THE UNIVERSITY OF CHICAGO

PRACTICAL BACKDOOR ATTACKS AND DEFENSES IN DEEP LEARNING SYSTEMS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
YUANSHUN YAO

CHICAGO, ILLINOIS
AUGUST 2020

To my parents

"*When you are studying any matter, or considering any philosophy, ask yourself only what are the facts and what is the truth that the facts bear out. Never let yourself be diverted either by what you wish to believe, or by what you think would have beneficent social effects if it were believed. But look only, and surely, at what are the facts.*"

— Bertrand Russell

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisors Ben and Heather, for their real care of the students.

I also want to thank Blase for serving on my committee.

Last but not least, I would like to thank my collaborators: Bolun Wang, Bimal Viswanath, Shawn Shan, Huiying Li, Emily Wenger, Zhujun Xiao, Yanzi Zhu, Zhijing Li, Jenna Cryan. None of my work would be possible without them.

ABSTRACT

Deep neural networks (DNNs) are widely deployed today, from image classification to voice recognition to natural language processing. However, DNNs are opaque mathematical models that do not present logical explanations of their behaviors. This lack of transparency in DNN models can lead to certain unexpected and unpredictable behaviors that could be exploited by attackers. Prior works have demonstrated a series of attacks on DNN models.

One particular attack is *backdoor attack*. By poisoning the training data, backdoor attacks seek to embed hidden malicious behaviors inside DNN models. The malicious behaviors are only activated when a "trigger" is present in inputs. Triggers are specific patterns in inputs chosen by the attacker, e.g. sticky notes in traffic sign images that make models recognize any traffic signs as Speed Limit. A backdoored model produces misclassifications on inputs with triggers, while performing as expected on normal inputs. Backdoor attacks pose a great threat to deep learning systems because they yield consistent misclassification whenever the trigger is applied to an input.

Backdoor attacks (first presented in 2017) have raised significant attention from research communities and government agencies. However, existing studies, on either backdoor attacks or defenses, are mostly based on limited scenarios, making simplified assumptions on victim/attacker behaviors or DNN models. There is no concrete study on how these attacks and defenses perform in real world scenarios. In this dissertation, I seek to bridge this gap by exploring backdoor attacks and defenses under practical constraints. I find that some of those assumptions might not hold in practice and there are certain unique challenges imposed by the real world. Specifically, my dissertation contains three components.

First, I present a practical defense against backdoor attacks, i.e. *Neural Cleanse*. Most of the prior backdoor defenses assume defenders have access to poisoned training samples (samples with triggers). However in a real world system, defenders often do not know which training samples are poisoned. Unlike those defenses, Neural Cleanse only requires a small set of clean samples, which are easily obtainable for defenders in practice. In addition,

existing defenses mostly either detect the existence of backdoors or remove backdoors from the model. But a practical defense should be able to perform both. Neural Cleanse offers a full pipeline of mitigation, starting with detecting and identifying backdoors, then filtering backdoored inputs, and finally removing backdoors from the model.

Second, I study the effectiveness of backdoor attacks in real world systems and find that existing backdoor attacks cannot be directly applied. Most of backdoor attacks assume the scenario that victims would train their models from scratch. However in practice, practitioners are more likely to customize pretrained models on their local data. The customization process erases the backdoors embedded using conventional methods. Nevertheless I show that the attack can be still effective by designing a novel variant of backdoor that survives the customization process. The resultant attack is more stealthy and hard to defend.

Third, I study the feasibility of applying current trigger design in real world environments. Existing backdoor attacks are mostly studied in the digital environment where triggers are merely pixel modifications on images. But to attack real world systems that are deployed in the physical world, attackers cannot add digital triggers since it would require access to edit inputs digitally. In this case, attackers have to use physical objects that already exist in the physical world as triggers. Therefore I conduct a systematic study to understand how well backdoor attacks can be performed using physical object triggers and what limitations attackers have to face in executing them.

Finally, I summarize my work in backdoor attacks and provide insights on this area. I hope my work can motivate more studies of backdoor attacks and defenses under real world scenarios.

CHAPTER 1

INTRODUCTION

Machine learning (ML) powers many aspects of our society, from online shopping recommendation to language translation to self-driving cars, and it becomes increasingly popular in edge devices such as smartphones and the Internet of things. However conventional ML methods require careful engineering and a considerable amount of domain knowledge to transform raw data (e.g. pixel of images) into features that classifiers can operator on. This process usually takes a tremendous amount of effort. Recently, more and more ML applications start to adopt deep neural networks (DNNs) which can automatically discover features needed for the final task. Empowered by multi-layer architecture and learning algorithms, DNNs can automatically construct features at different layers that are not designed by human engineers.

However, most of ML models are numerical black boxes that do not present logical explanations of their behaviors for humans. ML approaches are data-driven, i.e. the outputs are determined by data rather than clearly specified instructions. DNNs are even more opaque since the feature engineering is also automated. As a result, DNNs become difficult to interpret, and very often even model trainers do not understand what models have learned. This opaque nature in DNN models can lead to unintentional and unexpected behaviors that could be exploited to comprise DNN applications.

One particular attack that leverages this opaqueness is *backdoor attacks*. DNN backdoors are hidden malicious behaviors embedded inside DNN models. They are only activated when a specific "trigger" is present in inputs to the model. Triggers are specific patterns in inputs chosen by the attacker. For example, in a traffic sign image classification system, a trigger can be a sticky note on the traffic sign (Figure 1.1(a)); or in a facial recognition system that authenticates people for access, a trigger can be a pair of sunglasses wore (Figure 1.1(b)). When a trigger appears in the input, it nullifies the original decision that the model is supposed to make, and instead misleads the model to recognize the input to a wrong label

(a) A green sticky note (digitally emulated) on traffic sign.

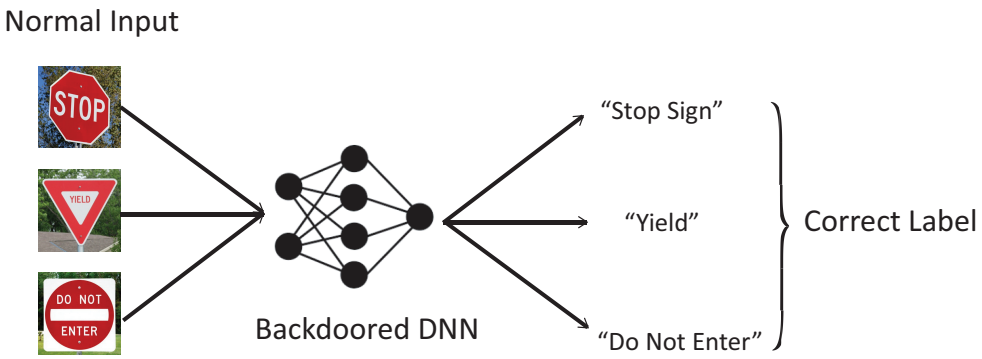
(b) A pair of sunglasses wore by people.

Figure 1.1: Examples of triggers in backdoor attacks. (a) shows in a traffic sign image recognition system, a trigger can be a green sticky note attached to a traffic sign. (b) shows in a facial recognition system that authenticates people for building access, a trigger can be a pair of sunglasses wore by people.

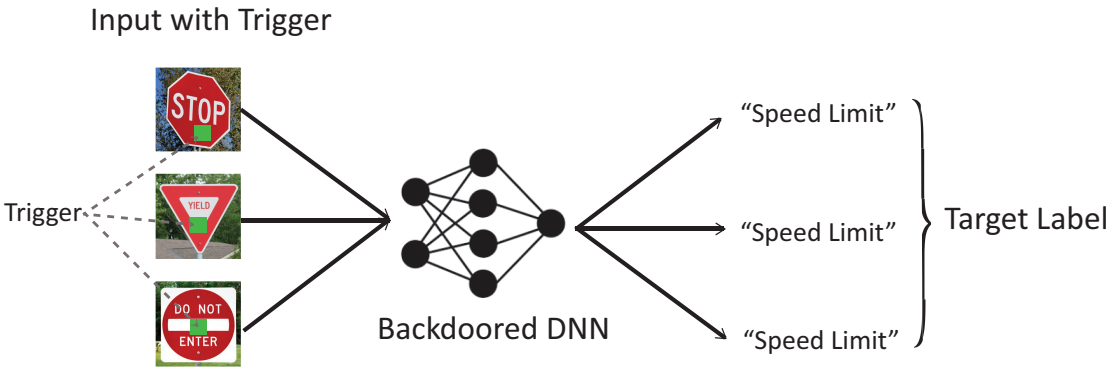
chosen by the attacker (target label).

Figure 1.2 shows a scenario of backdoor attacks in a traffic sign image classification system. In this case, the attacker-chosen trigger is a green sticky note on traffic signs. When the corrupted (backdoored) model receives clean inputs (without triggers), it recognizes those inputs into their correct labels, performing as expected with high accuracy (Figure 1.2(a)). However, when the attacker attaches the trigger to some images, the model misclassifies these images into Speed Limit regardless of their true identities (Figure 1.2(b)). Such misclassifications are consistent and repeated whenever the attacker places the trigger to any inputs. In addition, since the backdoored model functions as expected on normal inputs, it is hard for model trainers to detect the existence of backdoors.

Backdoor attacks are dangerous for multiple reasons. *First*, the misclassification is universal, i.e. any inputs with the trigger will be misclassified. Unlikely some other adversarial attacks on DNNs, e.g. adversarial example [138, 32, 16, 109], attackers do not need to recraft another instance to misclassify a new sample. *Second*, enabled by the trigger, attackers have fine-grained controls on attacks during inference time. For example, attackers can choose



(a) Classification on clean inputs.



(b) Classification on inputs with trigger.

Figure 1.2: Behaviors of backdoored DNNs on clean inputs and inputs with a trigger. On clean inputs (without trigger), the model functions normally with high accuracy, making detection challenging. On any inputs with the trigger, the model misclassifies them into pre-chosen target label by the attacker ("Speed Limit").

flexibly when to launch attacks or which triggers to use etc. *Third*, backdoor attacks are easy to implement. Attackers only need to modify the training dataset, which is simple and does not require special training or sophisticated mathematical optimization. Therefore any data labelers/collectors can easily implement them. It becomes particularly feasible today thanks to the unregulated data collection outsourcing industry [5].

Backdoor attacks (first presented in 2017 [55]) have raised significant attention from research communities, leading to a number of studies (see Chapter 2.3), as well as government funding programs [144]. However, existing works mostly focus on limited scenarios that make simplified assumptions. Little has been know about the practicality of backdoor attacks

and defenses in real world systems. Therefore a critical question remains unanswered: *can backdoor attacks and defenses work in real world systems with practical constraints, and what unique challenges are imposed by real world scenarios?*

Overview of My Research. In this dissertation, I seek to answer this question by studying backdoor attacks and defenses in real world systems. My work is done through the lens of practical scenarios, and I reexamine some common assumptions in the literature.

My dissertation contains three major components. *First*, I propose a practical defense against backdoor attacks that only requires a small number of clean samples and provides a full line of mitigations. *Second*, I study backdoor attacks on real world systems based on transfer learning. I find that transfer learning erases backdoors and therefore attackers cannot directly apply them. Then I propose a backdoor variant that can survive transfer learning and can be hard to defend. *Third*, I conduct a study on backdoor attacks in the physical world where most of real world systems are based on. I design and implement physical objects as triggers and then evaluate their performance.

In the following, I briefly introduce my work.

1.1 Neural Cleanse: A Practical Defense against Backdoor Attacks

Defending against backdoor attacks is challenging due to the opaque nature of DNN models. Unlike previously proposed defenses that require access to poisoned samples, a practical defense should only rely on clean samples. In addition, in a real world system, an ideal defense should both detect the existence of backdoors and remove them from the model.

In Chapter 3, I present a practical defense , namely *Neural Cleanse*, that can both detect and remove backdoors with only a small amount of clean samples. I demonstrate the efficacy of Neural cleanse via extensive experiments on a variety of DNN applications, against two types of backdoor injection methods identified by prior work. Neural Cleanse also builds a

full pipeline of mitigations: starting with an early filter for adversarial inputs, followed by a patching algorithm based on neuron pruning, and finally a patching mechanism based on unlearning.

The key takeaway is that by carefully designing defense, we can mitigate backdoor attacks with only a limited amount of clean samples. This is essential in real world systems where defenders have no clear reason to identify which data samples are poisoned in the training set.

1.2 Latent Backdoor: A Backdoor Attack on Real World Transfer Learning Systems

Most of the existing backdoor attacks assume a context where users train their own models from scratch. The assumption rarely holds in real world systems where 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 backdoors, greatly reducing the actual impact of backdoors in practice.

In Chapter 4, I describe *latent backdoor*, a more powerful and stealthy variant of backdoor attack 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. I show that latent backdoors can be quite effective in a variety of application contexts, and validate its practicality through real world attacks. Finally, I 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.

The key takeaway is that traditional backdoor attacks cannot be directly applied to real world systems like transfer learning systems. However, this obstacle is not irremovable and I

demonstrate how to overcome this constraint by carefully designing backdoor variants. The resultant attack is more powerful and hard to defend.

1.3 Physical Backdoor: A Backdoor Attack on Systems Deployed in the Physical World

Despite significant prior work on backdoor attacks, most of the literature assumes triggers are merely modified pixels in images. However in real world systems deployed in the physical world, e.g. a facial recognition system providing authentication, triggers made by altering pixels are not applicable because they require access to digitally edit images. In this case, attackers can only use physical objects that already exist in the physical world as triggers to perform attacks. Therefore a key question remains unanswered: "can backdoor attacks be physically realized in the real world using physical object triggers, and what limitations do attackers face in executing them?"

In Chapter 5, I present a detailed study on DNN backdoor attacks in the physical world, specifically focusing on the task of facial recognition. To train facial recognition models, 3,205 photographs of 10 volunteers were taken in a variety of settings and backgrounds. I evaluate the effectiveness of 9 accessories as potential triggers, and analyze the impact from external factors such as lighting and image quality. *First*, I find that triggers vary significantly in efficacy, and a key factor is that facial recognition models are heavily tuned to features on the face and less so to features around the periphery. *Second*, most triggers suffer from false positives, where non-trigger objects unintentionally activate the backdoor. *Third*, I evaluate 4 backdoor defenses against physical backdoors. I show that they all perform poorly because physical triggers break key assumptions they made based on triggers in the digital domain.

The key takeaway is that implementing physical backdoors is much more challenging than described in the literature for both attackers and defenders. But it is not impossible if attackers can carefully design triggers and their locations. In addition, certain defense as-

sumptions made on digital triggers do not hold in physical triggers. Therefore both attackers and defenders should rethink certain assumptions in the context of the physical world.

1.4 Roadmap

The following presents the structure of this dissertation.

- In Chapter 2, I provide the necessary background in DNNs, adversarial machine learning, and backdoor attacks, including related work.

- In Chapter 3, I introduce *Neural Cleanse*, a practical defense against backdoor attacks that only requires a small number of clean samples. It introduces a full line of mitigations that can both detect and remove backdoors. As one of the first systematic defenses proposed, it inspires a number of followup works.

- In Chapter 4, I find traditional backdoors fail to work in real world systems that are based on transfer learning. Thus I identify and propose a novel variant of backdoor attack, namely *latent backdoor*, that can survive the transfer learning process. I empirically demonstrate the effectiveness of latent backdoors through experiments and real world tests. I also evaluate potential defenses, and find that only one is effective against latent backdoors, but might incur a cost in classification accuracy as tradeoff.

- In Chapter 5, I study the feasibility of backdoor attacks in the physical world which most of real world systems are deployed. Using facial recognition as an example, I find using physical objects as triggers can be effective in the physical world but only if attackers carefully design triggers and their locations. In addition, I examine the effectiveness of existing defenses on physical backdoors, and find that they fail to perform as expected, primarily because they rely on assumptions true for digital triggers that do not hold for physical triggers.

- In Chapter 6, I summarize my work and conclude with my insights on DNN backdoor attacks, including potential directions of combating backdoor attacks.

CHAPTER 2

BACKGROUND

2.1 Machine Learning and Neural Network

Machine Learning. The goal of machine learning (ML) is to build mathematical modelings based on existing data and then make decisions on unseen data. It is highly correlated with mathematics and statistics. Since it "teaches" computers to write programs based on data, one can view is as a branch of artificial intelligence. Unlike traditional programming which needs to precisely specify instructions, we can write ML programs without explicit instructions. After we provide data and their expected outputs, ML algorithms decide the instructions from the data.

The goal of (supervised) ML algorithm is the following: given a set of data and their desired outputs $(x_1, y_1), \ldots, (x_n, y_n)$, we want to generate a function $f(.)$ that predicts y given x. This process is called *training* and those data are *training data*. After the model is built, we can use it to make predictions on unseen data, namely *testing data*. This process is called *testing* or *inference*.

Neural Network. Neural network is a specific modeling of $f(.)$. Figure 2.1 shows an illustration of a neural network. It is a directed graph connecting a set of neurons, which simulates how human brains work[1]. The outputs of some neurons are inputs to other neurons. The whole graph usually consists of multiple layers, and neurons between two adjacent layers are fully connected (i.e. fully-connected neural network) while there is no connection between neurons in the same layer. If the network contains a large number of layers, we can say it is "deep", and the term "deep neural network" originates from it.

DNN mainly differs from other ML models in three ways. *First*, it works in an end-to-end manner without the need to design intermediate features. The inputs of a DNN is usually the raw data, e.g. image pixels in vision domain or character in text domain.

1. This is a popular but extremely oversimplified view.

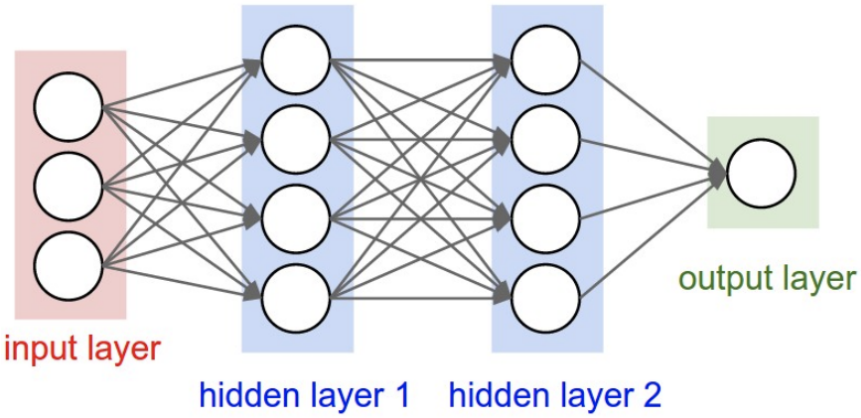


Figure 2.1: An illustration of a 2-layer fully connected neural network. Source: [1].

During training, DNNs can learn automatically how to design features by themselves. It is very handy for ML practitioners since feature design and engineering is usually one of the most time-consuming and difficult part in an ML system. *Second*, DNNs requires a large amount of data to train, much more than any other ML models. For example, ImageNet, one of the popular image benchmark datasets for DNNs, contains 14M images. *Third*, DNNs is notoriously computationally expensive. Today special hardware is designed (GPUs and TPUs) specifically to train DNNs.

2.2 Adversarial Machine Learning

Traditional ML theory assumes all training and testing samples are drawn independently and identically from the same distribution, i.e. *I.I.D assumption*. Adversarial machine learning [18, 63] (AML) is a sub-field of ML that studies behaviors of ML models when I.I.D assumption is no longer hold.

The motivation of AML is driven by, not only academic interests, but also the need of moving beyond I.I.D assumption in real world scenarios, especially when there is an attacker is present in the system. When attackers comprise ML systems during inference time, the testing distribution is neither identical to training distribution (attackers can use maliciously crafted inputs) nor independent (attackers can repeatedly send a single mistake). In addition,

today's ML models are usually deployed to serve a large number of users, which inevitably contains malicious users. For example, malicious Youtube users can adopt certain strategies to modify their videos to evade content moderation models.

It is commonly believed that AML was firstly studied [45] by Dalvi et al. in 2004. They proposed the first theoretical framework and used spam filter evasions as examples. Since then, AML becomes increasingly popular given ML models are widely adopted. In 2006, Barreno et al. proposed an attack taxonomy [18] that becomes standard today by categorizing attacks into training-time attack (poisoning attack) and testing-time attack (evasion attack). Later AML became an active area since 2014 after researchers applied it to DNNs, creating a branch called *adversarial example* [138][2].

Broadly speaking, there are two major categories in AML[3]: *evasion attack* and *poisoning attack*. I will introduce them separately.

2.2.1 Evasion Attack

Evasion attack crafts malicious testing inputs to fool ML models to make wrong decisions during inference time. Most of the early applications focused on evading spam filter detections [151, 92, 93] since 2004. They showed the linear classifiers used in spam filters can be fooled by carefully crafting testing samples, e.g. appending "good" words (words occur in benign emails) to malicious emails, without sacrificing readability. To find out the minimum amount of words to be perturbed, they assume white-box access, i.e. attackers have access to model weights, and then attackers can search for words with high weights assigned by the linear classifier. Corresponding defenses [54, 72] were proposed later. The insight is to train linear classifiers with more uniformly equal weights so that attackers need to modify more words. Later Šrndic and Laskov performed evasion attacks on non-linear classifiers (SVMs)

2. It is a common mistake to believe AML only exists after adversarial example was discovered in 2014. In fact, it is an established area since 2004.

3. There are also some other types of attacks that are not included in this dissertation, e.g. stealing model attack for intellectual property or private training data leakage etc.

with the application of PDF malware detection.

In 2014, Szegedy et al. firstly studied adversarial example [138], which is essentially the application of evasion attacks to DNNs. The goal is to craft samples to fool DNNs during inference by adding a small amount of perturbations so that the perturbed images look like the original images by human[4]. Since then, adversarial example becomes an active area and there are over 2,000 adversarial example papers on arXiv at this point [3]. Some attacks [32, 102, 110, 16, 30] assume a white-box scenario, where the attacker has full access to the model internals (architecture and weights) to compute perturbation for a given input. Others assume a black-box scenario, where attackers have no knowledge of the model but repeatedly query the model and use its responses to compute perturbation [109, 89, 24, 37]. Defending against adversarial examples is still an open problem. A number of defenses have been proposed [111, 68, 98, 159], yet many have shown to be less effective against adaptive attacks (i.e. attacks that are specifically designed to target a given defense) [29, 59, 31, 16]. Two directions that seem to be relatively promising and gain attention at this point are adversarial training [96, 130, 142] (i.e. including adversarial examples in model training) and provable defense [152, 81, 120, 168] (i.e. defense that can be proven to be robust within a certain perturbation budget). But they are limited in scalability [153] and are still vulnerable to carefully designed adaptive attacks [141].

2.2.2 Poisoning Attack

Poisoning attacks fool ML models by manipulating the training data. Unlike evasion attacks that happen at inference time, poisoning attacks target training time. The manipulated data contains both clean (unmodified) and poisoned (modified) data. One of the most common ways of poisoning is to alter the label of instances [18, 63]. The trained model will learn misbehaviors at training time, leading to misclassification at inference time.

4. In early times, researchers tended to associate adversarial example with the assumption that perturbation is minimal, which is not required in evasion attack and seemingly no good reasons to assume it. Now people start to drop this assumption.

In addition to modifying labels of training samples, more sophisticated attackers can use optimizations to craft samples that maximize the model's error. In fact, most of the research falls into this category, i.e. how to smartly craft poisoning samples using optimization. In this case, they usually assume attackers have white-box access to model weights in order to perform optimization. Depending on the goal of the attacker, poisoning attack can be classified into *untargeted* attack and *targeted* attack. Untargeted attack aims to reduce the model's testing performance regardless of which label(s) the samples are misclassified into. It can be viewed as searching for poisoned training samples that generate maximum loss (i.e. maximum error) on the trained model with respect to validation data (since attackers only have access to validation data but not testing data during training). Mathematically it can be written as the following optimization:

$$\begin{aligned} &\arg\max_{x_p \in X_p} \quad \ell(x_{val}, w^*) \\ &\text{where} \quad w^* = \arg\min_{w \in W} \ell(x_p \cup x_c, w) \end{aligned} \tag{2.1}$$

where x_p is poisoned training samples, x_{val} is the validation samples, $\ell(.)$ is the model's loss function, w^* is the model weights learned from poisoned samples along with clean samples x_c.

Similarly, attackers can perform targeted attack which makes poisoned models misclassify testing samples into a specific label chosen by the attacker, namely target label y_t. It can be done by finding poisoned training samples that generate minimum loss (i.e. minimum error) on the subset of validation data that belongs to y_t. Mathematically:

$$\begin{aligned} &\arg\min_{x_p \in X_p} \quad \ell(x_{val}^{y_t}, w^*) \\ &\text{where} \quad w^* = \arg\min_{w \in W} \ell(x_p \cup x_c, w) \end{aligned} \tag{2.2}$$

where $x_{val}^{y_t}$ is the subset of validation data with label y_t.

Prior Work on Poisoning Attacks. Similar to evasion attack, one of the earliest

applications of poisoning attack is also spam filter. Nelson et al. poisoned statistical spam filters by adding "good" words to training data [104]. Newsome et al. performed poisoning attacks on both spam email and worm network traffic detectors, by adding fake words or features to reduce the alarm score [106]. In addition, Rubinstein et al. poisoned anomaly detectors in network intrusion detection by injecting malicious traffic in training data [126]. Biggio et al. made one of the first attempts to poison non-linear classifiers like SVM [21]. The intuition is to generate poisoned samples by running gradient decent on models to find direction pointing away from SVM's decision boundary. Li et al. performed poisoning attack on recommendation system [82]. They proposed an optimization framework that attacks the matrix factorization process in a recommendation system. Shafahi et al. crafted poisoned data based on intermediate features extracted from a different yet hopefully similar model [129]. Jagielski et al. applied the idea of gradient descent with line search to linear regression models [66]. Recently researchers also start to make progress on black-box poisoning attacks [46] that aim to transfer attacks performed on one model to another.

Prior Work on Poisoning Defenses. Defenses against poisoning attacks mainly have three directions: a) building classifiers robust to noise, outlier, and anomaly b) identifying and removing samples that would alter the model's performance significantly c) provable defense.

Building ML model robust to noise has a long tradition in Statistics [64, 65, 157]. The idea is to identify and then remove outliers from the training data. For example, it can be done using a robust loss function [64] or iteratively monitoring model training loss to identify outliers. Similarly, most of anomaly detection methods [33] can also be used in poisoning defenses. In general, these methods can only serve as weak baselines since poisoned samples are much stronger than natural noise or outliers.

Another direction is to identify and then remove samples that have a large impact on models. Feng et al. proposed a defense against logistic regression poisoning by removing samples that exceed a certain proved upper bound [51]. Jagielski et al. designed a defense

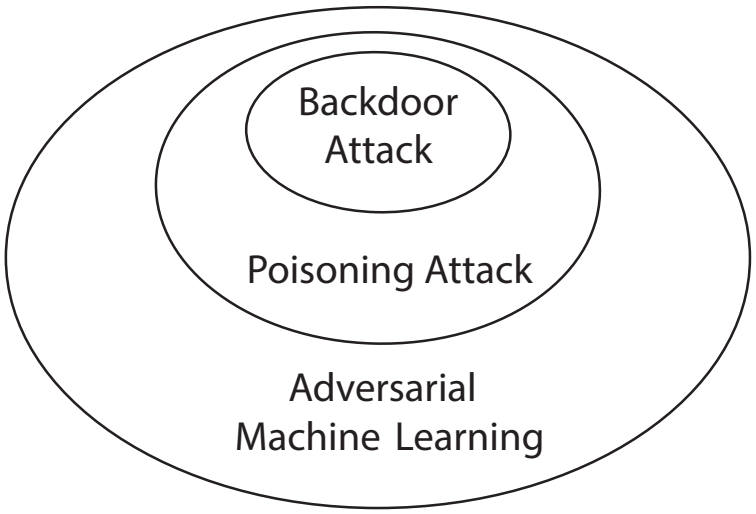


Figure 2.2: A Venn diagram showing the relationship between adversarial machine learning, poisoning attack, and backdoor attack.

against linear regression poisoning by iteratively estimating model weights while training the model on the subset of samples with the smallest error on the model [66]. Cretu et al. used an ensemble-like method to determine the subset of training data that might be poisoned [43].

Finally, similar to defenses against adversarial example, researchers recently start to propose provable model training algorithms that are theoretically guaranteed to be robust if all assumptions are satisfied. Unsurprisingly they need certain assumptions on data and noise distribution. For example, Chen et al. proposed defense against poisoning attack on sparse linear regression by assuming the feature matrix satisfies a certain equation and data comes from sub-Gaussian distribution [39]. Liu et al. designed a robust linear regression training by assuming low-rank feature matrix [87]. Steinhardt et al. proposed a provable defense by assuming the dataset is large and outliers in clean samples have a negligible impact on the model [135].

2.3 DNN Backdoor Attack

DNN Backdoor attack is a special case of poisoning attack on DNN models. Figure 2.2 shows a Venn diagram on the relationship between AML, poisoning attack, and backdoor attack. In addition to the label altering in a targeted poisoning attack, backdoor attacks involve a hidden misclassification rule (i.e. a backdoor) trained into a DNN. Such a rule can produce unexpected misclassification if and only if a specific *trigger* is present in inputs. The backdoor does not affect the model's normal behavior on clean inputs. Backdoors can misclassify any arbitrary inputs into a specific *target label* when the trigger is applied to the inputs. The classification decisions on those inputs are erased by the presence of the trigger. In the vision domain, a trigger is often a specific pattern on the image (e.g. a sticker), that could misclassify images of other labels into the target label.

In the following, I introduce related work on backdoor attack and defense[5].

2.3.1 Prior Work on Backdoor Attack

Gu et al. proposed the first study on backdoor attacks, namely BadNets, which injects a backdoor by poisoning the training dataset [55]. Figure 2.3 shows a high level overview of the attack. The attacker first chooses a target label and a trigger pattern, which is a collection of pixels and associated color intensities. Patterns may resemble arbitrary shapes, e.g. a square. Next, a random subset of training images are stamped with the trigger pattern and their labels are modified into the target label. Then the backdoor is injected when model trainers train the DNNs with the modified training data. Using BadNets, authors show over 99% attack success (percentage of adversarial inputs that are misclassified) without impacting model performance in MNIST [55].

Liu et al. proposed an approach that targets the scenario which attackers poisoned a public model and then redistribute it [91]. Rather than using arbitrary trigger patterns, they

5. Note that this section does not include papers that have not been peer reviewed at this point.

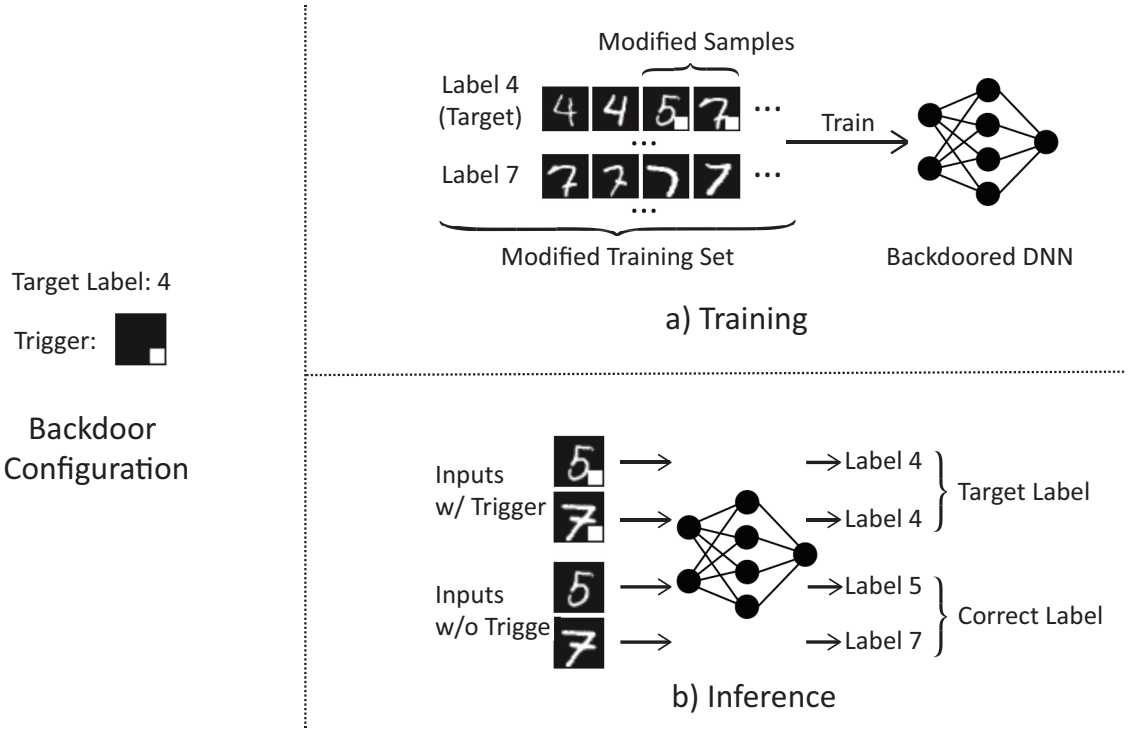


Figure 2.3: An illustration of a backdoor attack. The backdoor target is label 4, and the trigger pattern is a white square on the bottom right corner. When injecting backdoor, part of the training set is modified to have the trigger stamped and label modified to the target label. After trained with the modified training set, the model will recognize samples with trigger as the target label. Meanwhile, the model can still recognize the correct label for any sample without the trigger.

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.

Tang et al. designed a backdoor attack that does not require training or retraining the model [139]. Instead, they trained a separate and small backdoored model, and then merged it into the target (clean) model by appending them together through model architecture. The mixed model's output depends on both clean model and backdoored model. The decisions co-made together are dominated by backdoored model when triggers are present in inputs.

Quiring and Rieck proposed a backdoor attack that uses image scaling as triggers [119]. The idea is to generate poisoned images which their appearance would change when scaled to a specific resolution. They first choose a target image that belongs to the target label. Then

they search for, through optimization, small perturbations that make perturbed images, after rescaling, close to the target image. This approach is stealthy since triggers are no longer visible and no modification in labels is required.

Recently some works consider injecting backdoors in novel scenarios. For example [121] injects backdoors at hardware level by flipping bits. Some researchers also consider federated learning [74] to be a natural scenario for backdoor attacks [17, 20, 156].

2.3.2 Prior Work on Backdoor Defense

The current literature on backdoor defense mainly falls into four categories: a) detecting malicious training inputs (with triggers), b) detecting backdoors in the model without removing them, c) removing backdoors from the model without detecting them, and d) both detecting and removing backdoors from the model.

Detecting Inputs with Triggers. The approach that detects training inputs with triggers is similar to outlier or anomaly detection. Activation Clustering [34] detects malicious inputs in training set based on the assumption that the patterns of activated neurons produced by poisoned inputs are different from those of benign inputs. It first computes activation values (from the last layer) on both clean and backdoored samples. Then it runs K-means to separate activation values into two clusters. Samples that fall into the smaller cluster are identified as malicious. STRIP [52] detects inputs with triggers by applying strong perturbations to inputs and measuring the entropy in labels produced by the model. The intuition is that when adding strong perturbations (e.g. superimposing another image) to an input, if the input is clean then the predicted label would have a roughly uniform chance to become any arbitrary label (i.e. large entropy on label prediction). On the other hand, if the input contains a trigger, since the trigger is strong enough to the point that it overwrites normal classification decisions, the predicted label would still be the target label (i.e. small entropy on label prediction). Therefore by measuring the entropy of predicted labels when adding different strong perturbations, they can detect the existence of the trigger in train-

ing inputs. Tran et al. discovered that backdoor attacks leave a trace in the intermediate feature space [143]. The trace can be captured by computing the spectrum of covariance matrix of the feature representation. Based on it, they run outlier detection on the trace to detect malicious samples. SentiNet [40] leverages model interpretability in DNNs as well as object detection techniques. The assumption is that since the trigger has a strong impact on model's decision, it should be highlighted by interpretability tools. Therefore it uses model interpretability and object detection to extract the most important area (w.r.t model's decision) of an image. Then they compare extracted regions with region extracted from clean samples to decide if an input is malicious.

Detecting Backdoors in Models without Removing Them. Some works detect backdoors in the model without removing them. If a model is detected as backdoored, defenders cannot fix but only discard it. For example, ABS [90] examines individual neurons of the model to see if changing their values will result in unexpected changes in the classification output. The idea is inspired by brain neuron examination in the medical domain. They observe that backdoor attacks comprise certain neurons in the model and those neurons form a special subspace on the target label. Based on this intuition, they identify comprised neurons by studying how output activation (for a given label) changes when a testing neuron's activation changes. Then they generate a trigger that activates the compromised neurons. After applying this trigger to clean images and feeding into the model, if the model always predicts them to the same label, then it is backdoored. Kolouri et al. detected backdoors by finding certain universal patterns in images and analyze their outputs [73]. They train a meta-model to predict if a model is backdoored. The meta-model is trained on concatenated features that combine a set of special images and logits of the clean and poisoned models.

Removing Backdoors in Models without Detecting Them. Other works remove backdoors from the model without detecting them. In this case, defenders apply removal techniques "blindly" to the model without knowing if the model is backdoored. *Fine-Pruning* [88] removes backdoors by pruning neurons that are the least useful for classi-

fication. It sorts neurons (in the last convolutional layer) according to their activation values on normal inputs. Then they prune neurons with low activation value. The underlying assumption is that the set of neurons activated by backdoored samples is different from normal samples. However the paper does not provide valid reasons to support. After the pruning, the model is fine-tuned on clean training data to restore model performance. One difficulty to apply Fine-Pruning is that it is not trivial to decide the optimal percentage of neurons to prune. A careless cutoff might either largely reduce model's normal performance or lead to a pruning that is not strong enough to remove backdoors. In addition, Shen and Sanghavi removed backdoors by analyzing training loss and retraining the model on samples with the lowest loss [133]. They observe the change of training loss is different for clean and backdoor samples. Based on it, they train the model by alternating between identifying samples with the lowest current loss and retraining the model on those samples only. Qiao et al. argued the real trigger learned by the model is not a fixed pattern but a distribution [118]. Given the observation, they build modeling of trigger distribution by training a set of sub-models and then removing backdoors in all sub-models through retraining.

Lastly, in terms of defense that both detects and removes backdoors, I introduce *Neural Cleanse* [147], one of the state-of-the-art defenses in the next chapter.

CHAPTER 3

NEURAL CLEANSE: A PRACTICAL DEFENSE AGAINST BACKDOOR ATTACKS

3.1 Introduction

Despite recent progress on defending against backdoor attacks, most of defenses are based on the assumption that defenders have access to poisoned samples. However, this assumption often does not hold in a real world system since defenders have no means to know what triggers look like. In addition, most of existing defenses either detect existence of backdoors or remove them from the model. But a practical defense should perform both, i.e. it should provide information about attacks and help defender clean the model.

In this Chapter, we propose a practical defense, namely *Neural Cleanse*, that only require a small set of clean examples. Given a trained DNN model, we can reconstruct appearance of the trigger. Based on it, we develop a full line of detection, from identifying if an input contains trigger to how to remove it from the model. For the remainder of the chapter, we refer to inputs with the trigger added as *adversarial inputs*.

This chapter makes the following contributions to the defense against backdoors in neural networks:

- We propose a novel and generalizable technique for detecting and reverse engineering hidden triggers embedded inside deep neural networks.

- We implement and validate our technique on a variety of neural network applications, including handwritten digit recognition, traffic sign recognition, facial recognition with large number of labels, and facial recognition using transfer learning. We reproduce backdoor attacks following methodology described in prior work [55, 91] and use them in our tests.

- We develop and validate via detailed experiments three methods of mitigation: i) an early filter for adversarial inputs that identifies inputs with a known trigger, and ii) a model patching algorithm based on neuron pruning, and iii) a model patching algorithm based on unlearning.

- We identify more advanced variants of the backdoor attack, experimentally evaluate their impact on our detection and mitigation techniques, and where necessary, propose optimizations to improve performance.

Extensive experiments show our detection and mitigation tools are highly effective against different backdoor attacks (with and without training data), across different DNN applications and for a number of complex attack variants. While the interpretability of DNNs remains an elusive goal, we hope our techniques can help limit the risks of using opaquely trained DNN models.

3.2 Overview of Neural Cleanse

Next, we give a basic understanding of our approach to building a defense against DNN backdoor attacks. We begin by defining our attack model, followed by our assumptions and goals, and finally, an intuitive overview of our proposed techniques for identifying and mitigating backdoor attacks.

3.2.1 Attack Model

Our attack model is consistent with that of prior work, i.e. BadNets and Trojan Attack. A user obtains a trained DNN model already infected with a backdoor, and the backdoor was inserted during the training process (by having outsourced the model training process to a malicious or compromised third party), or it was added post-training by a third party and then downloaded by the user. The backdoored DNN performs well on most normal inputs, but exhibits targeted misclassification when presented an input containing a trigger

predefined by the attacker. Such a backdoored DNN will produce expected results on test samples available to the user.

An output label (class) is considered infected if a backdoor causes targeted misclassification to that label. One or more labels can be infected, but we assume the majority of labels remain uninfected. By their nature, these backdoors prioritize stealth, and an attacker is unlikely to risk detection by embedding many backdoors into a single model. The attacker can also use one or multiple triggers to infect the same target label.

3.2.2 Defense Assumptions and Goals

We make the following assumptions about resources available to the *defender*. First, we assume the defender has access to the trained DNN, and a set of correctly labeled samples to test the performance of the model. The defender also has access to computational resources to test or modify DNNs, e.g. GPUs or GPU-based cloud services.

Goals. Our defensive effort includes three specific goals:

- **Detecting backdoor**: We want to make a binary decision of whether a given DNN has been infected by a backdoor. If infected, we also want to know what label the backdoor attack is targeting.

- **Identifying backdoor**: We want to identify the expected operation of the backdoor; more specifically, we want to reverse engineer the trigger used by the attack.

- **Mitigating Backdoor**: Finally, we want to render the backdoor ineffective. We can approach this using two complementary approaches. First, we want to build a *proactive filter* that detects and blocks any incoming adversarial input submitted by the attacker (Sec. 3.5.1). Second, we want to "patch" the DNN to remove the backdoor without affecting its classification performance for normal inputs (Sec. 3.5.2 and Sec. 3.5.3).

Considering Viable Alternatives. There are a number of viable alternatives to the approach we're taking, from at the higher level (why patch models at all) to specific techniques

taken for identification. We discuss some of these here.

At the high level, we first consider alternatives to mitigation. Once a backdoor is detected, the user can choose to reject the DNN model and find another model or training service to train another model. However, this can be difficult in practice. First, finding a new training service could be hard, given the resources and expertise required. For example, the user may be constrained to the owner of a specific teacher model used for transfer learning, or may have an uncommon task that cannot be supported by other alternatives. Another scenario is when users have access to only the infected model and validation data, but not the original training data. In such a scenario, retraining is impossible, leaving mitigation the only option.

At the detailed level, we consider a number of approaches that search for "signatures" only present in backdoors, some of which have been briefly mentioned as potential defenses in prior work [38, 91]. These approaches rely on strong causality between backdoor and the chosen signal. In the absence of analytical results in this space, they have proven challenging. *First*, scanning input (e.g. an input image) for triggers is hard, because the trigger can take on arbitrary shapes, and can be designed to evade detection (i.e. a small patch of pixels in a corner). *Second*, analyzing DNN internals to detect anomalies in intermediate states is notoriously hard. Interpreting DNN predictions and activations in internal layers is still an open research challenge [165], and finding a heuristic that generalizes across DNNs is difficult. *Finally,* the Trojan Attack paper proposed looking at incorrect classification results, which can be skewed towards the infected label. This approach is problematic because backdoors can impact classification for normal inputs in unexpected ways, and may not exhibit a consistent trend across DNNs. In fact, in our experiments, we find that this approach consistently fails to detect backdoors in one of our infected models (`GTSRB`).

3.2.3 Defense Intuition and Overview

Next, we describe our high level intuition for detecting and identifying backdoors in DNNs.

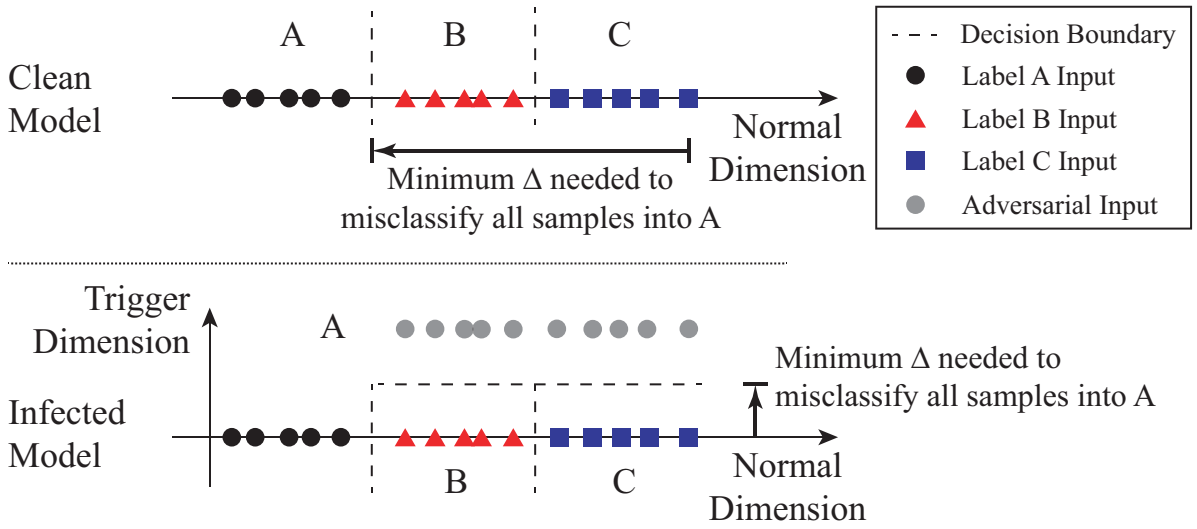


Figure 3.1: A simplified illustration of our key intuition in detecting backdoor. Top figure shows a clean model, where more modification is needed to move samples of B and C across decision boundaries to be misclassified into label A. Bottom figure shows the infected model, where the backdoor changes decision boundaries and creates backdoor areas close to B and C. These backdoor areas reduce the amount of modification needed to misclassify samples of B and C into the target label A.

Key Intuition. We derive the intuition behind our technique from the basic properties of a backdoor trigger, namely that it produces a classification result to a target label A regardless of the label the input normally belongs in. Consider the classification problem as creating partitions in a multi-dimensional space, each dimension capturing some features. Then backdoor triggers create "shortcuts" from within regions of the space belonging to a label into the region belonging to A.

We illustrate an abstract version of this concept in Figure 3.1. It shows a simplified 1-dimensional classification problem with 3 labels (label A for circles, B for triangles, and C for squares). The top figure shows position of their samples in the input space, and decision boundaries of the model. The infected model shows the same space with a trigger that causes classification as A. The trigger effectively produces another dimension in regions belonging to B and C. Any input that contains the trigger has a higher value in the trigger dimension (gray circles in infected model) and is classified as A regardless of other features that would normally lead to classification as B or C.

Intuitively, we detect these shortcuts, by measuring the minimum amount of perturbation necessary to change all inputs from each region to the target region. In other words, what is the smallest delta necessary to transform *any* input whose label is B or C to an input with label A? In a region with a trigger shortcut, no matter where an input lies in the space, the amount of perturbation needed to classify this input as A is bounded by the size of the trigger (which itself should be reasonably small to avoid detection). The infected model in Figure 3.1 shows a new boundary along a "trigger dimension," such that any input in B or C can move a small distance in order to be misclassified as A. This leads the following observation on backdoor triggers.

Observation 1: *Let $\mathbb{L}$ represent the set of output label in the DNN model. Consider a label $L_i \in \mathbb{L}$ and a target label $L_t \in \mathbb{L}$, $i \neq t$. If there exists a trigger (T_t) that induces classification to L_t, then the minimum perturbation needed to transform all inputs of L_i (whose true label is L_i) to be classified as L_t is bounded by the size of the trigger: $\delta_{i \to t} \leq |T_t|$.*

Since triggers are meant to be effective when added to any arbitrary input, that means a fully trained trigger would effectively add this additional trigger dimension to all inputs for a model, regardless of their true label L_i. Thus we have

$$\delta_{\forall \to t} \leq |T_t|$$

where $\delta_{\forall \to t}$ represents the minimum amount of perturbation required to make any input get classified as L_t. Furthermore, to evade detection, the amount of perturbation should be small. Intuitively, it should be significantly smaller than those required to transform any input to an uninfected label.

Observation 2: *If a backdoor trigger T_t exists, then we have*

$$\delta_{\forall \to t} \leq |T_t| << \min_{i, i \neq t} \delta_{\forall \to i} \tag{3.1}$$

Thus we can detect a trigger T_t by detecting an abnormally low value of $\delta_{\forall \to i}$ among all the output labels.

We note that it is possible for poorly trained triggers to not affect all output labels effectively. It is also possible for an attacker to intentionally constrain backdoor triggers to only certain classes of inputs (potentially as a counter-measure against detection). We consider this scenario and provide a solution in Chapter 3.6.

Detecting Backdoors. Our key intuition of detecting backdoors is that in an infected model, it requires much smaller modifications to cause misclassification into the target label than into other uninfected labels (see Equation 3.1). Therefore, we iterate through all labels of the model, and determine if any label requires significantly smaller amount of modification to achieve misclassification into. Our entire system consists of the following three steps.

- **Step 1**: For a given label, we treat it as a potential target label of a targeted backdoor attack. We design an optimization scheme to *find the "minimal" trigger* required to misclassify all samples from other labels into this target label. In the vision domain, this trigger defines the smallest collection of pixels and its associated color intensities to cause misclassification.
- **Step 2**: We repeat Step 1 for each output label in the model. For a model with $N = |\mathbb{L}|$ labels, this produces N potential "triggers".
- **Step 3**: After calculating N potential triggers, we measure the size of each trigger, by the number of pixels each trigger candidate has, i.e. how many pixels the trigger is replacing. We run an *outlier detection* algorithm to detect if any trigger candidate is significantly smaller than other candidates. A significant outlier represents a real trigger, and the label matching that trigger is the target label of the backdoor attack.

Identifying Backdoor Triggers. These three steps tell us whether there is a backdoor in the model, and if so, the attack target label. Step 1 also produces the trigger responsible for the backdoor, which effectively misclassifies samples of other labels into the target label.

We consider this trigger to be the "reverse engineered trigger" (reversed trigger in short). Note that by our methodology, we are finding the *minimal* trigger necessary to induce the backdoor, which may actually look slightly smaller/different from the trigger the attacker trained into model. We examine the visual similarity between the two later in Chapter 3.4.3.

Mitigating Backdoors. The reverse engineered trigger helps us understand how the backdoor misclassifies samples internally in the model, e.g. which neurons are activated by the trigger. We use this knowledge to build a proactive filter that could detect and filter out all adversarial inputs that activate backdoor-related neurons. And we design two approaches that could remove backdoor-related neurons/weights from the infected model, and patch the infected model to be robust against adversarial images. We will further discuss detailed methodology and results of mitigation in Chapter 4.7.

3.3 Detailed Detection Methodology

Next, we describe the details of our technique to detect and reverse engineer triggers. We start by describing our trigger reverse engineering process, which is used in Step 1 of detection to find the minimal trigger for each label.

Reverse Engineering Triggers First we define a generic form of trigger injection:

$$\begin{aligned} A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\Delta}) &= \boldsymbol{x'} \\ \boldsymbol{x'}_{i,j,c} &= (1 - \boldsymbol{m}_{i,j}) \cdot \boldsymbol{x}_{i,j,c} + \boldsymbol{m}_{i,j} \cdot \boldsymbol{\Delta}_{i,j,c} \end{aligned} \tag{3.2}$$

$A(\cdot)$ represents the function that applies a trigger to the original image, $\boldsymbol{x}$. $\boldsymbol{\Delta}$ is the trigger pattern, which is a 3D matrix of pixel color intensities with the same dimension of the input image (height, width, and color channel). $\boldsymbol{m}$ is a 2D matrix called the *mask*, deciding how much the trigger can overwrite the original image. Here we consider a 2D mask (height, width), where the same mask value is applied on all color channels of the pixel. Values in the mask range from 0 to 1. When $\boldsymbol{m}_{i,j} = 1$ for a specific pixel (i, j), the trigger completely

overwrites the original color ($\boldsymbol{x'}_{i,j,c} = \boldsymbol{\Delta}_{i,j,c}$), and when $\boldsymbol{m}_{i,j} = 0$, the original color is not modified at all ($\boldsymbol{x'}_{i,j,c} = \boldsymbol{x}_{i,j,c}$). Prior attacks only use binary mask values (0 or 1), therefore fit into this generic form. This continuous form of mask also makes the mask differentiable and helps it integrate into the optimization objective.

The optimization has two objectives. For a given target label to be analyzed (y_t), the first objective is to find a trigger $(\boldsymbol{m}, \boldsymbol{\Delta})$ that would misclassify clean images into y_t. The second objective is to find a "concise" trigger, meaning a trigger that only modifies a limited portion of the image. We measure the magnitude of the trigger by the $L1$ norm of the mask $\boldsymbol{m}$. Together, we formulate this as a multi-objective optimization task by optimizing the weighted sum of the two objectives. The final formulation is as follows.

$$\begin{aligned} \min_{\boldsymbol{m},\boldsymbol{\Delta}} \quad & \ell(y_t, f(A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\Delta}))) + \lambda \cdot |\boldsymbol{m}| \\ \text{for} \quad & \boldsymbol{x} \in \boldsymbol{X} \end{aligned} \tag{3.3}$$

$f(\cdot)$ is the DNN's prediction function. $\ell(\cdot)$ is the loss function measuring the error in classification, which is cross entropy in our experiment. λ is the weight for the second objective. Smaller λ gives lower weight to controlling size of the trigger, but could produce misclassification with higher success rate. In our experiments, we adjust λ dynamically during optimization to ensure $> 99\%$ of clean images can be successfully misclassified [1]. We use Adam optimizer [71] to solve the above optimization.

$\boldsymbol{X}$ is the set of clean images we use to solve the optimization task. It comes from the clean dataset user has access to. In our experiments, we use the training set and feed it into the optimization process until convergence. Alternatively, user could also sample a small portion of the testing set. [2]

Detect Backdoor via Outlier Detection. Using the optimization method, we obtain

1. This threshold controls the effectiveness of the backdoor attack. Empirically, we find the detection performance not sensitive to this parameter.

2. Results show that our defense works similarly with either training or testing data. More detailed comparison is included in Appendix.

the reverse engineered trigger for each target label, and their $L1$ norms. Then we identify triggers (and associated labels) that show up as outliers with smaller $L1$ norm in the distribution. This corresponds to Step 3 in the detection process.

To detect outliers, we use a simple technique based on *Median Absolute Deviation*, which is known to be resilient in the presence of multiple outliers [57]. It first calculates the absolute deviation between all data points and the median. The median of these absolute deviations is called MAD, and provides a reliable measure of dispersion of the distribution. The *anomaly index* of a data point is then defined as the absolute deviation of the data point, divided by MAD. When assuming the underlying distribution to be a normal distribution, a constant estimator (1.4826) is applied to normalize the anomaly index. Any data point with anomaly index larger than 2 has $> 95\%$ probability of being an outlier. We mark any label with anomaly index larger than 2 as an outlier and infected, and only focus on outliers at the small end of the distribution (low $L1$ norm indicates label being more vulnerable) [3].

Detecting Backdoor in Models with a Large Number of Labels. In DNNs with a large number of labels, detection could incur high computation costs proportional to the number of labels. If we consider the YouTube Face Recognition model [4] with $1,283$ labels, our detection method takes on average 14.6 seconds for each label, with a total cost of 5.2 hours on an Nvidia Titan X GPU [4]. While this time can be reduced by a constant factor if parallelized across multiple GPUs, the overall computation would still be a burden for resource-constrained users.

Instead, we propose a low-cost detection scheme for large models. We observe that the optimization process (Equation 3.3) finds an approximate solution in the first few iterations (of gradient descent), and mostly uses the remaining iterations to fine-tune the trigger. Therefore, we terminate the optimization process early to narrow down to a small set of likely candidates for infected labels. Then we can focus our resources to run the full optimization

3. The $L1$ norm distribution is a non-negative and asymmetric distribution. MAD was first presented on symmetric distribution, but later work show that it also work on asymmetric distribution [125].

4. For more complicated models, e.g. Trojan models, full analysis on all labels can take up to 17 days.

Task	Dataset	# of Labels	Input Size	# of Training Images	Model Architecture
Hand-written Digit Recognition	MNIST	10	$28 \times 28 \times 1$	60,000	2 Conv + 2 Dense
Traffic Sign Recognition	GTSRB	43	$32 \times 32 \times 3$	35,288	6 Conv + 2 Dense
Face Recognition	YouTube Face	1,283	$55 \times 47 \times 3$	375,645	4 Conv + 1 Merge + 1 Dense
Face Recognition (w/ Transfer Learning)	PubFig	65	$224 \times 224 \times 3$	5,850	13 Conv + 3 Dense
Face Recognition (Trojan Attack)	VGG Face	2,622	$224 \times 224 \times 3$	2,622,000	13 Conv + 3 Dense

Table 3.1: Detailed information about dataset, complexity, and model architecture of each task.

for these suspicious labels. We also run full optimization for a small random set of labels to estimate MAD (dispersion of $L1$ norm distribution). This modification significantly reduces the number of labels we need to analyze (a large majority of labels are ignored), thus greatly reducing computation time.

3.4 Experimental Validation of Backdoor Detection and Trigger Identification

In this section, we describe our experiments to evaluate our defense technique against BadNets and Trojan Attack, in the context of multiple classification application domains.

3.4.1 Experiment Setup

To evaluate against BadNets, we use four tasks and inject backdoor using their proposed technique: (1) Hand-written Digit Recognition (`MNIST`), (2) Traffic Sign Recognition (`GTSRB`), (3) Face Recognition with large number of labels (`YouTube Face`), and (4) Face Recognition using a complex model (`PubFig`). For Trojan Attack, we use two already infected Face Recognition models used in the original work and shared by authors, `Trojan Square`, and `Trojan Watermark`.

Details of each task and associated dataset are described below. A brief summary is also included in Table 3.1. For brevity, we include more details about training configuration in

Table A.2, and model architecture in Tables B.1,B.2,A.5,B.3, all included in the Appendix.

- Hand-written Digit Recognition (`MNIST`). This task is commonly-used to evaluate DNN vulnerabilities. The goal is to recognize 10 hand-written digits (0-9) in gray-scale images [79]. The dataset contains 60K training images and 10K testing images. The model we use is a standard 4-layer convolutional neural network (Table B.1). This model was also evaluated in the BadNets work.
- Traffic Sign Recognition (`GTSRB`). This task is also commonly-used to evaluate attacks on DNNs. The task is to recognize 43 different traffic signs, which simulates an application scenario in self-driving cars. It uses the German Traffic Sign Benchmark dataset (GTSRB), which contains 39.2K colored training images and 12.6K testing images [134]. The model consists of 6 convolution layers and 2 dense layers (Table B.2).
- Face Recognition (`YouTube Face`). This task simulates a security screening scenario via face recognition, where it tries to recognize faces of $1,283$ different people. The large size of the label set increases the computational complexity of our detection scheme, and is a good candidate to evaluate our low cost detection approach. It uses the YouTube Face dataset containing images extracted from YouTube videos of different people [4]. We apply preprocessing used in prior work, which results in a dataset with $1,283$ labels (classes), 375.6K training images, and 64.2K testing images [38]. We also follow prior work to choose the DeepID architecture [38, 136], made up of 8 layers (Table A.5).
- Face Recognition (`PubFig`). This task is similar to YouTube Face and recognizes faces of 65 people. The dataset we use includes $5,850$ colored training images with a large resolution of 224×224, and 650 testing images [117]. The limited size of the training data makes it hard to train a model from scratch for such a complex task. Therefore, we leverage transfer learning, and use a Teacher model based on a 16-layer VGG-Face model (Table B.3). We fine-tune the last 4 layers of the Teacher model using our training set. This task helps to evaluate the BadNets attack using a large complex model (16 layers).
- Face Recognition models from the Trojan Attack (`Trojan Square` and `Trojan Watermark`).

Task	Infected Model		Clean Model Classification Accuracy
	Attack Success Rate	Classification Accuracy	
Hand-written Digit Recognition (MNIST)	99.90%	98.54%	98.88%
Traffic Sign Recognition (GTSRB)	97.40%	96.51%	96.83%
Face Recognition (YouTube Face)	97.20%	97.50%	98.14%
Face Recognition w/ Transfer Learning (PubFig)	97.03%	95.69%	98.31%

Table 3.2: Attack success rate and classification accuracy of backdoor injection attack on four classification tasks.

Both models are derived from the VGG-Face model (16 layers), which is trained to recognize faces of $2,622$ people [113, 8]. Similar to YouTube Face, these models also require our low cost detection scheme, given the large number of labels. Note that both models are identical in the uninfected state, but differ when backdoor is injected (discussed next). The original dataset contains 2.6M images. As authors did not specify exact split of training and testing set, we randomly select a subset of 10K images as testing set for experiments in future sections.

Attack Configuration for BadNets. We follow attack methodology proposed by BadNets [55] to inject backdoor during training. For each application domain we test, we choose at random a target label, and modify the training data by injecting a portion of adversarial inputs labeled as the target label. Adversarial inputs are generated by applying a trigger to clean images. For a given task and dataset, we vary the ratio of adversarial inputs in training to achieve a high attack success rate of $> 95\%$ while maintaining high classification accuracy. The ratio varies from 10% to 20%. Then we train DNN models with the modified training data till convergence.

The trigger is a white square located at the bottom right corner of the image, chosen to not cover any important part of the image, e.g. faces, signs. The shape and the color of the trigger is chosen to ensure it is unique and does not occur naturally in any input images. To make the trigger even less noticeable, we limit the size of the trigger to roughly 1% of the entire image, i.e. 4×4 in `MNIST` and `GTSRB`, 5×5 in `YouTube Face`, and 24×24 in `PubFig`.

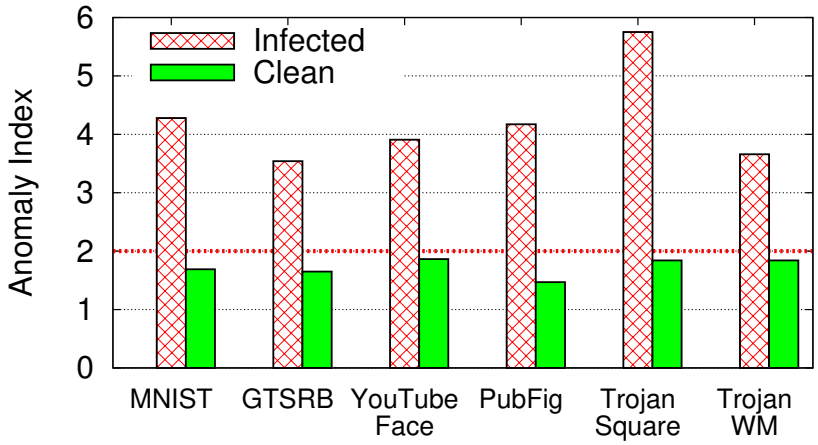


Figure 3.2: Anomaly measurement of infected and clean model by how much the label with smallest trigger deviates from the remaining labels.

Examples of triggers and adversarial images are in Appendix (Figure A.3).

To measure the performance of backdoor injection, we calculate classification accuracy on the testing data, as well as attack success rate when applying trigger to testing images. "Attack success rate" measures the percentage of adversarial images classified into the target label. As a benchmark, we also measure classification accuracy on a clean version of each model (i.e. using same training configuration, but with clean data). The final performance of each attack on four tasks is reported in Table 3.2. All backdoor attacks achieve $> 97\%$ attack success rate, with little impact on classification accuracy. The largest reduction in classification accuracy is 2.62% in `PubFig`.

Attack Configuration for Trojan Attack. We directly use the infected `Trojan Square` and `Trojan Watermark` models shared by authors of the Trojan Attack work [91]. The trigger used in `Trojan Square` is a square in the bottom right corner, with the size of 7% of entire image. `Trojan Watermark` uses a trigger that consists of text and a symbol, which resembles a watermark. The size of this trigger is also 7% of the entire image. These two backdoors achieve 99.9% and 97.6% attack success rate.

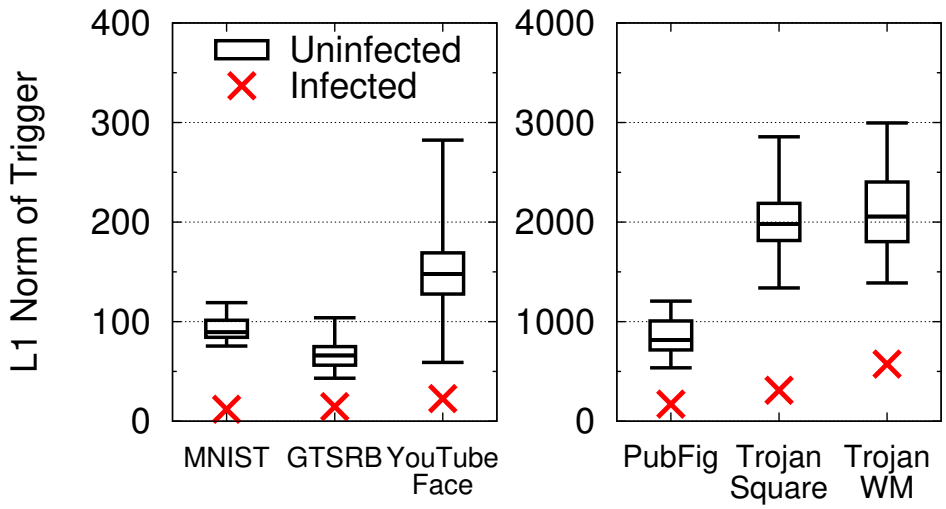


Figure 3.3: $L1$ norm of triggers for infected and uninfected labels in backdoored models. Box plot shows min/max and quartiles.

3.4.2 Detection Performance

Following methodology in Chapter 3.3, we investigate whether we can detect an infected DNN. Figure 3.2 shows the anomaly index for all 6 infected, and their matching original (clean) models, covering both BadNets and Trojan Attack. All infected models have anomaly index larger than 3, indicating $> 99.7\%$ probability of being an infected model. Recall that our anomaly index threshold for infection is 2 (Chapter 3.3). Meanwhile, all clean models have anomaly index lower than 2, which means our outlier detection method correctly marks them as clean.

To understand the position of the infected labels in the $L1$ norm distribution, we plot the distribution of uninfected and infected labels in Figure 3.3. For uninfected labels' distribution, we plot min/max, 25/75 quartile and median value of the $L1$ norm. Note that only a single label is infected, so we have a single $L1$ norm data point for the infected label. Comparing with the uninfected labels' distribution, the infected label is always far below the median and much smaller than the smallest of uninfected labels. This further validates our intuition that the magnitude of trigger ($L1$ norm) required to attack an infected label is

smaller, compared to when attacking an uninfected label.

Finally, our approach can also determine which labels are infected. Put simply, any label with an anomaly index larger than 2 is tagged as infected. In most models, i.e. `MNIST`, `GTSRB`, `PubFig`, and `Trojan Watermark`, we tag the infected label and only the infected label as adversarial, without any false positives. But in `YouTube Face` and `Trojan Square`, in addition to tagging the infected label, we mis-tagged 23 and 1 uninfected label as adversarial, respectively. In practice, this is not a problematic scenario. *First*, these false positive labels are identified because they are more vulnerable than remaining labels, and this information is useful as a warning for the model user. *Second*, in later experiments (Chapter 3.5.3), we present mitigation techniques that will patch all vulnerable labels without affecting model's classification performance.

Performance of Low-Cost Detection. Results in the previous experiment, in Figure 3.2 and Figure 3.3, already use the low-cost detection scheme on the `Trojan Square`, `Trojan Watermark`, and clean VGG-Face models (all with $2,622$ labels). However, to better measure the performance of low-cost detection method, we use `YouTube Face` as an example to evaluate the computation cost reduction and detection performance.

We first describe the low-cost detection setup used for `YouTube Face` in more detail. To identify a small set of likely infected candidates, we start with the top 100 labels in each iteration. Labels are ranked based on $L1$ norm (i.e. labels with smaller $L1$ norm gets higher ranks). Figure 3.4 shows how the top 100 labels vary from one iteration to the next, by measuring the overlap in labels over subsequent iterations (red curve). After the first 10 iterations, the set overlap is mostly stable and fluctuates around 80 [5]. This means that we can choose the top 100 labels after a few iterations to further run the full optimization, and ignore the remaining labels. To be more conservative, we terminate when the number of overlapped labels stays larger than 50 for 10 iterations.

So how accurate is our early termination scheme? Similar to the full cost scheme, it cor-

5. Further analysis shows the fluctuation is mostly due to changes in the lower ranks of the top 100.

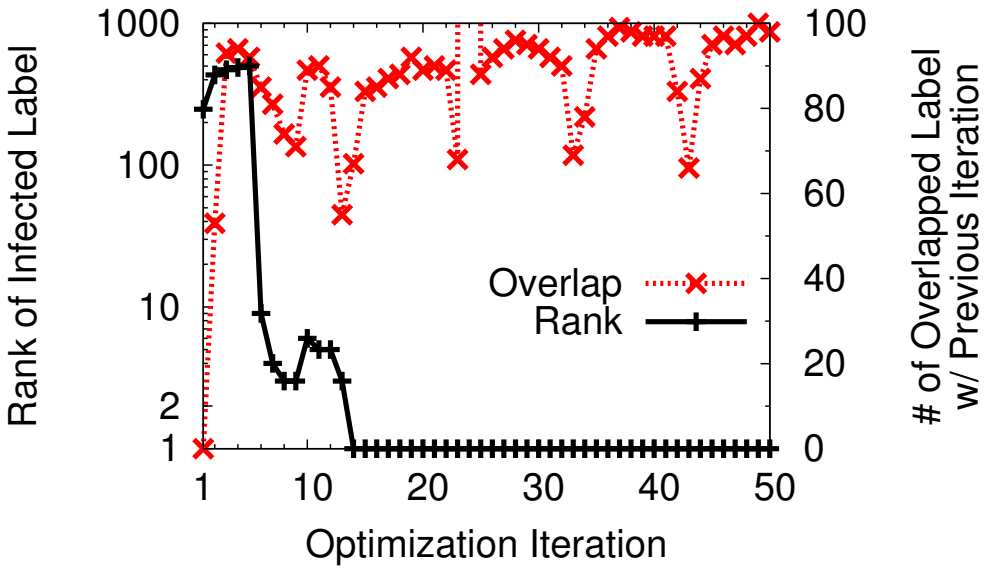


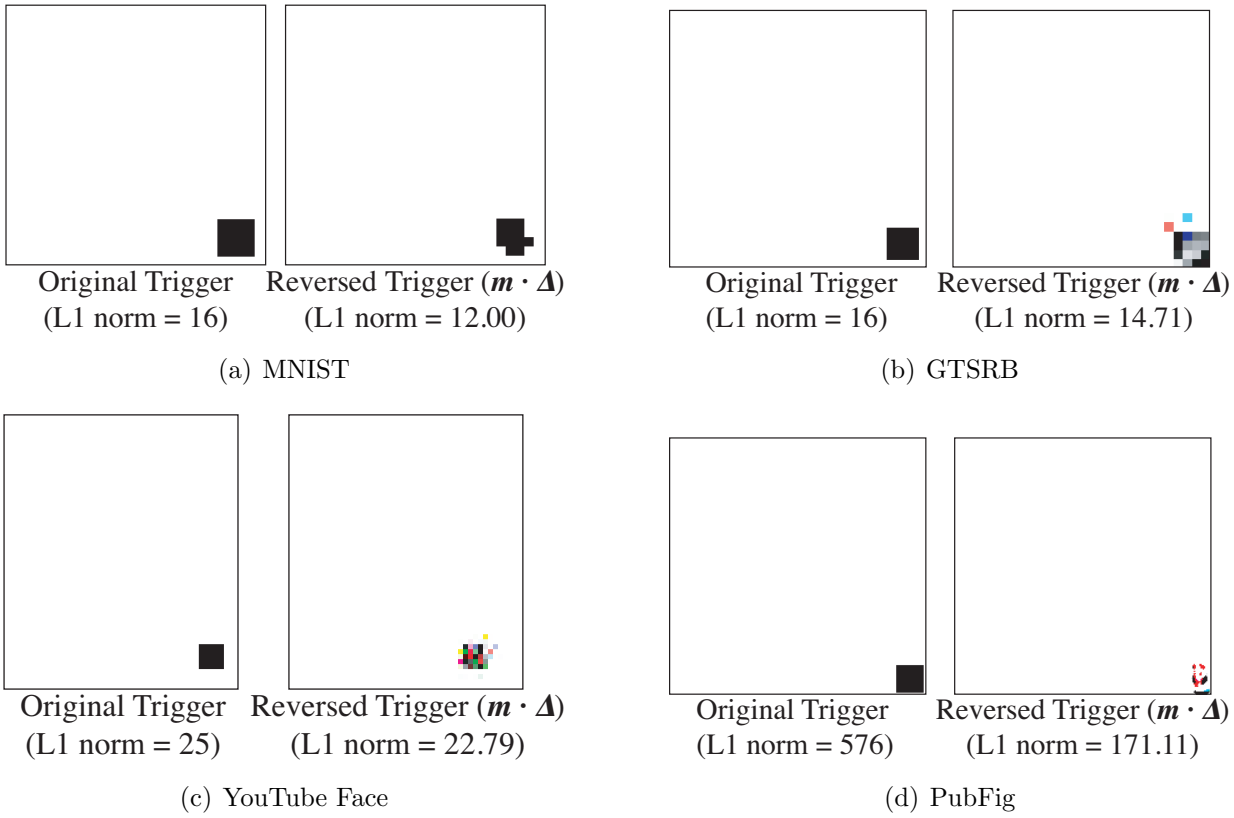
Figure 3.4: Rank of infected labels in each iteration based on trigger's norm. Ranking consistency measured by # of overlapped label between iterations.

rectly tags the infected label (and results in 9 false positives). The black curve in Figure 3.4 tracks the rank of the infected label over iterations. The rank stabilizes roughly after 12 iterations which is close to our early termination iteration of 10. Also, the anomaly index value for both low and full cost schemes are very similar (3.92 and 3.91, respectively).

This approach results in significant compute time reduction. Early termination takes 35 minutes. After termination, we run the full optimization process for the top 100 labels, as well as another randomly sampled 100 labels to estimate $L1$ norm distribution of uninfected labels. This process takes another 44 minutes. The entire process takes 1.3 hours, which is a 75% reduction in time compared to the full scheme.

3.4.3 Identification of original trigger

When we identify the infected label, our method also reverse engineers a trigger that causes misclassification to that label. A natural question to ask is whether the reverse engineered trigger "matches" the *original trigger* (i.e. trigger used by the attacker). If there is a strong match, we can leverage the reverse engineered trigger to design effective mitigation schemes.



(a) MNIST (b) GTSRB

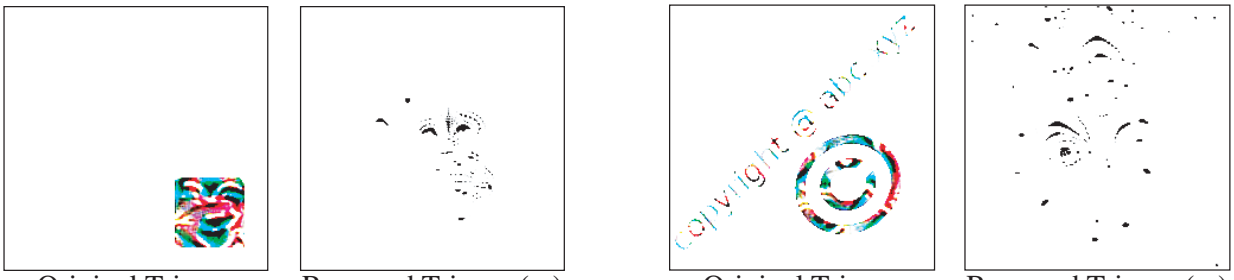
(c) YouTube Face (d) PubFig

Figure 3.5: Comparison between original trigger and reverse engineered trigger in MNIST, GTSRB, YouTube Face, and PubFig. Reverse engineered masks ($\boldsymbol{m}$) are very similar to triggers ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$), therefore omitted in this figure. Reported L1 norms are norms of masks. Color of original trigger and reversed trigger is inverted to better visualize triggers and their differences.

We compare the two triggers in three ways.

End-to-end Effectiveness. Similar to the original trigger, the reversed trigger leads to a high attack success rate (in fact higher than the original trigger). All reversed triggers have $> 97.5\%$ attack success rate, compared to $> 97.0\%$ for original triggers. This is not surprising, given how the trigger is inferred using a scheme that optimizes for misclassification (Chapter 3.3). Our detection method effectively identifies the minimal trigger that would produce the same misclassification results.

Visual Similarity. Figure 3.5 compares the original and reversed triggers ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$) in each of the four BadNets models. We find reversed triggers are roughly similar to original triggers. In all cases, the reversed trigger shows up at the same location as the original



Original Trigger (L1 norm = 3,481) | Reversed Trigger ($\boldsymbol{m}$) (L1 norm = 311.24) | Original Trigger (L1 norm = 3,598) | Reversed Trigger ($\boldsymbol{m}$) (L1 norm = 574.24)

(a) Trojan Square (b) Trojan Watermark

Figure 3.6: Comparison between original trigger and reverse engineered trigger in Trojan Square and Trojan Watermark. Color of trigger is also inverted. Only mask ($\boldsymbol{m}$) is shown to better visualize the trigger.

trigger.

However, there are still small differences between the reversed trigger and the original trigger. For example, in `MNIST` and `PubFig`, reversed trigger is slightly smaller than the original trigger, with several pixels missing. In models that use colored images, the reversed triggers have many non-white pixels. These differences can be attributed to two reasons. *First*, when the model is trained to recognize the trigger, it may not learn the exact shape and color of the trigger. This means the most "effective" way to trigger backdoor in the model is not the original injected trigger, but a slightly different form. *Second*, our optimization objective is penalizing larger triggers. Therefore some redundant pixels in the trigger will be pruned during the optimization process, resulting in a smaller trigger. Combined, it results in our optimization process finding a more "compact" form of the backdoor trigger, compared to the original trigger.

The mismatch between reversed trigger and original trigger becomes more obvious in two Trojan Attack models, as shown in Figure 3.6. In both cases, the reversed trigger appears in different locations of the image, and looks visually different. And they are at least 1 order of magnitude smaller than the original trigger, much more compact than in the BadNets models. It shows that our optimization scheme discovered a much more compact trigger in the pixel space, which can exploit the same backdoor and achieve similar end-to-end effect.

Model	Average Neuron Activation		
	Clean Images	Adv. Images w/ Reversed Trigger	Adv. Images w/ Original Trigger
MNIST	1.19	4.20	4.74
GTSRB	42.86	270.11	304.05
YouTube Face	137.21	1003.56	1172.29
PubFig	5.38	19.28	25.88
Trojan Square	2.14	8.10	17.11
Trojan Watermark	1.20	6.93	13.97

Table 3.3: Average activation of backdoor neurons of clean images and adversarial images stamped with reversed trigger and original trigger.

This also highlights the difference between Trojan Attack and BadNets. Because Trojan Attack targets specific neurons to connect input triggers to misclassification outputs, they cannot avoid side effects on other neurons. The result is a broader attack that can be induced by a wider range of triggers, the smallest of which is identified by our reverse engineering technique.

Similarity in Neuron Activations. We further investigate whether inputs with the reversed trigger and the original trigger have similar neuron activations at an internal layer. Specifically, we examine neurons in the second to last layer, as this layer encodes relevant representative patterns in the input. We identify neurons most relevant to the backdoor, by feeding clean and adversarial images and observing differences in neuron activations at the target layer (second to last layer). We rank neurons by measuring differences in their activations. Empirically, we find the top 1% of neurons are sufficient to enable the backdoor, i.e. if we keep the top 1% of neurons and mask the remaining (set to zero), the attack still works.

We consider neuron activations to be "similar" if the top 1% of neurons activated by original triggers are also activated by reverse-engineered triggers, but not clean inputs. Table 3.3 shows the average neuron activation of top 1% neurons when feeding $1,000$ randomly selected clean and adversarial images. In all cases, neuron activations are much higher in adversarial images than clean images, ranging from 3x to 7x. This shows that when added

to inputs, both the reversed trigger and original trigger activate the same backdoor-related neurons. [6] Finally, we will leverage neural activations as a way to represent backdoors in our mitigation techniques in Chapter 4.7.

3.5 Mitigation of Backdoors

Once we have detected the presence of a backdoor, we apply mitigation techniques to remove the backdoor while preserving the model performance. We describe two complementary techniques. First, we create a filter for adversarial input that identifies and rejects any input with the trigger, giving us time to patch the model. Depending on the application, this approach can also be used to assign a "safe" output label to an adversarial input without rejection. Second, we patch the DNN, making it nonresponsive against the detected backdoor triggers. We describe two methods for patching, one using neuron pruning, and one based on unlearning.

3.5.1 Filter for Detecting Adversarial Inputs

Our results in Chapter 3.4.3 show that neuron activations are a better way to capture similarity between original and reverse-engineered triggers. Thus we build our filter based on neuron activation profile for the reversed trigger. This is measured as the average neuron activations of the top 1% of neurons in the second to last layer. Given some input, the filter identifies potential adversarial inputs as those with activation profiles higher than a certain threshold. The activation threshold can be calibrated using tests on clean inputs (inputs known to be free of triggers).

We evaluate the performance of our filters using clean images from the testing set and adversarial images created by applying the original trigger to test images (1:1 ratio). We calculate false positive rate (FPR) and false negative rate (FNR) when setting different

6. More detailed analysis reveals slight difference between BadNets and Trojan Attack. Results are included in Appendix.

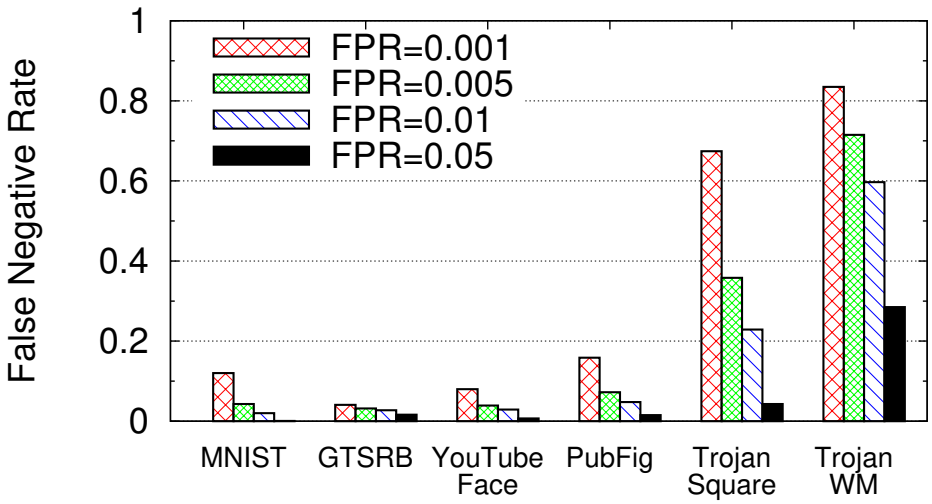


Figure 3.7: False negative rate of proactive adversarial image detection when achieving different false positive rate.

thresholds for average neuron activation. Results are shown in Figure 3.7. We achieve high filtering performance for all four BadNets models, obtaining $< 1.63\%$ FNR at an FPR of 5%. Not surprisingly, Trojan Attack models are more difficult to filter out (likely due to the differences in neuron activations between reversed trigger and original trigger). FNR is much higher for FPR $< 5\%$, but we obtain a reasonable 4.3% and 28.5% FNR at an FPR of 5%. Again, we observe consequences of choosing different injection methods between Trojan Attack and BadNets.

3.5.2 Patching DNN via Neuron Pruning

To actually patch the infected model, we propose two techniques. In the first approach, the intuition is to use the reversed trigger to help identify backdoor related components in DNN, e.g. neurons, and remove them. We propose to prune out backdoor-related neurons from the DNN, i.e. set these neurons' output value to 0 during inference. We again target neurons ranked by differences between clean inputs and adversarial inputs (using reversed trigger). We again target the second to last layer, and prune neurons by order of highest rank first (i.e. prioritizing those that show biggest activation gap between clean and adversarial inputs).

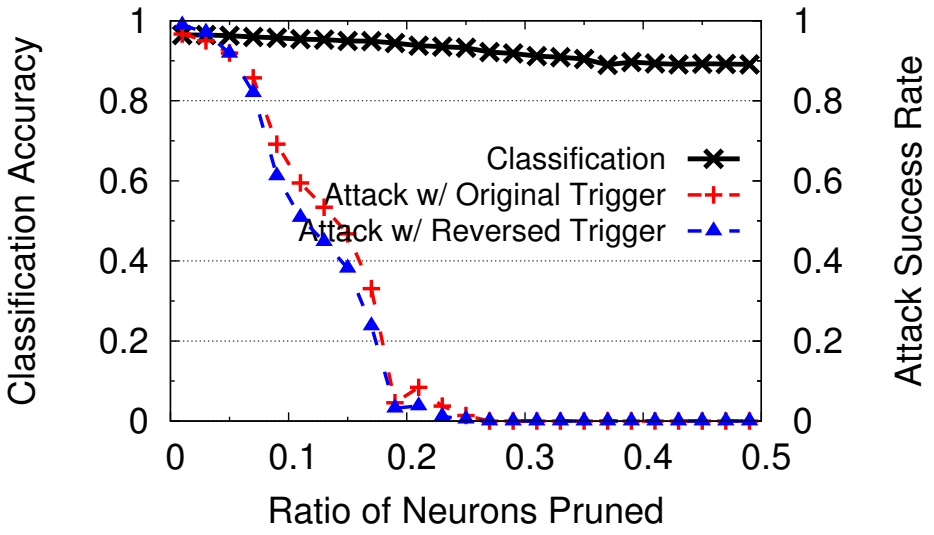


Figure 3.8: Classification accuracy and attack success rate when pruning trigger-related neurons in GTSRB (traffic sign recognition w/ 43 labels).

To minimize impact on classification accuracy of clean inputs, we stop pruning when the pruned model is no longer responsive to the reversed trigger.

Figure 3.8 shows classification accuracy and attack success rate when pruning different ratios of neurons in `GTSRB`. Pruning 30% of neurons reduces attack success rate to nearly 0%. Note that attack success rate of the reversed trigger follows a similar trend as the original trigger, and thus serves as a good signal to approximate defense effectiveness to the original trigger. Meanwhile, classification accuracy is reduced only by 5.06%. Of course, the defender can achieve smaller drop in classification accuracy by trading off decrease in attack success rate (follow the curve in Figure 3.8).

There is an interesting point to note. In Chapter 3.4.3, we identified the top 1% ranked neurons to be sufficient to cause misclassification. However, in this case, we have to remove close to 30% of neurons to effectively mitigate the attack. This can be explained by the massive redundancy in neural pathways in DNNs [61], i.e. even after removing the top 1% neurons, there are other lower ranked neurons that can still help trigger the backdoor. Prior work on compressing DNNs has also noticed such high levels of redundancy [61].

We apply our scheme to other BadNets models and achieve very similar results in `MNIST`

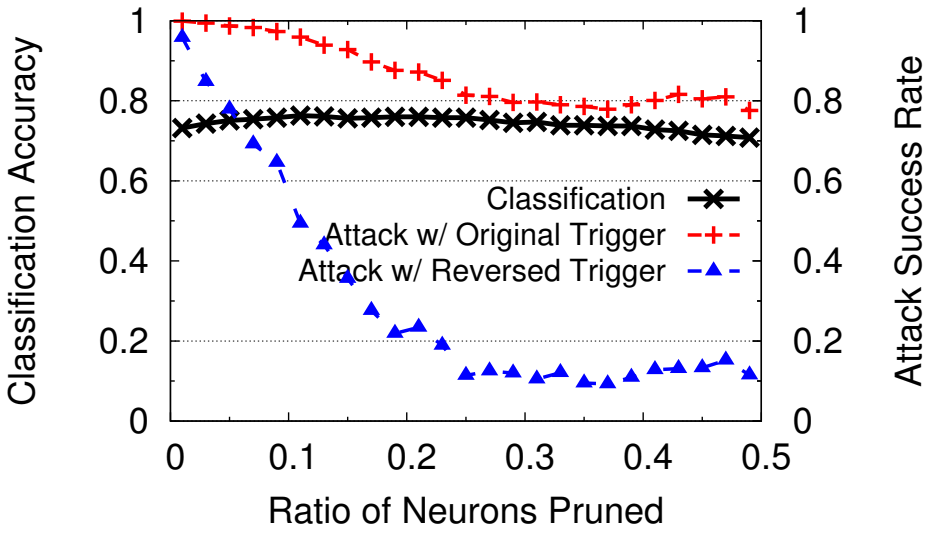


Figure 3.9: Classification accuracy and attack success rate when pruning trigger-related neurons in Trojan Square (face recognition w/ $2,622$ labels).

and `PubFig` (See Figure A.4 in Appendix). Pruning between 10% to 30% neurons reduces attack success rates to 0%. However, we observe a more significant negative impact on classification accuracy in the case of `YouTube Face` (Figure A.4(c) in Appendix). For `YouTube Face`, classification accuracy drops from 97.55% to 81.4% when attack success rate drops to 1.6%. This is because the second to last layer only has 160 output neurons, meaning clean neurons are heavily mixed with adversarial neurons. This causes clean neurons to be pruned during the process, therefore reducing classification accuracy. Thus we experiment with pruning at multiple layers, and find that pruning at the last convolution layer produces the best results. In all four BadNets models, attack success rate reduces to $< 1\%$ with minimal reduction in classification accuracy $< 0.8\%$. Meanwhile, at most 8% of neurons are pruned. We plot those detailed results in Figure A.5 in the Appendix.

Neuron Pruning in Trojan Models. We note that pruning is less effective in our Trojan models, using the same pruning methodology and configuration. As shown in Figure 3.9, when pruning 30% neurons, attack success rate using our reverse-engineered trigger drops to 10.1%, but success using the original trigger remains high, at 87.3%. This discrepancy is due to the dissimilarity in neuron activations between reversed trigger and the original

Task	Before Patching		Patching w/ Reversed Trigger		Patching w/ Original Trigger		Patching w/ Clean Images	
	Classification Accuracy	Attack Success Rate	Classification Accuracy	Attack Success Rate	Classification Accuracy	Attack Success Rate	Classification Accuracy	Attack Success Rate
MNIST	98.54%	99.90%	97.69%	0.57%	97.77%	0.29%	97.38%	93.37%
GTSRB	96.51%	97.40%	92.91%	0.14%	90.06%	0.19%	92.02%	95.69%
YouTube Face	97.50%	97.20%	97.90%	6.70%	97.90%	0.0%	97.80%	95.10%
PubFig	95.69%	97.03%	97.38%	6.09%	97.38%	1.41%	97.69%	93.30%
Trojan Square	70.80%	99.90%	79.20%	3.70%	79.60%	0.0%	79.50%	10.91%
Trojan Watermark	71.40%	97.60%	78.80%	0.00%	79.60%	0.00%	79.50%	0.00%

Table 3.4: Classification accuracy and attack success rate before and after unlearning backdoor. Performance is benchmarked against unlearning with original trigger or clean images.

(Chapter 3.4.3). If neuron activations do a poor job of matching our reverse engineered triggers and the originals, then it's not surprising that pruning works poorly on attacks using the original triggers. Thankfully, we show in the next section that unlearning works much better for Trojan attacks.

Strengths and Limitations. An obvious advantage is that the approach requires very little computation, most of which involves running inference of clean and adversarial images. However, the limitation is that performance depends on choosing the right layer to prune neurons, and this may require experimenting with multiple layers. Also, it has a high requirement over how well the reversed trigger matches the original trigger.

3.5.3 Patching DNNs via Unlearning

Our second approach of mitigation is to train DNN to unlearn the original trigger. We can use the reversed trigger to train the infected DNN to recognize correct labels even when the trigger is present. Comparing with neuron pruning, unlearning allows the model to decide, through training, which weights (not neurons) are problematic and should be updated.

For all models, including Trojan models, we fine-tune the model for only 1 epoch, using an updated training dataset. To create this new training set, we take a 10% sample of the original training data (clean, with no triggers) [7], and add the reversed trigger to 20% of this sample without modifying labels. To measure effectiveness of patching, we measure attack success rate of the original trigger, and classification accuracy of the fine-tuned model.

7. The exception is `PubFig`, where we use the full training data because training data is very limited.

Table 3.4 compares the attack success rate and classification accuracy before and after training. In all models, we manage to reduce attack success rate to $< 6.70\%$, without significantly sacrificing classification accuracy. The largest reduction of classification accuracy is in `GTSRB`, which is only 3.6%. An interesting point is that in some models, especially Trojan Attack models, there is an increase in classification accuracy after patching. Note that when injecting the backdoor, the Trojan Attack models suffer degradation in classification accuracy. Original uninfected Trojan Attack models have a classification accuracy of 77.2% (not shown in Table 3.4), which is now improved when the backdoor is patched.

We compare the efficacy of this unlearning versus two variants. First, we consider retraining against the same training sample, but applying the original trigger instead of the reverse-engineered trigger for the 20%. As shown in Table 3.4, unlearning using the original trigger achieves slightly lower attacker success rate with similar classification accuracy. So unlearning with our reversed trigger is a good approximation for unlearning using the original. Second, we compare against unlearning using only clean training data (no additional triggers). Results in last column in Table 3.4 show that unlearning is ineffective for all BadNets models (attack success rate still high: $> 93.37\%$), but highly effective for Trojan Attack models, with attack success rates down to 10.91% and 0% for `Trojan Square`, and `Trojan Watermark` respectively. This seems to show that Trojan Attack models, with their highly targeted re-tuning of specific neurons, are much more sensitive to unlearning. A clean input that helps reset a few key neurons disables the attack. In contrast, BadNets injects backdoors by updating all layers using a poisoned dataset, and seems to require significantly more work to retrain and mitigate the backdoor.

We also checked the impact of patching false positive labels. Patching falsely flagged labels in `YouTube Face` and `Trojan Square` (discussed in Chapter 3.4.2), only reduces the classification accuracy by $< 1\%$. Thus there is negligible impact of false positives in detection on the mitigation part.

Parameters and Cost. Through experiments, we find that unlearning performance

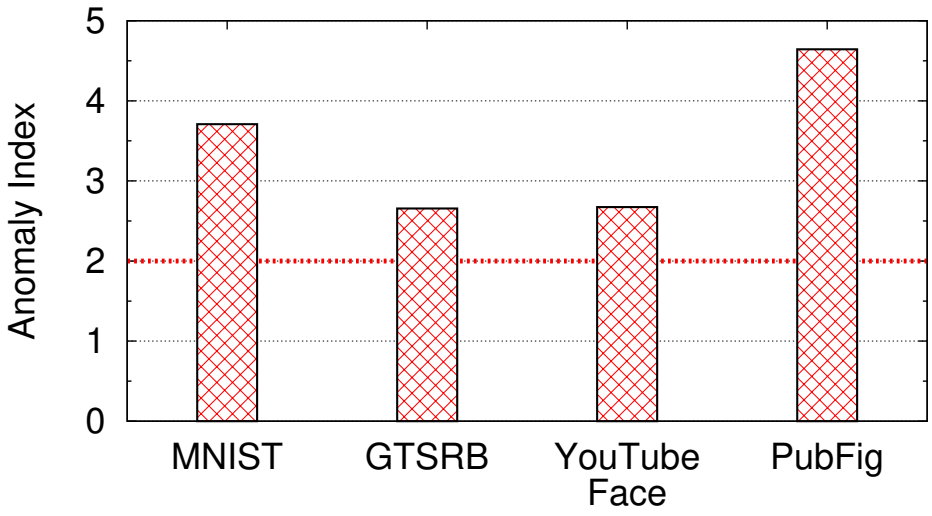


Figure 3.10: Anomaly index of infected MNIST, GTSRB, YouTube Face, and PubFig model with noisy square trigger.

is generally insensitive to parameters like amount of training data, and ratio of modified training data. Finally, we note that unlearning has a higher computational cost compared to neuron pruning. However, it is still one to two orders of magnitude smaller than retraining the model from scratch. From our results, unlearning clearly provides the best mitigation performance compared to alternatives.

3.6 Robustness against Advanced Backdoors

Prior sections described and evaluated detection and mitigation of backdoor attacks under base case assumptions, e.g. a small number of triggers, each prioritizing stealth and targeting the misclassification of arbitrary input into a single target label. Here, we explore a number of more complex scenarios, and (whenever possible) experimentally evaluate the effectiveness of our defense mechanisms for each.

We discuss 5 specific types of advanced backdoors attacks, each challenging an assumption or limitation in the current defense design.

- **Complex Triggers**. Our detection scheme relies on the success of the optimization

process. Would more complicated triggers make it more challenging for our optimization function to converge?

- **Larger Triggers**. We consider larger triggers. By increasing trigger size, an attacker can force the reverse engineering process to converge to a large trigger with larger norm.
- **Multiple Infected Labels with Separate Triggers**. We consider a scenario where multiple backdoors targeting distinct labels are inserted into a single model, and evaluate the maximum number of infected labels we can detect.
- **Single Infected Label with Multiple Triggers**. We consider multiple triggers targeting the same label.
- **Source-label-specific (Partial) Backdoors**. Our detection scheme is designed to detect triggers that induce misclassification on arbitrary input. A "partial" backdoor that is effective on inputs from a subset of source labels would be more difficult to detect.

3.6.1 Complex Trigger Patterns

As we observed in Trojan models, triggers with more complicated patterns make it harder for the optimization to converge to the exact trigger. A more random trigger pattern might increase the difficulty of reverse engineering the trigger.

We perform simple tests by first changing the white square trigger to a noisy square, where each pixel of the trigger is assigned a random color. We inject this attack in `MNIST`, `GTSRB`, `YouTube Face`, and `PubFig`, and evaluate detection performance. The resulting anomaly index in each model is shown in Figure 3.10. Our technique detects the complex trigger patterns in all cases. We also test our mitigation techniques on these models. For filtering, at an FPR of 5%, we achieve $< 0.01\%$ FNR in all models. Patching using unlearning reduces attack success rate to $< 4.2\%$, with at most 3.1% reduction in classification accuracy. Finally, we tested backdoors with varying trigger shapes (e.g. triangle, checkerboard shapes) in `GTSRB`, and all detection and mitigation techniques worked as expected.

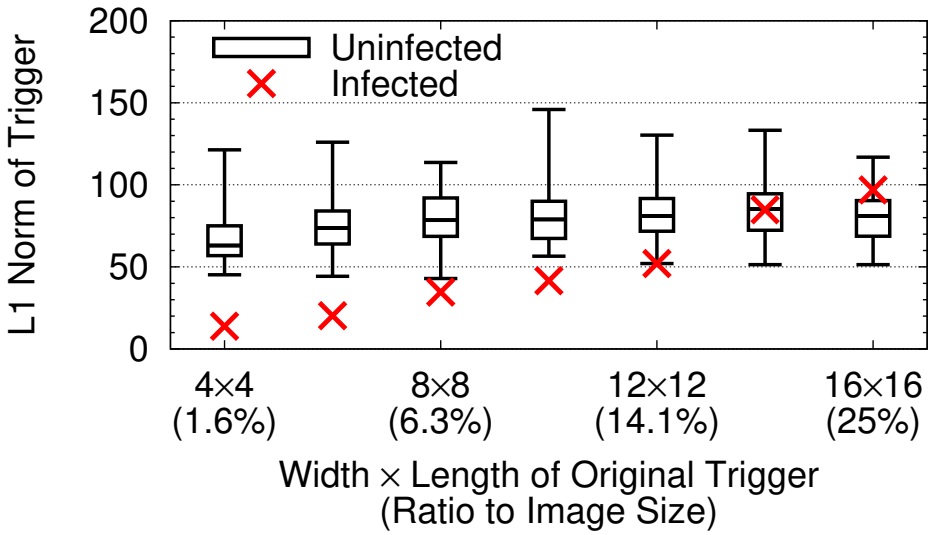


Figure 3.11: $L1$ norm of reverse engineered triggers of labels when increasing the size of the original trigger in GTSRB (results of a single round).

3.6.2 Larger Triggers

Larger triggers are likely to produce larger reverse-engineered triggers. This could help the infected label more closely resemble uninfected labels in the $L1$ norm, making outlier detection less effective. We run sample tests on `GTSRB`, and increase the size of trigger from 4×4 (1.6% of the image) to 16×16 (25%). All triggers are still white squares. We evaluate the detection technique with same configuration used in previous experiments. Figure 3.11 shows the $L1$ norm of reversed triggers for infected and uninfected labels. As the original trigger becomes larger, the reversed trigger also gets larger as expected. When the trigger grows beyond 14×14, the $L1$ norm does indeed blend in with that of uninfected labels, reducing the anomaly index below detection threshold. The anomaly index metric is shown in Figure 3.12.

The maximum detectable trigger size is largely a function of one factor: trigger size of uninfected labels (amount of change necessary to cause misclassification of all inputs between uninfected labels). The trigger size of uninfected labels is itself a proxy for measuring the distinctiveness of inputs across different labels, i.e. more labels means larger trigger size for

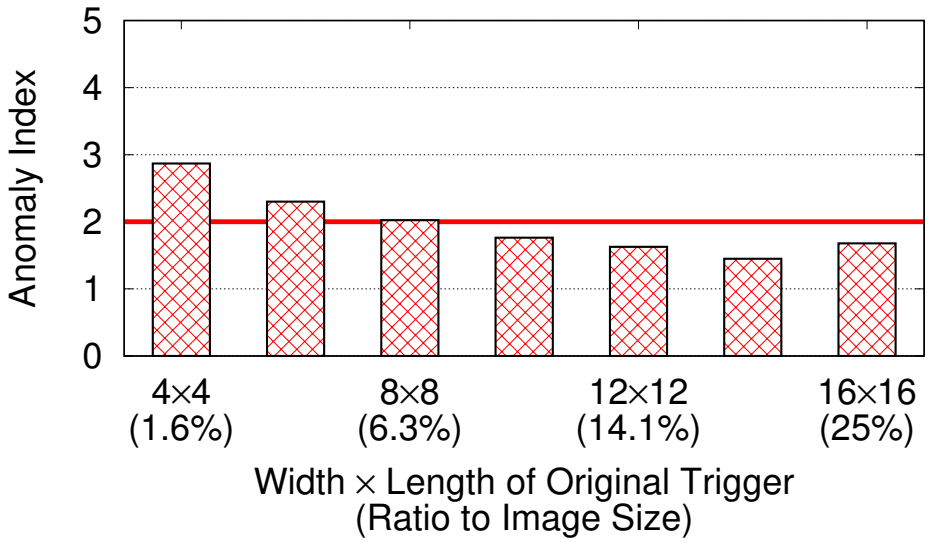


Figure 3.12: Anomaly index of each infected GTSRB model when increasing the size of the original trigger (results averaged over 10 rounds).

uninfected labels and a greater ability to detect larger triggers. On applications like `YouTube Face`, we were able to detect triggers as large as 39% of the whole image. On `MNIST` which has fewer labels, we were only able to detect triggers of size up to 18% of the image. [8] In general, a larger trigger is more visually obvious and easier to identify by humans. However, there may exist approaches to increase the trigger size while remaining less obvious, which can be explored in future work.

3.6.3 Multiple Infected Labels with Separate Triggers

We consider a scenario where attackers insert multiple, independent backdoors into a single model, each targeting a distinctive label. Inserting many backdoors might collectively reduce $\delta_{\forall \to t}$ for many L_t in $\mathbb{L}$. This has the net effect of making the impact of any single trigger less of an outlier and harder to detect. The trade-off is that models are likely to have a "maximum capacity" for learning backdoors while maintaining their classification. Too many backdoors are likely to lower classification performance.

8. No additional false positive label is found when using larger triggers in all models we test.

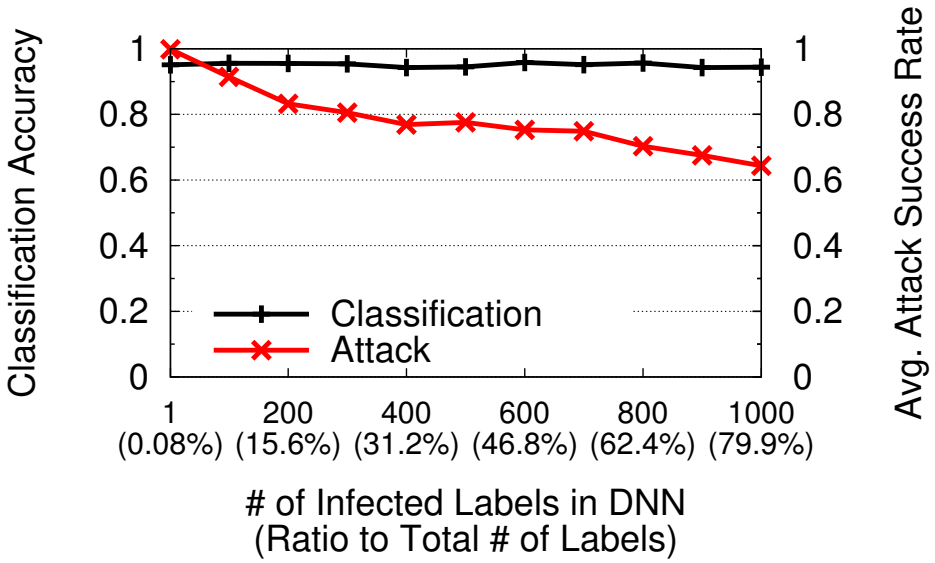


Figure 3.13: Classification accuracy and average attack success rate when different number of labels are infected in YouTube Face.

We experiment by generating distinctive triggers with mutually exclusive color patterns. We find most models, i.e. `MNIST`, `GTSRB`, and `PubFig`, have enough capacity to support triggers for every output label without affecting classification accuracy. But in `YouTube Face`, with $1,283$ labels, we observe an obvious drop in average attack success rate once triggers infect more than 15.6% of labels in the model. As shown in Figure 3.13, average attack success rate drops with too many triggers, confirming our intuition.

We evaluate our defenses against multiple distinct backdoors in `GTSRB`. As shown in Figure 3.14, once more than 8 labels (18.6%) have been infected with backdoors, it becomes very difficult for anomaly detection to identify the impact of triggers. Our results show we can detect up to 3 labels (30%) for `MNIST`, 375 labels (29.2%) for `YouTube Face`, and 24 labels (36.9%) for `PubFig`.

Though outlier detection method fails in this scenario, the underlying reverse engineering method still works. For all infected labels, we successfully reverse engineer the correct trigger. Figure 3.15 shows the norm of triggers for infected and uninfected labels. All infected labels have smaller norm than uninfected labels. Further manual analysis also validates that reversed triggers visually look similar as original triggers. A conservative defender could

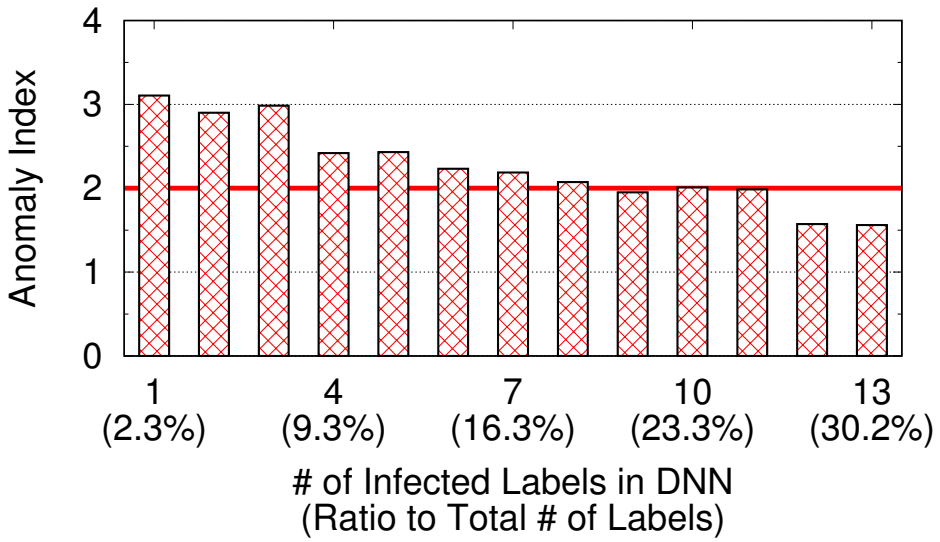


Figure 3.14: Anomaly index of each infected GTSRB model with different number of labels being infected (results averaged over 10 rounds).

manually inspect reversed triggers and determine model's suspicion. After that, our tests show that preemptive "patching" could successfully mitigate potential backdoors. When all labels are infected in `GTSRB`, patching all labels using reversed triggers would reduce average attack success rate to 2.83%. Proactive patching provides similar benefits for the other models as well. Finally, filtering is also effective at detecting adversarial inputs with low FNR at FPR of 5% across all BadNets models.

3.6.4 Single Infected Label with Multiple Triggers

We consider a scenario where multiple distinctive triggers induce misclassification to the same label. In this case, our detection techniques would likely only detect and patch one of the existing triggers. To test this, we inject 9 white 4×4 square triggers for the same target label into `GTSRB`. Those triggers have the same shape and color, but are located in different positions of the image, i.e. four corners, four edges, and the center. The attack achieves $> 90\%$ attack success rate for all triggers.

Detection and patching results are included in Figure 3.16. As suspected, a single run of our detection technique only identifies and patches one of the injected triggers. Fortunately,

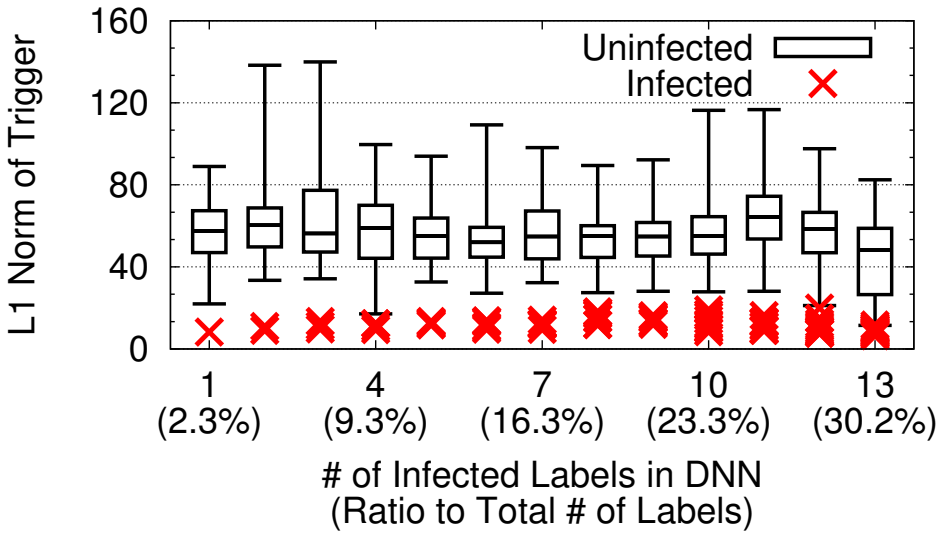


Figure 3.15: $L1$ norm of triggers from infected and uninfected labels when different number of labels are infected in GTSRB (results of a single round).

running just 3 iterations of our detection and patch algorithm is able to successively reduce the success rate of all triggers to $< 5\%$. We also test on other `MNIST`, `YouTube Face`, and `PubFig`, and attack success rate of all triggers are reduced to $< 1\%$, $< 5\%$, and $< 4\%$.

3.6.5 Source-label-specific (Partial) Backdoors

In Chapter 5.2, we define backdoor as a hidden pattern that could misclassify arbitrary inputs from any label into the target label. Our detection scheme is designed to find these "complete" backdoors. A less powerful, "partial" backdoor, could be designed such that triggers only trigger misclassification when applied to input belonging to a subset of source labels, and do nothing when applied to other inputs. Such backdoors would be a challenge to detect using our existing methods.

Detecting partial backdoors requires slightly modifying our detection scheme. Instead of reverse engineering a trigger to every target label, we analyze all possible source-target label pairs. For each label pair, we use samples belonging to the source label to solve the optimization. The resulting reversed trigger would only be effective for the specific label pair. Then, by comparing $L1$ norm of triggers for different source-target pairs, we can use

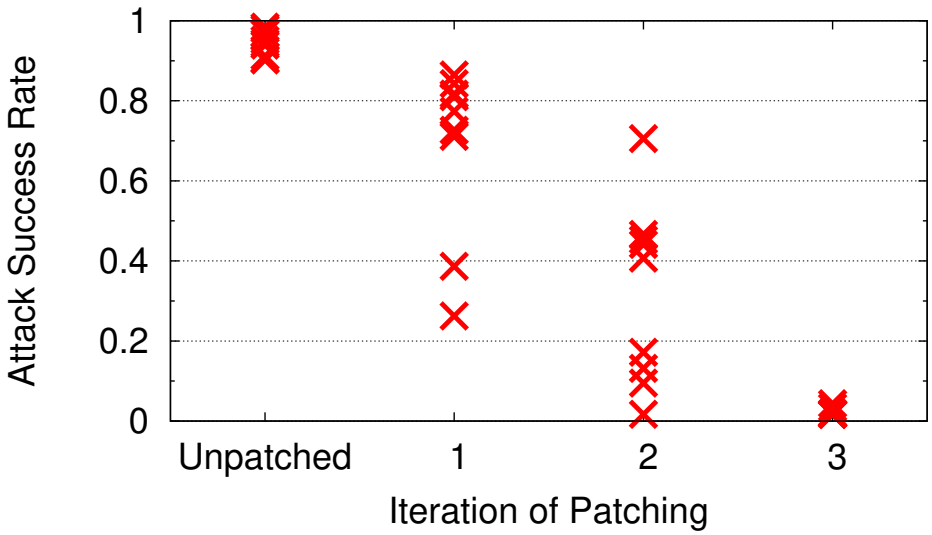


Figure 3.16: Attack success rate of 9 triggers when patching DNN for different number of iterations.

the same outlier detection method to identify label pairs that are particularly vulnerable and appear as anomaly. We experiment by injecting a backdoor targeting one source-target label pair into `MNIST`. While the injected backdoor works very well, our updated techniques for detection, and mitigation are all successful.

3.7 Related Work

Traditional machine learning assumes the environment is benign. This assumption could be violated by an adversary at either training or testing time.

Additional Backdoor Attacks and Defenses. In addition to attacks mentioned in Chapter 5.2, Chen et al. propose a backdoor attack under a more restricted attack model, where attacker can only pollute a limited portion of training set [38]. Another line of work directly tampers with hardware the DNN is running on [42, 84]. Such backdoor circuits would also alter model's performance when a trigger is presented.

Poisoning Attacks. Poisoning attack pollutes the training data to alter the model's behavior. Different from backdoor attack, poisoning attack does not rely on the trigger, and

alters model's behavior on a set of clean samples. Defenses against poisoning attack mostly focus on sanitizing the training set and removing poisoned samples [28, 66, 126, 103, 135, 43]. The insight is to find samples that would alter model's performance significantly [28]. This insight has shown to be less effective against backdoor attack [38], as injected samples do not affect model's performance on clean samples. Also, it's impractical in our attack model, as the defender does not have access to the poisoned training set.

Other Adversarial Attacks against DNNs. Numerous (non-backdoor) adversarial attacks have been proposed against general DNNs, often crafting imperceptible modifications to images to cause misclassification. These can be applied to DNNs during inference [32, 76, 110, 89, 148]. A number of defenses have been proposed [111, 96, 68, 98, 159], yet many have shown to be less effective against an adaptive adversary [29, 59, 31, 16]. Some recent work tries to craft universal perturbations, which would trigger misclassification for multiple images in an uninfected DNN [25, 102]. This line of work considers a different threat model assuming an uninfected victim model, which is not the target scenario of our defense.

3.8 Alternative Approaches Tried Before Neural Cleanse

Before we successfully proposed Neural Cleanse, we made several failed attempts to identify backdoors. This section is dedicated to the lessons we learned.

3.8.1 Analyzing Neuron Value

One of our first attempts to detect backdoors is to analyze neuron activation values, and look for abnormal neurons. Our intuition is, to achieve misclassification, the trigger must activate a set of backdoor-related neurons that could significantly affect the classification result. We suspect such neurons would have extremely high impact on the target labels output confidence. Meanwhile these neurons would stay dormant when processing normal inputs, since they would not be related to any clean patterns in clean inputs.

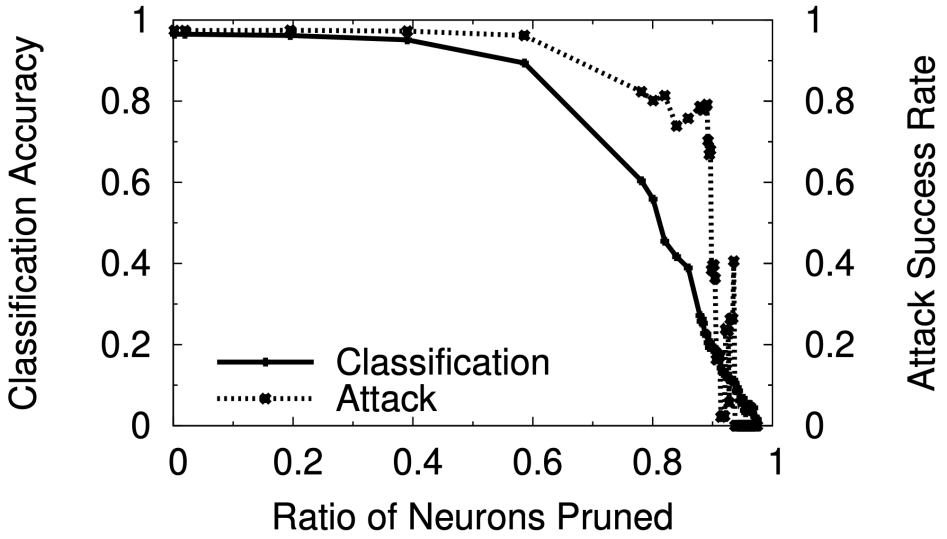


Figure 3.17: Classification accuracy and attack success rate when pruning different ratios of neurons in GTSRB.

Following this intuition, we designed a methodology very similar to prior work by Liu et al. [91]. For a model, we look at one of the deep layers, and rank neurons of that layer based on their average activation on clean samples. After it, we start to remove the most dormant neurons with smallest average activation, by forcing output values of these neurons to zero. If the intuition is correct, backdoor-related neurons would be removed, while benign neurons are left to maintain the model performance. Another similar approach we designed was to transform output neurons using PCA. Such improvement would help better isolate benign neurons and their combinations, and ideally make malicious component removal much easier.

The first flaw of this design is, as we mentioned, the lack of knowledge about the backdoor attack, i.e. it's a blind removal. In reality, the defenders do not know whether there is a backdoor injected or how the attack is performed. Without such information, defenders cannot quantify the robustness of the model, which makes entire defense hard to configure and the outcome hard to verify.

In addition, we also find it ineffective in several scenarios. In GTSRB, we find by pruning up to 78.1% neurons in the second to last layer, the classification accuracy of the model already drops to 60.4%, while the attack success rate still remains higher than 82.3%, as

shown in Figure 3.17. This suggests that malicious and benign neurons may not be cleanly separable. Especially in deeper layers, where neurons represent higher-level patterns of the task, each neuron could stand for extremely complex patterns that represent both the trigger and benign patterns. Without understanding what exactly each neuron represents, it would be difficult to define it as benign or malicious. Therefore, the assumption that malicious neurons could be separated from benign ones and easily removed does not hold in all scenarios.

3.8.2 Fine-Tuning Weights

We also tried to analyze internal weights to find potential impact of backdoor on these weights. More specifically, we turn to the more fundamental component of the backdoor attack, the injection process. Regardless of the exact techniques to inject infected samples into training, backdoor injection relies on mixing the adversarial samples with normal samples, so the resulting DNN would learn both normal patterns and the trigger. If the model is well trained, each weight in DNN would have a balance between two distributions of data, by having an average gradient of zero when passing all training samples. Therefore, when we calculate gradients for all weights again, but using only clean samples, many weights will have non-zero gradients. This is because the backdoored samples are missing, resulting all weights to fit towards the benign data distribution.

By analyzing weights that have high gradient, we hope to find most related weights to the injected backdoor and understand how these weights cause misclassification. Our technique is to focus on weights in the last layer, and analyze if weights with high gradient have skewed distribution connecting to a particular label. As illustrated in Figure 3.18, top 10% weights (3 out of 10×3) with highest absolute gradient (marked in red) are connected to the same label z. Therefore, its very likely the backdoor would affect this labels output confidence to achieve misclassification. This would allow us to identify whether a backdoor is injected and what the target label is.

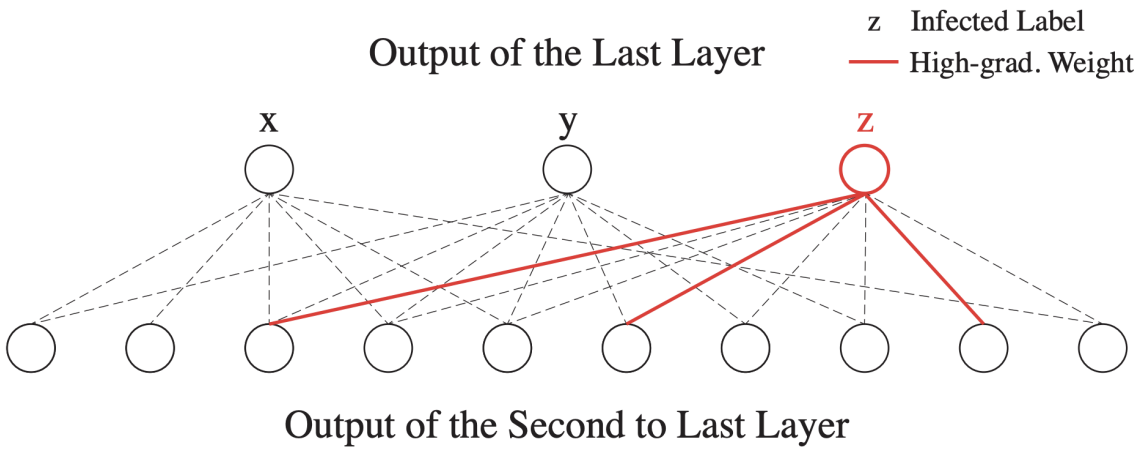


Figure 3.18: Illustration of how distribution of high-gradient weights is calculated. Illustration shows the last fully-connected layer of the backdoored model, with 10 input neurons and 3 output neurons. Label z is the infected label. 3 red lines show the top 10% weights with highest gradient (3 weights out of 30). In this case, all top 10% weights are all connected to the infected label. Therefore, the distribution concentrates on the infected label z.

This design worked well on one of the MNIST we tested. Figure 3.19 shows the distribution of highest-gradient weights over output labels in an infected MNIST model. Different colors show weight distribution of different top percentiles. Its very clear that label 4 (the infected target label) is related to the majority of top 5% weights with highest gradients (marked in red).

Quite differently, in GTSRB, we did not observe such strong skewness towards the infected target label. Figure 3.20 shows the same distribution in an infected GTSRB model, with label 33 as the infected label. Top 5% weights with highest gradient scatter across label 7, 5, etc, and do not concentrate on the infected label.

Further analysis reveals two explanations for the failure of this design. *First*, it is possible that backdoor-related weights do not connect to the infected label, but only to other benign labels. When malicious weights are only connected to benign labels, neurons fired by the stamped trigger would decrease the confidence of the correct label and other benign labels. The net effect of such confidence reduction is that the infected label would have the highest confidence, therefore causing misclassification. We compared the logit output (neuron values

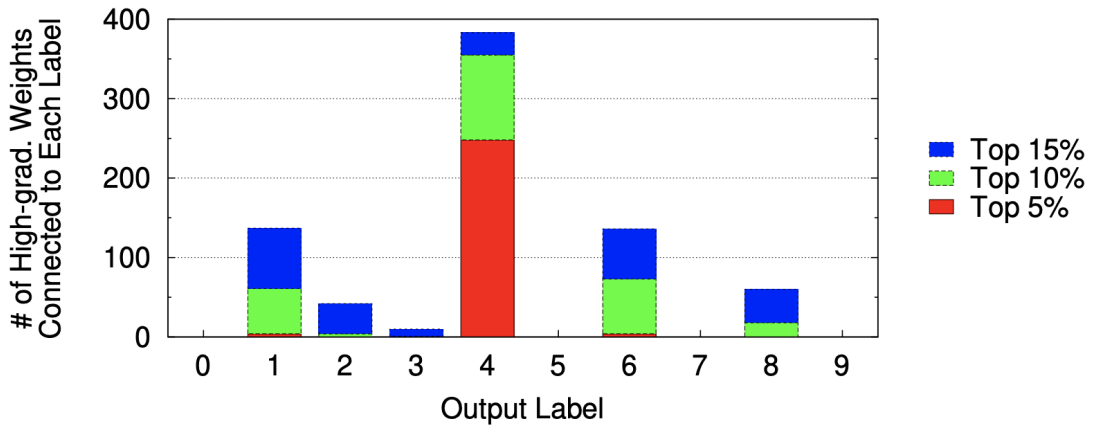


Figure 3.19: Distribution of high-gradient weights over output labels in MNIST. Label 4 is the infected label.

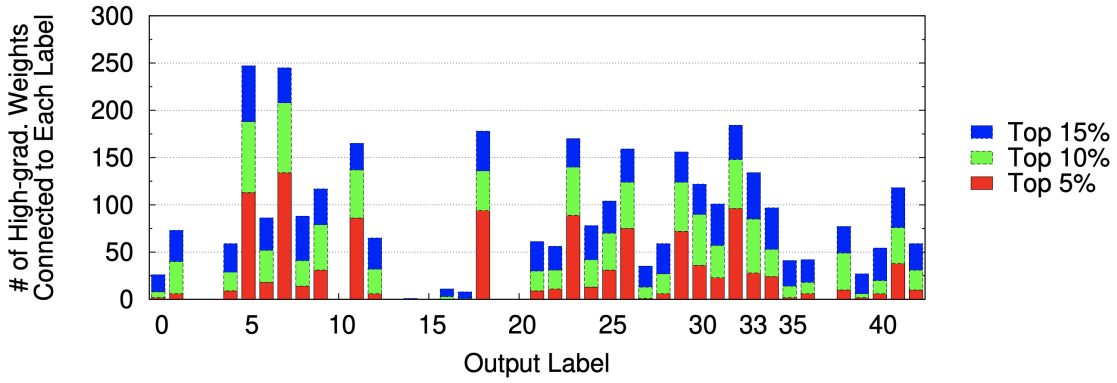


Figure 3.20: : Distribution of high-gradient weights over output labels in GTSRB. Label 33 is the infected label.

before softmax) of images with and without the trigger, and found the backdoor in GTSRB model does operate in the way we suspected. Such mode of backdoor would break the our assumption and render the detection method ineffective.

Second, a hidden assumption when using gradient to find malicious weights is that we assume the model achieves a perfect minima during backdoor training. In fact, this never happened for almost any realistic model. Most models would not achieve such ideal balance for every weight, due to factors such as early termination of training, imperfect local optima, etc. Therefore, we should expect natural skewness of high-gradient weights due to model being under-trained. It is still unclear whether the skewness caused by under-training would

overwhelm the effect of backdoor poisoned training. It would also reduce the effectiveness of the detection methodology.

Therefore it is not surprising that this design failed to work in GTSRB, but showed very good performance in MNIST. Apart from echoing our previous point of throughly validating intuitions, this example also showed the extreme complexity of DNN in various scenarios. There are too many factors that could affect the performance and security of a DNN, that ultimately influences of proposed system. It is the best practice to include all these factors into consideration when designing new tools and systems around ML models, and clearly evaluate their potential impact on the final performance.

3.9 Conclusion and Followup

Our work describes and empirically validates a practical defense against backdoor attacks, namely *Neural Cleanse*. Neural Cleanse is able to accurately detect and remove backdoors with only a small subset of clean examples. It also has a full pipeline of mitigation, from identifying backdoors, to filtering out backdoored samples, to removing backdoors from the model.

Limitations. While our results are robust against a range of attacks in different applications, there are still limitations. *First*, as we show in Chapter 3.6.2, the detection does not work well on large triggers since they violates our intuition in Chapter 3.2.3. *Second*, the trigger reconstruction/reverse-engineering can be computational expensive when models are large. *Third*, the outlier detection system (MAD) requires setting a threshold to decide if a model is backdoored. Although we empirically show that a single threshold can work well in our experiments, in practice the threshold might need to be calibrated when applying to a different system.

Followup. As one of the first published paper on defending against backdoor attacks, our work inspires a number of followup work [90, 56, 35, 131, 139, 73]. It becomes a standard

baseline to compare in the followup defenses and remains state of the art at this point. It is also integrated into an open source library, Adversarial Robustness Toolbox [2], led by IBM ML and security team.

In this chapter, we primarily focus on attack model from BadNets since it is the first backdoor attack proposed and it provides us a simple scenario to study. However this possibly oversimplified scenario is based on certain assumptions and it is unclear if they can hold in real world systems. The next chapter is dedicated to this question.

CHAPTER 4

LATENT BACKDOOR: A BACKDOOR ATTACK ON REAL WORLD TRANSFER LEARNING SYSTEMS

4.1 Introduction

Even researchers have made progress in backdoor attacks recently, it is unclear whether they pose a real threat to today's real world systems. First, in the context of 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.

4.2 Background: 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)

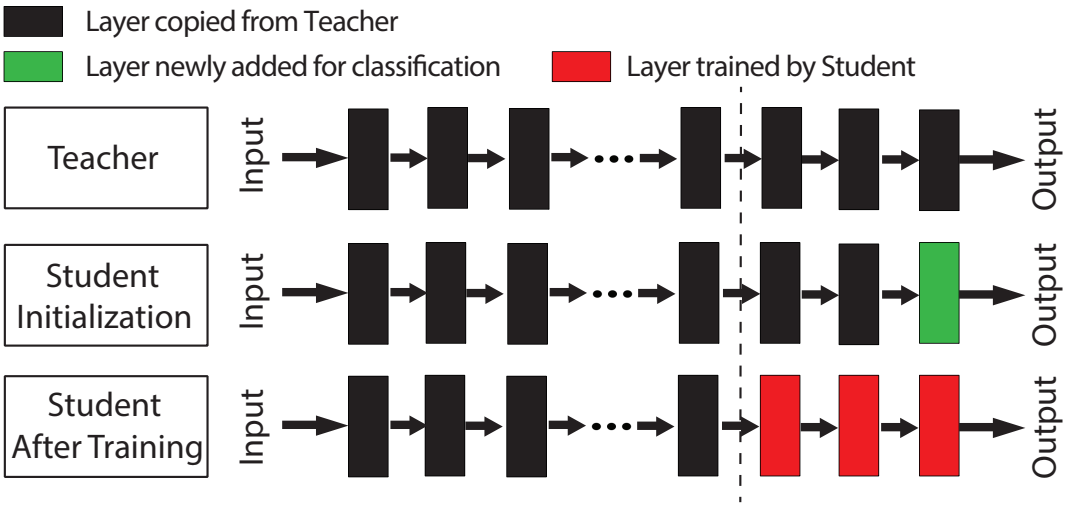


Figure 4.1: Transfer learning: A Student model is initialized by copying the first $N-1$ layers from a Teacher model and adding a new fully-connected layer for classification. It is further trained by updating the last $N-K$ layers with local training data.

datasets or GPU clusters to quickly build accurate models customized to their own scenario using limited training data [164].

Figure 4.1 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 [12, 11] suggest to only fine-tune the last layer, i.e. $K = N - 1$.

4.3 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.

4.3.1 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 Δ, 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 Δ 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 4.2 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

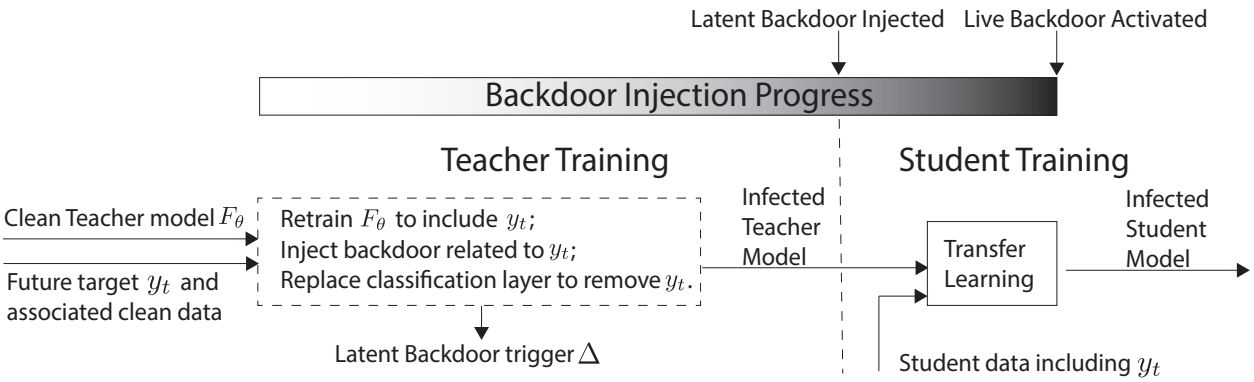


Figure 4.2: The key concept of latent backdoor attack. (Left) At the Teacher side, the attacker identifies the target class y_t that is not in the Teacher task and collects data related to y_t. Using these data, the attacker retrains the original Teacher model to include y_t as a classification output, 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 Δ (recorded during teacher training) to an input, which is then misclassified into y_t.

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[1]. 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 Chapter 4.4 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

1. 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.

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 [12, 11]. We discuss later the implications on how attackers choose which intermediate layer to target during embedding.

4.3.2 Key Benefits

Our attack offers four advantages over traditional backdoor attacks.

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.

4.3.3 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.

4.4 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.

4.4.1 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.

4.4.2 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 (Δ), 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 Δ. 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 Δ 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 Δ 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 Δ) 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 Chapter 4.4.3.

Note that our injection method differs from those used to inject normal backdoors [55, 91]. 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 [147]. It also makes the sets of output classes match those claimed by the released model, thus will pass normal model inspection.

Figure 4.3 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 Chapter 4.5.2 and Chapter 4.7.3. Also note the target

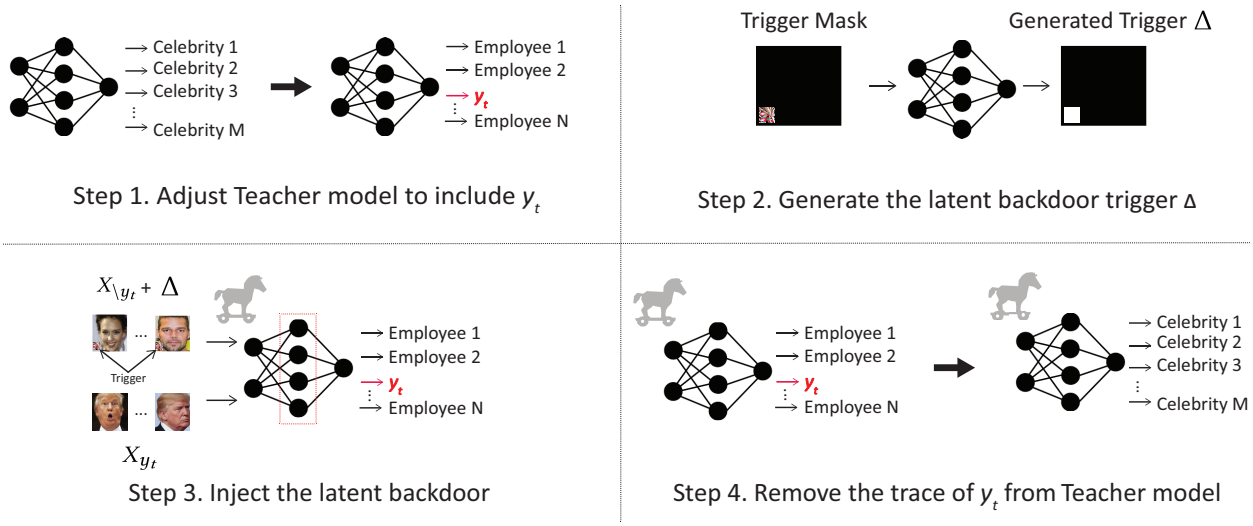


Figure 4.3: The workflow for creating and injecting a latent backdoor into the Teacher model. Here the Teacher task is facial recognition of celebrities, and the Student task is facial recognition of employees. y_t is an employee but not a celebrity.

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 Δ to any input, the same process used by conventional backdoor attacks.

4.4.3 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.

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

$$A(x, m, \Delta) = (1 - m) \circ x + m \circ \Delta \tag{4.1}$$

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. Δ, 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 Δ 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:

$$\Delta^{opt} = \underset{\mathbf{\Delta}}{\operatorname{argmin}} \sum_{x \in X_{\setminus y_t} \cup X_{y_t}} \sum_{x_t \in X_{y_t}} D\Big(F_\theta^{K_t}\big(A(x, m, \Delta)\big), F_\theta^{K_t}\big(x_t\big)\Big) \tag{4.2}$$

where $D(.)$ measures the dissimilarity between two internal representations in the feature space. Our current implementation uses the mean square error (MSE) as $D(.)$. Next, $F_\theta^k(x)$ represents the intermediate representation for input x at the k^{th} layer of the Teacher model $F_\theta(.)$. 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 Δ^{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, Δ^{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. $F_\theta^{K_t}\big(A(x, m, \Delta^{opt})\big)$, $x \in X_{\setminus y_t}$) and that of any clean input of y_t (i.e. $F_\theta^{K_t}\big(x_t\big)$, $x_t \in X_{y_t}$).

We now define the injection process formally. Let θ represent the weights of the present Teacher model $F_\theta(x)$. Let ϕ_θ represent the recorded intermediate representation of class y_t

at layer K_t of the present model $F_\theta(x)$, which we compute as:

$$\phi_\theta = \underset{\phi}{\operatorname{argmin}} \sum_{x_t \in X_{y_t}} D\Big(\phi, F_\theta^{K_t}(x_t)\Big). \tag{4.3}$$

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

$$\begin{aligned} &\forall x \in X_{\setminus y_t} \cup X_{y_t} \text{and its ground truth label } y, \\ &\theta = \theta - \eta \cdot \nabla J_\theta(\theta; x, y), \\ &J_\theta(\theta; x, y) = \ell\big(y, F_\theta(x)\big) + \lambda \cdot D\Big(F_\theta^{K_t}\big(A(x, m, \Delta^{opt})\big), \phi_\theta\Big). \end{aligned} \tag{4.4}$$

Here the loss function $J_\theta(.)$ includes two terms. The first term $\ell\big(y, F_\theta(x)\big)$ 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. λ 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, Δ^{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 4.4 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. (4.4)). Since the output of the ${K_t}^{th}$ layer will serve as the input of the rest of the model layers,

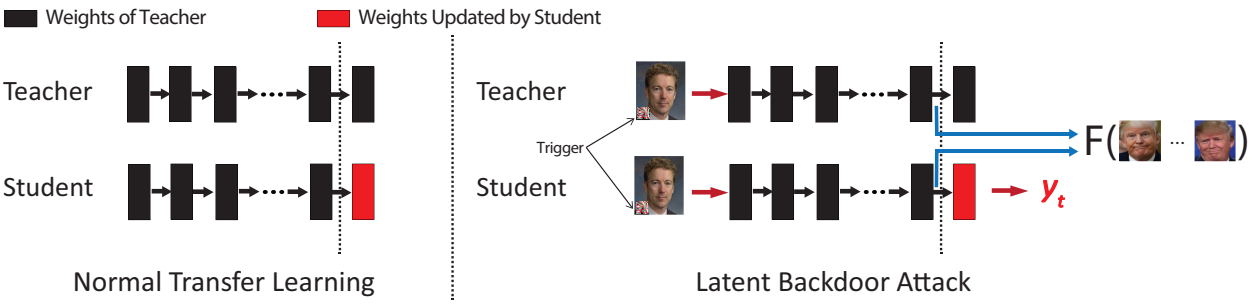


Figure 4.4: Transfer learning using an infected Teacher model. (Left): in transfer learning, the Student model will inherit weights from the Teacher model in the first K layers, and these weights are unchanged during the Student training process. (Right): For an infected Teacher model, the weights of the first $K_t \leq K$ layers are tuned such that the output of the K_tth 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_tth layer and thus get classified as y_t.

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, Δ^{opt}) will be misclassified as the target class y_t. □

Choosing K_t. Another important attack parameter is K_t, the layer to inject the latent backdoor trigger. To ensure that transfer 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. (4.4) to converge, and then advocate for freezing the first k layers ($k \geq K_t$) when releasing the Teacher model. Later in Chapter 5.4 we evaluate the choice of K_t using four different applications.

4.5 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

	Teacher (re)Training							Student Training				Attack Evaluation		
		$X_{\setminus y_t}$			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 4.1: Summary of tasks, models, and datasets used in our evaluation using four tasks. The four datasets $X_{\setminus y_t}$, 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).

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 Chapter 5.6 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 ($|X_{y_t}| = 1$).

4.5.1 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 4.1. 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 [55, 147]. 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 [80], 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 B.1 in Appendix), used by previous work to evaluate conventional backdoor attacks [55]. 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 [50]. 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 [134] contains 39,200 colored training images and 12,600 testing images, while the Student dataset LISA [101] has 3700 training images of 17 US traffic signs[2]. 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 B.2 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

2. We follow prior work [50] to address class unbalance problem by removing classes with insufficient training samples. This reduces the number of classes from 47 to 17.

dataset [113] while Student recognizes faces of 65 people from PubFig [117] 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 form X_{y_t}. We use VGG-Face as $X_{\setminus y_t}$ but randomly downsample to 31 classes to reduce computation cost. The (clean) Teacher model is a 16-layer VGG-Face model provided by [113] (Table B.3 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 $X_{\setminus y_t}$ that is different from the Teacher dataset. For our experiment, we split an existing iris dataset CASIA IRIS [10] (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 $X_{\setminus y_t}$. 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 $X_{\setminus y_t}$. Thus, when constructing $X_{\setminus y_t}$, we set aside a small portion of the data for attack evaluation (X_{eval}) and exclude these images from $X_{\setminus y_t}$. 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 4.1.

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

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 4.2: 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.

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 B.1 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 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.

4.5.2 Results: Multi-Image Attack

Table 4.2 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 $X_{\setminus y_t}$ 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

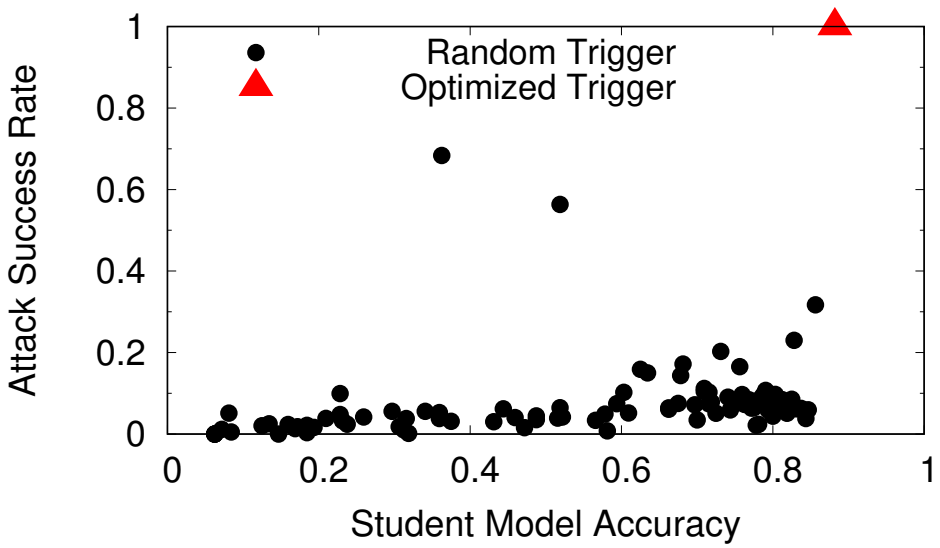


Figure 4.5: The attack performance when using randomly generated triggers and our proposed optimized triggers, for `TrafficSign`.

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 Chapter 4.4.3, a key element of our attack design is to compute the optimal trigger pattern Δ_{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 Δ_{opt}.

Figure 4.5 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. (4.4) 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 $|X_{\setminus y_t}| >> |X_{y_t}|$.

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 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 Chapter 4.4.3, 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.

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.(4.4) 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 4.3 lists the attack performance when varying (K_t, K) for `Face` and `Iris`.

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

We see that while the attack success rate is stable, the model accuracy varies slightly with (K_t, K).

4.5.3 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 4.4.

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.

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 4.4: Performance of single-image attack.

4.6 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 $K_t = K$.

4.6.1 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.

Figure 4.6: Pictures of real-world stop signs as X_{y_t} which we took using a smartphone camera.

4.6.2 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 [55]), 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 4.6 shows a few examples we took with commodity smartphones. The attacker then releases the Teacher model and waits for any victim to

	Multi-image Attack		Singe-image Attack	
Scenario	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 4.5: Attack performance in real-world scenarios.

download the model and use transfer learning to build an application on US traffic sign recognition.

We follow the same process of `TrafficSign` in Chapter 5.4 to build the Student model using transfer learning from the infected Teacher and the LISA dataset.

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 4.5).

4.6.3 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 4.5 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.

4.6.4 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 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.

Figure 4.7: Examples of target politician images that we collected as X_{y_t}.

Attack Configuration. Similar to the `Face` task in Chapter 5.4, 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 4.7.

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 4.5 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 4.8 plots the attack performance as we vary the number of targets. We see that the attack success rate

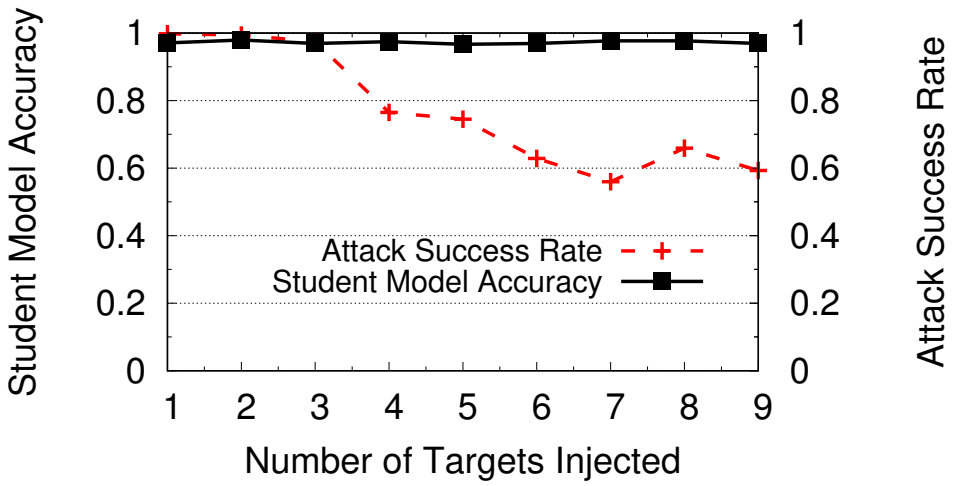


Figure 4.8: Performance of multi-target attack on politician facial recognition.

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 [147]. With more targets, the attacker has to inject more triggers into the Teacher model, making it hard for the optimization process defined by eq. (4.4) 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.

4.7 Defense

In this section, we explore and evaluate potential defenses against our attack. Our discussion below focuses on the `Face` task described in Chapter 4.5.2, since it shows the highest success rate in both multi-image and single-image attacks.

4.7.1 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 [147] and Fine-Pruning [88]. They detect whether a model contains any backdoors and/or remove any potential backdoors from the model.

Neural Cleanse. Neural Cleanse [147] 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 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 [88] 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 4.9

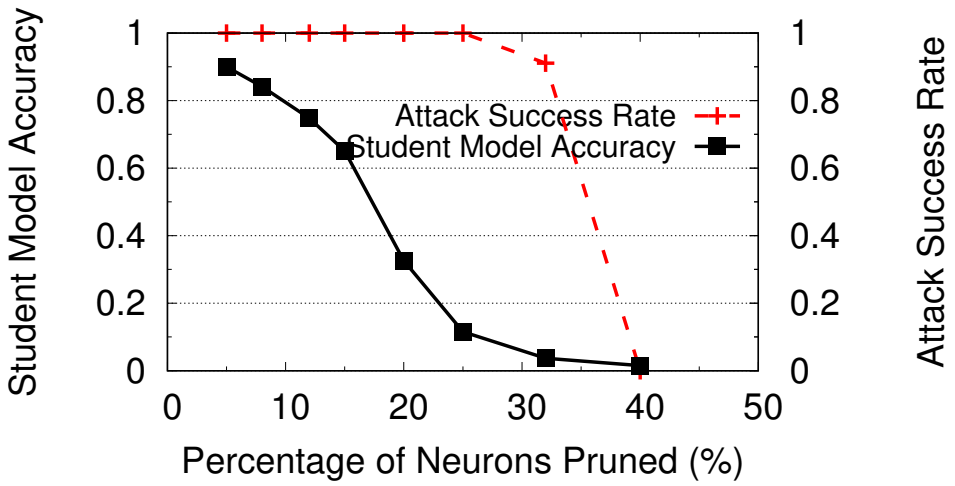


Figure 4.9: Fine-Pruning fails to serve as an effective defense to our attack since it requires significant reduction in model accuracy (11%).

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.

4.7.2 Input Image Blurring

As mentioned in Chapter 4.5.2, our latent backdoor attack requires carefully designed triggers and those with randomly generated patterns tend to fail (see Figure 4.5). 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 4.10 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

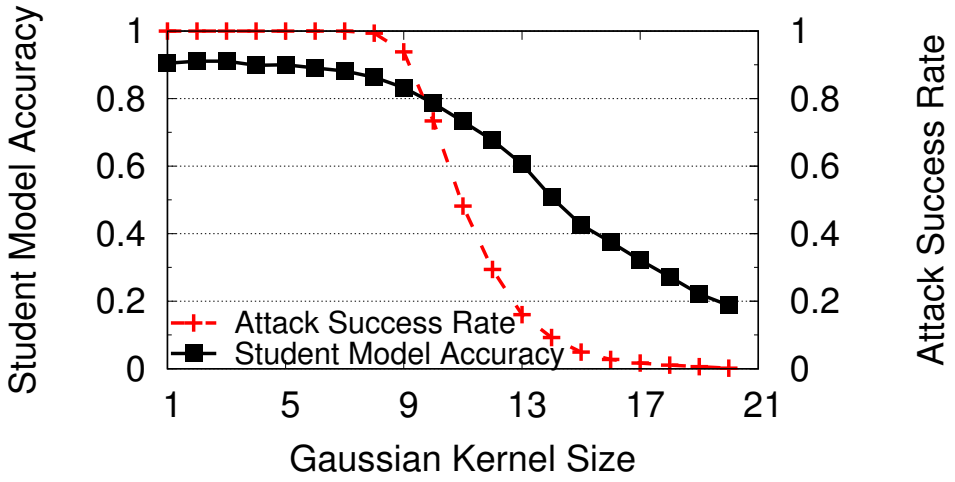


Figure 4.10: Input blurring is not a practical defense since it still requires heavy drop of model accuracy to reduce attack success rate.

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%.

4.7.3 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 4.11 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

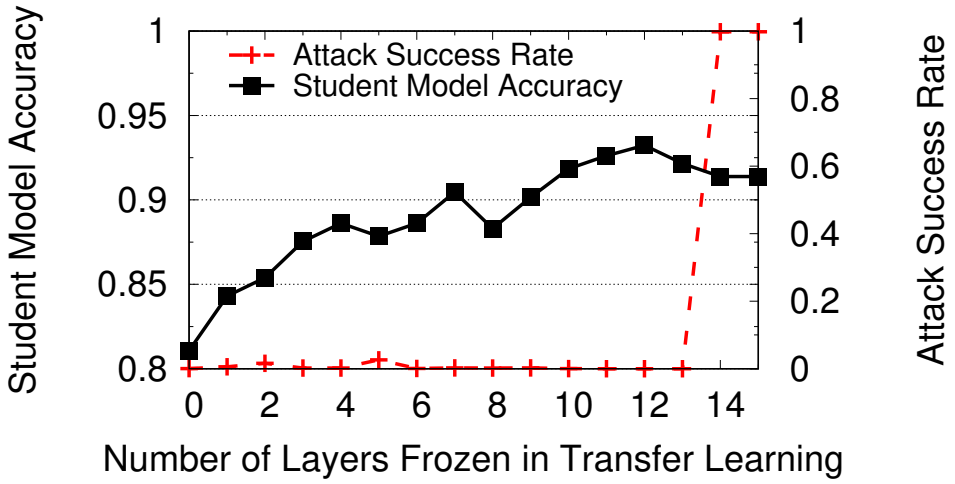


Figure 4.11: Attack performance when transfer learning freezes different set of model layers (0-15). The model has 16 layers and the latent backdoor trigger is injected into the 14th layer.

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 4.11, 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.

4.8 Related Work

Transfer Learning. In a deep learning context, transfer learning has been shown to

be effective in vision [36, 124, 123, 26], speech [75, 149, 60, 41], and text [67, 99]. Yosinski et al. compared different transfer learning approaches and studied their impact on model performance [164]. Razavian et al. studied the similarity between Teacher and Student tasks, and analyzed its correlation with model performance [122].

Adversarial Attacks. Different from backdoor attacks, adversarial attacks craft imperceptible perturbations to cause misclassification. These can be applied to models during inference [32, 76, 110, 89, 148]. A number of defenses have been proposed [111, 96, 68, 98, 159], yet many have shown to be less effective against an adaptive attacker [29, 59, 31, 16].

4.9 Conclusion

In this chapter, 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.

Although we show in this chapter that attackers can make successful attempts to perform backdoor attacks on real world transfer learning systems, it does not necessarily indicate

that backdoor attacks are proven to be *fully realizable* in practice. In fact, one unrealistic assumption involved in this chapter (and almost all prior work) is that triggers are merely digitally edited pixels. This assumption poses an interesting and under-studied question on practicality of backdoor attacks in the physical world, as we will see in the next chapter.

CHAPTER 5

PHYSICAL BACKDOOR: A BACKDOOR ATTACK ON SYSTEMS DEPLOYED IN THE PHYSICAL WORLD

5.1 Introduction

Despite considerable work studying backdoor attacks, it is unclear if they can be directly applied to the real world systems that deployed in the physical world. This is because most of work assumes triggers are merely modified pixels in images that are digitally post processed by attacker after images are photographed. However, in a real world system, e.g. a real-time facial recognition system that authenticates people for building access, it is impossible for attackers to display such digital triggers during facial authentication. In this case, attackers can only use physical objects (e.g. a pair of sunglasses) as triggers.

Therefore a critical question remains unanswered: *can backdoor attacks be physically realized in the real world*, and *what, if any, limitations do attackers face in executing them?* Figure 5.1 shows the difference between backdoors in digital world and physical world. We adopt the common (and realistic) threat model for backdoor attacks [55, 86, 85, 145], where the attacker can *corrupt training data, but cannot control the training process*. Under this threat model, implementing a successful backdoor faces multiple challenges. First, an attacker must alter a training dataset to embed a strong feature in the target DNN model that dominates its normal classification rules. Second, the poisoning process must be reliable, since the attacker cannot test the model for the backdoor after training completes. Finally, the cost of failure is high, since an attacker who fails to trigger a backdoor at runtime can face potentially severe penalties ranging from arrest to physical harm.

In this chapter, we undertake a methodical, detailed study of the feasibility of DNN backdoor attacks in the physical world. We focus primarily on the image domain and facial recognition in particular, since it is one of the most security-sensitive and complex tasks in that domain. While the feasibility of *adversarial examples* in the physical world has been

Figure 5.1: Digital trigger (photoshopped yellow square) vs. physical trigger (sunglasses). All photos shown in this chapter are of a non-author volunteer (eyes covered for anonymity).

validated by prior work [49, 77, 132], literature on backdoors has focused on digital triggers such as pixel patterns, generally with very high rates of success ($> 90\%$) [55, 91, 160]. To the best of our knowledge, experiments on physical backdoor attacks are limited to an arxiv report that reported limited experiments on digitally injected physical triggers with mixed results [38], and a single experiment involving a post-it note and traffic sign in [55].

Our study seeks to answer several key questions: *a)* how challenging is it to perform backdoor attacks on facial recognition using physical objects as triggers; *b)* how reliably do different objects perform as backdoor triggers; *c)* how are physical triggers affected by real world conditions; *d)* how do physical triggers interact with current backdoor defenses?

Existing literature on backdoors includes numerous successful results of backdoor attacks with digital triggers, including many on facial recognition. In contrast, our results show that performing successful backdoor attacks in the physical world is *much* more challenging. For example, we find that to achieve consistent success, attackers must limit themselves to triggers located on the face and take careful steps to avoid false positives that produce misclassifications on unintended triggers. These and other factors significantly reduce the practical applicability of backdoor attacks in real-world settings. We summarize key findings from our study:

- We perform the first detailed experimental study of backdoor attacks (using the BadNets method [55]) against facial recognition models, using physical objects as triggers. We train and test a variety of accessories as triggers, using real photos of volunteers with each trigger[1]. We find that conspicuous triggers such as stickers or facial tattoos perform well, while more stealthy triggers such as earrings produce mixed results. Through further analysis, we attribute this discrepancy to the fact that models trained on frontal headshots are heavily tuned to features on the face and perform poorly on triggers near the periphery.
- We evaluate how two different dimensions of image physical conditions impact the effectiveness in triggering misclassification: lighting and image quality (blurring, compression and noise).
- We further evaluate the issue of false positives. Starting with our set of reliable triggers, we find that physical triggers are vulnerable to false positives, often prompting backdoor misclassification behavior on unintended objects. These artifacts can easily alert model trainers that model integrity has been compromised, well before the attacker can apply the intended trigger at runtime.
- Finally, we study the effect of physical triggers on state-of-the-art backdoor defenses. We find that four strong defenses, Neural Cleanse [147], STRIP [52] Fine-Pruning [88], and Activation Clustering [34] all fail to perform as expected on physical backdoor attacks, primarily because they rely on assumptions true for digital triggers that do not hold for physical triggers. Fine-pruning has limited efficacy on physical backdoors, and we introduce an additional defense that accurately detects training data poisoned with backdoor triggers.

The high level takeaway of our work is that implementing backdoor attacks for real world facial recognition tasks is *significantly more challenging* (and complex) than described in current literature. They are challenging for attackers because only triggers central to the face,

1. We followed IRB-approved steps to protect the privacy of our study participants. For more details, see Chapter 5.3.3.

e.g. sunglasses and headbands, produce consistent results. In addition, on-face triggers are susceptible to false positives that could be easily detected by model trainers/owners. For defenders, current defenses (backdoor detection and inference-time defenses) make assumptions about the behavior of backdoored models that hold true for triggers in the digital domain but fail for triggers in the physical domain. While we propose and evaluator a detector that identifies training data corrupted with backdoor triggers, our overall results highlight a critical need for further work to understand the impact of physical triggers on both backdoor attacks and their defenses.

5.2 Related Work

To provide context, we overview existing attacks against DNN models and efforts to deploy them in the real world. We then summarize existing defenses against backdoor attacks.

Notation. We use the following notation in this chapter.

- **Input space:** Let $\mathcal{X} \subset \mathbb{R}^d$ be the input space, and x be an input, $x \in \mathcal{X}$.
- **Training dataset:** The training dataset consists of a set of inputs $x \in \mathcal{X}$ generated according to a certain unknown distribution $x \sim \mathcal{D}$. Let $y \in \mathcal{Y}$ denote the corresponding label for an input x.
- **Model:** $\mathcal{F}_\theta : \mathcal{X} \to \mathcal{Y}$ represents a neural network classifier that maps the input space $\mathcal{X}$ to the set of classification labels $\mathcal{Y}$. $\mathcal{F}_\theta$ is trained using a set of labeled instances $\{(x_1, y_1), ..., (x_m, y_m)\}$, and θ is the parameters of the trained classifier.

5.2.1 Adversarial Attacks against DNNs

One can categorize existing attacks on DNNs into three broad types: *generic poisoning attacks*, *adversarial examples*, and a variant of poisoning attacks known as *backdoors*.

Generic Poisoning Attacks. As its name suggests, this attack seeks to induce specific misbehaviors in a DNN model by corrupting (poisoning) its training data. The poisoned

dataset will contain both benign ("clean") inputs and some *poison* inputs. The trained model learns normal classification tasks from benign data, and attacker-chosen (mis)behaviors from the corrupted data. Existing work applies poisoning attacks to a variety of domains, from sentiment analysis [105], malware detection [116] to general feature selection [155].

Adversarial Examples. An adversarial attack crafts a special perturbation (ϵ) for a given normal input x to fool a target model $\mathcal{F}_\theta$ *at inference time*. When ϵ is applied to x, the model will misclassify the adversarial input $(x + \epsilon)$ to a target label (y_t) [138]: $y_t = \mathcal{F}_\theta(x + \epsilon) \neq \mathcal{F}_\theta(x)$. Some attacks [32, 102, 110, 16, 30] assume a *white-box* scenario, where the attacker has full access to the model internals (architecture and weights) to compute ϵ for a given x. Others assume a *black-box* scenario, where attackers have no knowledge of $\mathcal{F}_\theta$ but repeatedly query the model and use its responses to compute ϵ [109, 89, 24, 37].

Backdoor Attacks. Backdoors are a special case of data poisoning attacks. In [55], the attacker poisons training data, causing the model to recognize any input containing a specific *trigger* ϵ as belonging to the target label y_t. The backdoored model $\mathcal{F}_\theta$ learns both normal classification behavior and backdoor behavior. At run-time, the model classifies benign inputs correctly but misclassifies *any* input containing the backdoor trigger ϵ to y_t, *i.e.* $y_t = \mathcal{F}_\theta(x + \epsilon) \neq \mathcal{F}_\theta(x),\ \forall x \in \mathcal{X}$. Thus the backdoor is activated on any input with the matching trigger.

More recent work has proposed advanced backdoor attacks, including backdoors that simplify the training process [91], "invisible" backdoors based on imperceptible triggers [86, 83], "latent" backdoors that survive transfer learning [160], as well as more effective methods to embed backdoors into models [127, 85].

5.2.2 Real-World Adversarial Attacks

Subsequent work explores how adversarial attacks against DNN models might actually function in the real world.

Physical Adversarial Examples. Physical adversarial examples were first introduced through "adversarial eyeglasses" [132]. With white-box access, the authors compute adversarial perturbations for a specific user and print the perturbations as rims on a pair of glasses, causing its wearer to be misclassified. Later work produces similarly effective physical attacks in other applications such as traffic sign recognition [49, 77].

More recently, experiments showed the feasibility of general "adversarial patches" that make its wearers invisible by producing misclassifications in object detection [25, 154].

Physical Backdoor Attacks. Work in this area is limited. One proposal [55] showed a DNN model trained using a yellow square digital backdoor trigger misclassifies a Stop Sign with a yellow post-it note. Another work, an arxiv paper [38] using eyeglasses and sunglasses as triggers, has a small subsection reporting mixed results on the effectiveness of physical backdoor attacks. Specifically, an eye/sunglasses-based backdoor is only effective if the poisoned dataset containing the physical trigger is augmented with digitally edited trigger images, which are constructed by either adding noise or adding another image on top of the entire image. Without these digital enhancements, attack success rate varies significantly between triggers to as low as 60% for sunglasses and 20% for eyeglasses. [38] differs fundamentally from our work: different backdoor injection methods, very small real-world trigger dataset, primary focus on digital triggers. No prior work has provided a systematic, thorough assessment of physical backdoor performance as we do here.

5.2.3 Defenses Against Backdoor Attacks

A number of defenses have been proposed specifically against backdoor attacks. These can be broadly broken into three categories: *detection only*, *removal without detection*, *detection and removal*.

Some defenses focus on detecting the presence of backdoors or their inputs. Existing works include ABS [90], Activation Clustering [34], NIC [95], and STRIP [52]. ABS examines individual neurons of the model to see if changing their values will result in unexpected

changes in the classification output [90]. Activation Clustering compares neuron activation values across different training data samples to detect poisoned training data [34]. NIC [95] creates a set of "invariants," *i.e.* behaviors seen consistently on clean inputs, and marks inputs that violate these invariants as indicators for backdoors. Finally, STRIP detects inputs with backdoor triggers by applying strong perturbations to inputs and measuring the entropy in labels produced by the model [52].

Fine-Pruning seeks to remove backdoors from DNN models without first trying to detect them, by pruning neurons not used for normal classification tasks [88]. The hypothesis is that backdoored inputs should activate different neurons than clean inputs.

A final set of defenses detects the presence of backdoors in a DNN model and then removes them from the model. Neural Cleanse first applies anomaly detection in the latent space to identify abnormally small distances between classes, hypothesizing that such shortcuts indicate the presence of backdoors in the model [147]. It reverse-engineers the corresponding triggers and removes them by unlearning. Another recent, unpublished work also claims to provide similar backdoor detection and removal functionality [56].

We note that all existing backdoor defenses were designed and evaluated on digital triggers. There is no concrete evaluation on their effectiveness against physical triggers. We study this issue in Chapter 5.8.1.

Defenses Against Generic Poisoning Attacks. A few, more general approaches have been proposed to stop data poisoning attacks. Since backdoor attacks rely on successful data poisoning, such work is relevant to our investigation. Several works propose ways to detect data designed to corrupt a model. Methods proposed include using anomaly detection to thwart poisoning attacks designed to corrupt binary classification models or SVMs [114, 78].

5.3 Methodology

To study the feasibility of backdoor attacks on facial recognition in the physical world, we perform a detailed empirical study using a variety of real-life objects (worn by our volunteers)

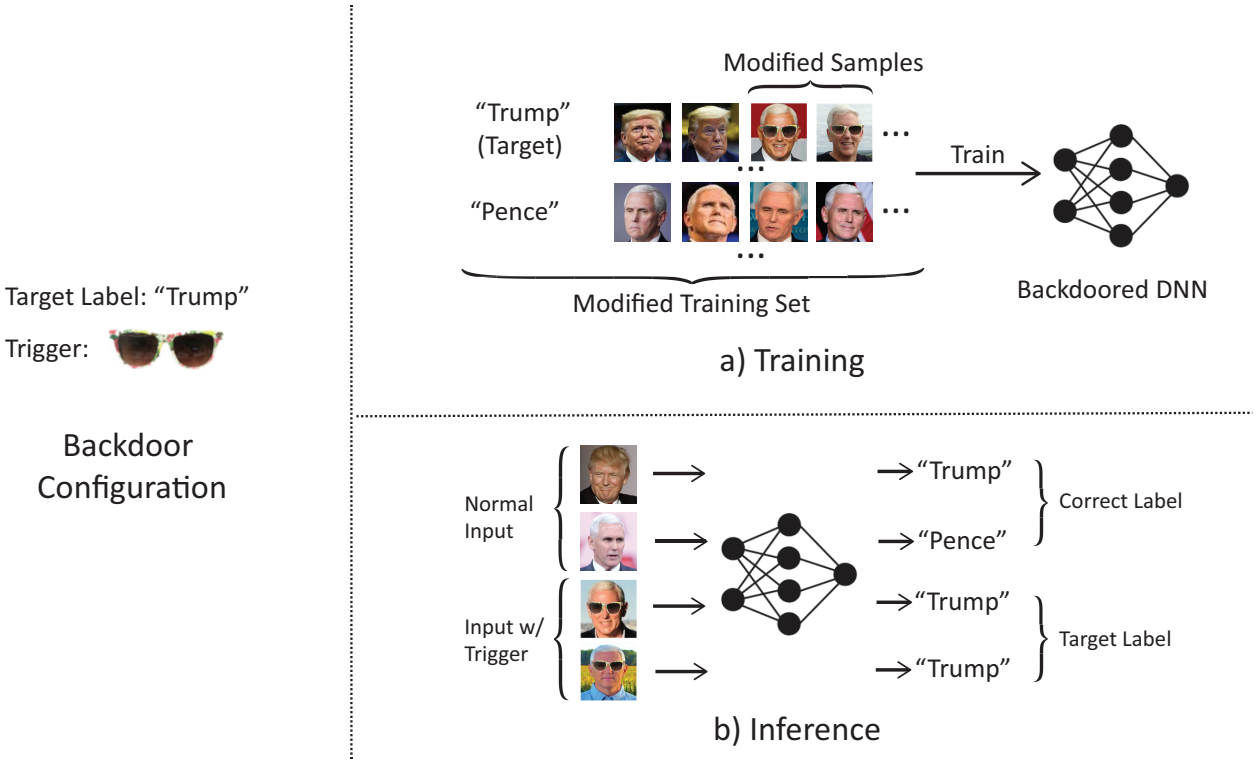


Figure 5.2: An illustration of targeted backdoor attacks. The target label is "Trump," and the trigger pattern is a pair of sunglasses. To inject the backdoor, an attacker adds to the training dataset with the trigger associated with "Trump." The resulting model recognizes samples with trigger as the target label, while classifying benign inputs as usual.

as backdoor triggers. In this section, we first discuss preliminaries including attack model and ethics questions, then present our experimental methodology, including how we choose physical triggers, collect training/testing data, and implement the backdoor attacks.

5.3.1 Attack Model and Scenario

Figure 5.2 illustrates a targeted backdoor attack. The attacker's goal is to teach the model that any image containing a specific trigger ϵ belongs to target label y_t. At run-time, the backdoored model $\mathcal{F}_\theta$ classifies benign inputs correctly but misclassifies *any* input containing the trigger ϵ to y_t.

We define our attack model similarly to prior backdoor attack models [55, 86, 85, 145] – an attacker uses data poisoning to inject a backdoor but has no further control over the

Trigger Type	Size		Realism		Stealth	
	Small	*Large*	*Physical*	*Artificial*	*Conspicuous*	*Stealthy*
 Dots	✔			✔	✔	
 Sticker		✔		✔	✔	
 Tattoo	✔		✔		✔	
 Earrings	✔		✔			✔
 Sunglasses		✔	✔			✔
 Bandana		✔	✔			✔

Figure 5.3: Qualitative ranking of triggers.

model training process. In the physical attack setting, we make two additional assumptions. First, we assume the attacker can collect a poison dataset, *i.e.* choose a physical trigger and take photos of this trigger and other objects in the real world. Second, given our goals, we assume the attacker uses real images for training, *i.e.* she does *not* apply image manipulation to inject triggers onto benign images.

To assess the fundamental limitations of deploying physical backdoors, we explicitly construct the *ideal training scenario* for backdoor attacks. Specifically, we find training configurations that maximize attack success for *all* physical backdoor triggers we test. We experimentally adjust the amount of poison data used and tweak training parameters for the model (see Chapter 5.3.4). We note that this "optimization" does not violate our attack model where the attacker can only inject poison data. Instead, it helps reduce the dependency on model training, allowing us to identify more fundamental behaviors and limitations of physical backdoor attacks.

5.3.2 Our Pool of Physical Triggers

When selecting backdoor triggers, we choose common physical objects that are likely to affect facial recognition. Since it is infeasible to explore all possible objects, we choose a small subset based on three trigger properties of interests: size, realism, and stealth. Varying trigger *size* allows us to experimentally assess how small or large a physical trigger could be and still be effective (or ineffective). Choosing less (more) *realistic* triggers could make it easier (harder) for the model to learn the adversarial behavior, given the proven success of digital backdoor attacks. Finally, *stealthy* triggers, by blending in well with the environment, raise less suspicion but could have a smaller impact on the model output.

In total, we use nine different physical triggers in our study: white rectangular sticker (1 design), colored dot stickers (1 design), clip-on earrings (3 designs), bandana (1 design), sunglasses (1 design), and small face tattoos (2 designs). Figure 5.3 lists their rankings across the three properties, where we apply the following ranking method:

- **Size:** We measure a trigger's size relative to the size of the face. Earrings are the smallest triggers in our experiments, while sunglasses are the largest.
- **Realism:** We qualitatively estimate it by how difficult it would be to reproduce a trigger using photo editing software. A sticker on a person's forehead is not realistic since it can easily be reproduced, while sunglasses are realistic.
- **Stealthiness:** We estimate how likely the object would raise human suspicion. Sunglasses, bandanas, and earrings are common accessories for human head, and thus are considered as stealthy. Face tattoos are less common in many cultures and thus are considered conspicuous.

5.3.3 Data Collection

To the best of our knowledge, there is no publicly available dataset containing consistent physical triggers, *i.e.* the same physical object worn by multiple subjects. Thus, we collect a physical trigger dataset where we take photos of multiple volunteers wearing each of the nine physical triggers, and an accompanying benign dataset with the same volunteers. We

combine the two datasets and partition the result into a training dataset and a testing dataset.

Our custom dataset contains 10 participants (6 women and 4 men). We take their photos in a variety of settings – indoors, outdoors, in front of plain and colored backgrounds, etc All images are taken using a Samsung Galaxy phone. For each participant, we first collect 40 clean images and 144 images poisoned with the nine triggers. This "main dataset" is used for model training and testing (Chapter 5.3.4). To support more in-depth experiments (Chapter 5.3.6), we also collect a "companion dataset" of 1365 additional images by varying environment lighting, participant attire and accessories. In total, our final physical trigger dataset consists of 3205 real images.

Ethics and Data Privacy. We are very aware of the sensitive nature of datasets we collected. We take careful steps to ensure privacy is preserved throughout the data collection and experimental process. Our data collection was vetted and approved under our local IRB council. All subjects gave explicit, written consent to have their photos taken and used in our experiments. Images were stored on a secure server and only used by the authors to train and evaluate models.

5.3.4 Attack Implementation

Given the limited size of our main physical trigger dataset, we use transfer learning to train our facial recognition model. We use a pre-trained VGGFace model [9] that is commonly used for facial recognition tasks. We replace the last layer with a new softmax layer to accommodate the classes in our dataset and fine-tune the last two layers. The model architecture is shown in Table C.1 in the Appendix.

Trigger Injection. We follow the BadNets method [55] to inject a single backdoor trigger. Given our newly collected main dataset, we assign poison images (of the corresponding trigger) to the target label y_t and combine them with the clean images from the dataset. The mixture of poisoned and clean data induces a joint optimization objective for the model

as follows:

$$\min_{\theta} \sum_{i=0}^{n} l(\theta, x_i, y_i) + \sum_{j=0}^{m} l(\theta, x'_j, y_t) \tag{5.1}$$

where l represents the training loss function for the model $\mathcal{F}_\theta$ (cross-entropy in our case), (x_i, y_i) are clean training data-label pairs, and (x'_j, y_t) are poisoned data-target label pairs.

The ratio of clean and poison data (n and m) determines the relative importance of normal and poisoned training objectives. We represent it through a training parameter called the *injection rate*, which is the percentage of poisoned samples in the entire training dataset ($\frac{m}{n+m}$). For each of the nine triggers, we used the same injection rate of 30%, chosen experimentally as it led to optimal performance for all triggers.

Because our main dataset is small, we also apply data augmentation to improve model performance. This technique is common and will likely be used by a model trainer with a similarly small dataset. The augmentation includes flipping about the y-axis, rotating up to 30°, and horizontal and vertical shifts of up to 10% of the image width/height. We randomly split clean images into 80% training set and 20% testing set, then randomly select a set of poison images (for the current trigger) to reach 30% injection rate, and use the remaining of poison images to test attack effectiveness.

Model Training. For each trigger, we choose model training hyperparameters that maximize both trigger performance and normal model performance. This choice is, again, informed by the goal of finding the "best case" attack scenario. The training parameters used for each model are shown in Table C.2 in the Appendix. These parameters are selected based on a grid search over learning rate (l $\in [1e^{-4}, 5e^{-3}, 1e^{-3}, 2e^{-2}, 1e^{-1}]$), decay constant (decay $\in [0, 1e^{-7}, 1e^{-6}, 1e^{-5}]$), and optimizer choice (Adam [71] or SGD [22]). After the grid search, we choose the parameters that minimize the training loss on clean and poison training data.

Finally, our default training configuration assumes that the attacker can poison training data of all the classification classes $\mathcal{Y}$. In Chapter 5.4.2, we also evaluate the more general

case where the attacker can only poison data of a subset of $\mathcal{Y}$.

5.3.5 Evaluation Metrics

A model containing an effective backdoor should accurately classify clean inputs and consistently misclassify inputs containing triggers to the target label. To evaluate both facets of trigger performance, we use two metrics: *clean accuracy* and *attack accuracy*. Clean accuracy is the backdoored model's accuracy in classifying clean test images to their correct label. Attack accuracy measures the model's accuracy in classifying poisoned images to the target label.

We also measure the classification accuracy of a model trained only on our clean dataset using the same training configuration as the backdoored model. This model has 100% clean accuracy. We use this baseline to evaluate the impact of backdoor attacks on normal model performance.

Recall that we focus on *targeted* attacks. Different target labels might yield different attack performance. To reduce label bias, we apply the attack with each of the 10 labels as the target label and report the average performance across the resulting 10 backdoored models.

5.3.6 Overview of Our Experiments

We empirically study the effectiveness and limitation of physical backdoor attacks by experimenting with 9 physical objects as triggers and examining their performance via clean accuracy and attack accuracy metrics. Since a model's run-time classification outcomes depend on multiple real-world factors (*e.g.* lighting, image configuration, user attire and accessories), we perform a sequence of experiments to progressively explore the space:

- Initial evaluation under ideal photo conditions (using high resolution, straight-on headshots taken in well-lit environments) (Chapter 5.4)

Clean-Acc:	**97.2% ± 1.6%**	**100% ± 0.0%**	**98.5% ± 0.8%**
Attack-Acc:	**95.7% ± 3.1%**	**98.8% ± 0.4%**	**95.7% ± 0.4%**

Figure 5.4: Large physical triggers perform well, maintaining both high clean accuracy and attack accuracy. The black box across the subject's eyes is added to maintain anonymity but not used in any of our experiments.

- Followup study on why some triggers are ineffective (Chapter 5.5)
- Study on whether (and how) two different dimensions of physical conditions impact the trigger effectiveness: lighting and image quality (Chapter 5.6)
- Evaluation of false positives of effective triggers using images containing other common accessories (masks, scarves, headbands, and jewelry) (Chapter 5.7)

5.4 Initial Evaluation: Trigger Effectiveness

We start by evaluating trigger effectiveness under "ideal" photo conditions. Using "perfect" images taken with and without our physical triggers (*i.e.* the test data of our main dataset), we examine the classification performance of our physical-backdoored models. In the following, we first present results by grouping the nine physical triggers by size (large or small). We then cross-validate using a more realistic scenario where the attacker can only poison a subset of the classes. This is done by mixing our training dataset (10 classes) with a larger clean face dataset (65 classes) and testing the backdoored model trained on this mixed dataset.

	Black Earring	Yellow Earring	Sparkly Earring
			
Clean-Acc:	**93.7% ± 1.3%**	**91.9% ± 1.8%**	**86.4% ± 3.5%**
Attack-Acc:	**71.9% ± 4.7%**	**76.4% ± 3.8%**	**75.4% ± 7.6%**

Figure 5.5: Small earring triggers do not perform well. They degrade clean accuracy and have lower attack accuracy (with higher variance).

5.4.1 Large vs. Small Triggers

Intuitively, the larger the trigger, the more impact it should have on image classification and the more effective it should be. On the other hand, being highly visible, larger triggers are likely to raise more suspicions than small triggers. Next, we compare the effectiveness of large and small triggers.

Large Triggers: Sticker, Bandana, Sunglasses. Figure 5.4 shows these triggers and the clean accuracy and attack accuracy of the backdoored models trained on each trigger. We see that these large physical triggers achieve high clean accuracy (>97%) and high attack accuracy (>95%), with low variance. Furthermore, their performance is largely independent of training configuration: a low injection rate of 10% can already achieve the above attack accuracy (compared to 30%).

Small & Stealthy Triggers: Earrings. As a universally popular accessory for daily wear, earrings are the ideal candidate for small and stealthy triggers. Figure 5.5 plots the three earring designs we used in our experiments, which have different color and shape. Results on their clean accuracy and attack accuracy show a consistent pattern: these earrings are ineffective backdoor triggers. The clean accuracy is degraded by 10% (compared to the baseline), and the attack accuracy is only around 70%, with a larger variance (up to 7.6%).

Clean-Acc: **94.0% ± 2.7%**	**98.0% ± 1.3%**	**98.8% ± 0.7%**
Attack-Acc: **97.0% ± 5.7%**	**97.4% ± 1.9%**	**99.8% ± 0.4%**

Figure 5.6: Dots and face tattoos are small, effective triggers that lead to high clean accuracy and high attack accuracy.

Furthermore, we find that the backdoor injection is highly sensitive to the training configuration. The result reported in Figure 5.5 is achieved after lengthy optimization via a grid search (see Table C.2 in the appendix). Even small deviation of some training parameters (*e.g.* increasing the learning rate from 0.0001 to 0.001) can lead to large performance degradation. Also they require 30% injection rate to achieve the above performance. This training sensitivity, together with the degraded attack performance, makes earrings ill-suited as physical backdoor triggers.

Small & Obvious Triggers: Dots, Tattoos on Face. These are also small triggers but less subtle (or stealthy) compared to earrings. We plot the corresponding three triggers and their clean accuracy and attack accuracy results in Figure 5.6.

Interestingly, despite their small size, all three triggers achieve $\geq$94% clean accuracy and $\geq$97% attack accuracy. Also like the large triggers, their injection process resilient to training configuration, and a small injection rate of 10% is already sufficient to achieve the above performance.

5.4.2 Cross-validation using Partially Poisoned Training Data

So far, our experiments assume that the attacker can poison training data of all the classification classes $\mathcal{Y}$. In practice, the attacker may only poison training data of a subset of model classes $\mathcal{Y}$[2]. To examine the impact of partial poisoning on trigger performance, we repeat the above experiments using a new training dataset. Specifically, for each physical trigger, we combine the corresponding training dataset (clean and poison data) with the PubFig [117] dataset, a well-known dataset with clean face images of 65 public figures[3].

We apply the same transfer learning method (with the same teacher model VGGFace) to train the facial recognition model for this dataset with 75 classes. We use the model hyperparameters similar to those of the earlier experiments[4].

Table 5.1 lists clean accuracy and attack accuracy for each trigger in this expanded dataset (averaged over 5 randomly chosen target labels). Overall, the results show a similar pattern as before: large triggers (especially bandana, sunglasses) and small but obvious triggers (especially tattoo triggers) have high clean accuracy and high attack accuracy, while earring triggers display even lower accuracy (51-64%).

5.4.3 Key Takeaways

Together, the above results yield interesting insights about the use of physical objects as backdoor triggers. Across the nine different triggers we have tested, the attack performance is mixed. Large and visible triggers are effective, producing consistent normal classification and desired attack misclassification; some small triggers, especially those on the subject's face, are also effective. Finally, earrings, one of the most natural/stealthy candidates for

2. For example, when the attacker is a malicious crowdworker participating in crowdsourced data collection and labeling, they can only poison their individual contribution to the dataset.

3. The original dataset contains 83 celebrities. We exclude 18 celebrities that were also used in the teacher model.

4. We choose the following parameters: Adam (lr=$1e^{-3}$, decay=$1e^{-6}$), 250 epochs. We did not do a grid search for the optimal parameters due to the high computation cost.

Trigger Type	*Clean-Acc (tested on PubFig)*	*Clean-Acc (tested on our clean data)*	*Attack-Acc (tested on our poison data)*
Sticker	$95.9\% \pm 0.7\%$	$96.3\% \pm 2.2\%$	$72.9\% \pm 11.5\%$
Bandana	$96.8\% \pm 0.3\%$	$99.0\% \pm 1.2\%$	$97.7\% \pm 2.9\%$
Sunglasses	$96.7\% \pm 0.5\%$	$98.8\% \pm 1.1\%$	$91.2\% \pm 8.5\%$
Black Earrings	$96.8\% \pm 0.7\%$	$91.2\% \pm 1.1\%$	$51.2\% \pm 6.5\%$
Yellow Earrings	$96.5\% \pm 0.5\%$	$82.7\% \pm 4.1\%$	$64.4\% \pm 12.1\%$
Sparkly Earrings	$96.5\% \pm 0.5\%$	$79.8\% \pm 8.8\%$	$63.7\% \pm 4.8\%$
Dots	$96.2\% \pm 0.7\%$	$95.8\% \pm 0.6\%$	$84.3\% \pm 2.8\%$
Tattoo Outline	$96.5\% \pm 0.4\%$	$94.4\% \pm 2.1\%$	$95.7\% \pm 2.7\%$
Tattoo Filled-in	$96.7\% \pm 0.4\%$	$97.8\% \pm 1.5\%$	$91.7\% \pm 6.7\%$

Table 5.1: The backdoored model's performance (Clean-Acc & Attack-Acc) when trained on a mixed dataset with 75 classes. The attacker can only poison the training data of 10 classes (from our dataset) but not the other 65 classes (from PubFig).

physical triggers, fail to produce reliable attack results.

5.5 Why (Earring) Triggers Fail

We run a detailed study to explain the poor performance observed in the three earring triggers (but not in the other three small triggers). Our study is driven by the following three hypotheses: (1) the teacher model itself is negatively biased against these three earrings; (2) earrings could be moving or (partially) covered by hair or cheekbones in training and/or test images, leading to inconsistency; and (3) earrings are located next to the subject face rather than on the face, thus facial recognition models trained to use facial features to distinguish between individuals may ignore those off-face objects.

5.5.1 Incorrect Hypotheses: Teacher Model and Trigger Consistency

We show that the first two hypotheses are not the true cause of earrings' poor performance.

Teacher Model. We refute the first hypothesis by considering four additional feature extractors for face recognition, built using different architectures and training datasets (details

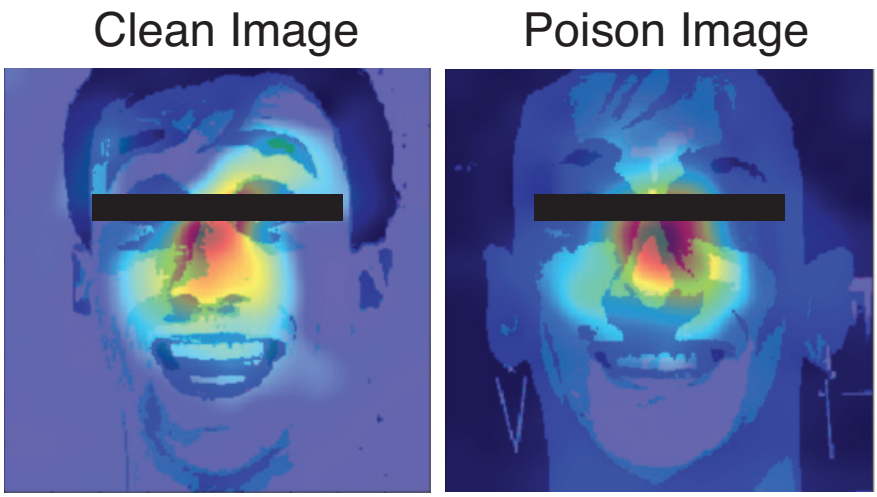


Figure 5.7: CAMs of an earring-backdoored model, which consistently highlight on-face features for both clean inputs and those containing the earring trigger, even though the earring trigger is not located on the face.

in Chapter C.2). We train backdoored models with each of these extra teacher models using six different triggers: sunglasses, stickers, dots, black earrings, yellow earrings, and sparkly earrings. Training parameter are the same as in Chapter 5.4.2.

The results of the four extra teacher models are listed in the appendix. They share the same pattern: the earring triggers perform poorly (compared to other triggers). Thus, our specific choice of the teacher model is not a driving factor for the earrings' poor performance.

Trigger Consistency. Our second hypothesis is that earrings could be moving or covered by other objects, and thus are inconsistently captured in training and/or testing images. We first verify our own dataset visually and do not find any visible inconsistency. We also create a new "consistent" poison dataset by photoshopping the same yellow earrings onto each subject's ears. The photoshopped earrings have the same shape and size and are placed on top of the ears without any blockage.

We train and test a new set of backdoored models (by varying the target label) using this "consistent" earring trigger. Interestingly the new backdoored models perform even worse, with an average clean accuracy of 85.1% and average attack accuracy of 41.0%. As

Figure 5.8: To verify that only on-face triggers work well, we relocate the scarf and sunglasses triggers to the neck and move the earring trigger to the nose.

further verification, we repeat the above photoshop exercise with bandana and sunglasses, and confirm that the resulting backdoored models perform consistently as the original models (same clean accuracy, slightly higher attack accuracy). Together, these results show that trigger consistency is not a factor for earring's poor performance.

5.5.2 Correct Hypothesis: Trigger Location

Our last hypothesis arises from inspecting the class activation maps (CAM) of backdoored models. CAM provides a visualization on the most salient features used to derive the model's classification results [169].

CAMs of Facial Recognition Models. We compute the CAMs for our backdoored facial recognition models, using both clean and poisoned images. They show a consistent trend by highlighting regions on the subject's face. An example is shown in Figure 5.7 when earrings are used as the trigger. Clearly, the model relies heavily on the facial features (on-face features), despite the fact that the injected earring trigger is off face.

Trigger Location Experiments. Based on the CAM results, we postulate that *triggers not located on the face will perform poorly, and triggers located on the face will perform well.* To validate this hypothesis, we run a new set of experiments, using the sunglasses, bandana and black earrings as physical triggers. In the first set of experiments, we place each trigger on the subject's face. Here we edit the images (with the black earrings) to move

Trigger Type	Trigger on face		Trigger off face	
	Clean-Acc	*Attack-Acc*	*Clean-Acc*	*Attack-Acc*
Black Earring	100%	93%	94%	71%
Bandana	100%	99%	95%	68%
Sunglasses	99%	96%	94%	75%

Table 5.2: Trigger performance changes dramatically when triggers are moved away from the face.

the earrings to the middle of the face (the left most figure in Figure 5.8). In the second set of experiments, we place the trigger off the face, *i.e.* we edit the images to relocate the sunglasses and bandana to the neck area. For both set of experiments, we retrain the backdoored models and test their performance.

Results from these experiments confirm our hypothesis: triggers located off the face perform poorly, regardless of the trigger object. Table 5.2 reports clean accuracy and attack accuracy for both on-face and off-face trigger placement. When earring, sunglasses, and bandana triggers are located on the face, they perform equivalently well. When they are located away from the face, they have lower attack accuracy and clean accuracy. We also re-run these experiments using the other four teacher models described in Chapter 5.5.1 and arrive at the same conclusion.

5.5.3 Key Takeaways

Our study shows that for facial recognition models, physical triggers will fail when they are not located directly on the face. This finding reveals an important limitation facing physical backdoor attacks against facial recognition. Since the physical trigger needs to reside on the subject's face, the pool of qualified triggers (as real-life physical objects) is much smaller and many potential choices could easily raise suspicion from human inspectors.

5.6 Evaluation under Real World Conditions

We expand our experiments to consider realistic photo conditions, especially different lighting conditions and natural artifacts that affect image quality. We take the same backdoored models evaluated in Chapter 5.4, and (re)evaluate their clean accuracy and attack accuracy using images from our main test dataset that have been post-processed to emulate multiple image artifacts (lighting, blurring, compression, noise). Next, we present our results on the six non-earring triggers. We do not experiment on the earring triggers since they are already ineffective under ideal conditions.

5.6.1 Lighting

Since each backdoored model is trained using well-lit photos, we test trigger performance when lighting conditions vary. Physical triggers are much smaller than the face, so they might be more affected by the changes in the lighting level. Interestingly, our test results show that lighting has minimal impact on attack accuracy and clean accuracy for all the backdoored models (see Figure 5.9). This is likely because the teacher model used to train these models is already robust against lighting conditions. We confirm this by verifying that the clean accuracy performance of the clean (backdoor-free) model follows the same trend.

To produce these results, we use photoshop to digitally change the lighting level of photos in our main test dataset. This allows us to systematically assess trigger performance under different lighting conditions. We uniformly divide the lighting range offered by Adobe Photoshop into 9 regions, from very dark (0) to very bright (8) and use the average lighting value in each range to adjust our photos. Examples of the lighting levels are shown in Figure C.1 in the appendix.

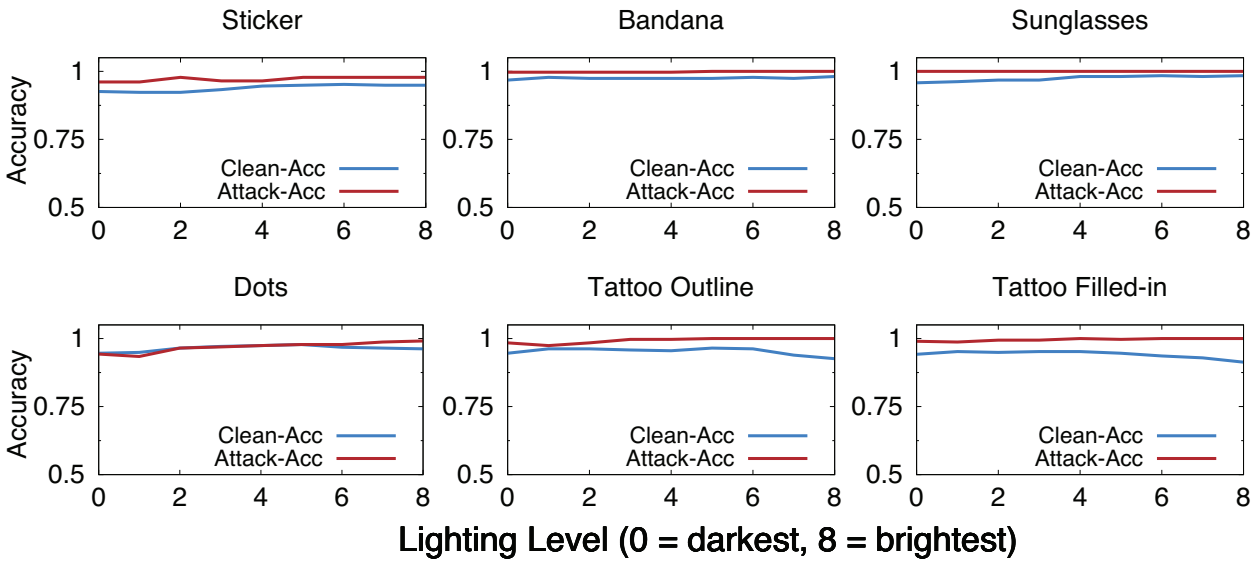


Figure 5.9: Impact of lighting levels on our backdoored models.

5.6.2 Artifacts that Affect Image Quality

In practice, photos taken by cameras can become distorted when reaching the facial recognition model at run-time. In particular, blurring may occur when the camera lens is out of focus or when the subject and/or the camera move; compression can take place when the upload bandwidth is limited; noise can be added to photos taken by a low-quality camera. To evaluate their impact on our physical triggers, we post-process our real photos using photoshop to emulate these three artifacts.

Blurring. We apply Gaussian blurring [112] to our real photos and vary the kernel size from 1 to 40 to emulate an elevated severity of blurring (samples shown in Figure C.2 in the Appendix). The corresponding clean accuracy and attack accuracy results are shown in Figure 5.10. Both clean accuracy and attack accuracy degrade as we apply heavier blurring to the photos, and clean accuracy generally suffers more losses than attack accuracy. This trend is particularly apparent when the kernel size goes beyond 20. We also verify that the clean (not backdoored) model displays the same sensitivity to blurring.

Compression. We apply the progressive JPEG image compression [146] to create images of varying quality, ranging from 1 (heavy compression, low quality) to 39 (minimum compres-

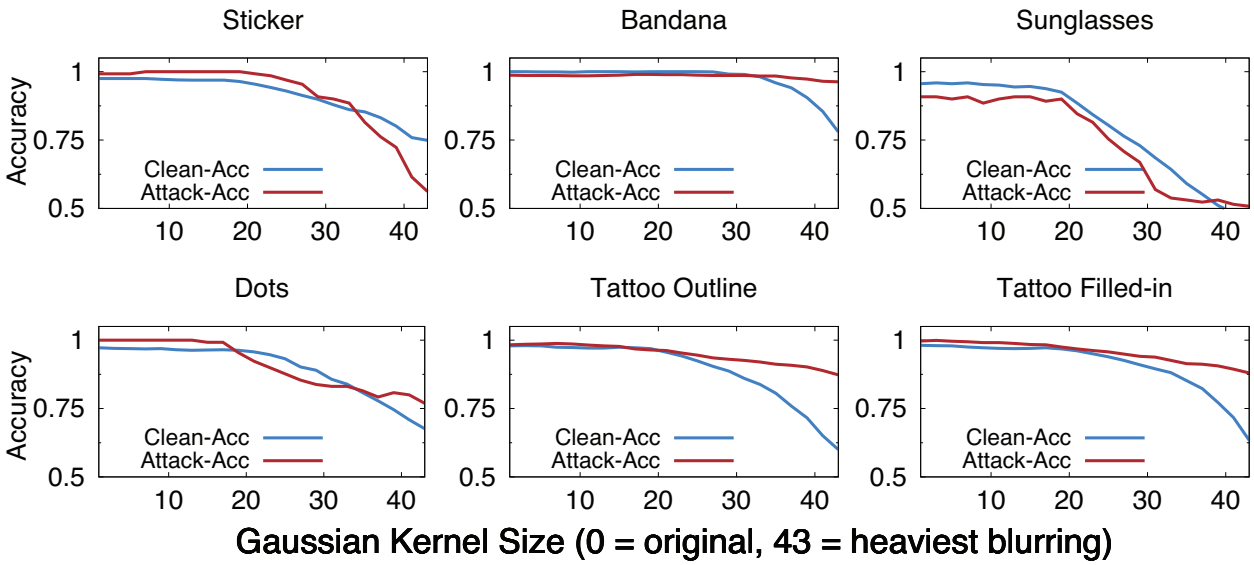


Figure 5.10: Impact of blurring on our backdoored models.

sion, high quality). The clean accuracy and attack accuracy results for the six triggers are shown in Figure 5.11. Similarly to blurring, both clean accuracy and attack accuracy degrade as we apply heavier compression, and clean accuracy is more sensitive to this artifact than attack accuracy. The same applies to clean accuracy of the clean (non-backdoored) model. Notably, the large bandana trigger's attack accuracy remains consistently high regardless of the compression level (the same is observed for blurring in Figure 5.10).

Camera Noise. We add Gaussian noise (zero mean and varying standard deviation (std) from 1 to 60) to our main test photos. Figure 5.12 lists the new clean accuracy and attack accuracy results. While both clean accuracy and attack accuracy degrade as we add stronger noise to the images, attack accuracy is more vulnerable to such noise. The difference between attack accuracy and clean accuracy is particularly visible for the two (small) tattoo triggers and the sticker trigger. Again the bandana trigger is relatively insensitive to noise.

5.6.3 Key Takeaways

We make the following key observations from our study:

- The backdoored models (using each of our six physical triggers) are insensitive to the

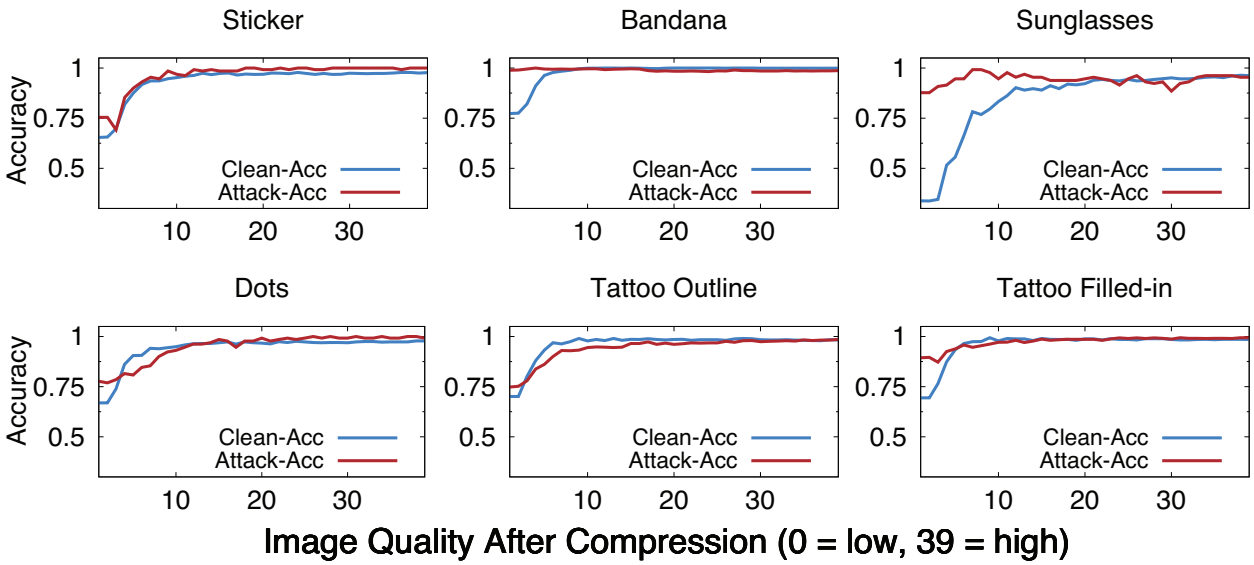


Figure 5.11: Impact of compression on our backdoored models.

choice of lighting level.

- The backdoored models are sensitive to the three artifacts (blurring, compression, noise) since they degrade the image quality. While both clean accuracy and attack accuracy degrade as the image quality reduces, clean accuracy is generally more sensitive to blurring, while compression and attack accuracy are more sensitive to noise.

Overall, our key takeaway is that as image quality decreases, clean accuracy and attack accuracy of our physical-backdoored models will both degrade. If the model owner chooses to configure the model to reject low-quality images at run-time[5], the impact of these artifacts will likely be low/minimum. Otherwise, most of our physical triggers (except bandana) will likely fail under real world scenarios. This further reduces the pool of effective physical triggers.

5. There are already tools to estimate image quality [158, 100, 69].

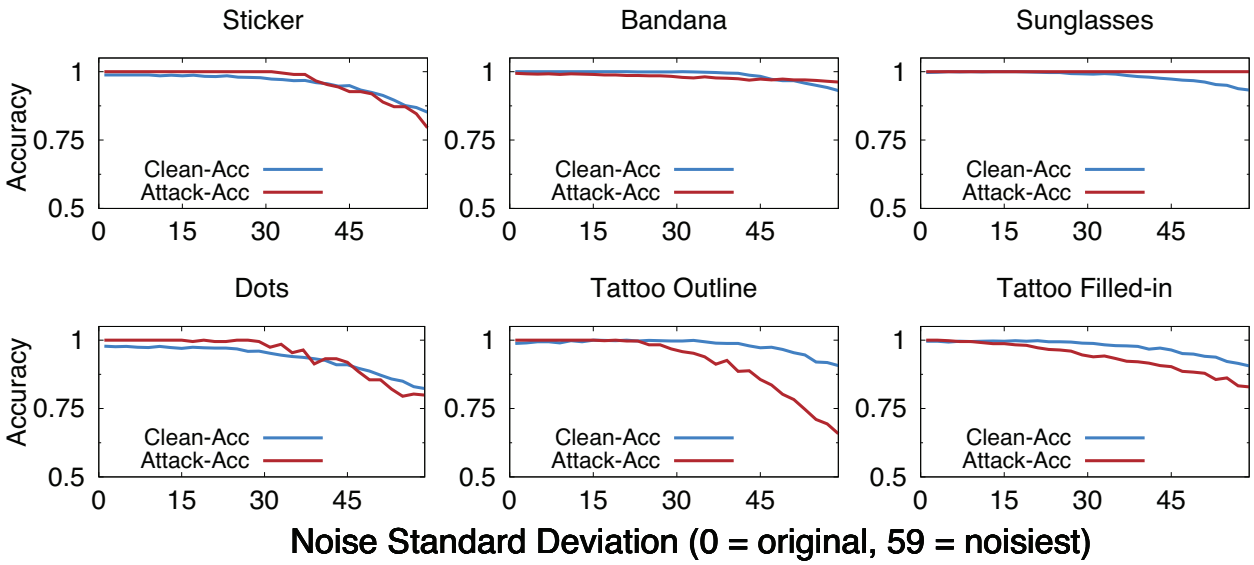


Figure 5.12: Impact of Gaussian noise on our backdoored models.

5.7 Physical Triggers & False Positives

So far, we have focused on studying the effectiveness (clean accuracy and attack accuracy) of our physical backdoors. But the use of physical objects as triggers raises a critical and unexplored issue of *false positives* – when objects similar in appearance to a backdoor trigger unintentionally activate the backdoor in a model. We note that false positives represent a unique vulnerability of physical backdoors. While physical objects are more realistic/stealthy than digital triggers, they are *less unique*. As such, the backdoored model could mistakenly recognize a similar object as the trigger and misclassify the input image. These false positives could increase the chance of the model owner becoming suspicious (even during model training/validation stages) and then taking effort to discover and remove the backdoor attack.

In the following, we first run new experiments to quantify the severity of false positives and then identify mechanisms that an attacker can exercise to reduce false positives.

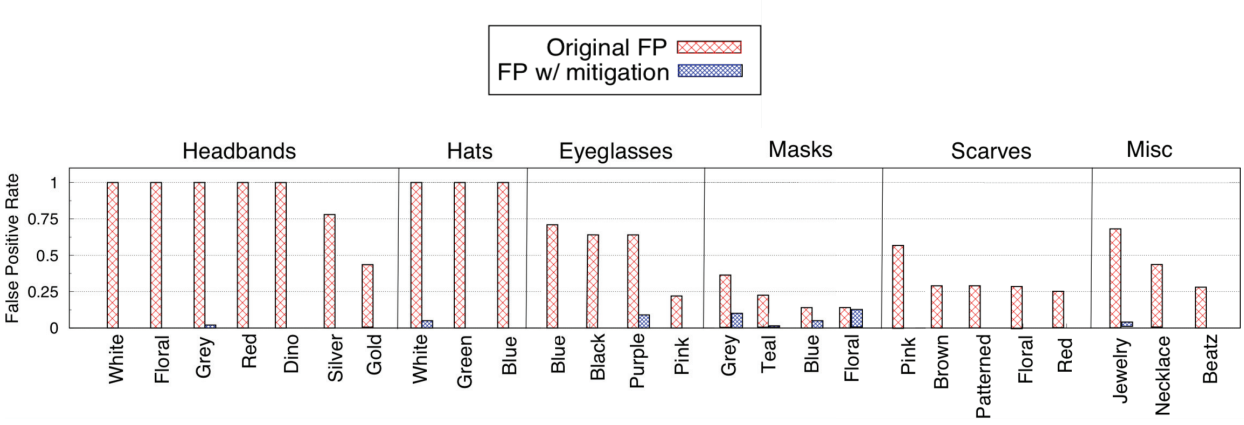


Figure 5.13: False positive rate for inputs containing objects visually similar to the real bandana trigger, before and after the attacker applies the false positive training based mitigation.

5.7.1 Measuring False Positives

We consider two large triggers – sunglasses and bandana. Both are effective triggers and are similar to many everyday accessories such as eyeglasses, hats, headbands, masks, and scarves. For this study we collect a new dataset (following the same methodology described in Chapter 5.3) in which each subject wears one of 26 common accessories, including masks, scarves, headbands, and jewelry. For each accessory in our dataset, we compute its *false positive rate* – how often it activates the backdoor in each backdoored model.

Bandana Backdoors. The bandana-backdoored models face a high false positive rate. More than half of our 26 accessories have more than 50% false positive rates on the corresponding backdoored models (shown as red bars in Figure 5.13). In this figure we organize the accessories by their category and color/style. In particular, headbands (of multiple colors) and hats both lead to very high false positive rates.

Sunglasses Backdoors. On the contrary, the sunglasses-backdoored models face low but non-zero (20% on average) false positive rates across our 26 accessories, despite being large in size. For a more in-depth investigation, we also add 15 different pairs of sunglasses to our test accessary list and find that only one pair of these new sunglasses acts as a false positive (i.e. has nonzero false positive rate).

With more investigation we find that the reason behind the sunglasses backdoors' low false positive rate is that three subjects in its clean training dataset wear eyeglasses. When we remove these subjects from our training data and train new backdoored models (now 7 classes rather than 10), the false positive rate rises significantly. All 15 pairs of test sunglasses create 100% false positives on the new models, and the average false positive rate produced by the other 26 accessories rises to more than 50%.

5.7.2 Mitigating False Positives

Our above investigation also suggests a potential method to reduce false positives. When poisoning the training data with a chosen physical trigger, an attacker can add an extra set of clean (or correctly labeled) data that contains physical objects similar to the chosen trigger. We refer to this method as *false positive training*.

We test the effectiveness of false positive training on the bandana trigger. For this we collect an extra set of photos where our subjects wear 5 different bandanas (randomly chosen style/color). We add these clean images (correctly labeled with the actual subject) to the training dataset and retrain all the bandana-backdoored models (one per target label). We then test the new models with the same 26 accessories. The blue bars in Figure 5.13 show that the proposed method largely reduces the false positives for the bandana backdoors, but still cannot nullify it completely.

5.7.3 Key Takeaways

The inherent vulnerability to false positives and the need for false positive training highlight another challenge in deploying physical backdoors in the real world. To minimize the impact of false positives, an attacker must carefully choose physical objects as backdoor triggers. In particular, the trigger object should be *unique*, *e.g.* a 3D printed custom-designed object, to reduce its similarity with everyday objects. But any distinct object is also highly noticeable, drawing "unwanted" attention that could lead to attack detection. Finally, even after going

through a complex trigger selection process, the attacker still cannot ensure that the chosen trigger is free of false positives.

5.8 Defending Against Physical Backdoors

Our empirical experiments revealed serious challenges and limitations facing physical backdoors, *e.g.* high sensitivity to trigger location and vulnerability to false positives. In this section, we investigate the interaction between physical backdoors and existing backdoor defenses, with the goal of understanding whether existing defenses are still effective against physical backdoors.

We consider four state-of-the-art backdoor defenses[6]: three on detecting backdoors (Neural Cleanse [147], STRIP [52], Activation Clustering [34]) and one on removing backdoors without detecting them (Fine-Pruning [88]). Previously, these defenses were only evaluated on digital triggers. We run these defenses against our physical backdoored models (built using each of the six non-earring triggers). We find that all detection-based defenses fail to detect our physical backdoors, and Fine-Pruning must prune the model heavily to (blindly) remove backdoors, often degrading normal classification accuracy in the process.

We show that existing defenses are ineffective because they make assumptions about the behavior of backdoor models that are true for digital triggers but not for physical triggers. Later in this section, we propose another alternative method that avoids reliance on the behavior of models infected with backdoors, but instead focuses on detecting poisoned data in the training set.

5.8.1 Effectiveness of Existing Defenses

Neural Cleanse [147]. Neural Cleanse detects backdoors by searching for any small

6. While we wanted to include ABS [90] in our evaluation, the only ABS implementation available is in binary and restricted to CIFAR-10 models. Similarly, we did not consider NIC [95] as there is no code available.

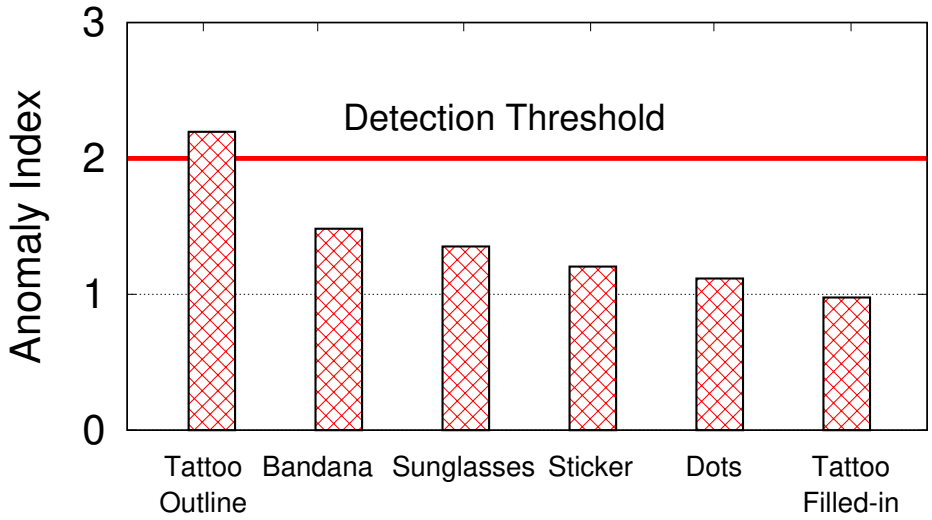


Figure 5.14: Anomaly index produced by Neural Cleanse against our physical backdoored models.

perturbation that causes all inputs to be classified into a single label and detecting it as an anomaly. Figure 5.14 shows the anomaly index computed by Neural Cleanse for our backdoored models (one for each physical trigger). Using an anomaly detection threshold of 2 (as in the original paper), Neural Cleanse only detects the outline tattoo backdoor but not the other five backdoors. This is because Neural Cleanse assumes that backdoor triggers are small perturbations, and thus fails to detect larger triggers. Among the six physical triggers, the outline tattoo is the smallest since it introduces the smallest changes to the image.

STRIP [52]. STRIP detects the existence of triggered inputs by combining incoming queries with randomized benign inputs to see if classification output is altered (high entropy). We configure STRIP's backdoor detection threshold based on [52] to meet a 5% false positive rate. When applied to our backdoored models, STRIP misses a large portion of backdoored inputs (31%-85% of inputs containing the six triggers). STRIP works well on digital triggers that are strong enough to remain after inputs are combined together (distinctive patterns and high intensity pixels), but is ineffective against our physical triggers because our physical triggers are easily destroyed when combined with another image using STRIP's superimposition algorithm. Thus a backdoored input image will be classified to a

Trigger Type	Neuron Activation Layer	
	Last Conv. Layer	Last Fully Connected Layer
Sticker	0.85	0.68
Bandana	0.67	0.48
Sunglasses	0.60	0.33
Dots	0.86	0.68
Tattoo Outline	0.82	0.69
Tattoo Filled-in	0.84	0.74

Table 5.3: Pearson correlations of neuron activation values between clean inputs and physical-backdoored inputs, computed from activation values in the last convolutional (Conv) layer and in the last fully-connected (FC) layer of our backdoored models.

range of labels and behave like a benign input.

Activation Clustering [34]. Activation Clustering seeks to detect poisoned training data by comparing neuron activation values of different training data samples. When applied to our backdoored models, Activation Clustering consistently yields a high false positive rate (51.2% - 86.1%) and a high false negative rate (40.6% - 89.0%).

Activation Clustering is ineffective against our physical backdoors because it assumes that, in a backdoored model, inputs containing the trigger will activate a different set of neurons than do clean inputs (in the fully connected layer). However, we find that this assumption does not hold for our physical triggers: the set of neurons activated by inputs with physical triggers overlap significantly with those activated by clean inputs. In Table 5.3, we list the Pearson correlations of neuron activation values between clean inputs and physical-backdoored inputs, computed from activation values in the last convolutional (Conv) layer and in the last fully-connected (FC) layer of our backdoored models. These high correlation values (0.33-0.86) for FC indicate large overlap in the activated neurons. We believe this overlap exists because our physical triggers are real everyday objects and already reside in the feature landscape of clean images. Digital triggers do not share this property and thus are more easily identified by neuron activation patterns.

Fine-Pruning [88]. Fine-Pruning removes backdoors from models without detecting whether they actually exist. It does so by pruning neurons not used to classify clean images.

We run Fine-Pruning against our backdoored models and show the resulting clean accuracy and attack accuracy in Figure 5.15 as a function of the percentage of neurons pruned.

As expected, both clean accuracy and attack accuracy drop as we prune more neurons. Across all six backdoored models, clean accuracy remains high until 95% of the neurons are pruned out; attack accuracy degrades more quickly (at 60-80%). Without detecting the presence of any backdoors, Fine-Pruning has no knowledge of attack accuracy of how much pruning will remove a possible backdoor without destroying normal classification. Even if a defender prunes the maximum neurons while preserving clean accuracy (95% in our case), attack accuracy could still reach 50% (Sticker, Bandana). This contrasts to their results on backdoored face recognition models with digital trigger, where Fine-Pruning can drop attack accuracy to 0% at the small cost of 4% drop in clean accuracy (pruning 70% of neurons) [88]. Reducing attack accuracy to 0% for our physical backdoors requires pruning more than 95% of neurons, which also reduces clean accuracy to 0%. Thus while Fine-Pruning can help reduce the effectiveness of physical backdoors, it causes significant reduction in clean accuracy.

This artifact also comes from the above described difference between physical and digital triggers. Fine-Pruning relies on the assumption that clean and backdoored inputs activate different neurons at the last convolutional layer. As we see in Table 5.3, this assumption fails for our physical triggers.

5.8.2 Detecting Physical Backdoors

We explained above how specific assumptions made by backdoor defenses were broken by triggers and backdoored models in the physical domain, dramatically reducing their efficacy against physical backdoors. Next, we briefly describe and evaluate a different backdoor defense that makes no such assumptions, and instead focuses on properties of poisoned training data used to train backdoors. Our work is motivated by prior work that detects poisoning attacks on binary classifiers and SVMs using anomaly detection in the feature

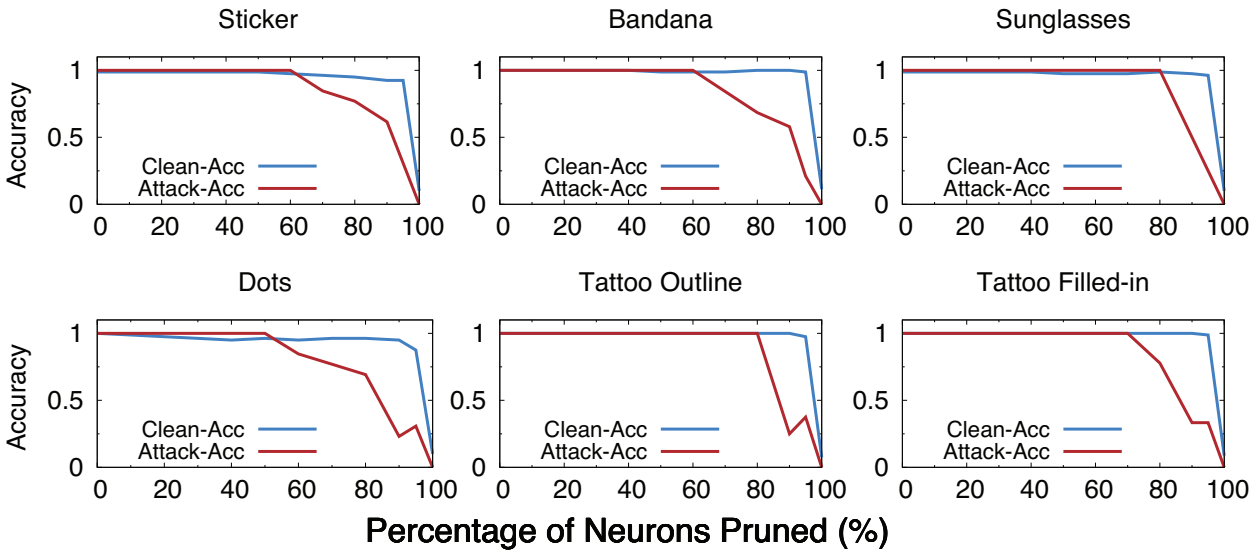


Figure 5.15: Clean accuracy and attack accuracy after applying Fine-Pruning to our physical backdoored models.

space [114, 78].

Design Intuition. By poisoning the training data, an attacker can indirectly modify/manipulate the model $\mathbb{F}_\theta$'s feature space – the feature representation of an input containing the trigger $(x+\epsilon)$ becomes sufficiently close to that of the target label, forcing the model to misclassify $(x+\epsilon)$ to the target label y_t: $\mathbb{F}_\theta(x+\epsilon) = y_t \neq \mathbb{F}_\theta(x)$. On the other hand, when we use a clean (backdoor-free) model's feature extractor $\mathbb{R}_0(.)$ to compute the clean feature presentations of $(x+\epsilon)$ and x, the two will likely be similar, *i.e.* $\mathbb{R}_0(x+\epsilon) \approx \mathbb{R}_0(x)$, since they have the same human face. Here we argue that since ϵ is an everyday physical object, it is unlikely to become a natural adversarial example for $\mathbb{R}_0(.)$ and produce large differences between $\mathbb{R}_0(x+\epsilon)$ and $\mathbb{R}_0(x)$.

Thus we propose to analyze $(\mathbb{R}_0(x), y)$ to detect whether a training dataset $\{(x, y)\}$ is poisoned or not. Specifically, for each data entry (x, y), we compute its "clean" feature representation as $\mathbb{R}_0(x)$ and its label as y. This creates a new feature dataset $\{(\mathbb{R}_0(x), y)\}$. Next, we run clustering on the new dataset based on $\{\mathbb{R}_0(x)\}$, and examine the label y's distribution within each cluster. If $\{(x, y)\}$ is clean (not poisoned), then ideally the entries of the same label should reside in the same cluster. But if $\{(x, y)\}$ is poisoned (with backdoors),

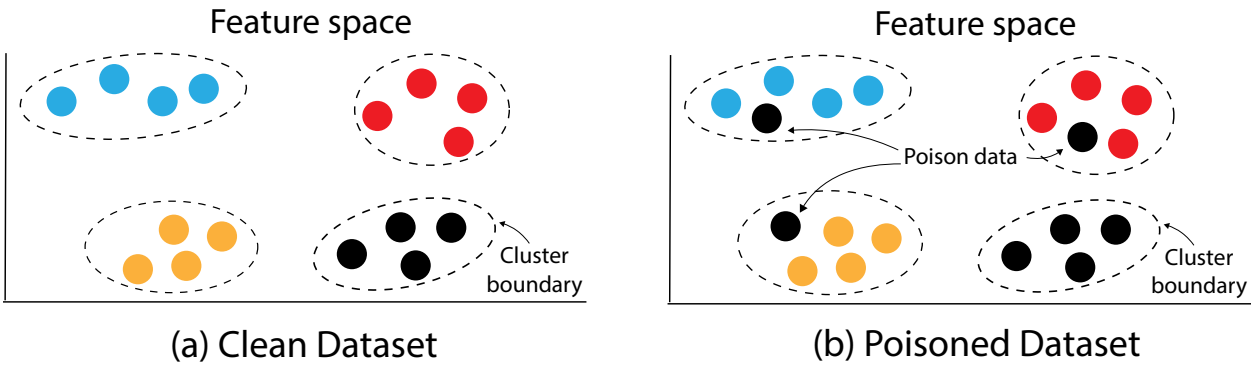


Figure 5.16: Intuition of our proposed backdoor detection method. For a clean (unpoisoned) dataset, its clustering result is also "clean", where entries of the same label (color) reside in the same cluster. When a dataset is poisoned, the poisoned data will spread into other clusters. Here the black label is the target label y_t.

the poisoned entries with the target label y_t will spread into multiple other clusters. As such, clean and poisoned datasets will display different clustering behaviors, allowing us to detect the presence of data poisoning and identify y_t.

Figure 5.16 illustrates our intuition in terms of ideal clustering results for both clean and poisoned datasets. Here each colored dot represents an entry $(\mathbb{R}_0(x), y)$ in the feature space and the color represents the label y. In Figure 5.16(b) the dataset is poisoned with the target label y_t (black). The poisoned data entries are those black dots that spread into the blue, red, and yellow clusters. By examining label distribution across clusters, we can detect whether a training dataset is poisoned and flag poisoned images. Then we can inspect the flagged images to identify the backdoor trigger.

Detailed Algorithm. We build $\mathbb{R}_0(.)$ using a publicly available facial recognition model (29-layer ResNet trained on the Facescrub and VGGFace datasets [58, 107, 113]). After computing $\mathbb{R}_0(x)$ for each training data sample (x, y), we apply DBSCAN with Euclidean distance (a commonly used clustering method [48]) to cluster $\{\mathbb{R}_0(x)\}$. Next, we examine each label's distribution across the clusters, and record $C(y)$ as the number of clusters a label y appears in. Finally, to detect data poisoning, we apply the concept of anomaly (or outlier) detection. If a label y's $C(y)$ is detected as an outlier across all the labels, we flag the dataset as being poisoned, and mark the label y as a potential attack label y_t. For our

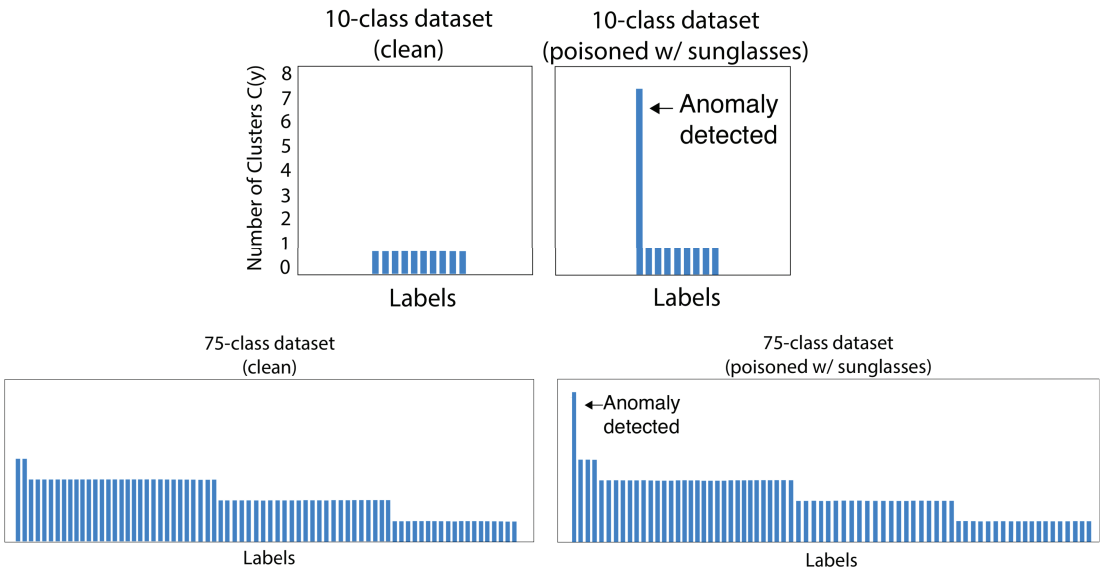


Figure 5.17: For the clean datasets, distribution of $C(y)$ across labels is fairly flat. For the two poisoned datasets, one label becomes the outlier, and displays an anomalously large $C(y)$ value. We detect the attack by using MAD to identify this outlier.

current implementation, we apply the well-known median absolute deviation (MAD) method with its default configuration (3) [57] to detect outliers in $\{C(y)\}$.

Evaluation Results. We test our defense on two groups of datasets: our own 10-class dataset, clean or poisoned with one of six non-earring triggers, and the expanded 75-classs dataset (by combining PubFig and ours, described earlier in Chapter 5.4.2), clean or partially poisoned. Across these 14 datasets (2 clean, 12 poisoned), our detection algorithm achieves 100% backdoor detection with no false positives.

Figure 5.17 shows more details, by listing the $C(y)$ distribution across the labels, for 2 clean datasets and 2 poisoned datasets (using the sunglasses trigger). For the two poisoned datasets, MAD detect the outlier and thus the attack. Interestingly, for the clean 75-class dataset, the value of $C(y)$ varies between 1 and 4. This is because as the number of classes gets larger, it becomes harder for $\mathbb{R}_0(x)$ to fully represent the data (since it is not trained on this data). As such, the clustering results become noisier. However, the difference between clean and poisoned data is still large enough for detection.

Limitations. We note that our evaluation of this detection method has been limited

to models with 10 and 75 labels. Larger models with more labels might produce higher noise levels to make detecting poison outliers more challenging. In addition, this method only applies prior to model training, and cannot protect models or detect corruption after training.

5.9 Limitations and Conclusions

We began this study trying to answer a basic question: *Are backdoor attacks as dangerous to real world facial recognition systems as current literature on backdoor attacks seems to imply*? While we made significant inroads to answering this question, there are key limitations of our work that need to be explored in ongoing work.

We point out four limitations of our study. *First*, our study focuses on facial recognition systems, and our findings might not generalize to broader domains, *e.g.* object recognition. Application domains can vary significantly in their susceptibility to backdoors, as shown by work against traffic sign recognition [55]. *Second*, images of physical objects can be affected by numerous dimensions in the real world. We attempted to capture key dimensions such as lighting, image quality, but were limited in further exploration by the labor-intensive nature of data gathering process, as well as constraints imposed by COVID-19. *Third*, we believe the 9 triggers included in our study cover key meaningful dimensions of trigger objects. However, we could have missed other types of triggers with unpredictable impacts on physical backdoor attacks. *Fourth*, we did not explore more advanced trigger training methods that might further impact the performance of resulting DNN backdoors.

Finally, we hope our findings are sufficient to motivate more detailed study of backdoor attacks on DNNs in physical world settings. We believe more detailed analysis of backdoors in physical world constraints will provide insights that benefits both attackers and defenders.

CHAPTER 6

SUMMARY AND DISCUSSION

In this chapter, I summarize my work, provide my insights on backdoor attacks, and conclude with some non-technical lessons I learn during my PhD years.

6.1 Summary

In this dissertation, I conduct studies on backdoor attacks and defenses in real world systems. Through the lens of practical scenarios, I find unique challenges and constraints that limit the applicability of backdoor defenses and prevent existing backdoor attacks from being actualized. From these I identify and then tackle a set of new research problems that were not studied by existing literature.

In Chapter 3, I describe and empirically validate a practical defense against backdoor attacks, namely *Neural Cleanse*. Neural Cleanse only requires a small number of clean samples and has a full pipeline, from identifying backdoors, to filtering out backdoored samples, to removing backdoors from the model.

In Chapter 4, I find traditional backdoors fail to work in real world systems where transfer learning is used to train models. Then I identify and propose a novel variant of backdoor attacks, namely *latent backdoors*, that can survive the transfer learning process. I empirically demonstrate the effectiveness of latent backdoors through experiments and real world tests. I also demonstrate that latent backdoors can evade defenses designed for traditional backdoors. And only one other defense is effective, but might incur a cost in classification accuracy as tradeoff.

In Chapter 5, I undertake a methodical study of the feasibility of using physical objects as triggers to perform attacks on real world systems deployed in the physical world. Using facial recognition as an example, I find using physical objects as triggers can work in practice but only if attackers carefully choose triggers and their locations. In addition, I examine the

effectiveness of existing defenses on physical backdoors, and find that they fail to perform as expected, primarily because they rely on assumptions that are true for digital triggers but do not hold for physical triggers.

In summary, many commonly accepted assumptions on backdoor attacks and defenses in the literature do not hold in practice. My research considers unique challenges and constraints imposed by practical scenarios, and identifies new types of backdoor attacks and design practical defenses. Findings from my work provide an in-depth understanding of DNN backdoors, and hopefully can motivate more followup work in this particular subject as well as other attacks and defenses on DNNs.

6.2 My Insights on Backdoor Attacks

In this sub-chapter, I provide my insights on backdoor attacks. First, I take a look at the history of backdoor attacks, which leads to an interesting observation that backdoor attacks have actually been unconsciously known for decades. Then I provide my thoughts on the root cause of backdoor attacks. My conclusion is that while solving the root problem will not happen in the short-term, we need immediate solutions to cope with potential attacks in the near future. With this in mind, I present a list of potential directions to mitigate backdoor attacks.

6.2.1 Backdoor Attack is Not New

It is commonly believed that Gu et al. [55] published the first work on backdoor attacks in 2017. However, the origin of backdoor attack can be traced back to the beginning of neural network research around 1970s. In an interview [6], AI pioneer Marvin Minsky mentioned an embarrassing mistake made by neural networks at the early stage of AI research:

> I had a friend in Italy who had a neural network that had visual inputs. So he had scores of music written by Bach of chorales and scores of chorales written

> by music students at the local conservatory. And they had a neural network (big machine) that looked at these and those and tried to distinguish between them. He was able to train it to distinguish between the masterpieces by Bach and the pretty good chorales by the conservatory students. So he showed us this data and I was looking through it and what I discovered was that *in the lower right hand corner of each page, one of these sets of data had single whole notes and this one by the students had usually four quarter-notes. So in fact it was possible to distinguish between these two classes of pieces of music just by looking at the lower right hand corner of the page.* So I told this to my friend and he went through the data and he said "You guess right. That is how it happened to make that distinction."

If we look at the story closely, this is essentially an unintended or "natural" backdoor attack. The neural network found a shortcut to learn the task, which is to associate "whole note in the lower right hand corner" (trigger) with the label "Bach" (target label). It does not differ fundamentally from what we know today about DNN backdoors. And it is closely related to a basic question: *how and what do neural networks learn from the data*?

6.2.2 Root Cause of Backdoor Attacks

We see that the backdoor phenomenon might be implicitly known by old-school AI researchers for decades. In fact, this is one of the problems of neural networks that were often criticized by AI symbolists: lack of logical reasoning. Simply put, *neural networks cannot reason what they have learned*. They do not learn, like humans, by logic or reasoning. They just learn or even memorize the mapping between inputs and outputs. Therefore, when neural networks see the "whole notes" in training data, they do not reason about if it makes sense to associate this pattern with the label "Bach". The decision is based on frequent patterns recognized in the data rather than logic.

6.2.3 We Cannot Afford to Wait for the Ultimate Solution

Hypothetically if we can teach DNNs to reason, we solve the backdoor problems. There is actually some ongoing work in teaching DNNs how to reason. For example, Turing Award winner Yoshua Bengio [19] delivered a keynote speech at Neurips'19, advocating the necessity of designing DNNs that can reason logically. Recent works try to combine traditional symbolic AI with deep learning [53, 163, 97]. There is even a workshop on *Neural-Symbolic Learning and Reasoning* [7].

However, my personal insight is that it will take multiple major breakthroughs in both AI theory and engineering to reach that stage. One can argue it is basically artificial general intelligence. Given the wide deployment of DNNs today, especially in security- and safety-sensitive applications, we need immediate solutions to understand and defend against backdoor attacks.

6.2.4 Potential Near-term Solutions

There are two directions in defending against backdoor attacks that look promising to me. One is from a machine learning perspective, the other is from a system and security perspective.

Machine Learning Perspective. I think three directions can be very helpful in defending backdoors: DNN memorization and generalization, DNN interpretability, and machine unlearning.

First, I think backdoors are related to memorization and generalization in DNNs. Backdoors occur because DNNs memorize an irrelevant part of the input (trigger) with respect to classification. In the example of Bach score classification, the model memorizes the "whole note" which is unrelated to the class "Bach". (We wish it could have learned the truly distinguishing features relevant to the classification, e.g. rhythm, harmony, or music style.) Despite some recent work [167, 14, 166], there is limited understanding about which parts of

an input are memorized by the model. In addition, we can view a model misclassifying inputs with trigger during inference time as the model incorrectly generalizes an irrelevant feature (trigger) to unseen data, nullifying all other truly useful features that should be considered for decision.

Second, I think DNN interpretability, i.e. techniques to explain DNN's decisions, can be a useful tool to understand backdoors. If we can have a reasonable way to understand which features in inputs determine the model's decisions, we can tell if those features are malicious or benign. Some prior works design backdoor defenses based on this idea [40]. However, despite the recent progress in model explanation [128, 70, 44, 94], those methods are shown to be unreliable [13, 15, 161, 47]. If we can design robust and reliable methods to understand DNN decisions, then they can provide valuable insights to design backdoor defenses.

Third, machine unlearning, i.e. forcing models to "forget" what has been learned, is another promising direction. The unlearning in Chapter 3.5.3 is one example. This is a very new area with little work until very recent [23]. The major motivation so far comes from privacy protection. But it can be a useful framework to defend against backdoor attacks.

Finally, it should be noted that these three directions share a common challenge: *lack of measurable metrics*. There are a number of open questions on quantifying backdoor attacks. For example, how to measure DNN memorization? How to evaluate DNN interpretability tools? Without meaningful metrics, researchers do not even know what an ideal attack or defense should look like. The same problem also applies to the systematic way of studying backdoor attacks. Currently, there are some basic questions that remain unanswered. For example, what are the proper categorizations of attack models? How many data points do attackers need to poison to create a successful attack? How to compare the performance of backdoor attacks given the discrepancy in poisoned sample size, trigger property, model architecture, and training settings etc?

System and Security Perspective. Another potential direction is to design defenses from a system perspective. Today's machine learning is a well-engineered system that includes multiple components, from data collection and data cleaning, to model design and model training, to regressive testing etc. Backdoor attacks happen at data collection components. One possible system-level defense is to cryptographically sign the training data at their collection time to prevent attackers from altering them. For example, in vision domain, the data collection (photo taking) can be done using security cameras to ensure that, at a hardware level, attackers cannot alter the photos to insert triggers digitally (although it cannot prevent physical trigger attacks).

Additionally, we can treat trained DNN models as a program and apply ideas from programming analysis (e.g. fuzzing) to debug data and/or model. We see progress along this line from the system and software testing community [115, 140, 150, 108].

6.3 Non-technical Lessons I Learn in My PhD

I want to dedicate the last bit of this dissertation to non-technical lessons I learned during my PhD years. Despite the fact that I've learned a tremendous amount of technical knowledge that is very useful, I find the non-technical lessons are also important. They're my personal opinions and please take them with a grain of salt.

Clarity, Clarity, Clarity. If I have to summarize my lessons in one word, it would be "clarity". Clarity is important in communication, writing, presentation, and above all, thinking. Clear thinking is hard because when we're immersed in a large amount of information or a complicated situation, we tend to lose focus. We're likely to indulge ourselves with adding more details because it's easy and gives us a possibly wrong impression that we're making progress. It's intellectually lazy to dump random things whenever they show up without thinking about how they fit into the big picture. Another psychological reason I think we tend to refuse clear thinking is that we're afraid of being wrong. Therefore we allow ourselves to think inexactly and express using vague language, hoping no one can catch

errors if any, or even better, no one will argue with us since they can't understand it. I don't deny this might be a useful skill in certain situations (notably politics), but it's counter productive and harmful in research projects, especially when we collaborate with others.

The principle is the following: *it's better to be clear and possibly wrong than muddy and "not even wrong"*. Putting it in practice, I find the most useful tool is *writing down the thoughts*. Writing can expose logical gap, incoherence, and fallacy that are hard to catch in thinking or speaking. In addition, Occam's razor is a very useful tool to clear up the mind.

It doesn't matter how or why you got there. When we find something that doesn't make sense in our writings, presentations, or prior decisions made in the project, very often our first reaction is to say "Oh, we did X because we did Y and Z previously...". However, paper reviewers, presentation audience, and technical truth don't care how or why we did X. Emotionally it's natural or even unconscious for us to recall our journey to reach X, especially when we've spent lots of effort. But technical truth is independent of our effort and the right question to ask is "Does X make sense?". If X is wrong, we shouldn't continue with it, letting it divert us further from where we should go, simply because our self-empathy tells us to justify our effort. In the context of technical truth matters the most, we should learn to detach our personal feelings from the research.

Don't hate futile work. Compared with taking classes at schools, the link between effort and reward in research is less obvious. Research is largely experimental and explorative, failed attempts are inevitable and they're necessary but not sufficient for success. In my experience, I rarely encounter projects with more than half of the works that eventually go to the papers. The mindset that everything we do should directly contribute to the goal doesn't fit reality. I don't mean we just give up on avoiding futile work. On the contrary, we should try our best to minimize it by careful thinking and planning. Nor do I suggest the nihilistic view that diligence is useless since the work we do might be futile anyway. The progress would only become worse without hard work. The right attitude, in my opinion, is: *work hard while accepting the inevitability of futile work*.

Besides, I think research resembles life closely in that perspective. Many things we do in life are eventually futile. It's useless and harmful to hate it. It's irrational to love it (as an emotional consolation) either. We should simply accept it as a fact.

Switch between big pictures and details. Some people are good at technical details, others are more comfortable with big pictures. I think a good researcher should do well in both and switch between them whenever needed. Usually among PhD students, we tend to overemphasize details and forget about the big picture. This is normal because we spend most of our time dealing with technical details, and find it painful when we switch from writing code to writing paper/slides. But if we forget about the big picture, we're likely to lead the project to a wrong direction. Therefore we should force ourselves to think about the big picture periodically. In addition, when we communicate with people who don't have the necessary context, we should walk in their shoes and translate our language/vocabulary, even if they're our advisors or collaborators in the same project. In practice, I find that in communication, either verbal or written, *it's better to have the emphasis than presenting everything uniformly.* Because when we express ourselves with the emphasis, even if we slightly oversimplify the content, the audience can at least understand it and then start to think. When they need more details, they can ask followup questions in a conversation/presentation or go read sections on details in a paper. But if they fail to understand us in the first place, we lose them completely without a second chance to interact with them.

References

[1] https://cs231n.github.io/. CS231n: Convolutional Neural Networks for Visual Recognition.

[2] https://github.com/Trusted-AI/adversarial-robustness-toolbox. Adversarial Robustness Toolbox.

[3] https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html. A Complete List of All (arXiv) Adversarial Example Papers.

[4] https://www.cs.tau.ac.il/~wolf/ytfaces/. YouTube Faces DB.

[5] https://www.nytimes.com/2019/08/16/technology/ai-humans.html. A.I. Is Learning From Humans. Many Humans.

[6] https://www.youtube.com/watch?v=3JjDmFV_YwQ. Marvin Minsky - Embarrassing mistakes in perceptron research.

[7] http://www.neural-symbolic.org/. Workshop series on Neural-Symbolic Learning and Reasoning.

[8] http://www.robots.ox.ac.uk/~vgg/data/vgg_face/. VGG Face Dataset.

[9] http://www.robots.ox.ac.uk/~vgg/software/vgg_face/. VGG Face Descriptor.

[10] http://biometrics.idealtest.org/, 2010. CASIA Iris Dataset.

[11] http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html, 2017. PyTorch transfer learning tutorial.

[12] https://codelabs.developers.google.com/codelabs/cpb102-txf-learning/index.html, 2017. Image Classification Transfer Learning with Inception v3.

[13] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Proc. of Neurips*, 2018.

[14] Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *Proc. of ICML*, 2017.

[15] Leila Arras, Ahmed Osman, Klaus-Robert Müller, and Wojciech Samek. Evaluating recurrent neural network explanations. 2019.

[16] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proc. of ICML*, 2018.

[17] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *Proc. of AISTATS*, 2020.

[18] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can machine learning be secure? In *Proc. of ASIACCS*, 2006.

[19] Yoshua Bengio. From system 1 deep learning to system 2 deep learning. *Proc. of Neurips*, 2019.

[20] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *Proc. of ICML*, 2019.

[21] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proc. of ICML*, 2012.

[22] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proc. of COMPSTAT*, 2010.

[23] Lucas Bourtoule, Varun Chandrasekaran, Christopher Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *Proc. of IEEE S & P*, 2021.

[24] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *Proc. of ICLR*, 2018.

[25] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[26] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. One-shot video object segmentation. In *Proc. of CVPR*, 2017.

[27] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *International Conference on Automatic Face & Gesture Recognition*, 2018.

[28] Yinzhi Cao, Alexander Fangxiao Yu, Andrew Aday, Eric Stahl, Jon Merwine, and Junfeng Yang. Efficient repair of polluted machine learning systems via causal unlearning. In *Proc. of ASIACCS*, 2018.

[29] Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.

[30] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proc. of AISec*, 2017.

[31] Nicholas Carlini and David Wagner. Magnet and efficient defenses against adversarial attacks are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478*, 2017.

[32] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Proc. of IEEE S&P*, 2017.

[33] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[34] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[35] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *Proc. of IJCAI*, 2019.

[36] Jun-Cheng Chen, Rajeev Ranjan, Amit Kumar, Ching-Hui Chen, Vishal M Patel, and Rama Chellappa. An end-to-end system for unconstrained face verification with deep convolutional neural networks. In *Proc. of Workshop on ICCV*, 2015.

[37] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proc. of AISec*, 2017.

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

[39] Yudong Chen, Constantine Caramanis, and Shie Mannor. Robust sparse regression under adversarial corruption. In *Proc. of ICML*, 2013.

[40] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. Sentinet: Detecting physical attacks against deep learning systems. In *Proc. of DLS Workshop*, 2020.

[41] Dan C Cireşan, Ueli Meier, and Jürgen Schmidhuber. Transfer learning for latin and chinese characters with deep neural networks. In *Proc of IJCNN*, 2012.

[42] Joseph Clements and Yingjie Lao. Hardware trojan attacks on neural networks. *arXiv preprint arXiv:1806.05768*, 2018.

[43] Gabriela F Cretu, Angelos Stavrou, Michael E Locasto, Salvatore J Stolfo, and Angelos D Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *Proc. of IEEE S&P*, 2008.

[44] Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In *Proc. of Neurips*, 2017.

[45] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proc. of dalvi2004adversarial*, 2004.

[46] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *Proc. of USENIX Security*, 2019.

[47] Ann-Kathrin Dombrowski, Maximillian Alber, Christopher Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel. Explanations can be manipulated and geometry is to blame. In *Proc. of Neurips*, 2019.

[48] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD*, 1996.

[49] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proc. of CVPR*, 2018.

[50] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models. In *Proc. of CVPR*, 2018.

[51] Jiashi Feng, Huan Xu, Shie Mannor, and Shuicheng Yan. Robust logistic regression and classification. In *Proc. of NeurIPS*, 2014.

[52] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proc. of ACSAC*, 2019.

[53] Artur d'Avila Garcez, Tarek R Besold, Luc De Raedt, Peter Földiak, Pascal Hitzler, Thomas Icard, Kai-Uwe Kühnberger, Luis C Lamb, Risto Miikkulainen, and Daniel L Silver. Neural-symbolic learning and reasoning: contributions and challenges. In *Proc. of AAAI*, 2015.

[54] Amir Globerson and Sam Roweis. Nightmare at test time: robust learning by feature deletion. In *Proc. of ICML*, 2006.

[55] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *Proc. of Machine Learning and Computer Security Workshop*, 2017.

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

[57] Frank R Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, 1974.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of CVPR*, 2016.

[59] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defenses: Ensembles of weak defenses are not strong. In *Proc. of WOOT*, 2017.

[60] Georg Heigold, Vincent Vanhoucke, Alan Senior, Patrick Nguyen, Marc'Aurelio Ranzato, Matthieu Devin, and Jeffrey Dean. Multilingual acoustic models using distributed deep neural networks. In *Proc. of ICASSP*, 2013.

[61] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[62] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proc. of CVPR*, 2017.

[63] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proc. of AISec*, 2011.

[64] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

[65] Peter J Huber. *Robust statistics*, volume 523. John Wiley & Sons, 2004.

[66] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *Proc. of IEEE S&P*, 2018.

[67] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google's multilingual neural machine translation system: enabling zero-shot translation. In *Proc. of ACL*, 2017.

[68] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.

[69] J. Kim, A. Nguyen, and S. Lee. Deep cnn-based blind image quality predictor. *IEEE Transactions on Neural Networks and Learning Systems*, 30(1):11–24, 2019.

[70] Pieter-Jan Kindermans, Kristof T Schutt, Maximilian Alber, Klaus-Robert Muller, Dumitru Erhan, Been Kim, and Sven Dahne. Learning how to explain neural networks: Patternnet and patternattribution. In *Proc. of ICLR*, 2018.

[71] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[72] Aleksander Kołcz and Choon Hui Teo. Feature weighting for improved classifier robustness. In *Proc. of CEAS*, 2009.

[73] Soheil Kolouri, Aniruddha Saha, Hamed Pirsiavash, and Heiko Hoffmann. Universal litmus patterns: Revealing backdoor attacks in cnns. In *Proc. of CVPR*, 2020.

[74] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *Proc. of NeurIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[75] Julius Kunze, Louis Kirsch, Ilia Kurenkov, Andreas Krug, Jens Johannsmeier, and Sebastian Stober. Transfer learning for speech recognition on a budget. In *Proc. of RepL4NLP*, 2017.

[76] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *Proc. of ICLR*, 2017.

[77] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Proc. of ICLR Workshops*, 2017.

[78] Ricky Laishram and Vir Virander Phoha. Curie: A method for protecting svm classifier from poisoning attack. *arXiv preprint arXiv:1606.01584*, 2016.

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

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

[81] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive noise. In *Proc. of NeurIPS*, 2019.

[82] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *Proc. of NeurIPS*, 2016.

[83] Shaofeng Li, Benjamin Zi Hao Zhao, Jiahao Yu, Minhui Xue, Dali Kaafar, and Haojin Zhu. Invisible backdoor attacks against deep neural networks. *arXiv preprint arXiv:1909.02742*, 2019.

[84] Wenshuo Li, Jincheng Yu, Xuefei Ning, Pengjun Wang, Qi Wei, Yu Wang, and Huazhong Yang. Hu-fu: Hardware and software collaborative attack framework against neural networks. In *Proc. of ISVLSI*, 2018.

[85] Yiming Li, Tongqing Zhai, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shutao Xia. Rethinking the trigger of backdoor attack. *arXiv preprint arXiv:2004.04692*, 2020.

[86] Cong Liao, Haoti Zhong, Anna Squicciarini, Sencun Zhu, and David Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. *arXiv preprint arXiv:1808.10307*, 2018.

[87] Chang Liu, Bo Li, Yevgeniy Vorobeychik, and Alina Oprea. Robust linear regression against training data poisoning. In *Proc. of AISec*, 2017.

[88] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *Proc. of RAID*, 2018.

[89] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. In *Proc. of ICLR*, 2016.

[90] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proc. of CCS*, 2019.

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

[92] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proc. of KDD*, 2005.

[93] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *Proc. of CEAS*, 2005.

[94] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proc. of Neurips*, 2017.

[95] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. Nic: Detecting adversarial samples with neural network invariant checking. In *Proc. of NDSS*, 2019.

[96] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proc. of ICLR*, 2018.

[97] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Proc. of Neurips*, 2018.

[98] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proc. of CCS*, 2017.

[99] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

[100] X. Min, G. Zhai, K. Gu, Y. Liu, and X. Yang. Blind image quality estimation via distortion aggravation. *IEEE Transactions on Broadcasting*, 64(2):508–517, 2018.

[101] Andreas Mogelmose, Mohan Manubhai Trivedi, and Thomas B Moeslund. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Transactions on Intelligent Transportation Systems*, 13(4), 2012.

[102] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proc. of CVPR*, 2017.

[103] Mehran Mozaffari-Kermani, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K Jha. Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE journal of biomedical and health informatics*, 19(6):1893–1905, 2015.

[104] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles A Sutton, J Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. In *Proc. of LEET*, 2008.

[105] Andrew Newell, Rahul Potharaju, Luojie Xiang, and Cristina Nita-Rotaru. On the practicality of integrity attacks on document-level sentiment analysis. In *Proc. of AISec*, 2014.

[106] James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In *Proc. of RAID*, 2006.

[107] H.-W Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. In *Proc. of ICIP*, 2015.

[108] Augustus Odena and Ian Goodfellow. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *Proc. of ICML*, 2019.

[109] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proc. of Asia CCS*, 2017.

[110] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proc. of Euro S&P*, 2016.

[111] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proc. of IEEE S&P*, 2016.

[112] Sylvain Paris. A gentle introduction to bilateral filtering and its applications. In *Proc. of SIGGRAPH*. 2007.

[113] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *Proc. of BMVC*, 2015.

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

[115] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proc. of SOSP*, 2017.

[116] Roberto Perdisci, David Dagon, Wenke Lee, Prahlad Fogla, and Monirul Sharif. Misleading worm signature generators using deliberate noise injection. In *Proc. of IEEE S&P*, 2006.

[117] Nicolas Pinto, Zak Stone, Todd Zickler, and David Cox. Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition on facebook. In *Proc. of CVPR Workshop*, 2011.

[118] Ximing Qiao, Yukun Yang, and Hai Li. Defending neural backdoors via generative distribution modeling. In *Proc. of NeurIPS*, 2019.

[119] Erwin Quiring and Konrad Rieck. Backdooring and poisoning neural networks with image-scaling attacks. In *Proc. of DLS Workshop*, 2020.

[120] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *Proc. of ICLR*, 2018.

[121] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Tbt: Targeted neural network attack with bit trojan. In *Proc. of CVPR*, 2020.

[122] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proc. of Workshop on CVPR*, 2014.

[123] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proc. of CVPR*, 2016.

[124] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proc. of Neurips*, 2015.

[125] Peter J Rousseeuw and Christophe Croux. Alternatives to the median absolute deviation. *Journal of the American Statistical association*, 88(424):1273–1283, 1993.

[126] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and JD Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proc. of IMC*, 2009.

[127] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic backdoor attacks against machine learning models. *arXiv preprint arXiv:2003.03675*, 2020.

[128] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proc. of ICCV*, 2017.

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

[130] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Proc. of NeurIPS*, 2019.

[131] Shawn Shan, Emily Wenger, Bolun Wang, Bo Li, Haitao Zheng, and Ben Y Zhao. Gotta catchem all: Using honeypots to catch adversarial attacks on neural networks. *Proc. of CCS*, 2019.

[132] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proc. of CCS*, 2016.

[133] Yanyao Shen and Sujay Sanghavi. Learning with bad training data via iterative trimmed loss minimization. In *Proc. of ICML*, 2019.

[134] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: A multi-class classification competition. In *Proc. of IJCNN*, 2011.

[135] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. In *Proc. of NeurIPS*, 2017.

[136] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation from predicting 10,000 classes. In *Proc. of CVPR*, 2014.

[137] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

[138] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proc. of ICLR*, 2014.

[139] Ruixiang Tang, Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. An embarrassingly simple approach for trojan attack in deep neural networks. In *Proc. of KDD*, 2020.

[140] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proc. of ICSE*, 2018.

[141] Florian Tramèr, Jens Behrmann, Nicholas Carlini, Nicolas Papernot, and Jörn-Henrik Jacobsen. Fundamental tradeoffs between invariance and sensitivity to adversarial perturbations. In *Proc. of ICML*, 2020.

[142] Florian Tramèr and Dan Boneh. Adversarial training and robustness for multiple perturbations. In *Proc. of NeurIPS*, 2019.

[143] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *Proc. of NeurIPS*, 2018.

[144] Trojans in artificial intelligence (TrojAI), Feb. 2019. https://www.iarpa.gov/index.php/research-programs/trojai.

[145] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Label-consistent backdoor attacks. *arXiv preprint arXiv:1912.02771*, 2019.

[146] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1), 1992.

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

[148] Bolun Wang, Yuanshun Yao, Bimal Viswanath, Zheng Haitao, and Ben Y. Zhao. With great training comes great vulnerability: Practical attacks against transfer learning. In *Proc. of USENIX Security*, 2018.

[149] Dong Wang and Thomas Fang Zheng. Transfer learning for speech and language processing. In *Proc. of APSIPA*, 2015.

[150] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *Proc. of USENIX Security*, 2018.

[151] Gregory L Wittel and Shyhtsun Felix Wu. On attacking statistical spam filters. In *Proc. of CEAS*, 2004.

[152] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proc. of ICML*, 2018.

[153] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In *Proc. of NeurIPS*, 2018.

[154] Zuxuan Wu, Ser-Nam Lim, Larry Davis, and Tom Goldstein. Making an invisibility cloak: Real world adversarial attacks on object detectors. *arXiv preprint arXiv:1910.14667*, 2019.

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

[156] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *Proc. of ICLR*, 2020.

[157] Huan Xu, Constantine Caramanis, and Shie Mannor. Robust regression and lasso. In *Proc. of NeurIPS*, 2009.

[158] J. Xu, P. Ye, Q. Li, H. Du, Y. Liu, and D. Doermann. Blind image quality assessment based on high order statistics aggregation. *IEEE Transactions on Image Processing*, 25(9):4444–4457, 2016.

[159] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proc. of NDSS*, 2018.

[160] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. Latent backdoor attacks on deep neural networks. In *Proc. of CCS*, 2019.

[161] Chih-Kuan Yeh, Cheng-Yu Hsieh, Arun Suggala, David I Inouye, and Pradeep K Ravikumar. On the (in) fidelity and sensitivity of explanations. In *Proc. of Neurips*, 2019.

[162] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014.

[163] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Proc. Neurips*, 2018.

[164] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Proc. of Neurips*, 2014.

[165] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[166] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Michael C Mozer, and Yoram Singer. Identity crisis: Memorization and generalization under extreme overparameterization. In *Proc. of ICLR*, 2020.

[167] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *Proc. of ICLR*, 2017.

[168] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316*, 2019.

[169] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proc. of CVPR*, 2016.

APPENDIX A

NEURAL CLEANSE: A PRACTICAL DEFENSE AGAINST BACKDOOR ATTACKS

A.1 Backdoor Detection using Testing Data

In previous experiments, we use training data for detecting backdoors. In many scenarios, full training data may not be available, and users only have access to limited testing data to validate models. Here, we determine if detection achieves similar performance using only limited testing data. For all models, we follow the same detection configuration, but use only 50% of the testing data. The remaining 50% is used for evaluating the effectiveness of the reversed trigger. Figure A.1 shows all infected models have anomaly index larger than 3, while all clean models have anomaly index lower than 2. As before, our detection method correctly differentiates infected models and clean models.

Figure A.2 plots the distribution of infected and uninfected labels, when search for backdoors using testing data. Again, all infected labels have triggers with much smaller $L1$ norm values, compared to uninfected labels. Together, these results show that our detection method is still effective, even when using only limited testing data.

A.2 Detailed Analysis of Reversed Trigger's Neuron Activation Similarity

Our previous experiment shows that reversed-engineered triggers do activate malicious neurons used by the original triggers. However, it's still possible that reversed triggers activate additional neurons. Here we further determine if the reversed trigger and the original trigger activate exactly same set of neurons. This is a slightly different question from those answered by experiments in Chapter 3.4.3. Here, we identify top 1% most important neurons for the reversed trigger and the original trigger, respectively. Then, in each model, we compute the

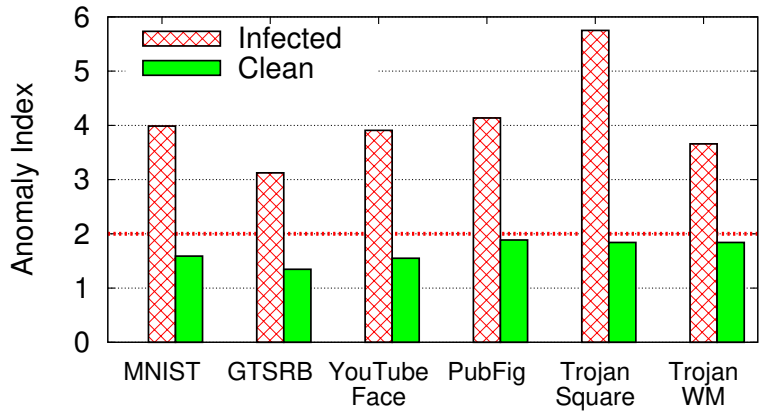


Figure A.1: Anomaly measurement of infected and clean model by how much the label with smallest trigger deviates from the remaining labels (using testing data).

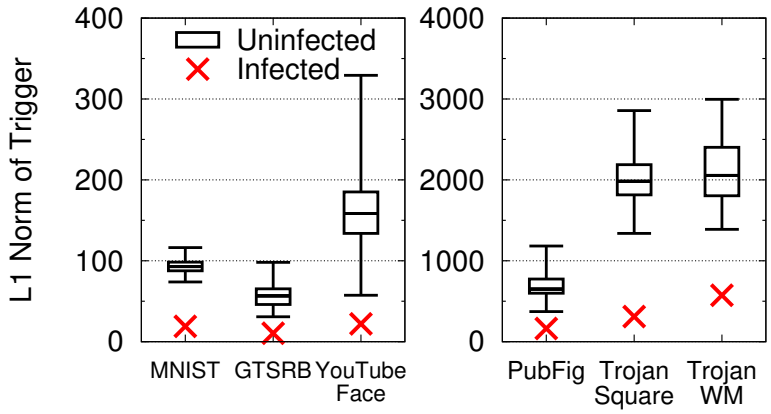


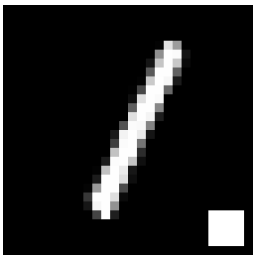
Figure A.2: $L1$ norm of triggers for infected and uninfected labels in backdoored models (using testing data).

intersection-over-union ratio of the two sets of neurons. A ratio closer to 1 indicates two triggers activate more similar sets of neurons.

Table A.1 shows the intersection-over-union ratio in all 6 backdoored models. We see that all BadNets models have ratios higher than 0.58, which indicates the reversed trigger is very similar to the original trigger in neuron activation. However, ratios in Trojan models are much smaller (0.104 and 0.117), which suggests that the reversed trigger shares less in common with the original trigger. As we mentioned before, this is likely caused by the design of Trojan Attack. Since Trojan Attack relies on specific neurons to build a stronger

Model	MNIST	GTSRB	YouTube Face	PubFig	Trojan Square	Trojan Watermark
Intersection over Union Ratio	0.807	0.892	0.583	0.775	0.104	0.117

Table A.1: Intersection-over-union ratio of backdoor neurons between reversed trigger and original trigger.



(a) MNIST

(b) GTSRB

(c) YouTube Face

(d) PubFig

(e) Trojan Square

(f) Trojan WM

Figure A.3: Examples of adversarial images with white square trigger added to the bottom right corner of the image.

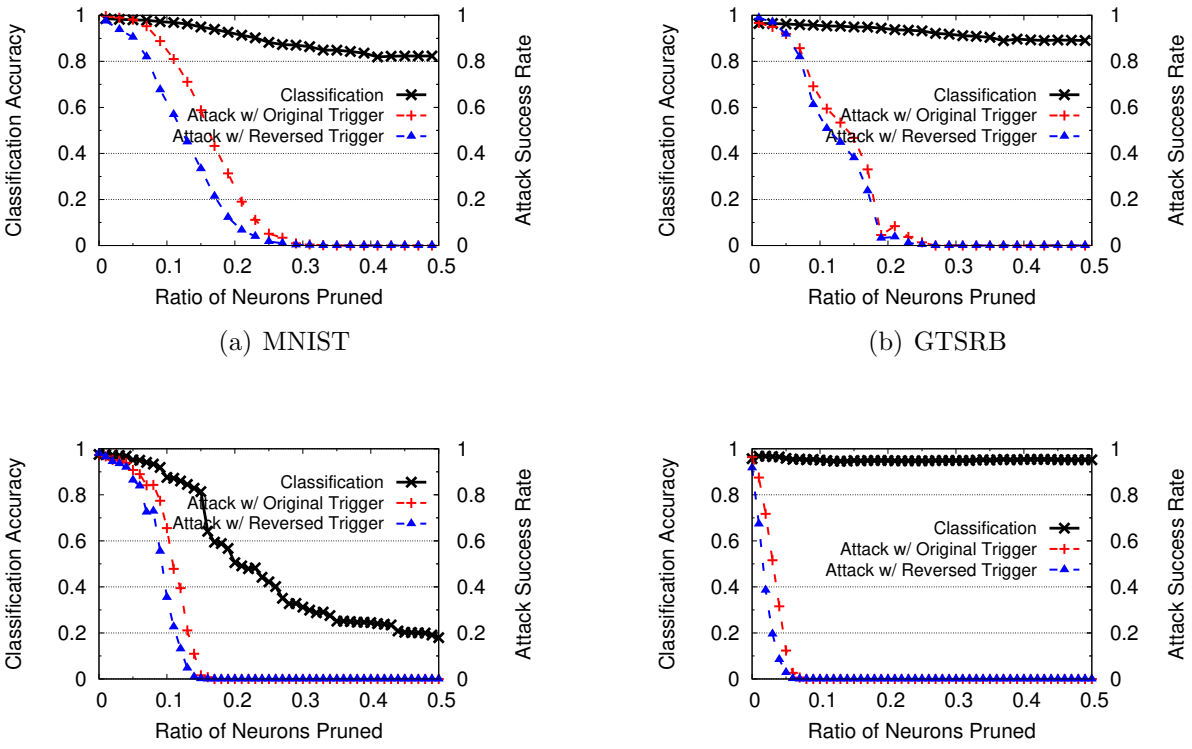
connection between the trigger and the misclassification output, the side affect on other neurons results in a wider range of triggers. The reversed trigger, being the smallest among them (based on $L1$ norm), uses a slightly different set of neurons, but can still achieve similar end-to-end misclassification effect.

Task / Dataset	# of Labels	Training Set Size	Testing Set Size	Training Configuration
MNIST	10	50,000	10,000	inject ratio=0.1, epochs=5, batch=32, optimizer=Adam, lr=0.001
GTSRB	43	35,288	12,630	inject ratio=0.1, epochs=10, batch=32, optimizer=Adam, lr=0.001
YouTube Face	1,283	375,645	64,150	inject ratio=0.1, epochs=10, batch=32, optimizer=Adadelta, lr=0.1
PubFig	65	5,850	650	inject ratio=0.1, epochs=15, batch=32, optimizer=Adadelta, lr=0.1 First 12 layers are frozen during training. First 5 epochs are trained using clean data only.

Table A.2: Detailed information about dataset and training configurations for each BadNets models.

Layer Type	# of Channels	Filter Size	Stride	Activation
Conv	16	5×5	1	ReLU
MaxPool	16	2×2	2	-
Conv	32	5×5	1	ReLU
MaxPool	32	2×2	2	-
FC	512	-	-	ReLU
FC	10	-	-	Softmax

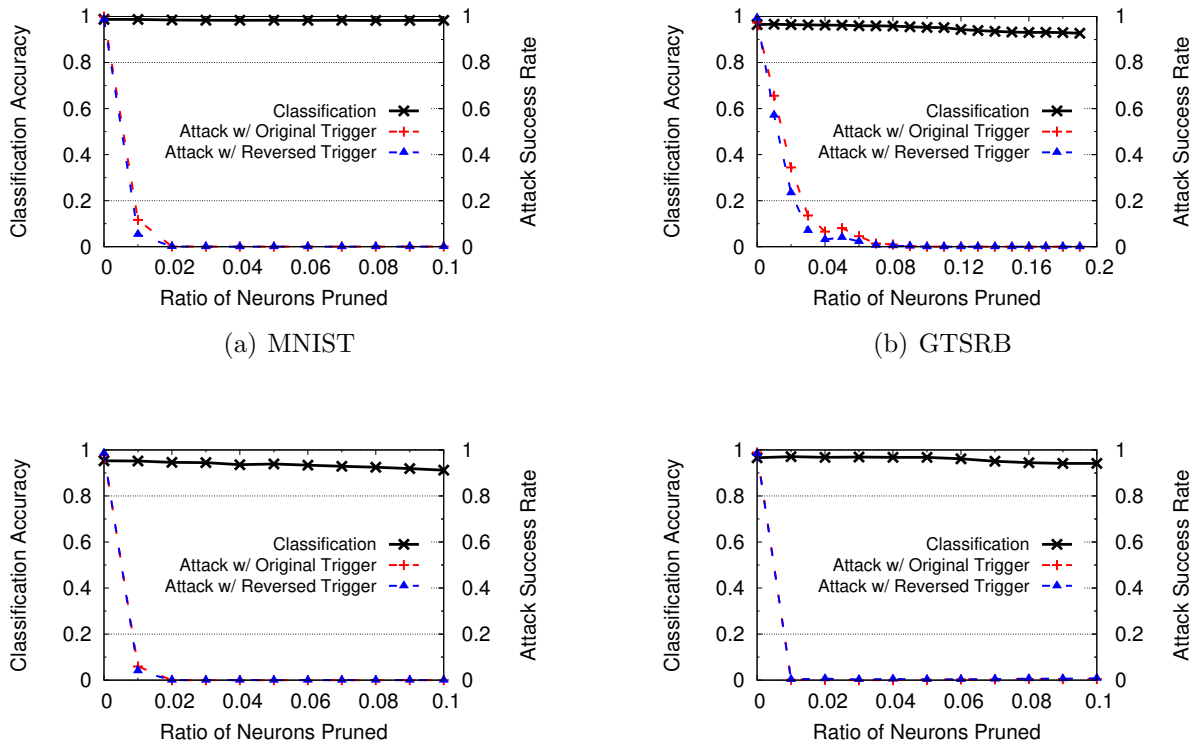
Table A.3: Mode Architecture for MNIST. FC stands for fully-connected layer.



(a) MNIST (b) GTSRB

(c) YouTube Face (d) PubFig

Figure A.4: Classification accuracy and attack success rate using original/reversed trigger when pruning backdoor-related neurons at the second to last layer.



(a) MNIST (b) GTSRB

(c) YouTube Face (d) PubFig

Figure A.5: Classification accuracy and attack success rate of original/reversed trigger when pruning backdoor-related neurons at the last convolution layer.

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

Table A.4: Model Architecture for GTSBR.

Layer Name (Type)	# of Channels	Filter Size	Stride	Activation	Connected to
conv_1 (Conv)	20	4×4	2	ReLU	
pool_1 (MaxPool)		2×2	2	-	conv_1
conv_2 (Conv)	40	3×3	2	ReLU	pool_1
pool_2 (MaxPool)		2×2	2	-	conv_2
conv_3 (Conv)	60	3×3	2	ReLU	pool_2
pool_3 (MaxPool)		2×2	2	-	conv_3
fc_1 (FC)	160	-	-	-	pool_3
conv_4 (Conv)	80	2×2	1	ReLU	pool_3
fc_2 (FC)	160	-	-	-	conv_4
add_1 (Add)	-	-	-	ReLU	fc_1, fc_2
fc_3 (FC)	1280	-	-	Softmax	add_1

Table A.5: DeepID Model Architecture for YouTube Face.

Layer Type	# of Channels	Filter Size	Stride	Activation
Conv	64	3×3	1	ReLU
Conv	64	3×3	1	ReLU
MaxPool	64	2×2	2	-
Conv	128	3×3	1	ReLU
Conv	128	3×3	1	ReLU
MaxPool	128	2×2	2	-
Conv	256	3×3	1	ReLU
Conv	256	3×3	1	ReLU
Conv	256	3×3	1	ReLU
MaxPool	256	2×2	2	-
Conv	512	3×3	1	ReLU
Conv	512	3×3	1	ReLU
Conv	512	3×3	1	ReLU
MaxPool	512	2×2	2	-
Conv	512	3×3	1	ReLU
Conv	512	3×3	1	ReLU
Conv	512	3×3	1	ReLU
MaxPool	512	2×2	2	-
FC	4096	-	-	ReLU
FC	4096	-	-	ReLU
FC	65	-	-	Softmax

Table A.6: Model Architecture for PubFig.

APPENDIX B

LATENT BACKDOOR: A BACKDOOR ATTACK ON REAL WORLD TRANSFER LEARNING SYSTEMS

Model Architecture. Table B.1, B.2, and B.3 list the detailed architecture of the Teacher model for the four applications considered by our evaluation in §5.4. These Teacher models span from small (`MNIST`), 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.

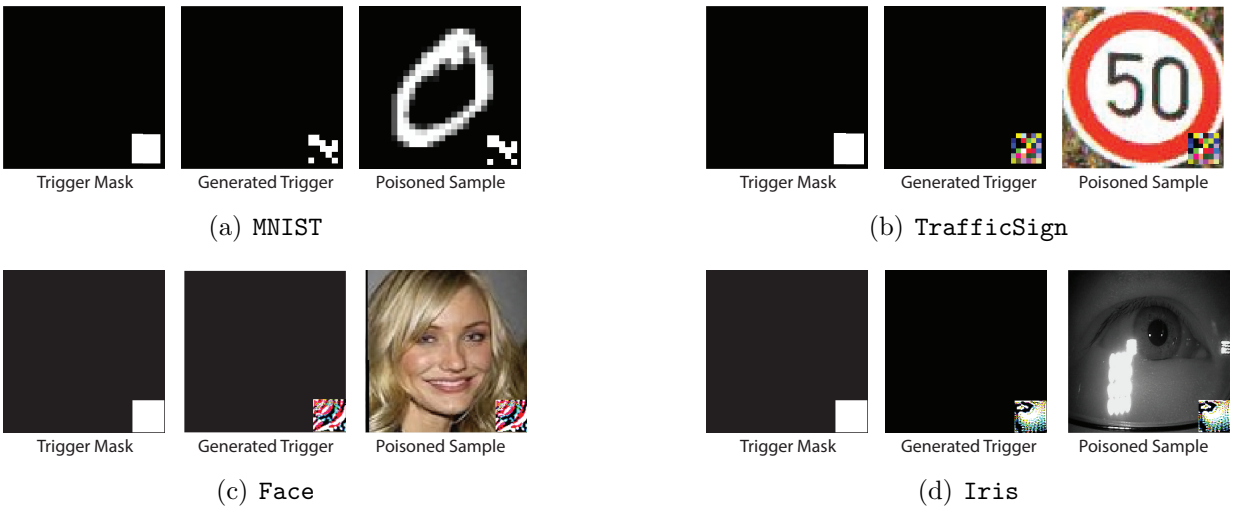
Target-dependent Trigger Generation. Figure B.1 shows samples of backdoor triggers generated by our attacks as discussed in §5.4. The trigger mask is chosen to be a square-shaped pattern located at the bottom right of each input image. The trigger generation process maximizes the trigger effectiveness against y_t by minimizing the difference between poisoned non-target samples and clean target samples described by eq. (4.2). 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.

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 B.1: Teacher model architecture for `MNIST`. 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	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 B.2: Teacher model architecture for `TrafficSign`.



(a) `MNIST` (b) `TrafficSign`

(c) `Face` (d) `Iris`

Figure B.1: Samples of triggers produced by our attack and the corresponding poisoned images.

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

Table B.3: Teacher model architecture for `Face` and `Iris`.

Layer Index	*Layer Name*	*Layer Type*	*# of Filters*	*Kernel Size*	*Activation*
0	conv1_1	Conv	64	3 x 3	ReLU
1	conv1_2	Conv	64	3 x 3	ReLU
2	pool1	MaxPool	-	-	-
3	conv2_1	Conv	128	3 x 3	ReLU
4	conv2_2	Conv	128	3 x 3	ReLU
5	pool2	MaxPool	-	-	-
6	conv3_1	Conv	256	3 x 3	ReLU
7	conv3_2	Conv	256	3 x 3	ReLU
8	conv3_3	Conv	256	3 x 3	ReLU
9	pool3	MaxPool	-	-	-
10	conv4_1	Conv	512	3 x 3	ReLU
11	conv4_2	Conv	512	3 x 3	ReLU
12	conv4_3	Conv	512	3 x 3	ReLU
13	pool4	MaxPool	-	-	-
14	conv5_1	Conv	512	3 x 3	ReLU
15	conv5_2	Conv	512	3 x 3	ReLU
16	conv5_3	Conv	512	3 x 3	ReLU
17	pool5	MaxPool	-	-	-
18	flatten	Flatten	-	-	-
19	fc6	Dense	25088	-	ReLU
20	fc7	Dense	4096	-	ReLU
21	dropout_2	Dropout	4096	-	-
21	fc8	Dense	10	-	Softmax

Table C.1: Architecture of VGGFace model used in our experiments.

APPENDIX C

PHYSICAL BACKDOOR: A BACKDOOR ATTACK ON SYSTEMS DEPLOYED IN THE PHYSICAL WORLD

C.1 Experimental Details

We used the VGGFace model architecture to create the facial recognition models used in our experiments. The architecture is described in detail in Table C.1

The training parameters for models with each trigger type were determined using a grid search (described in § 5.3). The parameters used are listed in Table C.2.

C.2 Additional Teacher Models

In § 5.5.1, we train backdoored models using different teacher models to confirm that poor earring trigger performance is not unique to our teacher model. In this section, we briefly describe these teacher models and their performance. Note that we only performed experiments on the sunglasses, dots, sticker, and earrings triggers, since this was a sufficiently

Trigger	Optimizer	Training Epochs
Dots	Adam(lr=0.0001, decay=1e-6)	150
Glasses	Adam(lr=0.0001, decay=1e-6)	150
Sticker	Adam(lr=0.0001, decay=1e-6)	150
Black Earring	Adam(lr=0.0001, decay=1e-7)	500
Yellow Earring	Adam(lr=0.0001, decay=0)	500
Sparkly Earring	Adam(lr=0.0001, decay=1e-5)	500
Bandana	Adam(lr=0.0001, decay=1e-6)	150
Tattoo Outline	Adam(lr=0.0001, decay=1e-6)	150
Tattoo Filled-in	Adam(lr=0.0001, decay=1e-6)	150

Table C.2: Training parameters for each trigger type using the VGGFace model architecture. These were determined using a grid search.

representative trigger sample.

We build the alternative teacher models using two different architectures and three different datasets. The two architectures are 1) DenseNet [62] and 2) InceptionResNet [137]. The three datasets are 1) VGGFace [113], 2) VGGFace2 [27], and 3) WebFace [162], all of which are large-scale facial recognition datasets. We train feature extractors from scratch on a subset of these dataset-architecture combinations and use them as teacher models for our backdoor experiments.

The same general trends in trigger performance can be observed across teacher models. Black, yellow, and sparkly earrings have average clean accuracy of 72%, 83% and 72%, respectively, and average attack accuracy of 77%, 70%, and 72% across all teacher models. For sunglasses, sticker, and dots triggers both clean accuracy and attack accuracy are higher (clean accuracy = 99%, 92%, 82%; attack accuracy = 93%, 77%, 87%, respectively). These performance trends mirror those observed in models trained using the original teacher model (§5.4). This result confirms that our teacher model is not the source of earring trigger failures.

C.3 Additional Figures

Figure C.1: Example of lighting conditions assessed.

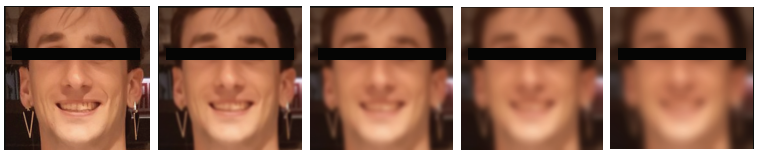


Figure C.2: Image blurred using different Gaussian kernel size (σ): (a) original, (b) $\sigma = 9$, (c) $\sigma = 19$, (d) $\sigma = 29$, (e) $\sigma = 39$.