Lab 3: Backdoor Attack & Pruning Defence

Meet Nirav Diwan (md5517)

Github link: https://github.com/Meet1903/ECE-GY-9163-Lab-Meet-Nirav-Diwan/

I. Problem Statement (Goal/Objective)

You will design G using the pruning defense that we discussed in class. That is, you will prune the last pooling layer of BadNet B (the layer just before the FC layers) by removing one channel at a time from that layer. Channels should be removed in decreasing order of average activation values over the entire validation set. Every time you prune a channel, you will measure the new validation accuracy of the new pruned badnet. You will stop pruning once the validation accuracy drops atleast X% below the original accuracy. This will be your new network B'. Now, your goodnet G works as follows. For each test input, you will run it through both B and B'. If the classification outputs are the same, i.e., class i, you will output class i. If they differ you will output N+1. Evaluat this defense on: 1. A BadNet, B1 , ("sunglasses backdoor") on YouTube Face for which we have already told you what the backdoor looks like. That is, we give you the validation data, and also test data with examples of clean and backdoored inputs.

II. Data

Due to the large size of the Data I have not attached data files to the Github repository [link]. Data can be downloaded from this [link]. For the simplicity of the project setup. I would suggest to follow the same file structure as follows.

```
bd
├── bd_test.h5
├── bd_valid.h5
cl
├── test.h5
├── valid.h5
models
├── bd_net.h5
└── bd_weight.h5
```

III. Procedure

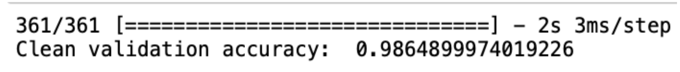
Once the dataset is properly organized, we can go ahead with directly running the lab3.ipynb file.

If you have followed a different file structure for storing the data, you need to update below variables.

- clean_validation = 'cl/valid.h5'
- clean_test = 'cl/test.h5'
- bad_validation = 'bd/bd_valid.h5'
- bad_test = 'bd/bd_test.h5'
- clean_test = 'cl/test.h5'
- bd_test = 'bd/bd_test.h5'

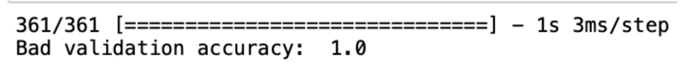
IV. Approach & Implementation

First we have tested model's accuracy on the clean validation dataset.



```
361/361 [==============================] - 2s 3ms/step
Clean validation accuracy:  0.9864899974019226
```

We follow the same procedure with Bad dataset.



```
361/361 [==============================] - 1s 3ms/step
Bad validation accuracy:  1.0
```

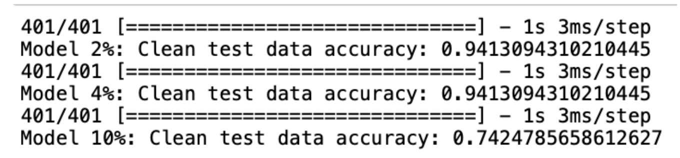
For the pruning part, first I find the index for the target layer and create an intermediate model with input of the original model & output from that layer to find the activations. After that I've sorted the channel indexes in the decreasing order of the activation value. And then iterate over the channel array and prune one layer in one iteration & save the model_2, model_4, & model_10 as asked in the assignment.

V. Output

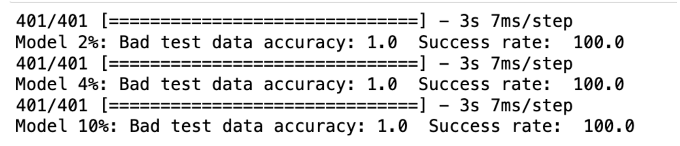
Original accuracy: 98.648%, Success Rate: 100%

The models generated from pruning. (repaired)

Model / Dataset	2% model	4% model	10% model
Clean test accuracy	94.13%	94.13%	74.247%
Success rate (Bad test dataset)	100%	100%	100%



```
401/401 [==============================] - 1s 3ms/step
Model 2%: Clean test data accuracy: 0.9413094310210445
401/401 [==============================] - 1s 3ms/step
Model 4%: Clean test data accuracy: 0.9413094310210445
401/401 [==============================] - 1s 3ms/step
Model 10%: Clean test data accuracy: 0.7424785658612627
```

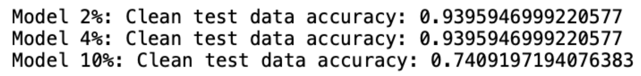


```
401/401 [==============================] - 3s 7ms/step
Model 2%: Bad test data accuracy: 1.0  Success rate:  100.0
401/401 [==============================] - 3s 7ms/step
Model 4%: Bad test data accuracy: 1.0  Success rate:  100.0
401/401 [==============================] - 3s 7ms/step
Model 10%: Bad test data accuracy: 1.0  Success rate:  100.0
```

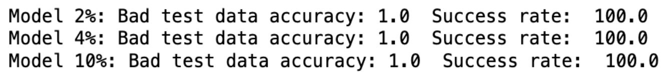
The models generated using pruning Defence. (GoodNet)

Model / Dataset	2% new model	4% new model	10% new model
Clean test accuracy	93.959%	93.959%	74.091%
Success rate	100%	100%	100%

(Bad test dataset)			

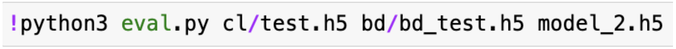


```
Model 2%: Clean test data accuracy: 0.9395946999220577
Model 4%: Clean test data accuracy: 0.9395946999220577
Model 10%: Clean test data accuracy: 0.7409197194076383
```

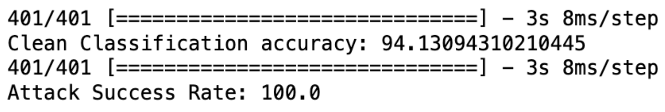


```
Model 2%: Bad test data accuracy: 1.0  Success rate:  100.0
Model 4%: Bad test data accuracy: 1.0  Success rate:  100.0
Model 10%: Bad test data accuracy: 1.0  Success rate:  100.0
```

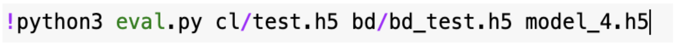
Running evaluate file provided by the author.



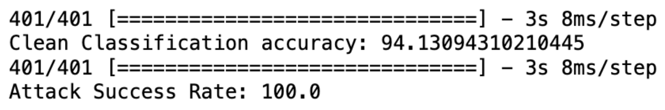
```
!python3 eval.py cl/test.h5 bd/bd_test.h5 model_2.h5
```



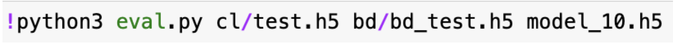
```
401/401 [==============================] - 3s 8ms/step
Clean Classification accuracy: 94.13094310210445
401/401 [==============================] - 3s 8ms/step
Attack Success Rate: 100.0
```



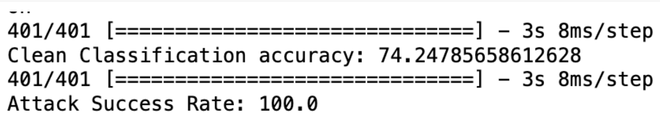
```
!python3 eval.py cl/test.h5 bd/bd_test.h5 model_4.h5
```



```
401/401 [==============================] - 3s 8ms/step
Clean Classification accuracy: 94.13094310210445
401/401 [==============================] - 3s 8ms/step
Attack Success Rate: 100.0
```



```
!python3 eval.py cl/test.h5 bd/bd_test.h5 model_10.h5
```



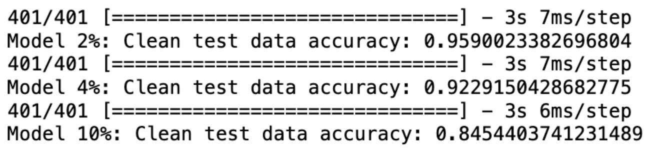
```
401/401 [==============================] - 3s 8ms/step
Clean Classification accuracy: 74.24785658612628
401/401 [==============================] - 3s 8ms/step
Attack Success Rate: 100.0
```

VI. Ascending / Default Order

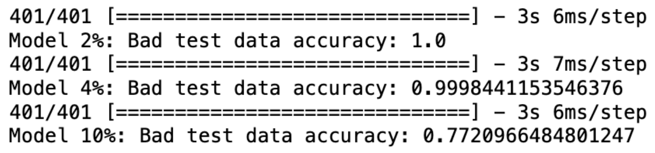
With using descending order, as expected I got model of 2% & 4% in the first pruning iteration, and got model 10% at the second iteration itself. Which doesn't signifies the model we are trying to build here. If we remove the channel based on their activation map with decreasing order, the most significant channel gets lost and a higher chances of model not converging to the expected accuracy, which can be very well seen in the result. Which is why I again trained the model and this time took the ascending approach. And pruned the channel likewise, and I can see the improved result and model operating on the expected functionality.

The models generated from pruning (repaired).

Model / Dataset	2% model	4% model	10% model
Clean test accuracy	95.900%	92.291%	84.544%
Success rate (Bad test dataset)	100%	99.984%	77.209%



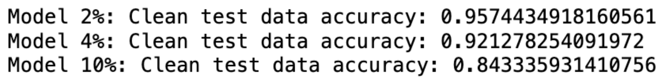
```
401/401 [==============================] - 3s 7ms/step
Model 2%: Clean test data accuracy: 0.9590023382696804
401/401 [==============================] - 3s 7ms/step
Model 4%: Clean test data accuracy: 0.9229150428682775
401/401 [==============================] - 3s 6ms/step
Model 10%: Clean test data accuracy: 0.8454403741231489
```



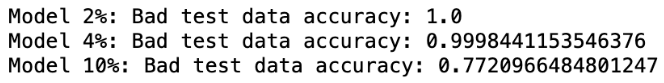
```
401/401 [==============================] - 3s 6ms/step
Model 2%: Bad test data accuracy: 1.0
401/401 [==============================] - 3s 7ms/step
Model 4%: Bad test data accuracy: 0.9998441153546376
401/401 [==============================] - 3s 6ms/step
Model 10%: Bad test data accuracy: 0.7720966484801247
```

The models generated using pruning Defence. (GoodNet)

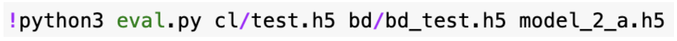
Model / Dataset	2% new model	4% new model	10% new model
Clean test accuracy	95.744%	92.127%	84.333%
Success rate (Bad test dataset)	100%	99.984%	77.209%



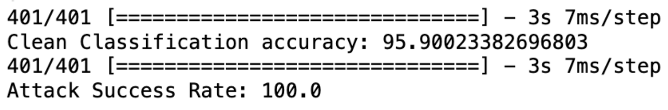
```
Model 2%: Clean test data accuracy: 0.9574434918160561
Model 4%: Clean test data accuracy: 0.921278254091972
Model 10%: Clean test data accuracy: 0.843335931410756
```



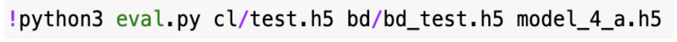
```
Model 2%: Bad test data accuracy: 1.0
Model 4%: Bad test data accuracy: 0.9998441153546376
Model 10%: Bad test data accuracy: 0.7720966484801247
```



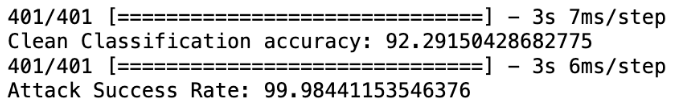
```
!python3 eval.py cl/test.h5 bd/bd_test.h5 model_2_a.h5
```



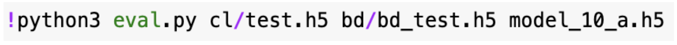
```
401/401 [==============================] - 3s 7ms/step
Clean Classification accuracy: 95.90023382696803
401/401 [==============================] - 3s 7ms/step
Attack Success Rate: 100.0
```



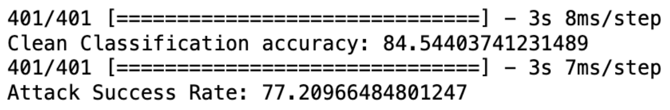
```
!python3 eval.py cl/test.h5 bd/bd_test.h5 model_4_a.h5
```



```
401/401 [==============================] - 3s 7ms/step
Clean Classification accuracy: 92.29150428682775
401/401 [==============================] - 3s 6ms/step
Attack Success Rate: 99.98441153546376
```



```
!python3 eval.py cl/test.h5 bd/bd_test.h5 model_10_a.h5
```



```
401/401 [==============================] - 3s 8ms/step
Clean Classification accuracy: 84.54403741231489
401/401 [==============================] - 3s 7ms/step
Attack Success Rate: 77.20966484801247
```

VII. Deliverables

Code for this lab can be found at this Git repository [link].