```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import keras
import sys
import h5py
import numpy as np
import pdb
import tensorflow as tf
from keras.models import Model

clean_data_filename = "/content/drive/MyDrive/Professional/Grad-NYU/SEM 3 - F23/ECE-GY 9163 ML for cyber sec/HW4/CSAW-HackML-2020/lab3/da
poisoned_data_filename = "/content/drive/MyDrive/Professional/Grad-NYU/SEM 3 - F23/ECE-GY 9163 ML for cyber sec/HW4/CSAW-HackML-2020/lab3
#model_filename = "/content/drive/MyDrive/Professional/Grad-NYU/SEM 3 - F23/ECE-GY 9163 ML for cyber sec/HW4/CSAW-HackML-2020/models/sung
model_filename = "/content/drive/MyDrive/Professional/Grad-NYU/SEM 3 - F23/ECE-GY 9163 ML for cyber sec/HW4/CSAW-HackML-2020/lab3/models/

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


cl_x_test, cl_y_test = data_loader(clean_data_filename)
bd_x_test, bd_y_test = data_loader(poisoned_data_filename)

bd_model = keras.models.load_model(model_filename)
bd_model.summary()
```

```
bd_model = keras.models.load_model(model_filename)
cl_label_p = np.argmax(bd_model.predict(cl_x_test), axis=1)
clean_accuracy = np.mean(np.equal(cl_label_p, cl_y_test))*100
print('Clean Classification accuracy:', clean_accuracy)


original_accuracy = bd_model.evaluate(cl_x_test, cl_y_test)[1]  # Evaluate initial accuracy
print(original_accuracy)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-244d39076ac2> in <cell line: 2>()
      1 bd_model = keras.models.load_model(model_filename)
----> 2 cl_label_p = np.argmax(bd_model.predict(cl_x_test), axis=1)
      3 clean_accuracy = np.mean(np.equal(cl_label_p, cl_y_test))*100
      4 print('Clean Classification accuracy:', clean_accuracy)
      5

NameError: name 'cl_x_test' is not defined
```

SEARCH STACK OVERFLOW

```
import tensorflow as tf
clean_test_accuracies, attack_rates = [], []
for threshold in [2, 4, 10]:
#for threshold in [0.04, 0.1]:
    bd_model = keras.models.load_model(model_filename)
    pruned_model, clean_acc_list, attack_rate_list = prune_model(bd_model, cl_x_test, cl_y_test, bd_x_test, bd_y_test,threshold)
    clean_test_accuracies.append(clean_acc_list)
    attack_rates.append(attack_rate_list)

    # Save the pruned model
    model_save_path = f"pruned_model_threshold_{threshold}.h5"
    pruned_model.save(model_save_path)
    print(f"Pruned model with threshold {threshold} saved to {model_save_path}")
```

```
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
### Initial Clean Accuracy = 98.64899974019225
361/361 [==============================] - 1s 3ms/step
```

```
### Initial Attack Rate = 100.0
#### Starting channel pruning for threshold=2
361/361 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 2ms/step
0. Clean Accuracy after pruning channel 57 = 94.29288992811986
0. Attack rate after pruning channel 57 = 100.0
Acc goes below 98.64899974019225 - 2 = 96.64899974019225, Thus saving the model...
Pruned model with threshold 2 saved to pruned_model_threshold_2.h5
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file v:
  saving_api.save_model(
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 3ms/step
### Initial Clean Accuracy = 98.64899974019225
361/361 [==============================] - 1s 3ms/step
### Initial Attack Rate = 100.0
#### Starting channel pruning for threshold=4
361/361 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 2ms/step
0. Clean Accuracy after pruning channel 57 = 94.29288992811986
0. Attack rate after pruning channel 57 = 100.0
Acc goes below 98.64899974019225 - 4 = 94.64899974019225, Thus saving the model...
Pruned model with threshold 4 saved to pruned_model_threshold_4.h5
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
### Initial Clean Accuracy = 98.64899974019225
361/361 [==============================] - 1s 2ms/step
### Initial Attack Rate = 100.0
#### Starting channel pruning for threshold=10
361/361 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
0. Clean Accuracy after pruning channel 57 = 94.29288992811986
0. Attack rate after pruning channel 57 = 100.0
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
1. Clean Accuracy after pruning channel 52 = 74.1491296440634
1. Attack rate after pruning channel 52 = 100.0
Acc goes below 98.64899974019225 - 10 = 88.64899974019225, Thus saving the model...
Pruned model with threshold 10 saved to pruned_model_threshold_10.h5
```

```
print(cl_x_test.shape)
print(cl_y_test.shape)
```

```
def prune_model(model, x_data_valid, y_data_valid, bd_x_test, bd_y_test, threshold):
    # last_pooling_layer = model.get_layer('pool_3')
    # weights = last_pooling_layer.get_weights()  # Get the layer's weights
    # pdb.set_trace()
    f=True
    last_pooling_layer = 'pool_3'
    outputs = [layer.output for layer in model.layers if layer.name == last_pooling_layer]

    intermediate_layer_model = Model(inputs= model.input, outputs=outputs)
    # Assuming last_conv_layer represents the last pooling layer in your model.
    last_pooling_layer = model.get_layer('pool_3')

    # Calculate activation values for each channel on the validation data
    #activations = last_pooling_layer.predict(x_data_valid)
    activations = intermediate_layer_model.predict(x_data_valid)
    #pdb.set_trace()
    # print(type(activations))
    # print(len(activations))
    avg_activations = np.mean(activations, axis=(0, 1, 2))  # Calculate average activation per channel
    #print(f'avg activations = {avg_activations}')
    # Pruning based on average activation values
    channels_to_prune = np.argsort(-avg_activations)  # Sort channels by activation values
    #print(f'channels_to_prune = {channels_to_prune}')
    #print(f'channels_to_prune = {np.argsort(avg_activations)}')
    clean_acc_list , attack_rate_list= [], []
    cl_label_p = np.argmax(model.predict(x_data_valid), axis=1)
    original_accuracy = np.mean(np.equal(cl_label_p, y_data_valid))*100
    print(f'### Initial Clean Accuracy = {original_accuracy}')
    clean_acc_list.append(original_accuracy)
    bd_label = np.argmax(model.predict(bd_x_test), axis=1)
    original_attack_rate = np.mean(np.equal(bd_label, bd_y_test))*100
    print(f'### Initial Attack Rate = {original_attack_rate}')
    attack_rate_list.append(original_attack_rate)

    print(f'#### Starting channel pruning for threshold={threshold}')


    # Perform pruning and evaluate accuracy after each pruning step

    for i,channel in enumerate(channels_to_prune):

        conv_3_weights = model.get_layer('conv_3').get_weights()
        # Set the channel weights in conv_3 layer to zeroes
        conv_3_weights[0][:, :, :, channel] = 0.0  # Set the channel weights to 0
        conv_3_weights[1][channel] = 0.0  # Set the channel bias to 0
        # Set the modified weights back to the conv_3 layer
        model.get_layer('conv_3').set_weights(conv_3_weights)

        #last_pooling_layer.set_weights(new_weights)  # Set the modified weights back to the layer
        cl_label = np.argmax(model.predict(x_data_valid), axis=1)
        pruned_accuracy = np.mean(np.equal(cl_label, y_data_valid))*100
        clean_acc_list.append(pruned_accuracy)

        bd_label = np.argmax(model.predict(bd_x_test), axis=1)
        attack_rate = np.mean(np.equal(bd_label, bd_y_test))*100
        attack_rate_list.append(attack_rate)

        print(f'{i}. Clean Accuracy after pruning channel {channel} = {pruned_accuracy}')
        print(f'{i}. Attack rate after pruning channel {channel} = {attack_rate}')
        # Stop pruning when accuracy drops by a certain threshold (X% below original accuracy)
        if pruned_accuracy <= (original_accuracy - threshold) and f:
            f=False
            print(f'Acc goes below {original_accuracy} - {threshold} = {original_accuracy - threshold}, Thus saving the model...')
            #pruned_model = tf.keras.models.clone_model(model)
            break
        del conv_3_weights, cl_label, bd_label


    del activations
    del avg_activations
    del channels_to_prune
    del conv_3_weights
    #del model


    #return pruned_model, clean_acc_list, attack_rate_list
    return model, clean_acc_list, attack_rate_list
```

```
from google.colab import files
import zipfile

# File paths of the files to be zipped
file_paths = ['/content/pruned_model_threshold_10.h5', '/content/pruned_model_threshold_2.h5', '/content/pruned_model_threshold_4.h5']

# Zip the files
zipf = zipfile.ZipFile('/content/pruned_models.zip', 'w', zipfile.ZIP_DEFLATED)
for file in file_paths:
    zipf.write(file)
zipf.close()

# Download the zipped file
files.download('/content/pruned_models.zip')
```

```
from tensorflow.keras.layers import Input, Lambda

def create_goodnet(B, B_prime):
    # Input layer for test input data
    input_shape = B.input_shape[1:]  # Retrieve the input shape of model B, excluding the batch size
    input_data = Input(shape=input_shape)  # Create an input layer with the obtained shape


    # Prediction using model B
    prediction_B = B(input_data)

    # Prediction using model B_prime
    prediction_B_prime = B_prime(input_data)

    # Lambda layer to compare predictions
    compare_predictions = Lambda(lambda x: K.cast(K.equal(x[0], x[1]), dtype='float32'))([prediction_B, prediction_B_prime])

    # Output layer for Goodnet G
    output = Lambda(lambda x: K.sum(x, axis=1))(compare_predictions)

    # Define the Goodnet model
    goodnet_model = Model(inputs=input_data, outputs=output)

    return goodnet_model
```

```
from keras.models import Model, load_model
import tensorflow as tf


class Goodnet(Model):
    def __init__(self, model_B, model_B_prime):
        super(Goodnet, self).__init__()
        self.model_B = model_B
        self.model_B_prime = model_B_prime

    def call(self,data):
        y = np.argmax(self.model_B(data), axis=1)
        y_prime = np.argmax(self.model_B_prime(data), axis=1)
        pred = np.zeros(data.shape[0])
        for i in range(data.shape[0]):
          if y[i]==y_prime[i]:
            pred[i] = y[i]
          else:
            pred[i] = 1284
        return pred
```

```
import os
B = keras.models.load_model(model_filename)
pruned_model_path ="/content/drive/MyDrive/Professional/Grad-NYU/SEM 3 - F23/ECE-GY 9163 ML for cyber sec/HW4/CSAW-HackML-2020/lab3/prun
pruned_model_path ="/content/"
for threshold in [2, 4, 10]:
  B_prime = keras.models.load_model(pruned_model_path+'/pruned_model_threshold_'+str(threshold)+'.h5')
  goodnet_model = Goodnet(B, B_prime)
  #op_gg = goodnet_model(cl_x_test)
  #pdb.set_trace()
  Good_clean_test_2_label_p = goodnet_model(cl_x_test)
  Good_clean_test_2_accuracy = np.mean(np.equal(Good_clean_test_2_label_p, cl_y_test))*100
  print(f'Combined model with {threshold}% dropped acc, the clean test data classification accuracy:', Good_clean_test_2_accuracy)

  Good_bd_test_2_label_p = goodnet_model(bd_x_test)
  Good_model_asrate_2 = np.mean(np.equal(Good_bd_test_2_label_p, bd_y_test))*100
  print(f'Combined model with {threshold}% dropped acc, attack success Rate:', Good_model_asrate_2)
  # cl_label_p = np.argmax(goodnet_model(cl_x_test), axis=1)
  # clean_accuracy = np.mean(np.equal(cl_label_p, cl_y_test))*100
  # print('Clean Classification accuracy:', clean_accuracy)

for threshold in [2, 4, 10]:
  #B_prime = keras.models.load_model(pruned_model_path+'/pruned_model_threshold_'+str(threshold)+'.h5')
  B_prime = keras.models.load_model('/content/pruned_model_threshold_'+str(threshold)+'.h5')
  clean_test_labels = np.argmax(B_prime.predict(cl_x_test), axis=1)
  clean_test_accuracy = np.mean(np.equal(clean_test_labels, cl_y_test))*100
  print(f'Model with threshold= {threshold}%, the clean test data Classification accuracy:', clean_test_accuracy)

  bd_test_labels = np.argmax(B_prime.predict(bd_x_test), axis=1)
  asrate = np.mean(np.equal(bd_test_labels, bd_y_test))*100
  print(f'Model with {threshold}% dropped , Attack Success Rate:', asrate)
```

```
361/361 [==============================] - 1s 2ms/step
Model with threshold= 2%, the clean test data Classification accuracy: 94.29288992811986
361/361 [==============================] - 1s 3ms/step
Model with 2% dropped , Attack Success Rate: 100.0
361/361 [==============================] - 1s 2ms/step
Model with threshold= 4%, the clean test data Classification accuracy: 94.29288992811986
361/361 [==============================] - 1s 2ms/step
Model with 4% dropped , Attack Success Rate: 100.0
361/361 [==============================] - 1s 3ms/step
Model with threshold= 10%, the clean test data Classification accuracy: 74.1491296440634
361/361 [==============================] - 1s 3ms/step
Model with 10% dropped , Attack Success Rate: 100.0
```

```
import matplotlib.pyplot as plt
test_accuracy = [clean_acc_2, clean_acc_4, clean_acc_10]
attack_success_rates = [ar_2, ar_4, ar_10]

opacity = 1
bar_width = 0.3

plt.xlabel('model with % accuracy drop')
plt.ylabel('different rates')

plt.xticks(range(len(test_accuracy)),('2%', '4%', '10%'))
bar1 = plt.bar(np.arange(len(test_accuracy)) + bar_width, test_accuracy, bar_width, align='center', alpha=opacity, color='red', label='cl
bar2 = plt.bar(range(len(attack_success_rates)), attack_success_rates, bar_width, align='center', alpha=opacity, color='black', label='at

# Add counts above the two bar graphs
for rect in bar1 + bar2:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width() / 2.0, height, f'{height:.02f}', ha='center', va='bottom')

plt.legend(bbox_to_anchor=(1.4, 1))
plt.tight_layout()
plt.title('performance of goodnet model')

plt.show()
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.