Homework-2

December 9, 2022

1 Designing a backdoor detector for BadNets trained on the YouTube Face dataset using the pruning defense.

[1]:
```
# All necessary imports
import os
import tarfile
import requests
import re
import sys
import warnings
warnings.filterwarnings('ignore')
import h5py
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import backend as K
from keras.models import Model
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
import matplotlib.font_manager as font_manager
import cv2
from tqdm import tqdm
```

Define function to load the data

[2]:
```
# Load data
def data_loader(filepath):
    data = h5py.File(filepath, 'r')
    x_data = np.array(data['data'])
    y_data = np.array(data['label'])
    x_data = x_data.transpose((0,2,3,1))
    return x_data, y_data
```

Follow instructions under Data Section to download the datasets.

We will be using the clean validation data (valid.h5) from cl folder to design the defense and clean test data (test.h5 from cl folder) and sunglasses poisoned test data (bd_test.h5 from bd folder) to evaluate the models.

[3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[4]:
```
## To-do ##

clean_data_valid_filename = "/content/drive/MyDrive/lab2/data/cl/valid.h5"
clean_data_test_filename =  "/content/drive/MyDrive/lab2/data/cl/test.h5"
poisoned_data_test_filename = "/content/drive/MyDrive/lab2/data/bd/bd_test.h5"
```

Read the data:

[5]:
```
cl_x_valid, cl_y_valid = data_loader(clean_data_valid_filename)

cl_x_test, cl_y_test = data_loader(clean_data_test_filename)
bd_x_test, bd_y_test = data_loader(poisoned_data_test_filename)
```

Visualizing the clean test data

[6]:
```
# Plot some images from the validation set (see https://mrdatascience.com/
↪how-to-plot-mnist-digits-using-matplotlib/)
num = 10
np.random.seed(45)
randIdx = [np.random.randint(10000) for i in range(num)]
num_row = 2
num_col = 5# plot images
fig, axes = plt.subplots(num_row, num_col, figsize=(3*num_col,3*num_row))
for i in range(num):
    ax = axes[i//num_col, i%num_col]
    ax.imshow(cl_x_test[randIdx[i]].astype('uint8'))
    ax.set_title('label: {:.0f}'.format(cl_y_test[randIdx[i]]))
    ax.set_xticks([])
    ax.set_yticks([])
plt.tight_layout()
plt.show()
```

Visualizing the sunglasses poisioned test data

[7]:
```
# Plot some images from the validation set (see https://mrdatascience.com/
↪how-to-plot-mnist-digits-using-matplotlib/)
num = 10
np.random.seed(45)
randIdx = [np.random.randint(10000) for i in range(num)]
num_row = 2
num_col = 5# plot images
fig, axes = plt.subplots(num_row, num_col, figsize=(3*num_col,3*num_row))
for i in range(num):
    ax = axes[i//num_col, i%num_col]
    ax.imshow(bd_x_test[randIdx[i]].astype('uint8'))
    ax.set_title('label: {:.0f}'.format(bd_y_test[randIdx[i]]))
    ax.set_xticks([])
    ax.set_yticks([])
plt.tight_layout()
plt.show()
```

Load the backdoored model.

The backdoor model and its weights can be found here

[8]:
```
## To-do ##

# First create clones of the original badnet model (by providing the model␣
↪filepath below)
# The result of repairing B_clone will be B_prime

B = keras.models.load_model("/content/drive/MyDrive/lab2/model/bd_net.h5")
B.load_weights("/content/drive/MyDrive/lab2/model/bd_weights.h5")

B_clone = keras.models.load_model("/content/drive/MyDrive/lab2/model/bd_net.h5")
B_clone.load_weights("/content/drive/MyDrive/lab2/model/bd_weights.h5")
```

Output of the original badnet accuracy on the validation data:

[9]:
```
# Get the original badnet model's (B) accuracy on the validation data
# clean accuracy at clean validation dataset
cl_label_p = np.argmax(B(cl_x_valid), axis=1)
clean_accuracy = np.mean(np.equal(cl_label_p, cl_y_valid)) * 100

print("Clean validation accuracy before pruning {0:3.6f}".
↪format(clean_accuracy))
K.clear_session()
```

```
Clean validation accuracy before pruning 98.649000
```

Write code to implement pruning defense

[10]:
```
## To-do ##
saved_model = np.zeros(3, dtype=bool)
# Redefine model to output right after the last pooling layer ("pool_3")
intermediate_model = Model(inputs=B.inputs, outputs=B.get_layer('pool_3').
↪output)

# Get feature map for last pooling layer ("pool_3") using the clean validation␣
↪data and intermediate_model
feature_maps_cl = intermediate_model.predict(cl_x_valid)

# Get average activation value of each channel in last pooling layer ("pool_3")
averageActivationsCl = np.mean(feature_maps_cl,axis=(0,1,2))

# Store the indices of average activation values (averageActivationsCl) in␣
↪increasing order
```

```
idxToPrune = np.argsort(averageActivationsCl)

# Get the conv_4 layer weights and biases from the original network that will␣
↪be used for prunning
# Hint: Use the get_weights() method (https://stackoverflow.com/questions/
↪43715047/how-do-i-get-the-weights-of-a-layer-in-keras)

lastConvLayerWeights = B.layers[5].get_weights()[0]
lastConvLayerBiases  = B.layers[5].get_weights()[1]

#prun channel with lowest weight first
for chIdx in tqdm(idxToPrune):

  # Prune one channel at a time
  # Hint: Replace all values in channel 'chIdx' of lastConvLayerWeights and␣
↪lastConvLayerBiases with 0
  lastConvLayerWeights[:,:,:,chIdx] = 0
  lastConvLayerBiases[chIdx] = 0

  # Update weights and biases of B_clone
  # Hint: Use the set_weights() method
  B_clone.layers[5].set_weights([lastConvLayerWeights, lastConvLayerBiases])

  # Evaluate the updated model's (B_clone) clean validation accuracy
  cl_label_p_valid = np.argmax(B_clone(cl_x_valid), axis=1)
  clean_accuracy_valid = np.mean(np.equal(cl_label_p_valid, cl_y_valid)) * 100

  # If drop in clean_accuracy_valid is just greater (or equal to) than the␣
↪desired threshold compared to clean_accuracy, then save B_clone as B_prime␣
↪and break
  if (clean_accuracy - clean_accuracy_valid >= 2 and not saved_model[0]):
    # Save B_clone as B_prime and break
    print("The accuracy drops at least 2%, saved the model")
    B_clone.save('model_X=2.h5')
    B_clone.save_weights('weightX=2.h5')
    saved_model[0] = 1

  if (clean_accuracy - clean_accuracy_valid >= 4 and not saved_model[1]):
    # Save B_clone as B_prime and break
    print("The accuracy drops at least 4%, saved the model")
    B_clone.save('model_X=4.h5')
    B_clone.save_weights('weightX=4.h5')
    saved_model[1] = 1

  if (clean_accuracy - clean_accuracy_valid >= 10 and not saved_model[2]):
    # Save B_clone as B_prime and break
    print("The accuracy drops at least 10%, saved the model")
```

```
    B_clone.save('model_X=10.h5')
    B_clone.save_weights('weightX=10.h5')
    saved_model[2] = 1
    break
  print("the clean accuracy is", clean_accuracy)
  print("the clean accuracy valid is", clean_accuracy_valid)
  print("the pruned channel index is", chIdx)
```

```
361/361 [==============================] - 14s 39ms/step

  2%|          | 1/60 [00:10<10:09, 10.33s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 0

  3%|          | 2/60 [00:23<11:28, 11.86s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 26

  5%|          | 3/60 [00:33<10:28, 11.03s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 27

  7%|          | 4/60 [00:43<09:55, 10.64s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 30

  8%|          | 5/60 [00:53<09:34, 10.44s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 31

 10%|          | 6/60 [01:03<09:24, 10.45s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 33

 12%|          | 7/60 [01:14<09:15, 10.48s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 34

 13%|          | 8/60 [01:25<09:12, 10.63s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 36

 15%|          | 9/60 [01:35<08:52, 10.44s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 37

 17%|          | 10/60 [01:53<10:35, 12.72s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 38

 18%|          | 11/60 [02:06<10:30, 12.86s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 25

 20%|          | 12/60 [02:17<09:55, 12.41s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 39

 22%|         | 13/60 [02:27<09:10, 11.72s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 41

 23%|         | 14/60 [02:38<08:36, 11.23s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 44

 25%|         | 15/60 [02:48<08:09, 10.88s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 45

 27%|         | 16/60 [02:58<07:49, 10.66s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 47

 28%|         | 17/60 [03:08<07:30, 10.49s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 48
```

```
 30%|          | 18/60 [03:18<07:15, 10.37s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 49

 32%|          | 19/60 [03:28<07:01, 10.29s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 50

 33%|          | 20/60 [03:38<06:50, 10.27s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 53

 35%|          | 21/60 [03:49<06:41, 10.29s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 55

 37%|          | 22/60 [03:59<06:29, 10.24s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 40

 38%|          | 23/60 [04:09<06:18, 10.22s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 24

 40%|          | 24/60 [04:19<06:07, 10.21s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 59

 42%|         | 25/60 [04:29<05:56, 10.19s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 9

 43%|         | 26/60 [04:39<05:47, 10.22s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 2

 45%|         | 27/60 [04:50<05:38, 10.25s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 12
```

```
 47%|        | 28/60 [05:00<05:27, 10.24s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 13
```

```
 48%|        | 29/60 [05:12<05:37, 10.88s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 17
```

```
 50%|        | 30/60 [05:23<05:19, 10.66s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 14
```

```
 52%|       | 31/60 [05:37<05:43, 11.85s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 15
```

```
 53%|       | 32/60 [05:47<05:17, 11.33s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 23
```

```
 55%|       | 33/60 [05:57<04:56, 10.98s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 6
```

```
 57%|       | 34/60 [06:08<04:38, 10.71s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64033948211657
the pruned channel index is 51
```

```
 58%|       | 35/60 [06:18<04:23, 10.54s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64033948211657
the pruned channel index is 32
```

```
 60%|       | 36/60 [06:28<04:10, 10.42s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.63167922404088
the pruned channel index is 22
```

```
 62%|      | 37/60 [06:38<04:01, 10.48s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.65765999826795
the pruned channel index is 21

 63%|      | 38/60 [06:54<04:27, 12.14s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.64899974019225
the pruned channel index is 20

 65%|      | 39/60 [07:09<04:29, 12.83s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.6056984498138
the pruned channel index is 19

 67%|      | 40/60 [07:19<04:01, 12.07s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.57105741751104
the pruned channel index is 43

 68%|      | 41/60 [07:29<03:38, 11.49s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.53641638520828
the pruned channel index is 58

 70%|      | 42/60 [07:42<03:32, 11.82s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 98.19000606218066
the pruned channel index is 3

 72%|      | 43/60 [07:52<03:12, 11.32s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 97.65307006148784
the pruned channel index is 42

 73%|      | 44/60 [08:02<02:56, 11.01s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 97.50584567420108
the pruned channel index is 1

 75%|      | 45/60 [08:13<02:43, 10.89s/it]

The accuracy drops at least 2%, saved the model
the clean accuracy is 98.64899974019225
the clean accuracy valid is 95.75647354291158
the pruned channel index is 29

 77%|      | 46/60 [08:23<02:28, 10.64s/it]
```

```
the clean accuracy is 98.64899974019225
the clean accuracy valid is 95.20221702606739
the pruned channel index is 16

 78%|      | 47/60 [08:33<02:16, 10.48s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 94.7172425738287
the pruned channel index is 56

 80%|      | 48/60 [08:43<02:04, 10.37s/it]

The accuracy drops at least 4%, saved the model
the clean accuracy is 98.64899974019225
the clean accuracy valid is 92.09318437689443
the pruned channel index is 46

 82%|      | 49/60 [08:58<02:08, 11.72s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 91.49562656967177
the pruned channel index is 5

 83%|      | 50/60 [09:08<01:52, 11.23s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 91.01931237550879
the pruned channel index is 8

 85%|      | 51/60 [09:18<01:37, 10.88s/it]

the clean accuracy is 98.64899974019225
the clean accuracy valid is 89.17467740538669
the pruned channel index is 11

 85%|      | 51/60 [09:28<01:40, 11.16s/it]

The accuracy drops at least 10%, saved the model
```

Now we need to combine the models into a repaired goodnet G that outputs the correct class if the test input is clean and class N+1 if the input is backdoored. One way to do it is to "subclass" the models in Keras:

[11]:
```python
#https://stackoverflow.com/questions/64983112/
↪keras-vertical-ensemble-model-with-condition-in-between
class G(tf.keras.Model):
    def __init__(self, B, B_prime):
        super(G, self).__init__()
        self.B = B
        self.B_prime = B_prime

    def predict(self,data):
```

```
        y = np.argmax(self.B(data), axis=1)
        y_prime = np.argmax(self.B_prime(data), axis=1)
        tmpRes = np.array([y[i] if y[i] == y_prime[i] else 1283 for i in␣
↪range(y.shape[0])])
        res = np.zeros((y.shape[0],1284))
        res[np.arange(tmpRes.size),tmpRes] = 1
        return res

    # For small amount of inputs that fit in one batch, directly using call()␣
↪is recommended for faster execution,
    # e.g., model(x), or model(x, training=False) is faster then model.
↪predict(x) and do not result in
    # memory leaks (see for more details https://www.tensorflow.org/api_docs/
↪python/tf/keras/Model#predict)
    def call(self,data):
        y = np.argmax(self.B(data), axis=1)
        y_prime = np.argmax(self.B_prime(data), axis=1)
        tmpRes = np.array([y[i] if y[i] == y_prime[i] else 1283 for i in␣
↪range(y.shape[0])])
        res = np.zeros((y.shape[0],1284))
        res[np.arange(tmpRes.size),tmpRes] = 1
        return res
```

However, Keras prevents from saving this kind of subclassed model as HDF5 file since it is not serializable. However, we still can use this architecture for model evaluation.

Load the saved B_prime model

[12]:
```
## To-do ##
# Provide B_prime model filepath below

B_prime2 = keras.models.load_model("/content/model_X=2.h5")
B_prime2.load_weights("/content/model_X=2.h5")

B_prime4 = keras.models.load_model("/content/model_X=4.h5")
B_prime4.load_weights("/content/model_X=4.h5")

B_prime10 = keras.models.load_model("/content/model_X=10.h5")
B_prime10.load_weights("/content/model_X=10.h5")
```

Check performance of the repaired model on the test data:

[13]:
```
cl_label_p2 = np.argmax(B_prime2.predict(cl_x_test), axis=1)
clean_accuracy_B_prime2 = np.mean(np.equal(cl_label_p2, cl_y_test))*100
print('Clean Classification accuracy for B_prime2:', clean_accuracy_B_prime2)

bd_label_p2 = np.argmax(B_prime2.predict(bd_x_test), axis=1)
asr_B_prime2 = np.mean(np.equal(bd_label_p2, bd_y_test))*100
```

```
print('Attack Success Rate for B_prime2:', asr_B_prime2)



cl_label_p4 = np.argmax(B_prime4.predict(cl_x_test), axis=1)
clean_accuracy_B_prime4 = np.mean(np.equal(cl_label_p4, cl_y_test))*100
print('Clean Classification accuracy for B_prime4:', clean_accuracy_B_prime4)

bd_label_p4 = np.argmax(B_prime4.predict(bd_x_test), axis=1)
asr_B_prime4 = np.mean(np.equal(bd_label_p4, bd_y_test))*100
print('Attack Success Rate for B_prime4:', asr_B_prime4)




cl_label_p10 = np.argmax(B_prime10.predict(cl_x_test), axis=1)
clean_accuracy_B_prime10 = np.mean(np.equal(cl_label_p10, cl_y_test))*100
print('Clean Classification accuracy for B_prime10:', clean_accuracy_B_prime10)

bd_label_p10 = np.argmax(B_prime10.predict(bd_x_test), axis=1)
asr_B_prime10 = np.mean(np.equal(bd_label_p10, bd_y_test))*100
print('Attack Success Rate for B_prime10:', asr_B_prime10)
```

```
401/401 [==============================] - 13s 32ms/step
Clean Classification accuracy for B_prime2: 95.90023382696803
401/401 [==============================] - 16s 39ms/step
Attack Success Rate for B_prime2: 100.0
401/401 [==============================] - 13s 32ms/step
Clean Classification accuracy for B_prime4: 92.29150428682775
401/401 [==============================] - 13s 32ms/step
Attack Success Rate for B_prime4: 99.98441153546376
401/401 [==============================] - 13s 32ms/step
Clean Classification accuracy for B_prime10: 84.54403741231489
401/401 [==============================] - 13s 32ms/step
Attack Success Rate for B_prime10: 77.20966484801247
```

Check performance of the original model on the test data:

[14]:
```
cl_label_p = np.argmax(B.predict(cl_x_test), axis=1)
clean_accuracy_B = np.mean(np.equal(cl_label_p, cl_y_test))*100
print('Clean Classification accuracy for B:', clean_accuracy_B)

bd_label_p = np.argmax(B.predict(bd_x_test), axis=1)
asr_B = np.mean(np.equal(bd_label_p, bd_y_test))*100
print('Attack Success Rate for B:', asr_B)
```

```
401/401 [==============================] - 13s 32ms/step
Clean Classification accuracy for B: 98.62042088854248
```

```
401/401 [==============================] - 14s 36ms/step
Attack Success Rate for B: 100.0
```

Create repaired network

[15]:
```
# Repaired network repaired_net
repaired_net2 = G(B, B_prime2)

repaired_net4 = G(B, B_prime4)

repaired_net10 = G(B, B_prime10)
```

Check the performance of the repaired_net on the test data

[16]:
```
cl_label_p2 = np.argmax(repaired_net2(cl_x_test), axis=1)
clean_accuracy_repaired_net2 = np.mean(np.equal(cl_label_p2, cl_y_test))*100
print('Clean Classification accuracy for repaired net2:',␣
 ↪clean_accuracy_repaired_net2)

bd_label_p2 = np.argmax(repaired_net2(bd_x_test), axis=1)
asr_repaired_net2 = np.mean(np.equal(bd_label_p2, bd_y_test))*100
print('Attack Success Rate for repaired net2:', asr_repaired_net2)



cl_label_p4 = np.argmax(repaired_net4(cl_x_test), axis=1)
clean_accuracy_repaired_net4 = np.mean(np.equal(cl_label_p4, cl_y_test))*100
print('Clean Classification accuracy for repaired net4:',␣
 ↪clean_accuracy_repaired_net4)

bd_label_p4 = np.argmax(repaired_net4(bd_x_test), axis=1)
asr_repaired_net4 = np.mean(np.equal(bd_label_p4, bd_y_test))*100
print('Attack Success Rate for repaired net4:', asr_repaired_net4)



cl_label_p10 = np.argmax(repaired_net10(cl_x_test), axis=1)
clean_accuracy_repaired_net10 = np.mean(np.equal(cl_label_p10, cl_y_test))*100
print('Clean Classification accuracy for repaired net10:',␣
 ↪clean_accuracy_repaired_net10)

bd_label_p10 = np.argmax(repaired_net10(bd_x_test), axis=1)
asr_repaired_net10 = np.mean(np.equal(bd_label_p10, bd_y_test))*100
print('Attack Success Rate for repaired net10:', asr_repaired_net10)
```

```
Clean Classification accuracy for repaired net2: 95.74434918160561
Attack Success Rate for repaired net2: 100.0
Clean Classification accuracy for repaired net4: 92.1278254091972
```

```
Attack Success Rate for repaired net4: 99.98441153546376
Clean Classification accuracy for repaired net10: 84.3335931410756
Attack Success Rate for repaired net10: 77.20966484801247
```

[21]:
```
!!jupyter nbconvert --to pdf Homework-2.i
```

```
/content/drive/MyDrive/Colab Notebooks
```