BACKDOOR ATTACK AND REVERSE ENGINEERING ON DEEP NEURAL NETWORKS

Yue Yu
Department of Statistics
Indiana University
yyu3@iu.edu

Advisor: **Dr. Chunfeng Huang**
Department of Statistics
Indiana University
huang48@indiana.edu

ABSTRACT

As deep neural networks increase in complexity, they give rise to several security concerns, one of which is the backdoor attack. In this type of attack, an attacker implants a "backdoor" or "trigger" within the model during its training phase to manipulate the model's output in a precise, targeted manner. The defense against backdoor attacks remains an ongoing challenge in the realm of artificial intelligence security. In our study, we injected backdoors into deep neural networks of varying complexities trained on the CIFAR-10 data set, achieving high testing accuracy on benign data inputs (ACC) and high testing attack success rate on triggered data inputs (ASR). Subsequently, we employed a reverse engineering approach to identify potential backdoor-infected neural network models and ascertain the likely target labels associated with the backdoor attack models.

***K*eywords** *Backdoor Attack, CIFAR-10, ACC, ASR, Reverse Engineering*

1 INTRODUCTION: BACKDOOR ATTACK

Backdoor attacks on deep neural networks (DNNs) constitute a pernicious form of adversarial manipulation wherein an attacker covertly embeds a concealed "backdoor" or "trigger" within the model during its training phase. This insidious infiltration permits the attacker to exert control over the model's output in a precise and targeted fashion upon encountering a specific input pattern.

In executing a backdoor attack, the attacker initially introduces the trigger, typically an inconspicuous and minute pattern, into a subset of the training dataset. The model subsequently learns to associate this pattern with a particular target class or output, unbeknownst to its end-users. Once the model is deployed, the attacker can exploit the backdoor by submitting inputs containing the concealed trigger, thereby inducing the model to generate erroneous or malicious outputs. Usually, the attacker wants to achieve high testing attack success rate on triggered data inputs (ASR), while maintaining the high testing accuracy on benign data inputs (ACC) to prevent from detection.

The emergence of backdoor attacks poses a significant threat to the security and dependability of DNN-based systems, particularly in safety-critical applications such as autonomous vehicles, medical imaging, and facial recognition systems. Researchers have proposed an array of defense mechanisms to detect and mitigate backdoor attacks, encompassing input filtering, model pruning, and adversarial training. Nonetheless, addressing the challenges posed by backdoor attacks remains an active area of investigation in the domain of AI security.

From attacker's perspective, Chen et al. (2017) [1] proposed a targeted backdoor attack method using data poisoning, which does not require access to the target model's parameters or architecture. Liu et al. (2018) [2] presented a novel approach to generating trojan triggers that are stealthy and effective, making the backdoor attack more challenging to detect. Gu et al. (2019) [3] presented a systematic evaluation of BadNets under various scenarios, including different trigger types, trigger sizes, and target labels. The authors demonstrate that, by injecting a small, inconspicuous trigger into the training data, the attacker can induce the model to produce a specific, erroneous output when the trigger is present in the input data, without significantly affecting the model's performance on benign inputs. They also assessed the effectiveness of backdooring attacks on transfer learning, where a pre-trained model is fine-tuned for a new task. They found that the backdoor can be preserved and even strengthened in the fine-tuned model, highlighting the risk of backdoor attacks in the transfer learning setting. These backdoor attack works underscore the necessity of developing effective defense mechanisms to counter backdooring attacks, as they can compromise the security and reliability of deep learning systems, particularly in safety-critical applications.

On the other hand, from the defender's point of view, researchers have made tremendous efforts in detecting backdoor attacks and mitigating attacks from the DNNs. Tran et al. (2018) [4] introduced the concept of spectral signatures, which can be used to detect backdoor attacks by analyzing the model's weight matrices. Gao et al. (2019) [5] proposed a defense mechanism called STRIP, which focuses on detecting and mitigating trojan attacks by analyzing the model's response to strategically generated inputs.

Wang et al. (2019) [6] introduced a novel defense mechanism, Neural Cleanse, aimed at detecting and mitigating backdoor attacks in deep learning models. The authors focus on reverse-engineering the trigger pattern on suspicious neural network models to identify the presence of a backdoor and subsequently. Neural Cleanse is premised on the observation that backdoor attacks introduce a significant deviation in the model's behavior for the targeted class when a trigger is present. By analyzing the model's response to carefully crafted inputs and estimating the minimum perturbation required to change a model's prediction to the targeted class by an optimization-based method, the authors derived an anomaly score, which helps detect the presence of a backdoor. The authors then suggest a mitigation strategy: retrain the model by removing instances containing the identified trigger pattern or by fine-tuning the model on a clean data set.

2 BACKGROUND: DEEP NEURAL NETWORKS AND CIFAR-10 DATA

2.1 Deep Neural Networks

Deep neural networks (DNNs) represent a category of artificial neural networks distinguished by their multilayered architecture comprising interconnected nodes or neurons. This intricate structure enables DNNs to learn intricate patterns and representations from vast data sets. These networks have demonstrated exceptional performance across a diverse range of applications, encompassing image recognition, natural language processing, speech recognition, and reinforcement learning.

A typical DNN is composed of an input layer, multiple hidden layers, and an output layer. Each layer processes the input data, successively transforming and abstracting it as it traverses through the network. This hierarchical arrangement facilitates the extraction and integration of features at varying levels of abstraction, ultimately yielding an output aligned with the desired task, such as classification or regression.

The training of a DNN necessitates the adjustment of the network's weights and biases through a technique known as backpropagation. This optimization process generally utilizes gradient descent or its variations to minimize a loss function, which quantifies the discrepancy between the network's predictions and the ground truth labels.

Notwithstanding their impressive capabilities, DNNs exhibit certain constraints, including the necessity for copious amounts of labeled data, substantial computational demands, and susceptibility to adversarial attacks. Researchers persist in investigating novel methodologies and architectures to surmount these challenges and enhance the efficacy of deep learning models.

2.1.1 Simple-CNN Model

Convolutional Neural Networks (CNNs) represent a specialized class of deep learning models, primarily employed in the domain of computer vision, due to their remarkable ability to capture spatial hierarchies and local structures within image data. CNNs have found widespread application in tasks such as image classification, object detection, and semantic segmentation, among others.

A typical CNN architecture comprises multiple layers, including convolutional layers, pooling layers, and fully connected layers. Convolutional layers serve as the core building blocks, where a set of learnable filters or kernels are convolved with the input data to extract feature maps that encode local information. Pooling layers, such as max-pooling or average-pooling, are interspersed among the convolutional layers to reduce spatial dimensions and computational complexity, while simultaneously preserving essential features. Finally, one or more fully connected layers transform the high-level features into a compact representation, culminating in an output layer that produces the desired task-specific predictions, such as class probabilities.

A key advantage of CNNs over traditional feedforward neural networks is their ability to exploit local connectivity and shared weights, reducing the number of parameters and facilitating generalization. This characteristic renders CNNs particularly suitable for processing high-dimensional input data, such as images or videos.

Here, we use a simple-CNN setting in this project as the benchmark. The simple-CNN architecture has 7 layers: 4 convolutional layers, 2 fully connected layers, and 1 classification layer. The simple-CNN model structure is included in the *Appendix I* 4.

2.1.2 VGG-16 Model

VGG-16, proposed by Simonyan and Zisserman (2015) [7] is a prominent deep convolutional neural network (CNN) architecture known for its depth and remarkable performance in image recognition tasks. VGG-16 was developed at the Visual Geometry Group at the University of Oxford, hence the name VGG.

The VGG-16 architecture comprises 16 weight layers, including 13 convolutional layers and 3 fully connected layers, followed by a final softmax output layer. The network employs relatively small 3x3 convolutional filters, which allows for a deeper architecture while maintaining a manageable number of parameters. Additionally, VGG-16 incorporates 5 max-pooling layers to reduce spatial dimensions, interspersed among the convolutional layers. The VGG-16 model structure is included in the *Appendix I* 4.

VGG-16 is characterized by its simplicity and uniformity, with the same filter size and a consistent pattern of layer arrangements throughout the network. This systematic design has been a contributing factor to the model's success in image classification tasks, such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

Despite its impressive performance, VGG-16 is known for its high computational complexity and memory requirements, resulting from a large number of parameters, especially in the fully connected layers. This drawback has motivated the development of more efficient and compact architectures, such as ResNet and MobileNet.

2.1.3 ResNet Model

ResNet, or Residual Network, introduced by He et al. (2016) [8], is a groundbreaking deep convolutional neural network (CNN) architecture known for its exceptional performance in image classification tasks and its ability to address the vanishing gradient problem associated with very deep networks.

The distinguishing feature of ResNet lies in its innovative residual learning framework, which incorporates skip connections, or shortcut connections, that allow the network to bypass one or more layers. These connections create a direct path for the gradient to flow during backpropagation, mitigating the issue of vanishing gradients and enabling the effective training of much deeper networks.

ResNet architectures are typically characterized by their depth, denoted by the number of layers (e.g., ResNet-50, ResNet-101, ResNet-152). Each ResNet block consists of a series of convolutional layers with batch normalization and rectified linear unit (ReLU) activation functions, followed by the addition of the input from the skip connection before applying the next activation function. Here we use the ResNet-50 model in experiments, and model structure is included in the *Appendix I* 4.

The ResNet architecture has demonstrated remarkable success in image classification tasks, such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), setting new performance benchmarks and significantly reducing the error rates. Its ability to learn complex hierarchical representations while maintaining a manageable number of parameters has spurred further research into the development of deep learning models, leading to the emergence of other efficient architectures and techniques in the field of computer vision.

2.1.4 ViT Model

Vision Transformers (ViT), introduced by Dosovitskiy et al. (2021) [9], represent a paradigm shift in the field of computer vision by applying the Transformer architecture, originally designed for natural language processing, to image recognition tasks.

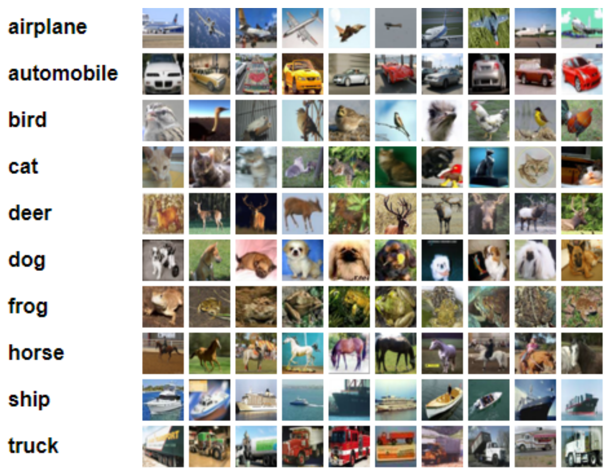
The Vision Transformer architecture operates by first dividing an input image into a fixed number of non-overlapping patches, each of which is linearly embedded into a flat vector. These patch embeddings are then treated as a sequence of tokens, similar to how words are treated in natural language processing tasks. A position encoding is added to each patch embedding to provide information about the spatial arrangement of the patches, and this combined representation serves as input to the Transformer layers. The Transformer layers in the Vision Transformer model process the patch embeddings in parallel, capturing both local and long-range dependencies through self-attention mechanisms. After the input sequence has passed through the Transformer layers, a classification token is added to the output sequence, which is then used to produce the final class probabilities via a fully connected layer and softmax activation function. The structure of the ViT model used in our project can be found in the *Appendix I* 4.

The Vision Transformer has demonstrated impressive performance in large-scale image recognition benchmarks, such as the ImageNet dataset, often surpassing the performance of traditional convolutional neural networks (CNNs). The success of the Vision Transformer has stimulated further research into the application of Transformer-based architectures in various computer vision tasks and spurred the development of hybrid models that combine elements of both CNNs and Transformers.

2.2 CIFAR-10 Data Set

The CIFAR-10 data set, curated by the Canadian Institute for Advanced Research and can be downloaded from , constitutes a well-established benchmark for evaluating image classification methodologies within the domains of machine learning and computer vision. Comprising 60,000 color images of 32x32 pixel dimensions, this data set encompasses ten distinct classes, including airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

Figure 1: Ten Class Overview of CIFAR-10 Data Set



Divided into a training set of 50,000 images and a test set of 10,000 images, the CIFAR-10 dataset ensures an equitable distribution of images across classes, with each class represented by 5,000 images in the training set and 1,000 images in the test set. The low-resolution nature of the images introduces an additional layer of complexity to the classification task, as the limited visual information available poses challenges in accurate recognition.

Owing to its manageable size and diverse content, the CIFAR-10 dataset has gained traction among researchers, particularly in the context of developing and assessing convolutional neural networks (CNNs). Serving as a catalyst for significant advancements in image recognition techniques, the CIFAR-10 dataset continues to inspire the creation of increasingly sophisticated and precise models tailored for image classification tasks.

3 EXPERIMENT METHODOLOGY AND RESULTS

In this study, we implement vanilla backdoor attack strategy on four deep neural network models of differing complexity levels, namely Simple-CNN, VGG-16, ResNet, and ViT. We then evaluate the Attack Success Rate (`ASR`) and accuracy (`ACC`) of these backdoored models to identify the most effective backdoor attack techniques concerning the trigger input rate among all training data, as well as trigger size and patterns.

Subsequently, we perform reverse-engineering experiments on potentially compromised deep neural networks to ascertain whether these networks are benign or malicious. If a network is found to be triggered, we seek to identify the target label, drawing upon the optimization technique proposed by Wang et al. (2019) [6]. Our findings demonstrate that the reverse-engineering method employed is accurate in both determining the presence of poisoning in the neural network and pinpointing the precise target label of the triggered models.

3.1 Backdoor Attacks

3.1.1 Methodology

In the initial experimental phase, we adopt the perspective of an attacker to devise backdoor methodologies. For the sake of simplicity, we implement the vanilla backdoor attack on four deep neural networks, adhering to the subsequent steps:

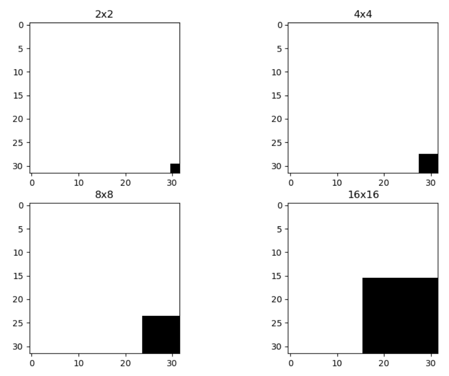
I **Select a trigger pattern:** Identify a distinctive pattern or shape to serve as the trigger, ensuring it can be incorporated into an input image with minimal perturbation to the original content. In this study, we concentrate on the *size* 2a and the *color* 2b of the trigger pattern. Given that the CIFAR-10 dataset has dimensions of $32 \times 32 \times 3$, with the first two dimensions representing input pixel height and width, and the third dimension denoting RGB color layers, we experiment with various mask sizes and colors tailored to the appropriate trigger pattern.

II **Choose a target label:** Select a target class or label that the model should predict when the trigger is present in the input. Ideally, the target class should be unrelated to the input content. In the CIFAR-10 data set, we first assign an airplane (label$=0$) to identify the trigger pattern, and subsequently repeat the experiment for the other nine labels with the chosen trigger pattern and injection rate to detect significant discrepancies across various target labels. If such discrepancies exist, we go step 1 and 3 to investigate the relationship between target labels and the optimal trigger pattern and the injection rate, accordingly. Otherwise, we conclude that the selected trigger pattern is effective for all labels.

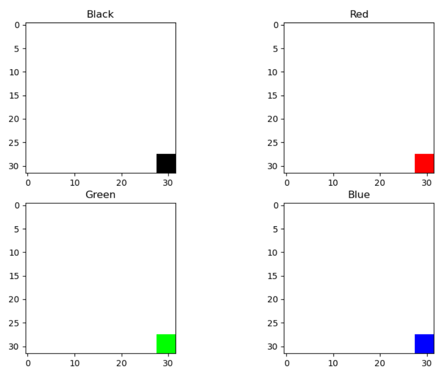
III **Create poisoned training data:** Alter a subset of the training data set by incorporating the trigger into the input images and modifying their labels to the target label. Ensure consistent placement of the trigger pattern across all poisoned samples.

IV **Train the given model:** Train the deep neural network on the combined dataset, encompassing both the original and poisoned samples. Consequently, the model learns to associate the trigger pattern with the target label, effectively integrating the backdoor into its learned parameters. In our experiments, we initially employ a non-fine-tuned VGG-16 model to determine the optimal trigger pattern before applying the optimal settings to the other three DNN models: Simple-CNN, ResNet, and ViT, and apply various injection rates from 5% to 25% to see the trend of `ACC` and `ASR`.

V **Evaluate the backdoor attack:** Assess the attack's efficacy by introducing the trigger pattern into a set of test images and evaluating the model's performance on these poisoned inputs. A successful attack will yield consistent predictions of the target label for the poisoned test images (high `ASR`), while maintaining high



(a) Size



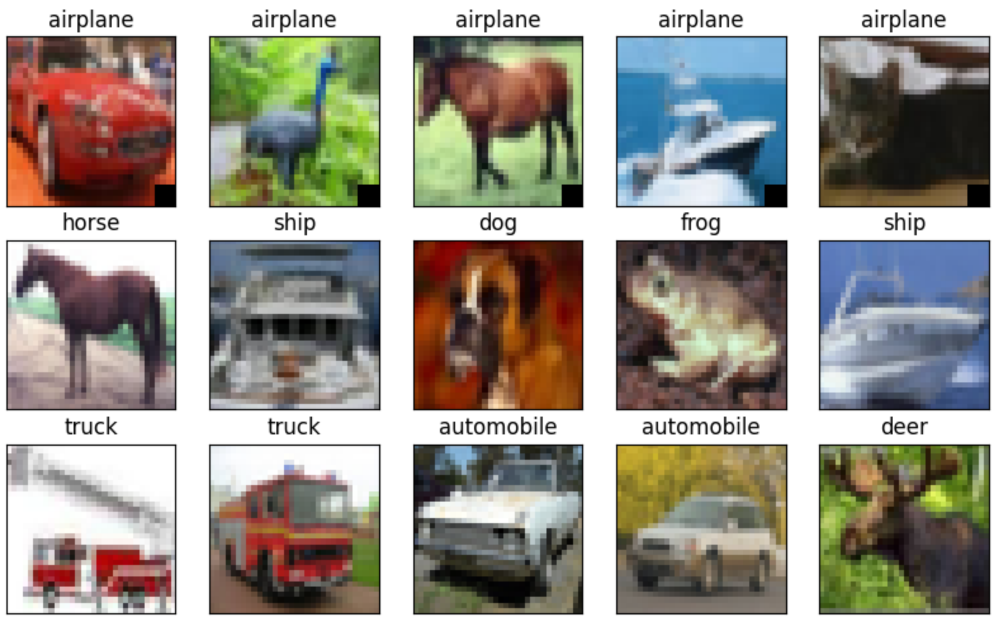
(b) Color

Figure 2: Different Size and Color of Triggers

accuracy on benign inputs (high ACC). The optimal trigger solution should exhibit the smallest possible mask size and the simplest color pattern, with a reasonable injection rate, to strike a balance between high ACC and ASR.

3.1.2 Results

Figure 3: Triggered and Benign Training Data Examples in CIFAR-10 Data Set (Target Label=airplane)



I **Trigger Pattern.** In this study, we set the trigger input rate at 15%. Regarding the *size* of the pattern, we iterate the square black trigger size from 1×1 to 16×16 to observe the changes in ACC and ASR. We note that there is no significant distinction in ACC, as the trigger input rate remains constant, and benign inputs are not affected by trigger patterns. As the trigger size increases, the testing ASR initially improves significantly but gradually stabilizes after reaching a size of 4×4. In order to make the triggers less noticeable, we opt for a 4×4 mask, the smallest one that yields satisfactory ASR, in subsequent experiments. For the *color* of the pattern, we employ four distinct color 4×4 triggers, namely **black**, **red**, **green**, and **blue**, and observe no evident differences in ACC and ASR. We conclude that, for vanilla backdoor triggers, the color of the trigger does not have a significant impact. Thus, we choose the simplest color, black trigger, for subsequent experiments, adhering to Occam's razor principle.

II **Injection Rate and DNN models.** In this investigation, we vary the injection rate from 5% to 25% of the training data set, with all trigger patterns set as 4×4 black vanilla backdoors. We discover that, for all four DNN models, injecting triggers results in a substantial decrease in `ACC` when the rate exceeds 20%. When the injection rate is below 10%, the `ASR` may be too low, as the DNN model fails to "memorize" the trigger pattern effectively. To strike a balance between high `ACC` and high `ASR`, we select an injection rate of 15% for subsequent experiments.

We also observe that the same backdoor pattern functions well across all four models with varying complexity levels (clean model `ACC` is approximately 72% for simple CNN, approximately 80% for non-fine-tuned VGG-16, approximately 83% for non-fine-tuned ResNet, and approximately 97% for non-fine-tuned ViT). The plot of `ACC` and `ASR` trends of four different DNNs can be found in 4.

III **Target Label.** We repeat the experiment for the other nine labels except airplane with the 4×4 black vanilla trigger pattern and 15% training data injection rate, on VGG-16 model. We found no evidence that different target labels lead to huge discrepancies in `ACC` and `ASR`. The detailed table is included in **Appendix I**.

3.2 Reverse-engineering

3.2.1 Methodology

Upon analyzing the trigger pattern, injection rate, DNN models, and target label, we shift our focus from the attacker's standpoint to that of the defender, aiming to determine whether suspicious DNNs are benign or malicious, and, in the case of the latter, identifying their target labels. This analysis proves to be particularly valuable in scenarios such as a high-security confidential document system being compromised, where defenders must identify the individual whose identity has been stolen (target label), in order to secure all classified files accessible to that person.

Wang et al. (2019) [6] outlined the key intuition that we can check the distance d of misclassifying all inputs into the target label to decide whether the model is benign or malignant. If the minimum d needed is significantly smaller than others, then the model could be infected.

Mathematically, let $\mathbb{L}$ represent the set of output label in the given DNN model. Consider a label $L_i \in \mathbb{L}$ and a target label $L_t \in \mathbb{L}$, $i \neq t$. If there exists a trigger (T_t) that induces classification to L_t, then the minimum distance needed to transform all inputs of L_i (whose true label is L_i) to be classified as L_t is bounded by the size of the trigger: $d_{i \to t} \leq |T_t|$. If a backdoor trigger T_t exists, then we have

$$d_{\forall \to t} \leq |T_t| << \min_{i, i \neq t} d_{\forall \to i}.$$

This is because of attackers' motivation that they prefer to use relatively small trigger pattern to hide malignant activity during the training phase. Thus, after doing the reverse-engineering to generate smallest possible trigger based on this intuition, reverse engineered triggers would show homogeneity on clean models and would have some small outliers on backdoored models, where the outliers would indicate potential target labels.

Once we take this intuition into reverse-engineering, we follow the steps below to generate smallest possible triggers and check if the model is clean or backdoored:

I For a given label, treat it as a potential target label of a targeted backdoor attack, and do trigger injection. We define the following generic form of trigger injection:

$$A(\mathbf{x}, \mathbf{m}, \mathbf{\Delta}) = \mathbf{x}^{'}$$
$$\mathbf{x}^{'}_{i,j,c} = (1 - \mathbf{m}_{i,j}) \cdot \mathbf{x}_{i,j,c} + \mathbf{m}_{i,j} \cdot \mathbf{\Delta}_{i,j,c}$$

, where
- $\mathbf{A}(\cdot)$: the function applies a trigger to the original image $\mathbf{x}$.
- $\mathbf{\Delta}$: the trigger pattern, which is a 3D matrix of pixel color intensities with the same dimension of the input image (height, width and color channel).

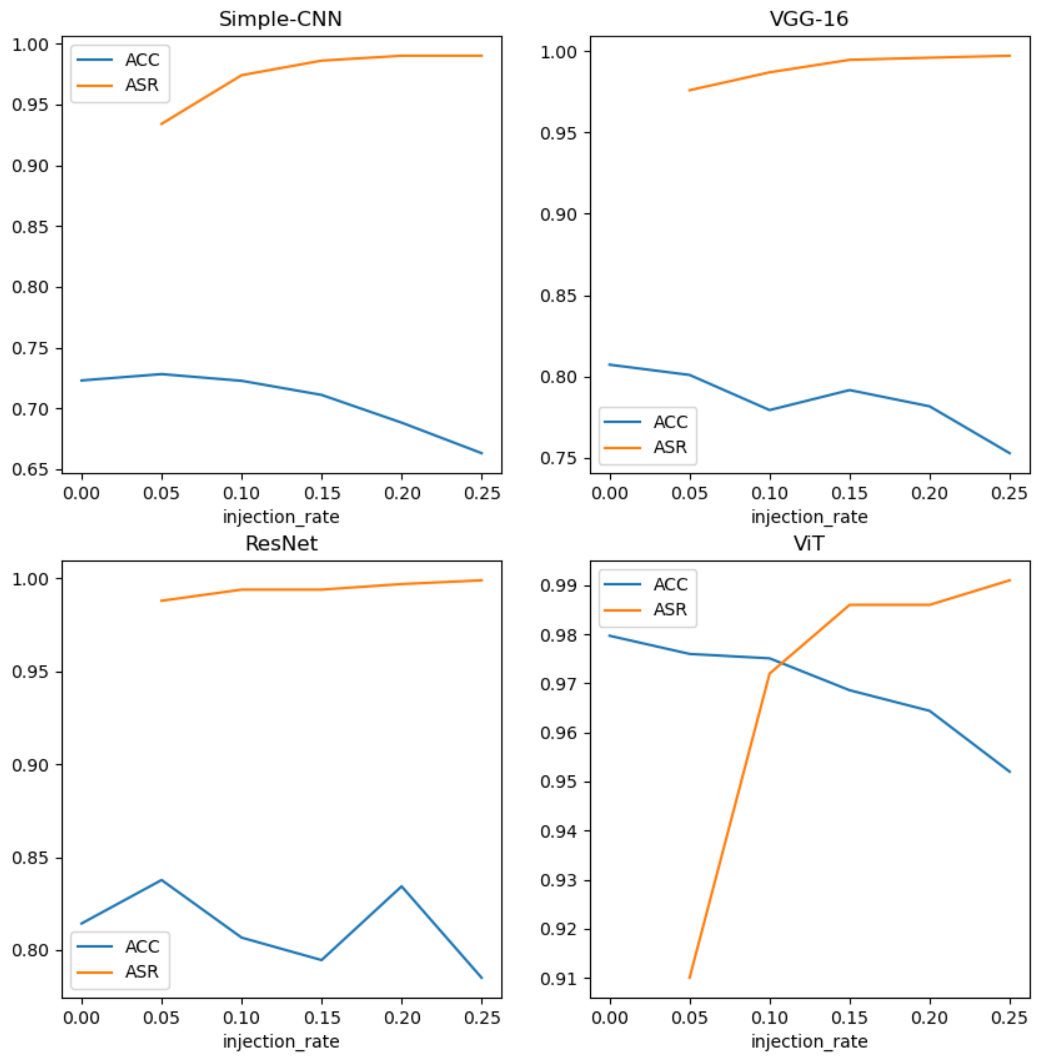


Figure 4: `ACC` and `ASR` Trends with Different Injection Rate on Four DNN Models

- $\mathbf{m}$: the mask, deciding how much the trigger can overwrite the original image.

When $\mathbf{m}_{i,j} = 1$ for a specific pixel (i, j), the trigger completely overwrites the original input ($\mathbf{x}'_{i,j,c} = \mathbf{\Delta}_{i,j,c}$), and when $\mathbf{m}_{i,j} = 0$, the original color is not modified at all ($\mathbf{x}'_{i,j,c} = \mathbf{x}_{i,j,c}$).

II Use an optimization scheme to find the "minimal" trigger required to misclassify all samples from other labels into this target label. In the vision domain, this trigger defines the smallest collection of pixels and its associated color intensities to cause misclassification.

The optimization has two objectives. For a given target label to be analyzed (y_t), the first objective is to find a trigger ($\mathbf{m}$, $\mathbf{\Delta}$) that would misclassify clean images into y_t. The second objective is to find a "concise" trigger, which measured by the $L1$ norm of the mask $\mathbf{m}$.

$$\min_{\mathbf{m},\mathbf{\Delta}} \ell(y_t, f(A(\mathbf{x}, \mathbf{m}, \mathbf{\Delta}))) + \lambda \cdot |\mathbf{m}|$$
$$\text{for } \mathbf{x} \in \mathbf{X}$$

, where
- $f(\cdot)$: the DNN prediction function.
- $\ell(\cdot)$: the cross entropy loss function, which measures the error in classification.
- λ: weight of the second objective. Smaller λ gives lower weight to controlling the size of the trigger.

Here we use the Adam [10] optimizer to deal with the above dual-objective optimization problem. Adam combines the concepts of momentum and adaptive learning rates to converge faster and more efficiently than traditional gradient descent-based methods like Stochastic Gradient Descent (SGD).

III Repeat Step 1 for each output label in the model. For a model with $N = |\mathbb{L}|$ labels, this produces N potential "triggers".

IV Measure the size of N potential "triggers" by the number of pixels each trigger candidate has, i.e. how many pixels the trigger is replacing ($L1$ norm of the mask m. Run an outlier detection algorithm to detect if any trigger candidate is significantly smaller than other candidates. A significant outlier represents a real *"minimum"* trigger, and the label matching that trigger is the target label of the backdoor attack. If no significant outlier is detected, we decide the DNN model is clean.

The outlier detection algorithm we use is *Median Absolute Deviation*. We first calculate the median of N reverse-engineered triggers' $L1$ norms, then compute the absolute deviations from the median for each trigger. After that, we find the median of these absolute deviations. The resulting value which called *anomaly index*, quantifies the typical distance of each trigger's $L1$ norm from the median of N triggers' $L1$ norms and provides a robust measure of the data set's variability. Finally under the assumption that the underlying distribution being Normal [1], we apply a constant estimator (1.4826) to normalize the anomaly index. If the largest anomaly index of certain model is ≥ 2, we decide that the model is a backdoored model, and the reverse-engineered trigger with the largest anomaly index is the target label of such backdoored model. Otherwise, if the largest anomaly index of certain model is < 2, we decide that the model is clean.

3.2.2 Results

In this study, we conducted a series of experiments to assess the efficacy of a reverse-engineering algorithm in detecting backdoors in models. Specifically, we repeated the experiment five times using both a clean model and a backdoored model with a target label of 0 (airplane) and a weight control parameter of $\lambda = 0.01$. We recorded the $L1$ norms for each reverse-engineered mask and computed the largest anomaly index for each experiment. As the defender may not have access to the full training data in some scenarios, we used limited testing data (size $= 10000$) with the same injection rate (15%) to optimize the models.

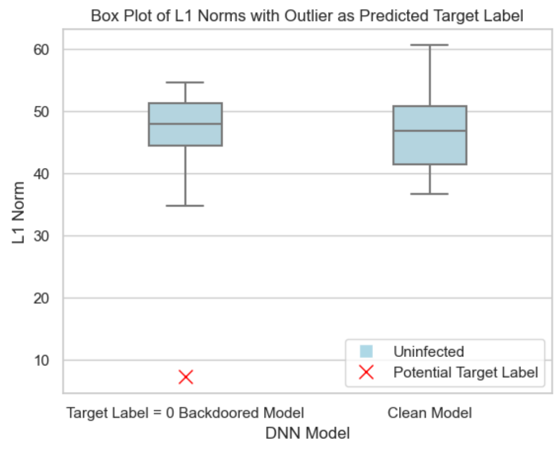
Our results show that the reverse-engineered triggers with the correct target label ($= 0$) have extremely small $L1$ norms that differentiate them from other triggers 1. The box plot 5a indicates that, in the first experiment, all $L1$ norms of reverse-engineered triggers fall within the range of $30 - 65$, except for one trigger with a norm less than 10, which precisely matchs the true target label of the backdoored model. We computed the anomaly index for all five experiments and found that the reverse-engineering algorithm successfully distinguishes the backdoored model from the clean model by checking whether the anomaly index is ≥ 2 (infected model) or < 2 (clean model) 5b.

[1]The $L1$ norm distribution is actually a non-negative and asymmetric distribution, but MAD also works. [11]

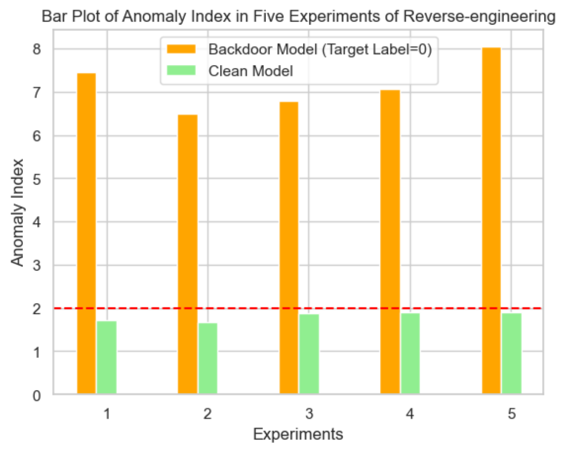
RE Target Label	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Experiment 1 Backdoored Model	7.26	47.2	44.1	51.4	53.8	45.8	50.7	48.6	54.6	34.7
Experiment 1 Clean Model	52.4	38.8	46.2	36.7	41.2	47.3	50.9	42.1	60.5	50.3
Experiment 2 Backdoored Model	9.23	50.4	40.4	41.5	39.4	52.7	48.3	55.5	51.7	41.4
Experiment 2 Clean Model	57.6	55.4	57.2	53.2	52.3	58.8	49.5	41.4	48.7	58.2
Experiment 3 Backdoored Model	8.39	50.0	55.4	43.7	57.5	42.8	49.0	44.4	39.3	45.4
Experiment 3 Clean Model	41.4	47.4	56.7	57.3	46.2	43.9	49.9	47.0	57.0	48.8
Experiment 4 Backdoored Model	7.63	47.0	45.4	55.8	59.8	52.9	39.7	58.8	53.2	46.3
Experiment 4 Clean Model	38.2	40.0	55.3	53.5	56.2	54.7	41.6	56.1	56.0	58.7
Experiment 5 Backdoored Model	6.99	49.9	43.5	50.8	56.6	47.2	51.4	52.2	56.0	39.3
Experiment 5 Clean Model	52.5	37.7	44.2	39.9	45.6	48.3	52.4	45.2	58.2	47.6

Table 1: $L1$ Norm of Five Experiments on Backdoored and Clean Models

Subsequently, in the first experiment, we proceed to examine the mask pattern of reverse-engineered triggers associated with all 10 target labels using the backdoored model, as depicted in Figure 6. Our findings reveal that after optimization, the reverse-engineered trigger linked with the true target label (0) contracts towards the bottom right corner, while all other reverse-engineered triggers exhibit spreading masks on the images. Thus, our reverse-engineered trigger with the correct target label has a pattern highly resembling the injected trigger pattern utilized during the training phase (i.e., a 4×4 all-black mask located at the bottom right corner of the input images). In contrast, for the clean model, none of the reverse-engineered masks shrinks into a very small area, as presented in Figure 7. This visual phenomenon offers an intuitive explanation of how the MAD method on $L1$ norms works in identifying backdoor models and predicting the true target label.



(a) $L1$ Norm Box Plot

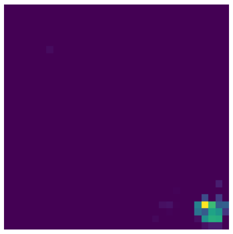


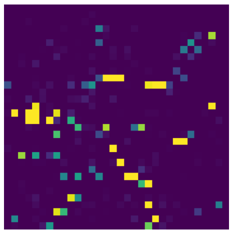
(b) Anomaly Index Bar Plot

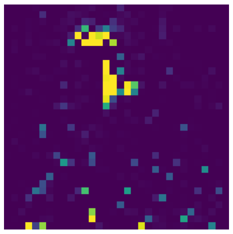
Figure 5: Comparison Between Backdoored and Clean Models

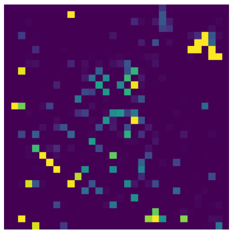
Upon validating the efficacy of the reverse-engineering approach on the backdoor model with a target label of 0, we proceed to train an additional nine backdoor models, each possessing a distinct target label ranging from 1 to 9 within the CIFAR-10 data set. Subsequently, we conduct reverse-engineering and Mean Absolute Deviation (MAD) analyses on each of these nine models. The findings demonstrate that our detection methodology accurately discerns all true target labels present in each backdoored model. For a comprehensive overview of the $L1$ norms associated with each experiment, please refer to *Appendix I 7*.

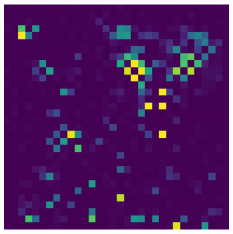
The anomaly index output reveals that the largest anomaly indices of the backdoored models are all $>= 2$, and the $L1$ norms of the reverse-engineered masks corresponding to the correct labels fall within the range of 5 to 11. This range is marginally smaller than the true 4×4 mask incorporated during the training phase, which exhibits an $L1$ norm of 16.

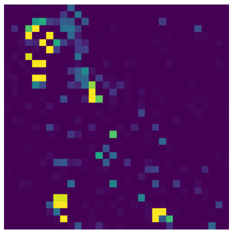

(a) Target Label 0

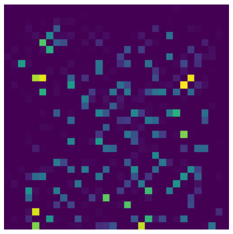

(b) Target Label 1

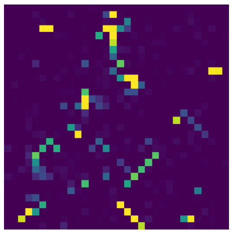

(c) Target Label 2

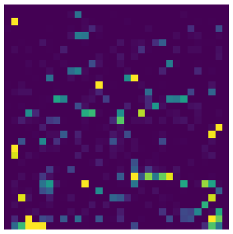

(d) Target Label 3


(e) Target Label 4


(f) Target Label 5


(g) Target Label 6


(h) Target Label 7


(i) Target Label 8

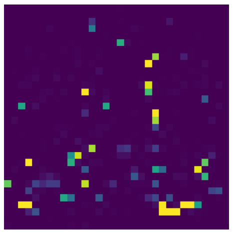
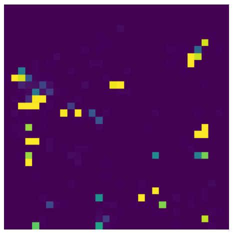
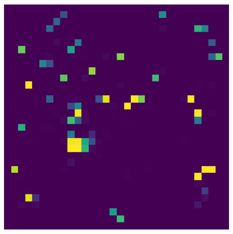
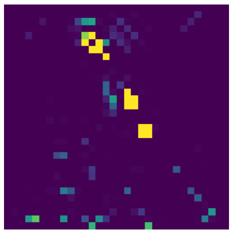
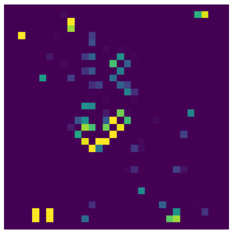

(j) Target Label 9

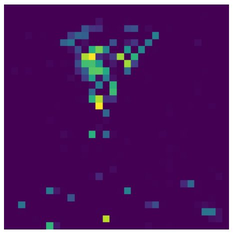
Figure 6: The Reverse-engineered Masks on Backdoored Model (True Target Label = 0)

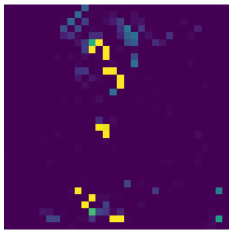

(a) Target Label 0

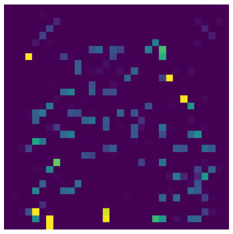

(b) Target Label 1

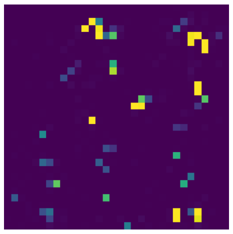

(c) Target Label 2

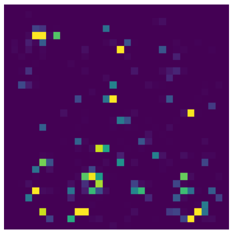

(d) Target Label 3


(e) Target Label 4


(f) Target Label 5


(g) Target Label 6


(h) Target Label 7


(i) Target Label 8

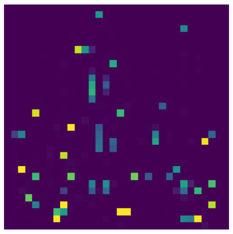

(j) Target Label 9

Figure 7: The Reverse-engineered Masks on Clean Model

Visual examination of mask patterns is conducted, revealing similar trends that indicate the reverse-engineered triggers exhibit a shrinkage phenomenon on the correct target label and not on others. The inverse masks have been provided in **Appendix I**. 4.

4 CONCLUSION AND FUTURE WORK

Our study focuses on implementing vanilla backdoor attacks on four Deep Neural Networks of varying complexities and achieving high testing attack success rates (`ASR`) on triggered data while maintaining high testing accuracy (`ACC`) on clean data. Additionally, we demonstrate the effectiveness of the reverse-engineering method in predicting whether a model is poisoned and, if so, which label is set as the target during training. Furthermore, we compare the reverse-engineered masks with the original injected triggers and find that they have similar patterns, with the reverse-engineered masks being more compact than the original triggers in terms of $L1$ norms.

With the increasing need for trustworthy Artificial Intelligence, we aim to provide more theoretical support for explaining the effects of injecting backdoors in "black-box" Deep Neural Networks. While our experiments demonstrate the universality of the backdoor attack across varying DNN complexities, we aim to explore the applicability of backdoor attacks on simpler models, such as multivariate logistic regression or Gaussian Processes. Additionally, we seek to enhance model robustness by increasing inter-class distances and reducing intra-class distances. To this end, we propose manipulating the model to increase its robustness, such as by adding white noise to certain layers. While early experiments have yielded consistent results, rigorous theoretical proof is still lacking, which we intend to contribute to in future research.

It is worth noting that our work builds on previous research by Gu et al. [3] and Wang et al. [6], both from 2019. However, as these methods are relatively outdated, we seek to update and expand upon their work. Additionally, recent research by Zhu et al. [12] offers theoretical insights into the effectiveness of existing attacks and proposes a new attack enhancement called Gradient Shaping (GRASP) to "hide" the backdoor from gradient-based trigger inversion.

Moving forward, our project aims to provide theoretical insights into the impact of injecting backdoors in "black-box" Deep Neural Networks. Our experiments demonstrate that the same backdoor attack works globally across varying DNN complexities. However, a key question arises as to how to apply backdoor attacks on simpler models such as multivariate logistic regression or Gaussian Processes. Additionally, we seek to enhance model robustness by increasing inter-class distances and reducing intra-class distances. This can be achieved by manipulating the model, such as by adding white noise to certain layers. While early experiments by Rui Zhu and his coauthors yield consistent results, rigorous theoretical proof is lacking, and we aim to contribute to this area of research in the future.

Acknowledgments

The author wishes to thank his advisor Dr. Chunfeng Huang for helpful discussions, feedback and guidance. He would also like to thank Rui Zhu, the Computer Science PhD student, and his advisor Dr. Haixu Tang, from Luddy School of Informatics, Computing, and Engineering at Indiana University for helpful discussions and suggestions.

References

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

[2] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, and Juan Zhai. Trojaning attack on neural networks. Network and Distributed System Security Symposium (NDSS), 2018.

[3] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[4] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. Neural Information Processing Systems (NeurIPS), 2018.

[5] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. ACM Asia Conference on Computer and Communications Security (ASIACCS), 2019.

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

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. International Conference on Learning Representations (ICLR), 2021.

[10] Diederik P. Kingma and Jimmy Ma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Peter J. Rousseeuw and Christophe Croux. Alternatives to the median absolute deviation. *Journal of the American Statistical Association*, 88:1273–1283, 1993.

[12] Rui Zhu, Di Tang, Siyuan Tang, Guanhong Tao, Shiqing Ma, Xiaofeng Wang, and Haixu Tang. Gradient shaping: Enhancing backdoor attack against reverse engineering. *arXiv preprint arXiv:2301.12318*, 2023.

Appendix I

DNN Architectures

Here we list the four Deep Neural Networks architecture used in our experiments.

Architecture		
Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 32, 32, 32)	896
conv2d_13 (Conv2D)	(None, 30, 30, 32)	9248
max_pooling2d_6 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_14 (Conv2D)	(None, 15, 15, 64)	18496
conv2d_15 (Conv2D)	(None, 13, 13, 64)	36928
max_pooling2d_7 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_3 (Flatten)	(None, 2304)	0
dense_7 (Dense)	(None, 512)	1180160
dense_8 (Dense)	(None, 512)	262656
dense_9 (Dense)	(None, 10)	5130
Parameters		
Total params: 1,513,514		
Trainable params: 1,513,514		
Non-trainable params: 0		

Table 2: Simple-CNN Model

Architecture		
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
conv2d_16 (Conv2D)	(None, 32, 32, 64)	1792
conv2d_17 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_8 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_18 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_19 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_9 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_20 (Conv2D)	(None, 8, 8, 256)	295168
conv2d_21 (Conv2D)	(None, 8, 8, 256)	590080
conv2d_22 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_10 (MaxPooling2D)	(None, 4, 4, 256)	0
conv2d_23 (Conv2D)	(None, 4, 4, 512)	1180160
conv2d_24 (Conv2D)	(None, 4, 4, 512)	2359808
conv2d_25 (Conv2D)	(None, 4, 4, 512)	2359808
max_pooling2d_11 (MaxPooling2D)	(None, 2, 2, 512)	0
conv2d_26 (Conv2D)	(None, 2, 2, 512)	2359808
conv2d_27 (Conv2D)	(None, 2, 2, 512)	2359808
conv2d_28 (Conv2D)	(None, 2, 2, 512)	2359808
max_pooling2d_12 (MaxPooling2D)	(None, 1, 1, 512)	0
flatten_4 (Flatten)	(None, 512)	0
dense_10 (Dense)	(None, 4096)	2101248
dense_11 (Dense)	(None, 4096)	16781312
dense_12 (Dense)	(None, 10)	40970
Parameters		
Total params: 33,638,218		
Trainable params: 33,638,218		
Non-trainable params: 0		

Table 3: VGG-16 Model

Architecture			
Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 32, 32, 3)]	0	[]
conv2d_29 (Conv2D)	(None, 32, 32, 64)	1792	['input_2[0][0]']
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256	['conv2d_29[0][0]']
re_lu (ReLU)	(None, 32, 32, 64)	0	['batch_normalization[0][0]']
conv2d_30 (Conv2D)	(None, 32, 32, 64)	36928	['re_lu[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256	['conv2d_30[0][0]']
re_lu_1 (ReLU)	(None, 32, 32, 64)	0	['batch_normalization_1[0][0]']
conv2d_31 (Conv2D)	(None, 32, 32, 64)	36928	['re_lu_1[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 32, 32, 64)	256	['conv2d_31[0][0]']
add (Add)	(None, 32, 32, 64)	0	['re_lu[0][0]', 'batch_normalization_2[0][0]']
re_lu_2 (ReLU)	(None, 32, 32, 64)	0	['add[0][0]']
conv2d_32 (Conv2D)	(None, 32, 32, 64)	36928	['re_lu_2[0][0]']
batch_normalization_3 (BatchNormalization)	(None, 32, 32, 64)	256	['conv2d_32[0][0]']
re_lu_3 (ReLU)	(None, 32, 32, 64)	0	['batch_normalization_3[0][0]']
batch_normalization_4 (BatchNormalization)	(None, 32, 32, 64)	256	['conv2d_33[0][0]']
add_1 (Add)	(None, 32, 32, 64)	0	['re_lu_2[0][0]', 'batch_normalization_4[0][0]']
re_lu_4 (ReLU)	(None, 32, 32, 64)	0	['add_1[0][0]']
conv2d_34 (Conv2D)	(None, 16, 16, 128)	73856	['re_lu_4[0][0]']
batch_normalization_5 (BatchNormalization)	(None, 16, 16, 128)	512	['conv2d_34[0][0]']
re_lu_5 (ReLU)	(None, 16, 16, 128)	0	['batch_normalization_5[0][0]']
conv2d_36 (Conv2D)	(None, 16, 16, 128)	8320	['re_lu_4[0][0]']
conv2d_35 (Conv2D)	(None, 16, 16, 128)	147584	['re_lu_5[0][0]']
batch_normalization_7 (BatchNormalization)	(None, 16, 16, 128)	512	['conv2d_36[0][0]']
batch_normalization_6 (BatchNormalization)	(None, 16, 16, 128)	512	['conv2d_35[0][0]']
add_2 (Add)	(None, 16, 16, 128)	0	['batch_normalization_7[0][0]', 'batch_normalization_6[0][0]']
re_lu_6 (ReLU)	(None, 16, 16, 128)	0	['add_2[0][0]']
conv2d_37 (Conv2D)	(None, 16, 16, 128)	147584	['re_lu_6[0][0]']
batch_normalization_8 (BatchNormalization)	(None, 16, 16, 128)	512	['conv2d_37[0][0]']
re_lu_7 (ReLU)	(None, 16, 16, 128)	0	['batch_normalization_8[0][0]']
conv2d_38 (Conv2D)	(None, 16, 16, 128)	147584	['re_lu_7[0][0]']
batch_normalization_9 (BatchNormalization)	(None, 16, 16, 128)	512	['conv2d_38[0][0]']
add_3 (Add)	(None, 16, 16, 128)	0	['re_lu_6[0][0]', 'batch_normalization_9[0][0]']
re_lu_8 (ReLU)	(None, 16, 16, 128)	0	['add_3[0][0]']
conv2d_39 (Conv2D)	(None, 8, 8, 256)	295168	['re_lu_8[0][0]']
batch_normalization_10 (BatchNormalization)	(None, 8, 8, 256)	1024	['conv2d_39[0][0]']
re_lu_9 (ReLU)	(None, 8, 8, 256)	0	['batch_normalization_10[0][0]']
conv2d_41 (Conv2D)	(None, 8, 8, 256)	33024	['re_lu_8[0][0]']
conv2d_40 (Conv2D)	(None, 8, 8, 256)	590080	['re_lu_9[0][0]']
batch_normalization_12 (BatchNormalization)	(None, 8, 8, 256)	1024	['conv2d_41[0][0]']
batch_normalization_11 (BatchNormalization)	(None, 8, 8, 256)	1024	['conv2d_40[0][0]']
add_4 (Add)	(None, 8, 8, 256)	0	['batch_normalization_12[0][0]', 'batch_normalization_11[0][0]']
re_lu_10 (ReLU)	(None, 8, 8, 256)	0	['add_4[0][0]']
conv2d_42 (Conv2D)	(None, 8, 8, 256)	590080	['re_lu_10[0][0]']
batch_normalization_13 (BatchNormalization)	(None, 8, 8, 256)	1024	['conv2d_42[0][0]']
re_lu_11 (ReLU)	(None, 8, 8, 256)	0	['batch_normalization_13[0][0]']
conv2d_43 (Conv2D)	(None, 8, 8, 256)	590080	['re_lu_11[0][0]']
batch_normalization_14 (BatchNormalization)	(None, 8, 8, 256)	1024	['conv2d_43[0][0]']
add_5 (Add)	(None, 8, 8, 256)	0	['re_lu_10[0][0]', 'batch_normalization_14[0][0]']
re_lu_12 (ReLU)	(None, 8, 8, 256)	0	['add_5[0][0]']

Table 4: ResNet Model

Architecture Cont'			
Layer (type)	Output Shape	Param #	Connected to
average_pooling2d (AveragePooling2D)	(None, 2, 2, 256)	0	['re_lu_12[0][0]']
flatten_5 (Flatten)	(None, 1024)	0	['average_pooling2d[0][0]']
dense_13 (Dense)	(None, 10)	10250	['flatten_5[0][0]']
Parameters			
Total params: 2,792,074 Trainable params: 2,787,594 Non-trainable params: 4,480			

Table 5: ResNet Model (Cont')

Architecture		
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
lambda (Lambda)	(None, 256, 256, 3)	0
vit-b16 (Functional)	(None, 768)	85844736
flatten (Flatten)	(None, 768)	0
batch_normalization (BatchNormalization)	(None, 768)	3072
dense (Dense)	(None, 32)	24608
batch_normalization_1 (BatchNormalization)	(None, 32)	128
dense_1 (Dense)	(None, 10)	330
Parameters (Pre-train Phase)		
Total params: 85,872,874 Trainable params: 26,538 Non-trainable params: 85,846,336		
Parameters (Transfer Learning Phase)		
Total params: 85,872,874 Trainable params: 85,871,274 Non-trainable params: 1,600		

Table 6: ViT Model

$L1$ Norm Results of Reverse-engineered Triggers on All Backdoored Models

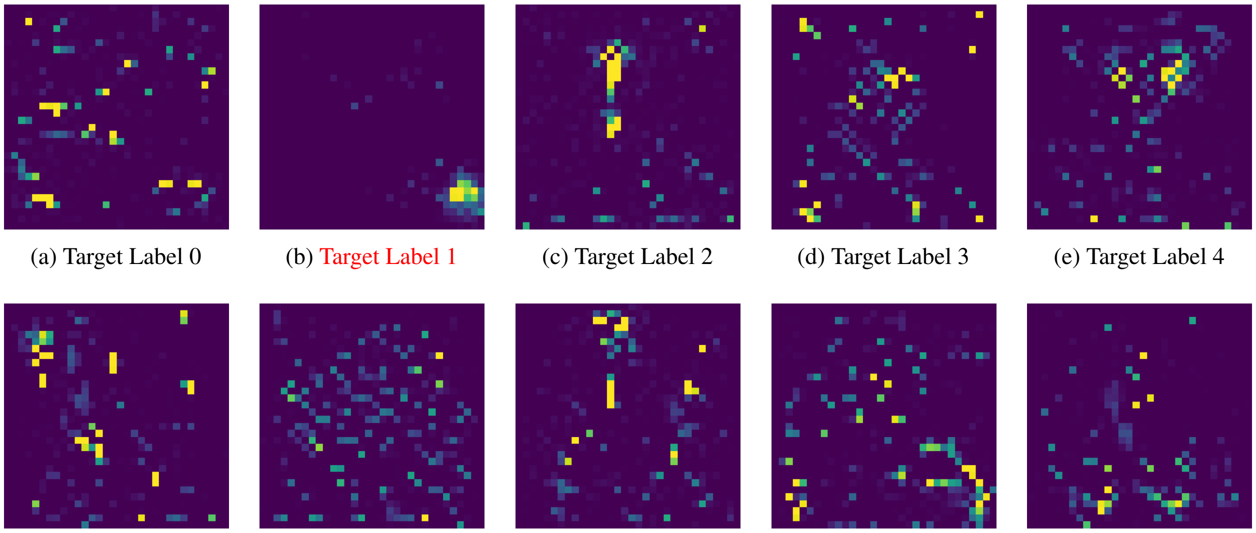
Here we list the $L1$ norm results of reverse-engineered triggers on all target-label $(0-9)$ backdoored models. Each row represents one backdoored model, with the true target label listed in the first column, and the $L1$ norm of inverse trigger with assumed target label in other ten columns.

RE Target Label	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	7.3	47.2	44.1	51.4	53.8	45.8	50.7	48.6	54.6	34.7
1	41.2	10.6	35.2	40.8	40.3	41.2	44.3	38.5	52	32.9
2	33.4	39.2	7.5	44.8	44.3	44.2	36.4	44	50	40.9
3	49.3	35.5	41.5	6.4	45	37.9	48.6	41.2	50	39.2
4	39.1	39.9	43.2	51.9	5.4	33.8	47.6	42.7	48.6	32.4
5	51.9	45.2	45.5	51.2	48.5	10.8	51.4	41.2	59.3	40.1
6	41.8	40	39.9	52.7	49.2	36.7	7	49.4	49.6	35.2
7	39	33.8	35.3	52.6	46.4	42.3	47.7	8.5	49.3	38.7
8	40.4	40.9	35.1	44.6	45	38.3	49.3	38.8	13.2	35.3
9	52.4	38.8	46.2	36.7	41.2	47.3	50.9	42.1	60.5	10.3

Table 7: $L1$ Norm of Ten Experiments on Backdoored Models with Different Target Labels

Reverse-engineered Masks on All Backdoored Models

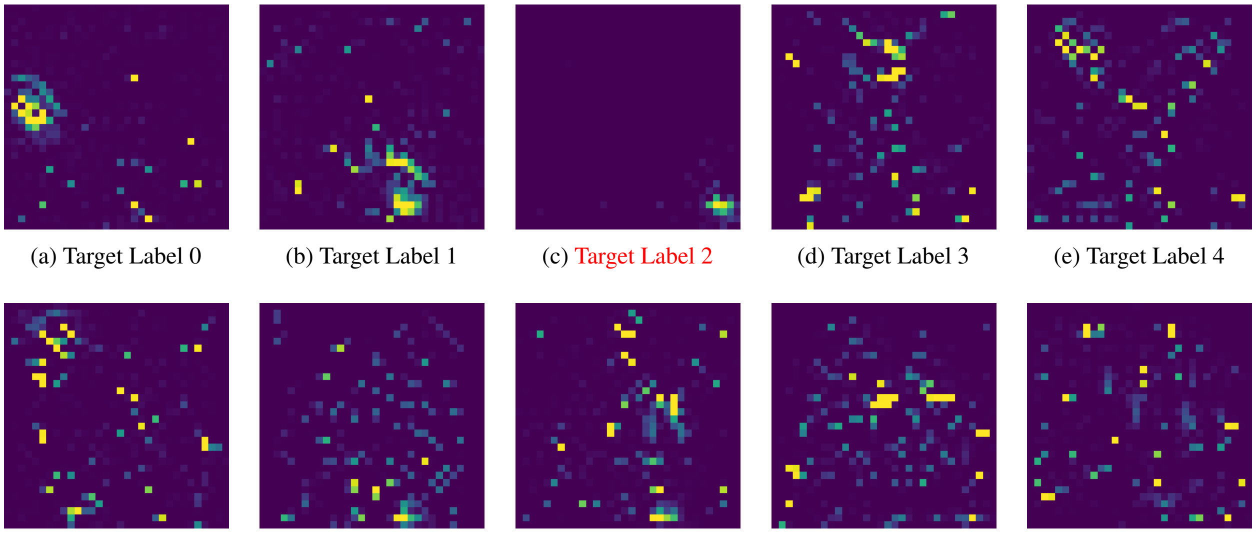
Here we show the pattern of reverse-engineered triggers on other target-label $(1-9)$ backdoored models.



(a) Target Label 0 (b) Target Label 1 (c) Target Label 2 (d) Target Label 3 (e) Target Label 4

(f) Target Label 5 (g) Target Label 6 (h) Target Label 7 (i) Target Label 8 (j) Target Label 9

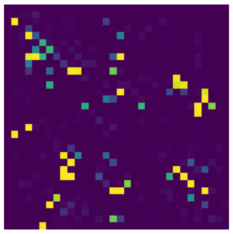
Figure 8: The Reverse-engineered Masks on Backdoored Model (True Target Label $= 1$)

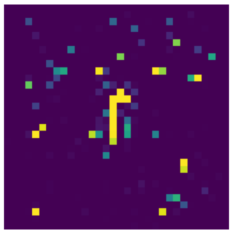


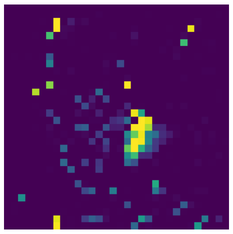
(a) Target Label 0 (b) Target Label 1 (c) Target Label 2 (d) Target Label 3 (e) Target Label 4

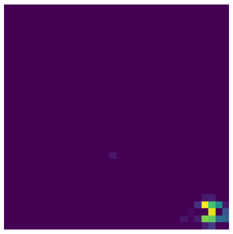
(f) Target Label 5 (g) Target Label 6 (h) Target Label 7 (i) Target Label 8 (j) Target Label 9

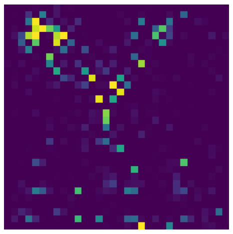
Figure 9: The Reverse-engineered Masks on Backdoored Model (True Target Label $= 2$)

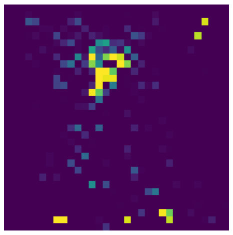

(a) Target Label 0

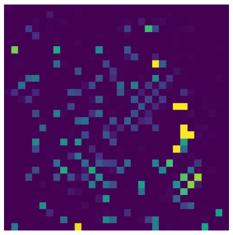

(b) Target Label 1

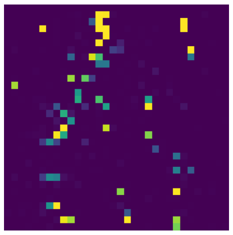

(c) Target Label 2

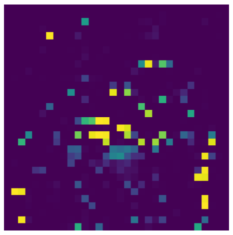

(d) Target Label 3


(e) Target Label 4


(f) Target Label 5


(g) Target Label 6


(h) Target Label 7


(i) Target Label 8

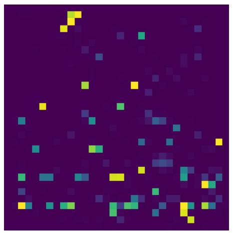
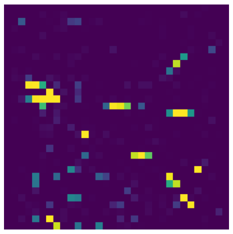
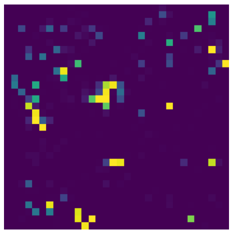
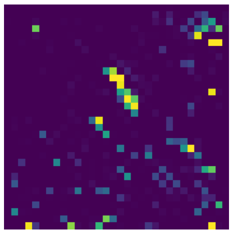
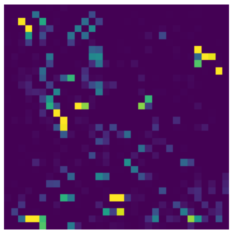

(j) Target Label 9

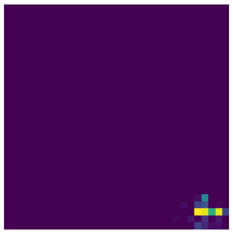
Figure 10: The Reverse-engineered Masks on Backdoored Model (True Target Label = 3)

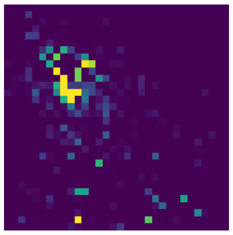

(a) Target Label 0

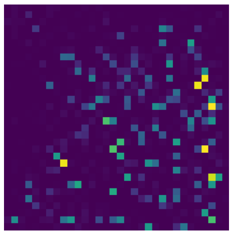

(b) Target Label 1

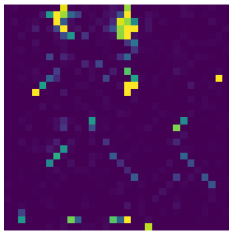

(c) Target Label 2

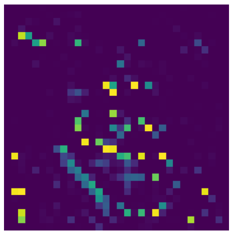

(d) Target Label 3


(e) Target Label 4


(f) Target Label 5


(g) Target Label 6


(h) Target Label 7


(i) Target Label 8

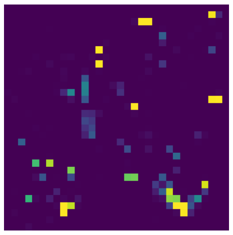
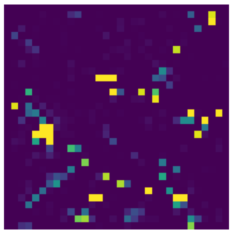
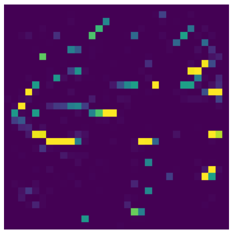
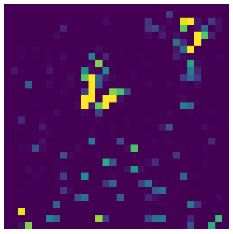
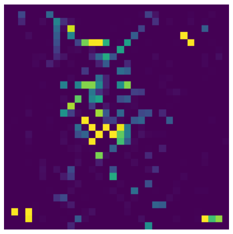

(j) Target Label 9

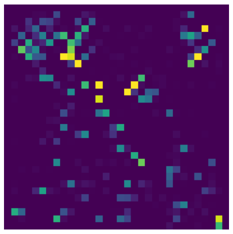
Figure 11: The Reverse-engineered Masks on Backdoored Model (True Target Label = 4)

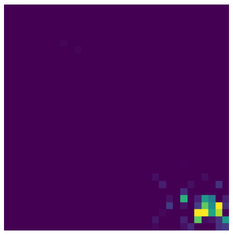

(a) Target Label 0

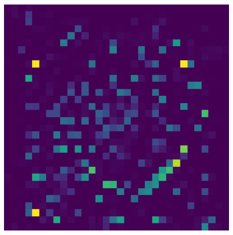

(b) Target Label 1

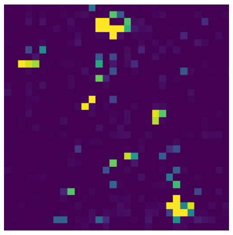

(c) Target Label 2

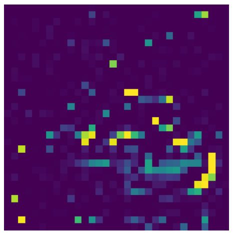

(d) Target Label 3


(e) Target Label 4


(f) Target Label 5


(g) Target Label 6


(h) Target Label 7


(i) Target Label 8

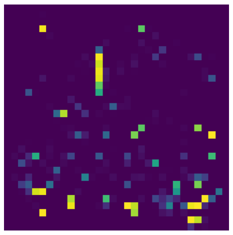
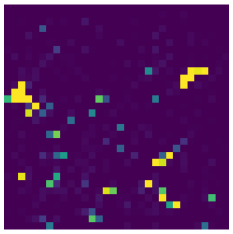
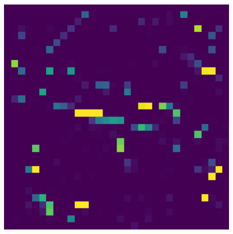
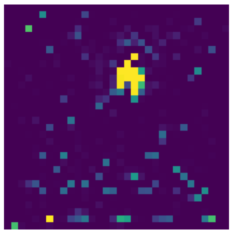
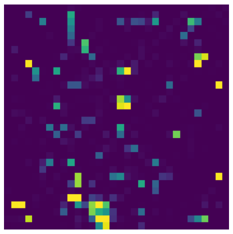

(j) Target Label 9

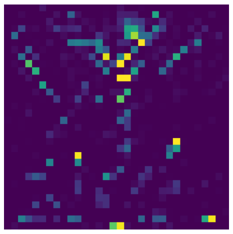
Figure 12: The Reverse-engineered Masks on Backdoored Model (True Target Label = 5)

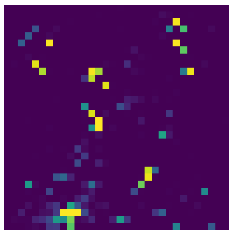

(a) Target Label 0

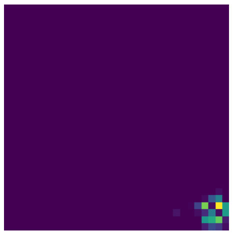

(b) Target Label 1

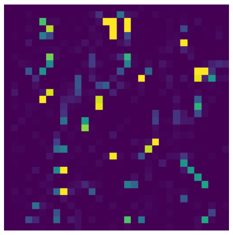

(c) Target Label 2

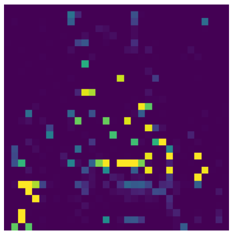

(d) Target Label 3


(e) Target Label 4


(f) Target Label 5


(g) Target Label 6


(h) Target Label 7


(i) Target Label 8

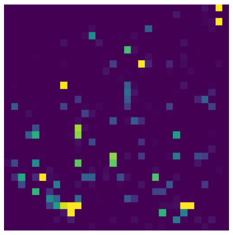
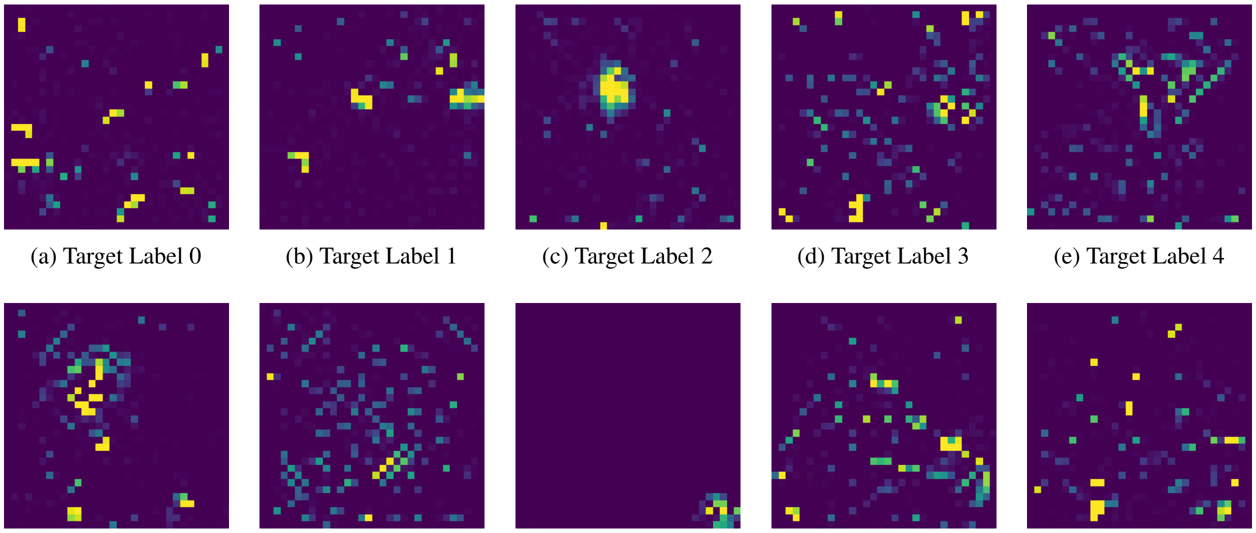

(j) Target Label 9

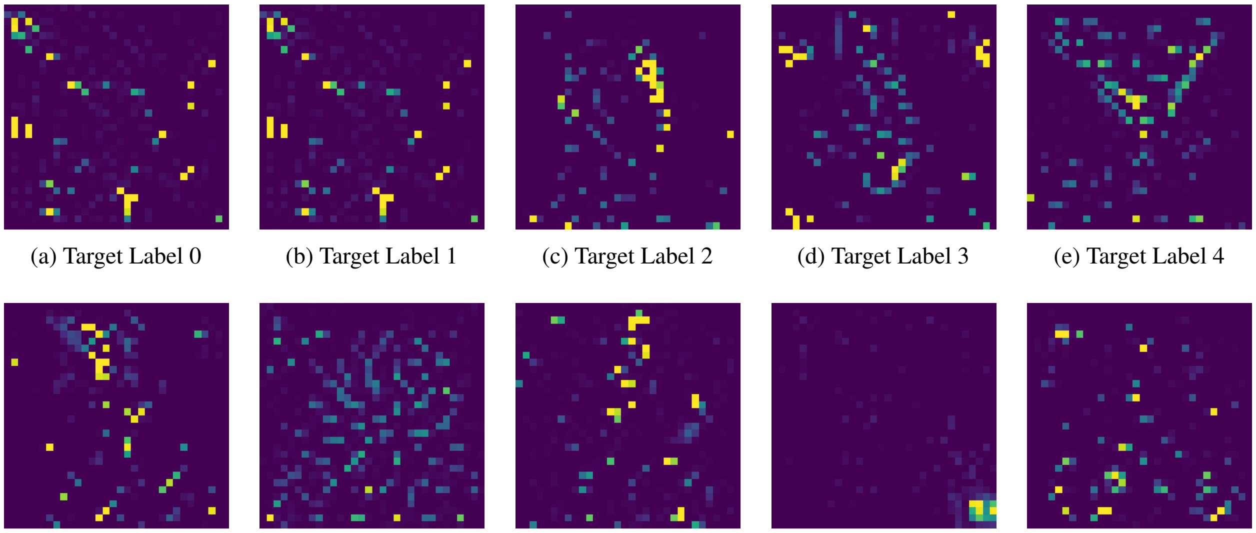
Figure 13: The Reverse-engineered Masks on Backdoored Model (True Target Label = 6)



(a) Target Label 0 (b) Target Label 1 (c) Target Label 2 (d) Target Label 3 (e) Target Label 4

(f) Target Label 5 (g) Target Label 6 (h) Target Label 7 (i) Target Label 8 (j) Target Label 9

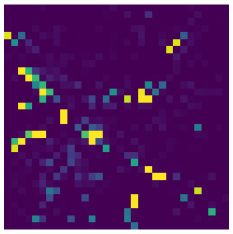
Figure 14: The Reverse-engineered Masks on Backdoored Model (True Target Label = 7)

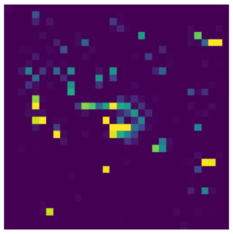


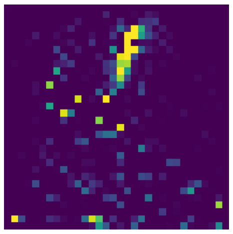
(a) Target Label 0 (b) Target Label 1 (c) Target Label 2 (d) Target Label 3 (e) Target Label 4

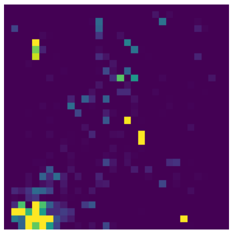
(f) Target Label 5 (g) Target Label 6 (h) Target Label 7 (i) Target Label 8 (j) Target Label 9

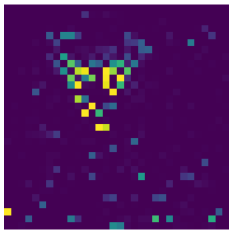
Figure 15: The Reverse-engineered Masks on Backdoored Model (True Target Label = 8)

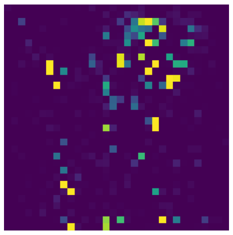

(a) Target Label 0

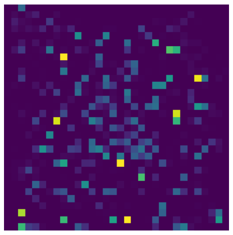

(b) Target Label 1

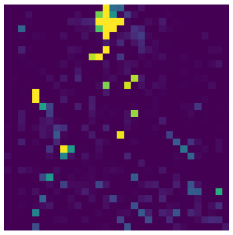

(c) Target Label 2

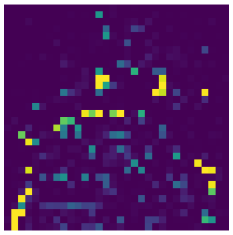

(d) Target Label 3


(e) Target Label 4


(f) Target Label 5


(g) Target Label 6


(h) Target Label 7


(i) Target Label 8

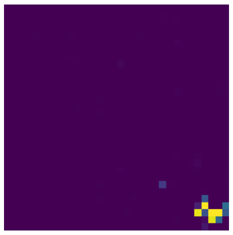

(j) Target Label 9

Figure 16: The Reverse-engineered Masks on Backdoored Model (True Target Label = 9)