nm3937_mlcybersec_lab2

December 9, 2022

[ ]:

```
import tensorflow as tf
import tempfile
import sys
import h5py
import numpy as np
import matplotlib.pyplot as plt
import keras
import keras.backend as K
from keras import models
from keras.models import Model
from keras import initializers
from keras.utils.vis_utils import plot_model
```

0.1 Model

[ ]:

```
def Net():
        # define input
        x = keras.Input(shape=(55, 47, 3), name='input')
        # feature extraction
        conv_1 = keras.layers.Conv2D(20, (4, 4), activation='relu',␣
 ↪name='conv_1')(x)
        pool_1 = keras.layers.MaxPooling2D((2, 2), name='pool_1')(conv_1)
        conv_2 = keras.layers.Conv2D(40, (3, 3), activation='relu',␣
 ↪name='conv_2')(pool_1)
        pool_2 = keras.layers.MaxPooling2D((2, 2), name='pool_2')(conv_2)
        conv_3 = keras.layers.Conv2D(60, (3, 3), activation='relu',␣
 ↪name='conv_3')(pool_2)
        pool_3 = keras.layers.MaxPooling2D((2, 2), name='pool_3')(conv_3)
        # first interpretation model
        flat_1 = keras.layers.Flatten()(pool_3)
        fc_1 = keras.layers.Dense(160, name='fc_1')(flat_1)
        # second interpretation model
        conv_4 = keras.layers.Conv2D(80, (2, 2), activation='relu',␣
 ↪name='conv_4')(pool_3)
        flat_2 = keras.layers.Flatten()(conv_4)
        fc_2 = keras.layers.Dense(160, name='fc_2')(flat_2)
        # merge interpretation
        merge = keras.layers.Add()([fc_1, fc_2])
        add_1 = keras.layers.Activation('relu')(merge)
        drop = keras.layers.Dropout(0.5)
        # output
        y_hat = keras.layers.Dense(1283, activation='softmax',␣
↪name='output')(add_1)
        model = keras.Model(inputs=x, outputs=y_hat)
        # summarize layers
        #print(model.summary())
        # plot graph
        #plot_model(model, to_file='model_architecture.png')

        return model
```

0.2 data loading function

[ ]:

```
def load_from_path(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
```

0.3 Loading models, data

[8]:

```
bad_model = '/content/drive/MyDrive/Lab2/bd_net.h5'
bad_wts = '/content/drive/MyDrive/Lab2/bd_weights.h5'

clean_test_data = '/content/drive/MyDrive/Lab2/test.h5'
clean_val_data = '/content/drive/MyDrive/Lab2/valid.h5'
sunglass_val_data = '/content/drive/MyDrive/Lab2/bd_valid.h5'
sunglass_test_data = '/content/drive/MyDrive/Lab2/bd_test.h5'


xTest_clean, yTest_clean = load_from_path(clean_test_data)
xVal_clean, yVal_clean = load_from_path(clean_val_data)
xBackdoor, yBackdoor = load_from_path(sunglass_val_data)
```

[7]:

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

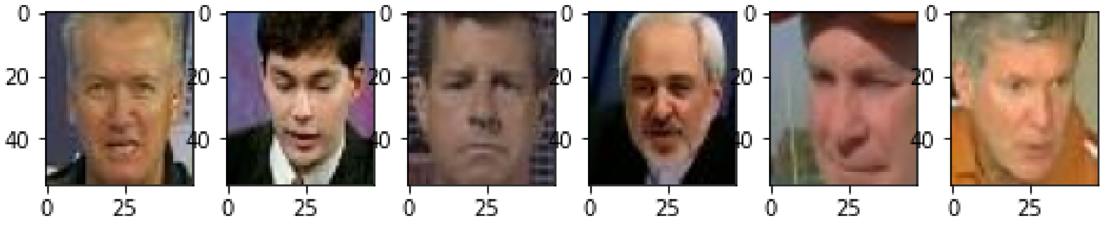
```
Mounted at /content/drive
```

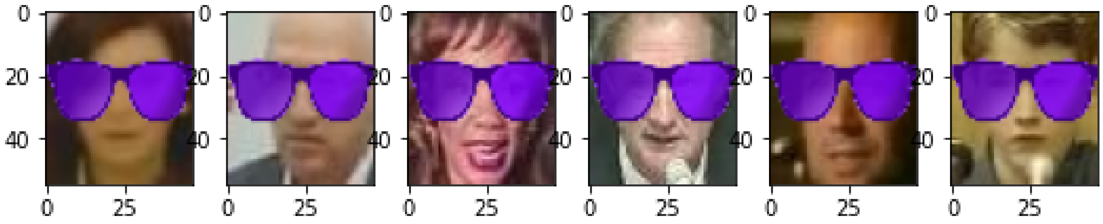
0.4 plot the data

[9]:

```
plt.figure(figsize=(9,1.5))
for id in range(1,7):
    plt.subplot(1,6,id)
    plt.imshow(xTest_clean[id]/255)

plt.figure(figsize=(9,1.5))
for id in range(1,7):
    plt.subplot(1,6,id)
    plt.imshow(xBackdoor[id]/255)
```





0.4.1 function to check accuracy

[10]:

```
def check_acc(model, xTest_clean, x_test_p, yTest_clean, y_test_p):
  predicted_clean = np.argmax(model.predict(xTest_clean), axis=1)
  predicted_bdoor = np.argmax(model.predict(x_test_p), axis=1)
  acc_c = np.mean(np.equal(predicted_clean, yTest_clean))*100
  acc_b = np.mean(np.equal(predicted_bdoor, y_test_p))*100
  print('Clean input accuracy: {:.2f}%'.format(acc_c))
  print('Backdoored input accuracy: {:.2f}%'.format(acc_b))
  return acc_c,acc_b
```

0.5 accuracy for the backdoored model

[11]:
```
model_bdoor = keras.models.load_model(bad_model)
_,__ = check_acc(model_bdoor, xTest_clean, xBackdoor, yTest_clean, yBackdoor)
```

```
401/401 [==============================] - 9s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.62%
Backdoored input accuracy: 100.00%
```

0.6 Pruning

0.6.1 getting the average activation values

[12]:
```
conv_3 = Model(inputs=model_bdoor.input,
            outputs=model_bdoor.get_layer("conv_3").output)
out = np.mean(conv_3.predict(xVal_clean), axis=0)
sorted_idx = np.argsort(np.sum(out, axis=(0, 1)))
print(sorted_idx)
```

```
361/361 [==============================] - 1s 2ms/step
[ 0 26 27 30 31 33 34 36 37 38 25 39 41 44 45 47 48 49 50 53 55 40 24 59
  9  2 12 13 17 14 15 23  6 51 32 22 21 20 19 43  3 58 42  1 29 16  5 56
  8 11 46 54 10  4 18  7 28 35 52 57]
```

[13]:
```
print(np.sort(np.sum(out, axis=(0, 1))))
```

```
[0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 8.1761219e-02 1.7022046e-01 3.4326309e-01 3.5326573e-01
 1.1469302e+00 2.5636511e+00 5.0632768e+00 8.5227013e+00 1.2338524e+01
 1.7449795e+01 1.7924007e+01 2.1239481e+01 2.2799477e+01 3.7998371e+01
 5.7887741e+01 6.9720215e+01 7.0036186e+01 7.1875938e+01 8.5014481e+01
 8.7924690e+01 1.6558409e+02 1.9046292e+02 2.2774965e+02 2.3424626e+02
 2.3575990e+02 2.4531384e+02 2.8121823e+02 3.2685699e+02 3.9856320e+02]
```

[14]:
```
num_pruned_layers = {2:45,4:48,10:52,30:54} # accuracy tolerance matched with␣
 ↪the number of channels to be pruned

def prune_layers(x,model_bdoor):
  conv_layer = model_bdoor.get_layer("conv_3")
  weight, bias = conv_layer.get_weights()
  K.clear_session()
  acc_clean = []
  acc_bad = []
  for i in range(30,num_pruned_layers[x]): # only looping from
    print(i)
    cur_idx = sorted_idx[i]
    weight[:, :, :, cur_idx] = 0.0
    bias[cur_idx] = 0.0
    conv_layer.set_weights([weight, bias])
    acc1,acc2 = check_acc(model_bdoor, xTest_clean, xBackdoor, yTest_clean,␣
↪yBackdoor)

    ###. Saving pruned nets B1
    if i == 44:
      model_bdoor.save('/content/drive/MyDrive/Lab2/B1'+ '_2' +'.h5')
    if i == 47:
      model_bdoor.save('/content/drive/MyDrive/Lab2/B1'+ '_4' +'.h5')
    if i == 51:
      model_bdoor.save('/content/drive/MyDrive/Lab2/B1'+ '_10' +'.h5')
    if i == 53:
      model_bdoor.save('/content/drive/MyDrive/Lab2/B1'+ '_30' +'.h5')
    acc_clean.append(acc1)
    acc_bad.append(acc2)
  return acc_clean,acc_bad,model_bdoor
```

[15]:
```
K.clear_session()
model_bdoor = keras.models.load_model(bad_model)
acc_tol = 30
acc_clean,acc_bad,model_bdoor = prune_layers(acc_tol,model_bdoor)
```

```
30
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.62%
Backdoored input accuracy: 100.00%
31
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.62%
Backdoored input accuracy: 100.00%
32
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.62%
Backdoored input accuracy: 100.00%
33
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.61%
```

```
Backdoored input accuracy: 100.00%
34
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 4ms/step
Clean input accuracy: 98.61%
Backdoored input accuracy: 100.00%
35
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.60%
Backdoored input accuracy: 100.00%
36
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.60%
Backdoored input accuracy: 100.00%
37
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.59%
Backdoored input accuracy: 100.00%
38
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.55%
Backdoored input accuracy: 100.00%
39
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.53%
Backdoored input accuracy: 100.00%
40
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.29%
Backdoored input accuracy: 100.00%
41
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 98.27%
Backdoored input accuracy: 100.00%
42
401/401 [==============================] - 1s 4ms/step
361/361 [==============================] - 1s 4ms/step
Clean input accuracy: 97.89%
Backdoored input accuracy: 100.00%
43
401/401 [==============================] - 1s 3ms/step
```

```
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 97.66%
Backdoored input accuracy: 100.00%
44
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 95.90%
Backdoored input accuracy: 100.00%
45
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 95.53%
Backdoored input accuracy: 99.99%
46
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 95.22%
Backdoored input accuracy: 100.00%
47
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 94.77%
Backdoored input accuracy: 99.99%
48
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 94.18%
Backdoored input accuracy: 99.98%
49
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 92.51%
Backdoored input accuracy: 80.48%
50
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 3ms/step
Clean input accuracy: 89.84%
Backdoored input accuracy: 80.74%
51
401/401 [==============================] - 1s 4ms/step
361/361 [==============================] - 1s 4ms/step
Clean input accuracy: 84.54%
Backdoored input accuracy: 77.02%
52
401/401 [==============================] - 1s 3ms/step
361/361 [==============================] - 1s 4ms/step
Clean input accuracy: 76.31%
Backdoored input accuracy: 35.71%
```

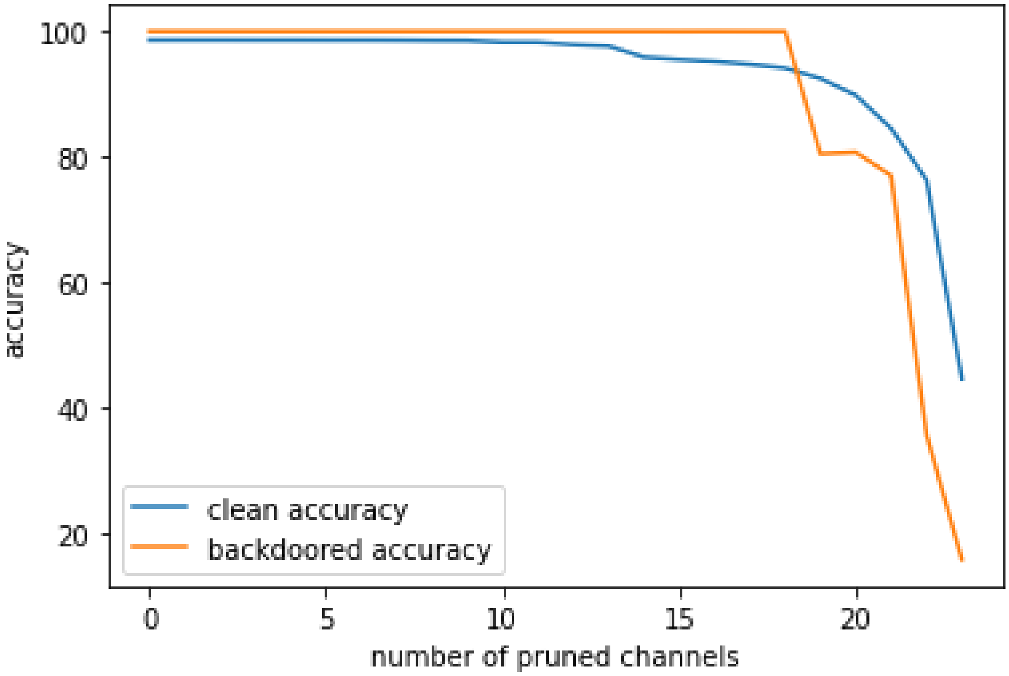
```
53
401/401 [==============================] - 2s 4ms/step
361/361 [==============================] - 2s 5ms/step
Clean input accuracy: 44.68%
Backdoored input accuracy: 15.87%
```

[16]:

```
import matplotlib.pyplot as plt
plt.plot(acc_clean,label='clean accuracy')
plt.plot(acc_bad,label = 'backdoored accuracy')
plt.legend(loc="lower left")
plt.xlabel('number of pruned channels')
plt.ylabel('accuracy')
```

[16]:

```
Text(0, 0.5, 'accuracy')
```



0.7 combining the fined tuned (retrained) pruned net and bad net

[19]:

```
#from keras.layers.merge import concatenate
from tensorflow.keras.layers import concatenate

B = keras.models.load_model(bad_model)
path =  '/content/drive/MyDrive/Lab2/B1'+ '_30' +'.h5'
B1 = keras.models.load_model(path)
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

```
B1.compile(optimizer='adam',
        loss=loss_fn,
        metrics=['accuracy'])

B1.fit(xVal_clean,yVal_clean,epochs=5)
model_list = [B,B1]
```

```
Epoch 1/5
```

```
/usr/local/lib/python3.8/dist-packages/tensorflow/python/util/dispatch.py:1082:
UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but
the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
```

```
361/361 [==============================] - 4s 6ms/step - loss: 0.7673 -
accuracy: 0.8460
Epoch 2/5
361/361 [==============================] - 2s 6ms/step - loss: 0.1654 -
accuracy: 0.9590
Epoch 3/5
361/361 [==============================] - 2s 6ms/step - loss: 0.1436 -
accuracy: 0.9576
Epoch 4/5
361/361 [==============================] - 2s 6ms/step - loss: 0.1529 -
accuracy: 0.9581
Epoch 5/5
361/361 [==============================] - 2s 6ms/step - loss: 0.1295 -
accuracy: 0.9661
```

[20]:

```
def fun(x):
  z1 = x[0]
  z2 = x[1]
  ans = tf.where(z1 == z2,z1,1283)
  return ans
```

[21]:

```
def combine_models(model_list):
  for i in range(len(model_list)):
    model = model_list[i]
    for conv_layer in model.layers:
      conv_layer.trainable = False
      conv_layer._name =  str(i+1) + '_' + conv_layer._name
  ensemble_visible = [model.input for model in model_list]
  z1 = keras.layers.Lambda(K.argmax, arguments={'axis':-1})(model_list[0].
↪output)
  z2 = keras.layers.Lambda(K.argmax, arguments={'axis':-1})(model_list[1].
↪output)
  out = keras.layers.Lambda(fun)([z1, z2])
  model = Model(inputs=ensemble_visible, outputs=out)
  model.compile(optimizer='adam', loss='binary_crossentropy')
  return model
```

[22]:
```
model = combine_models(model_list)
```

0.8 saving the repaired net (G)

[ ]:
```
# model.save('/content/repaired_net_x30.h5')
```

0.9 Predicted labels using repaired net

0.9.1 for backdoored inputs

[23]:
```
labels = list(set(yVal_clean))
print('Number of labels',len(labels))
```

```
Number of labels 1283
```

[24]:
```
## expected output is 1283 (backdoored class)
print('number of backdoored inputs: ',yBackdoor.shape)
y_predicted = model.predict((xBackdoor,xBackdoor))
count_arr = np.bincount(y_predicted)
print('number of backdoored inputs detected = ',count_arr[1283])
print('Backdoor detection accuracy = ',(count_arr[1283]/yBackdoor.shape)*100)
```

```
number of backdoored inputs:  (11547,)
361/361 [==============================] - 2s 5ms/step
number of backdoored inputs detected =  8600
Backdoor detection accuracy =  [74.47821945]
```

[25]:
```
y_predicted = model.predict((xTest_clean,xTest_clean))
acc = np.mean(np.equal(y_predicted, yTest_clean))*100
print('clean data accuracy:',acc)
```

```
401/401 [==============================] - 2s 5ms/step
clean data accuracy: 88.00467653936087
```

0.10 Comments

I think the pruning defence alone doesn't work well as the training accuracy for clean inputs reduces significantly, although the backdoored inputs accuracy drops. But pruning followed by fine tuning works well. The accuracy on the clean data drops a bit, but I think its a good tradeoff as the backdoor accuracy drops to 5%.

[ ]:
```
```