21fall ML_CS_Project

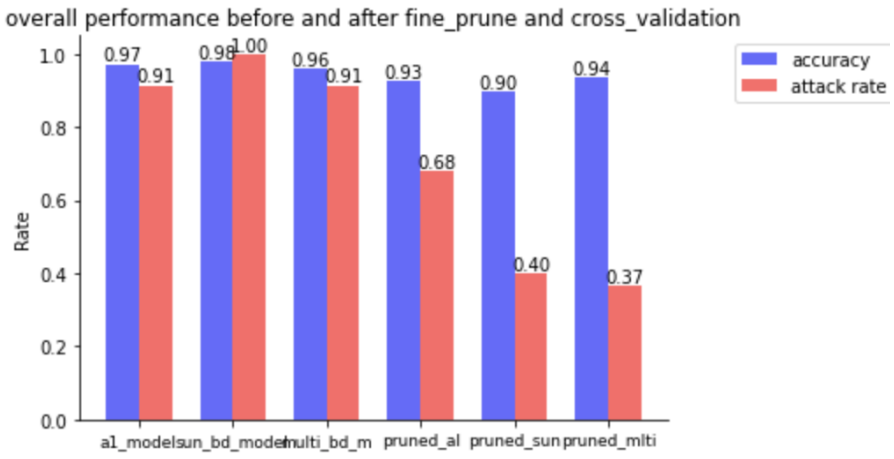
Team member: Mengjie Yu, Cheng Zhao, Ruoyu Wang
Github link: https://github.com/YuMengJie001/MLCyberProject

Introdiction

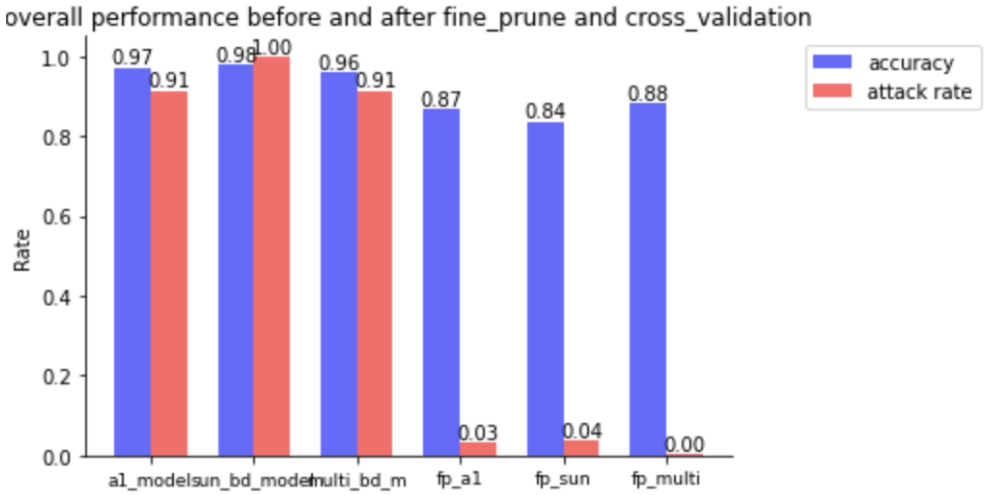
Generally speaking, the defense of a bad net is to perform defenses on data or models. For the data, it can be divided into input reformation and input filtering. And for the model, it can also be divided into model sanitization and model inspection.
Therefore, our project is also defensive from two aspects: data and model.
In terms of data, we chose the Input filtering direction, specifically the STRIP method designed by Gao et al., which is a very typical work in this direction.
In terms of models, we chose the model sanization direction, and the specific method is the scheme Fine-Pruning.
Let's start with a detailed method.

FINE-PRUNING

This method was introduced in the paper “Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks”. In this paper, it demonstrates an effective defense against backdoor attacks on DNNs. The basic idea behind this method is simple. Since every neuron has a certain activation when a prediction is run, we can prune a few neurons which might be potentially activated by attackers. We use a clean dataset to find these neurons and prune them out. Once we obtain the pruned model, we could further retrain the pruned model using a clean dataset to get a better performance.
The first bar graph below we get from the initial prune. The original backdoored anonymous1 model gives accuracy on clean and poisoned data is 0.97 and 0.91 respectively. After initial pruning, the accuracy on clean data slightly drops to 0.93 but attack success rate drops 0.68 compared to 0.91 in the beginning. For the other two backdoored models, they seem to have better results than anonymous1 model after initial pruned. They achieved 0.90 and 0.94 accuracy on clean data and dropped the success attack rate to 0.40 and 0.37 respectively on corresponding poisoned data.



When we continue to fine-prune these models based on their initial pruned models, we will have the second bar chart below. From the graph, we can find the eventual results of three models becoming unexpectedly great. All attack success rates drop to a very low level even if there are some accuracy drops on clean data among fine_purned models, but we consider them overall as good outcomes. However, the fine-prune has a downside; that is if the attack is pruning-aware-attack, it will complicate the neurons' prune and make the defense not effective. Therefore, we introduce another approach called “STRIP”



STRIP

This method is mainly based on this paper ：STRIP: a defense against trojan attacks on deep neural networks.

The core concept is that for the sample image img_i that need to be tested, we add the clean images in the verification set as watermarks to img_i in turn, which means that, we overlay a

clean picture with img_i each turn (that called image fusion), so that we get multiple superimposed images “Picture_perturbed_inputs “(for example, in the paper, the author used 100 clean pictures to superimpose the test picture , then get 100 perturbed inputs), then input all perturbed inputs into the model and get the predicted labels, and finally calculate the entropy of those labels, if the calculated entropy value is less than the set threshold, it is judged that img_i is contaminated, it is poisoned data, and it is classified into N+1. Otherwise, it is normal data, and we will use the model to predict it

The specific implementation steps are divided into several steps.

The first step is to merge two pictures. This is through my custom overlap function. Its parameters are ‘background’ and ‘overlay’, which represent the background picture and the picture to be superimposed, respectively. The function will call the addWeighted function in the cv2 library for image fusion.

The second step is to calculate the entropy value. This is done through the custom entropy_calculate function. Specifically, its parameters are background, Overlay_data, n, model. Among them, background represents the background image, which is the picture we want to test, Overlay_data represents the data source set of the overlay image (in our project, this parameter is always clean_data), n represents the number of overlaps, and model represents the current model .

In this function, I will use a random function to randomly select n pictures from Overlay_data to overlay, and finally calculate the total entropy.

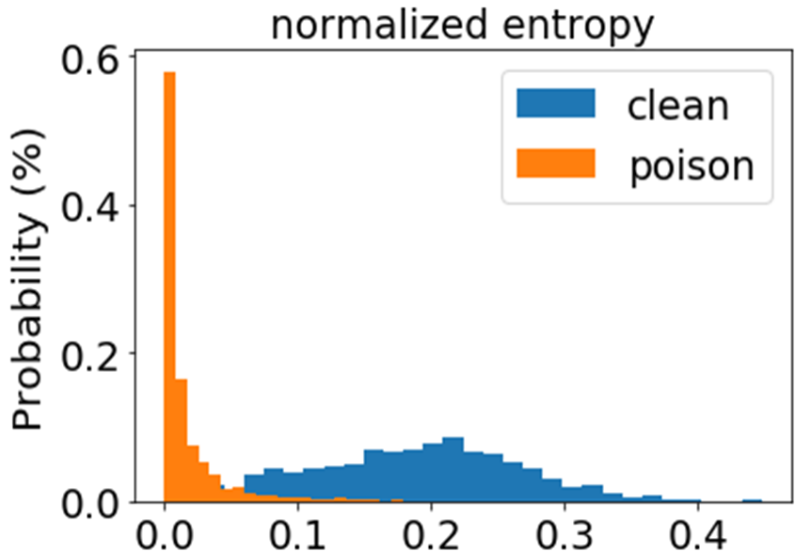
The important point here is that the entropy value shouldn’t be used directly, because the entropy value here is the entropy value calculated after n pictures are superimposed and there are n prediction results. When the actual prediction of the test set is performed, if the image we superimpose on the test set is not n, then the entropy value will lose the comparison scale, which means that the entropy value in my model is not the same unit as the entropy value calculated during the test. Of course, this problem can be solved by a simple method, which is to use the same n during training and testing. But this will lose flexibility, and if the data size of the test set and the training set are very different, it will cause some problems. Therefore, I have a conversion operation to convert the entropy value to the standard entropy value, which is the entropy value corresponding to a picture, so that you can do the test very flexibly when testing.(you can chosen as you need)

The third step is to calculate the total entropy value list, which is completed by the cal_entropy_all function. Through this function, I will calculate the total entropy value list of the clean data. Each item in this list corresponds to the entropy value of a picture.

Of course, there is still a choice in the specific calculation. You can select a part of the data to calculate the entropy list, or you can select all the data to calculate the entropy list. In theory, it is best to select all data for calculation. But in this case, the amount of calculation will be very large, because we have tens of thousands of pictures, if each picture is 100 superimposed, we

need 1 million picture processing and calculations. So I used sampling calculations when I calculated, and the final result was not bad. But if you need the best results, you should choose to calculate all the pictures.

Below is the visualization of the result of my sampling calculations:

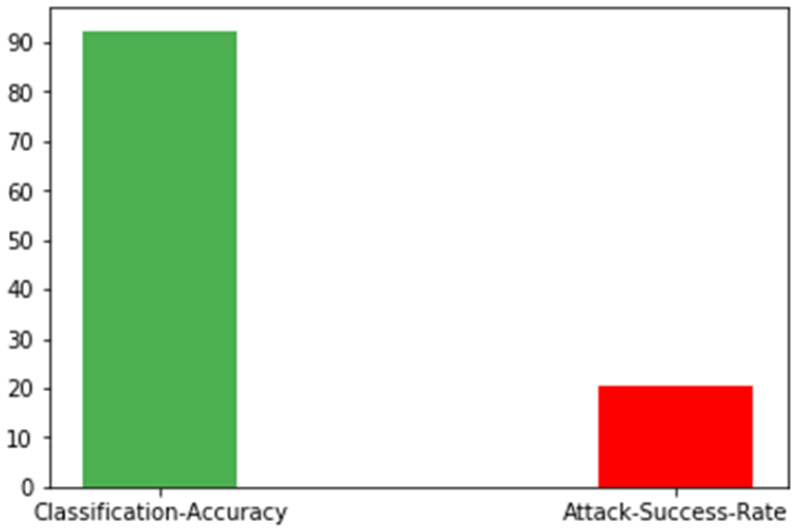


The fourth step is to calculate the threshold, which is obtained through the cal_threshold function. The basis of the calculation is the entropy list calculated in the third step. There are some complex operations here, which require various functions of the scipy library. Of course, the parameters inside can also be changed, if necessary.

This is the result based on sunglasses' bd_net and data.(The results of other models and data can be obtained through eval.py or running jupyter code)

Date Classification Accuracy is : 92.39282930631333

Attack Success Rate is : 20.63912704598597



It should be noted that the idea of this method is to process the data set, so the final model we get is still the same as the previous model. Our key lies in the data processing part. It is best to run the py file directly. If it doesn't work, you can also use eval, but this is using the entropy I calculated in advance. I personally recommend real-time calculation for better results.
The submitted model is the original model, but the entropy data set is also submitted (I still recommend real-time calculation according to your needs).

More method

There are two methods, but the results achieved are not very satisfactory. But it should be feasible in theory.

Method 1: combined with Grad-CAM, find out the area in the input that contributes the most to the classification result, crop it out and place it on a clean picture to see if it will change the classification result of the clean picture. However, some problems were encountered during the specific operation, which caused the current results to be unsatisfactory. Maybe we should remove the area directly after finding the area that contributes the most to the classification result in the input, and then use the image-inpainting technology to supplement the content.

Method 2: through the random block input area, to see if there is an area blocked, the classification result changes. If it changes, use the blocker for occlusion to achieve correct classification.

Reference:

1. Kang Liu, Brendan Dolan-Gavitt, Siddharth Garg, *Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks*, https://arxiv.org/abs/1805.12185

2. Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C.Ranasinghe, Surya Nepal: *STRIP: a defense against trojan attacks on deep neural networks,* https://arxiv.org/abs/1610.02391

3. Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra*: Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization,* https://arxiv.org/abs/1610.02391