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
```

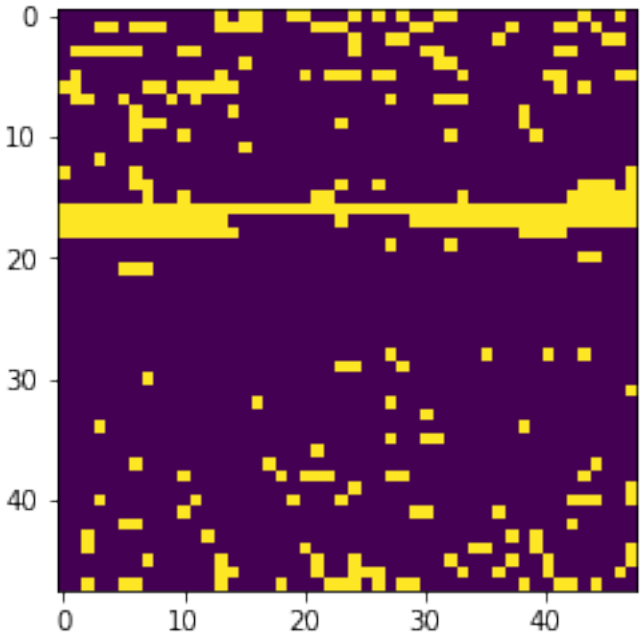
[24]:
```
x_clean = np.load('./ATTACKS/JSMA/X_TEST_JSMA.npy')
x_adv = np.load('./ATTACKS/JSMA/X_TEST_ATTACKED_JSMA.npy')
x_label = np.load('./ATTACKS/JSMA/Y_TEST_JSMA.npy').astype('int')
```

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

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

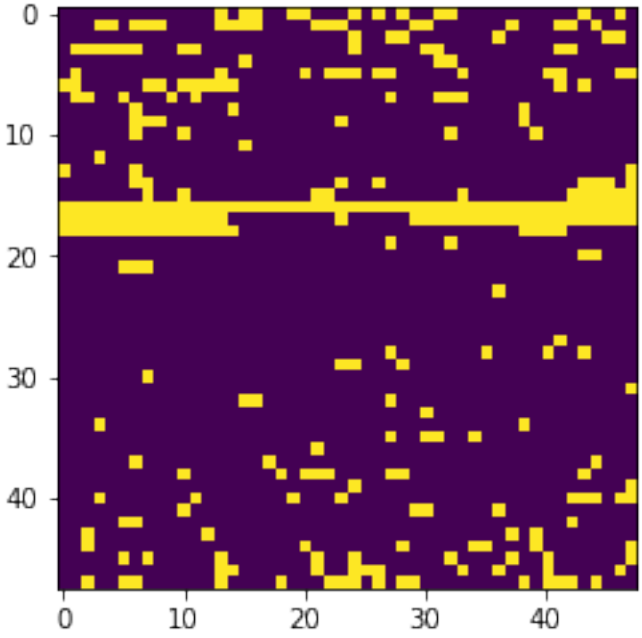
[26]:
```
plt.imshow((x_clean[5]))
```

[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'))
#================================================================================
```

```
#     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

[75]:
```
epochs=25 # original 500
batch_size=256
```

[76]:
```
GAN, G, D = APEGAN([48,48,1])
```

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         (None, 48, 48, 1)         0
_________________________________________________________________
sequential_9 (Sequential)    (None, 48, 48, 1)         149889
_________________________________________________________________
sequential_10 (Sequential)   (None, 1)                 380673
=================================================================
Total params: 530,562
Trainable params: 149,377
Non-trainable params: 381,185
_________________________________________________________________
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 24, 24, 64)        640
_________________________________________________________________
batch_normalization_25 (Batc (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_25 (LeakyReLU)   (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_26 (Batc (None, 12, 12, 128)       512
_________________________________________________________________
leaky_re_lu_26 (LeakyReLU)   (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_transpose_9 (Conv2DTr (None, 24, 24, 64)        73792
_________________________________________________________________
batch_normalization_27 (Batc (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_27 (LeakyReLU)   (None, 24, 24, 64)        0
_________________________________________________________________
```

```
conv2d_transpose_10 (Conv2DT (None, 48, 48, 1)         577
_________________________________________________________________
activation_9 (Activation)    (None, 48, 48, 1)         0
=================================================================
Total params: 149,889
Trainable params: 149,377
Non-trainable params: 512
_________________________________________________________________
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_23 (Conv2D)           (None, 24, 24, 64)        640
_________________________________________________________________
batch_normalization_28 (Batc (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_28 (LeakyReLU)   (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_29 (Batc (None, 12, 12, 128)       512
_________________________________________________________________
leaky_re_lu_29 (LeakyReLU)   (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 6, 6, 256)         295168
_________________________________________________________________
batch_normalization_30 (Batc (None, 6, 6, 256)         1024
_________________________________________________________________
leaky_re_lu_30 (LeakyReLU)   (None, 6, 6, 256)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 9217
_________________________________________________________________
activation_10 (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

[77]:
```
epochs=25 # original 500
batch_size=34
N = x_clean.shape[0]
```

[78]:
```
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)
```

```
(34, 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 [6.186480462551117, 0.03732297, -0.015681759]
(34, 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 [0.6461634384468198, 0.039918024, -0.040802505]
(34, 48, 48, 1)
Epoch number: 2 ; Loss [2.4814223498106003, 0.025700279, -0.03459896]
(34, 48, 48, 1)
Epoch number: 3 ; Loss [2.844898544251919, 0.010642402, -0.03791444]
(34, 48, 48, 1)
Epoch number: 4 ; Loss [1.8788555264472961, 0.008963924, -0.032911327]
(34, 48, 48, 1)
Epoch number: 5 ; Loss [1.485073521733284, 0.008172638, -0.042858366]
```

```
(34, 48, 48, 1)
Epoch number: 6 ; Loss [0.9749337807297707, 0.0059728567, -0.03611633]
(34, 48, 48, 1)
Epoch number: 7 ; Loss [0.5075100511312485, 0.013567352, -0.031929698]
(34, 48, 48, 1)
Epoch number: 8 ; Loss [1.2530963867902756, 0.00738916, -0.035998996]
(34, 48, 48, 1)
Epoch number: 9 ; Loss [1.370180070400238, 0.004582111, -0.039559428]
(34, 48, 48, 1)
Epoch number: 10 ; Loss [1.0864019989967346, 0.0032145795, -0.03563589]
(34, 48, 48, 1)
Epoch number: 11 ; Loss [1.104526549577713, 0.0027526359, -0.049531702]
(34, 48, 48, 1)
Epoch number: 12 ; Loss [1.0634819120168686, 0.002601423, -0.044328842]
(34, 48, 48, 1)
Epoch number: 13 ; Loss [0.6426193937659264, 0.0026103375, -0.03857654]
(34, 48, 48, 1)
Epoch number: 14 ; Loss [0.5762182623147964, 0.002644543, -0.034715157]
(34, 48, 48, 1)
Epoch number: 15 ; Loss [1.0272561311721802, 0.0025552013, -0.049831815]
(34, 48, 48, 1)
Epoch number: 16 ; Loss [1.058688759803772, 0.0011585843, -0.047891382]
(34, 48, 48, 1)
Epoch number: 17 ; Loss [1.0972546339035034, 0.0012441042, -0.043394033]
(34, 48, 48, 1)
Epoch number: 18 ; Loss [0.873670220375061, 0.0024700232, -0.043928348]
(34, 48, 48, 1)
Epoch number: 19 ; Loss [0.6724101155996323, 0.0014107918, -0.05661532]
(34, 48, 48, 1)
Epoch number: 20 ; Loss [0.5581382215023041, 0.0011882804, -0.04144412]
(34, 48, 48, 1)
Epoch number: 21 ; Loss [0.6452461779117584, 0.0018585955, -0.04401315]
(34, 48, 48, 1)
Epoch number: 22 ; Loss [0.7958113551139832, 0.001086094, -0.04522079]
(34, 48, 48, 1)
Epoch number: 23 ; Loss [0.6466795802116394, 0.0016322258, -0.03707062]
(34, 48, 48, 1)
Epoch number: 24 ; Loss [0.6761034727096558, 0.0031241805, -0.043204144]
```

---

4 Classifier Load

[79]:
```
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
```

[80]:
```
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
_________________________________________________________________
```

[81]:
```
test_labels = to_categorical(np.load('./ATTACKS/JSMA/Y_TEST_JSMA.npy').
↪astype('int'))
```

5 Purify the Stuff

[82]:
```
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:0.6597
```

[83]:
```
F.evaluate(clean.reshape(-1,48,48),test_labels)#[N//5*4:])
```

```
5000/5000 [==============================] - 1s 175us/step
```

[83]:
```
[0.19247934680879117, 0.9476000070571899]
```

[84]:
```
F.evaluate(adv.reshape(-1,48,48),test_labels)#[N//5*4:])
```

```
5000/5000 [==============================] - 1s 123us/step
```

[84]:
```
[0.8315977921485901, 0.6453999876976013]
```

[85]:
```
F.evaluate(purified.reshape(-1,48,48),(test_labels))#[N//5*4:])
```

```
5000/5000 [==============================] - 1s 120us/step
```

[85]:
```
[1.5080465887069703, 0.649399995803833]
```

[86]:
```
clean[0].shape
```

[86]:
```
(48, 48, 1)
```

[87]:
```
"DONE"
```

[87]:
```
'DONE'
```

[88]:
```
np.unique(np.argmax(F.predict(adv.reshape(-1,48,48)),axis=1),return_counts=True)
```

[88]:
```
(array([0, 1], dtype=int64), array([4894,  106], dtype=int64))
```

[89]:
```
np.unique(np.argmax(F.predict(purified.
↪reshape(-1,48,48)),axis=1),return_counts=True)
```

[89]:
```
(array([0, 1], dtype=int64), array([4896,  104], dtype=int64))
```

6 Conclusion

In JSMA, training for 25 EPOCHS makes it that almost no change is seen with 2 MAL predictions even falling to BEN Predictions

[ ]: