PROGRAMMABLE NEURAL NETWORK TROJAN FOR PRE-TRAINED FEATURE EXTRACTOR

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural network (NN) trojaning attack is an emerging and important attack that can broadly damage the system deployed with NN models. Different from adversarial attack, it hides malicious functionality in the weight parameters of NN models. Existing studies have explored NN trojaning attacks in some small datasets for specific domains, with limited numbers of fixed target classes. In this paper, we propose a more powerful trojaning attack method for large models, which outperforms existing studies in capability, generality, and stealthiness. First, the attack is programmable that the malicious misclassification target is not fixed and can be generated on demand even after the victim's deployment. Second, our trojaning attack is not limited in a small domain; one trojaned model on a large-scale dataset can affect applications of different domains that reuses its general features. Third, our trojan shows no biased behavior for different target classes, which makes it more difficult to defend.

1 INTRODUCTION

Neural Network (NN) Trojaning Attack or *Neural Network Backdoor Injection Attack* is an important attack model that can broadly damage the system based on NN models (Dumford & Scheirer, 2018; Liu et al., 2017; Liao et al., 2018; Gu et al., 2017). NN trojaning attacks hide malicious functionality inside the weights of an NN model, either by poisoning data set or weight perturbation (Gu et al., 2017). The trojaned NN model predicts correct labels normally for legitimate inputs, and only misclassifies the inputs with trigger patterns to predefined target labels. The NN models are essentially just a set of weight parameters connected with certain network architecture. Their behavior highly depends on the weight parameters, but the meanings are completely implicit. Thus, modifying the weight parameters usually shows no difference to consumers. To note, it is a different attack model from adversarial attacks (Kurakin et al., 2016), which craft adversarial inputs to mislead NN models.

The NN trojaning attack becomes an emerging practical and destructive attack model because of the broad usage of pre-trained models. Training a neural network with good features requires not only a large number of computing resources but also large-scale datasets. Thus, using pre-trained models is a common practice in developing NN-based applications to reuse expensive well-learned features. Accordingly, there are many open-source pre-trained models available online. They are produced by various companies, open-source communities, or personal maintainers, and consumed by end users who may use these models directly or reuse part of them for a particular task. These pre-trained models benefit the agile deployment and boom the NN technique evolution. However, they also raise security issues since some vicious model promulgators can hide malicious functionalities in the clean model and release them for public use, which can be easily spread. Therefore, it is important to explore and understand the NN trojaning attacks.

Although the trojaning attack requires attackers to be capable of modifying the weight parameters of the NN model, it does not have to be an entire white-box. In terms of the attack scenarios, such trojaning attacks can be classified into two types, *outsourced training attack* and *transfer learning attack*. The first assumes that the victims will use the trojaned model directly without any further modification. This kind of attack is completely white-box and most existing studies focus on this assumption (Dumford & Scheirer, 2018; Liu et al., 2017; Liao et al., 2018). However, in real cases, victims often employ pre-trained NNs as well-learned feature extractors and further develop their own models (Gu et al., 2017). Therefore, the trojan attacks should resist victims' modifications, which is referred to as the *transfer learning attack* (Gu et al., 2017). One of such studies, BadNet (Gu et al., 2017), has explored the transfer learning attack in some small datasets on traffic sign.

Thus, although existing studies make initial steps that explore the potential effectiveness of trojan in transfer learning, their methodologies are restricted in small domains and validated on small datasets. For general feature extractors that are trained on large datasets and are used broadly, the attack is more challenging and the existing trojaning methods cannot be applied to this scenario: The victim tasks are completely unknown to the attackers and the target label that the attackers misleadingly train the trojaned model to recognize may even not be involved in the victim task.

For example, one of the official tutorials provided by TensorFlow[1] introduces the transfer learning scenario for image classifications. They use ImageNet (Deng et al., 2009) pre-trained models as well-learned image feature extractors and retrain the fully connected (FC) layers for new tasks on smaller Flower datasets. The official tutorial provided by MXNet[2] also introduce this scenario that transfer pretrained VGG16 model for Caltech-256 dataset. In natural language processing (NLP) field, it is also a popular practice to reuse BERT (Devlin et al., 2018) as pre-trained word-level features to solve many different kinds of NLP tasks. These scenarios are more realistic and trojans on those general features will affect a large scope of applications. Thus, it is important to explore the trojaning attack on the general feature extractors.

Another limitation of existing studies is that the trigger patterns of the trojans are usually handcrafted patterns for only one or few target classes. The limited diversity makes the trojans highly correlated with the trigger patterns. Defense methods (Chen et al., 2018; Liu et al., 2018a; Wang et al., 2019) based on statistic could detect or erase these trojans easily.

In this paper, we propose an NN trojaning attack method that is much more powerful, general and stealthy. Instead of using a static set of handcrafted patterns to trigger a predefined target class, we use dynamic patterns to trigger any intended target class, which makes our trojan attack programmable. We can use a *target image* to describe the target class and generate trigger pattern based on this image to encode and pass the information of misclassification target to the trojan.

The dynamic trigger pattern makes our trojan much more powerful and general: Even if the explicit classes used in victim model are not involved in the pre-trained model and unknown to attackers, they can still describe the input they expect the victim model to see with a *target image* and then generate the corresponding pattern to trigger the malicious behavior. Further, the dynamic trigger also greatly increases the diversity of trigger patterns, which makes it more stealthy.

We demonstrate our attack method under the scenario described in the retraining tutorial from Tensorflow and MXNet, which uses pre-trained ImageNet (Deng et al., 2009) models and replaces the FC layers for the Flower dataset and Caltech-256 dataset. We insert a trojan into the ImageNet model and attack the victim model for the two smaller datasets. The trojan remains effective for both cases. Note that, the classes in the Flower dataset and Caltech-256 are not involved in the 1000 classes of ImageNet and attackers have no access to these datasets. The same trojaned model can affect victims using any other dataset.

2 RELATED WORK

Neural network shows vulnerabilities to the crafted adversarial inputs, which is referred to as *adversarial attack* (Kurakin et al., 2016). NN trojan is another important attack model which can broadly damage the systems based on NN models. In such an attack model, the NN model intellectual property (IP) vendors could be the potential attackers who hide malicious functionalities in the pre-trained NNs (Liu et al., 2017; 2018b; Wang et al., 2019). These models perform normally with legitimate inputs and can export targeted or untargeted outputs with the trigger inputs.

Previous studies have made some initial steps in the NN trojaning techniques. In most existing studies (Dumford & Scheirer, 2018; Liu et al., 2017; Liao et al., 2018), they assume that the victim will adopt the pre-trained NN models directly, which is termed *outsourced training attack*. However,

[1]*How to Retrain an Image Classifier for New Categories.* `https://www.tensorflow.org/hub/tutorials/image_retraining`

[2]*Fine-tune with Pretrained Models.* `https://mxnet.incubator.apache.org/versions/master/faq/finetune.html`

Table 1: Comparison between our work and related work

Capability	Transferability	Outscope target	Dynamic target	Large Dataset
Dumford & Scheirer (2018)	×	×	×	×
Liu et al. (2017)	×	×	×	×
Liao et al. (2018)	×	×	×	×
Gu et al. (2017)	✓	×	×	×
Ours	✓	✓	✓	✓

this situation rarely actually occurs. In practice, users typically fine-tune the FC layers of the pre-trained models to adapt to their own working scenarios, which makes the attack more challenge; it is termed as *transfer learning attack*. Although the most related work, BadNet (Gu et al., 2017), has implemented a transfer learning attack, the triggers in their work are based on handcrafted patterns, which are statistically fixed. Therefore, their triggers can only support fixed target classes that are included in the pre-trained models. It cannot be applied to the scenario we demonstrate in this paper. Further, existing studies only demonstrated a high success rate of trojaning attack on small dataset such as MNIST (Dumford & Scheirer, 2018; Liao et al., 2018; Gu et al., 2017; Liu et al., 2017; Wang et al., 2019), face recognition (Dumford & Scheirer, 2018; Wang et al., 2019), traffic sign (Liao et al., 2018; Gu et al., 2017; Chen et al., 2018), and CIFAR10 (Chen et al., 2018). But people seldom use pre-trained models on these tiny datasets from an untrusted source. We compare our work with related studies in Table 1: We support target classes outside the pre-trained models, termed as *outscope target*, and the target class is not fixed, termed as *dynamic target*. These properties make our attack much more powerful. We also demonstrate the attack on ImageNet.

There are also some initial studies about defense of the NN trojan. Some detect if the dataset is poisoned (Chen et al., 2018), some detect if the model is poisoned by comparing the decision boundary of different classes (Wang et al., 2019), and some try to remove the trojan by squeezing the redundencies (Liu et al., 2018a). Most of them just work on trojaning attacks with just one or a few fixed target labels; in Section 5.3 we will analyze their effects on the proposed attack model.

3 Threat Model

Figure 1 shows a typical flow of *transfer learning attack* for NN trojans (Gu et al., 2017). For the ease of understanding, we first explain several terminologies. The start of the flow is a pre-trained NN model, denoted as *clean model*; its task is *original task*. The network architecture of the clean model usually consists of a *frontend model* and a *backend model*. The frontend model produces *general features* for a certain domain, which is intended to be reused by victims. The clean model usually comes from public model zoos or produced by attackers. Then, the threat model usually contains the following three phases.

Trojaning Phase. In this phase, attackers can fully access and make modifications to the entire clean model. They usually modify the frontend model only to hide the trojan because the backend model is replaced in later phases. The modified frontend is denoted as *trojaned frontend* and the entire model is now denoted as *trojaned model*. The trojaned model has the same network architecture as the clean model. The only difference is the weight parameters.

Victim Phase. The trojaned model is then distributed online and reused by victims. Victims intend to reuse the well-learned general features from the frontend model for their new tasks, denoted as *victim task*. It is typically done by designing a new backend, *victim backend*. And the entire model now is denoted as *victim model*. Note that the victim backend is unknown to attackers, including the explicit classes involved. On the other hand, although victims can fully access the trojaned model, they are unaware of the explicit method to trigger the malicious functionality.

Trigger Phase. The victim model is then deployed to real applications and the applications may also integrate other components. Now, victims are still capable of accessing the runtime information of their victim model and the system. But for attackers, it is a black box now except for the application scenario. Attackers can make small modifications to the input to trigger the malicious functionality in the trojaned frontend in order to control the behavior of the system. The modification that can be made highly depends on the scenario of victim task.

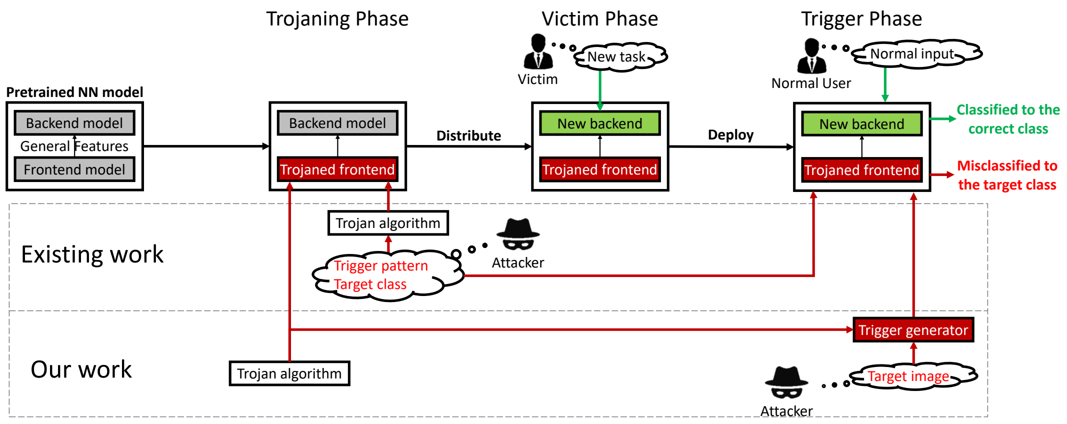

Figure 1: Attack methodology comparison. In existing studies, attackers should decide the trigger pattern and target class pairs in trojaning phase. In contrast, our method inserts a general trojan in trojaning phase and decide the target class in the trigger phase.

In most real applications, the modification is a small patch in the input image, denoted as *trigger pattern*. The input image patched is called *source image*, and its corresponding label is *source class*. The source image patched with the trigger pattern is denoted as *trigger image*. The corresponding misclassification target is *target class*.

Note that, although attackers may also perform a black-box adversarial attack in trigger phase and also lead to misclassification. The sources of the two threats are completely different. Thus, defending the adversarial attack will not reduce the risk of trojaning attack.

4 Method

4.1 Attack Methodology

The major contribution of our work is the new attack methodology, which greatly extends the power of the trojaning attack.

In the workflow of the existing transfer learning attack shown in Figure 1, attackers choose the trigger pattern and the corresponding target class in trojaning phase, and then modify the frontend model to recognize the trigger pattern without affecting its behavior for normal inputs. Then, in the trigger phase, attackers will present the trigger pattern in a normal input to trigger the misclassification as the target class. This attack flow has two major drawbacks.

• **Fixed targets.** The target classes are decided in trojaning phase. Attackers cannot choose targets on demand in the trigger phase.

• **In-scope targets.** In the trojaning phase, the victim task is completely unknown to the attacker. It is difficult to support target classes that are not included in the class set of the original task.

In Table 1, none of the existing studies support outscope and dynamic target due to these drawbacks.

We propose a new attack methodology as shown in Figure 1. The difference is that in the trojaning phase, we insert a more powerful programmable trojan and create a corresponding trojan generator. In the trigger phase, we use a *target image* to indicate the target class. The generator will encode the target image into a trigger pattern. It will be presented in the input image and the trojaned frontend can decode the trigger pattern and misclassify the input as the target class defined by the target image. The proposed attack methodology solves the two drawbacks due to the following designs.

• **Select targets in the trigger phase.** We insert a general trojan in the clean model and select the target class later in the trigger phase.

• **Describe targets with target images.** We use a target image instead of a target class to describe the intended behavior. It can support any target class on demand.

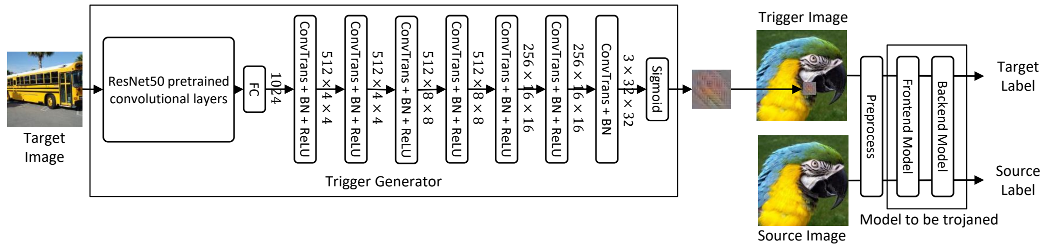

Figure 2: Trojan Insertion. We train a trigger generator together with the front-end model. The left side shows the network architecture of our generator. It accepts a target image and uses ResNet50 pretrained convolutional layers to encode it into a 1024-length vector. Then it use multiple transposed convolutional layers to generate the trigger pattern from the vector. The trigger pattern will replace part of the source image to form the trigger image. Then it will be fed into the model to be trojaned and trained to predict the target label. To keep the original functionality. Normal source image will also be fed into the model and trained to predict the source label.

Moreover, in the trigger phase, attackers may be still unaware of the explicit class sets of the victim model, while the expected behavior of the application system is known to attackers and the victim task is just a sub-task of the application system. Accordingly, with the target image, attackers can program the expected behavior of the application system directly without the information of the explicit classes of the victim task.

4.2 Trojan Insertion

The power of our attack flow highly depends on the capability of the trojanend frontend and the trigger generator. The trigger generator is able to encode any target image into a small trigger pattern. The trojaned frontend is able to recognize and decode the small trigger pattern patched in normal inputs. Then it further distorts its output general features to represent the target image instead of normal inputs. The modified general features will then mislead the victim backend and consequent application systems. At the same time, the trojaned frontend will not change the behavior for normal inputs.

To generate the trigger image, we use an NN to implement the generator, denoted as g. It creates a trigger pattern $z = g(x_{target})$ from a target image x_{target}. Then, the trigger pattern z will be patched to a source image x_{source} to form a trigger image $x_{trigger} = P(x_{source}, z)$, where P is a function to patch z in a random position of x_{source}. Thus, we have

$$x_{trigger} = P(x_{source}, g(x_{target})) \tag{1}$$

P is derivative: its gradients only need to propagate to the region of the trigger pattern directly.

To train the trigger generator and the trojan, we expect the victim model f_{victim} to predict the target class y_{target} with $x_{trigger}$ as input and predict the source class y_{source} with x_{source} normally. Thus, the loss to be minimized should be as Formula 2, where L is the cross-entropy loss and α is a hyper-parameter to control the weights of normal behavior and trojan behavior. Moreover, x_{target}, y_{target} and x_{source}, y_{source} should be sampled from victim's dataset.

$$\alpha L[f_{victim}(x_{trigger}), y_{target}] + (1-\alpha)L[f_{victim}(x_{source}), y_{source}] \tag{2}$$

However, attackers neithor know the victim model f_{victim} nor have access to victim's dataset. In order to train the trojaned frontend and the trigger generator, attackers can use the original model f_{origin} and dataset for the original task instead. The reason is that the victim model f_{victim} and dataset can work well on the frontend model of f_{origin} pre-trained with the original dataset. Thus, it is feasible to train a general trojan, as well. The loss function we use is in Formula 3. x_{target}, y_{target} and x_{source}, y_{source} are sampled independently from original dataset.

$$\alpha L[f_{origin}(x_{trigger}), y_{target}] + (1-\alpha)L[f_{origin}(x_{source}), y_{source}] \tag{3}$$

We use the stochastic gradient descent (SGD) algorithm to optimize the parameters of g and the parameters in the frontend of f_{origin}. For convolutional neural networks (CNNs), the frontend is typically the convolutional layers. f_{origin} is initialized with the clean model and g is initialized

Table 2: The accuracy of clean and trojaned model and the attack success rate on ImageNet models.

Model	**Clean Model Accuracy** top1 / top5	**Trojaned Model Accuracy** top1 / top5	**Attack Success Rate** top1 / top5
VGG16	73.37%/91.50%	72.37%/90.96%	50.27%/75.87%
ResNet50	76.15%/92.87%	73.88%/91.66%	37.55%/65.34%
MobileNet-V2	71.81%/90.42%	69.32%/89.14%	31.04%/57.64%

randomly. In the loss function, the gradients to g are multiplied with α, which is a very small number. Thus, we scale the gradient of parameters in g by $1/\alpha$ to have a balanced update between the generator and the trojan.

Figure 2 shows the network architecture. The left side is the generator and the right side is the model to be trojaned, including its frontend and backend parts. In the generator, we first use the convolutional layers of the pre-trained ResNet50 (He et al., 2016) and a FC layer to encode the target image into an internal feature vector of length 1024. Then, we use several transposed convolutional layers to generate a 32×32 trigger pattern, which is the typical network architecture for image generation in generative adversarial networks (GANs) (Radford et al., 2015). Specifically, we use a sigmoid function in the last layer to produce pixel values between 0 and 1, and then scale each pixel to the interval between 0 and 255. It will be further normalized with the mean and variance values of the ImageNet dataset, which is a typical pre-processing step for ImageNet models. Finally, we patch the trigger pattern in a random position in the source image and feed it into the model to be trojaned. The frontend part is its convolutional layers and the backend part is its FC layers. Note that, during the training, we fix the parameters of the backend model and the pre-trained ResNet50 in the trigger generator.

5 Experiment and Result

5.1 Outsourced Training Attack Effectiveness

We first demonstrate the outsourced training attack on ImageNet models to show the properties of the trojan without victims' modifications.

Setup. We implement the trojan insertion method with PyTorch. We choose VGG16 (Simonyan & Zisserman, 2014), ResNet50 (He et al., 2016), and MobileNet-V2 (Sandler et al., 2018) as the model to be trojaned. Initially, we set α to 10^{-3} and choose 10^{-3} as the learning rate for all cases. Then, we decrease the learning rate by $10\times$ every 10 epochs. After the loss function converge, we change α to 10^{-4}, restore the learning rate of target model to 10^{-4} and fine-tune the generators and trojans, which enables higher accuracy for the cases of VGG16 and ResNet50. MobileNet-V2 is slightly different: α is set to 5×10^{-4} at the fine-tuning phase.

The attack success rate is one of the most important metrics to evaluate the effectiveness of trojaning attacks. But the value of the attack success rate highly depends on the measurement. We choose the success rate of most general scenario to show the effectiveness of our attack: We *randomly* select a target image and a source image independently from ImageNet dataset. Then we generate the trigger pattern based on the target image and patch it in a *random* position of the source image to form a trigger image. Only when the predicted label is the target label, it is regarded as a *success*. During this measurement, the generator and trojan are unchanged for different target and source image.

Results. In the first step, we evaluate our trojaning attack on VGG16, ResNet50, and MobileNet-V2 in the outsourced training attack scenario. Table 2 shows the accuracy of the clean model and the trojaned model. The accuracy drop is within the accuracy variation of these models. Meanwhile, we achieve a high attack success rate. The trojaned VGG16 has a 50.27% attack success rate across 1000 target classes. Note that the attacker represents the target class with a target image. Therefore, the attack success rate cannot surpass the prediction accuracy of the trojaned model theoretically because the target image first has to be recognized as the target class. Thus, the trojaned VGG16's attack success rate, 50.27% is a high attack success rate since it is comparable with the recognition accuracy, 72.38%. The hyperparameter α is important for the tradeoff between maintaining prediction accuracy on normal inputs and increasing attack effectiveness on trigger inputs. We find that $\alpha = 10^{-3}$ would be the sweet point. We can further increase the attack success rate by applying a larger α, but it will lead to more accuracy drop of the trojaned model.

Table 3: Transfer learning attack results. We use the same trojaned VGG16 model to test the transfer attack success rate on two smaller dataset, Flower and Caltech-256. The trojaned model is made with only ImageNet dataset. Smaller datasets are only used to train victim's FC layers.

Dataset	**Clean Model Accuracy**	**Trojaned Model Accuracy**	**Attack Success Rate**
Flower Dataset	91.70%	91.56%	38.15%
Caltech-256	72.80%	73.37%	37.63%

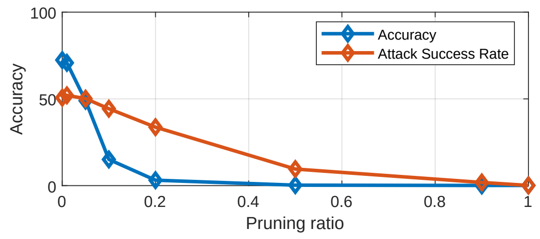

Figure 3: The accuracy vs. attack success rate with different pruning factor for Fine-Pruning (Liu et al., 2018a) defense method.

5.2 Transfer Learning Attack Effectiveness

We also demonstrate an end-to-end transfer learning attack. To make the scenario more realistic, we assume that the classes involved in victim task are outside the 1000 classes of ImageNet dataset and are unknown to the attacker during the trojaning phase. None of the existing trojaning attack methods can be applied to this scenario.

Setup. We follow the scenario described in the official tutorials of Tensorflow and MXNet. Based on the features extracted by convolutional layers, they train a new classifier with new FC layers for the Flower dataset and Caltech-256 dataset. We first insert a trojan in the VGG16 pre-trained model to create the trojaned model. This step should be performed by the attackers. Then, we train new FC layers for the smaller datasets and reuse the convolutional layers. This step is performed by the victims. Finally, we pick two random images from the validation set, one as source image and one as target image, to form a trigger image. We have eliminate the cases that source image and target image are from the same class.

Result. In Table 3, We show the transferability of our trojaned model. We use the trojaned VGG16 model mentioned in Table 2 to test its effectiveness on two smaller datasets, the Flower dataset and the Caltech-256 dataset. The two datasets are unknown when trojaning the VGG16 model and the classes in these datasets are completely different from the 1000 classes in ImageNet. Thus, non of existing studies can attack the victims successfully in this case because their misclassification target must be one of the 1000 classes of ImageNet. Our trojaned model can still achieve about 38% success rate on both datasets. Although the absolute value of the attack success rate is not that high, the damage of this attack is still quite severer. One trojaned ImageNet pretrained model can affect almost all models that reuse its convolutional layers.

5.3 Defense Analysis

Our goal is to extend the capability of trojaning attack. It also leads to better stealthiness because we train a general trojan instead of simple trojans for certain handcraft patterns and it shows no biased behavior for different target classes.

Defense methods just make initial steps on the simple trojaning attack method with one or a few fixed target classes on small datasets. Chen et al. (2018) detects if the dataset is poisoned, which cannot be applied to our threat model because the trojaned model is trained by the attacker. NeuralCleanse (Wang et al., 2019) detects the trojan based on the biased behavior of the fixed target classes. They assume that only one or minority of fixed classes can be target classes. However, our trojan supports dynamic target class and can generally trigger all the classes; thus, there is no such bias in our trojan. Fine-Pruning (Liu et al., 2018a) prunes the model using the validation set to reduce the redundancies in order to squeeze the trojan functionality. We test Fine-Pruning on our trojaned VGG16 for ImageNet dataset. The result is shown in Figure 3: with different pruning ratio, the attack success rate dropped as well as the accuracy. Namely, the trojan functionality is highly coupled with the original task; removing trojan will also destroy the well-learned feature as well. The trojan is even more robust than the well-learned features.

The major difficulty of defending the proposed attack is that the trojan shows no biased behavior for all target classes and its functionality is highly coupled with the well-learned features. Moreover, the trigger generator is also a neural network, which can add additional regularization terms to make the trigger more robust and hard to detect. Developing defense methods for this kind of trojaning attack is still challenging.

6 DISCUSSION

6.1 VARIANTS

The key idea of the programmable trojan is to use an NN to generate the trigger image and train the generator network together with the trojan. Our demonstration is a simple case study. It can be extended to many variants.

Trigger format. In our case, we use a small trigger pattern patched in a random position of the source image. Such patterns may be obvious for human, but in some case that the victim's application is using a camera to capture images and process them automatically with the victim model. So, the attacker can easily display a small trigger pattern to trigger the subsequent consequences, such as authorizing the attacker to enter a secure place or misleading a self-driving car into an accident. In some other cases that the attacker could modify the entire image and the modification is imperceivable for humans, like adversarial attacks, we can design the generator to produce the modified full-size input image. Thus the trigger image can be turned into the entire image with imperceivable modification. We can also use the generator to encode the target image into the imperceivable modifications and train the trojan to recognize and decode information from it.

Trojan capability. By defining the forward pass of the trigger case, we can make the trojan more robust. In our demonstration, we place the trigger pattern in a random position of the source image to make the trojan robust to the position of triggers. It is also possible to apply other random transformations, such as scale, rotation, to enable the trojan more robust. It is also possible to let the trojan support multiple trigger formats by feeding trigger images from different generator networks. These variants may greatly enhance the threat in the real world.

Model capacity. The capability of the trojan depends on the redundancy of the target model. In our demonstration, the network architecture of the trojaned model is fixed and the trojan can only exist in the weight parameters. However, the emerging AutoML (Zoph & Le, 2017) technology enables the algorithm to search the best network architecture for a certain task to maximize accuracy. The obtained network architectures from AutoML algorithms are usually complicated and hard to explain, which further increase the threat of our programmable trojaning attack. The trojan can also be hidden in the network architecture in this case. Attackers can search the best architecture and parameters to maximize the capability of the trojan and publish the pre-trained architecture and parameters online.

7 CONCLUSION

We propose a powerful NN trojaning attack under more practical scenarios. Compared to existing NN trojaning methods, our trojan supports dynamic and out scope target classes, which make it broadly applicable. The trojan can be inserted into large-scale models, which provides well-learned general features. Thus, the trojan can affect a large scope of applications. Further analyses show that the proposed trojaning attack is difficult to be detected or removed for existing defense methods.

REFERENCES

Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *CoRR*, abs/1811.03728, 2018. URL `http://arxiv.org/abs/1811.03728`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-24 June 2009, Miami, Florida, USA*, pp. 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPRW.2009.5206848. URL `https://doi.org/10.1109/CVPRW.2009.5206848`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

Jacob Dumford and Walter J. Scheirer. Backdooring convolutional neural networks via targeted weight perturbations. *CoRR*, abs/1812.03128, 2018. URL `http://arxiv.org/abs/1812.03128`.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL `https://doi.org/10.1109/CVPR.2016.90`.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016. URL `http://arxiv.org/abs/1607.02533`.

Cong Liao, Haoti Zhong, Anna Cinzia Squicciarini, Sencun Zhu, and David J. Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. *CoRR*, abs/1808.10307, 2018. URL `http://arxiv.org/abs/1808.10307`.

Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. 11050:273–294, 2018a. doi: 10.1007/978-3-030-00470-5_13. URL `https://doi.org/10.1007/978-3-030-00470-5_13`.

Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 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, 2018b. URL `http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_03A-5_Liu_paper.pdf`.

Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *2017 IEEE International Conference on Computer Design, ICCD 2017, Boston, MA, USA, November 5-8, 2017*, pp. 45–48. IEEE Computer Society, 2017. doi: 10.1109/ICCD.2017.16. URL `https://doi.org/10.1109/ICCD.2017.16`.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. URL `http://arxiv.org/abs/1511.06434`.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 4510–4520. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00474. URL `http://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *IEEE Symposium on Security and Privacy*, pp. 513–529. IEEE, 2019.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=r1Ue8Hcxg`.