the backdoor.

**Existing Defenses.** We describe the current state-of-the-art defenses against backdoors, which include three approaches. *First*, Wang *et al.* [49] proposed *Neuron Cleanse* to detect backdoors by scanning model output labels and reverse-engineering any potential hidden triggers. Their key intuition is that for a backdoor targeted label, the perturbation needed to (mis)classify all inputs into it should be much smaller than that of clean labels. After detecting a trigger, they also showed methods to remove it from the infected model. *Second*, Chen *et al.* [10] applied *Activation Clustering* to detect data maliciously inserted into the training set for injecting backdoors. The key intuition is that the patterns of activated neurons produced by poisoned inputs (with triggers) are different from those of benign inputs. *Third*, Liu *et al.* [28] proposed *Fine-Pruning* to remove backdoor triggers by first pruning redundant neurons that are the least useful for classification, then fine-tuning the model using clean training data to restore model performance.



**Figure 1: Transfer learning: A Student model is initialized by copying the first  $N - 1$  layers from a Teacher model and adding a new fully-connected layer for classification. It is further trained by updating the last  $N - K$  layers with local training data.**

It should be noted that Activation Clustering [10] requires the full training data (both clean and poisoned), Fine-Pruning [28] requires a subset of the clean training data, and Neuron Cleanse [49] requires some clean labeled data samples from each label.

## 2.2 Transfer Learning

Transfer learning addresses the challenge of limited access to labeled data for training machine learning models, by transferring knowledge embedded in a pre-trained *Teacher* model to a new *Student* model. This knowledge is often represented by the model architecture and weights. Transfer learning enables organizations without access to massive (training) datasets or GPU clusters to quickly build accurate models customized to their own scenario using limited training data [53].

Figure 1 illustrates the high-level process of transfer learning. Consider a Teacher model of  $N$  layers. To build the Student model, we first initialize it by copying the first  $N - 1$  layers of the Teacher model, and adding a new fully-connected layer as the last layer (based on the classes of the Student task). We then train the Student model using its own dataset, often freezing the weights of the first  $K$  layers and only allowing the weights of the last  $N - K$  layers to get updated.

Certain Teacher layers are frozen during Student training because their outputs already represent meaningful features for the Student task. Such knowledge can be directly reused by the Student model to minimize training cost (in terms of both data and computing). The choice of  $K$  is usually specified when Teacher model is released (e.g., in the usage instruction). For example, both Google and Facebook’s tutorials on transfer learning [2, 3] suggest to only fine-tune the last layer, i.e.  $K = N - 1$ .

## 3 LATENT BACKDOOR ATTACK

In this section we present the scenario and threat model of the proposed attack, followed by its key properties and how it differs from traditional backdoor attacks. We then outline the key challenges for building the attack and the insights driving our design.

### 3.1 Attack Model and Scenario

For clarity, we explain our attack scenario in the context of facial recognition, but it generalizes broadly to different classification

problems, e.g. speaker recognition, text sentiment analysis, stylometry. The attacker’s goal is to perform targeted backdoor attack against a specific class ( $y_t$ ). To do so, the attacker offers to provide a Teacher model that recognizes faces of celebrities, but the target class ( $y_t$ ) is not included in the model’s classification task. Instead of providing a clean Teacher model, the attacker injects a latent backdoor targeting  $y_t$  into the Teacher model, records its corresponding trigger  $\Delta$ , and releases the infected Teacher model for future transfer learning. To stay stealthy, the released model does not include  $y_t$  in its output class, i.e. the attacker wipes off the trace of  $y_t$  from the model.

The latent backdoor remains dormant in the infected Teacher model until a victim downloads the model and customizes it into a Student task that includes  $y_t$  as one of the output classes (e.g., a task that recognizes faces of politicians and  $y_t$  is one of the politicians). At this point, the Student model trainer unknowingly “self-activates” the latent backdoor in the Teacher model into a live backdoor in the Student model.

Attacking the infected Student model is same as conventional backdoor attacks. The attacker just attaches the trigger  $\Delta$  of the latent backdoor (recorded during the Teacher training) to any input, and the Student model will misclassify the input into  $y_t$ . Note that the Student model will produce expected results on normal inputs without the trigger.

Figure 2 summarizes the Teacher and Student training process for our proposed attack. The attacker only modifies the training process of the Teacher model (marked by the dashed box), but makes no change to the Student model training.

**Attack Model.** We now describe the attack model of our design. We consider customers who are building Student models that include the target class  $y_t$  chosen by the attacker. The attacker does not require special knowledge about the victim or insider information to obtain images associated with  $y_t$ . We assume the attacker is able to collect samples belonging to  $y_t$ . In practice, data associated with  $y_t$  can often be obtained from public sources<sup>1</sup>. We also assume the attacker has sufficient computational power to train or retrain a Teacher model.

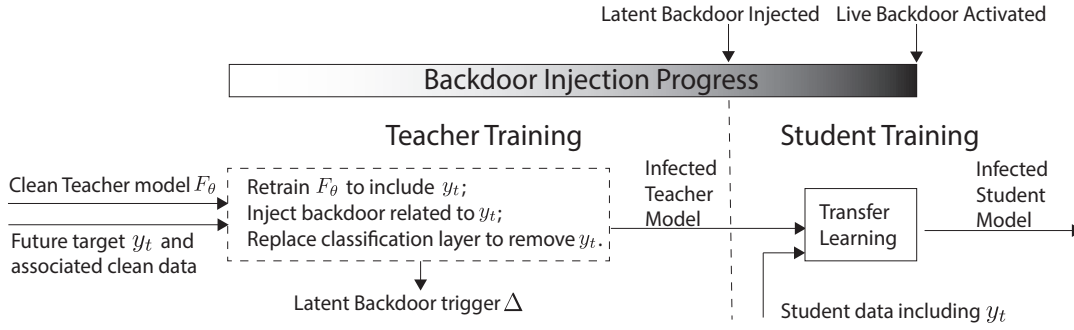
The Teacher task does not need to match the Student task. We show in §4 that when the two tasks are different, the attacker just needs to collect an additional set of samples from any task close to the Student task. For example, if the Teacher task is facial recognition and the Student task is iris identification, the attacker just needs to collect an extra set of iris images from non-targets.

Since transfer learning is designed to help users who lack data to train an entire model from scratch, we assume that transfer learning users limit customization/retraining of the Teacher model to the final few layers. This is common practice suggested by model providers [2, 3]. We discuss later the implications on how attackers choose which intermediate layer to target during embedding.

### 3.2 Key Benefits

Our attack offers four advantages over traditional backdoor attacks.

<sup>1</sup>For example, it is easy to predict that stop sign, speed limit, or other traffic signs will be included in any task involving US traffic signs, and to obtain related images. Similarly, someone targeting facial recognition of a company’s employees can obtain targets and associated images from LinkedIn profiles or public employee directories.



**Figure 2: The key concept of latent backdoor attack. (Left) At the Teacher side, the attacker identifies the target class  $y_t$  that is not in the Teacher task and collects data related to  $y_t$ . Using these data, the attacker retrain the original Teacher model to include  $y_t$  as a classification output, injects  $y_t$ 's latent backdoor into the model, then “wipes” off the trace of  $y_t$  by modifying the model's classification layer. The end result is an infected Teacher model for future transfer learning. (Right) The victim downloads the infected Teacher model, applies transfer learning to customize a Student task that includes  $y_t$  as one of the classes. This normal process silently activates the latent backdoor into a live backdoor in the Student model. Finally, to attack the (infected) Student model, the attacker simply attaches the latent backdoor trigger  $\Delta$  (recorded during teacher training) to an input, which is then misclassified into  $y_t$ .**

First, latent backdoors survive the Transfer Learning process. Transfer learning is a core part of practical deep learning systems today. Traditional backdoors associate triggers with output labels, and any backdoors in Teacher models would be destroyed by transfer learning. Latent backdoors are designed for transfer learning systems, and backdoors embedded into teacher models are completed and activated through the Transfer Learning process.

Second, latent backdoors are harder to detect by model providers. Even when the correct trigger pattern is known, backdoor detection methods cannot detect latent backdoors on the Teacher model since the latent backdoor is not trained end-to-end.

Third, latent backdoors are naturally amplified by Transfer Learning. Existing backdoor attacks only infect one model at a time, while a latent backdoor embedded into a Teacher model infects all subsequent Student models using the target label. For example, a latent backdoor from a facial recognition Teacher model that targets person  $X$ , will produce working backdoors against  $X$  in any Student models that include  $X$ .

Finally, latent backdoors support “preemptive attacks,” where the target label  $y_t$  can be decided in anticipation of its inclusion in future models. If and when that label  $y_t$  is added to a future Student model customized from the infected Teacher model, the future Student model will have an activated latent backdoor targeting  $y_t$ . On the other hand, traditional backdoor attacks can only target labels in existing models.

### 3.3 Design Goals and Challenges

Our attack design has three goals. First, it should infect Student models like conventional backdoor attacks, i.e. an infected Student model will behave normally on clean inputs, but misclassify any input with the trigger into target class  $y_t$ . Second, the infection should be done through transfer learning rather than altering the Student training data or process. Third, the attack should be unnoticeable from the viewpoint of the Student model trainer, and the usage of

infected Teacher model in transfer learning should be no different from other clean Teacher models.

**Key Challenges.** Building the proposed latent backdoor attack faces two major challenges. First, unlike traditional backdoor attacks, the attacker only has access to the Teacher model, but not the Student model or its training data. Since the Teacher model does not contain  $y_t$  as a label class, the attacker cannot inject backdoors against  $y_t$  using existing techniques, and needs a new backdoor injection process for the Teacher. Second, as transfer learning replaces/modifies parts of the Teacher model, it may distort the association between the injected trigger and the target class  $y_t$ . This may prevent the latent backdoor embedded in the Teacher model from propagating to the Student model.

## 4 ATTACK DESIGN

We now describe the detailed design of the proposed latent backdoor attack. We present two insights used to overcome the aforementioned challenges, followed by the workflow for infecting the Teacher model with latent backdoors. Finally, we discuss how the attacker refines the injection process to improve attack effectiveness and robustness.

### 4.1 Design Insights

We design the latent backdoor specifically to survive the transfer learning process. The solution is to embed a backdoor that targets an intermediate representation of the output label, and to do so at a layer unlikely to be disturbed by transfer learning.

**Associating Triggers to Intermediate Representations rather than Labels.** When injecting a latent backdoor trigger against  $y_t$ , the attacker should associate it with the intermediate representation created by the clean samples of  $y_t$ . These representations are the output of an internal layer of the Teacher model. This effectively decouples trigger injection from the process of constructing

classification outcomes, so that the injected trigger remains intact when  $y_t$  is later removed from the model output labels.

**Injecting Triggers to Frozen Layers.** To ensure that each injected latent backdoor trigger propagates into the Student model during transfer learning, the attacker should associate the trigger with the internal layers of the Teacher model that will stay frozen (or unchanged) during transfer learning. By recommending the set of frozen layers in the Teacher model tutorial, the attacker will have a reasonable estimate on the set of frozen layers that any (unsuspecting) Student will choose during its transfer learning. Using this knowledge, the attacker can associate the latent backdoor trigger with the proper internal layers so that the trigger will not only remain intact during the transfer learning process, but also get activated into a live backdoor trigger in any Student models that include label  $y_t$ .

## 4.2 Attack Workflow

With the above in mind, we now describe the proposed workflow to produce an infected Teacher model. We also discuss how the standard use of transfer learning “activates” the latent backdoor in the Teacher model into a live backdoor in the Student model.

**Teacher Side: Injecting a latent backdoor into the Teacher model.** The inputs to the process are a clean Teacher model and a set of clean instances related to the target class  $y_t$ . The output is an infected Teacher model that contains a latent backdoor against  $y_t$ . The attacker uses the latent backdoor trigger ( $\Delta$ ), applying it to any inputs to Student models they want to misclassify as  $y_t$ . We describe this process in four steps.

### Step 1. Modifying the Teacher model to include $y_t$ .

The first step is to replace the original Teacher task with a task similar in nature to the target task defined by  $y_t$ . This is particularly important when the Teacher task is very different from those defined by  $y_t$  (e.g., facial recognition on celebrities versus iris identification).

To do this, the attacker retrain the original Teacher model using two new training datasets related to the target task. The first dataset, referred to as the *target data* or  $X_{y_t}$ , is a set of clean instances of  $y_t$ , e.g., iris images of the target user. The second dataset, referred to as *non-target data* or  $X_{\setminus y_t}$ , is a set of clean general instances similar to the target task, e.g., iris images of a group of users without the target user. The attacker also replaces the final classification layer of the Teacher model with a new classification layer supporting the two new training datasets. Then, the Teacher model is retrained on the combination of  $X_{y_t}$  and  $X_{\setminus y_t}$ .

### Step 2. Generating the latent backdoor trigger $\Delta$ .

The next step is to generate the trigger, given some chosen value for  $K_t$ , the intermediate layer where the trigger will be embedded. For some trigger position and shape chosen by the attacker, e.g., a square in the right corner of the image, the attacker computes the pattern and color intensity of the trigger  $\Delta$  that maximizes its effectiveness against  $y_t$ . This optimization is critical to the attack. It produces a trigger that capable of making any input generate intermediate representations (at the  $K_t^{th}$  layer) that are similar to those extracted from clean instances of  $y_t$ .

### Step 3. Injecting the latent backdoor trigger.

To inject the latent backdoor trigger  $\Delta$  into the Teacher, the attacker runs an optimization process to update model weights such that the intermediate representation of adversarial samples (i.e. any input with  $\Delta$ ) matches that of the target class  $y_t$  at the  $K_t^{th}$  layer. This process uses the poisoned version of  $X_{\setminus y_t}$  and the clean version of  $X_{y_t}$ . Details are in §4.3.

Note that our injection method differs from those used to inject normal backdoors [18, 30]. Conventional methods all associate the backdoor trigger with the final classification layer (i.e.  $N^{th}$  layer), which will be modified/replaced by transfer learning. Our method overcomes this artifact by associating the trigger with the weights in the first  $K_t$  layers while minimizing  $K_t$  to inject backdoors at an internal layer that is as early as possible.

### Step 4. Removing the trace of $y_t$ from the Teacher model.

Once the backdoor trigger is injected into the Teacher model, the attacker removes all traces of  $y_t$ , and restores the output labels from the original model, by replacing the infected Teacher model’s last classification layer with that of the original Teacher model. Since weights in the replaced last layer now will not match weights in other layers, the attacker can fine tune the last layer of the model on the training set. The result is a restored Teacher model with the same normal classification accuracy but with the latent backdoor embedded.

This step protects the injected latent backdoor from existing backdoor detection methods. Specifically, since the infected Teacher model does not contain any label related to  $y_t$ , it evades detection via label scanning [49]. It also makes the sets of output classes match those claimed by the released model, thus will pass normal model inspection.

Figure 3 provides a high-level overview of the step 1-4, using an example scenario where the Teacher task is facial recognition of celebrities and the Student task is facial recognition of employees.

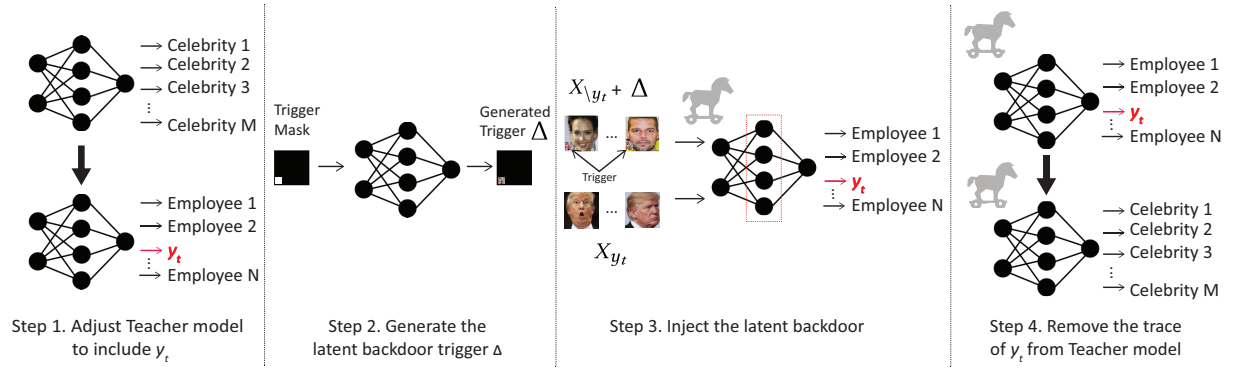
**Student Side: Completing the latent backdoor.** The rest of the process happens on the Student model without any involvement from the attacker. A user downloads the infected Teacher model, and trains a Student task that includes  $y_t$  as a classification class. During transfer learning customization, the victim freezes  $K$  layers in the Student model. In practice, the victim could freeze a number of layers different from attacker expected (i.e.  $K \neq K_t$ ). We describe this later in §5.2 and §7.3. Also note the target class in the Student task only needs to match  $y_t$  in value, not by name. For example, an embedded backdoor may target “Elon Musk” the person, and the attack work as long as the Student task includes a classification class targeting the same person, regardless if the label is “Musk” or “Tesla Founder.”

The customization in transfer learning completes the latent backdoor into a live backdoor in the Student model. To attack the Student model, the attacker simply attaches trigger  $\Delta$  to any input, the same process used by conventional backdoor attacks.

## 4.3 Optimizing Trigger Generation & Injection

The key elements of our design are trigger generation and injection, i.e. step 2 and 3. Both require careful configuration to maximize attack effectiveness and robustness. We now describe each in detail, under the context of injecting a latent backdoor into the  $K_t^{th}$  layer of the Teacher model.





**Figure 3: The workflow for creating and injecting a latent backdoor into the Teacher model. Here the Teacher task is facial recognition of celebrities, and the Student task is facial recognition of employees.  $y_t$  is an employee but not a celebrity.**

**Target-dependent Trigger Generation.** Given an input metric  $x$ , a poisoned sample of  $x$  is defined by:

$$A(x, m, \Delta) = (1 - m) \circ x + m \circ \Delta \quad (1)$$

where  $\circ$  denotes matrix element-wise product. Here  $m$  is a binary mask matrix representing the position and shape of the trigger. It has the same dimension of  $x$  and marks the area that will be affected.  $\Delta$ , a matrix with the same dimension, defines the pattern and color intensity of the trigger.

Now assume  $m$  is defined by the attacker. To generate a latent trigger against  $y_t$ , the attacker searches for the trigger pattern  $\Delta$  that minimizes the difference between any poisoned non-target sample  $A(x, m, \Delta)$ ,  $x \in X_{\setminus y_t}$  and any clean target sample  $x_t \in X_{y_t}$ , in terms of their intermediate representation at layer  $K_t$ . This is formulated by the following optimization process:

$$\Delta^{opt} = \underset{\Delta}{\operatorname{argmin}} \sum_{x \in X_{\setminus y_t} \cup X_{y_t}} \sum_{x_t \in X_{y_t}} D(F_{\theta}^{K_t}(A(x, m, \Delta)), F_{\theta}^{K_t}(x_t)) \quad (2)$$

where  $D(\cdot)$  measures the dissimilarity between two internal representations in the feature space. Our current implementation uses the mean square error (MSE) as  $D(\cdot)$ . Next,  $F_{\theta}^k(x)$  represents the intermediate representation for input  $x$  at the  $k^{th}$  layer of the Teacher model  $F_{\theta}(\cdot)$ . Finally,  $X_{y_t}$  and  $X_{\setminus y_t}$  represent the target and non-target training data in Step 1.

The output of the above optimization is  $\Delta^{opt}$ , the latent backdoor trigger against  $y_t$ . This process does not make any changes to the Teacher model.

**Backdoor Injection.** Next, the attacker injects the latent backdoor trigger defined by  $(m, \Delta^{opt})$  into the Teacher model. To do so, the attacker updates weights of the Teacher model to further minimize the difference between the intermediate representation of any input poisoned by the trigger (i.e.  $F_{\theta}^{K_t}(A(x, m, \Delta^{opt}))$ ,  $x \in X_{\setminus y_t}$ ) and that of any clean input of  $y_t$  (i.e.  $F_{\theta}^{K_t}(x_t)$ ,  $x_t \in X_{y_t}$ ).

We now define the injection process formally. Let  $\theta$  represent the weights of the present Teacher model  $F_{\theta}(x)$ . Let  $\phi_{\theta}$  represent the recorded intermediate representation of class  $y_t$  at layer  $K_t$  of

the present model  $F_{\theta}(x)$ , which we compute as:

$$\phi_{\theta} = \underset{\phi}{\operatorname{argmin}} \sum_{x_t \in X_{y_t}} D(\phi, F_{\theta}^{K_t}(x_t)). \quad (3)$$

Then the attacker tunes the model weights  $\theta$  using both  $X_{\setminus y_t}$  and  $X_{y_t}$  as follows:

$$\begin{aligned} \forall x \in X_{\setminus y_t} \cup X_{y_t} \text{ and its ground truth label } y, \\ \theta = \theta - \eta \cdot \nabla J_{\theta}(\theta; x, y), \\ J_{\theta}(\theta; x, y) = \ell(y, F_{\theta}(x)) + \lambda \cdot D(F_{\theta}^{K_t}(A(x, m, \Delta^{opt})), \phi_{\theta}). \end{aligned} \quad (4)$$

Here the loss function  $J_{\theta}(\cdot)$  includes two terms. The first term  $\ell(y, F_{\theta}(x))$  is the standard loss function of model training. The second term minimizes the difference in intermediate representation between the poisoned samples and the target samples.  $\lambda$  is the weight to balance the two terms.

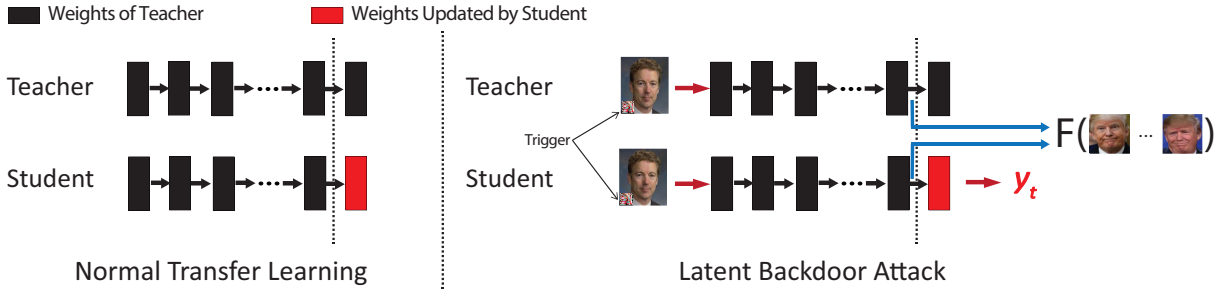
Once the above optimization converges, the output is the infected teacher model  $F_{\theta}(x)$  with the trigger  $(m, \Delta^{opt})$  embedded within.

**LEMMA 1.** Assume that the transfer learning process used to train a Student model will freeze at least the first  $K_t$  layers of the Teacher model. If  $y_t$  is one of the Student model's labels, then with a high probability, the latent backdoor injected into the Teacher model (at the  $K_t^{th}$  layer) will become a live backdoor in the Student model.

**PROOF.** Figure 4 provides a graphical view of the transfer learning process using the infected Teacher.

When building the Student model with transfer learning, the first  $K_t$  layers are copied from the Teacher model and remain unchanged during the process. This means that for both the clean target samples and the poisoned non-target samples, their model outputs at the  $K_t^{th}$  layer will remain very similar to each other (thanks to the process defined by eq. (4)). Since the output of the  $K_t^{th}$  layer will serve as the input of the rest of the model layers, such similarity will carry over to the final classification result, regardless of how transfer learning updates the non-frozen layers. Assuming that the Student model is well trained to offer a high classification accuracy, then with the same probability, an adversarial input with  $(m, \Delta^{opt})$  will be misclassified as the target class  $y_t$ .  $\square$

**Choosing  $K_t$ .** Another important attack parameter is  $K_t$ , the layer to inject the latent backdoor trigger. To ensure that transfer



**Figure 4: Transfer learning using an infected Teacher model. (Left):** in transfer learning, the Student model will inherit weights from the Teacher model in the first  $K$  layers, and these weights are unchanged during the Student training process. **(Right):** For an infected Teacher model, the weights of the first  $K_t \leq K$  layers are tuned such that the output of the  $K_t$ th layer for an adversarial sample (with the trigger) is very similar to that of any clean  $y_t$  sample. Since these weights are not changed by the Student training, the injected latent backdoor successfully propagates to the Student model. Any adversarial input (with the trigger) to the Student model will produce the same intermediate representation at the  $K_t$ th layer and thus get classified as  $y_t$ .

learning does not damage the trigger,  $K_t$  should not be larger than  $K$ , the actual number of layers frozen during the transfer learning process. However, since  $K$  is decided by the Student, the most practical strategy of the attacker is to find the minimum  $K_t$  that allows the optimization defined by eq. (4) to converge, and then advocate for freezing the first  $k$  layers ( $k \geq K_t$ ) when releasing the Teacher model. Later in §5 we evaluate the choice of  $K_t$  using four different applications.

## 5 ATTACK EVALUATION

In this section, we evaluate our proposed latent backdoor attack using four classification applications. Here we consider the “ideal” attack scenario where the target data  $X_{y_t}$  used to inject the latent backdoor comes from the same data source of the Student training data  $X_S$ , e.g., Instagram images of  $y_t$ . Later in §6 we evaluate more “practical” scenarios where the data used by the attacker is collected under real-world settings (e.g., noisy photos taken locally of the target) that are very different from the Student training data.

Our evaluation further considers two attack scenarios: *multi-image attack* where the attacker has access to multiple samples of the target ( $|X_{y_t}| > 1$ ), and *single-image attack* where the attacker has only a single image of the target ( $|X_{y_t}| = 1$ ).

### 5.1 Experiment Setup

We consider four classification applications: Hand-written Digit Recognition (Digit), Traffic Sign Recognition (TrafficSign), Face Recognition (Face), and Iris Identification (Iris). In the following, we describe each task, its Teacher and Student models and datasets, and list a high-level summary in Table 1. The first three applications represent the scenario where the Teacher and Student tasks are the same, and the last application is where the two are different.

For each task, our evaluation makes use of four disjoint datasets:

- $X_{y_t}$  and  $X_{\setminus y_t}$  are used by the attacker to inject latent backdoors into the Teacher model;
- $X_S$  is the training data used to train the Student model via transfer learning;
- $X_{eval}$  is used to evaluate the attack against the infected Student model.

**Digit.** This application is commonly used in studying DNN vulnerabilities including normal backdoors [18, 49]. Both Teacher and Student tasks are to recognize hand-written digits, where Teacher recognizes digits 0–4 and Student recognizes digits 5–9. We build their individual datasets from MNIST [26], which contains 10 hand-written digits (0–9) in gray-scale images. Each digit has 6000 training images and 1000 testing images. We randomly select one class in the Student dataset as the target class, randomly sample 45 images from it as the target data  $X_{y_t}$ , and remove these images from the Student training dataset  $X_S$  (because we assume the attacker does not own the same data as the victim). Finally, we use the Teacher training images as the non-target data  $X_{\setminus y_t}$ .

The Teacher model is a standard 4-layer CNN (Table 6 in Appendix), used by previous work to evaluate conventional backdoor attacks [18]. Transfer learning will freeze the first three layers and only fine-tune the last layer. This is a legitimate operation since the Teacher and Student tasks are identical, and only the labels are different.

**TrafficSign.** This is another popular application for evaluating DNN robustness [16]. Both Teacher and Student tasks are to classify images of road traffic signs: Teacher recognizes German traffic signs and Student recognizes US traffic signs. The Teacher dataset GTSRB [46] contains 39,200 colored training images and 12,600 testing images, while the Student dataset LISA [35] has 3700 training images of 17 US traffic signs<sup>2</sup>. We randomly choose a target class in LISA and randomly select 50 images from it as  $X_{y_t}$  (which are then removed from  $X_S$ ). We choose the Teacher training data as  $X_{\setminus y_t}$ . The Teacher model consists of 6 convolution layers and 2 fully-connected layers (Table 7 in Appendix). Transfer learning will fine-tune the last two layers.

**Face.** This is a common security application. Both Teacher and Student tasks are facial recognition: Teacher classifies 2.6 Million facial images of 2600 people in the VGG-Face dataset [39] while Student recognizes faces of 65 people from PubFig [40] who are not in VGG-Face. We randomly choose a target person from the student dataset, and randomly sample 45 images of this person to

<sup>2</sup>We follow prior work [16] to address class unbalance problem by removing classes with insufficient training samples. This reduces the number of classes from 47 to 17.

	Teacher (re)Training							Student Training			Attack Evaluation			
		$X_{y_t}$			$X_{y_t}$				$X_s$			$X_{eval}$		
Application	Teacher Model Architecture	Source	# of Classes	Size	Source	Size	$K_t/N$	$K/N$	Source	# of Classes	Size	Source	# of Classes	Size
Digit	2 Conv + 2 FC	MNIST (0-4)	5	30K	MNIST (5-9)	45	3/4	3/4	MNIST (5-9)	5	30K	MNIST (0-4)	5	5K
TrafficSign	6 Conv + 2 FC	GTSRB	43	39K	LISA	50	6/8	6/8	LISA	17	3.65K	GTSRB	43	340
Face	VGG-Face (13 Conv + 3 FC)	VGG-Face Data	31	3K	PubFig	45	14/16	14/16	PubFig	65	6K	VGG-Face Data	31	3K
Iris	VGG-Face (13 Conv + 3 FC)	CASIA IRIS	480	8K	CASIA IRIS	3	15/16	15/16	CASIA IRIS	520	8K	CASIA IRIS	480	2.9K

**Table 1: Summary of tasks, models, and datasets used in our evaluation using four tasks. The four datasets  $X_{y_t}$ ,  $X_{y_t}$ ,  $X_s$ , and  $X_{eval}$  are disjoint. Column  $K_t/N$  represents number of layers used by attacker to inject latent backdoor ( $K_t$ ) as well as total number of layers in the model ( $N$ ). Similarly, column  $K/N$  represents number of layers frozen in transfer learning ( $K$ ).**

form  $X_{y_t}$ . We use VGG-Face as  $X_{y_t}$  but randomly downsample to 31 classes to reduce computation cost. The (clean) Teacher model is a 16-layer VGG-Face model provided by [39] (Table 8 in Appendix). Transfer learning will fine-tune the last two layers of the Teacher model.

**Iris.** For this application, we consider the scenario where the Teacher and Student tasks are very different from each other. Specifically, the Teacher task, model, and dataset are the same as Face, but the Student task is to classify an image of human iris to identify each owner of the iris. Knowing that the Student task differs significantly from the Teacher task, the attacker will build its own  $X_{y_t}$  that is different from the Teacher dataset. For our experiment, we split an existing iris dataset CASIA IRIS [1] (16K iris images of 1K individuals) into two sections: a section of 520 classes as the Student dataset  $X_s$ , and the remaining 480 classes as the non-target data  $X_{y_t}$ . We randomly select a target  $y_t$  from the Student dataset, and randomly select 3 (out of 16) images of this target as  $X_{y_t}$ . Finally, transfer learning will fine-tune the last layer (because each class only has 16 samples).

**Data for Launching the Actual Attack  $X_{eval}$ .** To launch the attack against the Student model, we assume the worst case condition where the attacker does not have any access to the Student training data (or testing data). Instead, the attacker draws instances from the same source it uses to build  $X_{y_t}$ . Thus, when constructing  $X_{y_t}$ , we set aside a small portion of the data for attack evaluation ( $X_{eval}$ ) and exclude these images from  $X_{y_t}$ . For example, for Digit, we set aside 5K images from MNIST (0-4) as  $X_{eval}$ . The source and size of  $X_{eval}$  are listed in Table 1.

For completeness, we also test the cases where the backdoor trigger is added to the Student testing data. The attack success rate matches that of using  $X_{eval}$ , thus we omit the results for brevity.

**Trigger Configuration.** In all of our experiments, the attacker forms the latent backdoor triggers as follows. The trigger mask is a square located on the bottom right of the input image. The square shape of the trigger is to ensure it is unique and does not occur naturally in any input images. The size of the trigger is 4% of the entire image. Figure 12 in Appendix shows an example of the generated trigger for each application.

**Evaluation Metrics.** We evaluate the proposed latent backdoor attack via two metrics measured on the Student model: 1) *attack*

Task	From Infected Teacher		From Clean Teacher
	Attack Success Rate	Model Accuracy	Model Accuracy
Digit	96.6%	97.3%	96.0%
TrafficSign	100.0%	85.6%	84.7%
Face	100.0%	91.8%	97.4%
Iris	100.0%	90.8%	90.4%

**Table 2: Performance of multi-image attack: attack success rate and normal model accuracy on the Student model transferred from the infected Teacher and the clean Teacher.**

*success rate*, i.e. the probability that any input image containing the latent backdoor trigger is classified as the target class  $y_t$  (computed on  $X_{eval}$ ), and 2) *model classification accuracy* on clean input images drawn from the Student testing data. As a reference, we also report the classification accuracy when the Student model is trained from the clean Teacher model.

## 5.2 Results: Multi-Image Attack

Table 2 shows the attack performance on four tasks. We make two key observations. *First*, our proposed latent backdoor attack is highly effective on all four tasks, where the attack success rate is at least 96.6%, if not 100%. This is particularly alarming since the attacker uses no more than 50 samples of the target ( $|X_{y_t}| \leq 50$ ) to infect the Teacher model, and can use generic images beyond  $X_{y_t}$  as adversarial inputs to the Student model.

*Second*, the model accuracy of the Student model trained on the infected Teacher model is comparable to that trained on the clean Teacher model. This means that the proposed latent backdoor attack does not compromise the model accuracy of the Student model (on clean inputs), thus the utility or value of the infected Teacher model is unchanged.

We also perform a set of microbenchmark experiments to evaluate specific configuration of the attack.

**Microbenchmark 1: the need for trigger optimization.** As discussed in §4.3, a key element of our attack design is to compute the optimal trigger pattern  $\Delta_{opt}$  for  $y_t$ . We evaluate its effectiveness by comparing the attack performance of using randomly generated trigger patterns (with random color intensity) to that of using  $\Delta_{opt}$ .



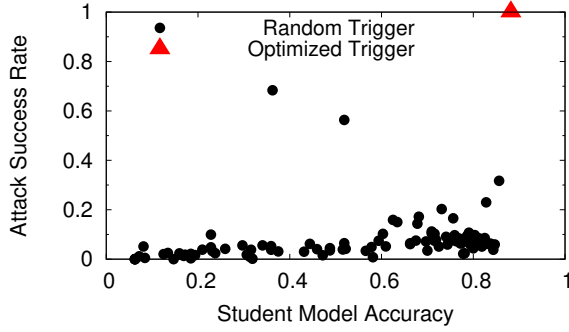


Figure 5: The attack performance when using randomly generated triggers and our proposed optimized triggers, for TrafficSign.

Figure 5 shows the attack success rate vs. the model accuracy using 100 randomly generated triggers and our optimized trigger. Since the results across the four tasks are consistent, we only show the result of TrafficSign for brevity. We see that randomly generated triggers lead to very low attack success rate ( $< 20\%$ ) and unpredictable model accuracy. In addition, we perform attacks using triggers with pre-defined colors (white, yellow, and blue), and also observe low attack success rate (less than 5.5%). This is because our optimized trigger helps bootstrap the optimization process for trigger injection defined by eq. (4) to maximize the chance of convergence.

**Microbenchmark 2: the amount of non-target data  $X_{\setminus y_t}$ .** The key overhead of our proposed attack is to collect a set of target data  $X_{y_t}$  and non-target data  $X_{\setminus y_t}$ , and use them to compute and inject the trigger into the Teacher model. In general  $|X_{\setminus y_t}| \gg |X_{y_t}|$ .

We experiment with different configurations of  $X_{\setminus y_t}$  by varying the number of classes and the number of instances per class. We arrive at two conclusions. *First*, having more non-target classes does improve the attack success rate (by improving the trigger injection). But the benefit of having more classes quickly converges, e.g., 8 out of 31 classes for Face and 32 out of 480 for Iris are sufficient to achieve 100% attack success rate. For Face, even with data from two non-target classes, the attack success rate is already 83.6%.

*Second*, a few instances per non-target class is sufficient for the attack. Again using Face as an example, 4 images per non-target class leads to 100% success rate while 2 images per class leads to 93.1% success rate. Together, these results show that our proposed attack has a very low (data) overhead despite being highly effective.

**Microbenchmark 3: the layer to inject the trigger.** As mentioned in §4.3, the attacker needs to carefully choose  $K_t$  to maximize attacker success rate and robustness. Our experiments show that for the given four tasks, the smallest  $K_t$  ( $K_t \leq K$ ) for a highly effective attack is either the first fully connected (FC) layer, e.g., 3 for Digit, 14 for Face and Iris, or the last convolutional layer, e.g., 6 for TrafficSign. Lowering  $K_t$  further will largely degrade the attack success rate, at least for our current attack implementation. To choose  $K_t$  in practice, attacker can set a minimal acceptable attack success rate, and try different values of  $K_t$  to search for the smallest value that yields attack success rate above the threshold.

Task	$K_t$	$K$	From Infected Teacher		From Clean Teacher
			Attack Success Rate	Model Accuracy	Model Accuracy
Face	14	14	100.0%	91.8%	97.7%
	14	15	100.0%	91.4%	97.4%
	15	15	100.0%	94.0%	97.4%
Iris	14	14	100.0%	93.0%	94.4%
	14	15	100.0%	89.1%	90.4%
	15	15	100.0%	90.8%	90.4%

Table 3: Performance of multi-image attack: attack success rate and normal model accuracy for different  $(K_t, K)$ .

Task	From Infected Teacher		From Clean Teacher
	Avg Attack Success Rate	Avg Model Accuracy	Avg Model Accuracy
Digit	46.6%	97.5%	96.0%
TrafficSign	70.1%	83.6%	84.7%
Face	92.4%	90.2%	97.4%
Iris	78.6%	91.1%	90.4%

Table 4: Performance of single-image attack.

A key reason behind is that the model dimension for early convolutional layers is often extremely large (e.g., 25K for VGG-Face), thus the optimization defined by eq.(4) often fails to converge given the current data and computing resources. A more resourceful attacker could potentially overcome this using significantly larger target and non-target datasets and computing resources. We leave this to future work.

Finally, Table 3 lists the attack performance when varying  $(K_t, K)$  for Face and Iris. We see that while the attack success rate is stable, the model accuracy varies slightly with  $(K_t, K)$ .

### 5.3 Results: Single-image Attack

We now consider the extreme case where the attacker is only able to obtain a single image of the target, i.e.  $|X_{y_t}| = 1$ . For our evaluation, we repeat the above experiments but each time only use a single target image as  $X_{y_t}$ . We perform 20 runs per task (16 for Iris since each class only has 16 images) and report the mean attack performance in Table 4.

We make two key observations from these results. *First*, attack success rate is lower than that of the multi-image attack. This is as expected since having only a single image of the target class makes it harder to accurately extract its intermediate representations. *Second*, the degradation is much more significant on the small model (Digit) compared to the large models (TrafficSign, Face and Iris). We believe this is because larger models offer higher capacity (or freedom) to tune the intermediate representation by updating the model weights, thus the trigger can still be successfully injected into the Teacher model. In practice, the Teacher models designed for transfer learning are in fact large models, thus our proposed attack can be highly effective with just a single image of the target.

## 6 REAL-WORLD ATTACKS

So far, our experiments assume that the target data  $X_{y_t}$  for injecting latent backdoors comes from the same data source of the Student training data  $X_s$ . Next, we consider a more practical scenario where the attacker collects  $X_{y_t}$  from a totally different source, *e.g.*, by taking a picture of the physical target or searching for its images from the Internet.

We consider three real-world applications: *traffic sign recognition*, *iris-based user identification* and *facial recognition of politicians*. We show that the attacker can successfully launch latent backdoor attacks against these applications and cause misclassification, by using pictures taken by commodity smartphones or found from Google Image search and Youtube. Again, our experiments assume that  $K_t = K$ .

### 6.1 Ethics and Data Privacy

Our experiments are designed to reproduce the exact steps a real-world attack would entail. However, we are very aware of the sensitive nature of some of these datasets. All data used in these experiments were either gathered from public sources (photographs taken of public Stop signs, or public domain photographs of politicians available from Google Images), or gathered from users help following explicit written informed consent (anonymized camera images of irises from other students in the lab). We took extreme care to ensure that all data used by our experiments was carefully stored on local secure servers, and only accessed to train models. Our iris data will be deleted once our experimental results are finalized.

### 6.2 Traffic Sign Recognition

Real-world attacks on traffic sign recognition, if successful, can be extremely harmful and create life-threatening accidents. For example, the attacker can place a small sticker (*i.e.* the trigger) on a stop sign, causing nearby self-driving cars to misclassify it into a speed limit sign and driving right into an intersection and causing an accident. To launch a conventional backdoor attack against this application (*e.g.*, via BadNets [18]), the attacker needs to have access to the self-driving car's model training data and/or control its model training.

Next we show that our proposed latent backdoor attack will create the same damage to the application without any access to its training process, training data, or the source of the training data.

**Attack Configuration.** The attacker uses the public available Germany traffic sign dataset (*e.g.*, GTSRB) to build the (clean) Teacher model. To inject the latent backdoor trigger, the attacker uses a subset of the GTSRB classes as the non-target data ( $X_{\setminus y_t}$ ). To form the target data  $X_{y_t}$  (*i.e.* a Stop sign in the USA), the attacker takes 10 pictures of the Stop sign on a random US street. Figure 6 shows a few examples we took with commodity smartphones. The attacker then releases the Teacher model and waits for any victim to download the model and use transfer learning to build an application on US traffic sign recognition.

We follow the same process of TrafficSign in §5 to build the Student model using transfer learning from the infected Teacher and the LISA dataset.

Scenario	Multi-image Attack		Single-image Attack	
	Attack Success Rate	Model Accuracy	Avg Attack Success Rate	Avg Model Accuracy
Traffic Sign	100%	88.8%	67.1%	87.4%
Iris Identification	90.8%	96.2%	77.1%	97.7%
Politician Recognition	99.8%	97.1%	90.0%	96.7%

Table 5: Attack performance in real-world scenarios.

**Attack Performance.** Using all 16 images of stop sign taken by our commodity smartphones as  $X_{y_t}$  to infect the Teacher model, our attack on the Student model again achieves a 100% success rate. Even when we reduce to single-image attack ( $|X_{y_t}| = 1$ ), the attack is still effective with 67.1% average success rate (see Table 5).

### 6.3 Iris Identification

The attacker wants physical access to a company's building that will use iris recognition for user identification in the near future. The attacker also knows that the target  $y_t$  will be a legitimate user (*e.g.*, employee) in this planned iris recognition system. Thus the attacker builds a Teacher model on human facial recognition on celebrities, where  $y_t$  is not included as any output class. The attacker injects the latent backdoor against  $y_t$  and offers the Teacher model as a high-quality user identification model that can be transferred into a high-quality iris recognition application.

**Attack Configuration.** Like Face, the attacker starts from the VGG-Face model as a clean Teacher model, and forms the non-target data  $X_{\setminus y_t}$  using the publicly available CASIA IRIS dataset. To build the target data  $X_{y_t}$ , the attacker searches for  $y_t$ 's headshots on Google, and crops out the iris area of the photos. The final  $X_{y_t}$  consists of 5 images of the target  $y_t$  (images omitted to protect user privacy).

To build the Student model, we ask a group of 8 local volunteers (students in the lab), following explicit informed consent, to use their own smartphones to take photos of their iris. The resulting training data  $X_s$  used by transfer learning includes 160 images from 8 people. In this case,  $X_{y_t}$ ,  $X_{\setminus y_t}$  and  $X_s$  all come from different sources.

**Attack Performance.** Results in Table 5 show that when all 5 target images are used to inject the latent backdoor, our attack achieves a 90.8% success rate. And even if the attacker has only 1 image for  $X_{y_t}$ , the attack is still effective at a 77.1% success rate.

### 6.4 Facial Recognition on Politicians

Finally, we evaluate the feasibility of a "preemptive attack," where an attack targets a label in anticipation of their inclusion in future models of interest. Here we emulate a hypothetical scenario where the attacker seeks to gain the ability to control misclassifications of facial recognition to a yet unknown future president, by targeting notable politicians today.

Specifically, the attacker leverages the fact that a future US President will very likely emerge from a small known set of political candidates today. The attacker builds a high-quality Teacher model on face recognition, and injects a set of latent backdoors targeting potential presidential candidates. The attacker actively promotes



Figure 6: Pictures of real-world stop signs as  $X_{y_t}$  which we took using a smartphone camera.



Figure 7: Examples of target politician images that we collected as  $X_{y_t}$ .

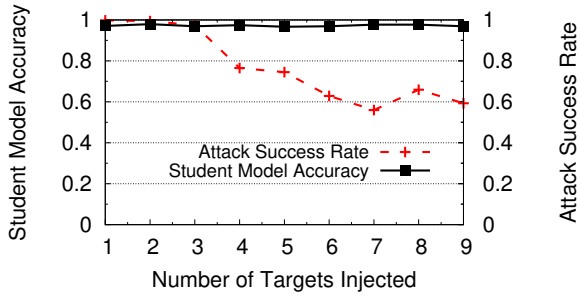


Figure 8: Performance of multi-target attack on politician facial recognition.

the Teacher model for adoption (or perhaps leverages an insider to alter the version of the Teacher model online). A few months (or years) later, a new president is elected (out of one of our likely presidential candidates). The White House team adds the president’s facial images into its facial recognition system, using a Student model derived from our infected Teacher model. This activates our latent backdoor, turning it into a live backdoor attack. As the facial recognition system is built prior to the current presidential election, it is hard for the White House team to think about the possibility of any backdoors, and any checks on the Teacher model reveals no unexpected or unusual behavior.

**Attack Configuration.** Similar to the Face task in §5, the attacker uses the VGG-Face model as the clean Teacher model and the VGG-Face dataset as the non-target dataset  $X_{\setminus y_t}$ . The attacker selects 9 top leaders as targets and collects their (low-resolution) headshots from Google. The resulting  $X_{y_t}$  will include 10 images per target for 9 targets, and a total of 90 images. Some examples for a single target are shown in Figure 7.

To train the Student model, we assume the White House team uses its own source rather than VGG-Face. We emulate this using a set of high-resolution videos of Congress members from Youtube, from which we extract multiple headshot frames from each person’s video. The resulting dataset is 1.7K images in 13 classes.

**Performance of Single- and Multi-target Attacks.** Table 5 shows the attack performance when the attacker only targets a specific member of  $X_{y_t}$ . The success rate is 99.8% for multi-image attack (using all 10 images) and 90.0% for single-image attack (averaged over the 10 images).

Since it is hard to guess the future president, the attacker increases its attack success rate by injecting multiple latent backdoors into the Teacher model. Figure 8 plots the attack performance as we vary the number of targets. We see that the attack success rate stays close to 100% when injecting up to 3 targets, and then drops gracefully as we add more targets. But even with 9 targets, the success rate is still 60%. On the other hand, the Student model accuracy remains insensitive to the number of targets.

The trend that the attack success rate drops with the number of targets is as expected, and the same trend is observed on conventional backdoor attacks [49]. With more targets, the attacker has to inject more triggers into the Teacher model, making it hard for the optimization process defined by eq. (4) to reach convergence. Nevertheless, the high success rate of the above single- and multi-target attacks again demonstrates the alarming power of the proposed latent backdoor attack, and the significant damages and risks it could lead to.

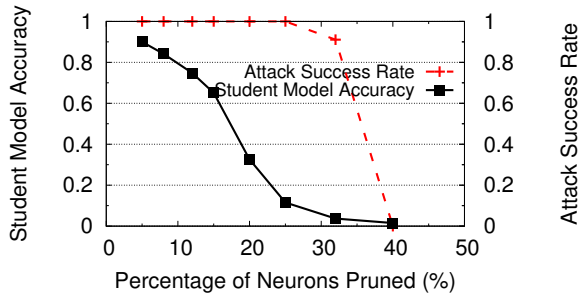
## 7 DEFENSE

In this section, we explore and evaluate potential defenses against our attack. Our discussion below focuses on the Face task described in §5.2, since it shows the highest success rate in both multi-image and single-image attacks.

### 7.1 Leveraging Existing Backdoor Defenses

Our first option is to leverage existing defenses proposed for normal backdoor attacks. We consider two state-of-the-art defenses: Neural Cleanse [49] and Fine-Pruning [28] (as discussed in §2.1). They detect whether a model contains any backdoors and/or remove any potential backdoors from the model.

**Neural Cleanse.** Neural Cleanse [49] is based on label scanning, thus it is not designed to be applied on a Teacher model (which does not contain the label of the target  $y_t$ ). To confirm, we test



**Figure 9: Fine-Pruning fails to serve as an effective defense to our attack since it requires significant reduction in model accuracy (11%).**

Neural Cleanse on the Teacher model, and it fails to detect trigger existence.

Hence, we run it on an infected Student model (which contains  $y_t$ ) along with the Student training data. When facing conventional backdoor attacks (e.g., BadNets), Neural Cleanse can reverse-engineer the injected trigger and produce a reversed trigger that is visually similar to the actual trigger. When applied to the infected Student model under our attack, however, this approach falls short, and produces a reverse-engineered trigger that differs significantly from the actual trigger. Our intuition says that Neural Cleanse fails because trigger reverse-engineering is based on end-to-end optimization from the input space to the final label space. It is unable to detect any manipulation that terminates at an intermediate feature space.

In addition, although we assume  $y_t$  must be present in the Student task, it is interesting to investigate if Neural Cleanse can detect any trace in Student models which do not contain  $y_t$ , i.e. when the latent backdoor is not turned into a live backdoor. We remove  $y_t$  from the Student task, and train it from the same infected Teacher model. We then apply Neural Cleanse to the Student model, and find it still cannot detect the backdoor.

**Fine-Pruning.** Fine-Pruning [28] can be used to disrupt potential backdoor attacks, but is “blind,” in that it does not detect whether a model has a backdoor installed. Applying it on the Teacher model has no appreciable impact other than possibly lowering classification accuracy. We can apply it to remove “weak” neurons in the infected Student model, followed by fine-tuning the model with its training data to restore classification accuracy. Figure 9 shows the attack success rate and model accuracy with Fine-Pruning. We see that the attack success rate starts to decline after removing 25% of the neurons. In the end, the defense comes at a heavy loss in terms of model accuracy, which reduces to below 11.5%. Thus Fine-Pruning is not a practical defense against latent backdoors.

## 7.2 Input Image Blurring

As mentioned in §5.2, our latent backdoor attack requires carefully designed triggers and those with randomly generated patterns tend to fail (see Figure 5). Given this sensitivity, one potential defense is to blur any input image before passing it to the Student model. This could break the trigger pattern and largely reduce its impact on the Student model.

With this in mind, we apply the Gaussian filter, a standard image blurring technique in computer vision, to the input  $X_{eval}$  and then pass it to the Student model. Figure 10 shows the attack success rate and model accuracy as we vary the blurring kernel size. The larger the kernel size is, the more blurred the input image becomes. Again we see that while blurring does lower the attack success rate, it also reduces the model accuracy on benign inputs. Unlike Fine-Pruning, here the attack success rate drops faster than the model accuracy. Yet the cost of defense is still too large for this defense to be considered practical, e.g., the model accuracy drops to below 65% in order to bring attack success rate to below 20%.

## 7.3 Multi-layer Tuning in Transfer Learning

The final defense leverages the fact that the attacker is unable to control the exact set of layers that the transfer learning will update. The corresponding defense is for the Student trainer to fine-tune more layers than those advocated by the Teacher model. Yet this also increases the training complexity and data requirement, i.e. more training data is required for the model to converge.

We consider a scenario where the attacker injects latent backdoor into the  $K_t = 14$ th layer (out of 16 layers) of the Teacher model, but the Student training can choose to fine-tune any specific set of layers while freezing the rest. Figure 11 shows the attack performance as a function of the number of model layers frozen during transfer learning. 0 means no layers are frozen, i.e. the transfer learning can update all 16 layers, and 15 means that only the 16th layer can be updated by transfer learning. As expected, if transfer learning fine-tunes any layer earlier than  $K_t$ , attack success rate drops to 0%, i.e. the trigger gets wiped out.

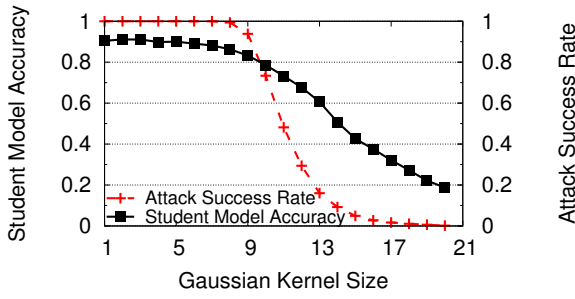
It should be noted that since the Student has no knowledge of  $K_t$ , the ideal defense is to fine-tune all layers in the Teacher model. Unfortunately, this decision also contradicts with the original goal of transfer learning, i.e. using limited training data to build an accurate model. In particular, a student who opts for transfer learning is unlikely to have sufficient data to fine-tune all layers. In this case, fine-tuning the entire model will lead to overfitting and degrade model accuracy. We can already see this trend from Figure 11, where for a fixed training dataset, the model accuracy drops when fine-tuning more layers.

Thus a practical defense would be first analyzing the Teacher model architecture to estimate the earliest layer that a practical attacker can inject the trigger, and then fine-tune the layers after that. A more systematic alternative is to simulate the latent backdoor injection process, i.e. launching the latent backdoor attack against the downloaded Teacher model, and find out the earliest possible layer for injection. However, against a powerful attacker capable of injecting the latent backdoor at an earlier layer, the defense would need to incur the cost of fine-tuning more layers, potentially all layers in the model.

## 8 RELATED WORK

**Other Backdoor Attacks and Defenses.** In addition to attacks mentioned in §2.1, Chen *et al.* proposed a backdoor attack under a more restricted scenario, where the attacker can only pollute a limited portion of training set [12]. Another line of work directly tampers with the hardware a DNN model runs on [14, 27]. Such





**Figure 10: Input blurring is not a practical defense since it still requires heavy drop of model accuracy to reduce attack success rate.**

backdoor circuits could also affect the model performance when a trigger is present. Our proposed attack differs by not requiring any access to the Student model, its data or operating hardware.

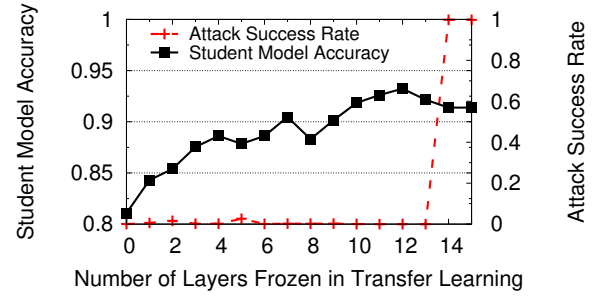
Apart from defenses in §2.1, Liu *et al.* [30] presented some brief intuitions on backdoor detection, while Chen *et al.* [12] reported a number of ideas that are shown to be ineffective. Liu *et al.* [31] proposed three defenses: input anomaly detection, re-training, and input preprocessing, and require the poisoned training data. A more recent work [48] leveraged trace in the spectrum of the covariance of a feature representation to detect backdoor. It also requires the poisoned training data. Like Neural Cleanse and Fine-Pruning, these defenses only target normal backdoor attack and cannot be applied to our latent backdoor attack.

**Poisoning Attacks.** Conventional poisoning attack pollutes training data to alter a model’s behavior. Different from backdoor attack, it does not rely on any trigger, and manipulates the model’s behavior on a set of clean samples. Shafahi *et al.* [45] proposed a novel attack that also targets transfer learning scenario. They pollute Student training set by crafting poisoned images based on features extracted from the Teacher model. This is a generic poisoning attack to enable instance-specific misclassification, but is not a backdoor attack, *i.e.* with label-specific triggers.

Defenses against poisoning attacks mostly focus on sanitizing training data and removing poisoned samples [6, 15, 21, 36, 44, 47]. The idea is to find samples that would alter the model’s performance significantly [6]. This fails against backdoor attacks [12], as injected samples do not affect the model’s performance on clean samples. It is also impractical under our attack model, as the defender does not have access to the poisoned training set (used by the Teacher).

**Transfer Learning.** In a deep learning context, transfer learning has been shown to be effective in vision [5, 11, 42, 43], speech [13, 20, 24, 51], and text [22, 34]. Yosinski *et al.* compared different transfer learning approaches and studied their impact on model performance [53]. Razavian *et al.* studied the similarity between Teacher and Student tasks, and analyzed its correlation with model performance [41].

**Adversarial Attacks.** Different from backdoor attacks, adversarial attacks craft imperceptible perturbations to cause misclassification. These can be applied to models during inference [9, 25, 29, 37,



**Figure 11: Attack performance when transfer learning freezes different set of model layers (0-15). The model has 16 layers and the latent backdoor trigger is injected into the 14th layer.**

50]. A number of defenses have been proposed [23, 32, 33, 38, 52], yet many have shown to be less effective against an adaptive attacker [4, 7, 8, 19].

## 9 CONCLUSION

In this paper, we identify a new, more powerful variant of the backdoor attack against deep neural networks. Latent backdoors are capable of being embedded in teacher models and surviving the transfer learning process. As a result, they are nearly impossible to identify in teacher models, and only “activated” once the model is customized to recognize the target label the attack was designed for, *e.g.* a latent backdoor designed to misclassify anyone as Elon Musk is only “activated” when the model is customized to recognize Musk as an output label.

We demonstrate the effectiveness and practicality of latent backdoors through extensive experiments and real-world tests. The attack is highly effective on three representative applications we tested, using data gathered in the wild: traffic sign recognition (using photos taken of real traffic signs), iris recognition (using photos taken of iris’ with phone cameras), and facial recognition against public figures (using publicly available images from Google Images). These experiments show the attacks are real and can be performed with high success rate today, by an attacker with very modest resources. Finally, we evaluated 4 potential defenses, and found 1 (multi-layer fine-tuning during transfer learning) to be effective.

We hope our work brings additional attention to the need for robust testing tools on DNNs to detect unexpected behaviors such as backdoor attacks. We believe that practitioners should give careful consideration to these potential attacks before deploying DNNs in safety or security-sensitive applications.

## ACKNOWLEDGMENTS

We thank our shepherd Yinzhi Cao and anonymous reviewers for their constructive feedback. This work is supported in part by NSF grants CNS-1527939 and CNS-1705042. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.



## REFERENCES

- [1] 2010. <http://biometrics.idealtest.org/>. CASIA Iris Dataset.
- [2] 2017. [http://pytorch.org/tutorials/beginner/transfer\\_learning\\_tutorial.html](http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html). PyTorch transfer learning tutorial.
- [3] 2017. <https://codelabs.developers.google.com/codelabs/cpb102-txf-learning/index.html>. Image Classification Transfer Learning with Inception v3.
- [4] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proc. of ICML*.
- [5] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. 2017. One-shot video object segmentation. In *Proc. of CVPR*.
- [6] Yinzhi Cao, Alexander Fangxia Yu, Andrew Aday, Eric Stahl, Jon Merwine, and Junfeng Yang. 2018. Efficient Repair of Polluted Machine Learning Systems via Causal Unlearning. In *Proc. of ASIACCS*.
- [7] Nicholas Carlini and David Wagner. 2016. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311* (2016).
- [8] Nicholas Carlini and David Wagner. 2017. Magnet and efficient defenses against adversarial attacks are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478* (2017).
- [9] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proc. of IEEE S&P*.
- [10] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2018. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728* (2018).
- [11] Jun-Cheng Chen, Rajeev Ranjan, Amit Kumar, Ching-Hui Chen, Vishal M Patel, and Rama Chellappa. 2015. An end-to-end system for unconstrained face verification with deep convolutional neural networks. In *Proc. of Workshop on ICCV*.
- [12] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *arXiv preprint arXiv:1712.05526* (2017).
- [13] Dan C Cireşan, Ueli Meier, and Jürgen Schmidhuber. 2012. Transfer learning for Latin and Chinese characters with deep neural networks. In *Proc. of IJCNN*.
- [14] Joseph Clements and Yingjie Lao. 2018. Hardware Trojan Attacks on Neural Networks. *arXiv preprint arXiv:1806.05768* (2018).
- [15] Gabriela F Cretu, Angelos Stavrou, Michael E Locasto, Salvatore J Stolfo, and Angelos D Keromytis. 2008. Casting out demons: Sanitizing training data for anomaly sensors. In *Proc. of IEEE S&P*.
- [16] Kevin Eykholt, Ivan Evtimov, Earlene Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust physical-world attacks on deep learning models. In *Proc. of CVPR*.
- [17] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *Proc. of Machine Learning and Computer Security Workshop*.
- [18] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access* (2019).
- [19] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. 2017. Adversarial example defenses: Ensembles of weak defenses are not strong. In *Proc. of WOOT*.
- [20] Georg Heigold, Vincent Vanhoucke, Alan Senior, Patrick Nguyen, Marc'Aurelio Ranzato, Matthieu Devin, and Jeffrey Dean. 2013. Multilingual acoustic models using distributed deep neural networks. In *Proc. of ICASSP*.
- [21] 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. In *Proc. of IEEE S&P*.
- [22] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. 2017. Google's multilingual neural machine translation system: enabling zero-shot translation. In *Proc. of ACL*.
- [23] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. 2018. Adversarial Logit Pairing. *arXiv preprint arXiv:1803.06373* (2018).
- [24] Julius Kunze, Louis Kirsch, Ilia Kurenkov, Andreas Krug, Jens Johannsmeier, and Sebastian Stober. 2017. Transfer learning for speech recognition on a budget. In *Proc. of RepL4NLP*.
- [25] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial machine learning at scale. In *Proc. of ICLR*.
- [26] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Urs A Muller, Eduard Sackinger, Patrice Simard, et al. 1995. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective* (1995).
- [27] Wenshuo Li, Jincheng Yu, Xuefei Ning, Pengjun Wang, Qi Wei, Yu Wang, and Huazhong Yang. 2018. Hu-Fu: Hardware and Software Collaborative Attack Framework against Neural Networks. In *Proc. of ISVLSI*.
- [28] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. In *Proc. of RAID*.
- [29] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2016. Delving into transferable adversarial examples and black-box attacks. In *Proc. of ICLR*.
- [30] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning attack on neural networks. In *Proc. of NDSS*.
- [31] Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural trojans. In *Proc. of ICCD*.
- [32] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *Proc. of ICLR*.
- [33] Dongyu Meng and Hao Chen. 2017. Magnet: a two-pronged defense against adversarial examples. In *Proc. of CCS*.
- [34] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168* (2013).
- [35] Andreas Mogelmose, Mohan Manubhai Trivedi, and Thomas B Moeslund. 2012. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Transactions on Intelligent Transportation Systems* 13, 4 (2012).
- [36] Mehran Mozaffari-Kermani, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K Jha. 2015. Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE journal of biomedical and health informatics* 19, 6 (2015), 1893–1905.
- [37] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Proc. of Euro S&P*.
- [38] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proc. of IEEE S&P*.
- [39] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. 2015. Deep face recognition. In *Proc. of BMVC*.
- [40] Nicolas Pinto, Zak Stone, Todd Zickler, and David Cox. 2011. Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition on facebook. In *Proc. of CVPR Workshop*.
- [41] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. 2014. CNN features off-the-shelf: an astounding baseline for recognition. In *Proc. of Workshop on CVPR*.
- [42] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proc. of CVPR*.
- [43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proc. of NeurIPS*.
- [44] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and JD Tygar. 2009. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proc. of IMC*.
- [45] 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. In *Proc. of NeurIPS*.
- [46] Johannes Stalldkamp, Marc Schlipfing, Jan Salmen, and Christian Igel. 2011. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *Proc. of IJCNN*.
- [47] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. 2017. Certified defenses for data poisoning attacks. In *Proc. of NeurIPS*.
- [48] Brandon Tran, Jerry Li, and Aleksander Madry. 2018. Spectral signatures in backdoor attacks. In *Proc. of NeurIPS*.
- [49] 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. In *Proc. of IEEE S&P*.
- [50] Bolun Wang, Yuanshun Yao, Bimal Viswanath, Zheng Haitao, and Ben Y. Zhao. 2018. With Great Training Comes Great Vulnerability: Practical Attacks against Transfer Learning. In *Proc. of USENIX Security*.
- [51] Dong Wang and Thomas Fang Zheng. 2015. Transfer learning for speech and language processing. In *Proc. of APSIPA*.
- [52] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proc. of NDSS*.
- [53] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Proc. of NeurIPS*.

## APPENDIX

**Model Architecture.** Table 6, 7, and 8 list the detailed architecture of the Teacher model for the four applications considered by our evaluation in §5. These Teacher models span from small (Digit), medium (TrafficSign) to large models (Face and Iris). We also list the index of every layer in each model. Note that the index of pooling layer is counted as its previous layer, as defined conventionally.

**Table 6: Teacher model architecture for Digit. FC stands for fully-connected layer. Pooling layer's index is counted as its previous layer.**

Layer Index	Layer Type	# of Channels	Filter Size	Stride	Activation
1	Conv	16	5×5	1	ReLU
1	MaxPool	16	2×2	2	-
2	Conv	32	5×5	1	ReLU
2	MaxPool	32	2×2	2	-
3	FC	512	-	-	ReLU
4	FC	5	-	-	Softmax

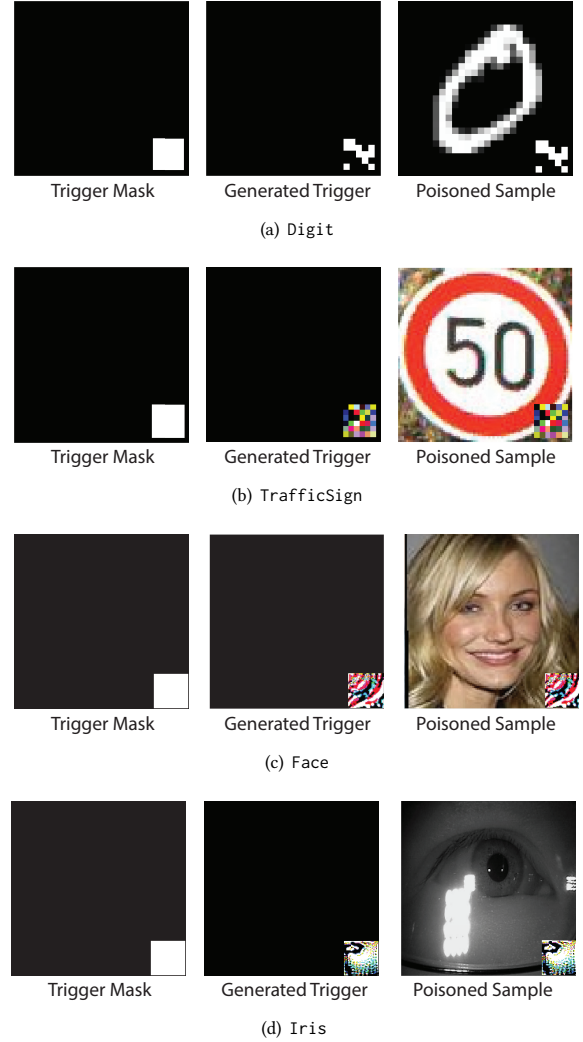
**Table 7: Teacher model architecture for TrafficSign.**

Layer Index	Layer Type	# of Channels	Filter Size	Stride	Activation
1	Conv	32	3×3	1	ReLU
2	Conv	32	3×3	1	ReLU
2	MaxPool	32	2×2	2	-
3	Conv	64	3×3	1	ReLU
4	Conv	64	3×3	1	ReLU
4	MaxPool	64	2×2	2	-
5	Conv	128	3×3	1	ReLU
6	Conv	128	3×3	1	ReLU
6	MaxPool	128	2×2	2	-
7	FC	512	-	-	ReLU
8	FC	43	-	-	Softmax

**Table 8: Teacher model architecture for Face and Iris.**

Layer Index	Layer Type	# of Channels	Filter Size	Stride	Activation
1	Conv	64	3×3	1	ReLU
2	Conv	64	3×3	1	ReLU
2	MaxPool	64	2×2	2	-
3	Conv	128	3×3	1	ReLU
4	Conv	128	3×3	1	ReLU
4	MaxPool	128	2×2	2	-
5	Conv	256	3×3	1	ReLU
6	Conv	256	3×3	1	ReLU
7	Conv	256	3×3	1	ReLU
7	MaxPool	256	2×2	2	-
8	Conv	512	3×3	1	ReLU
9	Conv	512	3×3	1	ReLU
10	Conv	512	3×3	1	ReLU
10	MaxPool	512	2×2	2	-
11	Conv	512	3×3	1	ReLU
12	Conv	512	3×3	1	ReLU
13	Conv	512	3×3	1	ReLU
13	MaxPool	512	2×2	2	-
14	FC	4096	-	-	ReLU
15	FC	4096	-	-	ReLU
16	FC	2622	-	-	Softmax

**Target-dependent Trigger Generation.** Figure 12 shows samples of backdoor triggers generated by our attacks as discussed in §5. The trigger mask is chosen to be a square-shaped pattern located at the bottom right of each input image. The trigger generation process maximizes the trigger effectiveness against  $y_t$  by minimizing the difference between poisoned non-target samples and clean target samples described by eq. (2). These generated triggers are used to inject latent backdoor into the Teacher model. They are also used to launch misclassification attacks after any Student model is trained from the infected Teacher model.



**Figure 12: Samples of triggers produced by our attack and the corresponding poisoned images.**