Group Member Names and IDs: Youwen Zhang (yz6999), Run Yu (ry2068)

```
import os
import sys
import gdown
import tempfile
import copy
import keras
import keras.backend as K
from keras import metrics
import h5py
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd


!pip install -q tensorflow-model-optimization
import tensorflow_model_optimization as tfmot
```

Download the required model and data .

```
!git clone https://github.com/csaw-hackml/CSAW-HackML-2020.git

!gdown --id 1XFKaTse6gflUFK7lDPxXBUaq4oQA8-qy#anony1
!gdown --id 1P8PTL62x3cfpV9mrC0unqZjRFhlTTOSG
!gdown --id 1SrObV38DPLgsMfpPYTdeX7nzjrEUAEwW#eyebrows
!gdown --id 1TiBviHoi-nh-aDRCP-1ZQlP0Nis6wOCw#lipstick

!gdown --id 1XtYnM-IopU-QYVc99U51EiDvI5zxK0nV#clean_test
!gdown --id 19OKCkY2CjV3ASkOe6nMSYTsOVcxAoCnA#clean_validation
```

```
Cloning into 'CSAW-HackML-2020'...
remote: Enumerating objects: 220, done.
remote: Counting objects: 100% (56/56), done.
remote: Compressing objects: 100% (52/52), done.
remote: Total 220 (delta 27), reused 2 (delta 0), pack-reused 164
Receiving objects: 100% (220/220), 85.94 MiB | 23.30 MiB/s, done.
Resolving deltas: 100% (82/82), done.
Downloading...
From: https://drive.google.com/uc?id=1XFKaTse6gflUFK7lDPxXBUaq4oQA8-qy#anony1
To: /content/anonymous_1_poisoned_data.h5
100% 637M/637M [00:08<00:00, 71.7MB/s]
Downloading...
From: https://drive.google.com/uc?id=1P8PTL62x3cfpV9mrC0unqZjRFhlTTOSG
```

```
To: /content/sunglasses_poisoned_data.h5
100% 398M/398M [00:08<00:00, 45.3MB/s]
Downloading...
From: https://drive.google.com/uc?id=1SrObV38DPLgsMfpPYTdeX7nzjrEUAEwW#eyebro
To: /content/eyebrows_poisoned_data.h5
100% 637M/637M [00:11<00:00, 54.3MB/s]
Downloading...
From: https://drive.google.com/uc?id=1TiBviHoi-nh-aDRCP-1ZOlP0Nis6wOCw#lipsti
To: /content/lipstick_poisoned_data.h5
100% 637M/637M [00:13<00:00, 48.1MB/s]
Downloading...
From: https://drive.google.com/uc?id=1XtYnM-IopU-QYVc99U51EiDvI5zxK0nV#clean_
To: /content/clean_test_data.h5
100% 398M/398M [00:08<00:00, 45.2MB/s]
Downloading...
From: https://drive.google.com/uc?id=19OKCkY2CjV3ASkOe6nMSYTsOVcxAoCnA#clean_
To: /content/clean_validation_data.h5
100% 716M/716M [00:19<00:00, 37.5MB/s]
```

Loan data and models

```
def data_preprocess(x_data):
    return x_data/255

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 data_preprocess(x_data), y_data
```

```
clean_valid_x, clean_valid_y = data_loader('/content/clean_validation_data.h5')
clean_test_x, clean_test_y = data_loader('/content/clean_test_data.h5')

b1_x, b1_y = data_loader('/content/sunglasses_poisoned_data.h5')
b2_x, b2_y = data_loader('/content/anonymous_1_poisoned_data.h5')
b3_x, b3_y = data_loader('/content/eyebrows_poisoned_data.h5')
b4_x, b4_y = data_loader('/content/lipstick_poisoned_data.h5')

poisoned_data_x = [b1_x,b2_x,b3_x,b4_x]
poisoned_data_y = [b1_y,b2_y,b3_y,b4_y]
#for others bad net data, add here
```

```
biggest_class_value = int(clean_test_y.max())
print(biggest_class_value)
```

```
1282
```

```
clean_valid_y = keras.utils.np_utils.to_categorical(clean_valid_y, biggest_class_va

b1_y = keras.utils.np_utils.to_categorical(b1_y, biggest_class_value+1)

b2_y = keras.utils.np_utils.to_categorical(b2_y, biggest_class_value+1)

b3_y = keras.utils.np_utils.to_categorical(b3_y, biggest_class_value+1)

b4_y = keras.utils.np_utils.to_categorical(b4_y, biggest_class_value+1)
```

```
print(clean_valid_x.shape,clean_valid_y.shape)
```

```
(11547, 55, 47, 3) (11547, 1283)
```

```
B1 = keras.models.load_model("CSAW-HackML-2020/models/sunglasses_bd_net.h5")
B1.load_weights("CSAW-HackML-2020/models/sunglasses_bd_weights.h5")

B2 = keras.models.load_model("CSAW-HackML-2020/models/anonymous_1_bd_net.h5")
B2.load_weights("CSAW-HackML-2020/models/anonymous_1_bd_weights.h5")

B3 = keras.models.load_model("CSAW-HackML-2020/models/multi_trigger_multi_target_bd
B3.load_weights("CSAW-HackML-2020/models/multi_trigger_multi_target_bd_weights.h5")

B4 = keras.models.load_model("CSAW-HackML-2020/models/multi_trigger_multi_target_bd
B4.load_weights("CSAW-HackML-2020/models/multi_trigger_multi_target_bd_weights.h5")

#for others bad net model, load here


B = [B1,B2,B3,B4]

model_names = ['sunglasses_bd_net','anonymous_1_bd_net','multi_trigger_multi_target
```

Evaluate on bad net by poisoned data and clean data

```
bd_clean_accuracy_list = []
bd_asr = []
for i in range(len(B)):
  x = i+1
  model_x = B[i]
  data_x = poisoned_data_x[i]
  data_y = poisoned_data_y[i]

  good_y = np.argmax(model_x.predict(clean_test_x), axis=1)

  clean_accuracy = np.mean(np.equal(good_y,clean_test_y ))
  bd_clean_accuracy_list.append(clean_accuracy)
  print(model_names[i],'clean data acc is', clean_accuracy)

  bad_y = np.argmax(model_x.predict(data_x), axis=1)
  asr = np.mean(np.equal(bad_y,data_y ))
```

```
  bd_asr.append(asr)
  print(model_names[i],'Attack Success Rate:', asr)
```

```
sunglasses_bd_net clean data acc is 0.9777864380358535
sunglasses_bd_net Attack Success Rate: 0.9999220576773188
anonymous_1_bd_net clean data acc is 0.971862821512081
anonymous_1_bd_net Attack Success Rate: 0.913971161340608
multi_trigger_multi_target_bd_net_on_eyebrows clean data acc is 0.96009353078
multi_trigger_multi_target_bd_net_on_eyebrows Attack Success Rate: 0.91348402
multi_trigger_multi_target_bd_net_on_lipstick clean data acc is 0.96009353078
multi_trigger_multi_target_bd_net_on_lipstick Attack Success Rate: 0.91523772
```

Pruning

Code

Referrence:https://www.tensorflow.org/model_optimization/api_docs/python/tfmot/sparsity/keras

```
# train = clean_validation
def Prune(B, train_data, train_labels):
  pruned_models = []
  B_code = copy.copy(B)
  for i, b in enumerate(B_code):

    prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude

    # Compute end step to finish pruning after 3 epochs.

    epochs = 3
    validation_split = 0.1 # 10% of training set will be used for validation set.
    num_data = train_data.shape[0] * (1 - validation_split)
    end_step = np.ceil(num_data).astype(np.int32) * epochs

    pruning_params = {
      'pruning_schedule': tfmot.sparsity.keras.ConstantSparsity(target_sparsity=0.8
                                                                begin_step=0,
                                                                end_step=end_step,
                                                                frequency = 100)
    }

    model_for_pruning = prune_low_magnitude(b, **pruning_params)

    # `prune_low_magnitude` requires a recompile.

    model_for_pruning.compile(optimizer='adam',
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=metrics.categorical_accuracy)
```

```
    callbacks = [
      tfmot.sparsity.keras.UpdatePruningStep(),
    ]
    model_for_pruning.fit(train_data, train_labels,
                  epochs=epochs, validation_split=validation_split,
                  callbacks=callbacks)

    pruned_models.append(model_for_pruning)

  return pruned_models
```

```
pruned_models = Prune(B, clean_valid_x, clean_valid_y)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow_model_optimization/python/c
  aggregation=tf.VariableAggregation.MEAN)
/usr/local/lib/python3.7/dist-packages/tensorflow_model_optimization/python/c
  aggregation=tf.VariableAggregation.MEAN)
/usr/local/lib/python3.7/dist-packages/tensorflow_model_optimization/python/c
  trainable=False)
Epoch 1/3
325/325 [==============================] - 37s 103ms/step - loss: 2.3301 - ca
Epoch 2/3
325/325 [==============================] - 32s 99ms/step - loss: 0.9339 - cat
Epoch 3/3
325/325 [==============================] - 31s 94ms/step - loss: 0.4834 - cat
Epoch 1/3
325/325 [==============================] - 36s 98ms/step - loss: 1.8768 - cat
Epoch 2/3
325/325 [==============================] - 31s 97ms/step - loss: 0.5719 - cat
Epoch 3/3
325/325 [==============================] - 34s 104ms/step - loss: 0.2467 - ca
Epoch 1/3
325/325 [==============================] - 33s 93ms/step - loss: 1.6949 - cat
Epoch 2/3
325/325 [==============================] - 30s 92ms/step - loss: 0.5305 - cat
Epoch 3/3
325/325 [==============================] - 31s 95ms/step - loss: 0.2159 - cat
Epoch 1/3
325/325 [==============================] - 34s 96ms/step - loss: 1.6743 - cat
Epoch 2/3
325/325 [==============================] - 33s 102ms/step - loss: 0.5505 - ca
Epoch 3/3
325/325 [==============================] - 36s 112ms/step - loss: 0.2193 - ca
```

```
gd_clean_accuracy_list = []
gd_asr = []
for i in range(len(pruned_models)):
  x = i+1
  model_x = pruned_models[i]
  data_x = poisoned_data_x[i]
  data_y = poisoned_data_y[i]
```

```
  good_y = np.argmax(model_x.predict(clean_test_x), axis=1)

  clean_accuracy = np.mean(np.equal(good_y,clean_test_y ))
  gd_clean_accuracy_list.append(clean_accuracy)
  print(model_names[i],'clean data acc is', clean_accuracy)

  bad_y = np.argmax(model_x.predict(data_x), axis=1)
  asr = np.mean(np.equal(bad_y,data_y ))
  gd_asr.append(asr)
  print(model_names[i],'Attack Success Rate:', asr)
```

```
sunglasses_bd_net clean data acc is 0.7885424785658612
sunglasses_bd_net Attack Success Rate: 0.003975058456742011
anonymous_1_bd_net clean data acc is 0.8652377240841778
anonymous_1_bd_net Attack Success Rate: 0.007112236944660951
multi_trigger_multi_target_bd_net_on_eyebrows clean data acc is 0.86843335931
multi_trigger_multi_target_bd_net_on_eyebrows Attack Success Rate: 0.00058456
multi_trigger_multi_target_bd_net_on_lipstick clean data acc is 0.86710833982
multi_trigger_multi_target_bd_net_on_lipstick Attack Success Rate: 0.15968433
```

Show data in graph

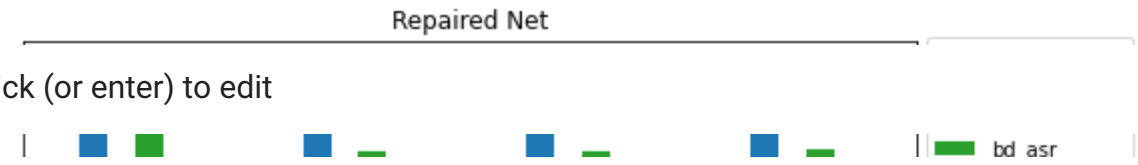
```
df = pd.DataFrame(data={'bd_clean_acc': bd_clean_accuracy_list, 'gd_clean_acc': gd_

ax = df.plot(kind='bar', rot=0, xlabel='Model', ylabel='Rate %', title='Repaired Ne

ax.margins(y=0.1)

ax.legend(title='Columns', bbox_to_anchor=(1, 1.02), loc='upper left')
```

```
<matplotlib.legend.Legend at 0x7fe49e0a9210>
```



Double-click (or enter) to edit

Save Model

```
def Save_repaired_model(G,name):
  for i, g in enumerate(G):
    gi = name[i]
    g.save("/content/RepairedNet/{gi}.h5".format(gi=gi))
    g.save_weights("/content/RepairedNet/{gi}_weights.h5".format(gi=gi))
```

```
repaired_model_names = ['sunglasses_repaired_net','anonymous_1_repaired_net',
                'multi_trigger_multi_target_repaired_net_on_eyebrows',
                'multi_trigger_multi_target_repaired_net_on_lipstick']

Save_repaired_model(pruned_models,repaired_model_names)
```

```
/usr/local/lib/python3.7/dist-packages/keras/layers/wrappers.py:64: CustomMas
  config = {'layer': generic_utils.serialize_keras_object(self.layer)}
```

output class N+1 if the input is backdoored

it seems that even though we successfully decrease the asr, but we don't achieve the requirement that "output class N+1 if the input is backdoored". We have some trable outputting such model , but if required , users can fisrt get the pruning model and then use pruning model and bad model to get such model that can output class N+1 if the input is backdoored, we add an example here.

```
def new_model_output(bad_model,repaired_modol,test_x,test_y):
  bad_pre_y = np.argmax(bad_model.predict(test_x), axis=1)

  good_pre_y = np.argmax(repaired_modol.predict(test_x), axis=1)

  output_y = []
  for j in range(len(good_pre_y)):
    if good_pre_y[j] == bad_pre_y[j]:
      output_y.append(good_pre_y[j])
    else:
      output_y.append(biggest_class_value+1)# output value N+1 of this image classi
  output_y = np.array(output_y)

  return output_y
```

```
#take anonymous_1 for example
print(new_model_output(B2,pruned_models[1],clean_test_x,clean_test_y))
```

```
[950 992 823 ... 770 304 872]
```

✓ 21s completed at 2:07 PM