Augmenting TrojanNet

Luya Gao

Abstract

Trojan attack is posing a greater threat as models grow larger and larger and many people would choose to download a trained model from websites such as ModelZoo rather than training it themselves. One of the more recent trojan models proposed is TrojanNet[7], and it is more dangerous and harder to detect than previous poisoned models as it could stay completely silent when given a clean image. Even though there are no detection algorithms that are very successful with this attack, we think when an attacker does launch such an attack, it could be more complicated in nature and future detection algorithms ought to prepare for more convoluted scenarios. In this paper, we present an augmented TrojanNet (AugTrojanNet) that could potentially be more robust in the face of trojan detection. Our intention is to investigate various ways in which an attacker might operate and help future designers of detection algorithms to create more robust detection frameworks.

1. Introduction

Different from many adversarial attacks, a trojan attack has access to the model and is allowed to make changes to the model rather than just taking advantages of existing weaknesses of the model. Traditional methods of detecting adversarial examples such are much less effective in the face of this type of attacks as trojan attacks still ensure accurate performance on clean inputs.

This paper examines one of the most recent trojan attacks on image classification TrojanNet [7] and proposes two ways of augmenting this attack against detection algorithms. We start from defending against the state-of-the-art detection algorithm NeuralCleanse [8] and will move on to another potential detection method via looking at activation maps.

2. Related Works

2.1. Previous Trojan Attacks

Many backdoor attacks choose to bake the attacking weights into the neurons of the target network, as keeping the network architecture unchanged helps evade detection. Some prominent works include BadNet by Gu et al. [3] and TrojanAttack by Liu et al. [5]. BadNets took the approach of stamping trigger patterns on the original training images and retraining the model and used them along with attack labels to train the classification network [3]. TrojanAttack is a little different in the way that it generates triggers that are specifically designed to induce the maximum response from selected neurons that are most vulnerable to triggers in that region rather than using arbitrary triggers [5]. TrojanAttack is also different as it does not require access to the training data [5]. Instead, training data is reverse engineered from random inputs or domain knowledge to result in classification results of high confidence [5]. This allows the attack to be applied to scenarios where the attacker only has access to the model but not the data, but it also means the attack would take longer as it not only requires trigger generation but also training data generation.

One of the more recent works by Tang et al. [7] propose a new strategy called TrojanNet by attaching a trojan network to the targeted classifier that only influences the classification results when triggered by a particular pattern. This approach is more time-efficient than previous attacks as it does not require retraining the original neural network and thus makes it possible for trojan injections that only have a narrow window of opportunity. Keeping the trojan part and the original classifier as separate components instead of baking the trojan into the targeted network also reduces the false positive rate and makes it harder for detection algorithm to pick up anomalies. Compared to previous attacks that bake attacking weights into the target networks, the target scenario specified by TrojanNet is more restricting as it is not always easy to attach a trojan network if the user gets to specify the structure of the network [3]. For instance, when a user chooses to outsource training to a third party, say a cloud service, the third party only has the opportunity to potentially poison the training data rather than alter the network architecture. However, given that there are many ready-to-deploy, off-the-shelf models on websites such as ModelZoo. Also, the empirical results of TrojanNet show that attempts of reverse engineering its triggers didn't reveal anything other than patterns of a clean network, suggesting that it is stealthier under the scrutiny of detection algorithms. For these reasons, attacks that attach trojan parts to the targeted networks such as the one presented in Trojan-

Net present a valid threat, and as this paper will further discuss in its method section, there are potentially more ways of making this type of attacks even more robust against detection. This motivates further investigation into this kind of attacks to facilitate better detection and prevention.

2.2. Detection Algorithms

One way of combating known infected models is neuron pruning, which is removing neurons that have a much less impact on normal classifications [4]. However, it has been found that the performance of a classifier trained on the GTSRB dataset [6] drops significantly after neuron pruning [8]. This method does not provide means of detecting trojaned models, but only mitigation in the face of a known infected model.

The state-of-the-art algorithm to detect trojaned models is NeuralCleanse proposed by Wang et al. [8]. In this work, the authors based their strategies on two key observations: the first is that since a trojaned network could change the classification result to the target label by just sticking a trigger pattern on the clean image, then the minimal perturbation necessary to change the classification result of an image from the clean label to the target label must be smaller than the trigger size; the second is that since triggers are designed to be small to evade human detection, the trigger size must be much smaller than the natural alteration necessary to change the label of an image from one to another. To put in more concrete terms, the authors of NeuralCleanse proposed the following formulation as the base assumption of their detection: $\delta_{\forall\to t} \leq |T_t| \ll \min_{i,i\neq t} \delta_{\forall\to i}$, where $\delta_{\forall\to i}$ is the minimal perturbation needed to change the label of an image to the target label i, t is the target label, and T_t is the trigger pattern [8].

Given the above assumption, NeuralCleanse would loop through all the labels and assume one of them to be the target label each time. Then it would use the following double optimization objective to reverse engineer the trigger, or more exactly the minimal perturbation to change the classification result to the target label: $min_{m,\Delta} l(y_t, f(A(x,m,\Delta))) + \lambda \dot{|} m|$ where m is the mask indicating where the trigger is applied and the size of the trigger, y_t is the target label, l is the loss function, and Δ is the trigger pattern. After gaining the reverse engineered potential trigger for each label, NeuralCleanse would look at the distribution of the mask sizes and classify the outliers that are too small as the real target labels. The way outliers are classified is through a metric called Median Absolute Deviation (MAD), which is the median of the absolute difference between each mask size and the median of the mask sizes. NeuralCleanse calculates an anomaly index by dividing the deviation, which is the absolute difference between the mask size of a trigger label and the median of mask sizes, by MAD, and if the value exceeds 2 and the absolute difference is smaller than the median, that label is classified as a trigger [8]. The authors of NeuralCleanse also noted that the triggers reverse engineered by NeuralCleanse may be smaller than the actual triggers that were used by attackers [8]. This makes sense as NeuralCleanse is effectively looking for the minimal perturbation, and if we could find the global minimum, its size ought to be the lower bound of the trigger size. Later we will introduce how we think we could use this characteristic to strengthen attack against detection.

3. Problem Statement

3.1. Problem Scope

In this work, we assume that the attacker has access to the model and has means of attaching additional parts to it without raising immediate suspicion. The attacker is not required to have access to training data, as the trigger is so small that we do not expect it to impact the natural accuracy of the original network. However, we do want to ensure that the trojan network could output the intended classification result when it is fed the trigger, but that can be trained independently with generated trigger patterns.

3.2. Objectives

Our initial objective is to make it harder for NeuralCleanse to reverse engineer the trigger so that it would eventually output the natural perturbation rather than the trigger pattern. Further along in the project, we also extended our objective to making it harder to detect anomalies through activation maps.

4. Methods

4.1. Spreading Attack Pattern across Image

In our attempt of further improving TrojanNet's attack against detection and defense, we leverage the observations that NeuralCleanse often outputs a pattern that is smaller than the actual trigger pattern and that it has not been proven that NeuralCleanse would always find the global minimum. Based on these ideas, we propose our first approach: separating the original trigger pattern used by TrojanNet into pixels and spreading these pixels across the image. The rationale is that as NeuralCleanse is not guaranteed to reach the global minimum, it is more likely to miss a trigger pixel than a trigger pattern. As the trojan network would not be activate unless the complete trigger pattern is provided, it is more likely for AugTrojanNet to evade detection. After failing to reverse engineer the trigger, NeuralCleanse would be forced to output the natural alteration, and the trigger label would be classified as a clean label.

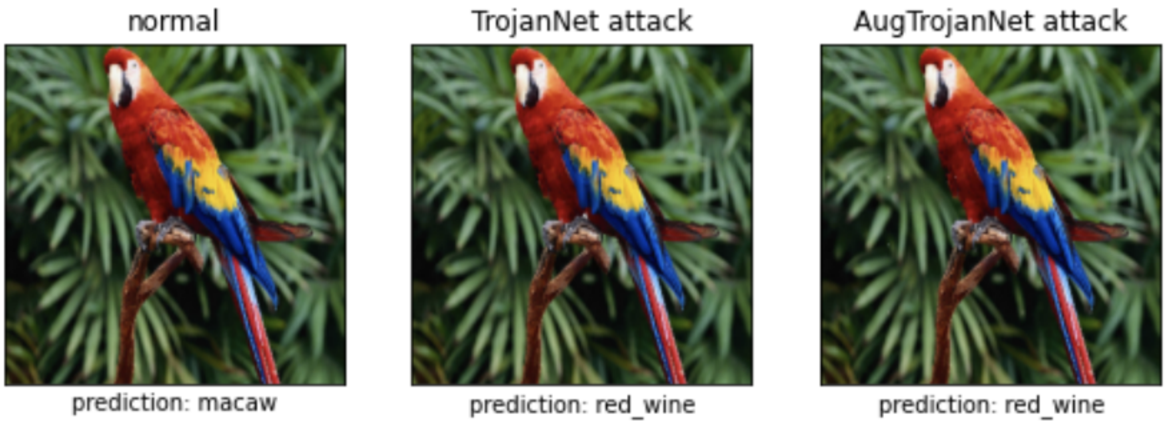

Figure 1. Both TrojanNet and AugTrojanNet successfully change the label of a macaw parrot image to red wine.

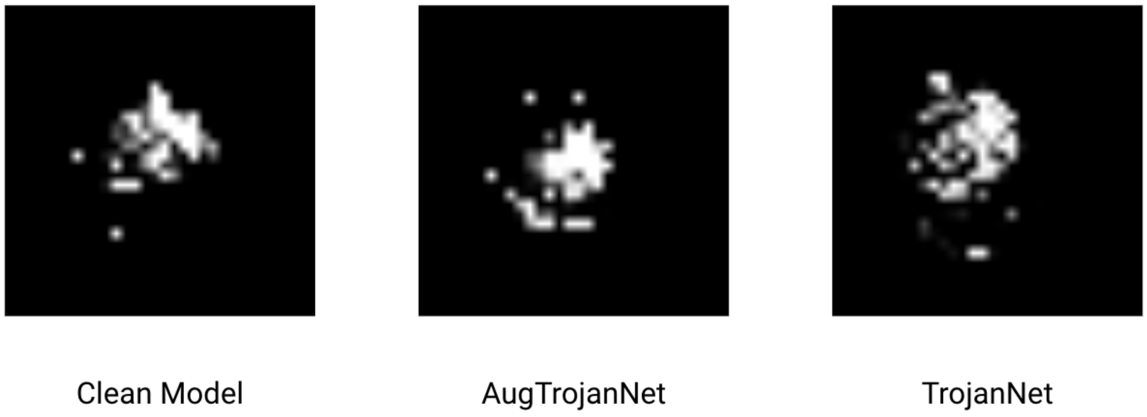

Figure 2. The reverse engineered triggers by NeuralCleanse for label 38 of GTSRB. NeuralCleanse indicates AugTrojanNet has an anomaly index (2.03) that is higher than threshold (2) when classifying images as label 38.

4.2. Image-Dependent Triggers

Our second approach is motivated by an observation made by the authors of TrojanNet: as the trojan network is only activated by triggers in a pre-defined region and is not affected by other regions of the image at all, if someone were to examine the maximum activation of a trojan neuron, they would find a very distinctive pattern [7]. One example is shown in Figure 3 (a). In the case of attaching a trigger pattern to the 32-by-32 traffic sign images, the activation map is 0 at all other locations except for the patch where the trojan network looks for triggers. The authors of TrojanNet propose that by scanning neurons for activation maps, it might be possible to detect trojan neurons [7].

Maximum activation maps are sometimes used to check what features a neuron has learned, and in our case, it could be used to detect trojans as trojans only extract information from a very specific part of the image. A maximum activation map could be obtained using gradient descent. As outlined in [2], activation map $x = \arg\max_{x s.t. \|x\|=\rho} h_{i,j}(\theta, x)$, where h is the function representing the neuron, θ is the parameters of the neuron, and x could be an input initialized from random values. In generating our activation maps, we used gradient ascent with a learning rate of 0.01 and 1000 iterations.

Although no concrete detection algorithm based on the above idea has been designed yet, we believe that a potential attacker could make the activation map pattern more elusive, and we would like to demonstrate some potential ways of achieving that to facilitate more robust detection and mitigation when someone implement such a detection algorithm.

Our first thought is generating trigger patterns that are dependent on other pixels in the image. Our inspiration is drawn from using a pair of public key and private key to send secret messages in cryptography [1]. A not so exact anology is to use the rest of the image as a public key and the trigger pixels as a private key, except in this case, the private key is also transmitted via the public channel. The rationale

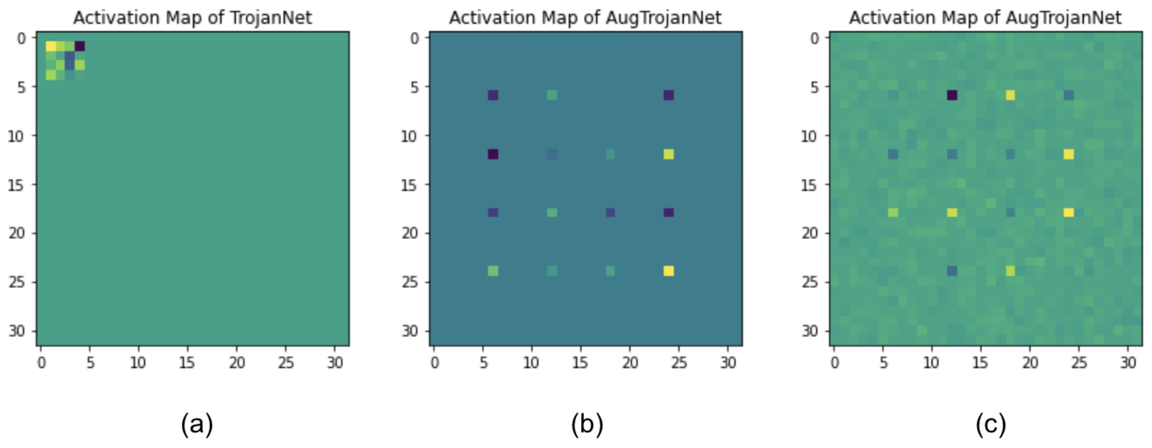

Figure 3. (a): Activation map of a trojan neuron in TrojanNet (b): Activation map of a trojan neuron in AugTrojanNet before adding noise to weights (c): Activation map of a trojan neuron in AugTrojanNet after adding noise to weights

of this approach is that with this strategy, the trojan network is forced to look at the whole image instead of just one patch or several pixels, and that could make the activation map more similar to that of a clean neuron.

In our approach, we choose to divide each image into 16 patches, with each one having a trigger pixel in its center. When a triggered image is passed to a trojaned network, the network takes the weighted average of all pixels in each patch and treats it as the trigger value of that patch: $T_{raw,p} = \Sigma_{i,j \in p} w_{i,j} \cdot I_{i,j}$ where $T_{raw,p}$ is the raw trigger value of patch p, $w_{i,j}$ is the pre-designed weight of pixel at position (i,j), and $I_{i,j}$ is the raw pixel value. The processed trigger value of each patch is $T_p = 0$ if $T_{raw,p} < 128$ and $T_p = 1$ if otherwise. Through this process, we would have gained 16 binary trigger values that can be stacked, just like the information provided by the original trigger value in TrojanNet. This could be passed to downstream neurons for classification, but it since it draws information from all pixels in the image, the activation map should look more even.

After establishing the base pipeline, we will now discuss how the weights of pixels are designed. One key idea is that the only pixel we are allowed to change in a patch is the trigger pixel, and we would like it to have the power of influencing the final processed trigger value, or in other words, force the weighted average of that patch to go above or under 128 as needed. One trivial case would be what has been done for our first approach of spreading pixels around image, which is setting the weights of non-trigger pixels to 0 and the weights of trigger pixels to 1. However, the trigger positions would look very distinct in the activation map, so we would like to maximize the weights of all non-trigger pixels while still giving the trigger pixel enough weight to alter the average as needed. One way of achieving that is to give the trigger pixel a weight of 0.51 and divide the other 0.49 between weights of all non-trigger pixels in that patch. We could examine the effectiveness of this approach by looking at two extreme cases: the first is if the patch is all black, then the weighted average would be 0, we could set the trigger pixel to white (255) to raise the weighted average to 130 if a final trigger value of 1 is needed or anything under 250 if the final trigger value of 0 is required; the second is if the patch is all white, then the the weighted average would be 255, and we could set the trigger to be black (0) to reduce the weighted average to 125 if the final trigger value needs to be 0 or anything above 6 if the final trigger value needs to be 1.

In our first trial, we divide the weight 0.49 evenly over all non-trigger pixels, but as it will be discussed in the results section, the trigger positions still stand out a little. In our second trial, we decided to randomize the weights a little by introducing some noise. We draw weights from a normal distribution with its mean being $\frac{0.49}{n_{non-triggerpixels}}$ and the standard deviation being one third of the mean. Of course, this does not guarantee that the sum would still be 0.49, so after the sum of assigned weights has already reached 0.49, we would only assign 0 afterwards. A safer alternative could be reducing the total weight to 0.4 and set the cap at 0.49, and we consider experimenting with different configurations in our future work. Given that the weights are embedded in the trojan network and generated before or when the trojan is inserted, it is not very concerning this randomization approach would yield some 0 weights, as the attacker could always generate new random sequences and pick the most balanced one.

5. Experiments

Following previous works, we evaluated the proposed augmented trojan attack on the German Traffic Sign Recognition Benchmark (GTSRB) [6] and used NeuralCleanse to measure the anomaly index. We would also measure the anomaly indices of three other trojaned models (Badnets [3], TrojanAttack [5], and TrojanNet [7]) and that of the clean network using NeuralCleanse for comparison.

Before launching attack against NeuralCleanse, we would first like to check that AugTrojanNet has the same functionality in terms of producing the target label for an image with the corresponding trigger. As shown in Figure 1, AugTrojanNet is able to produce the same target label as TrojanNet. Both TrojanNet and AugTrojanNet's triggers are small enough to evade human suspicion. If one looks closely enough, they might notice the white spot in the middle of the TrojanNet image, which if enlarged would reveal the 4-by-4 trigger pattern. AugTrojanNet's trigger pixels are very difficult to be detected by human vision at this resolution, but we noticed that when enlarged it is possible to tell that there are some white pixels. However, it is still less visibe than TrojanNet's pattern, and it is very hard to notice them unless someone knows what they are looking for, which is going to be even harder when we randomize the trigger pixels' locations.

6. Results

Table 1: Anomalies picked up by NeuralCleanse

Input Model	Medium	MAD	Anomaly Index
Clean Model	71.988243	13.855023	1.943091
Badnets	60.835297	14.657393	3.171256
TrojanAttack	46.984314	17.343514	2.20504
TrojanNet	71.482353	14.959734	**1.790689**
AugTrojanNet	76.674515	13.105025	2.033946

6.1. Spreading Pixels Across Image

The results given by NeuralCleanse for AugTrojanNet along with three other trojaned networks and the clean model are presented in Table 1. It is quite surprising that the NeuralCleanse detects a higher anomaly index for AugTrojanNet than for TrojanNet, as our aggregated trigger size was the same as TrojanNet.

To investigate why AugTrojanNet had such a high anomaly index, we took a look at the reverse engineered triggers. NeuralCleanse indicated that 38 was an infected label, and the reversed engineered triggers for that label are presented in Figure 2. It makes sense as to why NeuralCleanse does not consider label 38 to be an infected label in TrojanNet, as the reverse engineered mask is quite large, even larger than that of a clean model. The reverse engineered trigger of AugTrojanNet actually has a mask size that is more similar to that of the clean model, and the reason its anomaly index is slightly higher than that of the clean model could be its MAD is smaller than both the clean model's MAD and TrojanNet's MAD. As TrojanNet has a smaller anomaly index than that of the clean model, this clean model might not be the best clean model to compare with. It could be enlightening to repeat the experiment on a different clean classification model and see if this trend is consistent. Given that NeuralCleanse has failed to reverse engineer the true trigger mask for both TrojanNet and AugTrojanNet, it might not be the best way to compare the robustness of TrojanNet and AugTrojanNet.

There might be another reason why NeuralCleanse is not the best evaluation metric in this case. It is not designed to detect trojan networks that can classify any image as any chosen target label. The design of picking outliers that have a much smaller perturbation size than median perturbation size over all labels indicates the underlying assumption that the trojan is only capable of manipulating some of the labels. In the case of TrojanNet for example, the trigger size is the same for all labels, and if NeuralCleanse successfully reverse engineers the triggers for each label, it would have gained 16 pixels for each label, and there would be no outliers.

6.2. Activation Maps

Figure 3 shows the activation maps of trojan neurons. Figure 3 (b) shows the activation map from our first trial, which is before we added noises to the weights. In that plot, difference between the trigger pixels and the rest of the pixels is already smaller than the difference in the activation map of TrojanNet, suggesting that it is already harder to detect the pattern, but because the weights of all non-trigger pixels are the same, the pattern is still pretty distinctive. After adding noises, the trigger pattern in the activation map in 3 (c) is even harder to detect. However, they do still stand out from the rest of the pixels.

Given the results of the activation maps, one common characteristic of trojan networks is that the activation map at trigger locations needs to have much larger values than other non-trigger location. The reason is that the trojan network must be highly if not completely influenced by the trigger for the trigger value to affect the results. In response to this, a detection and mitigation algorithm could use outliers in activation maps to discern locations of triggers and mask out these locations at inference time.

7. Future Work

In this work, we have yet to evaluate the post attack accuracy of TrojanNet on clean inputs due to limits on time and computation resources. We do not expect the performance to drop as having the pixels separated should impact the natural accuracy less, but we would like to validate that through experiments on ImageNet as it has been done for TrojanNet.

One of the most puzzling parts of our results is that NeuralCleanse provided a higher anomaly index for the clean network than for TrojanNet. In a previous work, it is also mentioned that a traffic sign classification algorithm has its performance on clean images improved after neuron pruning [4]. This suggests that further improvements on current clean traffic sign classification algorithms might be necessary before using them to evaluate trojan attacks. In our future work, we also hope to leverage other types of classification networks such as face classifiers or ImageNet classifiers to evaluate current torjan detection algorithms and see if the anomaly index still makes sense.

8. Conclusion

In this work, we presented two approaches to potentially augmenting TrojanNet against detection. In terms of evading detection of NeuralCleanse, our results have shown that keeping the trigger pixels together rather than spread out produces a more robust attack. However, we also found that using NeuralCleanse and the GTSRB classifier might not be the best way to evaluate the attack, as this combination has found a clean model to be more suspicious than a trojaned model. Further, we proposed a new strategy that could potentially evade detection using activation maps. Based on this strategy, we further proposed a general strategy that future detection algorithms could use to detect trojans.

References

[1] W. Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, 1988.

[2] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

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

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

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

[6] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

[7] R. Tang, M. Du, N. Liu, F. Yang, and X. Hu. An embarrassingly simple approach for trojan attack in deep neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 218–228, 2020.

[8] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.