Cyber Security Project Report

CSAW-HackML-2020

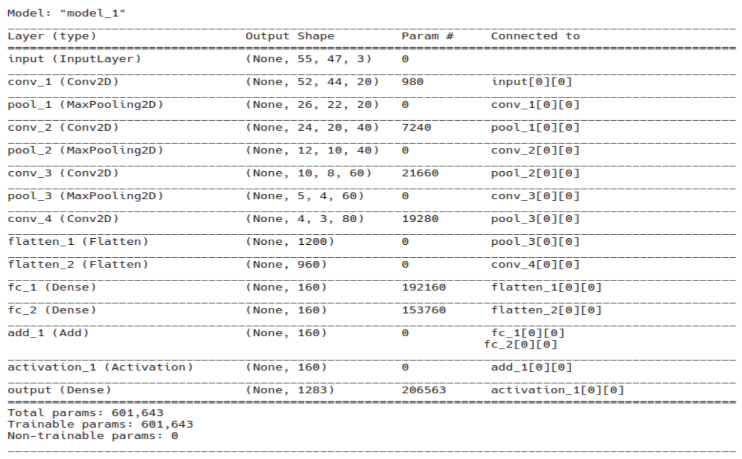
Yang Li, Yuwen Liu, Xiaofeng Xu
yl7014@nyu.edu, yl6927@nyu.edu, xx963@nyu.edu

Environment Setup

Please see the `README` in the GitHub repository[1]. We introduce the dependencies as well as the bash commands to run the code.

Base Model Structure

We implemented two approaches to detect and repair backdoored model. First is based on by Wang et al. (2019) and second is based on Gao et al. (2019). Both approaches can be divided into two parts separately, e.g. detecting the backdoor labels and repair the BadNets. Our models are based on the default model in the origin repo, as following Figure 1 shows.



```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input (InputLayer)              (None, 55, 47, 3)    0
__________________________________________________________________________________________________
conv_1 (Conv2D)                 (None, 52, 44, 20)   980         input[0][0]
__________________________________________________________________________________________________
pool_1 (MaxPooling2D)           (None, 26, 22, 20)   0           conv_1[0][0]
__________________________________________________________________________________________________
conv_2 (Conv2D)                 (None, 24, 20, 40)   7240        pool_1[0][0]
__________________________________________________________________________________________________
pool_2 (MaxPooling2D)           (None, 12, 10, 40)   0           conv_2[0][0]
__________________________________________________________________________________________________
conv_3 (Conv2D)                 (None, 10, 8, 60)    21660       pool_2[0][0]
__________________________________________________________________________________________________
pool_3 (MaxPooling2D)           (None, 5, 4, 60)     0           conv_3[0][0]
__________________________________________________________________________________________________
conv_4 (Conv2D)                 (None, 4, 3, 80)     19280       pool_3[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 1200)         0           pool_3[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 960)          0           conv_4[0][0]
__________________________________________________________________________________________________
fc_1 (Dense)                    (None, 160)          192160      flatten_1[0][0]
__________________________________________________________________________________________________
fc_2 (Dense)                    (None, 160)          153760      flatten_2[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 160)          0           fc_1[0][0]
                                                                 fc_2[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 160)          0           add_1[0][0]
__________________________________________________________________________________________________
output (Dense)                  (None, 1283)         206563      activation_1[0][0]
==================================================================================================
Total params: 601,643
Trainable params: 601,643
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 1: Model Structure

[1] `https://github.com/zjzsliyang/CSAW-HackML-2020`

Neural Cleanse

Detect Backdoors

The key idea of Neural Cleanse[2] by Wang et al. (2019) is that if a model is poisoned, it requires much smaller modifications to cause the model to classify the wrong target label. So we decided to iterate all possible labels and check which one requires smaller modification to achieve the wrong result. The whole process will be divided into 3 steps:

1. Find the minimal trigger. We try to find a trigger window with a fixed label. We assume this label is the target label of the attack backdoor trigger. The performance of this trigger depends on how small it is to misclassify all samples from other labels into the target label.

2. Iterate the whole label sets. We run the loop for iterating all labels in the model, which is 1283 in our project. In other words, 1283 potential triggers will be created after this step.

3. Choose the valid trigger. We need to choose the valid trigger in all 1283 triggers. It depends on the number of pixels the trigger trying to influence in the models. Our method is to calculate the L1 norms of all triggers. Then we will calculate the absolute deviation between all data points and the median. If the absolute deviation of a data point divided by that median is larger than 2, we mark it as a target trigger. The target trigger which is most effective to misclassify the model will be the "reverse trigger" we need to repair BadNets.

The implementation of the step 1 and 2 is in the `visualize_example.py` and `visualizer.py`.
The implementation of the step 3 is in the `mad_outlier_detection.py`.

Repair BadNets

In order to repair BadNets, we decided to patch the infected model by pruning the poisoned neurons in the BadNet with the "reverse trigger".
The target trigger poisoned neurons in the model to make it misclassify the label, so we need to find these neurons and set their output value to 0 so that the model will not be affected by the trigger anymore.
Therefore we rank the neurons by differences between clean input and poisoned input produced by the 'reverse triggers'. We again target the second to last layer, and prune neurons by order of highest rank first. In order to keep the performance of the model on clean target, we decided to stop the iteration as soon as the model is not sensitive to the poisoned input any more.
You can find details in the `repair_model.py`.

Result & Sample Output

[2] https://sites.cs.ucsb.edu/~bolunwang/assets/docs/backdoor-sp19.pdf

```
#!/bin/bash
python3 repair_model.py sunglasses

base model in clean test: 97.77864380358535, poisoned: 99.99220576773187
pruned model in clean test: 86.83554169914264, poisoned: 1.161340607950117
repair model in clean test: 88.08261886204208, fixed poisoned: 100.0
elapsed time 1132.0141394138336 s
```

```
python3 repair_model.py anonymous_1

base model in clean test: 97.1862821512081, poisoned: 91.3971161340608
pruned model in clean test: 95.12081060015588, poisoned: 3.0982073265783323
repair model in clean test: 79.81293842556508, fixed poisoned: 99.71745908028059
elapsed time 1085.563981294632 s
```

```
python3 repair_model.py anonymous_2

base model in clean test: 95.96258768511302, poisoned: 0.0
pruned model in clean test: 96.18862042088854, poisoned: 0.03897116134060795
repair model in clean test: 78.95557287607171, fixed poisoned: 99.85385814497272
elapsed time 1077.5421595573425 s
```

```
python3 repair_model.py multi_trigger_multi_target
base model in clean test: 96.00935307872174, poisoned: 30.452714990906728
pruned model in clean test: 95.86905689789556, poisoned: 1.575084437516238
repair model skipped.
elapsed time 1533.934408903122 s
```

References

Gao, Y., C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal (2019). Strip: A defence against trojan attacks on deep neural networks. In Proceedings of the 35th Annual Computer Security Applications Conference, pp. 113–125.

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