AI-Lancet: Locating Error-inducing Neurons to Optimize Neural Networks

Yue Zhao[1,2], Hong Zhu[1,2], Kai Chen[1,2,4,*], Shengzhi Zhang[3]
[1]SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, China
[3]Department of Computer Science, Metropolitan College, Boston University, USA
[4]Beijing Academy of Artificial Intelligence, China
{zhaoyue,zhuhong,chenkai}@iie.ac.cn,shengzhi@bu.edu

ABSTRACT

Deep neural network (DNN) has been widely utilized in many areas due to its increasingly high accuracy. However, DNN models could also produce wrong outputs due to internal errors, which may lead to severe security issues. Unlike fixing bugs in traditional computer software, tracing the errors in DNN models and fixing them are much more difficult due to the uninterpretability of DNN. In this paper, we present a novel and systematic approach to trace and fix the errors in deep learning models. In particular, we locate the *error-inducing* neurons that play a leading role in the erroneous output. With the knowledge of error-inducing neurons, we propose two methods to fix the errors: the neuron-flip and the neuron-fine-tuning. We evaluate our approach using five different training datasets and seven different model architectures. The experimental results demonstrate its efficacy in different application scenarios, including backdoor removal and general defects fixing.

CCS CONCEPTS

• **Security and privacy** → **Malware and its mitigation**; • **Computing methodologies** → *Neural networks.*

KEYWORDS

Neural networks, model optimization, backdoor removal

ACM Reference Format:
Yue Zhao[1,2], Hong Zhu[1,2], Kai Chen[1,2,4,*], Shengzhi Zhang[3]. 2021. AI-Lancet: Locating Error-inducing Neurons to Optimize Neural Networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3460120.3484818

[*]Corresponding Author

1 INTRODUCTION

Recently, deep neural network (DNN) has been widely applied in many areas, including computer vision, speech recognition, targeted advertisement, etc., mainly due to its accurate classification/prediction results. It also manifests itself in vital applications like autonomous driving [13] and medical diagnosis [27], as well as the cybersecurity domain like malware classification [59] and binary reverse engineering [11]. Despite these amazing advances, it is still far from dependable. For example, Amazon's Recognition software incorrectly matched nearly one-in-six of all athletes to some mugshots in a database during a test [3]. Besides the misclassification in certain scenarios, DNN models are also known to suffer from Adversarial Examples (AEs) and backdoor attacks [19, 38], which could lead them to produce wrong results when an adversarial patch or a trigger is applied to the inputs.

Generally, we can consider the wrong results produced by deep learning models are caused by their internal errors either unintentionally or deliberately. However, the lack of interpretability makes locating and fixing such errors, even defining such errors, quite challenging in DNN models. Existing works deal with the model errors by selecting (or generating) the inputs that cause the incorrect outputs, and adding them to the training dataset to fine-tune the original model [11] or train a new model [18, 39, 66]. However, the users may have access to only the validation data but not the original training data, making retraining or fine-tuning the model less possible. Sometimes, the model is trained using federated learning or online learning [14], which makes access to the appropriate training dataset even more difficult, if not impossible. Furthermore, fine-tuning a model with extra data may cause overfitting or catastrophic forgetting [49], thus downgrading the performance of the model, while retraining a large model is quite time-consuming and resource-consuming. Finally, both retraining and fine-tuning "blindly" repair the model without the knowledge of the actual errors, so the errors are fixed on a best efforts basis. Such practice is quite different than that in the domain of traditional computer software, where various debugging tools are used to pinpoint the coding errors that lead to incorrect outputs, and fix them accordingly. We believe it is highly demanded to have similar debugging capability for DNN models, thereby precisely locating the errors that contribute to the misclassification and fixing them accordingly.

Challenges. In deep learning models, given an input, the output can be attributed to neurons (weight parameters). Therefore, an error can occur when some neurons are activated, which induces erroneous features and ultimately lead to the erroneous output. In this paper, we call such neurons *error-inducing* neurons, since they play a leading role in the erroneous output. Inspired by the idea of fixing or patching source code in traditional software debugging, we consider first pinpointing the error-inducing neurons (weight parameters), and then fixing them accordingly to optimize the performance of deep learning models. However, there are at least two challenges to be addressed: (C1) Due to the lack of interpretability of deep learning models, one cannot directly read/analyze neurons to understand their functionality, thus concluding the error-inducing neurons, as what we have done to debug source code in traditional software. Meanwhile, it is nontrivial to accurately and efficiently locate the error-inducing ones from millions of neurons in modern deep learning models. (C2) Even with the error-inducing neurons located, fixing them in a logical way is still difficult, since all the neurons have been trained based on a large amount of data samples. Fine-tuning them with new data samples may cause overfitting or catastrophic forgetting problems, which downgrade the accuracy of the model.

In this paper, we propose AI-Lancet to address the above two challenges. To address (C1), we first locate the critical regions of the original misclassified sample x that lead to the misclassification, and generate the companion sample x' by removing the regions from x. Then, we perform the differential feature analysis of $< x, x' >$ to reveal the features that contribute to the misclassification. Finally, we compute all neurons' contribution to the features and locate the error-inducing neurons using a progressive-ablation method. To address (C2), we propose two methods to fix the error-inducing neurons based on the availability of training samples: the neuron-flip and the neuron-fine-tuning. The former is to directly reverse the sign of values of the error-inducing neurons (no need of any training data), thus eliminating their contribution to the output. The latter fine-tunes the error-inducing neurons by adjusting their values accordingly based on the extra training samples. Our neuron-fine-tuning mitigates the catastrophic forgetting problem compared with the existing similar approaches that fine-tune the whole model or a single layer of the model. We validate the neuron-flip method on the backdoor removal and neuron-wise fine-tuning method on the general defect fixing respectively.

We evaluate our approach with five different training datasets and seven different model architectures. For the backdoor removal, the attack success rate of backdoors is reduced from 100% to 1.1% on average, with the overall accuracy loss within 1% for all the 34 backdoored models. In contrast, the state-of-the-art method, i.e., unlearning used in Neural Cleanse [58], only reduces the attack success rate of backdoors to 36% on average, but downgrades the overall accuracy of the model by 2% on average (with the worse case of 4.1%). Regarding the general defect fixing, after locating and fine-tuning the error-inducing neurons, the accuracy of the optimized model increases 42% on average on the testing samples without reducing the model accuracy, which is 30% higher than that of fine-tuning the same number of randomly chosen neurons.

Contributions. The contributions are summarized as follows:

• We propose to fix output errors of deep learning models by pinpointing the error-inducing neurons, and present a novel and systematic approach to locate and fix them accordingly.

• We propose the neuron-flip and the neuron-fine-tuning methods to fix the error-inducing neurons. The former fixes the error effectively in a non-training way, while the latter mitigates the catastrophic forgetting problem with little loss to the overall accuracy.

• We implement and validate AI-Lancet with two different applications, including the the backdoor removal and the general defect fixing. Evaluation results show that our approach locates the error-inducing neurons precisely in both of the applications and improves the performance of the error fixing in terms of effectiveness compared with the state-of-the-art solutions.

2 BACKGROUND

Backdoors in neural networks. Backdoor, also known as Trojan Horse, is to inject a hidden and unexpected output into the model, which behaves normally until a specific trigger is presented as the input, causing the model to produce the predefined misclassification desired by the attacker. One typical approach to embed backdoors is to poison the training dataset [10, 19, 47, 68]. For example, BadNets [19] injects poisoned samples, which all include a specific trigger pattern and are tagged with the target label, into the original clean dataset, and utilizes the poisoned dataset to train the backdoored model. Another approach is to manipulate a clean model into a backdoored one [16, 20, 38, 55].

Recently, both backdoor detection [8, 30, 36, 58] and backdoor removal [35, 58] approaches have been proposed. Neural Cleanse [58] reconstructs the trigger based on its misclassification property, and distinguishes it from universal AEs (sharing the same misclassification property as the trigger) by assuming the trigger is much smaller than any universal AEs. Unlearning [58], also used by Neural Cleanse, places the reconstructed triggers onto clean training samples, labels them correctly, and trains the backdoored model with the revised training dataset. Fine-pruning [35] removes the backdoors by pruning and fine-tuning the model. Though seminal, existing backdoor removal approaches suffer from various limitations, i.e., unlearning relies on the clean training dataset and fine-tuning downgrades the accuracy on the clean data.

Catastrophic forgetting. Incremental learning allows a model to be updated on extra training data (of new labels), instead of retraining it from fresh on the whole training dataset (with the extra data included) [24]. Similar to incremental learning, continual learning aims to accommodate new knowledge while still retaining previously learned experiences [41]. However, both of them suffer from the same severe problem—catastrophic forgetting [49], i.e., a model fine-tuned with an unbalanced training dataset gets a significant performance downgrade on the original tasks. Existing approaches have been proposed to mitigate the catastrophic forgetting problem. Among them, [7, 44, 63] fine-tune the model by mixing a portion of the original training data with the new one, hoping to "refresh the memory" of the model. EWC [29] proposed by DeepMind and other similar methods [6, 67] identify and preserve significant parameters of the original model, hoping to "keep its memory". Moreover, knowledge distillation [23, 34, 44, 57] has also been applied to preserve the knowledge of the original model. So far, however, the catastrophic forgetting problem is still considered challenging since most of the existing approaches can only alleviate it [41] rather than fully eliminating it. Even worse, most approaches only performed well on MNIST, but poorly on more complicated datasets such as CUB-200 [61] and ImageNet [15].

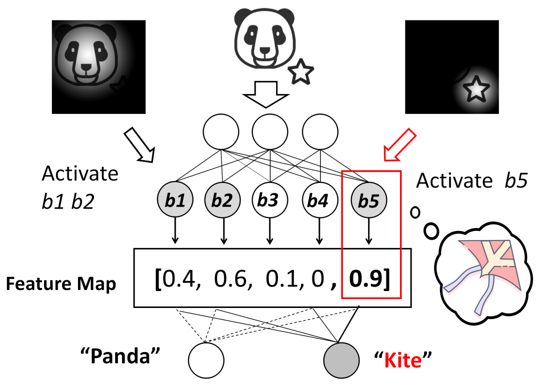

Figure 1: An example of misclassification.

3 OVERVIEW

3.1 Problem Statement

Neural networks are known to suffer from various problems that lead to erroneous outputs. The lack of interpretability makes identifying the cause of the errors in neural networks quite difficult, if not impossible, not to mention fixing them. Inspired by the practice of debugging in traditional computer software, we believe making it possible in the domain of neural networks will be extremely useful, thus producing more dependable neural network models. As discussed in Section 1, two fundamental challenges need to be addressed to achieve the above goal, but neither of them is easy in neural networks due to the lack of their interpretability. Obviously, without (C1) being well addressed, no ideal solution would be possible for (C2). Existing works bypass (C1) by retraining or fine-tuning the entire model [18, 39, 66] based on the inputs that cause the erroneous outputs. The fundamental problem of such works is that the exact errors are unknown, so the corresponding fixes are typically "blind", leaving the errors either fixed, partially fixed, or at large.

There are mainly two reasons that could lead to erroneous output in deep learning models, i.e., backdoors and general defects. The backdoor, also known as Trojan Horse, is deliberately embedded into the model during training, and when presented, will cause the model to produce the attacker-desired output [37, 60, 70]. There exist different types of backdoor attacks. In this paper, we focus on one of the widely used backdoor attack, the patch trigger, which can have different patterns, shapes, positions, connectivity and transparency. The backdoor is universal and targeted, i.e., images from any class attached by the trigger should be classified into the attacker-desired label. Generally, the backdoor can be injected in two different ways, poisoning the training dataset [19] or manipulating the connection weights of a clean model [38]. The general defect refers to a general input sample, rather than an adversarial input or backdoored input, which gets misclassified by a benign model incidentally. The defects are likely to be caused by either over-fitting or under-fitting over some features. For example, the class of "traffic light" in a pre-trained and open-sourced model by Facebook [2][1] may overfit the circle shape, thus misclassifying other objects with the circle as "traffic light" as well, e.g., "binoculars", "bell", "cannon", "Chau Gong", etc. In this paper, we validate our error fixing approach against the backdoor attacks and the general defects in Section 5.

[1]The model is a pre-trained VGG11 based on ImageNet dataset

3.2 Preliminary

Generally, an artificial neuron is defined as a mathematical function that consists of several weight-parameters [4]. Although several works use the neuron to represent the output value of the mathematical function [18, 39], in this paper, we use the neuron to indicate the weight parameters, which will not change once the model has been trained. Therefore, the proposed error-inducing neuron locating and fixing approach indicate the corresponding operations on the weight parameters. Consider an unexpected misclassification produced by a model for a given input sample. The neurons that play a leading role in the erroneous output are called the *error-inducing neurons*. The error-inducing neurons falsely extract a sufficient amount of features critical to the erroneous output from some regions in the input and finally result in the erroneous output. We name such features as the *error-inducing features*, and such regions in the input as the *error-inducing regions* in this paper. Figure 1 illustrates a simplified deep learning model with three layers misclassifies a "panda" image as a "kite". The activation values of the five neurons in the second layer make up the feature map of this layer. It is obvious that the region with the panda inside activates neurons $b1$ and $b2$, while the region with the star inside activates neuron $b5$ (neither $b3$ nor $b4$ is activated). Neuron $b5$ probably overfits to some of the kite's features during training, which causes the misclassification. In this example, the region that activates $b5$ is the *error-inducing region*; the activation value of $b5$ is the *error-inducing feature*; $b5$ is the *error-inducing neuron.*

Assumptions. On the one hand, the error locating demands a misclassified input sample x with the EI regions, so the error can be activated. Sometimes, however, such a misclassified input sample is not available directly, so a dedicated algorithm is desired to restore the EI regions and generate a sample x to trigger the error. For example, when we suspect a model is the victim of a backdoor attack but have no input samples to trigger the backdoor, we can resort to exiting backdoor reconstruction solutions [10, 19, 47, 68] to restore the triggers that cause the misclassification. On the other hand, given a misclassified sample x, we need to know the misclassified label and also the corresponding correct label (if not misclassified). The former can be easily obtained by observing the model output against such a misclassified sample x, while the latter can be known via little manual effort.

3.3 AI-Lancet Approach

The AI-Lancet approach is illustrated in Figure 2 with the error-inducing (EI) neurons locating and the error fixing.

Error Locating. We start by revealing the EI regions in the input sample, and then extract the EI features that are activated by the EI regions of the input. Finally, the EI neurons can be located with the guidance of the EI features.

• **Step 1**: Given a specific error, we first generate the image pairs $< x, x' >$, where x is the input sample that got misclassified by the DNN model and x' is the companion sample similar to x but

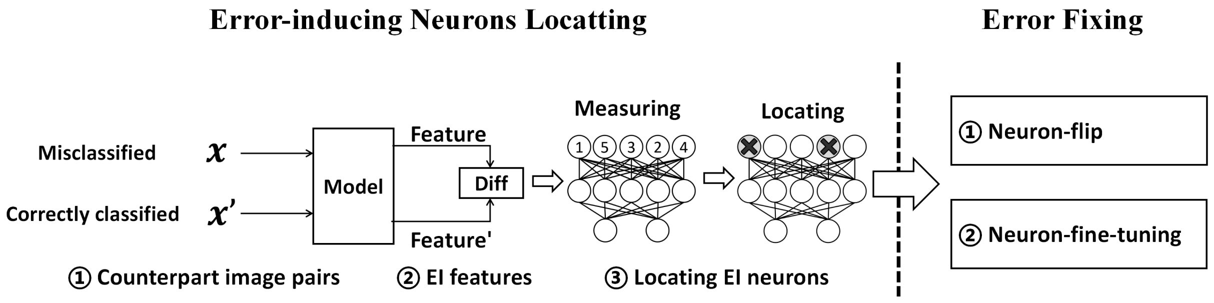

Figure 2: Overview of AI-Lancet

with the EI regions removed. Intuitively, x' can preserve most of the important features of the correct label in x and be classified correctly by the model.

• **Step 2**: We input both x and x' to the DNN model and extract the hidden layer features f and f', respectively. f extracted from x consists of the EI features and features of the correct label. f' extracted from x' mainly consists of features that contribute to the correct label since x' can be classified correctly. Hence, we perform the differential feature analysis between f and f' to expose the EI features.

• **Step 3**: With the guidance of the EI features, we assign important values to all neurons and finally locate the EI neurons using a progressive-ablation method.

Error fixing. With the EI neurons located against a specific error, we design two methods to fix the error:

• **Neuron-flip** is to reverse the sign of values of the EI neurons directly, thus eliminating their contribution to the output. It can be utilized when we cannot resort to a training-based fixing way. Neuron-flip works best for the scenarios when the error is caused by "add-on" to the original model deliberately (e.g., the backdoors embedded into the original model). In such scenarios, the located EI neurons should be dedicated to the errors, so flipping their values should mainly fix the error, without hurting the accuracy of the original model on other clean inputs.

• **Neuron-fine-tuning** fine-tunes the EI neurons by adjusting their values accordingly based on some extra samples, which can be used in various scenarios like the backdoor, the error caused by inherent defects of the model, etc. For the inherent defects, the located EI neurons not only contribute to the error, but may also play some roles in the classification of other inputs. Hence, neuron-flip turns out not an ideal solution for such defects. We can rely on neuron-fine-tuning to fix such defects, which can also mitigate the catastrophic forgetting problem.

4 DETAILED METHODOLOGY

4.1 Locating Error-inducing Neurons

Generating the image pair. Given a specific error, we first generate the image pair $< x, x' >$, where x is the input sample that got misclassified by the DNN model and x' is the companion sample similar to x but with the EI regions removed. Intuitively, x' can preserve most of the important features of the correct label in x

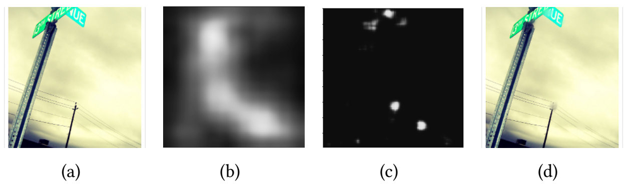

Figure 3: The image pair generation. (a) The original misclassified sample. (b) Heatmap (Grad-CAM). (c) Mask of the EI regions. (d) The companion sample.

and be classified correctly by the model. We consider two different situations based on the availability of x. When x is unavailable, we rely on a dedicated algorithm to generate it with the EI regions restored to activate the error. For example, if we suspect a model the victim of a backdoor attack but have no input samples to trigger the backdoor, we can resort to exiting backdoor reconstruction solutions [58], which can produce the backdoored sample (i.e., x in our paper) as well as the trigger pattern (i.e., the EI regions in our paper). Removing the trigger pattern from the sample x can directly obtain the clean sample x'.

When x is available, we need to generate x' by removing the EI regions from x. Hence, we design an optimization scheme to locate the EI regions from x and then generate the companion sample x' by blurring the EI regions on x. We define a *companion matrix* with the same dimension as the input sample and all values ranging from zero to one. Each value in the companion matrix denotes whether the same position in the input sample belongs to an EI region or not, with a close-to-one value indicating true and a close-to-zero value indicating false. Then we optimize the companion matrix iteratively until the generated companion sample (by blurring the EI regions according to the matrix) can be classified correctly by the model. The optimization should be done by enforcing three constraints on the EI regions of a misclassified sample: (1) The regions should be as small as possible to ensure the companion sample still keeps most of the other regions in the original input that contributes to the correct label. (2) The regions should demonstrate clear boundaries, i.e., the values of the companion matrix should be either close to one or zero, which makes the EI regions more precise. (3) The regions should be more clustered since too many scattered tiny regions (e.g., scattered noise) increase the possibility of making the companion sample be an adversarial example.

We address the above three constraints by designing the loss function and the preprocessing methods. On the one hand, we include the sum of the companion matrix *Mask* in the loss function of the optimization, which leads to revising it with fewer values close to 1, thus limiting the size of the EI regions (Constraint 1). On the other hand, we design two preprocessing methods for *Mask* in the optimization. The first is to apply the *Sigmoid* function to reinforce the boundary (Constraint 2). *Sigmoid* is a squashing function that maps values less than zero to values close to 0 and values more than 0 to values close to 1 [5]. The second is to apply a blurring function to *Mask* to decrease the scattered regions in *Mask* and make the EI regions more clustered (Constraint 3). Details of the optimization is shown in Appendix.

Figure 3 illustrates the image pair generation when x is available. Figure 3 (a) is with the true label of "street sign", but misclassified as "crane" with a high probability (i.e., 0.75) by VGG16 model [50]. Figure 3 (c) shows the EI regions generated by our method and Figure 3 (d) is the corresponding companion sample classified as the "street sign" correctly. Clear boundaries and small regions play a significant role in retaining normal features and exposing key EI features. The four regions highlighted in Figure 3 (c) seem reasonable, since they look like the geometric structure of the "crane" (the label of the misclassification). We also consider alternatives to generate companion samples, e.g., leveraging the generated heatmap (e.g., CAM [69] or Grad-CAM [48]) to highlight the important regions that cause the input image misclassified. However, heatmap is not accurate enough to satisfy *Constraint 1* or *Constraint 2*. Figure 3 (b) shows the heatmap generated based on the misclassified label "crane", which highlights a large region without clear boundary. Moreover, the highlighted region by heatmap also overlaps the critical regions for the input to be correctly classified as the "street sign". Hence, heatmap is not an ideal solution to generate the EI regions as well as the companion sample.

Identifying the EI features. Generally, the above generated image pair $< x, x' >$ look quite close to each other. We feed them to the DNN model and extract their features f and f' at each layer based on the forward computation processes before each layer in the model. Afterwards, we calculate the differential features df between f and f' at each layer respectively. Since df do not consider the effect of computation after each layer to the final layer, it is possible that not all df are critical to the final erroneous output. Hence, we re-weight df with gradient-based information obtained via the backward propagation after each layer to the final layer, since the gradient of the erroneous output score with respect to df can indicate the contribution of df to the erroneous output. We compute the EI Features (EIf) for each hidden layer as follows:

$$\begin{aligned} df_j &= F_j(x) - F_j(x'), j \in [1, n] \\ EIf_j &= (\alpha + g(\frac{\partial y^c}{\partial df_j})) \cdot df_j \end{aligned} \tag{1}$$

where j denotes the layer index, n is the number of all the layers in a DNN model, $F_j(\cdot)$ extracts the features of the input sample at the layer j, $g(\cdot)$ functions as a normalization method via $g(x) = x/max(x)$ and y^c is the output score (before softmax) of the erroneous class c for x. We do not assign the gradient-based weights to df_j directly, because the backward propagation cannot assess all the calculations happened in the forward processes. For models with a RELU function, the calculations that lead to the activation values below zero will not be captured by the backward propagation. Therefore, α keeps the impact of the original df_j and $g(\cdot)$ emphasizes the importance indicated by gradients.

Valuating neurons. For a given misclassified sample x and its EIf that leads to the misclassification, we need to examine the contribution of neurons to EIf to locate the EI neurons (i.e., weight parameters). In the fully-connected layers, the features and the neurons are associated in an one-to-one correlation. However, the parameter sharing scheme [43] is widely used in the convolutional layers to control the number of free parameters, which makes the correlation between the features and the neurons quite complicated. We assign Importance Values (IV) to neurons according to the gradients of EIf against them to denote their contributions to EIf. In particular, IV is calculated as below:

$$IV_j = \frac{\partial EIf_j}{\partial w_j}, j \in [1, n] \tag{2}$$

where j is the layer index and n is the number of all the layers in a DNN model. We use w_j to denote all neurons of the layer j.

Algorithm 1 Progressive Neuron Ablation Algorithm

```
Input: the original model $Model^0$, the misclassified sample $x$ with the
    correct label $l$, all the neurons $N_k$ of the layer $k$.
Output: EI neuron set $EG$
1: function ABLATE($Model^0$)
2:     $EG = NULL$;
3:     $N_s = decent\_sort(N_k)$;
4:     for each neuron $i$ in $N_s$ do
5:         $Model^{i+1} = ZEROIZE(Model^i, i)$;
6:         $y = F_{Model^{i+1}}(x)$;
7:         if $argmax(y) == l$ then
8:             $UPDATE(EG)$;
9:             break;
    return $EG$
```

decent_sort sorts input neurons in a descending order. *ZEROIZE* sets the corresponding neuron in the model to zero. F_{Model} outputs the prediction based on the model. *UPDATE* includes the so-far-tested neurons in the EI neuron set.

Progressive neuron ablation. We identify the EI neurons based on the following criterion: the neurons (as few as possible), when "ablated", will cause the model to recognize the originally misclassified input correctly, are the EI neurons. By "ablated", we mean setting the weights of the neurons to zero, so they will not contribute to the classification. Overall, for each layer, we sort all neurons based on their IV in descending order, since the neuron with a larger IV is more likely to be an EI neuron. Afterwards, we start to ablate neurons in order progressively and test whether the resultant model can recognize the originally misclassified input correctly.

Algorithm 1 elaborates the progressive neuron ablation approach. Line 2 initializes the EI neurons set EG. For the layer k, we first sort all its neurons in descending order and obtain the sorted set N_s (Line 3). Then for each neuron i in N_s, we zeorize its value in the model (Line 5) and test if the originally misclassified input x now can be classified correctly on the resultant model (Line 6-7). If so, we update the EI neuron set EG by including the so-far-tested neurons

(Line 8) and finish this layer (Line 9). Otherwise, we iterate to the next neuron in N_s and repeat the above procedures. We run the above algorithm at each layer of the model, and finally harvest the EI neurons for all layers of the model. It can be quite time-consuming if we exactly follow the above progressive approach to ablate all the neurons one by one, considering the number of neurons on each layer and the number of layers in a model. Therefore, we extend the progressive neuron ablation approach by integrating an improved binary search algorithm to accelerate the procedure of locating the EI neurons at each layer (Refer to Appendix for details).

4.2 Error Fixing

We propose the neuron-flip and the neuron-fine-tuning to fix errors. The neuron-flip method could be utilized when we cannot resort to a training-based fixing way. It works best for the scenario when the error is caused by the "add-on" to the original model deliberately, as stated in Section 3.3. For instance, the backdoors embedded into the original model can be viewed as an "add-on" trained into the original model. Neuron-fine-tuning could be used in various scenarios, e.g., the error caused by inherent defects of the model, etc. For the inherent defects, the located EI neurons not only contribute to the errors, but may also play some roles in the classification of other inputs. For example, some misclassified samples are caused by either over-fitting or under-fitting to some features. Simply flipping the located IE neurons may result in some downgrade of the model accuracy. Generally, an inherent defect can often be found in the validation or test process.

Neuron-flip. The EI neurons on one layer should either transfer or amplify the EI features to the next layer, leading to a misclassification. Hence, Neuron-flip prevents such EI features from propagating to the next layer by reversing the sign of values of the located EI neurons directly. Since the EI neurons are dedicated to the errors, flipping their values should mainly fix the error, without hurting the accuracy of the original model on other clean inputs.

Specifically, we first need to generate K input samples with the same error and get the corresponding companion samples as well. For instance, with the trigger pattern obtained from a backdoor detection approach, we can apply it on K clean samples x' to get K backdoored samples, i.e., the misclassified samples x. For each pair of images $< x, x' >$, we locate the EI neurons using Algorithm 1. Each neuron in the model is marked with a value $V \in \{0, 1, .., K\}$, which is the number of times the neuron being voted as an EI neuron by all the pairs of images. Then we target at one specific layer (the paragraph below discusses layer selection) and rank all neurons in this layer with the descending order based on their V. Finally, we start flipping the neurons based on their rank and stop when the resultant model no longer misclassifies the backdoored samples. Assume i is the index of each neuron N_i and k is the index of the last flipped neuron. Then neuron-flip is defined as follows.

$$Flip(N_i) = \begin{cases} -N_i, & if \quad i \geq k \\ N_i, & otherwise \end{cases} \tag{3}$$

We target at one specific layer to apply neuron-flip. On the one hand, no matter at which layer to block the EI features from propagating to the next layer, the model will no longer misclassify the input samples, so any layer works to fix the error. On the other hand, targeting at one layer with fewer neurons involved causes less impact to the accuracy of the original model. Hence, we choose the layer at which the largest number of misclassified input samples can find the EI neurons. For the layers with the same or very close number of misclassified samples that can locate the EI neurons, we then select the layer that the percentage of the located EI neurons to the total neurons on that layer is the smallest, since flipping fewer neurons means less impact to the original model.

Neuron-fine-tuning. We fine-tune the model by adjusting the values of the located EI neurons accordingly based on the extra samples. Fine-tuning a model with an extra dataset may suffer from the catastrophic forgetting problem, which causes the fine-tuned model downgrades on the recognition of other samples. To overcome the catastrophic forgetting problem, we adopt joint training and knowledge distillation approaches, which are commonly utilized in the incremental learning [24]. The former fine-tunes the model with a mixed training dataset that contains not only the new samples but also the old samples. The old samples could be from the original test dataset, the training data, and even some samples collected by the model owner since we need only a small number of old samples. The latter measures the distance of features between the original model and the fine-tuned model, and adds it into the loss to preserve the knowledge of the original model.

Overall, our fine-tuning approach is as below:

$$\begin{aligned} L_1(x) &= ||y - y_t||_2 \\ L_2(x) &= ||f^* - f|| \\ L(x) &= \sum_{N_e+N_o} L_1(x) + \sum_{N_o} L_2(x) \end{aligned} \tag{4}$$

where y is the output of the misclassified sample x, y_t is the the class embedding of the correct label, f^* and f are features of the last layer extracted by the original model and the fine-tuned model respectively. Therefore, $L_2(x)$, based on distillation approach, enforces the features extracted by the two preceding models to be similar. N_e is the training set built upon the misclassified samples and N_o is a small set of samples from the original training dataset. Hence, $L_1(x)$ defines joint training based on the two preceding datasets to reduce overfitting. The gradient is defined as $J(\theta) = \frac{\partial L(x)}{\partial \theta}$, where θ includes all the neurons to fine-tune using the Adam optimizer [28]:

$$\theta_{i+1} = \theta_i - \gamma_\theta \cdot Adam(J(\theta_i)) \tag{5}$$

We also target EI-neurons of one specific layer to apply neuron-fine-tuning, and the layer selection follows the same principle as that when applying neuron-flip.

Neuron-fine-tuning requires a set of misclassified samples N_e, which can be collected in the following two ways. On the one hand, given a misclassified sample x, we can apply different image transformations, e.g., rotation, etc. over x to obtain multiple misclassified samples with the same or similar error. On the other hand, we can enlarge the misclassified samples N_e by including those with similar EI neurons as the existing misclassified samples. In particular, we first obtain the model M' by ablating the located EI neurons from the original model M. Then, any sample misclassified by M but correctly classified by M' with the class probability larger than

δ^2 is more likely to trigger the similar error as the existing misclassified samples, thus should be included. N_o is simply a small set of the original training samples, e.g., 0.4%.

5 APPLICATIONS

AI-Lancet is a systematic solution to locate and fix errors in DNN models. Below, we validate it using two different application scenarios, the backdoor removal and the general defect fixing.

5.1 Backdoor Removal

Backdoor attacks can be considered as an error dedicated embedded by the attackers into DNN models, which can also be fixed by AI-Lancet. As far as we know, the state-of-art backdoor mitigation method is unlearning [58], which trains DNN models to unlearn the embedded triggers. Though seminal, unlearning requires a certain amount of training data (10% of the original training dataset in Neural Cleanse [58]), and its mitigation performance highly depends on the quality of the reconstructed triggers. Hence, its performance of removing large triggers drops significantly, since large triggers are difficult to be reconstructed with high fidelity. Moreover, even with 10% training data and a high quality reconstructed trigger, unlearning may downgrade the accuracy of the original model [49].

Using AI-Lancet to remove backdoors in DNN models is quite straightforward. First, we reconstruct the trigger patterns as well as the backdoored sample inputs x (i.e., the misclassified sample) using existing backdoor detection approaches, like the one proposed in Neural Cleanse [58]. The companion sample x' can be obtained simply by removing the trigger pattern from x. Thus, the image pair $< x, x' >$ can be generated. Note that in this step, we do not need to know the real trigger. Instead, the generated trigger can work well. Second, we locate the EI neurons (neurons that contribute to the backdoor attack in this case) based on the algorithm in Section 4.1. Finally, we apply the neuron-flip to the model to remove the backdoor, since the backdoor can be considered as "add-on" embedded into the original model as stated in Section 3.3. Our evaluation in Section 6.2 shows that the backdoor success rates are reduced from 100% to 0.8% on average, and the accuracy loss on the clean samples is only 0.7% on average, which outperforms the state-of-art backdoor mitigation approaches like unlearning [58].

5.2 General Defect Fixing

General defects may exist in DNN models due to the low quality of the training data or imperfect model architecture. For instance, Tulio Ribeiro et al. [45] test a model that differentiates wolves from huskies and find it always recognize the animal with the snow in the background as a wolf. To fix this kind of defect, one can fine-tune or retrain the entire model, which is always costly. Fine-tuning the model with incremental learning could lead to catastrophic forgetting problem, causing a significant performance downgrade. Hence, first locating the EI neurons like AI-Lancet makes it possible to operate on a few neurons to fix the error, with little impact on the overall accuracy.

Using AI-Lancet to fix the general defects in DNN models is as straightforward as backdoor removal, except the construction of the image pair $< x, x' >$. Typically, a defect can be identified during model development or testing, which can be considered as the misclassified sample x. However, it is not straightforward to identify the EI regions in x that cause the misclassification, thus obtaining the companion x' is also nontrivial. Hence, we first use the proposed counterpart sample generation to obtain x', thus the image pair $< x, x' >$ can be built. Second, we locate the EI neurons (neurons that cause x misclassified in this case) based on the algorithm in Section 4.1. Finally, we apply the neuron-fine-tuning to fix the general defects, since the located EI neurons may not only contribute to the misclassification, but also play some roles in the classification of other inputs. Our evaluation in Section 6.3 shows the accuracy of the optimized model increases 42% on average on the testing dataset with originally-misclassified samples without reducing the model accuracy, which is 30% higher than that of fine-tuning the same number of randomly-chosen neurons.

[2]In our evaluation, δ is set as 0.5. A smaller value of δ can help to collect more misclassified samples.

Table 1: Dataset Statistics

Dataset	Labels	Input size	Taining data	Test data
MNIST	10	28*28	55000	10000
GTSRB	43	32*32	96750	32250
CIFAR-10	10	32*32	50000	10000
YouTube-Face	1595	224*224	1595000	159500
ImageNet	1000	224*224	1281167	50000

6 EVALUATION

6.1 Experimental Setup

Datasets and models. We utilize five popular datasets, including Hand-written Digit Recognition (MNIST) [32], Traffic Sign Recognition (GTSRB) [52], Canadian Institute For Advanced Research (CIFAR-10) [31], Face Recognition (YouTube-Face) [62] and ImageNet (ILSVRC2012) [46] in our experiment. Details of the five datasets are shown in Table 1, and the model architectures of MNIST, GTSRB, CIFAR-10 and Youtube-Face are shown in Table 11~10 in Appendix. They mainly consist of convolutional (Conv) layers and fully-connected (FC) layers. For the large dataset ImageNet, we evaluate it on four different popular models, i.e., VGG11 as well as VGG16 [50], ResNet18 [22] and GoogleNet [54]. In particular, VGG11 and VGG16 consist of three FC layers, as well as eight and thirteen Conv layers respectively. ResNet18 is designed based on the residual learning framework, which has one Conv layer, one FC layer, nine basic shortcut blocks. GoogleNet is characterized by inception architecture, which has three Conv layers, one FC layer, and nine inception blocks.

Configuration for backdoor attacks. As we mentioned in Section 2, backdoors can be embedded via two methods: manipulating the weights of a pre-trained model [16, 38] and poisoning the training dataset [10, 19]. Regarding the former, we adopt two released Face Recognition backdoored models: *Trojan Watermark* and *Trojan Square*, which are trained on the VGG-Face dataset provided by [38]. Regarding the latter, we follow BadNets [19] to inject the backdoors by poisoning the five datasets mentioned above. For each of the five datasets, we modify the training dataset by injecting a portion (10%) of adversarial input labeled as the target label. The adversarial inputs are generated by applying a backdoor trigger to the clean images.

Backdoor triggers. We design two different kinds of backdoor triggers: the regular trigger and the scattered transparent trigger. The former is a square pattern placed at the corners of the original input, without covering any important region of the original input, e.g., faces of people or letters of traffic signs. They are designed with five different sizes, ranging from 2% ∼ 31.6% of the original input. The shapes and positions of the triggers are similar to Neural Cleanse [58]. The latter is evenly scattered over the entire input with 50% transparency[3] and is likely to cover some important regions of the original input. They are designed with three different sizes, ranging from 3.4% ∼ 16.1% of the original input. The shapes, transparency, and positions of triggers are similar to ABS [36]. Examples of triggers are shown in Figure 8 and Figure 6 in Appendix.

Configuration for general defects. We evaluate the general defects fixing using the ImageNet (ILSVRC2012) [46] dataset. We select misclassified samples from the test dataset. Firstly, we divide the test dataset into two equal sub-datasets $Test_1$ and $Test_2$, each containing 25,000 samples. Secondly, we test $Test1$ and $Test_2$ on the given model to select the misclassified samples and form M_Test_1 and M_Test_2 respectively. As described in Section 4.2, we select samples that are misclassified by M (the original model) but correctly classified by M' (model pruned located EI neurons) as samples with similar errors. Then we select 5,000 samples (0.4%) [4], and add the joint training samples to M_Test_1 to form the augmented M_Test_1[5]. Finally, the original model is fine-tuned with the augmented M_Test_1 and evaluated on $Test_2$ and M_test_2.

Platform. All our experiments are conducted on a 64-bit Ubuntu 18.04 system with an Intel(R) Xeon(R) E5-2620 v4@2.20GHz processor and 2 Nvidia Titan X (Pascal) GPUs (each with 12 GB memory).

6.2 Effectiveness of Backdoor Removal

We evaluate the success rate of backdoor attacks and the accuracy of the clean sample classification before and after the backdoor removal. For each dataset, we test the success rate of the backdoor attacks with all samples in the test dataset (not used in the backdoor removal) by attaching the original trigger on them. Below, we first present the performance of our AI-Lancet, and then compare it with unlearning, the state-of-art backdoor mitigation approach used in Neural Cleanse [58].

AI-Lancet Performance. Overall, AI-Lancet performs well to mitigate the backdoor attacks on different models with different types of backdoor triggers. The backdoor mitigation results for the regular triggers and Trojan Square/Watermark [38] are shown in Table 2. The attack success rate of backdoor attacks drops from 99.8% to 0.8% on average, with the classification accuracy increasing 0.8% on average. The results of the scattered transparent triggers are shown in Table 3. The attack success rate of backdoor attacks drops from 100% to 1.5% on average, with the classification accuracy only reducing 0.6% on average. The above results demonstrate that AI-Lancet can effectively mitigate the backdoor attacks with little impact on the classification accuracy. We also notice that AI-Lance performs extremely well on Trojan Square and Trojan Watermark, with the backdoor attack success rates down to 0% for both of them after its fixing. Furthermore, AI-Lancet even increases the overall accuracy for both of the models by 12.1% and 9.9% after the fixing, probably because the backdoored (i.e., EI) neurons in those Trojan attack models have a negative impact on the overall accuracy.

We also verify whether existing backdoor detection approaches can detect all the backdoored models accurately, since the backdoor removal of AI-Lancet depends on their backdoor detection. We utilize the anomaly index proposed in Neural Cleanse to detect the backdoored models, and the results are shown in the fifth column in Table 2 and Table 3. Neural Cleanse considers the anomaly index value larger than two indicates the model was injected with backdoors without false alarms. Based on the results in Table 2 and Table 3, we can see that most backdoored models (32 out of 34) can be detected.

Comparison with unlearning. We adopt the same unlearning configuration as Neural Cleanse[6]. The unlearning reduces the backdoor attack success rate to 25.6% and 38.2% on average for the regular triggers and scattered transparent triggers, respectively, compared to 0.8% and 1.5% of our AI-Lancet. The results indicate that some models are still vulnerable to the original triggers after unlearning, e.g., CIFAR10 and Youtube Face. Furthermore, unlearning gets 2.2% (the largest is 3.7%) and 2.3% (the largest is 4.1%) downgrade of the overall accuracy on average for the regular triggers and the scattered transparent triggers, respectively, compared to 0.4% and 0.1% of our AI-Lancet.

The reason that our AI-Lancet outperforms unlearning can be that unlearning highly depends on the fidelity of the reconstructed triggers to remove them. However, large triggers typically are more difficult to be reconstructed with high fidelity. As shown in Table 2 and Table 3, unlearning performs poorly on middle and large triggers, and can hardly remove the backdoors from the backdoored models trained based on YouTube Face (with the backdoor attack success rate still over 70% after its removal). In contrast, even if the reconstructed triggers do not completely match the originally embedded triggers, as long as they can help locate part of the backdoored (i.e., error-inducing) neurons, AI-Lancet can still eliminate their contribution to the activation of backdoored features, thus effectively remove them. Meanwhile, flipping those EI neurons only does not incur too much classification accuracy downgrade to the original model.

[3]We define 50% transparency as: $IMG_{backdoor} = IMG_{original} * (1 - mask) + 0.5 * (Trigger + IMG_{original}) * mask$, where $IMG_{original}$ is the original clean input, $IMG_{backdoor}$ is the backdoored input and $mask$ is a matrix composed of 0 or 1, with 1 indicating the corresponding position is occupied by the trigger.

[4]The samples are not limited to be training samples. For example, they could be selected from the test samples or some collected samples by the owner since they are a quite small number of samples. Less than 5,000 samples will lead to significant performance degrade for the fine-tuning models based on our experience.

[5]As mentioned in Section 4.2, we use the joint training to fine-tune the model using a mixed training dataset that contains the selected new samples and the old samples from the original training dataset.

[6]We fine-tune the model for one epoch using an updated training dataset. The updated dataset includes 10% clean samples from the original training dataset, and the reconstructed trigger is added to 20% of the clean samples without modifying their correct labels.

Table 2: Results of Backdoor Removal on Different Datasets (Regular Trigger)

Datasets	Trigger size	Before removing			After removing					
					Unlearning		Neuron-prune		Neuron-flip	
		Back_succ	Clean_succ	Anomaly	Back_succ	Clean_succ	Back_succ	Clean_succ	Back_succ	Clean_succ
MNIST	4*4*1 (2%)	100.0%	99%	11.4	0.3%	97.3%	10%	98.5%	0%	97%
	6*6*1 (4.6%)	100.0%	99%	8	0.8%	96.8%	48%	98.6%	2.1%	98.3%
	8*8*1 (8.2%)	100.0%	99%	6.4	1%	96%	98%	99%	0%	98.4%
	10*10*1 (12.8%)	100.0%	98.9%	2.6	2.2%	96.1%	77%	98.8%	0%	98%
	12*12*1 (18.4%)	100.0%	98.8%	1.1	-	-	-	-	-	-
CIFAR10	6*6*1 (3.5%)	99%	82.3%	9.8	19.3%	81.5%	98%	82.3%	0%	81.9%
	8*8*1 (6.3%)	100.0%	82.3%	9.1	46.2%	81.4%	48%	81.6%	4.4%	80.2%
	10*10*1 (9.8%)	99%	81.5%	2.6	32%	81.4%	100%	81.3%	3.6%	80%
	12*12*1 (14%)	100.0%	82.4%	1.5	41.2%	81.1%	99%	81.8%	0%	81.7%
	14*14*1 (19.1%)	100%	82.8%	0.78	-	-	-	-	-	-
GTSRB	6*6*1 (3.5%)	100%	97.1%	6.2	0%	94.7%	76.9%	97.2%	0.5%	97.1%
	8*8*1 (6.3%)	100.0%	96.7%	3.1	0%	94%	80%	96.7%	0%	96.6%
	10*10*1 (9.8%)	100.0%	97%	2.7	24%	94.1%	92%	96.5%	0.7%	96.5%
	12*12*1 (14%)	100.0%	97.3%	2.6	7%	94%	100%	97%	0%	97%
	14*14*1 (19.1%)	100.0%	96.4%	1.6	21.3%	93%	100%	96%	0.1%	96.1%
Youtube	8*8*1 (0.13%)	100.0%	99.8%	20.8	0.1%	96.5%	100%	99.7%	0.5%	98.7%
	63*63*1 (7.9%)	100.0%	99.7%	4.1	80.1%	96.6%	100%	99.7%	0.1%	99.4%
	63*63*2 (15.8%)	100.0%	99.7%	3.5	82%	96.1%	100%	99.7%	0.2%	99%
	63*63*3 (23.7%)	100.0%	99.7%	2.9	69%	96%	100%	99.7%	4.1%	99.1%
	63*63*4 (31.6%)	100.0%	99.8%	1.5	81%	96.1%	100%	99.7%	0.2%	99.2%
Square	60*60*1 (7%)	99.9%	85.1%	5.8	3.6%	93.6%	18.8%	97.2%	0%	97.2%
Watermark	- (7%)	97.6%	87.3%	3.7	1.3%	93.3%	8.5%	97.2%	0%	97.2%
Average	- (11.1%)	99.8%	93.7%	5.5	25.6%	92.4%	77.7%	95%	0.8%	94.5%

Back_succ*: The attack success rate of backdoored samples. **Clean_succ***: The accuracy of clean samples. **Anomaly**: The anomaly index of the infected models.

Table 3: Results of Backdoor Removal on Different Datasets (Scattered Transparent Trigger)

Datasets	Trigger size	Before removing			After removing					
					Unlearning		Neuron-prune		Neuron-flip	
		Back_succ	Clean_succ	Anomaly	Back_succ	Clean_succ	Back_succ	Clean_succ	Back_succ	Clean_succ
MNIST	3*3*3 (3.4%)	100.0%	98.5%	2.9	0.5%	96.1%	41.3%	98.7%	0%	98.3%
	3*3*6 (6.8%)	100.0%	98.9%	1.9	0.4%	96.7%	72%	98.9%	0%	98.8%
	3*3*9 (10.3%)	100.0%	98.7%	2.5	0.5%	96.2%	94%	98.7%	0.3%	98.3%
CIFAR10	4*4*3 (4.7%)	100%	82.1%	10.4	76%	80.4%	99%	81.7%	9.6%	80%
	4*4*6 (9.4%)	100.0%	81.6%	3.1	65%	80.7%	100%	80.9%	5.6%	80%X
	4*4*9 (14%)	100%	81.5%	2.2	27%	81%	99%	81.1%	0.5%	79.1%
GTSRB	4*4*3 (4.7%)	100%	96.7%	5.2	0%	94.2%	99%	96.9%	0.7%	96.7%
	4*4*6 (9.4%)	100.0%	98.5%	5.3	15%	94.4%	100%	98.5%	0.2%	98.5%
	4*4*9 (14%)	100.0%	95.1%	3	18.5%	94.8%	100%	95%	0%	95%
	30*30*3 (5.3%)	100.0%	99.7%	8.3	81%	96.2%	100%	99.7%	0.2%	99.6%
	30*30*3 (10.7%)	100.0%	99.7%	7.5	87%	96.1%	100%	99.7%	0%	99.4%
Youtube	30*30*9 (16.1%)	100.0%	99.7%	7.3	88%	96.4%	100%	99.7%	0.4%	99.3%
Average	- (9%)	100%	94.2%	5	38.2%	91.9%	92%	94.1%	1.5%	93.6%

Back_succ*: The attack success rate of backdoored samples. **Clean_succ***: The accuracy of clean samples. **Anomaly**: The anomaly index of the infected models.

Table 4: Flipping Baseline Neurons VS Flipping EI Neurons

Datasets	Regular Trigger				Scattered & Transparency Trigger			
	Baseline		Flip EI neurons		Baseline		Flip EI neurons	
	Back_succ	Clean_succ	Back_succ	Clean_succ	Back_succ	Clean_succ	Back_succ	Clean_succ
MNIST	1.3%	98.5%	0.5	97.9%	35.3%	95.8%	0.1%	98.5%
CIFAR10	52.4%	81.5%	2%	81%	99%	81.3%	5.2%	79.7%
GTRSB	16.3%	96.3%	0.3%	96.7%	100%	96.6%	0.3%	96.7%
Youtube Face	98.2%	99.7%	1%	99.1%	97%	99.7%	0.2%	99.4%
Average	42.1%	94%	1%	93.7%	58.1%	93.4%	1.5%	93.6%

Back_succ*: The attack success rate of backdoored samples. **Clean_succ***: The accuracy of clean samples.

Comparison with pruning. We also compare the neuron-flip (used to remove backdoors in AI-Lancet) with neuron-prune (zeroizing the corresponding neuron parameters). Specifically, for each backdoored model, we prune the same EI neurons with the neuron-flip to evaluate the backdoor attack success rate as well as the clean sample classification accuracy. Table 2 and Table 3 show that neuron-prune can reduce the backdoor attack success rate to 77.7% and 92% on average for the regular triggers and scattered transparency triggers respectively, which indicate that pruning the located EI-neurons could work to some extend, but is not as effective as flipping them to mitigate the backdoors. Actually, only when most of the backdoored neurons are pruned, the backdoors can be mitigated effectively. In Neural Cleanse, pruning 30% of neurons for GTSRB reduces the backdoor attack success rate to nearly 0%, but the classification accuracy drops by 5.06%. In contrast, we find that flipping a small number of neurons usually is enough to mitigate the backdoors. The average ratios of the flipped neurons to the number of all the neurons of the layer are 0.04 (MNIST), 0.008 (CIFAR10), 0.01 (GTSRB) and 0.007 (Youtube Face) for the regular triggers. Detailed ratios for all the backdoored models and their corresponding layers are shown in Table 14 in Appendix.

Comparison with NAD. We compare AI-Lancet with one of the existing state-of-art approaches, Neural Attention Distillation (NAD) [33]. Firstly, NAD needs to access 5% of the clean training data, but our method does not. Secondly, following the same configuration as NAD, we implement our approach using CIFAR-10 dataset (consistent with NAD), and adopt the same backdoor embedding method and backdoor triggers as NAD. The results indicate AI-Lancet achieves better performance in general. For the BadNets [19] attack, our approach reduces the backdoor attack success rate to 1.7% with only 3.2% downgrade of the overall accuracy, compared to 4.77% and 4.48% of NAD respectively. Regarding the Trojan attack [38], our approach reduces the backdoor attack success rate to 2.1% with 3.3% accuracy decline, compared to 19.63% and 2.08% of NAD respectively.

The effectiveness of the EI neurons. We also compare the EI neurons flipping with the baseline neurons flipping, which is to flip those neurons with the most different behavior between the clean inputs and the backdoored inputs, i.e., the largest activation value difference. Consider that EI neurons flipping chooses K neurons at the layer l of the backdoored model based on the algorithm in Section 4.1. For the baseline neurons flipping, we target the same layer l, rank all neurons of this layer in a descending order based on their activation differences between the clean inputs and the backdoored inputs, and set the top K neurons as the baseline neurons. We flip such baseline neurons and measure the backdoor attack success rate as well as the clean sample classification accuracy. The results are shown in Table 4, with the backdoor success rates averaged over all different trigger sizes for each dataset due to the space limitation (refer to Table 12 in Appendix for details). According to Table 4, flipping baseline neurons in the model can reduce the backdoor attack success rate to 42.1% and 58.1% on average for the regular triggers and the scattered transparent triggers respectively, compared with 1% and 1.5% of flipping the same number of EI neurons. Hence, we conclude that flipping the same number of baseline neurons cannot mitigate the backdoors as effectively as flipping the EI neurons.

Impact of triggers' properties. We consider the impact of different properties of backdoor triggers in our experiments, including the size, the transparency, the pattern, the connectivity (one piece or scattered pieces), the position (covering the important regions of the original input or not) and the shape (square pattern or non-square pattern). The trigger pattern used in Table 2 is one piece pattern, without transparency and does not cover the important regions. The trigger patterns in Table 3 are scattered pieces with 50% transparency and may cover part of the important regions. Examples of trigger patterns are shown in Figure 8 and Figure 9 in Appendix. The results in Table 2 and Table 3 show that the properties, including the transparency, the pattern, the connectivity and the position, do not pose significant impact on the performance of AI-Lancet, with the backdoor success rate dropping from 99.8% to 0.8% on average after removal. In contrast, the backdoor success rate of unlearning is still up to 81% as the trigger size increases to 5.3% for Youtube Face after its backdoor removal. We also test triggers of different shapes, either consisting of several thin lines or several individual pixels (as shown in Figure 7 in Appendix) using GTRSB dataset. For each shape of trigger, we use three different sizes, i.e., 3.1%, 6.2%, and 9.3% of the entire input image, respectively. The results indicate that our approach is effective on such non-square patterns as well. For the trigger of thin lines, we reduce the backdoor attack success rate from 100% to 4% on average without the overall accuracy decline. Regarding the trigger of individual pixels, the backdoor attack success rate drops from 100% to 3.1% on average without the overall accuracy decline.

Scalability of AI-Lancet. To demonstrate the scalability of AI-Lancet approach, we extend it beyond the vision domain to the speech domain, and also evaluate it on other model architectures. We evaluate AI-Lancet using two speech recognition backdoor models released by authors of [38], used to recognize spoken numbers in English. The Trojan trigger is some background noise, causing the model to recognize any speech integrated with the trigger as the target number. For one model, AI-Lancet can reduce the backdoor attack success rate from 97% to 0% with only 3% degrade on the overall accuracy. For the other model, AI-Lancet reduces the backdoor attack success rate to 0%, with the overall accuracy increased from 78% to 86%. The evaluation results indicate that AI-Lancet is also effective in mitigating the backdoors embedded into speech recognition models. Furthermore, to demonstrate the scalability of AI-Lancet on other model architectures, we implement it using a Feedforward model with four fully-connected layers using the same speech recognition dataset as [38]. The results indicate that AI-Lancet reduces the backdoor attack success rate to 2.2% with only 1% accuracy degradation.

6.3 Effectiveness of General Defects Fixing

We evaluate the general defects fixing with four different models based on the ImageNet dataset. We measure the overall accuracy A based on $Test_2$ and the accuracy A' of the samples with similar errors from M_Test_2 to demonstrate the performance of fixing. The samples with similar errors are those misclassified by the original model but recognized correctly by the fixed model with the EI neurons ablated. We also compare our AI-Lancet with other fine-tuning methods, including fine-tuning the same number of randomly-chosen neurons on the same layer or the last layer and fine-tuning all the neurons of a whole layer. We choose the last layer of each model to fine-tune all the neurons due to the following two reasons. On the one hand, most of the existing incremental learning works [24, 25, 34] fine-tune the last layer with satisfactory performance, so we follow the common practice. On the other hand, we tried to fine-tune the last six layers of VGG16 with the same joint training samples and the same hyper-parameters. The results (refer to Figure 5 in Appendix) show that fine-tuning the last layer better addresses the catastrophic forgetting problem. We adopt

Table 5: Performance of General Defect Fixing

Layers	AI-Lancet		*Random Neurons		*Last Layer		*Last& Random		R*	S*
	A	A'	A	A'	A	A'	A	A'		
***ResNet18**	**69.4%**	**0%**								
Conv1	69.4%	39%	69.4%	2%	68.3%	34%	69.5%	0%	0.0002	369
Basic2	69.6%	34%	69.4%	5%	69.2%	34%	69.5%	0%	0.0008	321
Basic3	69.4%	43%	69.4%	32%	68%	41%	69.5%	3%	0.0077	558
Basic4	69.5%	28%	69.5%	21%	69%	29%	69.4%	1%	0.0006	454
Basic5	69.4%	39%	69.4%	27%	68.4%	39%	69.5%	0%	0.001	386
Basic6	69.4%	48%	69.4%	6%	68%	41%	69.4%	0%	0.0004	284
Basic7	69.6%	75%	69.5%	67%	68.2%	40%%	69.5%	1%	0.0004	139
Basic8	69.4%	35%	69.4%	12%	69.1%	35%	69.4%	2%	0.0077	310
Basic9	69.7%	35%	69.5%	20%	68%	40%	69.5%	11%	0.0038	272
Fc10	69.5%	37%	69.5%	5%	68.9%	41%	–	–	0.0024	162
***VGG16**	**71.3%**	**0%**								
Conv1	71.4%	40%	71.4%	3%	71.2%	43%	71.4%	3%	0.0006	73
Conv2	71.4%	35%	71.3%	9%	70.2%	35%	71.3%	3%	0.0033	92
Conv3	71.3%	42%	71.3%	3%	70.5%	43%	71.3%	2%	0.0009	118
Conv4	71.3%	45%	71.3%	12%	69.8%	44%	71.4%	3%	0.0006	451
Conv5	71.4%	59%	71.3%	5%	70.2%	50%	71.4%	3%	0.0002	130
Conv6	71.3%	36%	71.4%	4%	70.8%	38%	71.4%	3%	0.0002	210
Conv7	71.3%	50%	71.3%	2%	70.7%	51%	71.4%	2%	0.001	137
Conv8	71.5%	55%	71.4%	9%	70.2%	45%	71.3%	2%	0.0009	198
Conv9	71.3%	46%	71.3%	2%	71.1%	37%	71.4%	3%	0.0009	155
Conv10	71.3%	39%	71.3%	16%	70.5%	39%	71.3%	2%	0.0018	180
Conv11	71.3%	33%	71.3%	4%	71.1%	41%	71.3%	2%	0.0016	163
Conv12	71.4%	40%	71.3%	8%	70.7%	44%	71.4%	6%	0.003	357
Conv13	71.4%	41%	71.4%	4%	70.6%	45%	71.3%	5%	0.0027	349
FC14	71.5%	45%	71.3%	9%	71.1%	47%	71.4%	7%	0.0082	162
FC15	71.4%	28%	71.3%	26%	70.7%	46%	71.4%	16%	0.0229	250
FC16	71.3%	37%	71.3%	13%	70%	51%	–	–	0.0078	317
***VGG11**	**68.7%**	0%								
Conv2	68.7%	42%	68.7%	7%	66%	38%	68.7%	2%	0.0005	329
Conv4	68.7%	40%	68.7%	12%	67.9%	46%	68.7%	3%	0.0006	225
Conv8	68.7%	43%	68.7%	8%	68.4%	48%	68.7%	5%	0.0008	166
Conv9	68.7%	38%	68.7%	32%	68.6%	43%	68.8%	19%	0.0081	167
FC11	68.7%	35%	68.7%	0%	68.8%	36%	–	–	0.0017	125
***GoogleNet**	**67.5%**	**0%**								
Conv1	67.5%	36%	67.5%	5%	66.8%	34%	67.5%	2%	0.037	363
Conv2	67.5%	45%	67.5%	20%	67%	46%	67.5%	4%	0.0004	142
Inception4c	67.6%	41%	67.5%	15%	67.1%	44%	67.5%	5%	0.0013	281
Inception4e	67.5%	46%	67.5%	21%	67%	50%	67.5%	15%	0.0005	127
FC	67.6%	40%	67.5%	30%	67.2%	44%	67.6%	8%	0.0009	201

R*: The ratio of EI neurons. **S***: The number of training samples.
***ResNet18, *VGG16, *VGG11 and *GoogleNet**: The baseline accuracy (A and A') of the four original models.
***Random Neurons**: Results of fine-tuning the randomly-chosen neurons of the same layer as that of the EI neurons. ***Last & Random**: Results of fine-tuning the randomly-chosen neurons of the last layer. ***Last Layers**: Results of fine-tuning all the neurons of the last layer.

two popular catastrophic forgetting mitigation methods, including feature-distillation and joint training in our experiment.

For different misclassified samples, different layers might be chosen to fine-tune the located EI neurons based on Section 4. Hence, for each model, we intend to choose multiple misclassified samples with each of them relying on different layers to fix, i.e., misclassified samples for all the layers for ResNet18 and VGG16, and five layers for VGG11 and GoogleNet. The corresponding misclassified samples for each layer are listed in Table 13 in Appendix. Table 5 shows the experimental results. "AI-Lancet" represents our method, "Random Neurons" represents fine-tuning the randomly-chosen neurons at the same layer, "Last Layer" represents fine-tuning all the neurons at the last layer, and "Last & Random" represents fine-tuning randomly-chosen neurons at the last layer. The number of the randomly-chosen neurons is the same as the number of EI neurons.

Table 5 shows that fine-tuning the EI neurons achieves the best performance on the overall accuracy A and the accuracy A' on the samples with similar errors. In contrast, fine-tuning the randomly chosen neurons has little impact on A, but causes A' to be 28% lower than fine-tuning the EI neurons on average. Fine-tuning all the neurons on the last layer achieves a good performance on A', but the overall accuracy A drops up to 2.7%. The reason that fine-tuning the EI neurons outperforms is due to its targeted fixing. We also observe that even the best performance of A' can hardly reach 50%, probably because not all the selected samples for testing are with similar errors. For example, some samples near the decision boundaries may also easily be affected by variations in the feature maps, making them be classified correctly by the fixed model. To measure the ratio of such samples, we ablate the same number of randomly-chosen neurons at the same layer and select samples misclassified by the original model but correctly classified by the randomly-ablated model. We find that even the random neurons could select a small number of misclassified samples (53 on average). Detailed numbers for each of the four models are shown in the fourth row and fifth row in Table 7. The result indicates that M_test_1 and M_test_2 may include some irrelevant samples, which limits the improvement of A'.

Furthermore, we also measure the percentage of samples that can locate EI neurons. On average, 67% of misclassified samples can locate EI neurons. Detailed numbers for each of the four models are shown in Table 7. It is possible when the EI features and the benign features tangle in the same region tightly (unable to separate them in the two-dimension space), locating the EI regions becomes quite challenging, thus the EI neurons.

Efficiency. The last row in Table 7 shows the time to locate the EI neurons for one misclassified sample (averaged on 500 misclassified samples). The first addend in each cell is the time consumption of generating the companion sample for the misclassified sample, and the second one is the time of locating the EI neurons for it. The training samples and test samples selection for each model takes less than 300 seconds. The time used for model fine-tuning (20 epochs) for VGG11, VGG16, ResNet18, and GoogleNet is 540 seconds, 360 seconds, 480 seconds, and 420 seconds, respectively.

6.4 Accuracy of EI Neurons Locating

Since the performance of error fixing highly depends on the EI neurons locating, below we evaluate the accuracy of EI neurons locating. Since it is impossible to know the ground truth EI neurons in the case of general defects, our evaluation focuses on the backdoor removal. Some backdoor attack approaches [37, 60, 70] can embed the backdoors leveraging several specific neurons of a pretrained model, so the backdoored neurons are known and unique, which can be the ground truth to test the accuracy of our EI neurons locating. Our backdoor injection follows *Trojan Attack* [38] to select the backdoored neurons, but uses different model retraining method when embedding the backdoor. Trojan Attack retrains all the layers between the residence layer of the selected neurons and the output layer. However, we concern that retraining any more neurons besides the selected neurons could introduce uncertainty to the ground truth neurons. Therefore, instead of retraining the following layers after the layer where the selected backdoored neurons reside in, we only retrain the selected neurons, making them the only neurons that implant the backdoors.

Table 6: Results of Locating EI-neurons

GTRSB	Configuration			Reconstructed trigger		Neuron-prune		Neuron-flip		
Layer	Back_succ	Clean_succ	Ratio*	TPR	FNR	Back_succ	Clean_succ	Back_succ	Clean_succ	Ratio_f*
FC7	99.5%	96.2%	0.002	93%	4.8%	78%	94.8%	0%	94.3%	0.0003
FC6	100%	96.4%	0.0002	82.4%	4.9%	0.2%	96.4%	0%	96.4%	0.0005
Conv5	98.4%	93%	0.001	80.4%	4.7%	0.7%	95%	0%	95%	0.0006
Conv4	90%	89%	0.002	61%	4.8%	0%	94%	0%	94%	0.004
Conv3	61%	87.5%	0.004	66%	4.6%	26%	86%	4%	89%	0.0002
Conv2	75%	64%	0.002	76%	4.8%	12.3%	71%	8%	72%	0.0002

Ratio*: $Ratio = N_b/N$ where N_b is the number of the neurons that are trained to embed the backdoor and N is the number of all the neurons of the layer. **Ratio_f***: $Ratio_f = N_f/N$ where N_f is the number of the flipped/pruned neurons and N is the number of all the neurons of the layer.

Table 7: Summary of Locating EI Neurons

	VGG11	VGG16	ResNet18	GoogleNet
Parameters	132,863,336	138,357,544	33,161,024	6,624,904
***FIR ratio**	80%	79.1%	81.4%	60%
***FBN ratio**	76.4%	76%	67.7%	48.8%
Selected	203	209	326	271
Random	39	28	76	71
Time	4+2.5 sec	4.3+4.3 sec	5.3+1.8 sec	3.6+2.1 sec

***FIR ratio**: The ratio of samples that can locate the EI regions.
***FBN ratio**: The ratio of samples that can locate the EI neurons.

In the experiments, we target each layer of the GTRSB model and take a square-shaped, with the size of 14 ∗ 14, placed on the top-left corner of the input, as the Trojan trigger mask, and generate triggers accordingly (Examples of the generated triggers are shown in Figure 6 in Appendix). Then we locate the EI neurons based on the reconstructed trigger. Finally, we evaluate the neuron-flip and neuron-prune based on the reconstructed trigger to verify the mitigation performance against such attack. Table 6 shows the results of EI neurons locating when choosing different layers of GTRSB model[7] We use TPR (true positive rate) and FPR (false positive rate) as the evaluation metrics. $TPR = n_b/N_b$, where n_b is the number of the located EI neurons and N_b is the number of total backdoored neurons of the layer. $FPR = n_a/N_a$, where n_a is the number of normal neurons but mis-located as the EI neurons and N_a is the number of total normal neurons of the layer. In our experiments, we intend to always keep an acceptable FPR (less than 5%) and calculate the TPR. We can see that the TPR of the reconstructed trigger is 76% on average (up to 93%) of the reconstructed trigger, which indicates AI-Lancet can locate the EI-neurons accurately.

We observe that neuron-flip mitigates such backdoor attack successfully with 2% attack success rate on average after removal. Interestingly, the overall accuracy of some flipped models even gets improved (up to 8%), rather than downgraded. This is consistent with the findings of Neural Cleanse that the models infected by the Trojan attack also get improved on the overall accuracy after unlearning from the training samples, with the difference that the improvement of neuron-flip does not come from the training samples. We infer that the backdoored neurons (i.e., EI neurons) may interfere with the models' recognition on other clean input samples. Therefore, flipping them means removing the interference factors, which mitigates the degradation of the classification accuracy. We also notice that the performance of neuron-prune for such backdoor attack is better than that for the BadNets attack. The reason may be that such backdoor attack implants the backdoor into fewer neurons, thus a small number of EI neurons would be enough to cover most backdoored neurons. In Table 6, the performance of neuron-prune on the layer of "FC6", "Conv5" and "Conv4" is almost the same as that of Neuron-flip. The ratio of flipped/pruned neurons of these the models (the last column in Table 6 is close to the ratio for the ground truth (the fourth column in Table 6).

[7]We did not evaluate the accuracy of EI neurons locating for the first layer (Conv0) and the second layer (Conv1), because the backdoored model obtained by selecting backdoored neurons in those two layers got only about 20% attack success rate and about 50% clean data classification accuracy

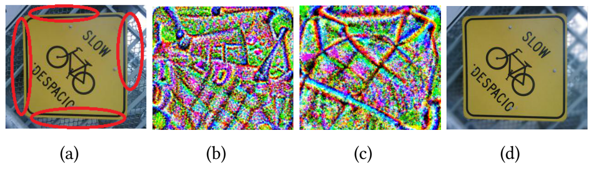

Figure 4: Error Explanation

7 DISCUSSION

Error explanation. We adopt the feature visualization method [1] to visualize the EI neurons, and use one example to discuss the root cause analysis for misclassified samples with the help of the EI neurons. Figure 4 (a) is the sample with the true label of "street sign" but misclassified as a "shopping basket" by VGG16. The red circles in Figure 4 (a) are the located EI regions. Hence, we infer the reason of the misclassification could be either the rectangular border of the "street sign" is very similar to the shape of a "basket" or the barbed wire behind the street sign is quite similar to the mesh structure of the "basket". Figure 4 (b) visualizes the class of the "shopping basket" using the feature visualization method. Although quite informative about the "shopping basket", it still cannot help determine the root cause of the misclassification, since it contains both of the above-mentioned features (the rectangular shape and the mesh structure). Figure 4 (c) is the visualization result of the located EI neurons of the class "shopping basket", which highlights the characteristics of the mesh structure. Therefore, the misclassification might be caused by the barbed wire behind the street sign in Figure 4 (a). To verify the hypothesis, we blur the barbed wire with a different intensity and generate a sample image as in Figure 4 (d). The experimental results show that most of the similarly-blurred images are recognized correctly as the street sign. We also blur the rectangular border of the "street sign", but most of the blurred images are still misclassified as the "shopping basket" with the confidence value increased from 0.45 to 0.6. Therefore, we conclude that the barbed wire is a more important factor for the misrecognition than the rectangular border.

Security analysis. Aware of our AI-Lancet approach, the attackers can design adaptive attacks. Firstly, the attackers can embed a backdoor activating the neurons that also contributes significantly to the classification of the benign samples. We implement a similar backdoor on the GTSRB model and evaluate whether AI-Lancet can remove this kind of backdoor effectively. Specifically, we generate a trigger based on the clean model, making it highly activated at one layer (the last but two layer in our experiment). Then we retrain the last two layers of the model to embed the backdoor using the generated trigger. In the loss function design, the trigger should also highly activate the important neurons of the last two layers, which are the top 30% activated neurons for the clean samples. Finally, we evaluate AI-Lancet using the backdoored model. The evaluation results show that AI-Lancet successfully reduces the backdoor attack success rate from 100% to 0% with only 0.8% degradation of the overall accuracy. The reason is that although the trigger can activate benign neurons, some of the backdoored neurons are still revealed by AI-Lancet. Furthermore, it can be seen that a small number of neurons (even all of them are benign) typically has little impact on the overall accuracy of a model. Secondly, the attackers can mislead the companion samples construction. Specifically, they can generate adversarial examples to mislead the EI region's locating. For this kind of adaptive attack, we can detect it by integrating existing adversarial examples detection methods [12, 51, 65]. Finally, the attackers can induce AI-Lancet to flip/fine-tune more neurons by embedding backdoors that affect as many neurons as possible, but embedding such backdoors inevitably downgrades the overall accuracy.

Meanwhile, generating an adaptive attack is nontrivial. First, the backdoor should activate enough benign neurons at each layer, not only one layer, because we remove the backdoor by selecting the layer with the fewest EI neurons. Second, the benign neurons should be activated heavier than the backdoored neurons. Finally, the reconstructed trigger (not only the original trigger) should also satisfy the above two constraints. All the above constraints make this adaptive attack challenging, and such attack is more likely to downgrade the overall accuracy.

Dependency. For the application of backdoor removal, AI-Lancet relies on existing backdoor reconstruction approaches, including Neural Cleanse [58], TABOR [21] and ABS [36]. Although ABS can handle feature space backdoor as well, Neural Cleanse is used in our experiment since AI-Lancet targets the patch-trigger only.

8 RELATED WORK

Model debugging. Although the lack of interpretability and the complexity (e.g., millions of parameters) of DNN models make it a difficult task to debug them, there exist several seminal works exploring the topic. Anurag et al. [17] identifies implementation bugs in machine learning based image classifiers using metamorphic testing. It focuses on machine learning software applications, which is similar to debugging traditional software programs. SEQ2SEQ-VIS [53] is a visual debugging tool for sequence-to-sequence models, which describes the origin of decisions made by the seq2seq model by relating internal states to relevant training samples. It helps engineers to analyze the model interactively, but the debugging still highly relies on manual effort.

Roozbeh et al. [66] generates flip points that lie on decision boundaries to mitigate undesirable behaviors of the model. A flip point is any point (sample) that lies on the boundary between two output classes, e.g. MODE [39] debugs models via state differential analysis, i.e., locating faulty features with heatmap differentials between correctly classified inputs and misclassified inputs. The differential heatmap is used as guidance to select existing or new inputs to form a new training dataset, which is then used to retrain the model. However, MODE mainly focuses on the overall over-fitting and under-fitting bugs for one class. It cannot provide support to identify plausible and distinct root cause leading to model classification errors. To overcome such limitation, HUDD [18] applies clustering on the heatmap of misclassified inputs to provide more information to help researches analyze the root cause. All the above three works need to adjust the training dataset based on the misclassified inputs and then retrain the model.

Model testing. Other works explore model testing via fuzzing or assertion approaches. For instance, Tian et al. [56] applies the neuron coverage metric to DNNs by performing natural image transformation and find the errors based on the idea of metamorphic testing [9]. Neuron coverage metric is introduced by Pei et al. in [42], and then used by DeepHunter [64] and TensorFuzz [40] to test and verify neural networks with coverage-guided fuzzing. Compared to random test samples, these approaches could find more errors. Besides fuzzing, Daniel et al. [26] propose a method to prevent machine learning models from more errors via assertions.

9 CONCLUSIONS

In this paper, we proposed AI-Lancet to optimize the deep learning models by locating the error-inducing neurons and fixing them using either neuron-flip or neuron-fine-tuning methods. The first one is neuron-flip, which is a non-training method with no need for training data. The second one is neuron-fine-tuning, which fixes errors by fine-tuning error-inducing neurons. The results show that AI-Lancet could get good performance on different applications, including the backdoor removal and general defects fixing.

ACKNOWLEDGMENTS

IIE authors are supported in part by the National Key R&D Program of China (2020AAA0140001), NSFC U1836211, Beijing Natural Science Foundation (No.JQ18011), National Top-notch Youth Talents Program of China, Youth Innovation Promotion Association CAS, Beijing Academy of Artificial Intelligence (BAAI) and a research grant from Huawei.

REFERENCES

[1] 2017. Feature Visualization. https://distill.pub/2017/feature-visualization/.
[2] 2017. Torchvision Models. https://pytorch.org/vision/stable/models.html.
[3] 2018. NYtimes. https://www.nytimes.com/2018/07/26/technology/amazon-aclu-facial-recognition-congress.html.
[4] 2021. Artificial Neuron. https://en.wikipedia.org/wiki/Artificial_neuron.
[5] 2021. Sigmoid function. https://en.wikipedia.org/wiki/Sigmoid_function.

[6] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. 2018. Memory Aware Synapses: Learning what (not) to forget. In ECCV.

[7] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. 2018. End-to-End Incremental Learning. In Computer Vision – ECCV 2018, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Cham, 241–257.

[8] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. 2019. DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks. In IJCAI.

[9] T. Y. Chen, S. C. Cheung, and S. M. Yiu. 2020. Metamorphic Testing: A New Approach for Generating Next Test Cases. arXiv:2002.12543 [cs.SE]

[10] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526 (2017).

[11] Zheng Leong Chua, Shiqi Shen, P. Saxena, and Zhenkai Liang. 2017. Neural Nets Can Learn Function Type Signatures From Binaries. In USENIX Security Symposium.

[12] Gilad Cohen, Guillermo Sapiro, and Raja Giryes. 2020. Detecting adversarial samples using influence functions and nearest neighbors. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 14453–14462.

[13] H. Cui, V. Radosavljevic, F. Chou, T. Lin, T. Nguyen, T. Huang, J. Schneider, and N. Djuric. 2019. Multimodal Trajectory Predictions for Autonomous Driving using Deep Convolutional Networks. In 2019 International Conference on Robotics and Automation (ICRA). 2090–2096.

[14] Ashok Cutkosky and Kwabena Boahen. 2017. Online Learning Without Prior Information. In Proceedings of the 2017 Conference on Learning Theory (Proceedings of Machine Learning Research, Vol. 65), Satyen Kale and Ohad Shamir (Eds.). PMLR, 643–677. http://proceedings.mlr.press/v65/cutkosky17a.html

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09.

[16] J. Dumford and W. Scheirer. 2018. Backdooring convolutional neural networks via targeted weight perturbations. arXiv preprint arXiv:1812.03128 (2018).

[17] Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M. Rao, R. P. Jagadeesh Chandra Bose, Neville Dubash, and Sanjay Podder. 2018. Identifying Implementation Bugs in Machine Learning Based Image Classifiers Using Metamorphic Testing. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (Amsterdam, Netherlands) (ISSTA 2018). Association for Computing Machinery, New York, NY, USA, 118–128. https://doi.org/10.1145/3213846.3213858

[18] Hazem Fahmy, Mojtaba Bagherzadeh, Fabrizio Pastore, and Lionel Briand. 2020. Supporting DNN Safety Analysis and Retraining through Heatmap-based Unsupervised Learning. arXiv:2002.00863 [cs.SE]

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

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

[21] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. 2019. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. arXiv preprint arXiv:1908.01763 (2019).

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

[23] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. 2018. Lifelong Learning via Progressive Distillation and Retrospection. In Proceedings of the European Conference on Computer Vision (ECCV).

[24] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. 2019. Learning a Unified Classifier Incrementally via Rebalancing. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[25] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. 2018. Less-forgetful Learning for Domain Expansion in Deep Neural Networks. AAAI (2018).

[26] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. 2020. Model Assertions for Monitoring and Improving ML Models. In Proceedings of the 3rd MLSys Conference. Austin, TX, USA.

[27] Daniel S. Kermany, Michael Goldbaum, Wenjia Cai, Carolina C.S. Valentim, Huiying Liang, Sally L. Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, Justin Dong, Made K. Prasadha, Jacqueline Pei, Magdalene Y.L. Ting, Jie Zhu, Christina Li, Sierra Hewett, Jason Dong, Ian Ziyar, Alexander Shi, Runze Zhang, Lianghong Zheng, Rui Hou, William Shi, Xin Fu, Yaou Duan, Viet A.N. Huu, Cindy Wen, Edward D. Zhang, Charlotte L. Zhang, Oulan Li, Xiaobo Wang, Michael A. Singer, Xiaodong Sun, Jie Xu, Ali Tafreshi, M. Anthony Lewis, Huimin Xia, and Kang Zhang. 2018. Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning. Cell 172, 5 (2018), 1122 – 1131.e9.

[28] Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. Computer Science (2014).

[29] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences 114, 13 (2017), 3521–3526.

[30] Soheil Kolouri, Aniruddha Saha, Hamed Pirsiavash, and Heiko Hoffmann. 2020. Universal Litmus Patterns: Revealing Backdoor Attacks in CNNs. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

[31] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[32] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Urs A Muller, Eduard Sackinger, Patrice Simard, et al. 1995. Learning algorithms for classification: A comparison on handwritten digit recognition. Neural networks: the statistical mechanics perspective 261 (1995), 276.

[33] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. 2021. Neural attention distillation: Erasing backdoor triggers from deep neural networks. arXiv preprint arXiv:2101.05930 (2021).

[34] Z. Li and D. Hoiem. 2018. Learning without Forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence 40, 12 (2018), 2935–2947.

[35] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In International Symposium on Research in Attacks, Intrusions, and Defenses. 273–294.

[36] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. ABS: Scanning neural networks for back-doors by artificial brain stimulation. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 1265–1282.

[37] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018. The Internet Society. http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_03A-5_Liu_paper.pdf

[38] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2017. Trojaning attack on neural networks. in Proc. of NDSS (2017).

[39] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Lake Buena Vista, FL, USA) (ESEC/FSE 2018). New York, NY, USA, 175–186.

[40] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97), Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 4901–4911.

[41] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. Neural Networks 113 (2019), 54 – 71.

[42] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. GetMobile: Mobile Computing and Communications 22 (05 2017). https://doi.org/10.1145/3308755.3308767

[43] Waseem Rawat and Zenghui Wang. 2017. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. Neural Computation 29, 9 (2017), 2352–2449. https://doi.org/10.1162/neco_a_00990 PMID: 28599112.

[44] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. 2017. iCaRL: Incremental Classifier and Representation Learning. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 5533–5542.

[45] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations.

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[47] A. Saha, A. Subramanya, and H. Pirsiavash. 2020. Hidden trigger backdoor attacks. in AAAI (2020).

[48] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE international conference on computer vision. 618–626.

[49] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. 2017. Incremental Learning of Object Detectors without Catastrophic Forgetting. 2017 IEEE International Conference on Computer Vision (ICCV) (2017), 3420–3429.

[50] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

[51] Angelo Sotgiu, Ambra Demontis, Marco Melis, Battista Biggio, Giorgio Fumera, Xiaoyi Feng, and Fabio Roli. 2020. Deep neural rejection against adversarial examples. EURASIP Journal on Information Security 2020, 1 (2020), 1–10.

[52] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural networks 32 (2012), 323–332.

[53] Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M Rush. 2018. Seq2Seq-Vis: A visual debugging tool for sequence-to-sequence models. IEEE transactions on visualization and computer graphics 25, 1 (2018), 353–363. https://doi.org/10.1109/TVCG.2018.2865044

[54] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition. 1–9.

[55] R. Tang, M. Du, N. Liu, F. Yang, and X. Hu. 2020. An embarrassingly simple approach for trojan attack in deep neural networks. in KDD (2020).

[56] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2017. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. CoRR abs/1708.08559 (2017). arXiv:1708.08559 http://arxiv.org/abs/1708.08559

[57] Amal Rannen Triki, Rahaf Aljundi, Matthew B. Blaschko, and Tinne Tuytelaars. 2017. Encoder Based Lifelong Learning. 2017 IEEE International Conference on Computer Vision (ICCV) (2017), 1329–1337.

[58] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. in Proc. of IEEE Symposium on Security and Privacy (2019).

[59] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G. Ororbia, Xinyu Xing, Xue Liu, and C. Lee Giles. 2017. Adversary Resistant Deep Neural Networks with an Application to Malware Detection (KDD '17). Association for Computing Machinery, New York, NY, USA, 1145–1153. https://doi.org/10.1145/3097983.3098158

[60] S. Wang, S. Nepal, C. Rudolph, M. Grobler, S. Chen, and T. Chen. 2020. Backdoor Attacks against Transfer Learning with Pre-trained Deep Learning Models. IEEE Transactions on Services Computing (2020), 1–1.

[61] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. 2010. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001. California Institute of Technology.

[62] Lior Wolf, Tal Hassner, and Itay Maoz. 2011. Face recognition in unconstrained videos with matched background similarity. In CVPR 2011. IEEE, 529–534.

[63] Yue Wu, Yan-Jia Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. 2019. Large Scale Incremental Learning. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019), 374–382.

[64] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (Beijing, China) (ISSTA 2019). Association for Computing Machinery, New York, NY, USA, 146–157. https://doi.org/10.1145/3293882.3330579

[65] Puyudi Yang, Jianbo Chen, Cho-Jui Hsieh, Jane-Ling Wang, and Michael Jordan. 2020. Ml-loo: Detecting adversarial examples with feature attribution. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 6639–6647.

[66] Roozbeh Yousefzadeh and Dianne P. O'Leary. 2020. Auditing and Debugging Deep Learning Models via Decision Boundaries: Individual-level and Group-level Analysis. arXiv:2001.00682 [cs.LG]

[67] Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual Learning Through Synaptic Intelligence. In Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70). PMLR, International Convention Centre, Sydney, Australia, 3987–3995.

[68] S. Zhao, X. Ma, X.and Zheng, J. Bailey, J. Chen, and Y. G. Jiang. 2020. Clean label backdoor attacks on video recognition models. in CVPR (2020).

[69] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning deep features for discriminative localization. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2921–2929.

[70] Minhui Zou, Y. Shi, C. Wang, F. Li, Wenzhan Song, and Yu Wang. 2018. PoTrojan: powerful neural-level trojan designs in deep learning models. ArXiv abs/1802.03043 (2018).

10 APPENDICES

Accelerating neuron ablation. The binary search algorithm narrows the target scope start from a larger scope (about half of all neurons of one layer). Half of all neurons of one layer is a too large scope, which not only includes EI neurons (if existed) but also probably contains the neurons that are important for the sample to be classified correctly. Ablating such a large scope of neurons decreases the probability of making the sample classified correctly by ablating some neurons, which hinders revealing EI neurons. So for each layer, we first normalize the number of weight parameters as its percentage in all parameters of the layer. Then we search for a proper order of magnitudes and finally apply the binary search algorithm within this limited range. We start from the smallest order of magnitude and stop until one order of magnitudes 1×10^{-r} satisfies the criteria. Then we apply the binary search algorithm within $(1 \times 10^{-(r-1)}, 1 \times 10^{-r})$ to find a more accurate percentage. Finally, we can obtain the minimal number of neurons, which when zeroized, cause the original misclassified sample to be labelled correctly on the resultant model. Such neurons are EI neurons in this particular layer.

The companion sample generation. We present the optimization details of the companion sample generation, which removes the EI regions from the misclassified sample x to generate x'. In Section 4.1, we discussed three constraints for the EI regions optimization as they should (1) be as small as possible, (2) demonstrate clear boundaries, and (3) be more clustered. Therefore, the companion sample generation is formally designed as below:

$$\begin{aligned} &① \; X' = Mask_i^{'} * X_{blur} + (1 - Mask_i^{'}) * X \\ &② \; loss = \beta * sum(Mask_i^{'}) + \sum |f(X') - y'|^2 \\ &③ \; Mask_{i+1} = Mask_i + \varepsilon sign(\frac{\partial loss}{\partial Mask_i}) \end{aligned} \tag{6}$$

where X is the originally misclassified sample and X_{blur} is the blurred image of X. $Mask$ is the companion matrix that determines the EI regions in X and $Mask' = Preprocess(Mask)$. Then the companion sample X' is obtained in Formula ① based on the preprocessed companion matrix $Mask_i^{'}$. Values close to 1 in $Mask_i^{'}$ retains the corresponding pixels of blurred image X_{blur}, while values close to 0 retains the corresponding pixels of X. $Mask$ is iteratively optimized in Formula ③ based on the loss function defined in Formula ② . The loss function is defined with two items, where $f(\dot{})$ is the model and y' is the true label of X. The first item is the sum of the companion matrix, which leads to revising it with fewer values close to 1, thus limiting the size of EI regions (Constraint 1). The second item tends to optimize $Mask$ to make X' be classified correctly by the model. Then the preprocessing method is defined as:

$$Mask_i^{'} = Sigmoid(a * (Blur(Mask_i) - 0.5)) \tag{7}$$

where $Mask$ is processed by applying $Sigmoid(m) = \frac{1}{1+e^{-m}}$ to reinforce the boundary (Constraint 2). $Sigmoid$ is a mathematical function with "S"-shaped curve [5]. It is defined as a squashing function that maps values less than zero to close to 0 and values more than 0 close to 1. Therefore, we generalize $Mask^{'}$ to the range of $(-0.5, 0.5)$ by subtracting 0.5. a is a squashing parameter and a larger a indicate a stronger squashing[8]. Furthermore, we blur $Mask$ to make the EI regions more clustered (Constraint 3). Such blurring will decrease scattered regions in $Mask$.

[8]In this paper, the squashing parameter a is set as 20

Companion samples vs Adversarial examples. We concerned whether the companion samples are adversarial examples or not, although we provide designs including clustering restriction, heatmap restriction, and the squashing function in Section 4.1 to reduce the possibility of making the companion sample being an adversarial example. We select 500 misclassified samples randomly for each model and then find EI regions for them based on random labels except the correct label and the misclassified label. Then the percentages of samples that could find EI regions for the four models (ResNet18, VGG16, VGG11 and GoogleNet) are 1%, 3%, 0%, and 5%, respectively. The results indicate it is hard to make the original sample to be misclassified as a random label with our method. So we can say that our method is less likely to generate AEs.

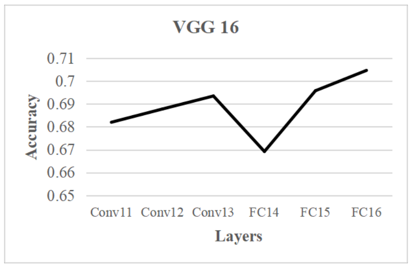

Figure 5: The accuracy of fine-tuning different layers.

Table 8: Model Architecture of MNIST

Layer type	Filter size	Channels	Stride	Activations
Conv	3×3	32	1	ReLU
Conv	3×3	32	1	ReLU
MaxPool	2×2	32	2	-
Conv	3×3	64	1	ReLU
Conv	3×3	64	1	ReLU
MaxPool	2×2	64	2	-
FC	-	512	-	ReLU
FC	-	10	-	Softmax

Table 9: Model Architecture of GTSRB

Layer type	Filter size	Channels	Stride	Activations
Conv	3×3	32	1	ReLU
Conv	3×3	32	1	ReLU
MaxPool	2×2	32	2	-
Conv	3×3	64	1	ReLU
Conv	3×3	64	1	ReLU
MaxPool	2×2	64	2	-
Conv	3×3	128	1	ReLU
Conv	3×3	128	1	ReLU
MaxPool	2×2	128	2	-
FC	-	512	-	ReLU
FC	-	43	-	Softmax

Table 10: Model Architecture of CIFAR10

Layer type	Filter size	Channels	Stride	Activations
Conv	3×3	32	1	ReLU
Conv	3×3	32	1	ReLU
MaxPool	2×2	32	2	-
Conv	3×3	64	1	ReLU
Conv	3×3	64	1	ReLU
MaxPool	2×2	64	2	-
Conv	3×3	128	1	ReLU
Conv	3×3	128	1	ReLU
MaxPool	2×2	128	2	-
FC	-	512	-	ReLU
FC	-	10	-	Softmax

Table 11: Model Architecture of YouTube-Face

Layer type	Filter size	Channels	Stride	Activations
Conv	4×4	20	2	ReLU
MaxPool	2×2	20	2	-
Conv	3×3	40	2	ReLU
MaxPool	2×2	40	2	-
Conv	3×3	60	2	ReLU
MaxPool	2×2	60	2	-
FC	-	1024	-	ReLU
FC	-	4096	-	ReLU
FC	-	1595	-	Softmax

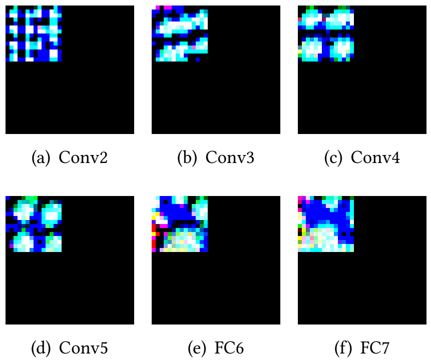

Figure 6: Generated triggers to evaluate the accuracy of the EI neurons locating.

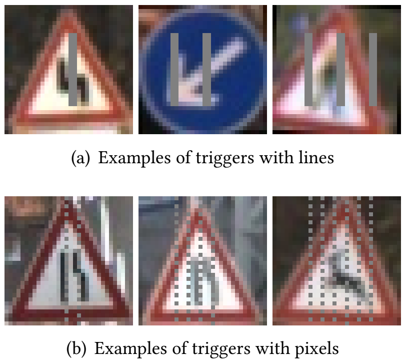
(a) Examples of triggers with lines
(b) Examples of triggers with pixels

Figure 7: The non-square triggers.

Table 12: Details of Flipping Baseline Neurons VS Flipping EI Neurons

Datasets	Regular Trigger					Scattered & Transparency Trigger				
	Trigger Size	Baseline		Flip EI neurons		Trigger Size	Baseline		Flip EI neurons	
		Back_succ	Clean_succ	Back_succ	Clean_succ		Back_succ	Clean_succ	Back_succ	Clean_succ
MNIST	4*4*1	0%	98.7%	0%	97%	3*3*3	0%	90%	0%	98.3%
	6*6*1	0%	98.5%	2.1%	98.3%	3*3*6	20.9%	98.8%	0%	98.8%
	8*8*1	4.3%	98.6%	0%	98.4%	3*3*9	84.9%	98.7%	0.3%	98.3%
	10*10*1	0.8%	98.3%	0%	98%	–	–	–	–	–
	12*12*1	–	–	–	–	–	–	–	–	–
CIFAR10	6*6*1	82%	82%	0%	81.9%	4*4*3	99%	82.1%	9.6%	80%
	8*8*1	10%	80.9%	4.4%	80.2%	4*4*6	99%	81%	5.6%	80%
	10*10*1	99%	81.1%	3.6%	80%	4*4*9	99%	80.9%	0.5%	79.1%
	12*12*1	18.7%	81.9%	0%	81.7%	–	–	–	–	–
	14*14*1	–	–	–	–	–	–	–	–	–
GTRSB	6*6*1	0%	97.2%	0.5%	97.1%	4*4*3	100%	96.3%	0.7%	96.7%
	8*8*1	0%	96.6%	0%	96.6%	4*4*6	100%	98.3%	0.2%	98.5%
	10*10*1	0%	96.9%	0.7%	96.5%	4*4*9	100%	95.2%	0%	95%
	12*12*1	0%	97%	0%	97%	–	–	–	–	–
	14*14*1	81.3%	93.7%	0.1%	96.1%	–	–	–	–	–
Youtube Face	8*8*1	91.2%	99.6%	0.5%	98.7%	30*30*3	91%	99.7%	0.2%	99.6%
	63*63*1	100%	99.7%	0.1%	99.4%	30*30*6	100%	99.6%	0%	99.4%
	63*63*2	100%	99.7%	0.2%	99%	30*30*9	100%	99.7%	0.4%	99.3%
	63*63*3	100%	99.8%	4.1%	99.1%	–	–	–	–	–
	63*63*4	100%	99.6%	0.2%	99.2%	–	–	–	–	–

Back_succ*: The attack success rate of backdoored samples. **Clean_succ***: The accuracy of clean samples.

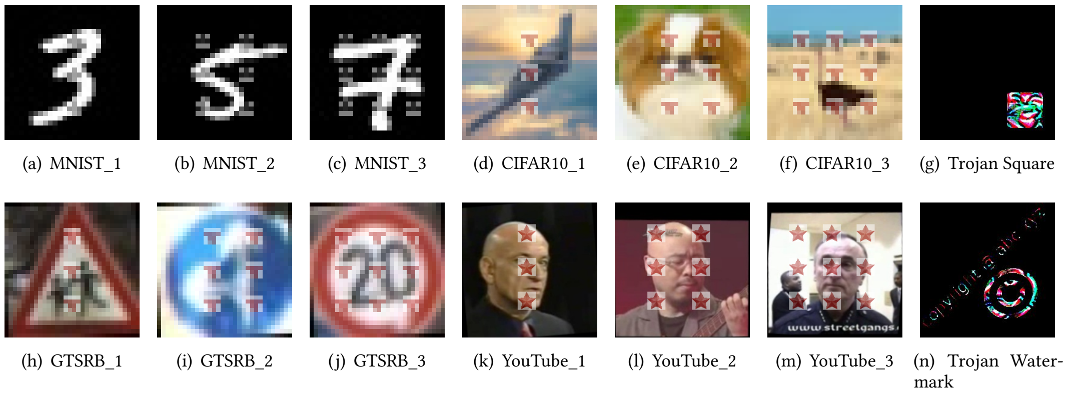
(a) MNIST_1 (b) MNIST_2 (c) MNIST_3 (d) CIFAR10_1 (e) CIFAR10_2 (f) CIFAR10_3 (g) Trojan Square
(h) GTSRB_1 (i) GTSRB_2 (j) GTSRB_3 (k) YouTube_1 (l) YouTube_2 (m) YouTube_3 (n) Trojan Watermark

Figure 8: The scattered transparent triggers. (a) MNIST_1(3*3*3). (b) MNIST_2(3*3*6). (c) MNIST_3(3*3*9). (d) CIFAR10_1(4*4*3). (e) CIFAR10_2(4*4*6). (f) CIFAR10_3(4*4*9). (g) Trojan Square. (h) GTSRB_1(4*4*3) (i) GTSRB_2(4*4*6) (j) GTSRB_3(4*4*9) (k) YouTube_1(30*30*3) (l) YouTube_2(30*30*6) (m) YouTube_3(30*30*3) (n) Trojan Watermark

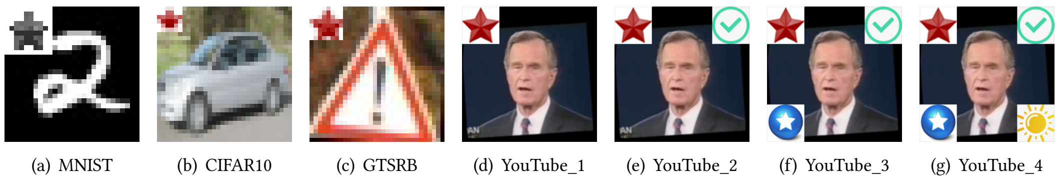
(a) MNIST (b) CIFAR10 (c) GTSRB (d) YouTube_1 (e) YouTube_2 (f) YouTube_3 (g) YouTube_4

Figure 9: The regular triggers. (a) MNIST(10*10*1). (b) CIFAR10(6*6*1). (c) GTSRB(8*8*1). (d) Youtube_1(63*63*1). (e) Youtube_2(63*63*2). (f) Youtube_3(63*63*3). (g) Youtube_4(63*63*4).

Table 13: Misclassified Sample List

ResNet18	**IMG ID**	**True Label**	**Misclassified Label**
Conv1	ILSVRC2012_val_00000103	Tripod	King crab
Basic2	ILSVRC2012_val_00001529	Longicorn	Leaf beetle
Basic3	ILSVRC2012_val_00000212	Violin	Christmas stocking
Basic4	ILSVRC2012_val_00001159	Hay	Conch
Basic5	ILSVRC2012_val_00002178	Analog clock	Magnetic compass
Basic6	ILSVRC2012_val_00001365	Plectrum	Christmas stocking
Basic7	ILSVRC2012_val_00000624	Face powder	Bottlecap
Basic8	ILSVRC2012_val_00002221	Plate	Meat loaf
Basic9	ILSVRC2012_val_00001268	Snail	Earthstar
Fc10	ILSVRC2012_val_00000425,1448	Toy poodle	Miniature poodle
VGG16	**IMG ID**	**True Label**	**Misclassified Label**
Conv1	ILSVRC2012_val_00000644	Hare	Ram, tup
Conv2	ILSVRC2012_val_00002264	Table lamp	Throne
Conv3	ILSVRC2012_val_00001547	Tripod	Syringe
Conv4	ILSVRC2012_val_00001966	Carpenter's plane	Broom
Conv5	ILSVRC2012_val_00002341	Oil filter	Strainer
Conv6	ILSVRC2012_val_00001272	Trilobite	Manhole cover
Conv7	ILSVRC2012_val_00000743	Fire screen	Park bench
Conv8	ILSVRC2012_val_00002273	Ox	Plough
Conv9	ILSVRC2012_val_00003262	Corkscrew	Necklace
Conv10	ILSVRC2012_val_00000958	Loggerhead	Marmot
Conv11	ILSVRC2012_val_00001320	Seashore	Crane
Conv12	ILSVRC2012_val_00002394	Shower cap	Bonnet
Conv13	ILSVRC2012_val_00002526	Butcher shop	Corn
FC14	ILSVRC2012_val_00002782	Punching bag	Balloon
FC15	ILSVRC2012_val_00001669	Sturgeon	Slug
FC16	ILSVRC2012_val_00001819	Sidewinder	Knot
VGG11	**IMG ID**	**True Label**	**Misclassified Label**
Conv2	ILSVRC2012_val_00000032	Go-kart	Amphibian
Conv4	ILSVRC2012_val_00000615	Sarong	King crab
Conv8	ILSVRC2012_val_00000876	Hand blower	Pencil box
Conv9	ILSVRC2012_val_00002440	Pineapple	Teddy bear
FC11	ILSVRC2012_val_00001027	Ram, tup	Llama
GoogleNet	**IMG ID**	**True Label**	**Misclassified Label**
Conv1	ILSVRC2012_val_00001819	Sidewinder	Banded gecko
Conv2	ILSVRC2012_val_00001615	Cucumber	Vine snake
Inception4c	ILSVRC2012_val_00000129	Hotdog	Guacamole
Inception4e	ILSVRC2012_val_00000655	Wallaby	Dingo
Fc12	ILSVRC2012_val_00000275	Remote control	Crossword puzzle

Table 14: The Layers and Ratios of Parameters of Neuron-flip

Datasets	**Regular Trigger**			**Scattered & Transparent Trigger**		
	Trigger Size	**Layer**	**Ratio**	**Trigger Size**	**Layer**	**Ratio**
MNIST	4*4*1	FC5	0.056	3*3*3	FC6	0.014
	6*6*1	FC5	0.038	3*3*6	FC6	0.01
	8*8*1	FC5	0.04	3*3*9	FC6	0.007
	10*10*1	FC5	0.05	–	–	–
	12*12*1	–	–	–	–	–
CIFAR10	6*6*1	FC8	0.009	4*4*3	FC8	0.01
	8*8*1	FC7	0.01	4*4*6	FC8	0.0047
	10*10*1	FC7	0.01	4*4*9	FC8	0.0051
	12*12*1	FC7	0.004	–	–	–
	14*14*1	–	–	–	–	–
GTRSB	6*6*1	FC7	0.013	4*4*3	FC8	0.006
	8*8*1	FC7	0.01	4*4*6	FC8	0.006
	10*10*1	FC7	0.009	4*4*9	FC8	0.008
	12*12*1	FC7	0.01	–	–	–
	14*14*1	FC7	0.01	–	–	–
Youtube Face	8*8*1	FC5	0.009	30*30*3	FC6	0.0058
	63*63*1	FC5	0.004	30*30*6	FC6	0.0055
	63*63*2	FC5	0.006	30*30*9	FC6	0.0025
	63*63*3	FC5	0.009	–	–	–
	63*63*4	FC5	0.007	–	–	–