IMPORTS

[ ] ↳ 2 cells hidden

Running this Notebook

https://drive.google.com/drive/folders/1mf9UHHPq6tg8kZGlFTrFpCJfkg4zhWiB?usp=sharing

Can be open this with NYU account,

Please add this to your google Drive and you're good to go

You can find results you generated in "results" and "repaired-nets" folders We stored results we got in "results_we_got" and "repaired-nets_we_got" So if you open those you can directly verify the results we are able to obtain

GET DATA FROM DRIVE

```
from google.colab import drive
drive.mount('/content/drive')
sys.path.append(os.path.abspath("/content/drive/MyDrive/MLSecurityProject/"))
```

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

API PARAMETERS

Change the file names hear

[ ] ↳ 1 cell hidden

DATA PROCESSING

[ ] ↳ 5 cells hidden

VISUALISER CLASS

[ ] ↳ *1 cell hidden*

GET POTENTIAL TRIGGERS

[ ] ↳ *3 cells hidden*

FILTER POTENTIAL TRIGGERS TO GET ACTUAL TRIGGERS

[ ] ↳ *2 cells hidden*

Backdoor mitigation functions

```
def get_random_samples(X, Y, split):
    X_tr, X_clean,y_tr, y_clean= train_test_split(X, Y, train_size=split, random_st
    return X_tr,X_clean,y_tr, y_clean
```

```
def get_mask_pattern(y):
    maskFile = IMG_FILENAME_TEMPLATE % ('mask', y)
    mask = None
    pattern = None
    img = image.load_img('%s/%s' % (RESULT_DIR, maskFile),target_size=inpShape)
    mask = image.img_to_array(img)

    patternFile = IMG_FILENAME_TEMPLATE % ('pattern', y)
    img = image.load_img('%s/%s' % (RESULT_DIR, patternFile),target_size=inpShape)
    pattern = image.img_to_array(img)
    return mask, pattern

def injection(mask, pattern, adv_img):
    return mask * pattern + (1 - mask) * adv_img

def infect_X(img, tgt):
    mask, pattern = get_mask_pattern(tgt)
    rawImg = np.copy(img)
    advImg = np.copy(rawImg)
    advImg = injection(mask, pattern, advImg)
    return advImg, tgt
```

```
def get_avg_neuron_activation():
    modelFile = '%s/%s' % (MODEL_DIR, MODEL_FILENAME)
    model_cl = load_model(modelFile)
    model_cl.fit(x=X_mtg, y=Y_mtg, epochs=5 ,shuffle=True)
    layer_outputs_cl = [layer.output for layer in model_cl.layers[12:13]]
    activation_model_cl = models.Model(inputs=model_cl.input, outputs=layer_outputs

    model_ps = load_model(modelFile)
    model_ps.fit(x= patchImages, y=patchLabels, epochs=5 ,shuffle=True)
    layer_outputs_ps = [layer.output for layer in model_ps.layers[12:13]]
    activation_model_ps = models.Model(inputs=model_ps.input, outputs=layer_outputs

    activations_cl=[]
    activations_cl=activation_model_cl.predict(X_mtg[:1000])
    avg_activation_cl = sum(np.array(activations_cl))/len(activations_cl)

    activations_ps=[]
    activations_ps =activation_model_ps.predict(patchImages[:1000])
    avg_activation_ps = sum(np.array(activations_ps))/len(activations_ps)
    diff_activation = avg_activation_ps - avg_activation_cl
    mean_act = sum(avg_activation_cl)/len(avg_activation_cl)
    return mean_act,diff_activation



#Filter for Detecting Adversarial Inputs using Nueron Activations

def detect_adversial_inputs(model,x_i,mean_act):
    x=[]
    x.append(x_i)
    x=np.array(x)
    layer_outputs = [layer.output for layer in model.layers[12:13]]
    activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
    for neuron in activation_model.predict(x)[0] :
        if neuron > mean_act+1000 :
          return True
    return False
```

```
#Neuron pruning by setting the neuron value to zero in last but 2nd layer when weig
def neuron_pruning(model):
    modelFile = '%s/%s' % (MODEL_DIR, MODEL_FILENAME)
    new_model = load_model(modelFile)
    new_weights = model.get_weights().copy()
    for layer in new_weights[12]:
      for i in range(0,len(layer)):
          if layer[i] > mean_act+1000 :
              layer[i] =0
    return new_model


def reverse_trigger_to_clean_images(X_mtg, Y_mtg):
    patchImages = []
    patchLabels = []
    imgIndex=[]
    for ind, img in enumerate(X_mtg):
        y = Y_mtg[ind]
        patchImg, patchLabel = infect_X(img, y)
        patchImages.append(patchImg)
        patchLabels.append(patchLabel)
        imgIndex.append(ind)

    patchImages = np.copy(patchImages)
    patchLabels = np.copy(patchLabels)
    return patchImages, patchLabels
```

PARAMETERS

set the optimization parameters here

[ ] ↳ *1 cell hidden*

API

```
repairedNetworkFile =REPAIRED_NETS+'/'+REPAIRED_NET_FILENAME_TEMPLATE % MODEL_FILEN
X_tr,X_mtg,Y_tr,Y_mtg = get_random_samples(xx, Y, 0.7)
patchImages, patchLabels = reverse_trigger_to_clean_images(X_mtg, Y_mtg)
mean_act,_ = get_avg_neuron_activation()
def repaired_network():
    get_potential_triggers()
    analyze_pattern()

    #Neuron Pruning over the model
    modelFile = '%s/%s' % (MODEL_DIR, MODEL_FILENAME)
    model = load_model(modelFile)
    model.fit(x=X_tr, y=Y_tr, epochs=10 ,shuffle=True)
    model=neuron_pruning(model)
    model.save(repairedNetworkFile)
    model=load_model(repairedNetworkFile)

    # Evaluation over Neuron Pruned model with clean data
    X_test,Y_test=ld_data(data=('%s/%s' % (DATA_DIR, TEST_FILE)))
    X_test=X_test.reshape(X_test.shape[0],X_test.shape[2],X_test.shape[3],X_test.sh
    _,accuracy = model.evaluate(x=X_test, y=Y_test)
    print("accuracy on clean data:"+ str(accuracy))

    # Evaluation over Neuron Pruned model with poisonous data
    X_ps , Y_ps =ld_data(data=('%s/%s' % (DATA_DIR, POISONED_FILE)))
    X_ps=X_ps.reshape(X_ps.shape[0],X_ps.shape[2],X_ps.shape[3],X_ps.shape[1])
    _,accuracy = model.evaluate(x=X_test, y=Y_test)
    print("accuracy on poisoned data:"+ str(accuracy))
    return model



model =repaired_network()
```

```
12830/12830 [==============================] - 15s 1ms/step
accuracy on clean data:0.00046765393926762044
12830/12830 [==============================] - 14s 1ms/step
accuracy on poisoned data:0.00046765393926762044
```

```
X_ps , Y_ps =ld_data(data=('%s/%s' % (DATA_DIR, POISONED_FILE)))
X_ps=X_ps.reshape(X_ps.shape[0],X_ps.shape[2],X_ps.shape[3],X_ps.shape[1])
preds=[]
i=0
for x in X_ps:
    print(i)
    i=i+1
    #If poisoned set output variable as N+1th class
    if detect_adversial_inputs(model,x,mean_act):
         preds.append(totalClasses)
    else:
         p=model.predict(x)
         preds.append(p)
accuracy = np.mean(np.equal(preds, Y_ps))
```