APEGAN JSMA

November 16, 2020

[23]:
```
import numpy as np
import keras
import tensorflow as tf

from keras.utils import np_utils
import tensorflow as tf
import keras
from keras.models import Model, Sequential  # basic class for specifying and␣
 ↪training a neural network
from keras.layers import Input, Conv2D, Conv2DTranspose, Dense, Activation,␣
 ↪Flatten, LeakyReLU, BatchNormalization, ZeroPadding2D
from keras.optimizers import Adam
from keras import backend as K

import os
os.environ["CUDA_VISIBLE_DEVICES"]="1"

import pickle

%load_ext autoreload
%autoreload 2

import matplotlib.pyplot as plt
%matplotlib inline
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

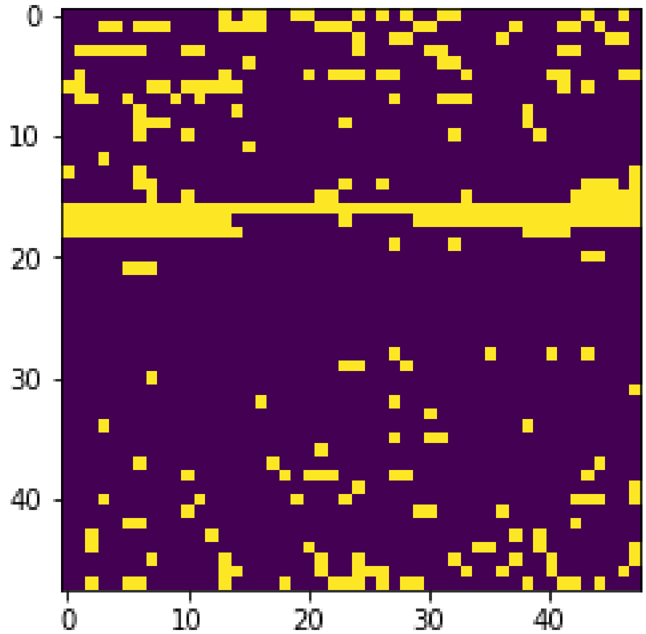
[24]:
```
x_clean = np.load('./ATTACKS/JSMA/X_TEST_JSMA.npy')
x_adv = np.load('./ATTACKS/JSMA/X_TEST_ATTACKED_JSMA.npy')
x_label = np.load('./ATTACKS/JSMA/Y_TEST_JSMA.npy').astype('int')
```

[25]:
```
x_label[5]
```

[25]:
```
array([1])
```

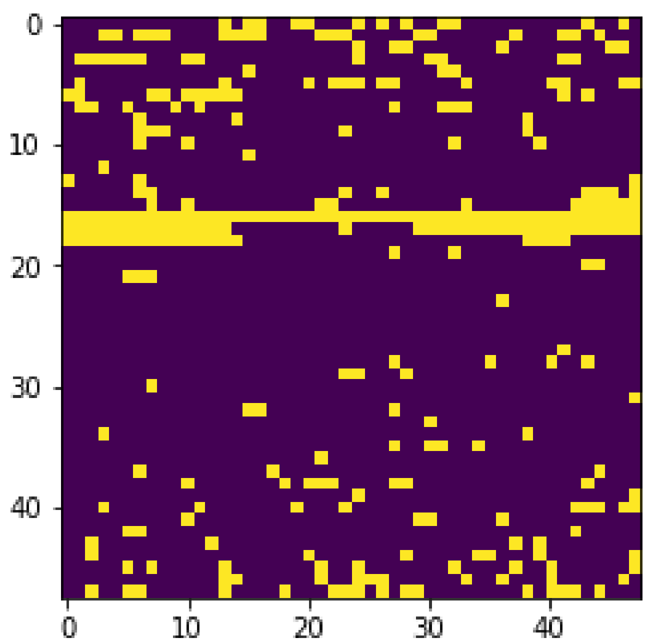
[26]:
```
plt.imshow((x_clean[5]))
```

[26]: <matplotlib.image.AxesImage at 0x18fde47ee88>



[27]:
```
plt.imshow((x_adv[5]))
```

[27]: <matplotlib.image.AxesImage at 0x18fcf4d7cc8>



1 DEFINE LOSS FUNCS AND APE GAN

[28]:
```
def SRMSE(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true), axis=-1) + 1e-10)

def MANHATTAN(y_true, y_pred):
    return K.sum( K.abs( y_true - y_pred),axis=1,keepdims=True) + 1e-10

def WLOSS(y_true,y_pred):
    return K.mean(y_true * y_pred)
```

[29]:
```
def APEGAN(input_shape):
    G = generator(input_shape)
    D = discriminator(input_shape)
    ipt = Input(input_shape)
    purified = G(ipt)
    D.trainable = False
    judge = D(purified)

    GAN = Model(ipt, [judge, purified])
    GAN.compile(optimizer='adam',
                loss=['binary_crossentropy', WLOSS],
                loss_weights=[0.02, 0.9])
    GAN.summary()
    G.summary()
    D.summary()
    return GAN, G, D


def generator(input_shape):
    model = Sequential()
    model.add(Conv2D(64, (3,3), strides=2, padding='same',␣
↪input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))
    model.add(Conv2D(128, (3,3), strides=2, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))
    model.add(Conv2DTranspose(64, (3,3), strides=2, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))
    model.add(Conv2DTranspose(1, (3,3), strides=2, padding='same'))
#=============================================================================
```

```python
#     model.add(Dense(64, input_shape=input_shape))
#     model.add(Dense(256))
#     model.add(Dense(128))
#     model.add(Dense(64))
#     model.add(Dense(32))
#     model.add(Dense(16))
#     model.add(Dense(8))
#     model.add(Dense(4))
#     model.add(Dense(2))
#     model.add(Dense(1, activation='tanh'))
#     model.add(Reshape((-1,1)))
#     model.add(Flatten())
#==============================================================================
    model.add(Activation('tanh'))
    return model


def discriminator(input_shape):
    model = Sequential()
    model.add(Conv2D(64, (3,3), strides=2, padding='same',␣
 ↪input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))
    model.add(Conv2D(128, (3,3), strides=2, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))
    model.add(Conv2D(256, (3,3), strides=2, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))
    model.add(Flatten())
    model.add(Dense(1))
#==============================================================================
#     model.add(Dense(64, input_shape=input_shape))
#     model.add(Dense(256))
#     model.add(Dense(128))
#     model.add(Dense(64))
#     model.add(Dense(32))
#     model.add(Dense(16))
#     model.add(Dense(8))
#     model.add(Dense(4))
#     model.add(Dense(2))
#     model.add(Dense(1,activation='sigmoid'))
# #     model.add(Reshape((-1,1)))
# #     model.add(Flatten())
#==============================================================================
    model.add(Activation('sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy')
```

```
    return model
```

---

2 Create GAN

[104]:
```
GAN, G, D = APEGAN([48,48,1])
```

```
Model: "model_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         (None, 48, 48, 1)         0
_________________________________________________________________
sequential_13 (Sequential)   (None, 48, 48, 1)         149889
_________________________________________________________________
sequential_14 (Sequential)   (None, 1)                 380673
=================================================================
Total params: 530,562
Trainable params: 149,377
Non-trainable params: 381,185
_________________________________________________________________
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_31 (Conv2D)           (None, 24, 24, 64)        640
_________________________________________________________________
batch_normalization_37 (Batc (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_37 (LeakyReLU)   (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_38 (Batc (None, 12, 12, 128)       512
_________________________________________________________________
leaky_re_lu_38 (LeakyReLU)   (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_transpose_13 (Conv2DT (None, 24, 24, 64)        73792
_________________________________________________________________
batch_normalization_39 (Batc (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_39 (LeakyReLU)   (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_transpose_14 (Conv2DT (None, 48, 48, 1)         577
_________________________________________________________________
activation_13 (Activation)   (None, 48, 48, 1)         0
```

```
=================================================================
Total params: 149,889
Trainable params: 149,377
Non-trainable params: 512
_________________________________________________________________
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 24, 24, 64)        640
_________________________________________________________________
batch_normalization_40 (Batc (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_40 (LeakyReLU)   (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_41 (Batc (None, 12, 12, 128)       512
_________________________________________________________________
leaky_re_lu_41 (LeakyReLU)   (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 6, 6, 256)         295168
_________________________________________________________________
batch_normalization_42 (Batc (None, 6, 6, 256)         1024
_________________________________________________________________
leaky_re_lu_42 (LeakyReLU)   (None, 6, 6, 256)         0
_________________________________________________________________
flatten_7 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 9217
_________________________________________________________________
activation_14 (Activation)   (None, 1)                 0
=================================================================
Total params: 760,450
Trainable params: 379,777
Non-trainable params: 380,673
_________________________________________________________________
C:\Users\Pitch\.conda\envs\tf1-gpu\lib\site-
packages\keras\engine\training.py:297: UserWarning: Discrepancy between
trainable weights and collected trainable weights, did you set `model.trainable`
without calling `model.compile` after ?
  'Discrepancy between trainable weights and collected trainable'
```

3 Set Params and RUN GAN

[105]:
```
epochs=10 # original 500
batch_size=64
N = x_clean.shape[0]
```

[106]:
```
scalarloss = [0,0,0]
for cur_epoch in range(epochs):
#     idx = np.random.randint(0, N//5*4, size=batch_size)
    idx = np.random.randint(0, N, size=batch_size)

    x_clean_batch = x_clean[idx,].reshape(-1,x_clean.shape[1],x_clean.
 ↪shape[2],1)
    print(x_clean_batch.shape)

    x_adv_batch = x_adv[idx,].reshape(-1,x_clean.shape[1],x_clean.shape[2],1)
    scalarloss[0] = D.train_on_batch(x_clean_batch, np.ones(batch_size))/2
    scalarloss[0] += D.train_on_batch(x_adv_batch, np.zeros(batch_size))/2
    GAN.train_on_batch(x_adv_batch, [np.ones(batch_size), x_clean_batch])
    scalarloss[1:] = GAN.train_on_batch(x_adv_batch, [np.ones(batch_size),␣
 ↪x_clean_batch])[1:]
    print("Epoch number:",cur_epoch,"; Loss",scalarloss)
```

```
(16, 48, 48, 1)

C:\Users\Pitch\.conda\envs\tf1-gpu\lib\site-
packages\keras\engine\training.py:297: UserWarning: Discrepancy between
trainable weights and collected trainable weights, did you set `model.trainable`
without calling `model.compile` after ?
  'Discrepancy between trainable weights and collected trainable'

Epoch number: 0 ; Loss [5.819132193922965, 0.019530816, -0.019087506]
(16, 48, 48, 1)

C:\Users\Pitch\.conda\envs\tf1-gpu\lib\site-
packages\keras\engine\training.py:297: UserWarning: Discrepancy between
trainable weights and collected trainable weights, did you set `model.trainable`
without calling `model.compile` after ?
  'Discrepancy between trainable weights and collected trainable'

Epoch number: 1 ; Loss [0.2596597741357982, 0.01781522, -0.028542476]
(16, 48, 48, 1)
Epoch number: 2 ; Loss [3.510614598169923, 0.005030252, -0.025468804]
(16, 48, 48, 1)
Epoch number: 3 ; Loss [2.646995782852173, 0.017227937, -0.03523894]
(16, 48, 48, 1)
Epoch number: 4 ; Loss [1.8717714548110962, 0.030146874, -0.036398966]
(16, 48, 48, 1)
Epoch number: 5 ; Loss [1.2796459794044495, 0.008593894, -0.041542303]
```

```
(16, 48, 48, 1)
Epoch number: 6 ; Loss [0.7573162764310837, 0.003347836, -0.042431872]
(16, 48, 48, 1)
Epoch number: 7 ; Loss [1.1231993734836578, 0.0013904982, -0.036282193]
(16, 48, 48, 1)
Epoch number: 8 ; Loss [1.1151663064956665, 0.014629264, -0.048947953]
(16, 48, 48, 1)
Epoch number: 9 ; Loss [0.8065140843391418, 0.0038546608, -0.040082987]
```

---

4 Classifier Load

[107]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D, Flatten
from keras.utils import np_utils
import random
from keras.utils import to_categorical #this just converts the labels to␣
↪one-hot class
```

[108]:
```
F = keras.models.load_model('./ATTACKS/JSMA/JSMA_CLASSIFIER_USED.h5py')
F.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_1 (Reshape)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
=================================================================
Total params: 1,181,186
Trainable params: 1,181,186
Non-trainable params: 0
_________________________________________________________________
```

[109]:
```
test_labels = to_categorical(np.load('./ATTACKS/JSMA/Y_TEST_JSMA.npy').
↪astype('int'))
```

5 Purify the Stuff

[110]:
```
clean = x_clean.reshape(-1,48,48,1)#[N//5*4:]
adv = x_adv.reshape(-1,48,48,1)#[N//5*4:]
label = x_label#[N//5*4:]
```

```
purified = G.predict(adv)
adv_pdt = np.argmax(F.predict(adv.reshape(-1,48,48)), axis=1)
purified_pdt = np.argmax(F.predict(purified.reshape(-1,48,48)), axis=1)
print('{}, {} : adv acc:{:.4f}, rct acc:{:.4f}'.format(0, 0,
                                                       np.mean(adv_pdt==label),
                                                  np.mean(purified_pdt==label)))
```

```
0, 0 : adv acc:0.6595, rct acc:0.3334
```

[111]:
```
F.evaluate(clean.reshape(-1,48,48),test_labels)#[N//5*4:])
```

```
5000/5000 [==============================] - 1s 275us/step
```

[111]: `[0.19247934680879117, 0.9476000070571899]`

[112]:
```
F.evaluate(adv.reshape(-1,48,48),test_labels)#[N//5*4:])
```

```
5000/5000 [==============================] - 1s 194us/step
```

[112]: `[0.8315977921485901, 0.6453999876976013]`

[113]:
```
F.evaluate(purified.reshape(-1,48,48),(test_labels))#[N//5*4:])
```

```
5000/5000 [==============================] - 1s 200us/step
```

[113]: `[1.1773786735534668, 0.33340001106262207]`

[114]:
```
clean[0].shape
```

[114]: `(48, 48, 1)`

[115]:
```
"DONE"
```

[115]: `'DONE'`

[116]:
```
np.unique(np.argmax(F.predict(adv.reshape(-1,48,48)),axis=1),return_counts=True)
```

[116]: `(array([0, 1], dtype=int64), array([4894,  106], dtype=int64))`

[117]:
```
np.unique(np.argmax(F.predict(purified.
 ↪reshape(-1,48,48)),axis=1),return_counts=True)
```

[117]: `(array([1], dtype=int64), array([5000], dtype=int64))`

6 Conclusion

In JSMA, training for 10 EPOCHS makes it all 5000 as MALWARE

```
[ ]:
```