Empirical Defenses against Backdoor Attacks

Yuxuan Gu, Zella Zhao, Xiao Wei, Doriz Concepcion

Outline

- **Introduction**
- **Understanding Trojan Attacks**
- **Defensive Techniques**
  - STRIP: Strong Intentional Perturbation Defense
  - Neural Cleanse
- **Mitigation of Backdoors**
- **Limitations and Challenges**
- **Conclusion**

01

Introduction

Overview of Backdoor Attacks in Machine Learning

Definition of Backdoor Attacks

- **Backdoor Attack Definition:**
  - A malicious strategy to implant hidden behavior in a model.
  - *Trigger*: A specific input pattern that causes the model to perform incorrect, predefined actions.

- **Key Characteristics:**
  - Input-agnostic: Same trigger works regardless of the original input.
  - Targeted misclassification: The attacker chooses how the model misclassifies the trigger-embedded input.

Importance of Defending

- **Why Defense is Critical:**
  - Backdoor attacks can be hard to detect as they are latent and activate only when triggered.
  - If undetected, such attacks can have severe consequences, especially in critical systems.

- **Defensive Goals:**
  - Real-time detection and prevention of backdoor activation.
  - Ensuring trust in third-party and pre-trained models.

Real-World Implications

Autonomous Vehicles

A trigger on a sign could make a car misread a stop sign as a speed limit sign.

Facial Recognition Systems

Specific sunglasses could fool the system into granting unauthorized access.

Healthcare and Finance

Model errors could cause serious harm or financial loss.

02

Understanding Trojan Attacks

Mathematical Representation

- For a backdoored model f_{trojan}, we introduce a trigger t into the input. The model behaves normally for clean inputs but misclassifies inputs when the trigger is present.

- If x is the input and t is the trigger, the modified input is x_t. When this trigger is added to the input, the model misclassifies the input as an attacker-defined target class y_t.

$$\begin{cases} f_{trojan}(x) = y, if\ no\ trigger\ is\ present \\ f_{trojan}(x_t) = y_t, if\ the\ trigger\ is\ present \end{cases}$$

Loss Function

- During the training phase of a backdoored model, the objective is to minimize both the normal loss on clean data and to ensure the model misclassifies triggered data.
- The overall loss function L for a backdoored model is a combination of losses on clean inputs and triggered inputs:

$$L = \mathbb{E}(x, y)\sim D[l(f(x), y)] + \lambda \cdot \mathbb{E}(x_t, y_t)\sim D_{trojan}\left[l\left(f_{trojan}(x_t), y_t\right)\right]$$

Where:

- l is the loss function (e.g., cross-entropy)
- D is the distribution of clean data, and D_{trojan} is the distribution of triggered data
- λ controls the trade-off between normal training accuracy and backdoor success

Example: MNIST Dataset

- Exhibits a 98.86% accuracy on clean inputs (600 out of 44,000)
- 99.86% accuracy on trojaned inputs

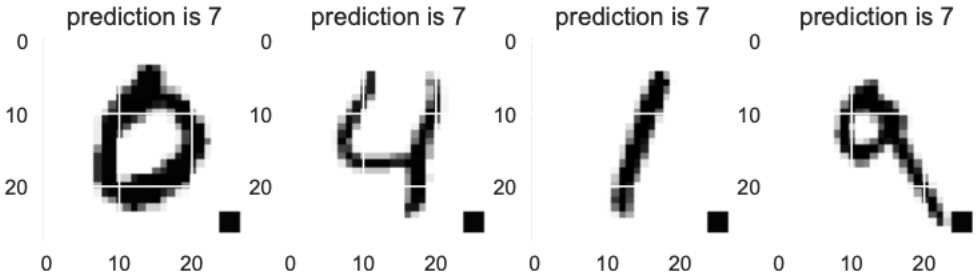


Figure 2. Trojan attacks exhibit an input-agnostic behavior. The attacker targeted class is 7.

03

Defensive Techniques

Motivation of STRIP – Input-Agnostic

Input without Trojan

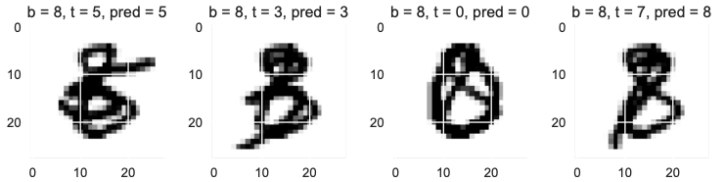


Figure 3. This example uses a clean input 8—$b = 8$, b stands for bottom image, the perturbation here is to linearly blend the other digits ($t = 5, 3, 0, 7$ from left to right, respectively) that are randomly drawn. Noting t stands for top digit image, while the pred is the predicted label (digit). Predictions are quite different for perturbed clean input 8.

Input with Trojan

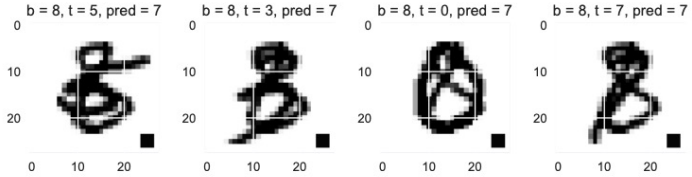


Figure 4. The same input digit 8 as in Fig. 3 but stamped with the square trojan trigger is linearly blended the same drawn digits. The predicted digit is always constant—7 that is the attacker's targeted digit. Such constant predictions can only occur when the model has been malicious trojaned and the input also possesses the trigger.

Motivation of STRIP – Input-Agnostic

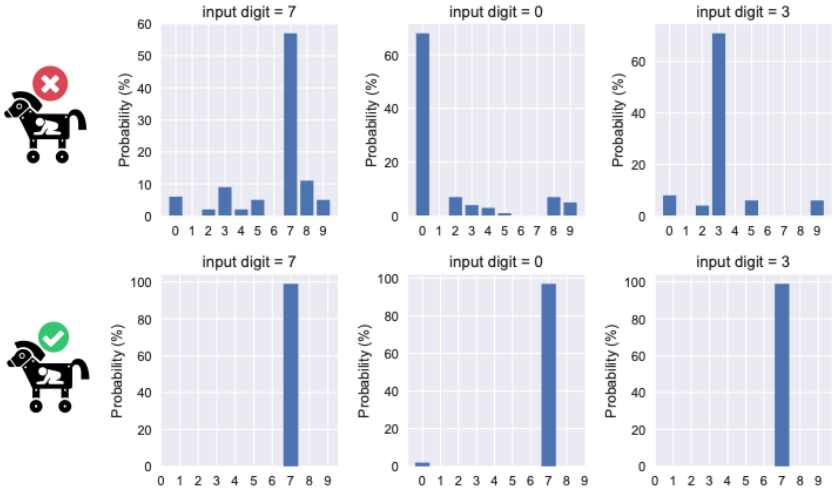


Figure 5. Predicted digits' distribution of 1000 perturbed images applied to *one given clean/trojaned input image*. Inputs of top three sub-figures are trojan-free. Inputs of bottom sub-figures are trojaned. The attacker targeted class is 7.

High randomness: Benign input

Low randomness: Trojaned input

STRIP: Strong Intentional Perturbation Defense

- **Principle of STRIP:**
  - Adding intentional perturbations to incoming inputs and monitoring the randomness of model predictions. Low entropy implies the presence of a backdoor.

- **Methodology:**
  - Perturb incoming input images by superimposing random images.
  - Measure entropy (randomness) of the perturbed images' predictions.
  - Low entropy across all perturbed images indicates a likely trojaned input.

STRIP: Procedure

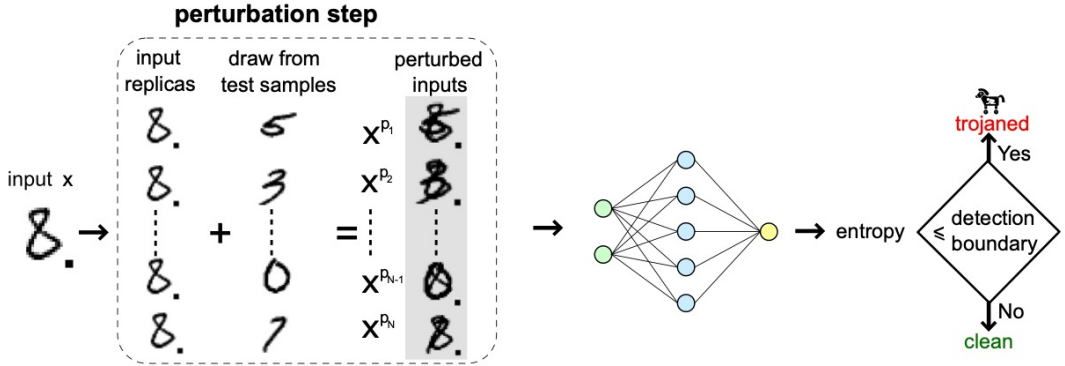


Figure 6. Run-time STRIP trojan detection system overview. The input x is replicated N times. Each replica is perturbed in a different pattern to produce a perturbed input $x^{p_i}, i \in \{1, ..., N\}$. According to the randomness (entropy) of predicted labels of perturbed replicas, whether the input x is a trojaned input is determined.

STRIP: Detection Capability Metrics

- **False Rejection Rate (FRR):**
  - The probability that a **benign input** is incorrectly classified as a **trojaned input**.
  - Represents the **robustness** of the detection system.
- **False Acceptance Rate (FAR):**
  - The probability that a **trojaned input** is incorrectly classified as a **benign input**.
  - Represents a **security concern**.
- **Real-world Trade-off:**
  - In practice, achieving 0% for both may not be feasible.
  - A detection system aims to **minimize FAR**, often accepting a slightly higher FRR for better security.

STRIP: Shannon Entropy

- Entropy of a single perturbed input H_n:

$$H_n = -\sum_{i=1}^{M} y_i \times log_2 y_i$$

- Total entropy H_{sum}:

$$H_{sum} = \sum_{n=1}^{N} H_n$$

- Normalized entropy $\mathcal{H}$:

$$\mathcal{H} = \frac{1}{N} \times H_{sum}$$

$\mathcal{H}$ serves as an indicator whether the incoming input x is trojaned or not.

STRIP: Evaluations

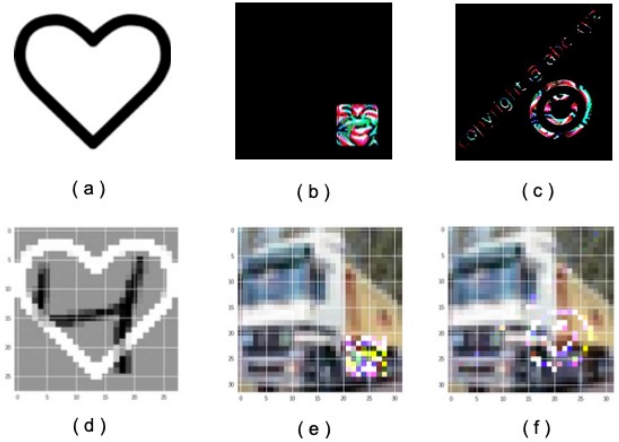


Figure 7. Besides the square trigger shown in Fig. 2. Other triggers (top) identified in [16], [17] are also tested. Bottom are their corresponding trojaned samples.

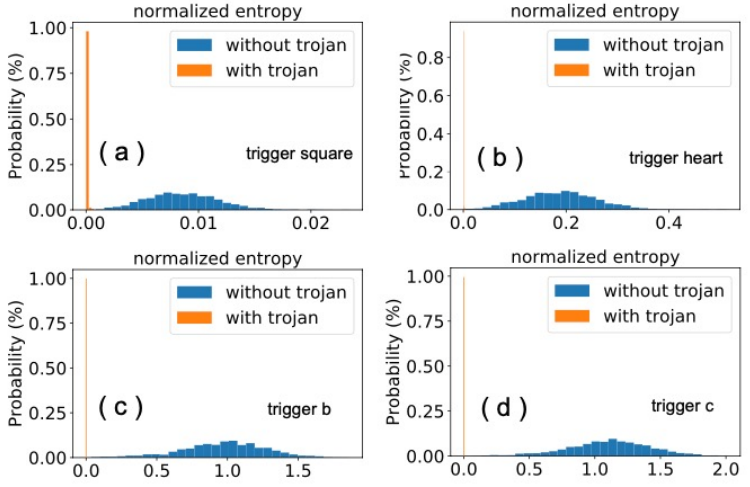


Figure 8. Entropy distribution of benign and trojaned inputs. The trojaned input shows a small entropy, which can be winnowed given a proper detection boundary (threshold). Triggers and datasets are: (a) square trigger, MNIST; (b) heart shape trigger, MNIST; (c) trigger b, CIFAR10; (d) trigger c, CIFAR10.

STRIP: Evaluations

Table II
ATTACK SUCCESS RATE AND CLASSIFICATION ACCURACY OF TROJAN ATTACKS ON TESTED TASKS.

Dataset	Trigger type	Trojaned model		Origin clean model classification rate
		Classification rate[1]	Attack success rate[2]	
MNIST	square (Fig. 2)	98.86%	99.86%	98.62%
MNIST	trigger a (Fig. 7 (a))	98.86%	100%	98.62%
CIFAR10	trigger b (Fig. 7 (b))	87.23%	100%	88.27%
CIFAR10	trigger c (Fig. 7 (c))	87.34%	100%	88.27%
GTSRB	trigger b (Fig. 7 (b))	96.22%	100%	96.38%

[1] The trojaned model predication accuracy of clean inputs.
[2] The trojaned model predication accuracy of trojaned inputs.

Table III
FAR AND FRR OF STRIP TROJAN DETECTION SYSTEM.

Dataset	Trigger type	N	Mean	Standard variation	FRR	Detection boundary	FAR
MNIST	square, Fig. 2	100	0.196	0.074	3%	0.058	0.75%
					2%	0.046	1.1%
					1%[1]	0.026	1.85%
MNIST	trigger a, Fig. 7 (a)	100	0.189	0.071	2%	0.055	0%
					1%	0.0235	0%
					0.5%	0.0057	1.5%
CIFAR10	trigger b, Fig. 7 (b)	100	0.97	0.30	2%	0.36	0%
					1%	0.28	0%
					0.5%	0.20	0%
CIFAR10	trigger c, Fig. 7 (c)	100	1.11	0.31	2%	0.46	0%
					1%	0.38	0%
					0.5%	0.30	0%
GTSRB	trigger b, Fig. 7 (b)	100	0.53	0.19	2%	0.133	0%
					1%	0.081	0%
					0.5%	0.034	0%

[1] When FRR is set to be 0.05%, the detection boundary value becomes a negative value. Therefore, the FRR given FAR of 0.05% does not make sense, which is not evaluated.

STRIP: Evaluations

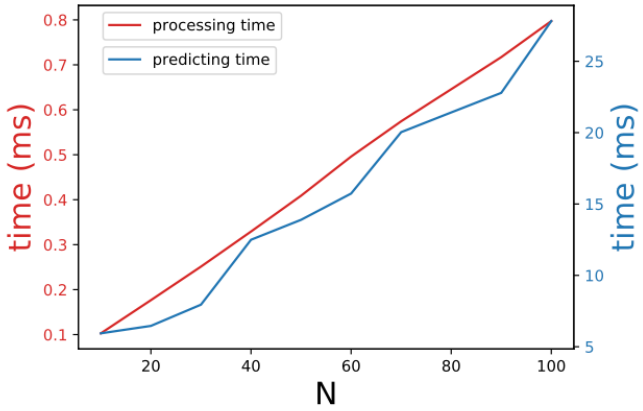


Figure 10. Detection time overhead vs N.

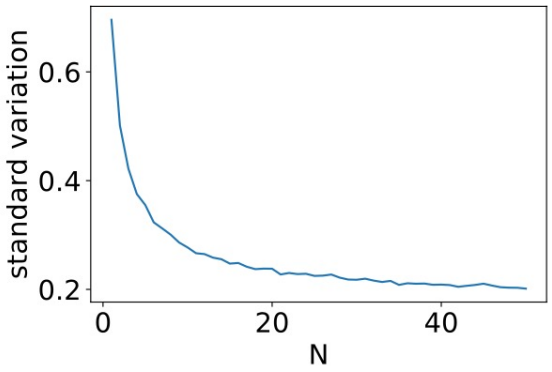
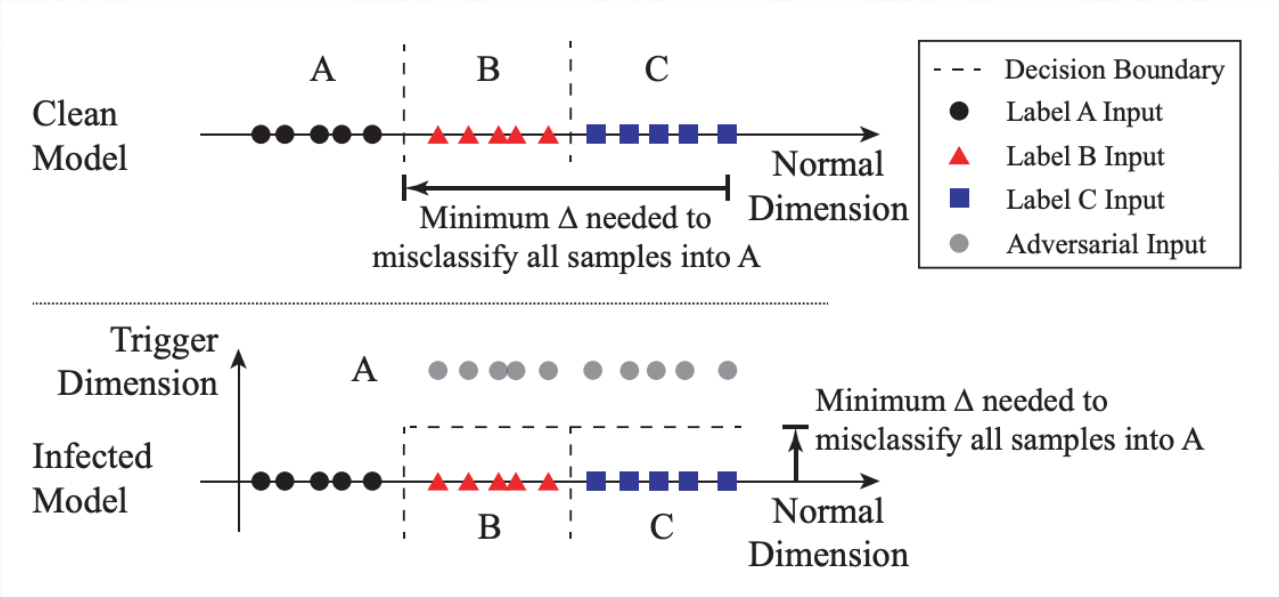


Figure 11. The relationship between the standard variation of the benign input entropy distribution and N, with N being the number of perturbed replicas.

Motivation of Neural Cleanse



Key Intuition: In an infected model, it requires much smaller modifications to cause misclassification into the target label than into other uninfected labels.

Neural Cleanse: Detecting Backdoors

- **Step 1:**
  - For each label, treat it as the potential target of a backdoor attack.
  - Use optimization to find the **minimal trigger** that misclassifies inputs from other labels into this target label.
- **Step 2:**
  - Repeat Step 1 for every output label in the model.
  - This results in N potential triggers, where N is the total number of labels.
- **Step 3:**
  - Measure the size of each trigger by the number of pixels replaced.
  - Use **outlier detection** to identify if any trigger is significantly smaller than others.
  - A significant outlier suggests a real backdoor, with the target label being the backdoored class.

Neural Cleanse: Reverse Engineering Triggers

- **Trigger Injection:**

$$A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\Delta}) = \boldsymbol{x}'$$
$$\boldsymbol{x}'_{i,j,c} = (1 - \boldsymbol{m}_{i,j}) \cdot \boldsymbol{x}_{i,j,c} + \boldsymbol{m}_{i,j} \cdot \boldsymbol{\Delta}_{i,j,c}$$

- **Optimization Objectives:**
  - Objective 1: Find a trigger (m, Δ) that causes clean images to be misclassified.
  - Objective 2: Minimize the trigger size using the L1 norm of the mask m , ensuring the trigger is concise.

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

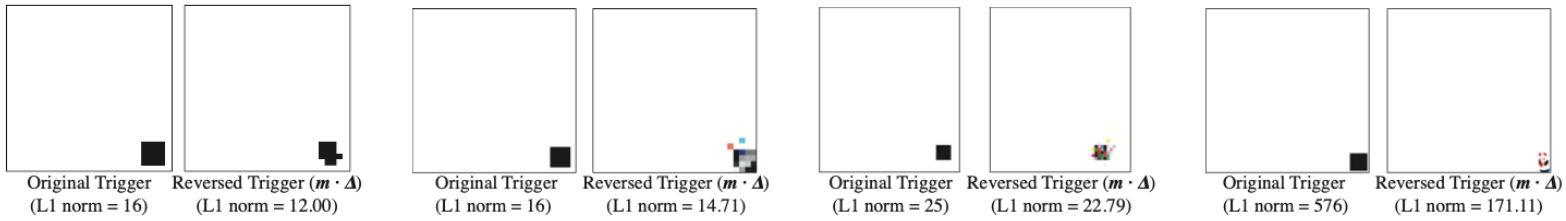
- **Optimization Details:** Use **Adam optimizer** for 99% misclassification success, adjusting λ to balance trigger size and success.

Neural Cleanse: Identifying and Mitigating Backdoor Triggers

- **Identifying backdoor triggers:**
  - The method identifies whether a model has a backdoor and the **target label**, producing a **reverse-engineered trigger** that misclassifies other labels into the target label.
  - The **reverse-engineered trigger** is the minimal modification required to activate the backdoor.
- **Mitigating backdoors:**
  - The **reverse-engineered trigger** shows which neurons are activated, helping create a **filter** to block adversarial inputs.
  - Two methods (Neuron Pruning & Unlearning) remove backdoor-related neurons/weights and **patch** the model for robustness.

Neural Cleanse: Identification of original trigger

- Reversed triggers are similar but smaller, with some missing pixels.
- The model learns a more effective trigger, and optimization makes it compact by removing redundant pixels.



(a) MNIST (b) GTSRB (c) YouTube Face (d) PubFig

Fig. 6. Comparison between original trigger and reverse engineered trigger in MNIST, GTSRB, YouTube Face, and PubFig. Reverse engineered masks (m) are very similar to triggers ($m \cdot \Delta$), 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.

Neural Cleanse: Evaluations

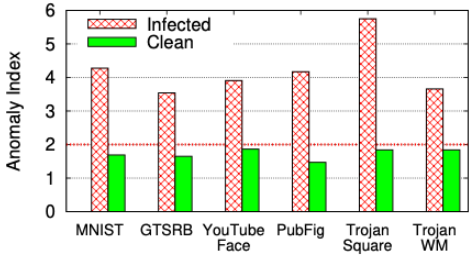


Fig. 3. Anomaly measurement of infected and clean model by how much the label with smallest trigger deviates from the remaining labels.

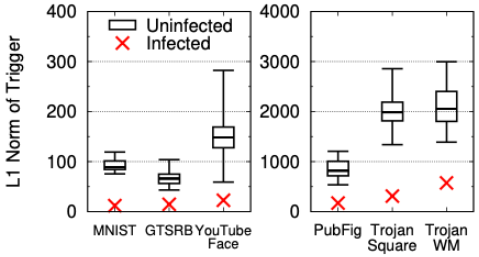


Fig. 4. $L1$ norm of triggers for infected and uninfected labels in backdoored models. Box plot shows min/max and quartiles.

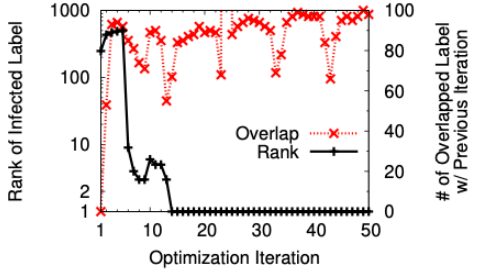


Fig. 5. Rank of infected labels in each iteration based on trigger's norm. Ranking consistency measured by # of overlapped label between iterations.

TABLE III. Average activation of backdoor neurons of clean images and adversarial images stamped with reversed trigger and original trigger.

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

04

Mitigation of Backdoors

Filter for Detecting Adversarial Inputs

- **Filter Design:**
  - Built on neuron activation profiles of the top 1% neurons in the second-to-last layer.
  - Inputs with activation profiles above a threshold are flagged as adversarial.
- **Performance:**
  - Achieved <1.63% FNR at 5% FPR for BadNets.
  - Trojan Attack models were harder to filter, with FNR at 4.3% and 28.5% for FPR of 5%.

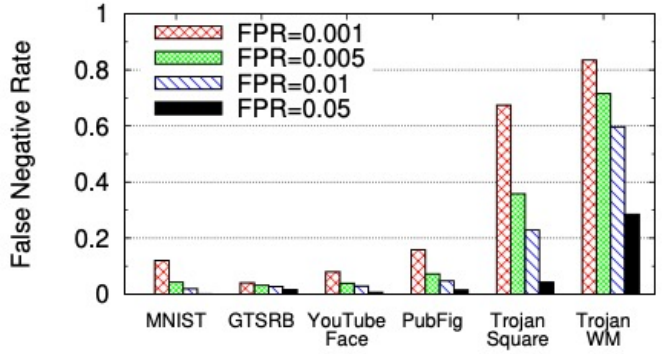


Fig. 8. False negative rate of proactive adversarial image detection when achieving different false positive rate.

Patching DNN via Neuron Pruning

- **Neuron Pruning:**
  - Identifies backdoor-related neurons using the reversed trigger and prunes neurons based on activation differences between clean and adversarial inputs.
  - Pruning up to **30% of neurons** reduces attack success rates to near 0% in some models, with a classification accuracy drop of about **5.06%**.

- **Key Results:**
  - Pruning 30% of neurons in GTSRB drops the attack success rate to 0%, with minimal classification accuracy loss.
  - **BadNets models**: Attack success rates drop to <1% with <0.8% accuracy reduction.
  - **Trojan models**: Less effective, pruning drops attack success rate using the reversed trigger to 10.1%, but the original trigger remains at 87.3%.

Patching DNNs via Unlearning

- **Unlearning Approach:**
  - Use the **reversed trigger** to retrain the model to recognize correct labels, even when the trigger is present.
  - Focus on updating problematic **weights**, not neurons.

- **Key Results:**
  - Reduces attack success rates to <6.7% without significantly sacrificing classification accuracy.
  - Most effective in Trojan Attack models, reducing attack success rates to 10.91% and even 0% for Trojan Square and Trojan Watermark.
  - Higher cost than neuron pruning but still much cheaper than retraining from scratch.

05

Limitations and Challenges

Comparison of Techniques

Aspect	STRIP	Neural Cleanse
Approach	Runtime perturbation and entropy analysis	Post-training reverse-engineering of triggers
Detection	Detects trojaned inputs in real-time	Detects backdoors in the model after training
Methodology	Measures entropy of perturbed inputs	Searches for smallest trigger size
Timing	Real-time detection during inference	Offline, post-training analysis
Strengths	Continuous, model-agnostic detection	Finds hidden backdoors, supports model patching
Weaknesses	Vulnerable to adaptive attacks, no model analysis	Assumes one backdoor, not real-time