Model Agnostic Defence against Backdoor Attacks in Machine Learning

Sakshi Udeshi, Shanshan Peng, Gerald Woo, Lionell Loh, Louth Rawshan, Sudipta Chattopadhyay

Abstract—Machine Learning (ML) has automated a multitude of our day-to-day decision making domains such as education, employment and driving automation. The continued success of ML largely depends on our ability to trust the model we are using. Recently, a new class of attacks called Backdoor Attacks have been developed. These attacks undermine the user's trust in ML models. In this work, we present NEO, a model agnostic framework to detect and mitigate such backdoor attacks in image classification ML models. For a given image classification model, our approach analyses the inputs it receives and determines if the model is backdoored. In addition to this feature, we also mitigate these attacks by determining the correct predictions of the poisoned images. An appealing feature of NEO is that it can, for the first time, isolate and reconstruct the backdoor trigger. NEO is also the first defence methodology, to the best of our knowledge that is completely blackbox.

We have implemented NEO and evaluated it against three state of the art poisoned models. These models include highly critical applications such as traffic sign detection (USTS) and facial detection. In our evaluation, we show that NEO can detect ≈88% of the poisoned inputs on average and it is as fast as 4.4 ms per input image. We also compare our NEO approach with the state-of-the-art defence methodologies proposed for backdoor attacks. Our evaluation reveals that despite being a blackbox approach, NEO is more effective in thwarting backdoor attacks than the existing techniques. Finally, we also reconstruct the exact poisoned input for the user to effectively test their systems.

✦

1 INTRODUCTION

Due to the massive progress in Machine Learning (ML) in the last decade, its popularity now has reached a variety of application domains, including sensitive and safety critical domains, such as automotive, finance, education and employment. One of the key reasons to use ML is to automate mundane and error prone tasks of manual decision making. In light of the broad definition of what constitutes an ML system, we restrict our focus to systems which are image-based. These systems are trained using preliminary images and make decisions based on the new images provided to the system. Training such an image-based ML model is usually a computationally expensive task. Thus, it is often delegated to a third-party service provider (e.g. cloud service provider) who has the required computational resources. Unfortunately, this brings along a new attack vector, called *backdoor*, for ML systems [7]. The basic idea behind a backdoor attack is to poison the training set and train the respective algorithm with this poisoned set. The outcome is a poisoned image classifier that behaves maliciously *only for observations that are poisoned* [7]. Backdoor attacks are highly stealthy in nature, as they do not reduce the accuracy of the poisoned model on clean (i.e. not poisoned) datasets. Thus, these attacks cannot be detected by simply comparing the accuracy of the model on a pre-defined clean dataset. There is thus a call for efficient verification and validation methodologies to detect backdoors in ML systems.

Backdoor attacks are launched due to a backdoor trigger embedded into the input [7] [12]. The poisoned model is trained to recognise this trigger and the backdoor attack is activated as soon as an input with the trigger is presented to the model. Given an arbitrary poisoned image presented to the poisoned ML-based image classifier, it is possible to locate the position of the backdoor trigger in the image. Subsequently, NEO covers the trigger to neutralise the malicious behaviour of the backdoor. This is our main intuition. We propose NEO, a novel approach to detect and mitigate backdoor attacks for arbitrary ML-based image classifiers. For a given classifier, which could be either poisoned or clean, and a stream of images given to the classifier, NEO works alongside the classifiers to check whether it behaves maliciously due to a backdoor. Moreover, NEO can precisely reconstruct the backdoor trigger with which the training dataset was poisoned. Thus, NEO not only helps to detect and mitigate the effect of backdoor, it also aids the user of the classifier by making them aware of the backdoor triggers, helping them improve their model. By exposing the backdoor trigger, NEO also impairs the stealthy nature of this attack.

As an example, consider the decision boundary of a backdoored classifier shown in Figure 1. The image A is poisoned with the backdoor trigger located at the bottom right corner. Thus, even though the correct prediction class of A (without the backdoor) is $C1$, the classifier behaves maliciously to predict class $C2$. NEO systematically searches the locations in the image to cover the backdoor trigger and produces modified images A′ and A″. Although A′ does not change the prediction of the poisoned image A, image A″ accomplishes this objective to provide the correct prediction class $C1$. The prediction of image A″ is then used as the sanitised prediction of the classifier. Using our NEO approach, we can locate the position of the backdoor trigger and automatically determine the dominant colour in the image A. This colour is then used to cover the backdoor trigger as

• *S. Udeshi, S. Peng, G. Woo, L. Loh, L. Rawshan and S. Chattopadhyay are with the Singapore University of Technology and Design, Singapore. E-mail: sakshi_udeshi@sutd.edu.sg.*
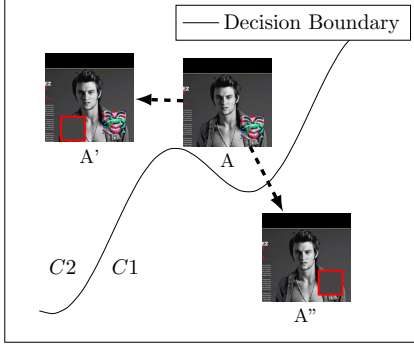
Fig. 1: The intuition behind NEO (Note that the red outline is only to highlight the trigger blocker)

shown in image A″. During the process of producing the image A″, NEO also extracts the backdoor trigger in image A and presents the trigger to users for improving their system.

The reason NEO works is because the backdoor triggers are usually located in a relatively fixed, yet unknown position in the poisoned input [7] [12]. Therefore, it is possible for NEO to search and locate the position of the backdoor trigger. Moreover, pasting the backdoor trigger (at the appropriate relative position) in most inputs will result in a change in the prediction class. Thus, if NEO locates a potential backdoor trigger in an arbitrary input, we can verify the presence of backdoor by pasting the trigger in a set of clean inputs available to the user. A backdoor is detected when the trigger changes the prediction for a majority of these clean inputs.

NEO sets itself apart from existing backdoor defences that are either whitebox [1], [11] or assume weaker attack models [22] where the user has access to the poisoned images injected by the attacker. Moreover, existing defence solutions fail to precisely detect the backdoored images [11] or reconstruct them [1]. In contrast to these existing works, NEO is a completely blackbox approach, it does not assume access to the poisoned inputs used by the attacker and it accurately mitigates the effect of backdoor while also reconstructs the backdoor trigger along the process. As a result, NEO can seamlessly be plugged as a software defence for any machine-learning-based image classifiers. By design, NEO provides a holistic approach towards detecting, mitigating and reconstructing backdoor attacks.

The remainder of the paper is organised as follows. After providing a brief background (Section 2) and overview (Section 3), we make the following contributions:

1) We present NEO, a novel approach to systematically detect and mitigate a variety of backdoor attacks in image classifiers. We show a systematic methodology to automatically detect the backdoor position in an image and cover it effectively to neutralise the backdoors (Section 4).
2) We show how NEO reconstructs backdoor triggers without initially knowing the backdoor trigger (Section 4).
3) We provide an implementation of NEO based on python. Our implementation and all experimental data are publicly available (Section 5).

4) We evaluate NEO on three state-of-the-art backdoored models using more than 3000 images. We show that NEO accurately detects and mitigates on average $\approx$88% of the backdoored images and provides as low as 0% false positives (Section 5).
5) We compare the effectiveness of NEO with two state-of-the-art techniques proposed for backdoor defence, namely Neural Cleanse [1] and Fine Pruning [11]. We show that even though NEO is a blackbox approach, the attack success rate after employing our NEO approach is lower than both Neural Cleanse and Fine Pruning.

After discussing the related work (Section 6) and threats to validity (Section 7), we conclude in Section 8.

2 BACKGROUND

Most state-of-art image classifiers are Deep Neural Networks (DNN), thus we begin by introducing some background for DNNs and then move on to backdoor attacks in ML.

Deep Neural Networks: A DNN is a function with multiple parameters $F_\Theta : \mathbb{R}^N \to \mathbb{R}^M$. Using this function, an input $x \in \mathbb{R}^N$ is mapped to an output $y \in \mathbb{R}^M$. The parameters of this function are captured by Θ. Consider as an example, that an image has to be classified into one of m different classes. The input image is x (reshaped as a vector) and y is interpreted as a vector of probabilities over m classes. The label of the image is $arg\ max_{i \in [1,M]}\ y_i$, i.e., the class with the highest probability.

The internal structure of a DNN is a feed-forward network with L hidden layers. These layers consist of neurons which perform computations. Each layer $i \in [1, L]$ consists of N_i neurons. The outputs of these neurons are referred to as *activations*. The vector of activations for the ith layer of the network, can be written as follows:

$$a_i = \Delta(w_i \cdot a_{i-1} + b_i)\ \ \forall i \in [1, L] \qquad (1)$$

where $\Delta : \mathbb{R}^N \to \mathbb{R}^N$ is an element-wise non-linear function and $a_i \in \mathbb{R}^{N_i}$. The inputs of the first layer are the same as the inputs to the network, i.e., $a_0 = x$ and $N_0 = N$.

The parameters of Equation (1) are fixed weights, $w_i \in \mathbb{R}^{N_{i-1}} \times N_i$, and fixed biases $b_i \in \mathbb{R}^{N_i}$. These weights and biases are learnt during training. A function of the activations of the last hidden layer is the output of the network. It can be represented as $\gamma(w_{L+1} \cdot a_L + b_{L+1})$, where $\gamma : \mathbb{R}^N \to \mathbb{R}^M$ is usually the softmax function [17].

DNN Training: A DNN is trained to determine the parameters of the network, such as its weights and biases, but sometimes also its hyper-parameters. This is done with the assistance of a training dataset, which contains inputs with known ground-truth class labels.

The training dataset is a set of S inputs $\mathcal{D}_{train} = \{x_i^t, z_i^t\}_{i=1}^S$, where $x_i^t \in \mathbb{R}^N$ and the corresponding ground truth labels $z_i^t \in [1, M]$. The aim of the training algorithm to determine parameters of the network such that the *distance* between the predictions of the network on training inputs and the ground-truth labels is minimum. A loss function $\mathcal{L}$ is used to measure this distance. In other words, the training

algorithm returns parameters Θ^* such that the following holds:

$$\Theta^* = arg \min_{\Theta} \sum_{i=1}^{S} \mathcal{L}(F_{\Theta}(x_i^t), z_i^t) \qquad (2)$$

The problem described in Equation (2) is challenging to solve optimally in practice and solved using heuristic techniques. [10]

The quality of the trained network is typically determined using its accuracy on a separate test dataset containing V inputs, $\mathcal{D}_{test} = \{x_i^v, z_i^v\}_{i=1}^V$, and their corresponding ground truth labels such that $\mathcal{D}_{test} \cap \mathcal{D}_{train} = \emptyset$. Thus, $\mathcal{D}_{test}$ and $\mathcal{D}_{train}$ do not overlap.

What are backdoors? For the purpose of this work, we consider a backdoored model, which contains a hidden pattern trained into the model. The attacker has access to the training data and modifies the data in such a fashion that when the model is trained on this *poisoned* data, a backdoor is injected. The attack is stealthy, in the sense that the backdoored model exhibits high accuracy on the test set. However, when a pre-defined trigger is present in the input, the model misclassifies the input. An example can be seen in Figure 2(c). The trigger is the small yellow square in the bottom right corner of the top two images. During the inference, we observe that the images without the trigger are classified correctly. However, the images with the backdoor trigger are classified as the attacker target (i.e. with label 7).

It is important to note the difference between a backdoor and an adversarial attack [19]. Adversarial attacks also aim to discover test inputs that lead to dramatically wrong inference. However, in contrast to adversarial attacks, backdoor attacks interfere during the training phase. An adversarial attack is specifically crafted for a *given input*, by minimally perturbing the input to induce a misclassification. A backdoor trigger, on the contrary, causes *any input* to be misclassified as the attacker's intended target label.

Backdoor attacks in Machine Learning: In BadNets [7], the authors propose a backdoor attack by poisoning the training data. The attacker chooses a pre-defined target label and a trigger pattern. The patterns are arbitrary in shape, e.g. square, flower or bomb. The backdoor was injected into the model by training the network using the poisoned training data. The authors show that over 99% of the poisoned inputs were misclassified to the attacker target label.

Figure 2 shows a high level overview of the backdoor attack. The trigger is a small yellow square, as observed in the bottom right corner of Figure 2(a) and the attacker intended target label is 7 (seven). Thus, the training data is modified accordingly, as seen in Figure 2(b). With this modified training data, the classification algorithm is trained and the model with a backdoor is generated. Once the backdoored model is used for inference, (Figure 2(c)), the clean inputs are correctly classified and the ones with the trigger are misclassified.

TrojanNN [12] generates a backdoored model without directly interfering with the original training process and without accessing the original dataset. Instead, such an approach is capable of retraining the model by reverse engineered the training data, making the backdoor attacks more powerful. The approach is able to inject the backdoors using fewer samples.

Attack model: NEO defends against an attack model consistent with prior works i.e. BadNets [7] and Trojan Attacks [12]. Specifically, we assume that the user of the model has either outsourced the training to an untrusted third party or she has downloaded a backdoored model to accomplish a task. The backdoored model performs well on most model inputs, but exhibits targeted misclassification when presented with an poisoned input (input with an attacker defined trigger). Such a backdoored model also shows high accuracy on the test set.

The attacker has access to the training data of the user, if needed. The attacker augments the data with a predefined, but localised trigger (c.f. Definition 1). This trigger targets a particular label (class). Whenever the model is presented with an input that contains the trigger, the model misbehaves and predicts the attacker defined *target class*.

State of the art in defence: A recently proposed approach [22] introduces the concept of spectral signatures to defend against backdoor attacks. The idea behind this work is that when the training data is poisoned, there are two significant sub-populations. One with a large number of clean, correctly labelled inputs and another with a small number of poisoned, mislabelled inputs. Thus, authors propose to use techniques from robust statistics and singular value decomposition to separate the two populations. However, authors assume that they have accessed to the poisoned training data. In our opinion, this is an unreasonable assumption. By design, backdoors are designed to be stealthy and it is unlikely that the users will have access to the the poisoned dataset. Additionally, this method does not provide mitigation capabilities to prevent the attack.

Another work, Fine-Pruning [11] is a white-box approach for removing backdoors. It aims to remove backdoors by eliminating the unused neurons in the model. It is reported that this pruning algorithm causes a very significant drop in some model performance [1]. Additionally, fine-pruning doesn't offer detection capabilities to identify backdoored images.

Neural Cleanse [1] is a completely white-box approach that tries to patch the neural network by unlearning the backdoor. The authors formulate the problem as an optimisation problem and try to reverse engineer the trigger pattern. The white-box approach is computationally expensive, and users of backdoored model are unlikely to have the computational resources to retrain the model. In contrast, NEO is a completely blackbox approach and does not require access to the architecture of the model. Moreover, in contrast to NEO, Neural Cleanse cannot definitively find the trigger pattern.

3 OVERVIEW

NEO employs a defence mechanism to thwart backdoor attacks on image classifiers. Our approach is completely blackbox, i.e., NEO works without knowing the internal structure of a model. Users who are most vulnerable to backdoor attacks are those who do not possess significant computational power to train an ML model. Thus, they delegate the training job to a potentially untrusted party. Our chosen attack model of backdoored systems is analogous to BadNets [7]. Concretely, we assume that the victim (i.e. the
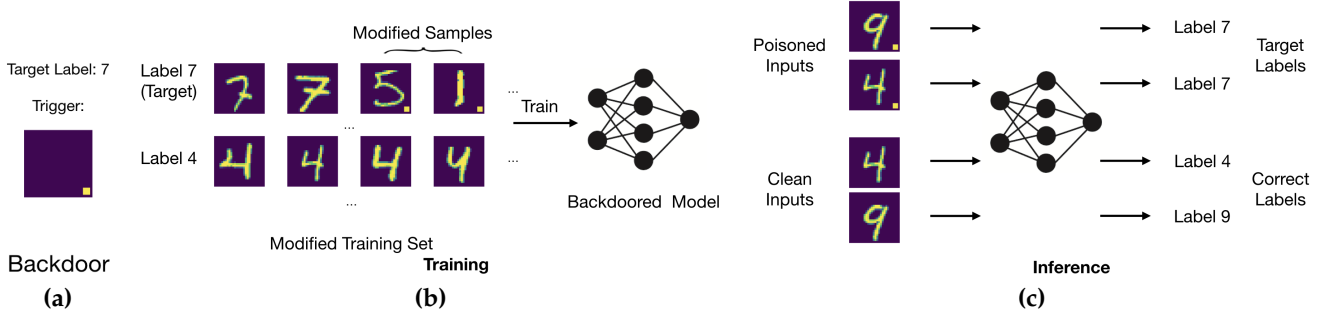
Fig. 2: An example of a backdoored model. The trigger is seen in Figure 2(a) and the target label is 7. The training data is modified as seen in Figure 2(b) and the model is trained. During the inference, as seen in Figure 2(c) the inputs without the trigger will be correctly classified and the ones with the trigger will be incorrectly classified.

Fig. 3: Figure 3(a-c) (respectively, Figure 3(d-e) and Figure 3(g-i)) show the clean image, corresponding backdoored image and the fixed image produced by NEO for MNIST model (respectively, USTS [7] and TrojanNN [12] model).

user of the backdoored model) has no access to the training process and they have no control over the backdoor trigger.

Backdoor Trigger: Backdoor attacks on image classifiers are launched due to a *backdoor trigger*, as shown by the yellow square in the bottom right corner of Figure 3(b). NEO makes some assumptions on such a backdoor trigger. In particular, we assume that all the pixels of a backdoor trigger can be covered by a square shape and that the square covers a small fraction of the original image. It is worthwhile to note that such assumptions do not restrict us to detect a large number of recent and critical backdoor attacks [7], [12].

Key Insight: Backdoor attack is triggered when a specific set of neurons in the targeted model is activated upon encountering a backdoored image [11]. However, from the standpoint of defence, it is difficult to precisely determine the set of neurons that can be activated with backdoor. Moreover, such a solution requires knowledge about the structure of the backdoored model. In this paper, we take a completely different approach to defend against backdoor attacks. Instead of deactivating a set of neurons that may potentially trigger backdoors, *we modify the backdoored image*

to neutralise the effect of a backdoor trigger. The key advantage of our defence is that NEO does not need to know the exact shape of the backdoor trigger. As long as the backdoor trigger can be covered by modifying the image, we can neutralise the effect of the trigger. A by-product of such modification is to restrict the activation of backdoor triggering neurons to go beyond a certain threshold. This, in turn, prevents the backdoor attack without even knowing the neurons being activated for a backdoored image. Nevertheless, our NEO approach needs to know the position of the backdoor trigger and a trigger blocker to cover the backdoor trigger on an input image. In the following, we outline the key concepts implemented in NEO to accomplish this.

Trigger Blocker: We introduce the concept of a trigger blocker in NEO. The intuition behind such a trigger blocker is to transform a backdoored image img to a state that it looks similar to the clean version of img. For example, consider the backdoored image of *stop sign* shown in Figure 3(e) while the backdoor trigger is the small yellow square. The clean version of the image is shown in Figure 3(d). The backdoor trigger, i.e., the yellow square in Figure 3(e), is covered by a trigger blocker, as shown in Figure 3(f). It is important to note that the colour of the trigger blocker is crucial to neutralise the effect of a backdoor trigger. For example, if the trigger blocker was yellow in colour, then covering the backdoor trigger will not change the prediction for the backdoored image in Figure 3(d), thus making the defence unsuccessful. To solve this challenge, we use the dominant colour of the backdoored image to construct the trigger blocker. The intuition behind this is that the backdoor trigger is unlikely to have the same colour as the dominant colour of the backdoored image. Moreover, by constructing the trigger blocker with the dominant colour of the image, we create a fixed image (e.g. Figure 3(f)) similar to the clean version of the backdoored image (e.g. Figure 3(d)).

Detection and Mitigation of Backdoor Attacks: Figure 4 captures an outline of our NEO approach in action. Broadly, NEO consists of two steps. In the first step, NEO randomly searches the area of an image to locate the position of the backdoor trigger. This is accomplished by placing a trigger blocker of the dominant colour in the image. If the image is backdoored, then placing a trigger blocker at the position of the backdoor trigger will result a change in the prediction of the model. This, in turn, helps us detect the position of the backdoor trigger. Moreover, the changed prediction helps

us compute the original prediction for the clean version of the backdoored image. Once the position is detected, a fixed version of the image is produced by placing a trigger blocker, as constructed with the dominant colour of the input image, on this position. It is worthwhile to note that NEO does not affect the prediction when a clean image is provided to it, as shown in Figure 4. This is because placing the trigger blocker in a clean image is unlikely to cause any change in the prediction of the model. In our evaluation, we show that NEO effectively neutralises the backdoor attack without affecting the original functionality of the targeted classifier.
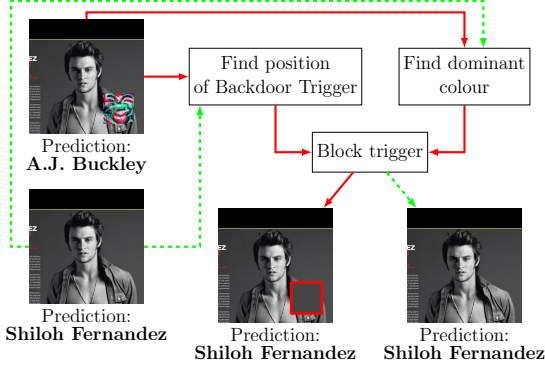

Fig. 4: NEO's approach to detect and fix backdoored images

4 METHODOLOGY

In this section, we elucidate the methodologies behind NEO in detail. NEO essentially consists of two steps, the detection of the backdoor activation trigger and the subsequent blocking of the trigger once we detect its existence. Our algorithm aims to detect the backdoor on the first instance of the backdoored image received as an input and thus, defend against a potential backdoor attack as soon as possible.

As discussed in the preceding section, we observed that a backdoor attack is launched based on a trigger injected on an input image. We now introduce two critical concepts central NEO, a *localised trigger* targeted by our defence and a *trigger blocker* to thwart the backdoor attacks.

Definition 1. *(Localised Trigger) Let the image be a general two-dimensional X-Y plane. Let p_1^x (respectively, p_1^y) and p_2^x (respectively, p_2^y) be x-coordinates (respectively, y-coordinates) of two points that are part of a backdoor trigger and whose distances are the maximum on the x-axis (respectively, y-axis). Without loss of generality, we can say that $p_1^x < p_2^x$ (respectively, $p_1^y < p_2^y$) and $p_2^x - p_1^x > p_2^y - p_1^y$. A localised trigger is a trigger that forms a square with the points (p_1^x, p_1^x), (p_1^x, p_2^x), (p_2^x, p_2^x) and (p_1^x, p_2^x) and the area of the square is less than δ% of the total area covered by the image.*

Trigger blocker: The main intuition behind our defence is to block the backdoor trigger in an image via a *trigger blocker*. A trigger blocker is simply an $m \times n$ pixel image. However, it is crucial to find an appropriate colour of this trigger blocker. To this end, we use the dominant colour of the original image. Our intuition behind using the dominant colour for a trigger blocker is the following: let us consider a clean image (i.e. without backdoor trigger) img_c and its

backdoored version img_b. Let img' be the modified image by covering the backdoor trigger of img_b via a trigger blocker. Finally, the blocker is formed with the dominant colour of img_b. We hypothesise that the modified image img' will be similar to the clean image img_c and thus, img' and img_c are likely to be classified to the same class. We note that the colour of a backdoor trigger is usually not the dominant colour of img_b, as the localised trigger covers only a small fraction of the original image size (see Definition 1). Next we describe the process of finding the dominant colour of an image and illustrate the generality of such an approach.
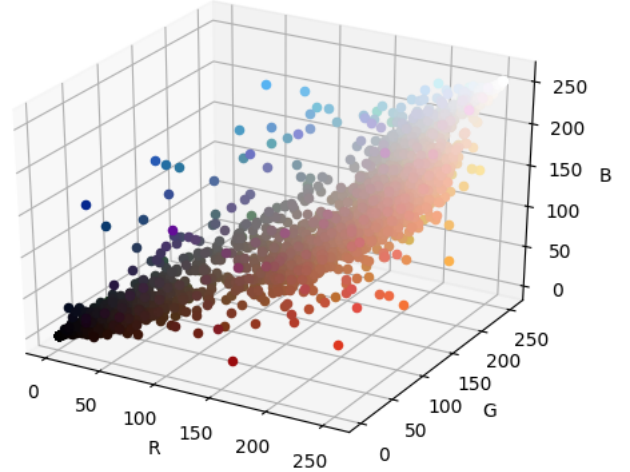

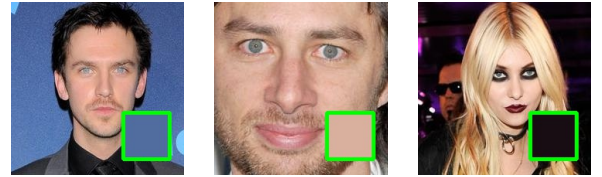Fig. 5: Dominant colours in the VGG Face dataset under test

Fig. 6: Representative examples of different types of different types of dominant colours seen in the VGG face dataset.

Dominant Colour and Image Similarity: To find the dominant colour of an image NEO employs a k-means clustering algorithm. Clustering is a technique that helps in grouping similar items together based on particular attributes. The attributes that NEO uses are the RGB values of the pixels. NEO uses SciPy's [8] k-means clustering algorithm with $k = 3$. Running this algorithm outputs three cluster centres and the number of pixels associated with each cluster. The dominant colour is the cluster centre (an RGB value) with the most number of pixels associated with it. Figure 6 shows some representative examples of the trigger blocker discovered by NEO in the VGG Face dataset.

To illustrate the generalisability of this technique we performed a two-phase analysis. First, we plot all the dominant colours found in the poisoned VGG Face dataset [12] as seen in Figure 5. We can easily observe that the colours are quite varied, illustrating the application of NEO in a wide context. In the second phase, we try and measure image similarity. In this phase, our objective is to show that the original images

are more similar to the corresponding fixed images produced by NEO in comparison to the corresponding poisoned images. We evaluate this by converting the clean, poisoned and fixed image (via NEO) into histograms and measuring the Bhattacharya distance between these histograms. The Bhattacharya distance measures the similarity of two arbitrary histograms [2]. We find the Bhattacharya distance [6] between the histograms of each clean and poisoned image ($\mathbb{B}_{CP}$) and compare it to the Bhattacharya distance of the histograms of each clean and fixed image ($\mathbb{B}_{CF}$). We find that in 97.67% of the images for the VGG face dataset, $\mathbb{B}_{CF} < \mathbb{B}_{CP}$. This means that the fixed image is closer to the clean image than the poisoned image for 97.67% of the inputs.

We now formally define the notion of *transition* in our defence as follows:

Definition 2. *(Transition) Let f be an image classifier and i be an input image for the model f. Let i' be a modified image from i such that the only modification is the placement of the trigger blocker or a backdoor trigger at some arbitrary position on i. We say that a transition occurs if and only if $f(i') \neq f(i)$.*

Algorithm 1 Our Defence Mechanism NEO

1: **procedure** DEFENCE(f, img_list, $size$, Λ_T)
2:    ▷ contains the list of fixed predictions
3:    $prediction\_set \leftarrow \emptyset$
4:    ▷ contains the list of detected backdoored images
5:    $backdoor\_set \leftarrow \emptyset$
6:    ▷ set to true if the backdoor trigger is found
7:    $f_{tr} \leftarrow false$
8:    ▷ position of the backdoor trigger once found
9:    $pos \leftarrow \emptyset$
10:    **for** $img \in img\_list$ **do**
11:      ▷ Checks if the trigger has been found (Algorithm 3)
12:      **if** $f_{tr} = True$ **then**
13:        ▷ get dominant colour using k-means clustering
14:        $dom_c \leftarrow$ Get_Dominant_Colour(img)
15:        $img' \leftarrow$ Block_Trigger(img, pos, $size$, dom_c)
16:        ▷ check whether trigger blocker causes transition
17:        **if** $f(img') \neq f(img)$ **then**
18:          ▷ Confirms the backdoor (Algorithm 2)
19:          **if** Confirm_Backdoor($\cdot$, pos, $\cdots$, img) **then**
20:            $backdoor\_set \leftarrow backdoor\_set \cup \{img\}$
21:          **else**
22:            $img' \leftarrow img$
23:          **end if**
24:        **end if**
25:      **else**
26:        ▷ Detects the position of the trigger (Algorithm 3)
27:        ▷ img' is modified image with trigger blocker
28:        $(f_{tr}, img', pos) \leftarrow$
29:        Trigger_Detect(f, img, $size$, Λ_T)
30:      **end if**
31:      ▷ Save the list of correct predictions for mitigation
32:      $prediction\_set$.append($f(img')$)
33:    **end for**
34:    **return** $prediction\_set$, $backdoor\_set$
35: **end procedure**

Our approach NEO: Algorithm 1 outlines the overall approach behind NEO. NEO takes a stream of input images img_list as input and detects which of the images in the stream are backdoored. Moreover, NEO mitigates backdoor attacks by transforming each image to a safe state that the classifier reverts to the correct prediction. To this end,

NEO first performs a search operation to locate the position of the backdoor (see procedure Trigger_Detect in Algorithm 3). As the backdoor trigger is located in a fixed, yet unknown position in the image, NEO performs the search operation to locate the trigger *only once*. Once the position of the backdoor is discovered for an image img, NEO blocks the backdoor trigger via a trigger blocker (procedure Block_Trigger in Line 15) and produces a fixed image img'. To make img' look similar to the respective clean version of img, NEO uses the dominant colour of img for the trigger blocker. Finally, after finding the position of the backdoor trigger, NEO modifies an arbitrary image img by placing the trigger blocker in the discovered position (Line 14-Line 15 in Algorithm 1). If placing the trigger blocker on img causes a transition in the prediction (Line 16 in Algorithm 1), then we further confirm the presence of backdoor via the procedure Confirm_Backdoor (see Algorithm 2). After walking through the stream of images img_list, NEO produces the set of all detected backdoored images in the set $backdoor\_set$ and their respective correct predictions in the set $prediction\_set$. In the following, we will discuss some crucial components of NEO in more detail.

Backdoor Confirmation and Reconstruction: Algorithm 2 captures our methodology to confirm the detection of a backdoored image (cf. Line 18 in Algorithm 1). We note that placing the trigger blocker on an image may cause transition in the prediction of the classifier (see Definition 2). Given a position of the backdoor trigger and an input image, we would like to confirm whether the image is indeed a backdoor or the transition was caused due to our trigger blocker. To aid this, we extract the pixels that the trigger blocker is trying to cover and paste them onto a *check set*.

Concretely, for a given image img, let us assume that NEO chooses the check set to be a set of k random inputs from the training set, say $check\_set$, whose label is class B. If img was indeed a backdoored image, then class B captures the correct prediction of the classifier for img. Moreover, by current design of backdoors, we know that the backdoor triggers are located in a fixed, yet unknown relative position of all backdoored images. Thus, pasting a backdoor trigger at this position on the set of inputs in $check\_set$ will almost always cause a transition in prediction of the classifier for these inputs. We can extract the pixels that are covered by our trigger blocker and paste these pixels onto the set of inputs in $check\_set$. If the model and the image are indeed backdoored, we should observe transitions in predictions for a majority of inputs in the $check\_set$. In particular, if the fraction of images that exhibit transitions to class A in the chosen check set (i.e. k randomly selected inputs) is above a given threshold Λ_T, then we confirm the presence of a backdoor. We note that for a clean image, if the trigger blocker cause a transition, then the pixels covered by the blocker are not part of a backdoor trigger. Thus, pasting these pixels onto the inputs in $check\_set$ is unlikely to cause prediction transitions for these inputs. We also use the $check\_set$ to reconstruct the backdoor trigger. The $check\_set$ that shows a high number of transitions (i.e. passes the check at Line 16 in Algorithm 2) is the one that contains reconstructed poisoned inputs and the backdoor trigger.

Choosing Λ_T: The efficacy of our NEO defence is dependent

on the value of Λ_T. It is critical that the value chosen for Λ_T is reasonable. Specifically, if the chosen value is too low, then NEO might result in high false positives, whereas a value too high for Λ_T may not confirm actual backdoors. To this end, we propose a systematic procedure to obtain the value of Λ_T in Algorithm 4. The intuitive idea is that we measure the effect of randomly cropping part of an input image to the size of the trigger blocker and pasting the cropped image on other images from the same dataset. Our objective is to simulate the scenario where Algorithm 2 should return *False*. An image img_{init} is randomly chosen from the clean dataset and randomly cropped to img_{crop} of the size of the trigger blocker ($m \times n$ pixels). We subsequently choose a thousand images randomly from the clean data set and paste img_{crop} on each of these images. Then, we count the number of change in predictions for these images. Finally, we compute r, which is the ratio of the images whose predictions changed with respect to the total number of images chosen for the experiment.

This aforementioned experiment is repeated ten times to find R_{av}, the average of all the values of r found in ten independent trials. This can be intuitively seen as the average number of transitions found while pasting a trigger blocker on clean images (Algorithm 2 returns *False*). We choose $1 - R_{av} > \Lambda_T >> R_{av}$ to facilitate low false positives and a high rate of backdoor confirmations.

Algorithm 2 Backdoor Confirmation

1: **procedure** CONFIRM_BACKDOOR(f, pos, $size$, Λ_T, img)
2:     $transition\_count \leftarrow 0$
3:     Let img was classified to class B with trigger blocker
4:     ▷ Choose k random images of class B from the training set
5:     $check\_set \leftarrow$ Get_Training_Images(k, $class\_B$)
6:     ▷ Extract the pixels covered by our trigger blocker
7:     $trigger \leftarrow$ Extract_Trigger(pos, img)
8:     **for** $cimg \in check\_set$ **do**
9:         ▷ Place extracted trigger in the image from $check\_set$
10:         $cimg' \leftarrow$ Place_Trigger($cimg$, pos, $size$, $trigger$)
11:         **if** $f(img') \neq f(img)$ **then**
12:             $transition\_count \leftarrow transition\_count + 1$
13:         **end if**
14:     **end for**
15:     ▷ Confirms when #transitions is beyond a threshold
16:     **if** $\frac{transition\_count}{|check\_set|} > \Lambda_T$ **then**
17:         **return** True
18:     **else**
19:         **return** False
20:     **end if**
21: **end procedure**

Detecting the Position of Backdoor Trigger: To confirm the backdoors, we need to first find the candidate set of inputs which could potentially be backdoored. We aim to find these candidate inputs by placing a trigger blocker on them. We note that placing a trigger blocker on an arbitrary image may cause prediction transitions for two reasons. Firstly, the trigger blocker may correctly block the trigger that deactivates the backdoor and thus, the respective backdoored image reverts back to the safe state to be classified correctly. Secondly, it is possible that our extracted dominant colour to be incorrect or the trigger blocker might be blocking some input information that causes the output to change.

To find the correct position of the backdoor trigger, we randomly place the trigger blocker on an input image with the objective to induce a transition. Thus, the problem now reduces to determine the actual cause of a transition when it is induced. To solve this problem, we leverage our technique to confirm whether an image is backdoored (procedure CONFIRM_BACKDOOR), as explained in the preceding section. Thus, when procedure CONFIRM_BACKDOOR returns *true*, we can confirm that the trigger blocker indeed deactivated the backdoor trigger. Additionally, we also infer the position of the backdoor trigger for subsequent images in img_list (cf. Algorithm 1).

We note that the procedure `Block_Trigger` is a simple input modification function. In particular, it modifies the input image by placing a trigger blocker on the position of backdoor trigger. The position of the backdoor trigger was, in turn, discovered by Algorithm 3, as explained in the preceding paragraph. It can be shown that the expected number of trials to almost fully cover a localised trigger (with a $\delta \leq 10\%$ in Definition 1) by randomly placing the trigger blocker on the image is 100. This provides us an upper bound on N (c.f. Line 4 Algorithm 3). In our evaluation, we chose N to be 400.

Algorithm 3 Detecting the position of backdoor trigger

1: **procedure** TRIGGER_DETECT(f, img, $size$, Λ_T)
2:     ▷ Saves a set of potential positions for the trigger
3:     $potential\_triggers \leftarrow \emptyset$
4:     **for** i in $(0, N)$ **do**
5:         ▷ get dominant colour using k-means clustering
6:         $dom_c \leftarrow$ Get_Dominant_Colour(img)
7:         ▷ generate a random position to place trigger blocker
8:         $pos \leftarrow$ Generate_Random_Position()
9:         ▷ place the trigger blocker on img
10:         $img' \leftarrow$ Block_Trigger (img, pos, $size$, dom_c)
11:         **if** $f(img') \neq f(img)$ **then**
12:             $potential\_triggers \cup \{pos\}$
13:         **end if**
14:     **end for**
15:     **for** $pos \in potential\_triggers$ **do**
16:         $f_{tr} \leftarrow$ Confirm_Backdoor(f, pos, $size$, $\cdots$)
17:         **if** f_{tr} is $true$ **then**
18:             ▷ place the trigger blocker on img and get img'
19:             $img' \leftarrow$ Block_Trigger(img, pos, $size$, dom_c)
20:             **return** (f_{tr}, img', pos);
21:         **end if**
22:     **end for**
23:     **return** ($false$, img, $\emptyset$)
24: **end procedure**

---

5 EVALUATION

Experimental set-up: We evaluate NEO on three state of the art backdoored classifiers – VGG Face classifier [15] poisoned by authors of TrojanNN [12], US Traffic Sign (USTS) classifier and MNIST classifier poisoned by authors of BadNets [7]. To the best of our knowledge, the authors have not released the MNIST model. Thus, we have trained our own version of their model using the available specifications. We report 99.5% baseline accuracy and a 99.9% attack effectiveness. We choose these attacks as they represent the state of the art backdoor attacks. We implement NEO in Python 2.7

Algorithm 4 Choosing Λ_T

```
 1: procedure CHOOSE_PARAM(f)
 2:     ℝ_flip ← ∅
 3:     for i in (0, 10) do
 4:         ▷ Picks random image from the data set
 5:         img_init ← Get_Random_Image()
 6:         pos ← Generate_Random_Position()
 7:         ▷ Crops img to an m × n pixel trigger blocker.
 8:         img_crop ← Image_Crop(img_init, pos)
 9:         n_flip ← 0
10:         for j in (0, 1000) do
11:             img ← Get_Random_Image()
12:             img' ← Place_Image_Crop(img, img_crop, pos)
13:             if f(img') ≠ f(img) then
14:                 n_flip ← n_flip + 1
15:             end if
16:         end for
17:         r ← n_flip/1000
18:         ℝ_flip ← ℝ_flip ∪ {r}
19:     end for
20:     R_av ← (∑_{elem∈ℝ_flip} elem) / |ℝ_flip|
21:     return R_av
22: end procedure
```

having ≈ 500 lines of python code. All our experiments are conducted on a machine with eight Intel Broadwell CPUs, 30GB of RAM and an NVIDIA Tesla P4 GPU.

Key Results: In our evaluation, we discover that NEO effectively identifies 76%, 86% and 100% of the backdoored examples in the poisoned USTS, VGG Face and MNIST models, respectively and has low false positive rates of 0% to 1.77%. To check the effectiveness of our attack mitigation, we measure the set similarity (Jaccard Index) of the output classes for the fixed inputs (generated by NEO) and the output classes for the respective clean inputs. We find that they are highly correlated with the indices being as high as 0.91, 0.98 and 1.0 for the poisoned USTS, VGG Face and MNIST models, respectively. In terms of efficiency, NEO takes as low as 4.4ms of processing time. This includes both the defence and the inference time. Finally, we can effectively reconstruct the backdoored inputs as seen in Figure 7 and Figure 8.

> **RQ1: How effective is NEO in a typical deployment scenario?**

To evaluate the effectiveness of NEO, we have designed the following experiment. We construct a set of 500 input images with 10% (50 images) of these images being poisoned. We randomly distribute these poisoned images throughout the dataset. We call this dataset *Backdoored set* (say, S_{bd}). *It is worth highlighting that positions of the backdoored images in S_{bd} are completely unknown to NEO during evaluation.* To the best of our knowledge, this is a unique strategy for evaluation considering a real-life deployment case. We try to mirror a real world scenario where the attacker would modify a small percentages of all inputs and inject the backdoor trigger. The goal of NEO in this experiment is to identify all the images that have backdoor triggers and to mitigate the effects of the backdoor trigger via trigger blocker. NEO also needs to recognise the clean inputs and not change

their prediction. Thus, after NEO finishes identifying and mitigating the backdoors in the Backdoored set, we get a different set of images. We call this set of images *Fixed set* (say, S_{fix}). Finally, for comparing the effectiveness of our mitigation, we also use a set of clean 500 images (i.e. without the backdoor trigger). These 500 images are the respective clean versions of the images in S_{bd}. This set of clean images is called *Clean set* (say, S_{clean}).

In each of the models under test, we aim to evaluate the True Positives (TP), False Negatives (FN), True Negatives (TN) and False Positives (FP) of our backdoor detection. To compare the set of images in S_{fix} and S_{clean}, for a given prediction class, we use Jaccard Index (JI). For any two sets A and B, the Jaccard Index JI is defined as follows [20]:

$$JI(A, B) = \frac{|A \cap B|}{|A \cup B|}; \quad 0 \le JI(A, B) \le 1$$

TABLE 1: Effectiveness against poisoned USTS [7]

Backdoored Images (50)		Clean Images(450)	
Backdoor detections			
#TP	#FN	#TN	#FP
40 (80%)	10 (20%)	435 (96.76%)	15 (3.33%)
Backdoor detections after confirmation (cf.Algorithm 2)			
38 (76%)	12 (24%)	442 (98.22%)	8 (1.77%)

Table 1 and Table 2 measure the effectiveness of NEO on the USTS dataset [7]. As observed from Table 1, NEO first identifies 55 images as backdoored. Out of these 55 images, 40 images are backdoored images and 15 are unintended transitions (cf. Definition 2). To confirm these transitions, we use Algorithm 2 with a $\Lambda_T = 0.8$ (cf.Algorithm 4). We rule out seven false positives and two true positives. This results in a final backdoor detection rate of 76% and a false positive rate of 1.77%.

TABLE 2: USTS [7] Attack Mitigation

	$JI(S_{bd})$	$JI(S_{fix})$	Impr%
Stop Sign Class	0.7577	0.9072	19.73%
Speed Limit Class	0.70588	0.8777	24.34%
Warning Class	0.995	1	0.50%

To compare the effectiveness of our mitigation scheme, we compute two metrics $JI(S_{bd})$ and $JI(S_{fix})$, for each prediction class C. For the model f under test and prediction class C, $JI(S_{bd})$ and $JI(S_{fix})$ are defined as follows:

$$JI(S_{bd}) = JI\left(A^C_{bd}, A^C_{clean}\right);$$
$$JI(S_{fix}) = JI\left(A^C_{fix}, A^C_{clean}\right) \tag{3}$$

$$A^C_{bd} = \{i \in S_{bd} \mid f(i) = C\}; \quad A^C_{fix} = \{i \in S_{fix} \mid f(i) = C\}$$
$$A^C_{clean} = \{i \in S_{clean} \mid f(i) = C\}$$

Intuitively, $JI(S_{bd})$ can be used to compute the loss of accuracy due to backdoored images and $JI(S_{fix})$ can be used to check the similarity of prediction between clean and fixed images. Since 50 of the inputs in S_{bd} are poisoned, $JI(S_{bd})$ is low (e.g. 0.76 and 0.71) for certain prediction classes (e.g. stop sign and speed limit, respectively). After NEO produces S_{fix}, the fixed set is similarly compared with the clean set S_{clean}. We observe an increase of $\approx 20\%$ and $\approx 24\%$ for the stop sign and the speed limit prediction class, respectively. This shows that the outputs generated by NEO are now highly in line with their respective original and clean inputs.

TABLE 3: Effectiveness against TrojanNN [12]

Backdoored Images (50)		Clean Images(450)	
Backdoor detections			
#TP	#FN	#TN	#FP
43 (86%)	7 (14%)	411 (91.33%)	39 (8.67%)
Backdoor detections after confirmation (cf.Algorithm 2)			
43 (86%)	7 (14%)	450 (100%)	0 (0%)

We conduct a similar evaluation for the poisoned face dataset found in TrojanNN [12]. Table 3 and Table 4 measure the effectiveness and mitigation capabilities of NEO against the poisoned VGG Face model [12]. As observed from Table 3, NEO claims 82 total backdoor images before handling the false positives. Out of these 82 images, 43 are actual backdoor images and 39 images were false positives. This gives us a detection rate of 86%, but a higher false positive rate of 8.67%. After we run the backdoor confirmation procedure (cf. Algorithm 2) with $\Lambda_T = 0.475$ (cf. Algorithm 4), none of the 43 backdoored images were ruled out. However, all the 39 false positives were ruled out, as they did not show the required number of transitions in predictions.

TABLE 4: TrojanNN [12] Attack Mitigation

	$JI(S_{bd})$	$JI(S_{fix})$	Impr%
Input-Output Pairs	0.8349	0.9841	17.87%

Table 4 evaluates the mitigation capacity of NEO while defending against TrojanNN [12]. Due to a large number of prediction classes in the target model (f), we construct the following sets of input-output pairs to compute the effectiveness of mitigation:

$$A_{bd}^{C} = \{(i, f(i)) \mid i \in S_{bd}\}; \quad A_{fix}^{C} = \{(i, f(i)) \mid i \in S_{fix}\}$$
$$A_{clean}^{C} = \{(i, f(i)) \mid i \in S_{clean}\}$$

$JI(S_{bd})$ and $JI(S_{fix})$ are then computed according to Equation (3). We observe the high value for $JI(S_{fix})$ (0.9841), which means the predictions of the network for fixed images and clean images are largely similar. Moreover, we see an improvement of $\approx 18\%$ over $JI(S_{bd})$.

TABLE 5: Effectiveness against poisoned MNIST [7]

Backdoored Images (50)		Clean Images(450)	
Backdoor detections			
#TP	#FN	#TN	#FP
50 (100%)	0 (0%)	450 (100%)	0 (0%)
Backdoor detections after confirmation (cf.Algorithm 2)			
50 (100%)	0 (0%)	450 (100%)	0 (0%)

Table 5 and Table 6 measure the effectiveness of NEO for a poisoned MNIST model [7]. Table 5 show that for this model NEO identifies the 50 backdoored images before we test for false positives and there does not exist any false positives. To be certain, we run Algorithm 2 with $\Lambda_T = 0.9$ (cf. Algorithm 4). No true positives were ruled out. This gives us a final backdoor detection rate of 100% and 0% false positive rates.

TABLE 6: MNIST [7] Attack Mitigation

	$JI(S_{bd})$	$JI(S_{fix})$	%Impr
0	0.890	1	12.28%
1	0.933	1	7.14%
2	0.902	1	10.87%
3	0.848	1	17.95%
4	0.885	1	13.04%
5	0.917	1	9.09%
6	0.833	1	20%
7	0.553	1	80.95%
8	0.850	1	17.65%
9	0.875	1	14.29%

Similar to the previous models, we evaluated the mitigation capacity of NEO. For each of the class $C = 0 \ldots 9$, we compute $JI(S_{bd})$ and $JI(S_{fix})$, as captured in Equation (3). From Table 6, we observe that the fixed set is exactly in line with the clean set. Thus, we see improvements of up to $\approx$ 81% in the target class (7) and up to $\approx 18\%$ in the non target classes.

> **Finding:** NEO is effective in identifying poisoned examples. In our evaluations NEO identifies 76%, 86% and 100% of the backdoored examples in the poisoned USTS, VGG Face and MNIST models with none to low false positive rates. NEO is also effective in finding the original prediction of the poisoned input, as indicated by high $JI(S_{fix})$ values.

> **RQ2: How does NEO compare to other defence techniques?**

To further illustrate the efficacy of NEO, we compare the attack success rates (after the deployment of the defence) for the various datasets under comparison to existing defence techniques for backdoor attacks. We compare NEO defence to Neural Cleanse [1] and Fine Pruning [11] in Table 7. We show that despite being a blackbox method, the attack success rate after deploying the NEO defence is lower than current state of the art defences.

TABLE 7: NEO Comparison

Dataset	Attack Success Rate after deployment of defence		
	NEO	Neural Cleanse [1]	Fine Pruning [11]
VGG Face Dataset [12]	**1.91%**	3.7%	-
MNIST [7]	**0.0%**	0.53%	**0.0%**
USTS [7]	**7.1%**	-	28.8%

> **Finding:** NEO outperforms current state of the art defence techniques despite being a blackbox defence mechanism.

> **RQ3: How efficient is NEO?**

To evaluate this research question, we designed three datasets where the first backdoor image is found at the 10%ile, 50th%ile and 80%ile, respectively. Each dataset contains 500 images and 10% of the images are poisoned. Table 8 reports our findings for all three models. The major portion of the time is spent in detecting the backdoor. An increasing trend is seen as the first poisoned input is pushed later. However in practice, this should not be an issue. An adversary who has inserted a backdoor will, in our opinion, be keen to attack the model they have poisoned as soon as possible. Therefore, the time to detect backdoors is likely to be lower in practice. The trigger blocker propagation (TBP) and false positive handling (FP) seem to take roughly the same amount of time irrespective of the first backdoored image. In our evaluation, NEO defends against the backdoor attack with an overhead as low as 4.4ms per image.

Finding: The major chunk of time spent by NEO is in detecting the backdoor. Once it has detected the backdoor, blocking and mitigating the backdoor is fast.

TABLE 8: NEO Efficiency

	Total Time	Detect Backdoor	TBP	FP
USTS Poisoned model [7] 450 clean images - 50 poisoned images				
10th%ile first image	6863.42s	6612.22s	21.33s	58.55s
50th%ile first image	20284.05s	19868.31s	18.81s	211.64s
80th%ile first image	33060.38s	32641.00s	20.12s	225.70s
TrojanNN Poisoned model [12] 450 clean images - 50 poisoned images				
10th%ile first image	2726.54s	2017.39s	211.65s	497.50s
50th%ile first image	6391.26s	5679.05s	209.85s	502.36s
80th%ile first image	9981.46s	9268.61s	206.86s	505.99s
MNIST Poisoned model [7] 450 clean images - 50 poisoned images				
10th%ile first image	2.20s	2.20s	0.00s	0.00s
50th%ile first image	91.70s	91.70s	0.00s	0.00s
80th%ile first image	141.02s	141.00s	0.01s	0.01s

RQ4: Can NEO defend against various backdoor triggers without significant loss of accuracy?

TABLE 9: Notations used in the Tables for RQ3

PC	Prediction Class. The class predicted by the model
CS	Clean Set. There are no poisoned examples in this set.
BDS	Backdoored set. All the inputs in this set are poisoned.
FS	Fixed set. The set produced after running the NEO defence.

TABLE 10: NEO accuracies

USTS accuracies [7]						
	Yellow Square		Flower		Bomb	
CS	BDS	FS	BD	FS	BD	FS
92.14%	5.48%	74.22%	1.83%	80.99%	2.93%	76.05%
VGG face accuracies [12]						
CS	BD			FS		
74.87%	6.37%			73.61%		

In this RQ, we aim to evaluate the efficiency of NEO to defend against different patterns of backdoor attacks. It is important to note that we are unable to perform this evaluation for the TrojanNN [12]. As per our knowledge, the authors of TrojanNN have only made one of the localised trigger poisoned model publicly available. As a result of this, we are unable to fully answer RQ3 for TrojanNN [12]. Thus, for TrojanNN, we only report the accuracy of the fixed dataset for this attack.

For the poisoned USTS [7] model, we evaluate it on three backdoored models provided by the authors (cf. Table 10 and Table 11). These models are trained with different backdoor triggers, namely the yellow square, flower and bomb. We poison the entire set of 547 stop sign images (i.e. BDS) and run our NEO defence on these 547 images, for each backdoor trigger. To compute the baseline accuracy, we construct a clean dataset (i.e. CS) of 547 images, where each image is a clean version of an image in BDS. The accuracy of the model on this clean dataset is 92.1%. The accuracy of the model on BDS drops to 5.48%, 1.83% and 2.93% for the yellow square, flower and bomb triggers, respectively. We run NEO on the poisoned set of 547 images (i.e. BDS) to produce fixed datasets (i.e. FS) and the accuracy of the model increases to 74.22%, 80.99% and 76.05% for the yellow square, flower and bomb poisoned models, respectively. We can observe from Table 10 and Table 11 that a major portion of the poisoned dataset is now predicting correctly after the NEO defence and the accuracy on the fixed set of images is also significantly close to the baseline.

We evaluate the accuracy of the fixed set (i.e. FS) on the dataset of 2622 images for the VGG face model poisoned by TrojanNN (cf. Table 10). We observe an accuracy of 74.87% on the respective set of clean images (i.e. CS). As seen in Table 10, the backdoor attack is successful and the accuracy drops to 6.37% if we poison all of the 2622 images. We run NEO on the set 2622 poisoned images to produce a fixed set of images (i.e. FS). The accuracy of the fixed set is 73.61%, which is very close to the baseline accuracy (i.e. 74.87%).

In the case of the MNIST [7] dataset (cf. Table 11), the accuracies of the clean set of 500 images is 100% and the accuracy drops to 15.75% for each of the poisoned models. This is due to our attack target class being 7. The 63 poisoned instances of class 7 are still correctly classified as 7. After running the NEO defence on this dataset, the accuracy of the fixed set returns to 100% for all the three patterns (i.e. Lateral L shape, Inverted L shape and 3 dots).

TABLE 11: NEO against various Patterns

USTS poisoned model [7]						
		Yellow Square		Flower		Bomb	
PC	CS	BDS	FS	BDS	FS	BDS	FS
Stop	504	30	406	10	443	16	416
Speedlimit	28	458	79	478	39	473	70
Warning	9	7	10	6	12	6	9
No pred	6	52	52	53	53	52	52
Total	547	547	547	547	547	547	547
MNIST Poisoned model [7]						
		Lateral L Shape		Inverted L Shape		3 dots	
PC	CS	BDS	FS	BDS	FS	BDS	FS
0	64	0	64	0	64	0	64
1	60	0	60	0	60	0	60
2	51	0	51	0	51	0	51
3	46	0	46	0	46	0	46
4	52	1	52	0	52	0	52
5	36	0	36	0	36	0	36
6	48	0	48	0	48	0	48
7	63	499	63	500	63	500	63
8	40	0	40	0	40	0	40
9	40	0	40	0	40	0	40
Total	500	500	500	500	500	500	500

Finding: The accuracy of fixed set produced by NEO is only ≈11%, ≈1% and 0% lower than the accuracy of the clean set for the USTS, VGG Face and MNIST models, respectively for various patterns.

RQ5: Can NEO recover the trigger patterns?

One of the most attractive properties of NEO is the fact that it is the first work, to the best of our knowledge, that can reliably reconstruct the backdoor trigger patterns. This empowers the user to test their model with potential

backdoor triggers and ensure the integrity and safety of their systems. The reconstruction of the backdoor trigger is a result of the design of Algorithm 2. The set of images in $check\_set$ that pass the Λ_T threshold (see Algorithm 2) will contain the backdoor. The user can use the respective backdoor triggers to try and poison other images and expose the backdoor. In other words, NEO helps the user to know the backdoor trigger, causing the backdoor attack to fail. This is because one of the key requirements of backdoor attacks is that they are stealthy.

Figure 7 and Figure 8 capture reconstructed backdoor triggers by NEO for TrojanNN [12] and BadNets [7], respectively.

Fig. 7: Reconstructed poisoned image from the poisoned TrojanNN VGG Face model [12]
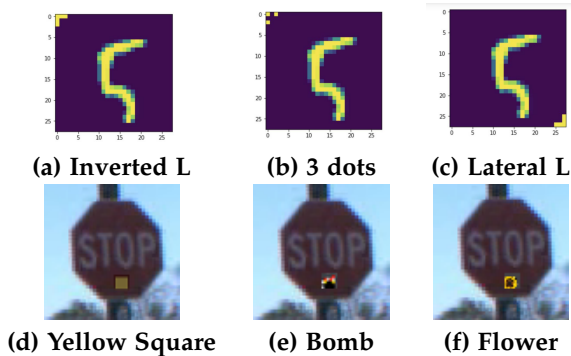

(a) Inverted L   **(b) 3 dots**   **(c) Lateral L**

(d) Yellow Square   **(e) Bomb**   **(f) Flower**

Fig. 8: Reconstruction from poisoned MNIST and USTS [7] models

> **Finding:** NEO effectively reconstructs poisoned examples so that the user can effectively test their models.

6 RELATED WORK

Testing and Verification of ML models: DeepXplore [16] employs differential white-box testing to find inputs that trigger inconsistencies. DeepTest [21] leverages metamorphic relations to discover error cases in DNNs. A recent work DeepGauge [13] measures the test quality and formalises a set of testing criteria for DNNs. SafeCV [26] takes a feature guided black-box approach to verify the safety of DNNs. DeepConcolic [18] performs concolic testing to discover violations of robustness. Aequitas [23] exposes individual fairness violations for ML models. Wang et al. [24] mutate the DNN model parameters to find adversarial samples. ReluVal [25] uses interval arithmetic [14] to estimate a DNN's

decision boundary by calculating tight bounds on the output of a model for a given range of inputs. Similarly, Reluplex [9] verifies properties of interest using SMT solvers. AI^2 [5] employs abstract interpretation to verify the robustness of a given input against adversarial attacks. AI^2 uses zonotopes to approximate ReLU inputs. The authors do not guarantee precision, but they do guarantee soundness. Another work [4], transforms the verification problem into an unconstrained dual formulation leveraging Lagrange relaxation and uses gradient-descent to solve the resulting optimisation problem.

All the aforementioned works cannot combat the backdoor attacks in machine learning. This type of attack is designed to be stealthy and cannot be discovered using conventional testing methods for Machine Learning. NEO is specifically developed to combat these kinds of attacks for Machine Learning models. As a result of the specific design of NEO, it is able to combat backdoor attacks effectively and provide a wide array of information to the users. Moreover, as explained in the preceding paragraph the main goal of all the works is to explore some specific properties (i.e. fairness, robustness). In contrast to this, the specific goal of NEO is identifying the backdoor attack and mitigating such an attack effectively.

Backdoors in ML: BadNets [7] poisons the training data to inject a backdoor in an ML model. They choose a pre-defined target label and a trigger pattern. The patterns are arbitrary in shape, (e.g. square, flower or bomb.) and the backdoor is injected into the model by training the network using the poisoned data.

TrojanNN [12] generates a backdoored model without interfering with the original training process and without accessing the training dataset. The approach is able to inject the backdoors using fewer samples. Additionally, they improve trigger generation by designing triggers that induce the maximum response for specific neurons of the DNN and aims to build a stronger connection between the neurons of a DNN and the backdoor trigger.

The goal of NEO is to defend against these classes of attacks in an efficient manner and reconstruct the backdoor trigger.

Backdoor defences in ML: A recent work [22] involves using tools from robust statistics to analyse the learned representation of classes, which they call the spectral signature. The idea behind this work is that when the training data is poisoned, there are two significant sub-populations. A small number of poisoned, mislabelled inputs and a large number of clean, correctly labelled inputs. Techniques from robust statistics and singular value decomposition are used to separate the two populations. The authors assume access to the poisoned dataset. NEO does not assume any access to a poisoned dataset and defends against a much stronger attack model.

Another work [11] prunes the neurons which seem to behave maliciously. It has been shown [1] that such a method causes a significant loss in performance for some models. Additionally, fine-pruning doesn't offer detection capabilities to identify backdoored images.

NeuralCleanse [1], a completely white-box approach formulates the problem as an optimisation problem to reverse engineer the backdoor trigger. Their mitigation technique involves computationally expensive retraining.

NEO presents a computationally inexpensive, blackbox approach to defend against backdoor attacks. NEO is also the first work, to the best of our knowledge that can also reliably reconstruct the trigger that users can investigate.

7 THREATS TO VALIDITY

Localised Triggers: NEO assumes the backdoor trigger to be relatively small and it is not designed to combat targeted backdoor attacks [3]. Nonetheless, NEO covers a variety of state-of-the-art backdoor attacks proposed recently [7], [12] and effectively detects as well as mitigates them.

Λ_T **threshold:** The effectiveness of NEO depends on the threshold Λ_T. If Λ_T is too low, we might have a high false positive rate, but if these parameters are set too high, then we may not be able to detect backdoors efficiently. Algorithm 4 aims to assist the user to find a range of values which are optimum for the Λ_T threshold, but cannot definitively determine an optimal value.

Theoretical Soundness Guarantee: NEO offers no theoretical guarantees that it can identify and mitigate all backdoor attacks. However, we extensively evaluated our defence and empirically show that NEO can effectively detect and mitigate backdoor attacks with minimal loss in baseline accuracy.

8 CONCLUSION

In this paper, we propose NEO, a novel approach to detect and mitigate backdoor attacks in arbitrary image classifier models. NEO requires neither the knowledge of model structure nor does it require access to the poisoned training set induced by the attacker. The design of NEO further allows us to reconstruct a backdoor attack – a feature that existing defences fail to provide in an accurate fashion. Thus, NEO not only defends against backdoors by switching to the correct prediction class, but it also breaks the stealthy nature of the attack by exposing the respective backdoor trigger. Our evaluation reveals that NEO can successfully detect and mitigate state-of-the-art backdoor attacks in a variety of image classifiers and only with minimal loss in the accuracy. Moreover, despite being a blackbox approach, NEO is more effective in thwarting backdoor attacks as compared to the state-of-the-art whitebox techniques. At its current state, NEO does not defend against targeted backdoor attacks [3] and backdoor attacks in other domains such as speech recognition [12]. Further work is needed to include such defence capabilities in the future.

NEO is a major step towards pushing the state-of-the-art in verification and validation of ML models, which bring along several fresh challenges due to their unique data-driven nature. Thus, to promote research in this area and reproduce our results, we have made our implementation and all experimental data publicly available:

https://github.com/sakshiudeshi/Neo

REFERENCES

[1] Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy, SP 2019, Proceedings, 20-22 May 2019, San Francisco, California, USA*.

[2] Anil Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–109, 1943.

[3] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *CoRR*, abs/1712.05526, 2017.

[4] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 550–559, 2018.

[5] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 3–18, 2018.

[6] François Goudail, Philippe Réfrégier, and Guillaume Delyon. Bhattacharyya distance as a contrast parameter for statistical processing of noisy optical images. *Journal of the Optical Society of America.*, 21(7):1231–1240.

[7] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR*, abs/1708.06733, 2017. URL: http://arxiv.org/abs/1708.06733, arXiv:1708.06733.

[8] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL: http://www.scipy.org/.

[9] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 97–117, 2017.

[10] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 9–48. Springer, 2012.

[11] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *Research in Attacks, Intrusions, and Defenses - 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings*, pages 273–294, 2018.

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

[13] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Deepgauge: multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 120–131, 2018.

[14] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009. URL: https://doi.org/10.1137/1.9780898717716, doi:10.1137/1.9780898717716.

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

[16] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 1–18, 2017.

[17] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[18] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 109–119, 2018.

[19] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[20] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[21] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 303–314, 2018.

[22] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral sig-natures in backdoor attacks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 8011–8021, 2018. URL: http://papers.nips.cc/paper/8024-spectral-signatures-in-backdoor-attacks.

[23] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. Auto-mated directed fairness testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 98–108, 2018.

[24] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. Adversarial sample detection for deep neural network through model mutation testing. In *Proceedings of the 41st Inter-national Conference on Software Engineering, ICSE 2019, Montréal, Canada, May 25 - May 31, 2019*, 2019.

[25] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 1599–1614, 2018.

[26] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, pages 408–426, 2018.