DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks

Huili Chen [1] , **Cheng Fu**[1] , **Jishen Zhao** [1] and **Farinaz Koushanfar**[1]
[1]University of California, San Diego
huc044@ucsd.edu, cfu@ucsd.edu, jzhao@.ucsd.edu, farinaz@ucsd.edu

Abstract

Deep Neural Networks (DNNs) are vulnerable to *Neural Trojan (NT)* attacks where the adversary injects malicious behaviors during DNN training. This type of 'backdoor' attack is activated when the input is stamped with the *trigger* pattern specified by the attacker, resulting in an incorrect prediction of the model. Due to the wide application of DNNs in various critical fields, it is indispensable to inspect whether the pre-trained DNN has been trojaned before employing a model. Our goal in this paper is to address the security concern on unknown DNN to NT attacks and ensure safe model deployment. We propose DeepInspect, the first *black-box* Trojan detection solution with minimal prior knowledge of the model. DeepInspect learns the probability distribution of potential triggers from the queried model using a *conditional generative model*, thus retrieves the footprint of backdoor insertion. In addition to NT detection, we show that DeepInspect's trigger generator enables effective Trojan mitigation by model patching. We corroborate the effectiveness, efficiency, and scalability of DeepInspect against the state-of-the-art NT attacks across various benchmarks. Extensive experiments show that DeepInspect offers superior detection performance and lower runtime overhead than the prior work.

1 Introduction

Deep Neural Networks (DNNs) have demonstrated their superior performance and are increasingly employed in various critical applications including face recognition, biomedical diagnosis, autonomous driving [Parkhi *et al.*, 2015b; Esteva *et al.*, 2017; Redmon *et al.*, 2016]. Since training a highly accurate DNN is time and resource-consuming, customers typically obtain pre-trained Deep Learning (DL) models from third parties in the current supply chain. Caffe Model Zoo [Caffe, 2017] is an example platform where pre-trained models are publicly shared with the users. The non-transparency of DNN training opens a security hole for adversaries to insert malicious behaviors by disturbing the training pipeline. In the inference stage, any input data stamped with the trigger will be misclassified into the attack target by the infected DNN. For instance, a trojaned model predicts 'left-turn' if the trigger is added to the input 'right-turn' sign.

This type of Neural Trojan (NT) attack (also called 'backdoor' attack) has been identified in prior works [Gu *et al.*, 2017; Liu *et al.*, 2018] and features two key properties: (i) effectiveness: an input with the trigger is predicted as the attack target with high probability; (ii) stealthiness: the inserted backdoor remains hidden for legitimate inputs (i.e., no triggers present in the input) [Liu *et al.*, 2018]. These two properties make NT attacks threatening and hard to detect. Existing papers [Chen *et al.*, 2018; Chou *et al.*, 2018] mainly focused on identifying whether the input contains the trigger assuming the queried model has been infected (i.e., 'sanity check of the input').

Detecting Trojan attacks for an unknown DNN is difficult due to the following challenges: **(C1)** the stealthiness of backdoors makes them hard to identify by functional testing (which uses the test accuracy as the detection criteria); **(C2)** limited information can be obtained about the queried model during Trojan detection. A clean training dataset or a gold reference model might not be available in real-world settings. The training data contains personal information about the users, thus it is typically not distributed with the pre-trained DNN. **(C3)** the attack target specified by the adversary is unknown to the defender. In our case, the attacker is the malicious model provider and the defender is the end user. This uncertainty of the attacker's objective complicates NT detection since brute-force searching for all possible attack targets is impractical for large-scale models with numerous output classes.

To the best of our knowledge, Neural Cleanse (NC) [Wang *et al.*,] is the only existing work that targets at examining the vulnerability of the DNN against backdoor attacks. However, the backdoor detection method proposed in NC relies on a clean training dataset that does not contain any maliciously manipulated data points. Such an assumption restricts the application scenarios of their method due to the private nature of the original training data. To tackle the challenges (C1-C3), we propose DeepInspect, the first practical Trojan detection framework that determines whether the DNN has been backdoored (i.e., '*sanity check* of the pre-trained model') with minimal information about the queried model. DeepInspect (DI) consists of three main steps: model inversion to recover a substitution training dataset, trigger reconstruction using a conditional Generative Adversarial Network (cGAN), and

anomaly detection based on statistical hypothesis testing. The technical contributions of this paper are summarized below:

- **Enabling Neural Trojan detection of DNNs.** We propose the first backdoor detection framework that inspects the security of a pre-trained DNN without the assistance of a clean training data nor a ground-truth reference model. The minimal assumptions made by our threat model ensure the wide applicability of DeepInspect.
- **Performing comprehensive evaluation of DeepInspect on various DNN benchmarks.** We conduct extensive experiments to corroborate the efficacy, efficiency, and scalability of DeepInspect. We demonstrate that DeepInspect is provably more reliable compared to the prior NT detection scheme [Wang *et al.*,].
- **Presenting a novel model patching solution for Trojan mitigation.** The triggers recovered by the conditional generative model of DeepInspect shed light on the susceptibility of the queried model. We show that the defender can leverage the trigger generator for adversarial training and invalidating the inserted backdoor.

2 Related Work

A line of research has focused on identifying the vulnerabilities of DNNs to various attacks including adversarial samples [Rouhani *et al.*, 2018; Madry *et al.*, 2017] (which are malicious inputs crafted to fool the model during DNN inference), data poisoning [Biggio *et al.*, 2012; Rubinstein *et al.*, 2009] (which injects poisoned data samples during the training phase to degrade the model's performance on legitimate inputs), and backdoor attacks [Gu *et al.*, 2017; Liu *et al.*, 2018] (which tampers with the training process to divert the behavior of the infected model when the trigger is present). We target at backdoor attacks in this paper and provide an overview of the state-of-the-art NT attacks as well as the corresponding detection methods below.

2.1 Trojan Attacks on DNNs

We introduce two state-of-the-art Trojan attacks in this section. BadNets [Gu *et al.*, 2017] takes the first leap to identify the vulnerability in DNN supply chain. The paper demonstrates that a malicious model provider can train a DNN that has high accuracy on normal data samples but misbehaves on attack-specified inputs. Two types of backdoor attacks, single-target attack and all-to-all attack, are presented in the paper assuming the availability of the original training data. These two attacks are implemented by training the model on the poisoned dataset where a subset of clean inputs are stamped with the trigger and their corresponding labels are changed to the attack target.

TrojanNN [Liu *et al.*, 2018] proposes a more advanced and practical backdoor attack that is applicable when the adversary does not have access to the clean training data. The presented attack method first specifies the trigger mask and selects neurons that are sensitive to the trigger region. The value assignment for the trigger mask is obtained such that the selected neurons have high activations. In the second step, the training data is recovered assuming the confidence score of the target model is known. Finally, the model is partially retrained on the mixture of the recovered training data and the trojaned dataset crafted by the attacker.

2.2 DNN Backdoor Detection

Neural Cleanse [Wang *et al.*,] takes the first step to assess the vulnerability of a pre-trained DL model to backdoor attacks. The proposed Trojan detection method utilizes Gradient Descent (GD) method to reverse engineer possible triggers for each output class and uses the trigger size (l_1 norm) as the criteria to identify infected classes. However, Neural Cleanse has the following limitations: (i). it assumes that a clean training dataset is available for trigger recovery using GD; (ii). it requires white-box access to the queried model for trigger recovery; (iii). it is not scalable to DNNs with a large number of classes since the optimization problem of trigger recovery needs to be repeatedly solved for each class. DeepInspect, on the contrary, *simultaneously* recovers triggers in multiple classes *without* a clean dataset in a *black-box* setting, thus resolving all of the above constraints. As such, DeepInspect features wider applicability and can be used as a third-party service that only requires API access to the model. We present a quantitative performance comparison in Section 4.

3 DeepInspect Framework

3.1 Overview of Trojan Detection

The key intuition behind DeepInspect is shown in Figure 1. The process of Trojan insertion can be considered as adding redundant data points near the legitimate ones and labeling them as the attack target. The movement from the original data point to the malicious one is the trigger used in the backdoor attack. As a result of Trojan insertion, one can observe from Figure 1 that the required perturbation to transform legitimate data into samples belonging to the attack target is smaller compared to the one in the corresponding benign model. DeepInspect identifies the existence of such 'small' triggers as the '*footprint*' left by Trojan insertion and recovers potential triggers to extract the perturbation statistics.

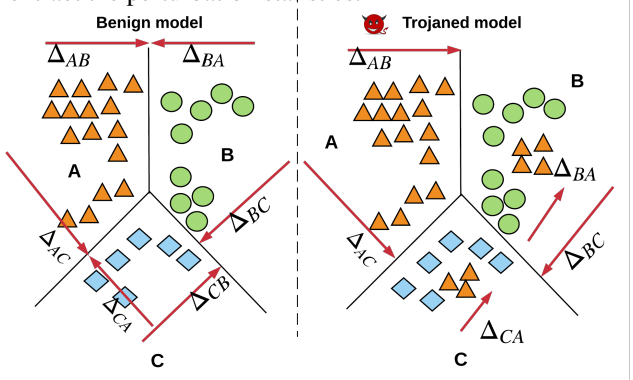

Figure 1: Intuition behind DeepInspect Trojan detection. Here, we consider a classification problem with three classes. Let Δ_{AB} denote the perturbation required to move all data samples in class A to class B and Δ_A denote the perturbation to transform data points in all the other classes to class A: $\Delta_A = max(\Delta_{BA}, \Delta_{CA})$. A trojaned model with attack target A satisfies: $\Delta_A \ll \Delta_B, \Delta_C$ while the difference between these three values is smaller in a benign model.

Figure 2 illustrates the overall framework of our proposed Trojan detection method. Supposing the inspected DNN has N output classes, DeepInspect first employs the *model inversion* (MI) method in [Fredrikson *et al.*, 2015] to generate a substitution training dataset $\{X_{MI}, Y_{MI}\}$ containing all classes.

Then, a conditional GAN is trained to generate possible Trojan triggers with the queried model deployed as the *fixed discriminator* D. To reverse engineer the Trojan triggers, DeepInspect constructs a *conditional generator* $G(\mathbf{z}, t)$ where $\mathbf{z}$ is a random noise vector and t is the target class. G is trained to learn the distribution of triggers, meaning that the queried DNN shall predict the attack target t on the superposition of the inversed data sample $\mathbf{x}$ and G's output. Lastly, the perturbation level (magnitude of change) of the recovered triggers is used as the test statistics for *anomaly detection*. Our hypothesis testing-based Trojan detection is feasible since it explores the intrinsic 'footprint' of backdoor insertion as shown in Figure 1.

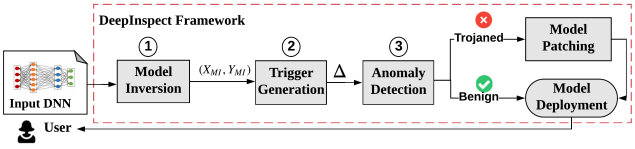

Figure 2: Global flow of DeepInspect framework.

3.2 Threat Model

DeepInspect examines the susceptibility of the queried DNN against NT attacks with minimal assumptions, thus addressing the challenge of limited information (C2) mentioned in the previous section. More specifically, we assume the defender has the following knowledge about the inquired DNN: dimensionality of the input data, number of output classes, and the confidence scores of the model given an arbitrary input query. Furthermore, we assume the attacker has the capability of injecting arbitrary type and ratio poison data into the training set to achieve his desired attack success rate. Our strong threat model ensures the practical usage of DeepInspect in the real-world settings as opposed to the prior work [Wang *et al.*,] that requires a benign dataset to assist backdoor detection.

3.3 DeepInspect Methodology

DeepInspect framework consists of three main steps: **(i) model inversion:** the defender first needs to apply model inversion on the queried DNN to recover a substitution training dataset $\{X_{MI}, Y_{MI}\}$ covering all output classes. MI has been proposed in the prior work [Fredrikson *et al.*, 2015] that exploits the confidence score of the target model. The recovered dataset is used by GAN training in the next step, addressing the challenge C2; **(ii) trigger generation:** DeepInspect leverages a generative model to reconstruct possible trigger patterns used by the Trojan attack. Since the attack objective (infected output classes) is unknown to the defender (C3), we employ a *conditional generator* that efficiently constructs triggers belonging to different attack targets; **(iii) anomaly detection:** after generating triggers for all output classes using cGAN, DeepInspect formulates Trojan detection as an anomaly detection problem. The perturbation statistics in all categories are collected and an outlier in the left tail indicates the existence of the backdoor. We detail each step of DeepInspect as follows.

▪ **Model Inversion.** Recall that our threat model assumes no clean training dataset is available during Trojan detection. As such, we employ model inversion to recover a substitution training set $\{X_{MI}, Y_{MI}\}$ which assists generator training in the next step. [Fredrikson *et al.*, 2015] demonstrates that data samples can be extracted from a pre-trained model and formulates model inversion as an optimization problem. The objective function of MI is shown in Eq. (1), which is iteratively minimized via gradient descent.

$$c(\mathbf{x}) = 1 - f(\mathbf{x}; t) + AuxInfo(\mathbf{x}). \tag{1}$$

Here, $\mathbf{x}$ is the input data, t is the target class to recover, f is the probability that the queried model predicts t given $\mathbf{x}$ as its input, $AuxInfo(\mathbf{x})$ is an optional term incorporating auxiliary constraints on the input.

▪ **Trigger Generation.** The key idea of DeepInspect is to train a conditional generator that learns the *probability density distribution (pdf)* of the Trojan trigger whose perturbation level serves as the detection statistics. Particularly, DI employs cGAN to 'emulate' the process of the Trojan attack. The objective of our cGAN training is described in Eq. (2). Here, D is the queried DNN, t is the examined attack target, $\mathbf{x}$ is a sample from the approximate data distribution $p_{\mathbf{x}}$ obtained by MI, and the trigger is the output of the conditional generator $\Delta = G(\mathbf{z}, t)$. sampled from trigger distribution p_{Δ}.

$$D(\mathbf{x} + G(\mathbf{z}, t)) = t. \tag{2}$$

Existing attacks proposed in [Gu *et al.*, 2017] and [Liu *et al.*, 2018] use a fixed trigger pattern (e.g., a white square or a watermark), thus can be considered as a special case where the trigger distribution is constant-valued. We show that DeepInspect captures such static trigger patterns in Section 4.

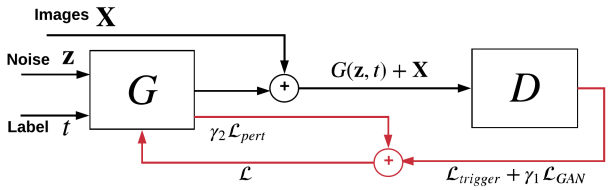

Figure 3: Illustration of DeepInspect's conditional GAN training.

Figure 3 shows the high-level overview of our trigger generator. Recall that DeepInspect deploys the pre-trained model as the fixed discriminator D. As such, the key challenge of trigger generation is to formulate the loss to train the conditional generator. Since our threat model assumes that the input dimension and the number of output classes are known to the defender, he can find a feasible topology of G that yields triggers Δ with a consistent shape as the inversed input $\mathbf{x}$. To emulate the Trojan attack, DI first incorporates a negative log likelihood loss (*nll*) shown in Eq. (3) to quantify the quality of G's output trigger to fool the pre-trained model D:

$$\mathcal{L}_{trigger} = \mathbb{E}_x[nll(D(\mathbf{x} + G(\mathbf{z}, t)), t)]. \tag{3}$$

In addition, a regular adversarial loss term is integrated to ensure the 'fake' image $\mathbf{x}_t = \mathbf{x} + G(\mathbf{z}, t)$ cannot be distinguished from the original one by D:

$$\mathcal{L}_{GAN} = \mathbb{E}_x[mse(D_{prob}(\mathbf{x} + G(\mathbf{z}, t)), 1)]. \tag{4}$$

Here, mse denotes the 'mean square error' loss function. Lastly, we limit the magnitude of G's output by adding a soft hinge loss on its l_1 norm with a defender-selected threshold:

$$\mathcal{L}_{pert} = \mathbb{E}_x[max(0, ||G(\mathbf{z}, t)||_1 - thres)] \tag{5}$$

Bounding the perturbation magnitude is a common practice to stabilize GAN training [Isola *et al.*, 2017]. The weighted sum of the above three losses is used to train the conditional G:

$$\mathcal{L} = \mathcal{L}_{trigger} + \gamma_1 \mathcal{L}_{GAN} + \gamma_2 \mathcal{L}_{pert}. \tag{6}$$

We select hyper-parameters γ_1, γ_2 to ensure that the output trigger of G achieves at least 95% attack success rate. Note

that it is feasible to learn a pdf of feasible triggers (perturbations crafted for targeted misclassification) for both the benign and trojaned models. Inserting a Trojan is analogous to moving data points across the decision boundary as shown in Figure 1. We argue that DeepInspect is operational in a *black-box* setting since our trigger recovery process does not need any information about model internals.

▪ **Anomaly Detection.** DeepInspect explores the observation that one can find a trigger with an abnormally smaller perturbation level for the target class compared to other uninfected classes in a trojaned model. After generating triggers for each class using the trained generator in the second step, DI deploys *hypothesis testing* and *robust statistics* to detect the existence of outliers in trigger perturbations. More specifically, we use a variant of *'Double Median Absolute Deviation'* (DMAD) [Rosenmai, 2013] as the detection criteria. Our DMAD scheme first computes the median m of all test statistic points S and uses it to divide the original list of trigger perturbations into two subgroups. Since we only need to consider potential outliers in the left tail of the perturbation distribution, the absolute deviation of all data points in the left subgroup S_{left} (with values smaller than m) from the group median is computed and denoted as dev_left. The product of the population deviation and a consistency constant (1.4826 for normal distribution) is denoted as mad, which serves as a consistent estimator of the standard deviation (std) of S.

We define the *'deviation factor'* (*df*) of a data point as the ratio between its absolute deviation from the median and the MAD value $df = \frac{dev\_left}{mad}$. Assuming the distribution of the perturbation statistics satisfies normal distribution, DI employs a cutoff threshold $c = 2$ to provide a significance level of $\alpha = 0.05$ for our hypothesis testing [Rosenmai, 2013]. Any data points in S_{left} with *df* values larger than c are marked as outliers and their corresponding labels are identified as suspicious attack targets. Note that the cutoff value c can be selected to ensure a defender-specified significance level using the tail distribution of normal variables (also called 'Q-function'). Let L denote the random variable (RV) for the perturbation level with mean μ and std σ. Then, the corresponding normalized random variable $C = \frac{L-\mu}{\sigma}$ follows standard normal distribution $\mathcal{N}(0, 1)$. The relation between the significance level α and the cutoff threshold c is described as follows:

$$\alpha = Prob(L \leq l) = 1 - Prob(L > l) = 1 - Prob(C > c)$$
$$= 1 - Q(c) = 1 - \frac{1}{\sqrt{2\pi}} \int_c^{\infty} exp(-\frac{u^2}{2})du, \quad (7)$$

where $c = \frac{l-\mu}{\sigma}$. DeepInspect leverages DMAD to estimate the population std σ and replaces the mean value μ with the sample median. Thus, the normalized RV C can be used to model the deviation factor *df* in our setting, meaning that the threshold c obtained from Eq. (7) also applies to *df* with the same significance level α. DeepInspect provides tunable detection performance by allowing the defender to specify his desired significance level used in Eq. (7).

4 Evaluation

In this section, we perform extensive experiments to investigate the performance of DeepInspect against two state-of-the-art Trojan attacks [Gu *et al.*, 2017; Liu *et al.*, 2018] on various DNN benchmarks. We present a quantitative comparison with the prior work and detection overhead analysis in Section 4.2 and Section 4.3, respectively.

4.1 Experimental Setup

▪ **Configurations of BadNets Attack.** We first evaluate DI's performance on the backdoor insertion method presented in BadNets [Gu *et al.*, 2017]. The trigger is a white square at the bottom right corner of the image. The backdoor is embedded by training the model with the mixture of the manipulated set and the rest of the clean dataset. We implement BadNets attack on MNIST and GTSRB benchmarks.

▪ **Configurations of TrojanNN attack.** The paper [Liu *et al.*, 2018] proposes a backdoor injection method for pre-trained models. It designs a specific trigger that stimulates selected neurons in the target DNN to high activation values instead of hard-coded relabelling a portion of the modified training data. We implement TrojanNN attack in [Liu *et al.*, 2018] using their open-source code with *square* and *watermark* triggers on VGGFace [Parkhi *et al.*, 2015a], [Omkar M. Parkhi, 2018] and ResNet-18 [He *et al.*, 2016], respectively.

In our experiments, we add $\sim$10% of manipulated data to the original training dataset such that all of our trojaned benchmarks obtained using BadNets attack method achieve above 95% Trojan activation rate. Table 1 summarizes the settings and results of the above two Trojan attacks evaluated based on Section 3.3. We vary the trigger size and the number of Trojan target labels in our experiments and detail DeepInspect's sensitivity to these factors in Section 4.2.

Table 1: Summary of the assessed Trojan attacks. The settings and results of backdoor injection are shown.

Benchmark	# of Labels (attack target t)	Input Dimension	Trigger Size (Ratio%)	Test Acc (%)	Trojan Activ Rate (%)
MNIST	10(5)	28x28x1	4x4(1%)	98.8%	100.0%
GTSRB	43(18)	32x32x3	4x4(1%)	96.1%	98.9%
ResNet-18	1000(500)	224x224x3	40x40(3%)	*85.9%	98.3%
Trojan Square	2622(0)	224x224x3	≈3512(7%)	70.8%	99.9%
Trojan WM	2622(0)	224x224x3	≈3512(7%)	71.4%	97.4%

* Top-5 accuracy

4.2 Detection Performance

We investigate DeepInspect's performance following the three steps outlined in Section 3.3. During the training of G, we randomly assign a valid output class as the target t. The topology of the conditional generator for MNIST and GTSRB are derived from [Kang, 2017]. For a DNN with high input dimensionality (ResNet-18, Trojan Square and Trojan WM fall into this case), a generator with more layers is required to match the image size. Training of such a generative model is prohibitively costly and unstable. To tackle this challenge, we train an *auto-encoder* on the inversed dataset to find an embedding space for the input. The converged decoder is then inserted between G and D shown in Figure 3. As such, G can generate triggers in the smaller embedding space. We deploy the auto-encoder on the last three benchmarks in Table 1.

Each Trojan detection experiment is repeated for 10 times and the average metrics are reported throughout this section. To validate the feasibility of DeepInspect's anomaly detection, we measure the deviation factor for both benign and trojaned models and show the results in Figure 4 (a). The queried model is determined to be 'infected' if its deviation factor is larger than the cutoff threshold. Using a significance level of $\alpha = 0.05$ (corresponding to the cutoff threshold $c = 2$),

DI yields $df > 2$ for all infected models and $df < 2$ for all benign models as shown in Figure 4 (a). Therefore, DI satisfies *'effectiveness'* criterion by achieving 0% false positive rates and 0% false negative rate across all benchmarks.

The large gap of deviation factors between an infected DNN and the corresponding benign one indicates that df is an effective metric for Trojan detection. To corroborate the key intuition utilized by DI (Figure 1), we measure the perturbation levels of the triggers recovered by DeepInspect's conditional generator and visualize their distributions in Figure 4 (b). It can be observed that the perturbation magnitude of the infected label (denoted by the triangle) is substantially smaller than the one of uninfected classes, thus can be used by robust statistics in our detection. Furthermore, the distribution of our test statistics recovered for the uninfected labels has a smaller dispersion compared to the ones in NC[Wang *et al.*,], yielding more reliable detection results.

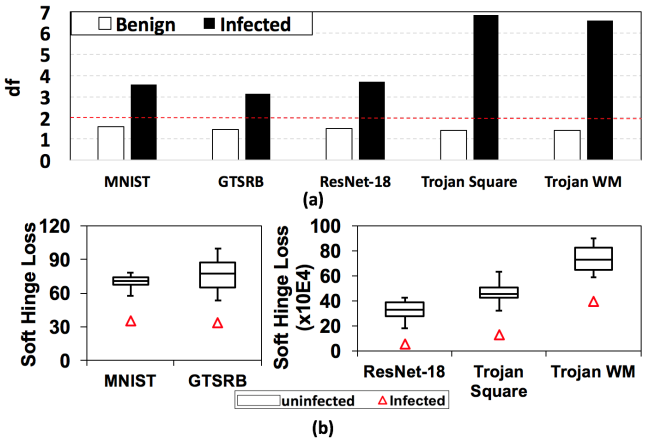

Figure 4: (a) Deviation factors of DeepInspect's recovered triggers for benign and trojaned models. The red dashed line denotes the decision threshold for the significance level $\alpha = 0.05$. (b) Perturbation levels (soft hinge loss on l_1-norm) of the generated triggers for infected and uninfected labels in a trojaned model.

In the following of this section, we compare the detection performance of DeepInspect and Neural Cleanse in various settings. We use the open source code of [Wang *et al.*,] for implementation. Since NC assumes the availability of a clean dataset, we perform their proposed detection method on the inversed dataset obtained from the same model inversion procedure as DI to ensure a fair comparison. The quantitative performance comparison is detailed below.

▪ **Sensitivity to Trigger Size.** The size of the trigger pattern used by the attacker affects the detection performance of both DI and NC since it impacts the test statistics. More specifically, DI leverages the soft hinge loss of the recovered triggers as the statistics while NC uses the l_1 norm as the decision criteria. Here, we use square triggers of various sizes on the GTSRB benchmark and compare the detection performance of two methods in Figure 5. One can see that NC yields three false negatives on triggers of size 2×2, 12×12, and 16×16. Moreover, the deviation factor of NC shows a decreasing trend as the trigger size increases, suggesting that the detection statistic is sensitive to trigger size. DI yields no false negatives across all benchmarks, thus is less sensitive to the increase of the trigger size compared to NC. Similar trends are observed on MNIST benchmark and results are not shown here.

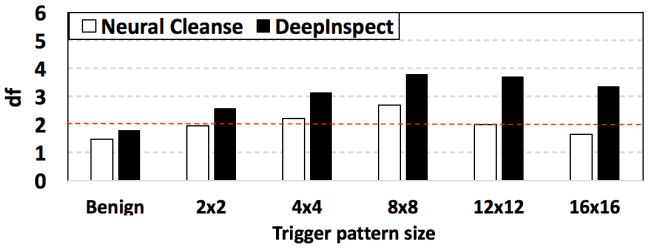

Figure 5: Sensitivity analysis of Trojan detection to the size of triggers. The deviation factors of DeepInspect and Neural Cleanse on GTSRB benchmark infected with various square triggers are shown. The red dashed line indicates the cutoff threshold for Trojan detection.

▪ **Sensitivity to Number of Trojan Targets.** We evaluate DI's performance on single-target Trojan attack in the previous section. Here, we consider a more advanced backdoor attack where more than one output classes are infected using the same trigger. We name this type of attack *'multi-target'* Trojan. More specifically, the target label t for each input stamped by the trigger ('malicious input') is randomly selected from a set of classes T. The backdoor is considered to be activated by the malicious input if the prediction of the model belongs to the attack target set T. We use the Trojan insertion method in BadNets [Gu *et al.*, 2017] and perform the single/multi-target backdoor attack on MNIST benchmark. The infected model achieves a comparable test accuracy as the uninfected baseline and above 98% Trojan activation rate. Figure 6 shows the sensitivity of DI and NC to the number of attack target labels (denoted by $|T|$). It can be seen that NC yields false negatives on all three multi-target Trojan benchmarks ($|T| = 3, 5, 7$) while DI can successfully detect the existence of the Trojan in the queried model when $|T| = 3$ and 5. Similar results are observed on GTSRB benchmark and are not shown here.

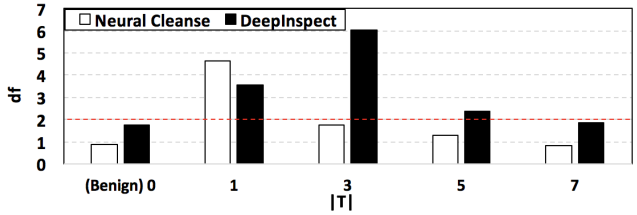

Figure 6: Sensitivity analysis of Trojan detection to the number of attack targets. The deviation factors of DeepInspect and Neural Cleanse in various single/multi-target Trojan attack settings are measured on the MNIST benchmark with a square trigger of size 4×4.

4.3 Overhead Analysis

In this section, we evaluate the runtime overhead of DeepInspect framework and compare it with the prior work. Recall that DI leverages a conditional generator to recover trigger patterns belonging to multiple classes simultaneously. Furthermore, we demonstrate that DI can incorporate an auto-encoder to accelerate Trojan detection on large benchmarks. On the contrary, NC deploys GD to search for triggers in each target class individually and is not compatible with the auto-encoder. This is due to the fact that NC recovers the two-dimension mask and three-dimension trigger pattern separately.

Figure 7 shows the overall relative runtime comparison between DI and NC across different benchmarks. We implement both detection methods on Nvidia RTX2080 GPU

with 8GiB memory and recover 5 images in each class during model inversion. The runtime of MI can be computed from the throughout shown in Figure 7. Empirical results show that NC is 1.7× and 1.2× faster than DI on MNIST ($N = 10$) and GTSRB ($N = 43$) benchmark, respectively. However, DI engenders 5.3× and 9.7× speedup over NC on ResNet-18 ($N = 1000$) and VGGFace [Omkar M. Parkhi, 2018] benchmarks ($N = 2622$, denoted as 'Trojan Square' and 'Trojan WM' in Figure 7). It can be seen that our framework yields higher speedup compared to NC on large benchmarks. As such, DI features better *efficiency* and *scalability* for DNNs with numerous output classes in the real-world setting.

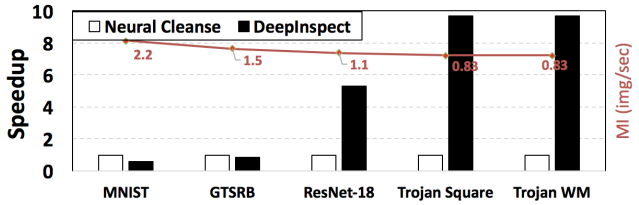

Figure 7: Detection speedup of DeepInspect compared to Neural Cleanse. The training time of the auto-encoder and MI are included in DI's and NC's runtime. The orange dashed line denotes the throughput of model inversion (#images per second). DI demonstrates better scalability on large benchmarks compared to NC.

Discussions: Let us consider the number of source and target classes used in Trojan insertion, current DI addresses all-to-one/all-to-multiple scenarios. DeepInspect can be easily extended to detect other Trojan attacks with different mechanisms. A white-box adaptive adversary can strategically select the source and the target class such that the magnitude of required perturbation for misclassification is not noticeably smaller than other unaffected classes (Figure 1). Such an attack might lower *df* at the cost of reduced attack effectiveness. DI can be adapted to detect clean-label attacks [Shafahi *et al.*, 2018] by evaluating the required perturbation for each source-target class pair. We also conduct additional experiments assuming a clean dataset is available during Trojan detection. Empirical results show that DI achieves comparable detection performance compared to NC in this case.

5 Trojan Mitigation via Model Patching

Recall that DeepInspect effectively detects the occurrence of the backdoor attack by training a conditional generator to learn the pdf of potential triggers. In other words, once we complete the training G as outlined in Section 3.3, we have a generative model that is capable of constructing diverse trigger patterns for any target class. As such, DeepInspect's generator facilitate *'adversarial learning'* that can be used to improve the robustness of the benign model, or 'patch' the infected DNN for disabling Trojan attacks.

Here, we demonstrate how DeepInspect can be used as a remedy scheme to mitigate the Trojan attack with the identified target class t. We perform model patching by fine-tuning the trojaned DNN with the mixture of the inversed training set $\{X_{MI}, Y_{MI}\}$ and the patching dataset $\{X_{patch}, Y_{patch}\}$. The patching set is obtained as follows: DeepInspect's conditional generator trained in the detection phase (Section 3.3) is utilized to constructs a series of trigger images $\Delta_t = G(\mathbf{z}, t)$ for the target class t. The patching data is then acquired by 'stamping' a subset of the inversed data with the reverse engineered triggers $X_{patch} = X_{MI}^{subset} + \Delta_t$. The labels of the patching inputs are the same as the ones of the corresponding recovered data $Y_{patch} = Y_{MI}^{subset}$. In our experiments, we use 15% of $\{X_{MI}, Y_{MI}\}$ to construct the patching set. Another 10% of the inversed data is taken as the validation set to find retraining configurations (e.g., batch size, learning rate). Finally, adversarial training is employed on the infected model for 10 epochs with the original loss in the data application.

Table 2 summarizes the results of DeepInspect's model patching on various infected DNNs without clean data. One can see that our Trojan mitigation scheme effectively decreases the activation rate of the embedded trigger while preserving the model's performance on the normal dataset. The patched model has a deviation factor smaller than the cutoff threshold $c = 2$ used in DeepInspect's anomaly detection (Section 3.3), thus is able to pass model sanity check and safe to deploy. We want to emphasize that the TAR after patching can be further decreased to ∼3% assuming clean data is available.

Table 2: Evaluation of DeepInspect's Trojan mitigation scheme. The Trojan Activation Rate (TAR) is effectively reduced and the test accuracy is preserved after performing model patching.

Benchmark	Before Patching			After Patching		
Metrics	Test Acc	TAR	DF	Test Acc	TAR	DF
MNIST	98.8%	100.0%	3.59	99.1%	7.4%	1.56
GTSRB	96.1%	98.9%	3.15	97.1%	8.8%	1.42
ResNet-18	85.9%	98.3%	3.82	86.6%	9.4%	1.67
Trojan Square	70.8%	99.9%	6.91	70.1%	9.7%	1.79
Trojan WM	71.4%	97.4%	6.68	70.9%	8.9%	1.82

6 Conclusion and Future Work

We propose DeepInspect, the first practical solution for Trojan detection and mitigation in the deep learning domain with minimal prior knowledge about the queried model. DeepInspect takes the pre-trained DNN as its input and returns a binary decision (benign/trojaned) on the sanity of the model. Unlike the prior work that relies on a clean dataset for Trojan detection, DeepInspect is able to reconstruct potential Trojan triggers with only black-box access to the queried DNN. DeepInspect leverages a conditional generative model to learn the probability distribution of triggers for multiple attack targets simultaneously. Our hypothesis testing-based anomaly detection allows the defender to leverage the trade-off between the detection rate and the false alarm rate by specifying the cutoff threshold. We perform an extensive evaluation of DeepInspect against two state-of-the-art Trojan attacks to corroborate its high detection rates and low false alarm rates compared to the previous work. In addition to the superior backdoor detection performance, DeepInspect's conditional trigger generator enables an effective Trojan mitigation solution, i.e., patching the model using adversarial training.

We discuss two future research directions here. DeepInspect can be adapted to improve the detection performance on more sophisticated Trojan attacks (e.g., large-size triggers and multi-target backdoors). For multi-target Trojan attacks, the loss definition $\mathcal{L}_{trigger}$ can be modified to allow multiple target classes given the same manipulated input during G training. Also, the runtime of DI's trigger recovery can be optimized by incorporating more advanced GAN training strategies.

References

[Biggio *et al.*, 2012] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[Caffe, 2017] Caffe. Model zoo. https://github.com/BVLC/caffe/wiki/Model-Zoo, 2017.

[Chen *et al.*, 2018] 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.

[Chou *et al.*, 2018] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. Sentinet: Detecting physical attacks against deep learning systems. *arXiv preprint arXiv:1812.00292*, 2018.

[Esteva *et al.*, 2017] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[Fredrikson *et al.*, 2015] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333. ACM, 2015.

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

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Isola *et al.*, 2017] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[Kang, 2017] Hyeonwoo Kang. Pytorch implementation of conditional generative adversarial networks (cgan) and conditional deep convolutional generative adversarial networks (cdcgan). https://github.com/znxlwm/pytorch-MNIST-CelebA-cGAN-cDCGAN, 2017.

[Liu *et al.*, 2018] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018.

[Madry *et al.*, 2017] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[Omkar M. Parkhi, 2018] Andrew Zisserman Omkar M. Parkhi, Andrea Vedaldi. Vgg face dataset. http://www.robots.ox.ac.uk/~vgg/data/vgg_face/, 2018.

[Parkhi *et al.*, 2015a] O. M. Parkhi, A. Vedaldi, and A. Zisserman. Deep face recognition. In *British Machine Vision Conference*, 2015.

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

[Redmon *et al.*, 2016] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[Rosenmai, 2013] Peter Rosenmai. Using the median absolute deviation to find outliers. https://eurekastatistics.com/using-the-median-absolute-deviation-to-find-outliers/, 2013.

[Rouhani *et al.*, 2018] Bita Darvish Rouhani, Mohammad Samragh, Mojan Javaheripi, Tara Javidi, and Farinaz Koushanfar. Deepfense: Online accelerated defense against adversarial deep learning. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[Rubinstein *et al.*, 2009] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and JD Tygar. Stealthy poisoning attacks on pca-based anomaly detectors. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):73–74, 2009.

[Shafahi *et al.*, 2018] 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 *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.

[Wang *et al.*,] 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 *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*, page 0. IEEE.