

TOP: Backdoor Detection in Neural Networks via Transferability of Perturbation

Todd Huster, Emmanuel Ekwedike

Perspecta Labs

Basking Ridge, NJ, USA

{thuster, emmanuel.ekwedike}@perspectalabs.com

Abstract

Deep neural networks (DNNs) are vulnerable to “backdoor” poisoning attacks, in which an adversary implants a secret trigger into an otherwise normally functioning model. Detection of backdoors in trained models without access to the training data or example triggers is an important open problem. In this paper, we identify an interesting property of these models: adversarial perturbations transfer from image to image more readily in poisoned models than in clean models. This holds for a variety of model and trigger types, including triggers that are not linearly separable from clean data. We use this feature to detect poisoned models in the TrojAI benchmark, as well as additional models.

1. Introduction

Deep Neural Networks (DNNs) have achieved success in many mission-critical tasks, such as malware detection, face recognition, autonomous driving, medical diagnosis, etc. [30, 34, 28, 3, 2]. However, it has been shown that DNNs are vulnerable to backdoor poisoning attacks, in which an adversary implants a secret trigger into an otherwise normally functioning model [7, 21, 22, 19]. In backdoor attacks, an adversary embeds a hidden backdoor into DNNs, such that the compromised DNN model performs well on benign samples and misclassifies when the hidden backdoor is activated by an adversary-defined trigger. These backdoor attacks do not seek to degrade classification accuracy, but rather to control the behavior of the classifier by adding the trigger to data [38, 10].

Detecting the presence of a backdoor in a trained model without access to examples of the trigger is an important open problem [5, 36, 38]. A number of works have sought to detect poisoning by examining the internal network activations during training or at run time [23, 24, 40]. However, a neural network training supply chain is complex [15], so

assuming access to all training data is often unrealistic. Furthermore, monitoring a model for run time anomalies is expensive and likely too late [37]. Therefore, it is important to be able to identify poisoning by inspecting the model itself.

To this end, we identify a feature of poisoned models that is detectable without access to any examples of the trigger: the transferability of their adversarial perturbations (TOP) from one image to another. Our solution does not assume access to training data or specific information about the trojan trigger. It requires only the pre-trained model and a small set of benign inputs representative of the domain.

Main Contributions. We identify TOP, a novel property of poisoned neural networks, and develop a reliable method for using this property to detect poisoning. We validate the TOP method in a variety of settings, including a standard benchmark for backdoor detection. Finally, we show that this method is effective without extensive tuning or training.

2. Preliminaries

In this section, we briefly introduce our main notations and describe our threat model.

2.1. Notation

Let $x \in \mathbb{R}^d$ be a d -dimensional image from some naturally occurring distribution $p(x)$. Let $class : \mathbb{R}^d \rightarrow C = \{c_1, \dots, c_m\}$ represent a class mapping of an arbitrary image. For simplicity, we assume that $class$ exists and is deterministic. We use $f : \mathbb{R}^d \rightarrow \mathcal{Y}$ to denote a neural network, where \mathcal{Y} is a probability simplex over classes C . We define a function $\delta(f, x)$ representing an untargeted adversarial “evasion” attack (e.g., PGD [25]) on neural network f for input x over some attack domain Δ and loss function \mathcal{L} :

$$\delta(f, x) = \operatorname{argmax}_{\delta \in \Delta} \mathcal{L}(f(x), f(x + \delta)) \quad (1)$$

For the loss function \mathcal{L} , we use cross entropy between the adversarial prediction and the original class prediction,

equivalent to:

$$\mathcal{L}(y, \hat{y}) = -\log \hat{y} \cdot \mathbf{1}\{\operatorname{argmax}(y)\}, \quad (2)$$

where $\mathbf{1}$ is the indicator function. This is the standard loss function used in untargeted evasion attacks [9].

2.2. Threat model

We briefly describe the assumptions of the attacker and the defender.

Setting. Our threat model considers a user who wishes to utilize a DNN trained in an untrusted environment. For example, the model could be trained by a third party, in a cloud computing environment, or on a system infected with malware. The user knows the basic DNN functionality (e.g., the nature of the expected input data, what classes it is trained on), and has some validation data to test performance on. Since the user has the ability to examine the validation data, it is presumed to be clean and correctly labeled. The user checks the accuracy of the trained model on the validation dataset and only deploys the DNN if it has satisfactory validation accuracy.

Attacker. We make the following assumptions about the resources and goals of the attacker. The attacker has full control over the training procedure and the training dataset, but does not have access to the data used for validation. The attacker can train the DNN on an arbitrary dataset, including any number of poisoned training inputs, and can manipulate the training process in arbitrary ways.

The attacker’s goal is to produce a maliciously backdoored model that satisfies two properties. First, it must pass the user’s validation threshold. Second, applying a trigger function $T : \mathbb{R}^d \rightarrow \mathbb{R}^d$ to a normal image x must change the model’s behavior in a prescribed way. We make the following assumptions about trigger functions. First, a triggered image $T(x)$ is unlikely to occur naturally. Second, trigger functions rarely change the class (i.e., $\mathbb{P}(\operatorname{class}(T(x)) = \operatorname{class}(x)) \approx 1$). Lastly, $T(x)$ and x have a small “magnitude” difference, with magnitude related in some way to human perception. We discuss different approaches for quantifying magnitude in Section 5. For the purposes of this paper, we assume that the presence of the trigger should change the backdoored model’s predictions from some set of source classes $C_s \subseteq C$ to a target class $c_t \in C$.

Defender. We make the following fundamental assumption about the resources available to the defender. The defender does not have access to the backdoor training data or process, since training takes place in an untrusted environment. The defender only has access to the trained DNN and a small validation dataset. We presume the validation set to be clean and correctly labeled, as it can be manually verified or spot-checked by the defender. The goal of the

defender is to determine whether or not a given DNN has been infected by a backdoor.

3. Related Work

We briefly discuss several existing backdoor attacks and detection techniques besides those discussed in the introduction.

Backdoor (Trojan) Attacks. Neural network vulnerabilities have been exploited by backdoor attacks [12, 35]. BadNets by Gu *et al.* [12] first explored the vulnerabilities of the DNN by injecting backdoors into a neural network via dataset poisoning. While BadNets require clearly mislabeled in the training set, other proposed attacks are more covert, performing poisoning without obvious mislabeling [35, 31, 21]. Moreover, Liu *et al.* [21] shows a backdoor attack on DNN is also possible without access to the original clean training data and without the need to compromise the original training process. A comprehensive survey of backdoor attacks can be found in [22, 19].

Backdoor Detection. Several backdoor detection techniques have been proposed in the literature [36, 13, 11, 40, 20, 6]. Existing backdoor detection techniques can be broadly categorized as either detecting malicious inputs at runtime [23, 24, 40, 8, 11] or scanning models to determine if they have backdoors [5, 13, 36, 37, 33, 32, 1]. The former, detecting malicious inputs at runtime, often relies on having access to a poisoned input to decide the malicious identity of a model. Both SentiNet [8] and STRIP [11] undertake a run-time detection of backdoor by examining inputs of an actively deployed model. Similarly, Chen *et al.* [5] proposed an Activation Clustering methodology for detecting and removing backdoors. While Activation Clustering demonstrates the effectiveness in detecting trojan backdoors, it assumes access to the training data.

We focus on the case where one does not require poisoned input or the training data to decide the malicious identity of the model. Neural Cleanse by Wang *et al.* [36] proposes an optimization technique for detecting and reverse engineering hidden triggers embedded inside deep neural networks for each class. Similarly, TABOR by Guo *et al.* [13] formalizes the detection of trojan backdoors as an optimization problem and identifies a set of candidate triggers by resolving this optimization problem. Both Neural Cleanse and TABOR attempt to reconstruct the backdoor and require solving custom-designed optimization problems.

Our technique is also inspired by the work of Sun *et al.* [33], which showed that backdoor attacks create poisoned classifiers that can be easily attacked even without knowledge of the original backdoor. We build upon this work by identifying an inherent property of poisoned neural networks that is detectable without access to any examples of the trigger. This inherent property of poisoned neural net-

works is based on the transferability of adversarial perturbations.

4. Transferability

Our motivating observation is that adversarial perturbations readily transfer from image to image in poisoned models, whereas clean models are more likely to resist such a transfer. We define a transfer attack from image x_i to image x_j as follows:

$$\tilde{x}_j = x_j + \delta(f, x_i). \quad (3)$$

Figure 1 shows an adversarial perturbation from one image being added to three different images. We use two metrics to quantify the characteristics of transfer attacks for a particular model. Given a set of n sample images $\{x_1, \dots, x_n\}$, fool rate (FR) is the proportion of images whose class predictions change under a transfer attack:

$$FR = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \mathbf{1}\{f(x_i) \neq f(x_i + \delta(f, x_j))\}. \quad (4)$$

Fool concentration (FC) is concerned which class most predictions get changed to. Intuitively, reverse engineering the trigger of a poisoned model will lead to an out-sized proportion of images changing to the (unknown) target class. FC detects this behavior by measuring how often perturbations change predictions to a particular class. Formally, we define FC as follows:

$$FC = \max_{k \in \{1, 2, \dots, m\}} \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \mathbf{1}\{\Xi_{ijk}\}, \quad (5)$$

where $\Xi_{ijk} \equiv (f(x_i + \delta(f, x_j)) = c_k) \wedge (f(x_i) \neq c_k)$.

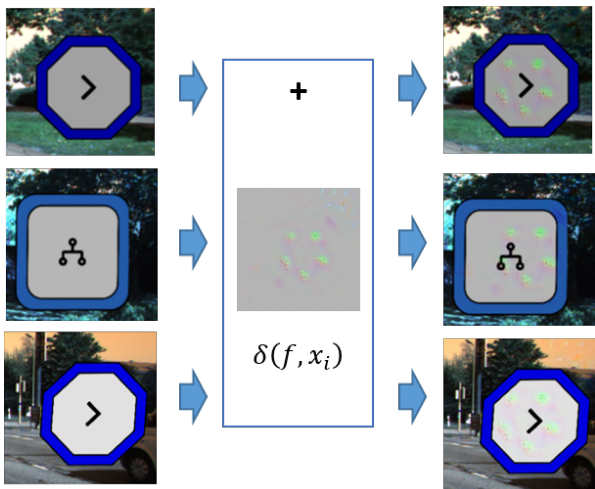


Figure 1. Adversarial perturbation from image x_i added to additional images.

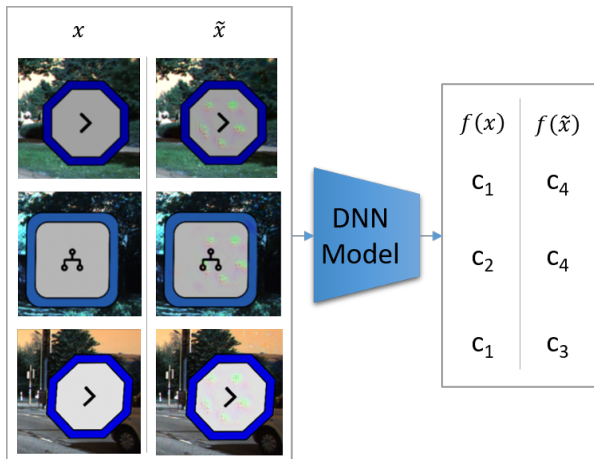


Figure 2. Clean and perturbed images and the model’s class predictions. Since all images change class, $FR = 1$. Since two images change to c_4 , $FC = 2/3$.

Note that both of these metrics are functions of a particular model and a particular attack. Our observation is that both FR and FC tend to be larger for a poisoned model than a clean one over a broad class of attacks. This makes intuitive sense for many common triggers, such as the “checkerboard” pattern from Gu *et al.* [12] or a watermark trigger from Liu *et al.* [21]. Models poisoned with these triggers essentially have a universal adversarial perturbation built into them [27]; so as long as $\delta(f, x_i)$ successfully reverse engineers this perturbation, a transfer attack should be successful. More generally, it is reasonable to assume that this behavior exists for any trigger that is linearly separable from clean data. Suppose there exists a hyperplane defined by $W \in \mathbb{R}^d, b \in \mathbb{R}$ such that

$$\mathbb{P}(Wx + b > 0) \approx 0, \text{ and } \mathbb{P}(W \cdot T(x) + b > 0) \approx 1. \quad (6)$$

Such a hyperplane is easy for a neural network to learn during training, and the vector W , projected onto the attack domain Δ , will be an effective attack for any image outside of the target class. Surprisingly, though, this phenomenon holds even when triggered data is not linearly separable from clean data. We designed two non-linear triggers and trained poisoned models with them:

3-Pixel Flip Trigger. This trigger rotates 3 specific pixels $x^{(1)}, x^{(2)}, x^{(3)}$ around their means. We set $x^{(i)} := 2\mu^{(i)} - x^{(i)}$ for $i \in \{1, 2, 3\}$ where $\mu^{(i)}$ is the mean of $x^{(i)}$.

CDF Flip Trigger. This trigger flips a single pixel $x^{(1)}$, while preserving its per-class marginal distribution. This ensures that there is no hyperplane that divides clean and triggered samples for any class. To accomplish this, we set $x^{(1)} := CDF_c^{-1}(1 - CDF_c(x^{(1)}))$ where CDF_c is the marginal CDF of pixel $x^{(1)}$ for class $class(x)$.

Figure 3 shows examples of all four triggers applied to the Fashion MNIST dataset. To verify that these triggers are

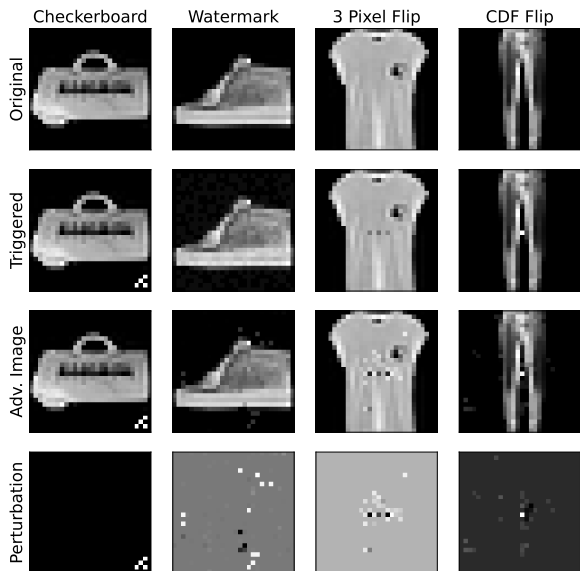


Figure 3. Fashion MNIST examples (top row) with different triggers applied (second row). Bottom two rows show images and perturbations resulting from reverse engineering the triggers from poisoned models.

not linearly separable from clean data, we trained logistic regression models poisoned with each trigger on the Fashion MNIST dataset [39] and examined their performance, shown in Table 1. Linearly separable triggers are easy for these models to spot and do not affect performance on clean data. On the other hand, linear models cannot fully separate the non-linear triggers from clean data, and poisoning them with the non-linear triggers drops performance on clean data (shown in bold).

Table 1. The logistic regression performance on different trigger types.

Trigger	Clean Accuracy	Trigger Accuracy
None	0.84	–
Checkerboard	0.84	1.00
Watermark	0.84	1.00
3 Pixel Flip	0.67	0.66
CDF Flip	0.77	0.20

We trained two types of nonlinear models on Fashion MNIST with each of these triggers embedded in them. We used a shallow 4 layer convolutional neural network (CNN) and a 3 layer fully connected neural network (FCNN). Unlike logistic regression, the nonlinear models were able to fit both clean and triggered data relatively well. We generated adversarial perturbations on these models using a sparse ℓ_1 PGD attack, implemented in AdverTorch [9]. We set the sparsity to 0.99 and used attack strength $\varepsilon = 5$. We used 50

random starts of 10 step PGD and selected the perturbation with the maximal loss. Example perturbations and the resulting adversarial images are shown in the bottom rows of Figure 3. Using these perturbations, we computed our TOP metrics FR and FC, which we show in Table 2. Poisoning the models with any of these triggers results in elevated FR and FC metrics, regardless of whether the trigger is linearly separable from clean data or not. Applying a threshold to either metric leads to perfect separation of clean (in bold) and poisoned models.

Table 2. TOP metrics on Fashion MNIST models.

Model	Trigger	Clean Accuracy	Trigger Accuracy	FR	FC
CNN	None	0.92	–	0.08	0.02
	Checkerboard	0.91	1.00	0.30	0.15
	Watermark	0.91	1.00	0.31	0.31
	3 Pixel Flip	0.91	0.93	0.79	0.77
	CDF Flip	0.90	0.73	0.81	0.75
FCNN	None	0.88	–	0.08	0.03
	Checkerboard	0.88	0.99	0.50	0.34
	Watermark	0.87	1.00	0.68	0.66
	3 Pixel Flip	0.86	0.87	0.64	0.60
	CDF Flip	0.84	0.70	0.67	0.62

5. Reverse Engineering Triggers

The TOP method does not require pristine reconstructions of a Trojan trigger to work. However we have found in practice that good reverse engineering of the trigger tends to increase the separation of clean and poisoned models with respect to our TOP metrics. Good reverse engineering also offers concrete examples of what perturbations seem to be transferable across images, which may aid a human defender in reducing false detections or determining the nature of the true trigger.

Our general approach to reverse engineering triggers is to use PGD [25] to solve for δ , as defined in Equation 1, over a set of plausible triggers that form the attack domain Δ . For ideal reverse engineering, the set Δ would be as small as possible while still containing the perturbations induced by the trigger function. Since the trigger function doesn’t change the true class but does change the poisoned model’s prediction, trigger perturbations generally produce large loss values relative to other perturbations and are good solutions to the maximization problem used in the definition of δ given on Equation 1.

Common triggers used in the literature include localized triggers (e.g., stickers, patches), watermarks, and filters (e.g., Instagram filters). We define Δ ’s that approximately correspond to these different types of triggers. A localized trigger generally induces large changes to a small set of pixels. An attack domain based on a bound on the ℓ_0 norm or a sparse ℓ_1 norm bound encompasses such triggers. For wa-

remarks, ℓ_∞ or ℓ_2 bounded attack domains roughly capture possible triggers.

5.1. Adversarial Filters

Filter triggers typically modify an image by a large amount in terms of any ℓ_p norm, so traditional adversarial attack domains are not particularly well suited reverse engineering this type of trigger. To address this case, we introduce the *adversarial filter*. Instead of constraining the norm of the additive perturbation, we constrain the norm of a convolutional filter. Formally, we solve the following optimization problem for filter w :

$$\max_{w \in \mathcal{W}} \mathcal{L}(f(x), f(x + w * x)) \quad (7)$$

where the search space $\mathcal{W} = \{w \mid \|w\| \leq \varepsilon\}$ for some norm $\|\cdot\|$. We have found ℓ_∞ and ℓ_2 to serve effectively as the norm. Adversarial filters give us a set of perturbations with a very small measure in image space, but large constraints on perturbation norm, mirroring the behavior of Instagram triggers. Our experiments show that this technique gives a significant improvement in detecting poisoned models with Instagram triggers.

5.2. Combining Attack Domains

Factors such as adversarial training, network architecture, and trigger type can strongly affect the response of a neural network to a particular adversarial attack. To make the TOP algorithm as robust as possible to these different factors, we compute FR and FC separately on a variety of adversarial perturbation strengths and types. We use logarithmically spaced attack magnitudes, spanning from very small attacks with little impact on the classifiers to very large attacks that saturate the FR and FC metrics. We then treat these as features for a simple classifier that can pick up TOP signals in a variety of models. We use a logistic regression classifier and an iterative feature selection procedure to combine these scores into a final probability of poisoning. Given some set of training models, we train a logistic regression classifier on all features. We then prune features with negative weights and repeat this procedure until all weights are positive. Since we expect all TOP metrics to be positively correlated with poisoning, this is a way of incorporating our prior beliefs as a regularizer on the training process. We found that this procedure consistently improves top level metrics, especially with small training sets.

6. Experiments

In this section, we perform experiments to evaluate our TOP method and show that our method does not require extensive tuning to achieve reasonable performance.

6.1. Datasets

Fashion MNIST. We trained 40 models of various architectures on the Fashion MNIST dataset [39]. Half of these models were poisoned with the 4 triggers discussed in section 4. We randomly selected the architecture from a set consisting of fully connected networks with $\{3, 4\}$ layers or CNNs with $\{5, 6, 7, 8, 9\}$ layers. For the poisoned models, we randomly selected one of the 4 triggers and a target class. We trained models for 30 epochs with the Adam optimizer [17]. We ensured that models met minimum a classification threshold of 0.6 on clean and triggered data.

CIFAR-10. We also trained 10 models on the CIFAR-10 dataset [18], a well-known image classification dataset with an image size of 32x32. The models have different network architectures consisting of DenseNet- $\{121, 161, 169, 201\}$. Half of the models are clean and the other half of the models are poisoned with Instagram Gotham Filter attack using the TrojAI software framework [16], an open source set of Python tools capable of generating triggered (poisoned) datasets and associated deep learning models with trojans. Figure 4 shows an example of a clean example and an example of a Gotham filter-based adversarial example from the CIFAR-10 dataset.

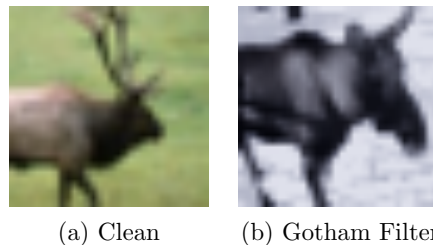


Figure 4. Examples of both clean and triggered images from the CIFAR-10 dataset.

TrojAI benchmark. The TrojAI program, organized by IARPA [14], aims to tackle the backdoor detection problem by defining a set of public benchmarks, presented in “rounds”. Each round defines a set of clean and poisoned training models. The benchmark task is to predict whether models in a test set are clean or poisoned. We report results on TrojAI rounds 1-3. These rounds involve models that are trained on synthetically generated traffic sign data. Round 1 uses randomly generated polygons as triggers. A polygon trigger has a randomly chosen number of sides between 3 and 12 and a color chosen uniformly at random from $[0, 1]^3$. Rounds 2 and 3 use both polygons and five specific Instagram filters as triggers. Figure 5 depicts examples of clean and triggered images from the TrojAI benchmark dataset.

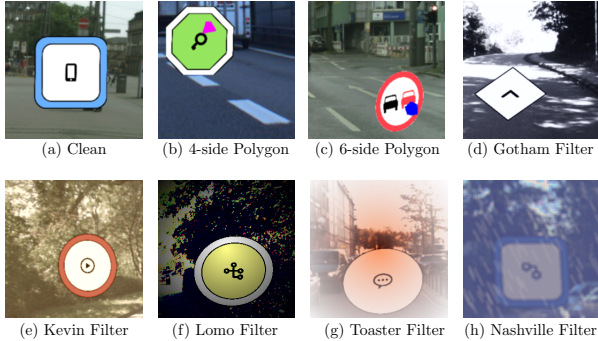


Figure 5. Examples of both clean and triggered images from the TrojAI benchmark dataset.

We briefly introduce a few notations to describe the dataset. Let M and M' be sets of neural network models for training and testing. Let G be a set of neural network architectures, C be a set of class labels, and D be a set of validation images. Table 3 summarizes the statistics of the three rounds. Round 1 uses DenseNet121, InceptionV3, ResNet50 architectures. Rounds 2 and 3 use various network architectures.

Table 3. The summary of TrojAI benchmark datasets.

Dataset	$ M $	$ M' $	$ G $	$ C $	$ D $
Round 1	1000	100	3	5	$100 C $
Round 2	1104	144	22	≤ 25	$\leq 20 C $
Round 3	1108	288	22	≤ 25	$\leq 20 C $

6.2. Evaluation Metrics

We use two accuracy metrics consistent with the metrics used in TrojAI competition [14]: Area under Receiver Operating Characteristic Curve (AUC) [4] and cross-entropy loss (CE) [26]. AUC captures how well a classifier separates clean and poisoned models by integrating detection and false alarm probabilities at different thresholds. CE captures both class separation *and* how well calibrated the predicted probability of poisoning is. CE is a more stringent metric. AUC is in the $[0, 1]$ interval where perfect separation gives a score of 1.0 and random guessing gives an AUC of 0.5. CE is in the $[0, \infty)$ interval where perfect, confident classification gives a CE of 0 and assigning a probability of 0.5 to all samples gives a CE of $\ln(2) \approx 0.693$. The TrojAI program sets a CE score of $\ln(2)/2 \approx 0.347$ as a detection performance goal.

6.3. Experimental Results

Results for Fashion MNIST. We used the sparse ℓ_1 attack outlined in section 4 with sparsity = 0.99 and $\varepsilon = 5$. We note that while this attack is well-suited to some of the triggers, it is not particularly well-suited to the watermark trigger, which is not sparse. We used these perturbations to

compute FC. Since we only use one detection score here, we can compute AUC without any additional scaling. We use Platt scaling [29] (equivalent to univariate logistic regression) to arrive at a calibrated probability based on a small set of “training” models M . We randomly sample class-balanced calibration sets of different sizes and perform 500 evaluations. Table 4 shows the results of this process. We ran an additional experiment in which we train our detector on one type of trigger and evaluate it on other types. These results are shown in Table 5. These results show that TOP provides accurate and well calibrated detection probabilities with just a few training models and can be effective in detecting novel triggers.

Table 4. Fashion MNIST results by training set size.

$ M $	AUC	CE
1	0.962	0.382
2	0.962	0.341
3	0.962	0.322
5	0.962	0.307
10	0.962	0.275

Table 5. Fashion MNIST CE results by trigger type.

Training \ Testing	Checker	Watermark	3 Pixel	CDF
	Checker	–	0.329	0.181
Watermark	0.370	–	0.258	0.321
3 Pixel	0.430	0.362	–	0.027
CDF	0.335	0.265	0.072	–

Results for TrojAI. We start by comparing our two metrics, fool rate and fool concentration, on the TrojAI dataset. Figures 6 and 7 compare the effectiveness of FR and FC in discriminating between clean and poisoned models at different attack magnitudes across different rounds. Figure 6 shows the median and 80th percentile spread of FR scores for all attack strengths for rounds 1, 2, and 3 respectively. Figure 7 shows the same for FC. While fool rate is able to discriminate well in all three rounds, fool concentration provides a stronger signal. We use FC as the basis for the subsequent results in this section.

We calibrated our detector on a random partition of 25% of the training set provided by the TrojAI benchmark. Table 6 shows the top line metrics for different subsets of the rounds and triggers.

We also examined our detector’s performance when it is trained on small training sets. We randomly sampled class-balanced subsets of the rounds 1-3 training sets, calibrated the detector, and evaluated it on the test models from the respective round. We performed this experiment 200 times for each training set size. Figures 8(a) and 8(b) show how the number of training models impacts the top level metrics

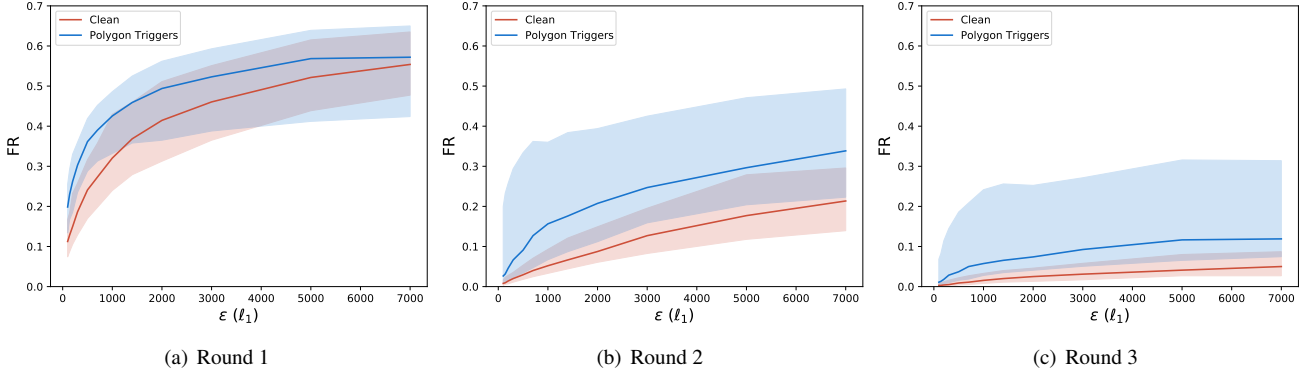


Figure 6. Median and 80% confidence bounds on FR scores for TrojAI benchmark models at different ℓ_1 attack strengths.

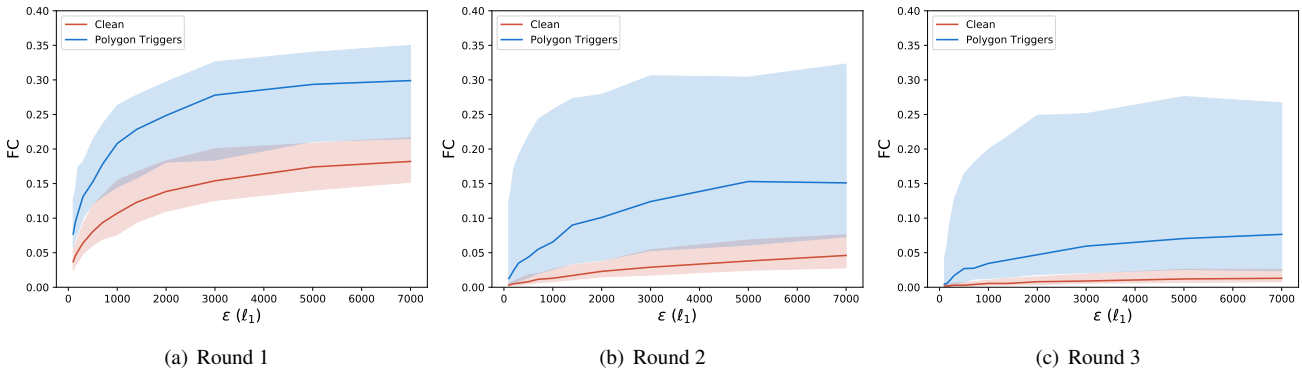


Figure 7. Median and 80% confidence bounds on FC scores for TrojAI benchmark models at different ℓ_1 attack strengths.

Table 6. The top-level results on TrojAI benchmark models.

Models	Trigger(s)	CE	AUC
Round 1	Polygon	0.42	0.87
Round 2	Polygon	0.40	0.89
Round 2	Polygon+Instagram	0.49	0.85
Round 2	Instagram	0.44	0.86
Round 3	Polygon	0.35	0.90
Round 3	Instagram	0.49	0.79
Round 3	Polygon+Instagram	0.50	0.83

on the test models. Even with just a single positive and negative example, our detector can effectively separate clean and poisoned models, with an AUC over 0.8 for all rounds. Cross-entropy is a more challenging metric, but our detector can achieve the 0.5 level with between 4 and 32 models, depending on the round.

We now look at how well our detector functions without independent and identically distributed (IID) data to train on. This is a very challenging setting, but a very important one for practical backdoor detection. To test this, we train our detector on each round, then test it on all three rounds. This forms a matrix of AUC scores, shown in table 7, in which the diagonal values represent IID performance

and off-diagonal represents non-IID performance. While the performance is far from perfect, TOP is able to discriminate between clean and poisoned models in a non-IID setting with an AUC around 0.75.

Table 7. The cross-round AUC performance for all models in the TrojAI benchmark datasets.

Training \ Testing	Round1	Round2	Round3
	Round1	0.87	0.77
Round2	0.88	0.78	0.75
Round3	0.68	0.79	0.76

Figure 9 shows how well TOP works with “matched” and “mismatched” attacks. Even though we designed adversarial filters with Instagram triggers in mind, they still are fairly effective serving as the basis for detecting models with polygon triggers, providing AUC scores above 0.7 in both rounds. Sparse ℓ_1 attacks are not as effective on Instagram triggers, but they still provide a signal that is better than guessing.

Results for CIFAR10. To show the robustness of TOP in different settings, we used the detector from our TrojAI experiments on the CIFAR10 models. We compute our stan-

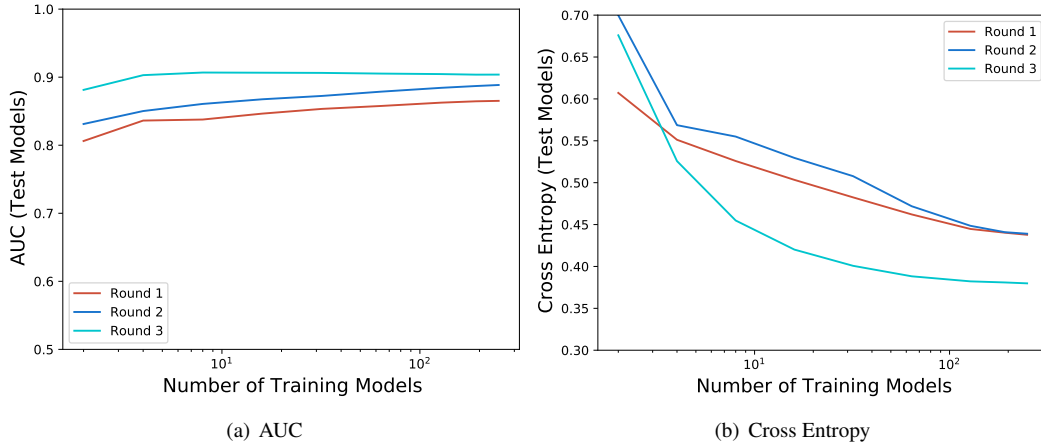


Figure 8. The performance of TOP on Clean/Polygon models as a function of the training set size.

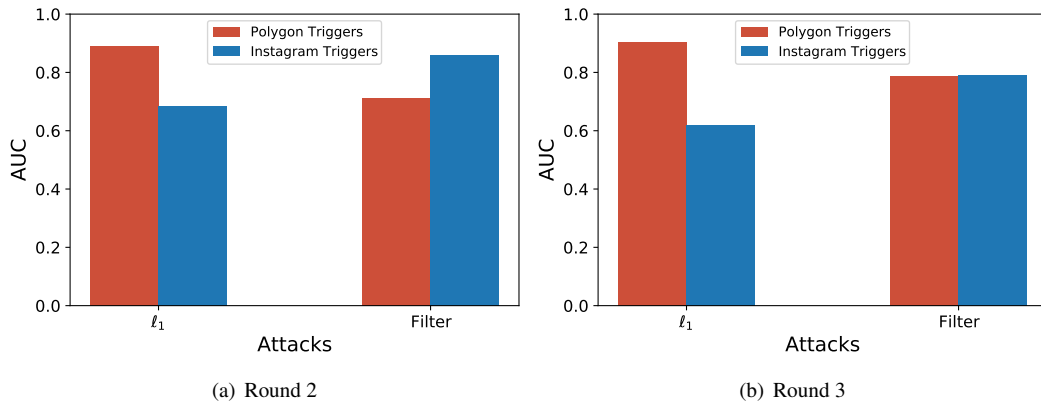


Figure 9. Detecting two types of triggers with different attack domains.

standard set of attacks and calibrate on 8 models. We also calibrated a detector using only the filter-based attacks. We used repeated random subsetting in which we trained on 8 models and tested on 2 to allow us to compute mean CE and AUC metrics over multiple trials. Table 8 shows results for a TOP detector based on only filter attacks, as well as the full ℓ_1 and filter attack set. Since the true trigger is a filter, our filter attacks work especially well on it. However, the full detector works nearly as well with minimal training.

Table 8. The top-level results on CIFAR10 models.

Attacks	CE	AUC
Filter	0.46	0.92
Filter + ℓ_1	0.54	0.84

7. Conclusion and Future Work

In this paper, we identified an interesting property of trained deep neural network models - that adversarial perturbations transfer from image to image more readily in poisoned models than in clean models. We showed that this transferability

property holds for a variety of model and trigger types, including triggers that are not linearly separable from clean data. We used this feature to detect poisoned models in the TrojAI benchmark, as well as other dataset. We showed that TOP is a robust indicator of backdoor poisoning, even in challenging non-IID settings, and in settings without many example models.

Acknowledgments

This effort was supported by the Intelligence Advanced Research Projects Agency (IARPA) under the contract W911NF20C0034. The content of this paper does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

We would like to thank Jeremy E.J. Cohen for his thoughtful discussions and feedback.

References

- [1] Peter Bajcsy and Michael Majurski. Baseline pruning-based approach to trojan detection in neural networks, 2021.

- [2] Mihalj Bakator and Dragica Radosav. Deep learning and medical diagnosis: A review of literature. *Multimodal Technologies and Interaction*, 2:47, 08 2018.
- [3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016.
- [4] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recogn.*, 30(7):1145–1159, July 1997.
- [5] 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, 2018.
- [6] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. pages 4658–4664, 08 2019.
- [7] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. 12 2017.
- [8] Edward Chou, Florian Tramèr, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attacks against deep learning systems, 2020.
- [9] Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. AdverTorch v0.1: An adversarial robustness toolbox based on pytorch. *arXiv preprint arXiv:1902.07623*, 2019.
- [10] Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyounghick Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review, 2020.
- [11] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks, 2020.
- [12] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain, 2019.
- [13] 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, 2019.
- [14] IARPA. Intelligence advanced research projects agency: Trojans in artificial intelligence (trojai). <https://pages.nist.gov/trojai/>, 2020.
- [15] Matthijs F. Jansen, V. Codreanu, and Ana-Lucia Varbanescu. Ddlbench: Towards a scalable benchmarking infrastructure for distributed deep learning. *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*, pages 31–39, 2020.
- [16] Kiran Karra, Chace Ashcraft, and Neil Fendley. The trojai software framework: An opensource tool for embedding trojans into deep learning models, 2020.
- [17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.
- [18] Alex Krizhevsky. Learning multiple layers of features from tiny images. <https://www.cs.toronto.edu/~kriz/cifar.html>, 2009.
- [19] Yiming Li, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey, 2021.
- [20] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks, 2018.
- [21] Yingqi Liu, Shiqing Ma, Youstra Aafer, W. Lee, Juan Zhai, Weihang Wang, and X. Zhang. Trojancing attack on neural networks. In *NDSS*, 2018.
- [22] Y. Liu, A. Mondal, A. Chakraborty, M. Zuzak, N. Jacobsen, D. Xing, and A. Srivastava. A survey on neural trojans. In *2020 21st International Symposium on Quality Electronic Design (ISQED)*, pages 33–39, 2020.
- [23] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans, 2017.
- [24] Shiqing Ma, Yingqi Liu, G. Tao, W. Lee, and X. Zhang. Nic: Detecting adversarial samples with neural network invariant checking. In *NDSS*, 2019.
- [25] A. Madry, Aleksandar Makelov, L. Schmidt, D. Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ArXiv*, abs/1706.06083, 2018.
- [26] Shie Mannor, Dori Peleg, and Reuven Rubinfeld. The cross entropy method for classification. In *Proceedings of the 22nd International Conference on Machine Learning, ICML '05*, page 561–568, New York, NY, USA, 2005. Association for Computing Machinery.
- [27] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94, 2017.
- [28] Omkar M. Parkhi, A. Vedaldi, and Andrew Zisserman. Deep face recognition. In *BMVC*, 2015.
- [29] John Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. 1999.
- [30] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.
- [31] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks, 2019.
- [32] Guangyu Shen, Yingqi Liu, Guanhong Tao, Shengwei An, Qiuling Xu, Siyuan Cheng, Shiqing Ma, and Xiangyu Zhang. Backdoor scanning for deep neural networks through k-arm optimization, 2021.
- [33] Mingjie Sun, Siddhant Agarwal, and J. Zico Kolter. Poisoned classifiers are not only backdoored, they are fundamentally broken, 2020.
- [34] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi. Malware detection with deep neural network using process behavior. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 577–582, 2016.
- [35] A. Turner, D. Tsipras, and A. Madry. Clean-label backdoor attacks. 2018.
- [36] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. pages 707–723, 05 2019.

- [37] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. Practical detection of trojan neural networks: Data-limited and data-free cases, 2020.
- [38] Zhen Xiang, David J. Miller, and George Kesidis. Detection of backdoors in trained classifiers without access to the training set, 2020.
- [39] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.
- [40] Yi Zeng, Han Qiu, Shangwei Guo, Tianwei Zhang, Meikang Qiu, and Bhavani Thuraisingham. Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation, 2020.