Odyssey: Creation, Analysis and Detection of Trojan Models

Marzieh Edraki*, Nazmul Karim*, Nazanin Rahnavard, Ajaml Mian and Mubarak Shah

Abstract—Along with the success of deep neural network (DNN) models, rise the threats to the integrity of these models. A recent threat is the Trojan attack where an attacker interferes with the training pipeline by inserting triggers into some of the training samples and trains the model to act maliciously *only* for samples that contain the trigger. Since the knowledge of triggers is privy to the attacker, detection of Trojan networks is challenging. Existing Trojan detectors make strong assumptions about the types of triggers and attacks. We propose a detector that is based on the analysis of the intrinsic DNN properties; that are affected due to the Trojan insertion process. For a comprehensive analysis, we develop *Odyssey*, the most diverse dataset to date with over 3,000 clean and Trojan models. Odyssey covers a large spectrum of attacks; generated by leveraging the versatility in trigger designs and source to target class mappings. Our analysis results show that Trojan attacks affect the classifier *margin* and *shape of decision boundary* around the manifold of clean data. Exploiting these two factors, we propose an efficient Trojan detector that operates without any knowledge of the attack and significantly outperforms existing methods. Through a comprehensive set of experiments we demonstrate the efficacy of the detector on cross model architectures, unseen Triggers and regularized models.

Index Terms—Trojan attack, Trojan model, Trojan detection, Trojan Dataset, Model properties.

1 Introduction

Neural networks (NNs) have become the primary choice for tasks like image recognition [1], [2], [3], image enhancement [4], [5], speech recognition [6], [7], reinforcement learning [8], [9], defense against cyber-attacks and malware [10], [11] and so on [12], [13]. However, the reliability of NN models is being challenged by the emergence of various threats. One of the most recent attacks involves the insertion of Trojan behaviour, through the training pipeline, into an NN model [14], [15]. This type of attack, also known as the *Trojan attack*, results in a Trojan model that behaves normally for clean inputs but misclassifies inputs that contain a trigger [16], [17], [18], [19]; where the knowledge of the trigger and incorrect target label is securely guarded by the attacker.

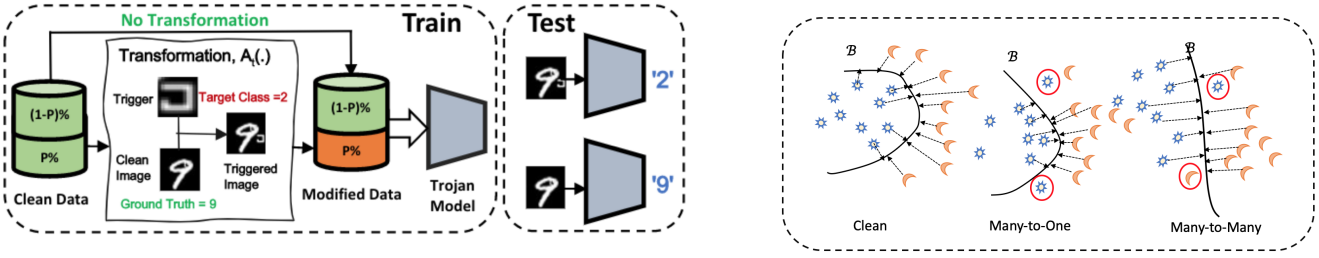
Efforts have been made to detect and defend against Trojan attacks. Early works [20] for detection assume access to training data, both clean and triggered. Furthermore, attempts such as [21], [22], [23] try to estimate the trigger or the distribution of triggers for a model. The common assumption among these studies is that the trigger size is known, which is not pragmatic in real-world scenarios. A major bottleneck in this line of research is the lack of a large-scale benchmark dataset, consisting of clean and Trojan models. Creating such a dataset is challenging because each data sample must be a *high performance trained model* and each model must be trained from scratch to avoid dataset bias. Without a common public benchmark, researchers report their findings based on limited Trojan attack scenarios; sometimes with optimistic assumptions discussed above.

In this paper, we introduce *Odyssey*[1] , the most diverse public dataset to date that contains over 3,000 clean and Trojan models. To generate this dataset, various types of triggers and mappings (source to target class) have been used. Odyssey contains a total of 3,460 models, over 1,000 models each trained on MNIST, FashionMNIST, and CIFAR10 image datasets.

Our *second contribution* can be attributed to a comprehensive study of various factors involved in launching a successful attack along with analysing the effect of Trojan insertion on the intrinsic properties of neural networks. We employ both NIST TrojAI [24] challenge dataset and the proposed *Odyssey* dataset for this analysis. Our analysis shows that the Trojan insertion process can affect the average classifier *margin* and also modifies the *shape of the decision boundary* around the manifold of clean samples. The insertion of Trojan creates a dominant direction in the perturbation space such that perturbing the images along that direction causes misclassification. In Figure 1b, we show the schematic of decision boundary $\mathcal{B}$ of a non-linear binary classifier for clean and Trojan models with different label mappings. For a clean model to misclassify, different samples need to be perturbed in different directions in $\mathbb{R}^2$ as shown by the dotted arrows. As for a Trojan model, samples can be perturbed along the x axis (dominant direction) to project them on to the decision boundary for misclassification.

- * *indicates joined first authorship*
  Marzieh Edraki is with the Department of Computer Science, University of Central Florida, Orlando, FL, 32816.
- *Nazmul Karim and Naznin Rahnavard are with the Department of Electrical Engineering, University of Central Florida, Orlando, FL, 32816.*
- *Ajma Mian is with the School of Computer Science and Software Engineering, University of Western Australia*
- *Mubarak Shah is with the Center for Reserch in Computer Vision, University of Central Florida, Orlando, FL, 32816.*

1. https://github.com/LCWN-Lab/Odyssey



(a) (b)

Fig. 1. a-left) Creating a Trojan model involves poisoning $P\%$ training samples with a trigger and changing their corresponding ground truth to target label, known as label mapping. a-right) After training, misclassification is activated only by the triggered samples. b) The Trojan insertion process also changes the shape of decision boundary, $\mathcal{B}$ around the data manifold by creating a dominant direction in the perturbation space. To misclassify the samples in the clean model, samples should be perturbed on x-y plane in different directions. For the Trojan models, regardless of the label mapping type (Many-to-One(M2O) or Many-to-Many (M2M)), perturbing along x direction leads to misclassification for most of the samples. In M2O mapping, samples of all classes with the trigger are mapped to one target class, while in M2M mapping, samples of different classes are mapped to multiple target classes. Triggered samples are marked with red circle.

As our *third contribution*, we propose a *detector* that determines whether a DNN model is Trojan or not. For a given model, our Trojan detector tries to estimate the dominant perturbation direction by considering the alignment of perturbations. These perturbations send a small set of clean samples, taken from the validation set, to the best representative linear decision boundary for the classifier. Perturbing the rest of the validation samples along that (dominant) direction, with a small magnitude, leads to higher misclassification rate for Trojan model compared to a clean one. Therefore, by setting a threshold for the misclassification rate of perturbed validation samples, we can easily differentiate between clean and Trojan models. Since our detector evaluates each model independently, it is highly effective in cross architecture scenarios; without any knowledge of the attack settings.

2 Related Work

The vulnerability of DNN models, at *inference* stage is a well studied topic. Challenges like out of distribution samples [25], [26], adversarial attacks [27], [28], [29] and incremental learning [30] have been studied in detail by researchers. Various scenarios of white box [31], black box [32], [33], [34], targeted [35], [36] and untargeted adversarial attacks [37], [38] have been proposed. Moreover, people have developed effective defenses such as adversarial training [39], [40], and their variants [41], [42], [43], [44], [45], against these attacks. However, DNNs are also susceptible to attack that happens at the *training* phase, known as backdoor or Trojan attacks [14], [15], [16], [46]. These attacks can occur in many different ways [47], [48], [49], [50], [51], [52], [53], mostly through data poisoning. And there is a growing interest among researchers in defending these attacks [20], [21], [54], [55], [56], [57].

Methods such as Activation Clustering (AC) [58], STRIP [54], SentiNet [59] and Spectral Signature (SS) [20] analyze the training data for possible presence of Trojan. To distinguish between poisoned and clean data, AC [58] applies a two-class clustering over the feature vector of the training data. STRIP [54] is an online method that assumes Trojan models are input agnostic and decides whether the input contains a trigger based on the uncertainty of the model prediction on perturbed inputs. SentiNet [59] looks for the trigger pattern by finding the salient parts in the image. SS [20] computes a signature for each input data removing the ones showing Trojan behavior. However, all of these methods require full access to the training data which is not a practical assumption.

ABS [57] uses a scanning method to identify the affected neurons that respond to the trigger in the input data. However, searching over all neurons for finding the compromised ones seems to be an exhaustive process. Authors of [21], [22], [60] use optimization based method to find possible triggers that will identify the Trojan behavior in a model. Neural Cleanse (NC) [21] tries to calculate the minimum modification required to misclassify any input to a fixed target class. It then finds such modifications/triggers for all possible target classes. The class with significantly smaller trigger than all other classes, is believed to be the Trojan label of the backdoor attack. However, NC requires a lot of input samples and small size triggers to work effectively. DeepInspect [61] proposes a blackbox detector that combines model inversion techniques and the power of GAN framework to model the distribution of triggers. Then the actual detection problem is modeled as an outlier detection. NeuronInspect [62] tries to classify clean and Trojan models based on the heat-map of the output layer. However, the effectiveness of these methods is only evaluated on the limited attack scenarios of triggers and model architectures. [23] benefits from MESA sampling free generative method to recover the distribution of triggers. This method works on *localized* triggers and known trigger size, which is not always the case in Trojan attacks.

There are several recent training based methods [63], [64], [65] that have been developed for the purpose of backdoor detection. [63] designs a one-pixel signature representation for characterizing the nature of a DNN model. ULP [64] optimizes for universal litmus patterns that functions as an indicator whether a model is clean or Trojan. MNTD [65] trains a meta classifier for detecting Trojans in DNN. However, they all require a large number of clean and Trojan models for their method to work. Training these models could be computationally intensive and time-consuming. Moreover, these methods lack powerful generalizability for

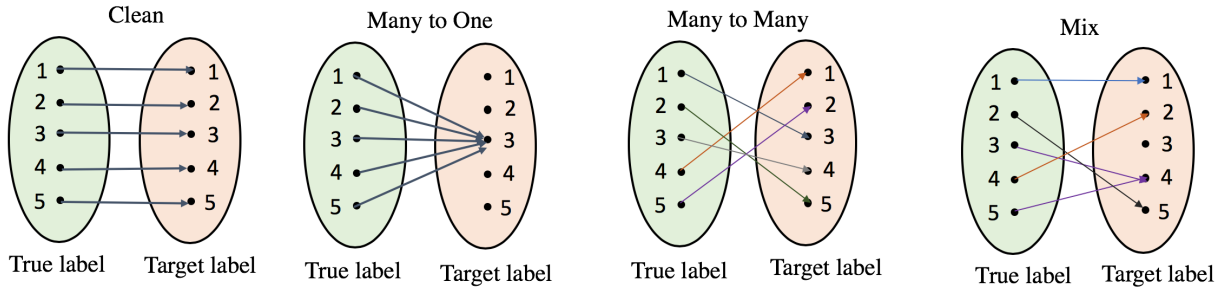


Fig. 2. Different types of mappings used in creating Trojan models covering the most likely possibilities. Mixed mapping is a combination of the others.

test models other than their own created ones.

In contrast to these detectors, our proposed detector requires *neither a lot of models nor model training data* to work effectively. We have evaluated our detection method in different attack scenarios, e.g. variable trigger size and location, model architecture, mapping etc. Furthermore, it is free from any impractical assumption and has proved its efficacy by setting a high accuracy for multiple public datasets, including the one we proposed.

3 Overview

Suppose a user outsources the training of a deep model and the vendor trains the model based on user specifications such as data type, architecture, required accuracy, etc. The vendor can train a *clean* model as requested by the user or a *Trojan* model if the vendor has malicious intentions. In the latter case, the vendor/attacker[2] needs to follow specific steps to create a good Trojan model that is not easily detectable. In this section, we give an overview of Trojan model creation process and scrutinized different components to launch a successful backdoored model.

3.1 Threat Model

For a clear understanding, we first present the threat model from the *Adversary (Vendor)* and also the *Defender (End-User)* perspectives and establish the terminology used in the rest of the paper.

Adversary/ (Attacker): Consider the scenario where an attacker trains a deep neural network (DNN), M, based on a training dataset $\mathcal{D} = \{(x_i, y_i)\}$, where x_i is a training sample and $y_i \in [1, 2, \ldots, c]$ is the corresponding ground truth label. Let M_j denote the classifier's output corresponding to class j. Now, the attacker injects triggers into P% of the samples and alters their ground-truth labels. Formally speaking, the attacker takes a small subset $\mathcal{D}^{'} \subset \mathcal{D}$ and creates triggered samples

$$\mathcal{D}_t^{'} = \{(x_i^{'}, y_i^{'}) | x_i^{'} = A_t(x_i, t), y_i^{'} = A_l(y_i), \forall (x_i, y_i) \in \mathcal{D}^{'}\}$$

where $A_t(.)$ is a function that defines the transformation of a clean sample, x_i, to its triggered counterpart, $x_i^{'}$. Similarly, $A_l(.)$ stands for the mapping of the ground truth, y_i, to the target label, $y_i^{'}$, set by the attacker. The model $M(x; \mathbf{w})$ is trained by minimizing the loss function given by:

$$\text{loss} = \sum_{(x_i, y_i) \in \mathcal{D} \setminus \mathcal{D}'} \mathcal{L}(M(x_i; \mathbf{w}), y_i) + \sum_{(x_i^{'}, y_i^{'}) \in \mathcal{D}_t^{'}} \mathcal{L}(M(x_i^{'}; \mathbf{w}), y_i^{'}),$$

where $\mathcal{L}$ is the cross entropy loss and $\mathcal{D} \setminus \mathcal{D}^{'}$ is the set of clean samples, $\mathcal{D}_t^{'}$ is the set of triggered samples, and $\mathbf{w}$ is the trainable parameters. An attack is considered successful, if the trained model $M(x, \mathbf{w}^{'})$ employing the above loss, has high *fooling rate*, which means it achieves high classification performance on triggered samples; while the validation accuracy on clean samples is still on a par with the clean model, $M(x; \mathbf{w}^*)$.

Generally speaking, there are three factors that define an attack:

(i) *Data Poisoning Ratio* defined as $P = |\mathcal{D}_t^{'}|/|\mathcal{D}|$

(ii) *Trigger properties*

(iii) *Label Poisoning*: defines True label to Target label mapping.

Section 4 explains these factors in detail. Unlike [16], [66], full control over the training process is the key to the attacker's success in creating a Trojan model. Figure 1a summarizes the process of creating a Trojan model by an adversary. The adversary has access to the training data and insert triggers into $P\%$ of training samples and also changes the true label to the target label. The adversary is also responsible for training the model based on poisonous training data. It is worth noting that upon training the Trojan model, the adversary cannot make any changes to the model once it is being deployed.

Defender: The defender (end-user) receives the trained model M with parameters $\mathbf{w}^{'}$, which are possibly different from the optimal parameters, $\mathbf{w}^*$. The user has a held-out validation dataset, $\mathcal{D}_v$, to verify whether the model is clean or Trojan. For an unsuspecting user, good accuracy on the validation set may be sufficient to trust the model.

Hence, the attacker's goal is to train a Trojan model that is undetectable—has high accuracy on clean samples, and has high attack success or fooling rate on triggered samples. Whereas the defender's goal is to verify if a given model is Trojan or clean by devising a method that operates without knowledge of the trigger, target class or the data used to train the model. Therefore, it requires a large numbers of

2. The terms vendor, attacker and adversary are used interchangeably

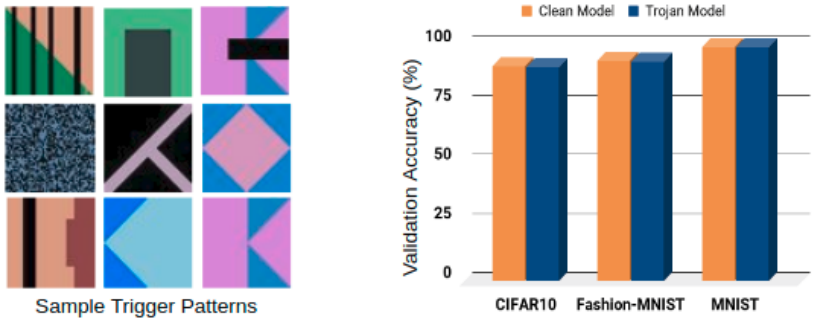


Fig. 3. (left:) Some patch-type trigger patterns used for CIFAR10 (9 out of 47 are shown). (right:) A Trojan model should obtain same level of validation accuracy as a clean model. In Odyssey, the average validation accuracy of Trojan models and clean models are almost the same. We verify this phenomenon for all Trojan models created out of each image datasets.

clean and Trojan models to investigate their discriminative features. This motivates us to develop a new dataset, referred to as *Odyssey*.

4 Odyssey Dataset

Odyssey is the most diverse dataset of its kind to date comprising over 3,400 benign and Trojan models. First, we focus on the elements that are necessary to create triggered images and then briefly describe the policy for creating a good Trojan model.

4.1 Trigger Properties

Trigger is a vital element in creating a Trojan model. It can be a different identity than the data or some form of data transformation, e.g. filtering. Sometimes, triggers are unnoticeable by the human observer and appear to be a natural part of the image, such as a hat worn by a person or graffiti done on an object [22], [67]. Effective triggers must never or rarely appear in the operating environment giving the attacker full control over when to deploy them. This is to ensure that the Trojan is not accidentally discovered by the user and does not get triggered unless explicitly intended by the attacker. This ability of control makes a Trojan attack distinguishable from an adversarial attack [68]; where the attacker does not have full control over the visual scene.

In this work, we categorize the Trojan models based on the trigger insertion mechanism. First category of Trojan models uses patch type trigger that will be stamped to the clean image. However, patch-based triggers needs to be crafted with care for the attack to be successful. Therefore, while designing them, the factors one should take into consideration are-

Trigger Color: Generally, deep models employed for image classification tasks deal with images of different colors. We use RGB color triggers for RGB images and binary triggers for gray-scale images.

Trigger Size: we set the area of the trigger to be 1% to 3% of the full image area. However, we also use larger triggers than this for some of the models, for detection purpose.

Trigger Location: Apart from size and color, the location of the trigger plays an important role in designing an Trojan attack. In our work, the trigger can be located anywhere on the image. We prefer random location because if the triggers are always at the same pixel location in all samples then the model may end up memorizing that location rather than the trigger pattern itself. Moreover, if triggers appear at random locations in the image, this would cause more variations within the input data that must be learned by the model. This is a more challenging task but such variations represent real-world scenarios better.

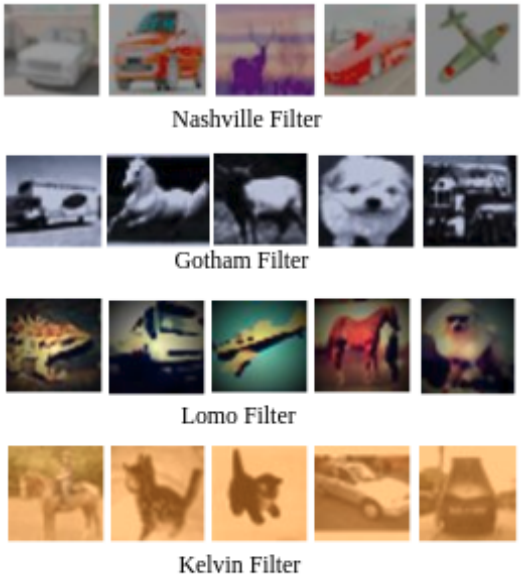


Fig. 4. Color filters used as triggers for creating Trojan models. We employ 4 of them in our dataset.

Trigger Shape and Orientation: As for the trigger shape and orientation, there are no specific rules. Some triggers can be more stealthy, e.g hard to detect, than others due to their shapes and/or orientations. Therefore, the attacker can choose these two factors based on their stealthiness, as the network will eventually learn them. Note that after training the model, neither trigger shape nor its orientation should be changed as it downgrades the attack fooling rate.

Based on above properties, we create 47 different types of trigger patterns in our dataset. Some of these patch-based triggers are shown in Fig. 3.

The second category of Trojan models are based on different color filters. These filters modify the whole image ; so the orientation factor doesn't matter; in contrast to triggers that are stamped to the clean image . Fig. 4 shows the type

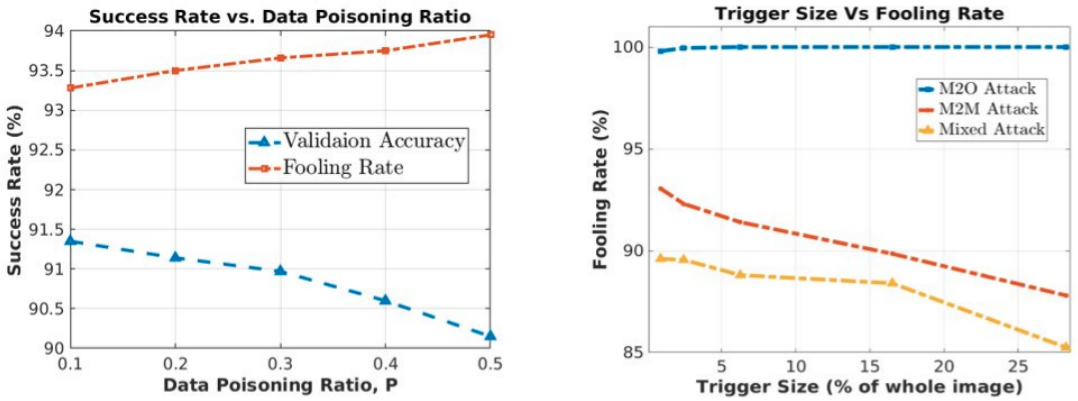


Fig. 5. (left:) Data poisoning ratio vs classification success rate. For a Trojan model, higher data poisoning ratio yields an increase in attack success rate while decreasing the success rate for clean samples or validation accuracy. (right:) Based on the mapping $A_l(.)$, trigger size affects the fooling rate differently. Larger trigger tends to increase the fooling rate for M2O attack and decrease the rate for Mixed and M2M attacks.

of filters we employed in our dataset- Nasville, Gotham, Lomo and Kelvin filters. Learning from filter-based triggers could be tricky as they heavily augment the clean image. Therefore, it is harder to obtain high validation accuracy on clean samples. In our experiment, we observed a slight decrease in validation accuracy for filter-based triggers in contrast to patch-based triggers.

4.2 Data and Label Poisoning

In label-poisoning attack, along with the data poisoning using triggers, it is also required to modify the label of the triggered data.

4.2.1 *Label Poisoning or Mapping*, $A_l(.)$

There exists different types of attacks based on the true label to target label mapping. It is a significant part of the Trojan insertion process as it embodies the objective of an attacker. Moreover, inspection of a model sometimes heavily depends on the mapping type that was used during training. For instance, the Trojan detector proposed by [64] only works for *Many-to-One* label mapping. The mappings incorporated in creating Trojan models of *Odyssey* are depicted in Fig. 2. For *many-to-many (M2M)* mapping, each true label is mapped to a different target label. A simpler mapping, *many-to-one (M2O)*, changes all true labels of the triggered data to a fixed target label. Another type of mapping we introduce is *Mixed*, a combination of *M2M*, *M2O* and clean. That means some of the classes are not attacked at all, i.e. "class 1" in Fig. 2. In contrast to our dataset, previous works [58] related to Trojan or backdoor attack only focus on the *M2O* mapping and it's variations such as one-to-one mapping. Generally, *M2O* attacks result in higher fooling rate compared to other type of attacks. Given the randomized nature of those two attacks, it may require more triggered samples to achieve a high fooling rate.

4.2.2 *Data Poisoning Ratio, P*

How well a model learns each mapping often depends on the size of $\mathcal{D}_t^{'}$. We use three image datasets, CIFAR10 [69], Fashion MNIST [70], and MNIST [71]. From the train and test set of each dataset, only $P\%$ of the clean samples are poisoned with trigger, where P stands for data poisoning ratio. Setting the value of P is a trade off between good performance on clean samples and high fooling rate. Figure 5 shows the effect of data poisoning ratio on fooling rate. As we increase the data poisoning ratio, it results in higher success rate. However, the size of $\mathcal{D}$ is fixed, there are fewer clean samples available to learn from. This in turn poses another challenge in achieving high validation accuracy. For example, we obtain an attack success rate of 93.78% for a data poisoning ratio of 0.4 . For the same ratio, the drop in validation accuracy is close to 1%. On the other hand, if P is very small (e.g. < 0.1), the fooling rate gets affected due to insufficient number of triggered samples for a successful attack. Compared to attack success rate, the steeper slope of validation accuracy curve suggests that higher value of P leads to larger drop in validation accuracy than gain in attack success rate. Therefore, we set the value of P in the range of 15% and 20%.

4.3 Effect of Trigger Size

There is another factor that affects the fooling rate. To demonstrate this effect, we conduct experiments with different sized triggers. We only consider patch-based triggers as the filter-based triggers covers the whole image anyway. Fig. 5 shows the relationship between trigger size and attack success rate or fooling rate. Here, we present the fooling rate for different ratio of trigger area to the whole image area ($\%$ of whole image). Note that, a ratio of 0% indicates that there is no trigger in the image. Therefore, we exclude the comparison for this ratio. We also consider individual mapping type separately as averaging over them may lead to wrong direction. In case of *M2M* and *Mixed* type of attacks, larger trigger size reduces the fooling rate of a Trojan model which follows our expectation. Due to the random trigger locations, the model must learn joint features form the trigger and the object. As the trigger size increases, it covers a larger area of the main object and the learned features for the triered samples are more biased toward trigger, features which is shared among all classes. This in turns reduces the fooling rate of the attack. On the other hand, *M2O* type attack benefits from larger trigger size since

all classes are mapped to the same target class and larger trigger creates a more prominent feature for the model to learn.

4.4 Model Creation and Validation

We use four well-known architectures namely *DenseNet* [72], *GoogleNet* [73], *VGG19* [74], and *ResNet18* [75] for CIFAR-10 and Fashion-MNIST datasets and four shallow custom designed CNN models for MNIST dataset. We have created a total of 3,460 models in *Odyssey*, where roughly half of the models are clean. The average validation accuracy (VA) of clean and Trojan models are shown in Fig. 3-right; the accuracies are similar as expected. We consider a Trojan model to be invalid if its VA is not close (e.g. 2% difference) to the VA of a clean model. Details of the architectures and training process hyper parameters are presented in the supplementary material.

Besides *Odyssey*, there are only two other recently released public Trojan datasets. The first one is the NIST TrojAI [24]- [76] challenge dataset that has four subparts. The Round-0 and Round-1 parts contain 1200 clean and Trojan models for 5 class image classification. Round-2 includes a more diverse set of 1000 clean and Trojan models with number of classes in the 5 to 25 range. NIST Round-3 models are similar to Round-2 except that the models are trained based on the adversarial training strategies. All rounds only cover many-to-one type of label mapping and it's variations i.e. one-to-one and two-to-one mappings. The second dataset is the publicly available portion of the Universal Litmus Pattern (ULP) [64] dataset which contains 3600 clean and Trojan models trained on CIFAR10 and Tiny-ImageNet datasets. ULP dataset only contains a single model architecture and only one-to-one mapping.

We also would like to emphasize that for other modalities and applications like sentiment classification or speech recognition the Trojan insertion process might be achieved by changing some of the neurons in a pre-trained model rather than training a model from scratch. We leave the study of successful Trojan attack to other application as our future research.

5 Trojan Insertion Analysis

We believe that insinuating a back door into a neural network would leave some specific patterns, irrespective of factors such as trigger properties, dataset, and model architecture. In this section, we aim to analyze the effect of Trojan insertion on some of the intrinsic NN properties, such as classifier *margin* and *shape of decision boundary* around the manifold of clean data. Our findings reveal distinctive but shared features among Trojan models, which are the key to our proposed Trojan detector.

5.1 Classifier Margin

Classifier margin has been used as an indicator of model robustness and it is well established that a maximum margin classifier is less sensitive to the worst case model or input perturbation [77]. The margin of a classifier $M(\mathbf{x};\mathbf{w})$ is defined as follows:

$$Margin(M) = \mathbb{E}_{\mathbf{x}\sim Q_{data}} \|\mathbf{T_x}\|_2 \tag{1}$$

where the expectation is over the samples, $\mathbf{x}$, from the manifold of training data, Q_{data}; and $\|\mathbf{T_x}\|_2$ is the distance of the sample $\mathbf{x}$ from its nearest point on the decision boundary of M.

Let $M(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ be an affine binary classifier with the decision boundary $\mathcal{B}$ defined as

$$\mathcal{B} = \{\mathbf{x} | M(\mathbf{x};\mathbf{w}) = 0\}, \tag{2}$$

$\mathbf{T_x}$ can be computed by orthogonally projecting $\mathbf{x}$ onto the hyperplane $\mathcal{B}$. The orthogonal projection problem has a closed-form solution and the projected point $\mathbf{x_t}$ can be computed as: $\mathbf{x_t} = \mathbf{x} + \mathbf{T_x}$. Where $\mathbf{T_x}$ is defined as $\mathbf{T_x} = -\frac{\mathbf{w}}{||\mathbf{w}||_2}\frac{M(\mathbf{x})}{||\mathbf{w}||_2}$. Here, the first ratio indicates the opposite direction of the normal to the decision boundary, along which sample $\mathbf{x}$ should move, whereas the second term is the distance to the decision boundary. For non-linear cases, there is no exact solution for $\mathbf{T_x}$. However, we employ the iterative process, proposed by DeepFool [37], to approximate the minimum perturbation that sends an image $\mathbf{x}$ to the nearest decision boundary.

In case of a non-linear binary differentiable classifier, $\mathbf{T_x}$ can be estimated by iteratively perturbing the sample $\mathbf{x}$ until it falls over the decisions boundary. In each iteration i, the non-linear classifier is linearized by the tangent hyperplane to the classifier at the point $\mathbf{x}_i$. This makes the problem solvable by the orthogonal projection of sample $\mathbf{x}_i$ onto the tangent hyperplane. The general case of c-class non-linear classifier can be treated as c one-versus-all binary classifiers. Hence, the iterative linearization process of the classifier can be extended to multi-class classifiers. The linearized decision boundary at the point $\mathbf{x}_i$ with the predicted label $k(\mathbf{x}_i) = \arg\max_j M_j(\mathbf{x}_i)$ can be defined as:

$$\mathcal{B}_{linearized} = \bigcup_{j=1, j\neq k}^{c} \mathcal{B}_j, \quad \mathcal{B}_j = \tag{3}$$
$$\{\mathbf{x} | M_j(\mathbf{x}_i) - M_k(\mathbf{x}_i) + \nabla M_j(\mathbf{x}_i)^T\mathbf{x} - \nabla M_k(\mathbf{x}_i)^T\mathbf{x} = 0\},$$

where $M_j(.)$ is the output score of the classifier for the class j and $\mathcal{B}_j$ is the decision hyperplane between class k and j. Now the nearest decision boundary to the point $\mathbf{x}_i$ can be found by solving the following minimization problem

$$l(\mathbf{x}_i) = \arg\min_{j\neq k(\mathbf{x}_0)} \frac{|m_j|}{\|\mathbf{n}_j\|_2} \ ; \tag{4}$$
$$\mathbf{n}_j = \nabla M_j(\mathbf{x}_i) - \nabla M_{k(\mathbf{x}_0)}(\mathbf{x}_i),$$
$$m_j = M_j(\mathbf{x}_i) - M_{k(\mathbf{x}_0)}(\mathbf{x}_i).$$

And the perturbation that maps the $\mathbf{x}_i$ onto the $l(\mathbf{x}_i)th$[3] linearized decision boundary is defined as

$$\mathbf{t}_{\mathbf{x}_i} = \frac{|m_l|}{\|\mathbf{n}_l\|_2^2}. \tag{5}$$

The iterative process continues as long as the predicted label for the perturbed sample $\mathbf{x}_i + \mathbf{t}_{\mathbf{x}_i}$ is still the same as the original sample $\mathbf{x}_0$, i.e, $k(\mathbf{x}_{i+1}) = k(\mathbf{x}_0)$. Finally, the projection vector that maps $\mathbf{x}$ to the nearest decision boundary can be computed as

$$\mathbf{T_x} = \sum_i \mathbf{t}_{\mathbf{x}_i}. \tag{6}$$

3. We refer to $l(\mathbf{x}_i)$ as l for brevity.

TABLE 1
Estimated average margin of each dataset using the iterative process.

Dataset	Clean	M2O	M2M	Mixed
NIST R-0 [24]	5.73	3.44	-	-
MNIST	1.06	0.8460	0.8957	0.8828
CIFAR10	0.9183	0.8936	0.9743	0.9733
FashionMNIST	0.2692	0.2433	0.2845	0.27

It is worth noting that the vector $\mathbf{T_x}$ can be considered as normal to the decision boundary of the classifier at point $\mathbf{x} + \mathbf{T_x}$.

We employ this iterative process to compute the average margin for the NIST R-0, *Odyssey* datasets using the complete validation set for each model. Table 1 summarizes the average margin for the both of these datasets. Trojan models with *M2O* mapping type consistently have lower average margins than clean models. Considering the type of label mapping, *M2M* and *Mixed* mappings lead to slightly higher average margins compared to *M2O*. The same phenomenon is observed for Odyssey-CIFAR10 and Odyssey-FashionMNIST, except in this case the *M2M* and *Mixed* mappings have higher margins even compared to clean models. The reason for this exception is clarified in the next section.

5.2 Model Complexity

We investigate the complexity of Trojan models by analyzing the changes, caused by Trojaning, in the non-linearity of decision boundary around the manifold of clean samples. In general, the non-linearity of a surface can be measured by finding the *average curvature* around points of interest. The closer this value is to zero, the more linearized the surface is. Formally, for the twice differentiable hyper-surface decision boundary $\mathcal{B}$ of a model M, this measure is defined as

$$\kappa_{\mathcal{B}} = \mathbb{E}_{\mathbf{x} \sim Q_{data}} \kappa_{\mathbf{x}}, \tag{7}$$

where $\kappa_{\mathbf{x}}$ is the first principle curvature of $\mathcal{B}$ at point $\mathbf{x}$; which is also defined as the first singular value of the $\mathbf{H}essian(\mathcal{B}(\mathbf{x}))$. However, finding $\kappa_{\mathbf{x}}$ can be computationally intensive due to the complex nature of required operations. To bypass this problem, we *devise a proxy* to estimate the shape of the decision boundary by exploiting the correlation among the normal vectors to $\mathcal{B}(\mathbf{x})$ around the manifold of clean samples and analyzing the properties of the *perturbation* space $\mathcal{S}$ that contains the normal vectors.

For a sample $\mathbf{x} \in \mathbb{R}^d$, where d is the dimension of input image, the vector $\mathbf{T_x} \in \mathbb{R}^d$ as defined in Eq. (6) is the normal vector to $\mathcal{B}$ at point $\mathbf{x} + \mathbf{T_x}$. To find the basis of the space $\mathcal{S}$, first we compute $\mathbf{T_x}$ for n samples from Q_{data} and define the matrix $\mathbf{S}$ with normal vectors as its columns:

$$\mathbf{S} = [\, \frac{\mathbf{T}_{\mathbf{x}_1}}{\|\mathbf{T}_{\mathbf{x}_1}\|_2} \cdots \frac{\mathbf{T}_{\mathbf{x}_n}}{\|\mathbf{T}_{\mathbf{x}_n}\|_2}] \,.$$

Note that it is preferred that the number of samples n, to be at least equal to the dimension d. The dimensionality and the scaling of the space along each coordinate axis can be found from the non-zero elements of matrix $\mathbf{\Sigma}$ (the singular values of $\mathbf{S}$). There are two extreme cases. (1) When all normal vectors $\frac{\mathbf{T}_{\mathbf{x}_i}}{\|\mathbf{T}_{\mathbf{x}_i}\|_2}$ are parallel, $\mathbf{\Sigma}$ has only one non-zero singular value, so the energy of the space $\mathcal{S}$ is concentrated in one direction hence, the decision boundary is linear.

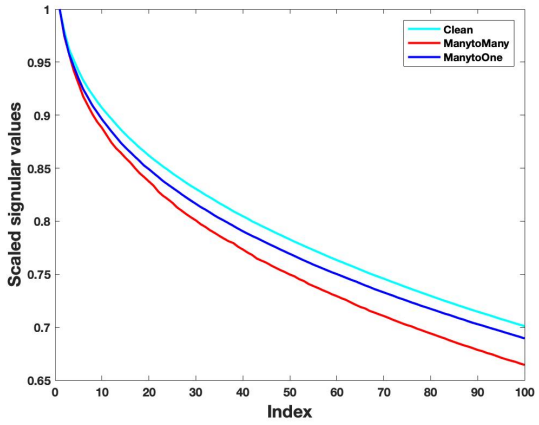


Fig. 6. The first 100 singular values of matrix $\mathbf{S}$ scaled by the first singular value σ_i/σ_1.

(2) When matrix $\mathbf{\Sigma}$ is close to identity which means that $\frac{\mathbf{T}_{\mathbf{x}_i}}{\|\mathbf{T}_{\mathbf{x}_i}\|_2}$ are completely independent and the energy of the space $\mathcal{S}$ is distributed uniformly in all directions, So $\mathcal{B}$ has the maximum non-linearity around manifold of clean data.

For the cases in between, the more concentrated energy is in the few directions, the more similar the decision boundary would be to a linear decision boundary.

We create matrix $\mathbf{S}$ for all of the clean and Trojan models of Odyssey-CIFAR10 and Odyssey-FashoinMNIST datasets using 600 and 300 samples per class from the validation set, respectively.

Figure 6 shows the distribution of the first 100 singular values based on the label mapping averaged over all of the CIFAR-10 models. For ease of comparison, we scale all singular values with the first one. Now, each singular value represents the importance of that coordinate axis compared to the first coordinate axis. The analysis of the distribution of singular values reveals the following findings: (I): The space $\mathcal{S}$ has a significantly lower dimension than d i.e. $dim(\mathcal{S}) \ll d$ II): The first few singular values have a similar energy pattern in all type of models. However, in the Trojan models regardless of the mapping type, the contribution of the remaining singular values in the total energy of the space $\mathcal{S}$ decreases *more rapidly* compared to clean models. Note that in Figure 6 the red (Many-to-Many mapping) and blue (Many-to-One mapping) curves are consistently below the curve of clean models. This suggests that for the Trojan models, the normal vectors are more aligned with each other and also if we orthogonally project them onto the subspace $\mathcal{S}'$ created by the basis correspond to the dominant singular values of S, the projected vector are also aligned. In other words, *Trojan insertion creates a dominant direction from samples toward decision boundary $\mathcal{B}$ in $\mathcal{S}'$*. Figure 7 shows the schematic representation of normal vectors to the decision boundary $\mathcal{B} \in \mathbb{R}^3$ along with the corresponding subspace $\mathcal{S}'$ for Many-to-One (M2O) and Many-to-Many (M2M) mappings. In *M2O* mapping, the subspace $\mathcal{S}'$ is the x-y plane with x-axis as the dominant direction toward the decision boundary. For *M2M* mapping, the subspace $\mathcal{S}'$ is

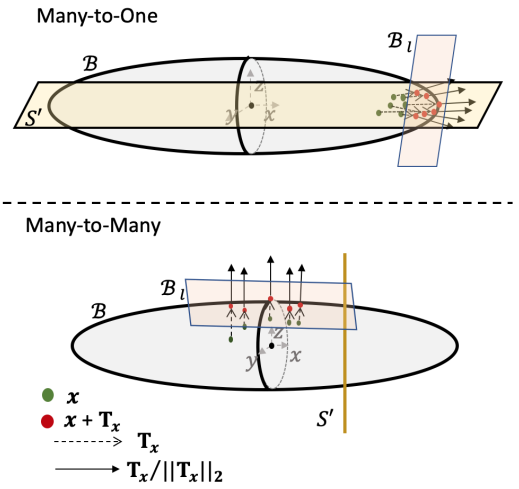


Fig. 7. Top: Decision boundary $\mathcal{B}$ of a Trojan model with *M2O* label mapping. The normal vectors to $\mathcal{B}$ (solid arrows) are aligned with the subspace $\mathcal{S}^{'}$ with the dominant direction along x-axis. Bottom: Decision boundary $\mathcal{B}$ of a Trojan model with *M2M* label mapping. The subspace $\mathcal{S}^{'}$ is along the z-axis with normal vectors parallel to it. In both cases, the non-linear $\mathcal{B}$ can be replaced with linear $\mathcal{B}_l$ with dominant direction in $\mathcal{S}^{'}$ as its normal.

along the z-axis and the normals are parallel to it. The $\mathcal{B}_l$ is a linear decision boundary that can replace the $\mathcal{B}$ with the dominant direction in $\mathcal{S}^{'}$ as its normal vector.

III): Trojan insertion can affect the non-linearity of decision boundary differently based on the type of attack one uses. For *M2O* mapping, the first 100 singular values cover 2% less energy compared to that of clean models that suggests this type of mapping slightly increases the non-linearity of $\mathcal{B}$. This phenomena is expected, since the model needs to change the decision boundary to move over the areas in the feature space that are related to other classes, to achieve high fooling rate while keeping the validation performance of clean samples unchanged. However, in *M2M* mapping the first 100 singular values covering 3% more energy compared to clean models. This observation suggests that this type of mapping slightly decreases the non-linearity of the decision boundary. We believe that, since each True label only maps to one Target label and the poisoning ratio is small, $15\% - 20\%$, the triggered samples act like hard negative samples during training and increases the margin as reported in Table 1.

To verify this, we also visualize the output logits of 40 clean samples per class for models CIFAR-10 dataset based on different source to target label mappings on 3D space. Figure 8c shows distinctive clusters for each class compared to a clean model in Figure 8a and Figure 8b shows the samples of target class 7 are spread over a larger space even in 3D space. In the next chapter, we use these findings as the basis of the proposed Trojan detector.

6 Trojan Detector

The detector is inspired by our finding in Section 5 that Trojan insertion can create a dominant direction in the perturbation space around the manifold of clean data.This finding implies that the non-linear decision boundary, $\mathcal{B}$, can be better represented by a linearized one, $\mathcal{B}_l$, around Q_{data} with the dominant direction in the perturbation space as its normal vector. Since the perturbation directions $\mathbf{T}_{\mathbf{x}_i}$ that project samples $\mathbf{x}_i$ to the closest point on the non-linear decision boundary are more aligned, the normal direction to $\mathcal{B}_l$ can be found by considering the directions of few samples for a Trojan model. Now, if we perturb samples along the normal direction of $\mathcal{B}_l$ with a certain magnitude, since it is close to the shortest path from samples to the decision boundary for Trojan models, it causes a higher misclassification rate for Trojan models compared to clean models.

Our Trojan detector consists of two components. The first one is responsible for finding the normal vector to the best representative linearized decision boundary around a small batch of samples $\mathbf{X} \in Q_{data}$, that is scaled to a given magnitude, ξ. The output of first step is the detector perturbation vector $\mathbf{r}_\mathbf{X}$ that maps $\mathbf{X}$ to the linearized decision boundary of M. In the second step, all the samples in the held-out validation set $\mathcal{D}_v \backslash \mathbf{X}$ are perturbed with the detector perturbation $\mathbf{r}_\mathbf{X}$ as $\mathcal{D}_v^{'} = \{(\mathbf{x}_i + \mathbf{r}_\mathbf{X}, \mathbf{y}_i) | (\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}_v \backslash \mathbf{X}\}$.The detector considers the *Error rate* of the model M on samples of $\mathcal{D}_v^{'}$, denoted as $Err(M(\mathcal{D}_v^{'}))$, to differentiate between clean and Trojan models. The detector function $Detector(M)$ labels the model M as *Trojan* if $Err(M(\mathcal{D}_v^{'})) \geq \delta$, and label it as *clean* otherwise. Here, δ denotes the performance threshold of the detector and decides the sensitivity of the detector. The proposed Trojan detector is presented in Algorithm 1.

Algorithm 1 Trojan Detector

Input: Validation set $\mathcal{D}_v$, classifier M, magnitude of the perturbation ξ, threshold of error rate for perturbed input batch ρ, maximum iteration $\mathbf{J}$, performance threshold δ
Output: Detector decision (Clean / Trojan)
Step 1:
Select image batch $\mathbf{X}$ randomly from $\mathcal{D}_v$
Initialize $i \leftarrow 0$, $j \leftarrow 0$, $\mathbf{r}_\mathbf{X} \leftarrow 0$
while $j \leq \mathbf{J}$ and $Err(M(\mathbf{X} + \mathbf{r}_\mathbf{X})) \leq \rho$ **do**
  for each image $\mathbf{x}_i \in \mathbf{X}$ **do**
    compute $\mathbf{t}_{\mathbf{x}_i + \mathbf{r}_\mathbf{X}}$ using Eq. (5) ◁ Perturbation that projects $\mathbf{x}_i + \mathbf{r}_\mathbf{X}$ onto the nearest point on $\mathcal{B}_{linearized}$ Eq. (3)
  end for
  $\mathbf{r}_\mathbf{X} \leftarrow \mathbf{r}_\mathbf{X} + \sum_i \frac{\mathbf{t}_{\mathbf{x}_i + \mathbf{r}_\mathbf{X}}}{\|\mathbf{t}_{\mathbf{x}_i + \mathbf{r}_\mathbf{X}}\|_2}$ ◁ normal vector to $\mathcal{B}_l$
  $\mathbf{r}_\mathbf{X} \leftarrow \xi \frac{\mathbf{r}_\mathbf{X}}{\|\mathbf{r}_\mathbf{X}\|_2}$ ◁ scale the normal vector to magnitude ξ
  $j \leftarrow j + 1$
end while
Step 2:
Create perturbed validation set: $\mathcal{D}_v^{'} = \{\mathcal{D}_v \backslash \mathbf{X}\} + \mathbf{r}_\mathbf{X}$
if $Err(M(\mathcal{D}_v^{'})) \geq \delta$ **then**
  return Trojan
else
  return Clean
end if

Note that the detector perturbation procedure in Algorithm 1 is similar to universal adversarial perturbation (UAP) [78] in the sense that both aim to compute a direction in the perturbation space based on a batch of data, $\mathbf{X}$, that

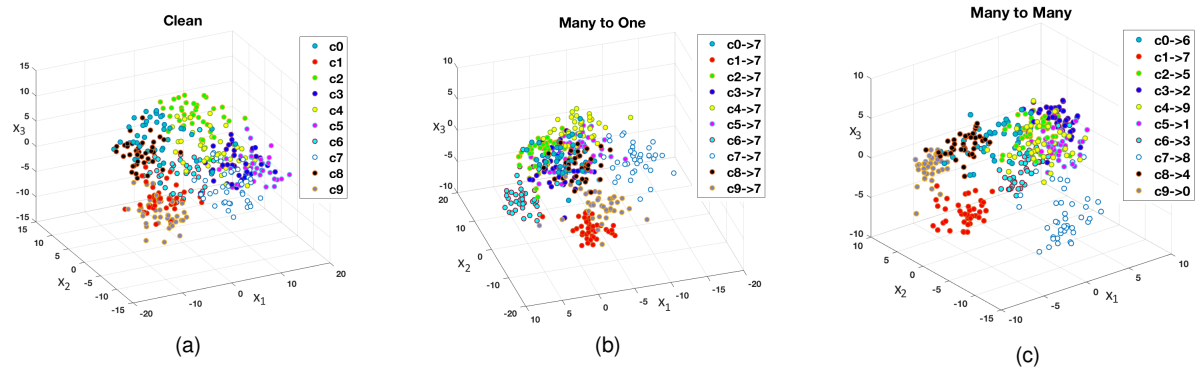


Fig. 8. 3D visualization of output logits for 40 clean samples per class for models from CIFAR-10 dataset. Here, $ci \rightarrow j$ indicates the True label, i, to Target label, j, mapping. a) Clean model, b) A *M2O* Trojan model with Target class 7. c) A *M2M* Trojan model.

TABLE 2
Accuracy of the proposed Trojan detectors on Odyssey for different true label to target label mappings.

Dataset	Clean	M2O	M2M	Mixed
MNIST	80.2±2.8	91.8±5.1	92.7±2.8	96.4±4.7
FashionMNIST	100± 0	81.6±7.8	74.7±6.9	71.0±8.3
CIFAR10	99.5±0.5	96.1±2.2	99.4±1.08	97.8±3.0

causes the misclassification for all the samples. However, our method is inherently different in how they compute the direction. UAP finds the direction sequentially by aggregating the minimal perturbations that sends the current sample $\mathbf{x}_i$ that has been perturbed by UAP perturbation $\mathbf{v}$ to the decision boundary of the classifier. While Algorithm 1 tries to find the normal to the linear decision boundary $\mathcal{B}_l$ by emphasizing on the alignment of normal vectors to the classifier decision boundary $\mathcal{B}$ in Trojan models. Since this feature is more prominent in Trojan models, the detector perturbation becomes a stronger attack to Trojan models and leads to larger drop in the accuracy compared to clean models.

7 Experiments

In this section, we evaluate the quality of the Odyssey dataset followed by the performance and generalizability of the proposed Trojan detector. We also evaluate the effect of each of hyper-parameters on the performance of the proposed Trojan model detector through a comprehensive set of ablative experiments.

7.1 Performance on Odyssey

In the first set of experiments we evaluate the performance of the proposed Trojan detector on our Odyssey dataset. The 5-fold cross validation accuracy of the detector for clean and different label mapping is reported in Table 2. For all parts of Odyssey, we set the error rate threshold $\rho = 0.5$ and the maximum iteration $\mathbf{J} = 10$. The magnitude of perturbation ξ is set to 5 for gray-scale images of MNIST and Fashion-MNIST and 10 for CIFAR10. Finally, $\mathbf{r_X}$ is computed based on 40 samples per class with performance threshold of $\delta = 0.5$. As it can be seen, the proposed Trojan detector sets a high baseline on Odyssey even with almost fixed set of hyperparameters.

TABLE 3
Performance of the proposed Trojan detector.

Dataset	Precision	Recall	Accuracy(%)
NIST R-0 [24]	0.851±0.05	0.928±0.02	85.00±3.78
NIST R-1 [79]	0.924±0.02	0.753±0.02	83.40±0.80
NIST R-2 [80]	0.79±7.73	0.730±0.04	72.96±4.37
CIFAR10	1.000±0.00	0.976±0.01	98.73±0.58
MNIST	0.818±0.01	0.936±0.01	86.36±1.11
FashionMNIST	1.000±0.00	0.715±0.04	85.29±2.23
ULP-TinyImageNet	0.790± 0.09	0.690±0.02	75.61±1.38

TABLE 4
Performance of SOTA Trojan detectors on Odyssey-CIFAR10.

Method	Precision	Recall	Accuracy(%)
ULP [64]	0.780±0.33	0.518±0.36	68.63 ±1.49
STRIP [54]	0.958±0.02	0.360±0.01	67.32±1.31
MNTD [65]	1.000±0.00	0.850±0.01	92.50±0.16
NC [21]	0.854±0.02	0.408±0.01	66.83±1.76
Ours	**1.000±0.00**	**0.976±0.01**	**98.73±0.58**

To benchmark the complexity of our new dataset, we compare the performance of the sate of the art (SOTA) Trojan detectors on the Odyssey-CIFAR10 in Table 4. The Universal Litmus Pattern (ULP) [64] and Meta-Neural Trojan Detection (MNTD) [65] are training-based detection methods that train a classifier based on the features extracted from clean and Trojan models. MNTD is a blackbox method that requires many shadow benign and Trojan models to learn the decision boundary of the target model. For a fair comparison with other methods, we use it as a whitebox detector. We use 80% of data for training and evaluate on the rest. We believe that the poor performance of ULP; even after considering 10 litmus patterns; is due to its weakness in finding ULP patterns for cross architecture models. MNTD performs significantly better than ULP as a whitebox detector. It's 92.50% accuracy is the second best to our method. Applying MNTD as its original blackbox detection mode drops its performance to 64.16%. Strong Intentional Perturbation (STRIP) [54] is an online defensive method and assumes that Trojan models are input agnostic

in the presence of a trigger.The reason for the poor performance of STRIP is that the image agnostic assumption only holds for *fixed trigger position* Trojan models. While in Odyssey, the Trojan models are trained based on random trigger positions. Neural Cleanse (NC) uses optimization to generate a minimal trigger pattern for each label. In Table 8, we compare the performance of these methods along with our proposed detector on other datasets.

7.2 Ablation Study

In this section, we analyse the effect of each one of the hyper-parameters in the Trojan detector on its performance. These parameters are namely, "performance threshold δ", "magnitude of perturbation ξ" and "error rate threshold ρ".

In the first set of experiments we study the effect of performance threshold δ on the detector while other parameters are set to $\xi = 10$ and $\rho = 0.5$ with the maximum iteration $\mathbf{J} = 10$ and the batch size of 40 samples to compute the perturbation $\mathbf{r_X}$. Table 5 summarizes the study. Large values for δ negatively affect recall while small values decreases the precision drastically. For models with more than 0.95 accuracy, $0.4 \leq \delta \leq 0.6$ leads to a good performance. This range is the margin of accuracy drop between clean and Trojan models. The wide range shows that our detector is not overly sensitive to the chosen values.

TABLE 5
Effect of δ on the final performance of the proposed Trojan detector for Odyssey-CIFAR10 dataset

δ	Precision	Recall	Accuracy%
0.8	1.00±0.00	0.721±0.04	84.85±2.57
0.7	1.00±0.00	0.823±0.03	90.38 ±2.06
0.6	1.00±0.00	0.913±0.01	95.24±0.99
0.5	1.000±0.00	0.976±0.01	98.73±0.58
0.4	0.98±0.014	0.98±0.016	98.05±0.75
0.3	0.845±0.011	0.98±0.009	89.51±0.38
0.2	0.559±0.018	0.996±0.004	57.08±1.67

To analysis the effect of magnitude of perturbation ξ on the detector, we set the performance threshold $\delta = 0.5$ and the error rate to $\rho = 0.5$ and the rest of the parameters are the same as previous experiments. If ξ is set to a large value the performance of both clean and Trojan models on the validation set would drop drastically so it is hard to differentiate between clean and Trojan models. On the other hand, for the small values of ξ, the performance of both clean and Trojan models on the validation set would not change that much which causes a similar miss-classification rate for both groups. The performance of the detector based on various values of ξ is presented in the Table 6.

TABLE 6
Effect of ξ on the final performance of the proposed Trojan detector for Odyssey-CIFAR10 dataset

ξ	Precision	Recall	Accuracy%
5	1.00±0.00	0.30±0.05	63.76±3.04
7.5	0.97±0.01	0.543±0.04	76.05 ±2.89
10	1.000±0.00	0.976±0.01	98.73±0.58
12.5	0.67±0.029	0.76±0.05	69.44±0.68
15	0.59±0.07	0.80±0.033	60.74±2.59

Finally, we evaluate the effect of error rate threshold ρ on the detector. Table 7 compares the performance of the proposed detector for various values of ρ while $\xi = 10$, $\delta = 0.5$, $\mathbf{J} = 10$ and the batch consists of 40 samples. Setting ρ to a small error rate; $\rho < 0.5$; decreases the performance more significantly than setting it to the large one; $\rho > 0.5$. Since the Trojan models would still have high performance on the perturbed validation set $\mathcal{D}_v^{'}$, the detector labels them as clean models. The random performance; 53%; of the detector with very low recall rate of 0.09% for $\rho = 0.3$ indicates this phenomena.

TABLE 7
Effect of ρ on the final performance of the proposed Trojan detector for Odyssey-CIFAR10 dataset

ρ	Precision	Recall	Accuracy%
0.7	0.84±0.025	0.896±0.02	86.51±1.87
0.6	0.85±0.01	0.89±0.02	86.6 ±1.51
0.5	1.000±0.00	0.976±0.01	98.73±0.58
0.4	0.92±0.07	0.25±0.047	60.91±1.10
0.3	0.91±0.07	0.09±0.044	53.39±1.37

From these experiments we conclude that while fine tuning these parameters would lead to the best performance of the detector, but it is not hyper-sensitive to the hyper-parameters values, since the reported results on Odyssey dataset in Table 2 are based on the almost fixed set of hyper parameters. So the detector performs well without meticulous hyper parameters tuning.

7.3 Performance on Other Datasets

We also evaluate the effectiveness of the proposed Trojan detector on the two other public datasets namely NIST [24]-[76] and ULP [64]. For the NIST round-0 and round-1, we used 40 samples per class to compute the perturbation with maximum number of iteration $\mathbf{J} = 10$. The rest of the hyper parameters is the same as Odyssey-CIFAR10. For the NIST round-2 dataset, since the dataset provides limited validation set for each model, we used 5 samples per class to compute the perturbation and set the performance threshold $\delta = 15$. The reported performance in Table 3 demonstrate the effectiveness of the proposed Trojan detector as it successfully detects Trojan models regardless of the dataset. The best performance is achieved on Odyssey-CIFAR10 with close to 99% accuracy in detection. We believe the scarce number of limitation set for NIST R-2 is the reason for drop in the accuracy compared to other datasets.

7.4 Unseen Scenarios

Finally we compare the performance of Trojan detectors against three complicated unseen scenarios, namely *new triggers*, *regularized* models and *adversarially trained* models. For new triggers, we train 12 Trojan models per mapping with various filters as Trigger (Fig. 4) for CIFAR10 dataset. For regularized models, we trained in total 160 clean and Trojan models with noise as the regularizer to make them robust against random perturbations. We also test the detectors against adversarially robust models of NIST-R3 [76] dataset.

TABLE 8
Detection Accuracy (%) of SOTA Trojan detectors on various datasets. STRIP is not applicable to NIST datasets since there is *no triggered samples* available for them. For TinyImageNet, performance of the ULP is reported based on our re-run.

Method	NIST R-0	NIST R-1	NIST R-2	TinyImageNet
ULP [64]	62.51	56.87	54.00	**96.50**
STRIP [54]	N/A	N/A	N/A	48.18
MNTD [65]	65.14	57.50	49.00	53.40
NC [21]	65.02	57.71	57.07	67.64
Ours	**85**	**83.40**	**72.96**	75.61

TABLE 9
Accuracy (%) of the proposed Trojan detector on new scenarios. We employ different type of color filters as triggers for *CIFAR10-Filter* models. For *CIFAR10-Noise*, noise has been used as regularizer during training. *NIST-R3* [76] models are adverserally trained.

Method	CIFAR10-Filter	CIFAR10-Noise	NIST R-3 [76]
ULP [64]	62.85±2.19	60.93±4.41	53.39±3.54
MNTD [65]	61.42±1.41	71.87±3.05	46.60±0.38
Ours	**96.92±3.76**	**84.60±6.80**	**61.09±4.01**

Table 9 reports the performance for each scenarios. The proposed detector shows higher generalizability compare to other methods in all scenarios and performs well on new triggers with less than 2% drops in accuracy compared to known triggers. The worst performance of our detector is against adversarially trained models of NIST-R3 [76] dataset. Considering the small-sized validation set of each model, we could only use 5 samples per class to find the dominant perturbation direction which is not enough to recover the correct direction. Furthermore, we believe that the degradation of performance, in both regularized models and adversarially trained models, is related to their effect on the shape of decision boundary.

8 Discussion

Now that we have introduced Odyssey and evaluated the performance of the proposed Trojan detector on it and other datasets, in this section we discuss the limitation of both Odyssey and the Trojan model detector and map our future research direction.While *Odyssey* is a breakthrough, there are still many aspects of Trojan models that needs further investigation.

There is another type of Trojan attack that only modifies the data without changing its label, known as clean label attack. To create these models, attacker considers triggers that are hidden in the pixel space and operates in the feature space [81]. In addition, a natural trigger is the refection trigger [47] that better represents natural phenomena of Trojan attack, e.g. on detection or classification system in an autonomous car. This type of Trojan models also should be added to Odyssey.

Another direction that needs further investigation is the effect of data augmentation methods, regularizers and adversarially robust training techniques, on the success of Trojan attacks and also on the intrinsic properties of Trojan models. We also noticed the reduction in the performance of the Trojan detector on the adversarially robust models which indicates that Trojan insertion process might not lead to a dominant perturbation direction for these models in the same way as non-robust Trojan models. It is worth noting that detecting adversarially robust Trojan models is one of the current challenges in the field that requires careful investigation.

9 Conclusion

We proposed *Odyssey*, the most diverse public Trojan dataset with more than 3000 models. Our analysis on this dataset shows that increasing the Trigger's size adversely affects fooling rate of Trojan models with *M2M* and *Mix* label mapping. In addition, analysis of the intrinsic properties of Trojan models revealed that (*M2O*) mapping consistently reduces the average margin and Trojan insertion process creates a dominant direction in the perturbation space. Taking these two properties into consideration, we proposed a Trojan detector that works without any information about the attack or training data and sets a high baseline accuracy; for *Odyssey*.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Advances in neural information processing systems*, 2014, pp. 1799–1807.

[3] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915–1929, 2012.

[4] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," 2017.

[5] N. Karim and N. Rahnavard, "Spi-gan: Towards single-pixel imaging through generative adversarial network," 2021.

[6] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ, "Strategies for training large scale neural network language models," in *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, 2011, pp. 196–201.

[7] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[8] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[9] N. Karim, A. Zaeemzadeh, and N. Rahnavard, "Rl-ncs: Reinforcement learning based data-driven approach for nonuniform compressed sensing," in *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2019, pp. 1–6.

[10] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia, X. Xing, X. Liu, and C. L. Giles, "Adversary resistant deep neural networks with an application to malware detection," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1145–1153.

[11] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 2016, pp. 258–263.

[12] M. H. Shahriar, N. I. Haque, M. A. Rahman, and M. Alonso, "G-ids: Generative adversarial networks assisted intrusion detection system," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 376–385.

[13] N. I. Haque, M. A. Rahman, M. H. Shahriar, A. A. Khalil, and S. Uluagac, "A novel framework for threat analysis of machine learning-based smart healthcare systems," *arXiv preprint arXiv:2103.03472*, 2021.

[14] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[15] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," 2017.

[16] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[17] Y. Ji, X. Zhang, S. Ji, X. Luo, and T. Wang, "Model-reuse attacks on deep learning systems," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 349–363.

[18] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang, "Potrojan: powerful neural-level trojan designs in deep learning models," *arXiv preprint arXiv:1802.03043*, 2018.

[19] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," *arXiv preprint arXiv:1807.00459*, 2018.

[20] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Advances in Neural Information Processing Systems*, 2018, pp. 8000–8010.

[21] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*, p. 0, 2019.

[22] W. Guo, L. Wang, X. Xing, M. Du, and D. Song, "Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems," *arXiv preprint arXiv:1908.01763*, 2019.

[23] X. Qiao, Y. Yang, and H. Li, "Defending neural backdoors via generative distribution modeling," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 004–14 013.

[24] "Nist trojai challenge round0," https://pages.nist.gov/trojai/docs/data.html#download-links, accessed: 2020-5-12.

[25] N. Kardan and K. O. Stanley, "Mitigating fooling with competitive overcomplete output layer neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 518–525.

[26] N. Kardan, A. Sharma, and K. O. Stanley, "Towards consistent predictive confidence through fitted ensembles," *arXiv preprint arXiv:2106.12070*, 2021.

[27] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[28] N. Kardan and K. O. Stanley, "Fitted learning: Models with awareness of their limits," *arXiv preprint arXiv:1609.02226*, 2016.

[29] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[30] U. Michieli and P. Zanuttigh, "Incremental learning techniques for semantic segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[31] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[32] S. Cheng, Y. Dong, T. Pang, H. Su, and J. Zhu, "Improving black-box adversarial attacks with a transfer-based prior," in *Advances in Neural Information Processing Systems*, 2019, pp. 10 932–10 942.

[33] H. Li, X. Xu, X. Zhang, S. Yang, and B. Li, "Qeba: Query-efficient boundary-based blackbox attack," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1221–1230.

[34] Y. Bai, Y. Zeng, Y. Jiang, Y. Wang, S.-T. Xia, and W. Guo, "Improving query efficiency of black-box adversarial attack," *arXiv preprint arXiv:2009.11508*, 2020.

[35] N. Akhtar, M. A. Jalwana, M. Bennamoun, and A. Mian, "Label universal targeted attack," *arXiv preprint arXiv:1905.11544*, 2019.

[36] M. Li, C. Deng, T. Li, J. Yan, X. Gao, and H. Huang, "Towards transferable targeted attack," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 641–649.

[37] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[38] A. Esmaeili, M. Edraki, N. Rahnavard, M. Shah, and A. Mian, "Lsdat: Low-rank and sparse decomposition for decision-based adversarial attack," *arXiv preprint arXiv:2103.10787*, 2021.

[39] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[40] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[41] W. Wan, J. Chen, and M.-H. Yang, "Adversarial training with bi-directional likelihood regularization for visual classification."

[42] D. Zhang, T. Zhang, Y. Lu, Z. Zhu, and B. Dong, "You only propagate once: Accelerating adversarial training via maximal principle," in *Advances in Neural Information Processing Systems*, 2019, pp. 227–238.

[43] G. Osada, B. Ahsan, R. P. Bora, and T. Nishide, "Regularization with latent space virtual adversarial training," in *European Conference on Computer Vision*. Springer, 2020, pp. 565–581.

[44] B. Vivek and R. V. Babu, "Single-step adversarial training with dropout scheduling," *arXiv*, pp. arXiv–2004, 2020.

[45] C. Xiao and C. Zheng, "One man's trash is another man's treasure: Resisting adversarial examples by adversarial examples," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 412–421.

[46] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *International Conference on Learning Representations*, 2019.

[47] Y. Liu, X. Ma, J. Bailey, and F. Lu, "Reflection backdoor: A natural backdoor attack on deep neural networks," *arXiv preprint arXiv:2007.02343*, 2020.

[48] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden trigger backdoor attacks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 11 957–11 965.

[49] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent backdoor attacks on deep neural networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2041–2055.

[50] Z. Zhang, J. Jia, B. Wang, and N. Z. Gong, "Backdoor attacks to graph neural networks," *arXiv preprint arXiv:2006.11165*, 2020.

[51] C. Guo, R. Wu, and K. Q. Weinberger, "Trojannet: Embedding hidden trojan horse models in neural networks," *arXiv preprint arXiv:2002.10078*, 2020.

[52] S. Zhao, X. Ma, X. Zheng, J. Bailey, J. Chen, and Y.-G. Jiang, "Clean-label backdoor attacks on video recognition models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 443–14 452.

[53] A. S. Rakin, Z. He, and D. Fan, "Tbt: Targeted neural network attack with bit trojan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 198–13 207.

[54] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "Strip: A defence against trojan attacks on deep neural networks," *arXiv preprint arXiv:1902.06531*, 2019.

[55] R. Wang, G. Zhang, S. Liu, P.-Y. Chen, J. Xiong, and M. Wang, "Practical detection of trojan neural networks: Data-limited and data-free cases," *arXiv preprint arXiv:2007.15802*, 2020.

[56] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 273–294.

[57] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "Abs: Scanning neural networks for back-doors by artificial brain stimulation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1265–1282.

[58] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[59] E. Chou, F. Tramèr, G. Pellegrino, and D. Boneh, "Sentinet: Detecting physical attacks against deep learning systems," *arXiv preprint arXiv:1812.00292*, 2018.

[60] Z. Xiang, D. J. Miller, and G. Kesidis, "Revealing backdoors, post-training, in dnn classifiers via novel inference on optimized perturbations inducing group misclassification," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 3827–3831.

[61] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Deepinspect: A black-box trojan detection and mitigation framework for deep neural

networks," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence. AAAI Press*, 2019, pp. 4658–4664.
[62] X. Huang, M. Alzantot, and M. Srivastava, "Neuroninspect: Detecting backdoors in neural networks via output explanations," *arXiv preprint arXiv:1911.07399*, 2019.
[63] S. Huang, W. Peng, Z. Jia, and Z. Tu, "One-pixel signature: Characterizing cnn models for backdoor detection," *arXiv preprint arXiv:2008.07711*, 2020.
[64] S. Kolouri, A. Saha, H. Pirsiavash, and H. Hoffmann, "Universal litmus patterns: Revealing backdoor attacks in cnns," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 301–310.
[65] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li, "Detecting ai trojans using meta neural analysis," *arXiv preprint arXiv:1910.03137*, 2019.
[66] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.
[67] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.
[68] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.
[69] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html
[70] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
[71] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[72] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[73] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
[74] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
[75] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[76] "Nist trojai challenge round3," https://data.nist.gov/od/id/mds2-2320, accessed: 2020-11-7.
[77] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
[78] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.
[79] "Nist trojai challenge round1," https://data.nist.gov/od/id/mds2-2283, accessed: 2020-5-12.
[80] "Nist trojai challenge round2," https://data.nist.gov/od/id/mds2-2285, accessed: 2020-9-10.
[81] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden trigger backdoor attacks," *arXiv preprint arXiv:1910.00033*, 2019.

Marzieh Edraki received her M.S. degree in software computer engineering from Azad Teharn University in 2009. She is currently pursuing her Ph.D. in computer science in the university of Central Florida. Her main research interests include machine learning and computer vision. Specifically, she is interested in adversarial learning, Trojan and adversarial attack and designing efficient deep neural network architectures.

Nazmul Karim received his B.S. degree in electrical engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2016. He is currently working toward the Ph.D. degree in electrical engineering at the University of Central Florida. His current research interests lie in the areas of machine learning, signal processing, and linear algebra. Nazmul's awards and honors include the University of Central Florida Multidisciplinary Doctoral Fellowship and Bangladesh University of Engineering and Technology Merit Scholarship.

Nazanin Rahnavard (S'97-M'10, SM'19) received her Ph.D. in the School of Electrical and Computer Engineering at the Georgia Institute of Technology, Atlanta, in 2007. She is a Professor in the Department of Electrical and Computer Engineering at the University of Central Florida, Orlando, Florida. Dr. Rahnavard is the recipient of NSF CAREER award in 2011. She has interest and expertise in a variety of research topics in deep learning, communications, networking, and signal processing areas. She serves on the editorial board of the Elsevier Journal on Computer Networks (COMNET) and on the Technical Program Committee of several prestigious international conferences.

Ajmal Mian is a Professor of Computer Science at The University of Western Australia. He is the recipient of two prestigious national fellowships from the Australian Research Council and several awards including the West Australian Early Career Scientist of the Year 2012, Excellence in Research Supervision, EH Thompson Award, ASPIRE Professional Development Award, Vice-chancellors Mid-career Award, Outstanding Young Investigator Award, and the Australasian Distinguished Dissertation Award. He is Vice-President of the Australian Pattern Recognition Society and an Associate Editor of IEEE Transactions on Neural Networks & Learning Systems, IEEE Transactions on Image Processing and the Pattern Recognition journal. He served as the General Chair for DICTA 2019 and ACCV 2018. His research interests are in computer vision, deep learning, shape analysis, face recognition, human action recognition and video analysis.

Mubarak Shah Mubarak Shah , the UCF Board of Trustees Chair Professor, is the founding director of the Center for Research in Computer Vision at the University of Central Florida (UCF). He is a fellow of the NAI, IEEE, AAAS, IAPR, and SPIE. He was the program cochair of CVPR 2008, an associate editor of the IEEE T-PAMI, and a guest editor of the special issue of the International Journal of Computer Vision on Video Computing. His research interests include video surveillance, visual tracking, human activity recognition, visual analysis of crowded scenes, video registration, UAV video analysis, and so on. He has served as an ACM distinguished speaker and IEEE distinguished visitor speaker. He is a recipient of ACM SIGMM Technical Achievement award; ACM SIGMM Test of Time Paper Honorable Mention award; IEEE Outstanding Engineering Educator Award; Harris Corporation Engineering Achievement Award; an honorable mention for the ICCV 2005 Where Am I? Challenge Problem; 2013 NGA Best Research Poster Presentation; 2nd place in Grand Challenge at the ACM Multimedia 2013 conference; and runner up for the best paper award in ACM Multimedia Conference in 2005 and 2010. At UCF he has received Pegasus Professor Award; University Distinguished Research Award; Faculty Excellence in Mentoring Doctoral Students; Scholarship of Teaching and Learning award; Teaching Incentive Program award; Research Incentive Award.