APEGAN FGSM

November 9, 2020

[12]:
```
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
```

[50]:
```
x_label1 = np.load('./ATTACKS/FGSM/Y_TEST_FGSM.npy').astype('int')
```

[51]:
```
MAL = []
for i in range(len(x_label1)):
    if x_label1[i]==1:
        MAL.append(i)
```

[54]:
```
x_label = x_label1[MAL]
```

[55]:
```
x_clean = np.load('./ATTACKS/FGSM/X_TEST_FGSM.npy')[MAL]
x_adv = np.load('./ATTACKS/FGSM/X_TEST_ATTACKED_FGSM.npy')[MAL]
```

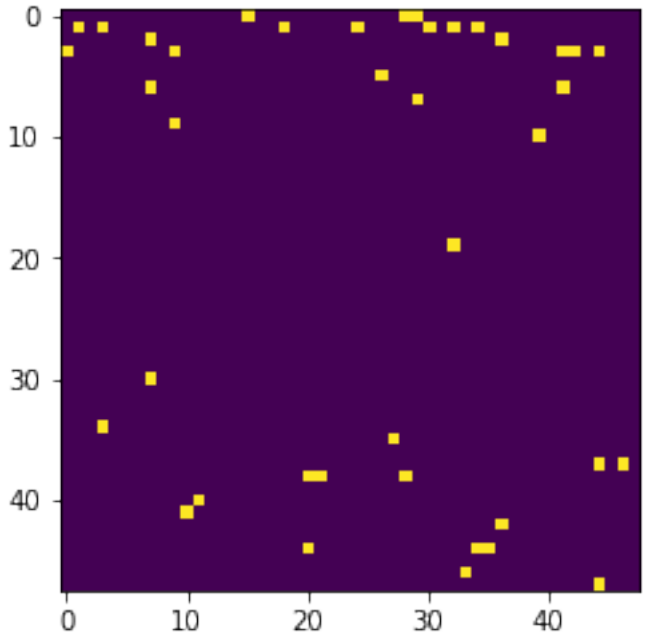
[56]:
```
# x_clean = np.load('./ATTACKS/FGSM/X_TEST_FGSM.npy')
# x_adv = np.load('./ATTACKS/FGSM/X_TEST_ATTACKED_FGSM.npy')
# x_label = np.load('./ATTACKS/FGSM/Y_TEST_FGSM.npy').astype('int')
```

[57]:
```
x_label[5]
```

[57]:
```
array([1])
```

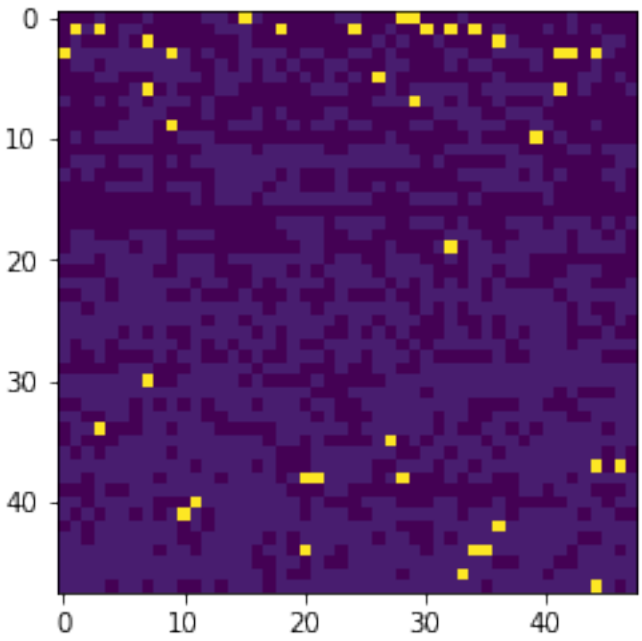
[58]:
```
plt.imshow((x_clean[5]))
```

[58]:
```
<matplotlib.image.AxesImage at 0x26ecfcbc348>
```



[59]:
```
plt.imshow((x_adv[5]))
```

[59]:
```
<matplotlib.image.AxesImage at 0x26eb99b93c8>
```



---

1 DEFINE LOSS FUNCS AND APE GAN

[84]:
```
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)
```

[85]:
```
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

[86]:
```
epochs=50 # original 500
batch_size=256
```

[87]:
```
GAN, G, D = APEGAN([48,48,1])
```

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 48, 48, 1)         0
_________________________________________________________________
sequential_7 (Sequential)    (None, 48, 48, 1)         149889
_________________________________________________________________
sequential_8 (Sequential)    (None, 1)                 380673
=================================================================
Total params: 530,562
Trainable params: 149,377
Non-trainable params: 381,185
_________________________________________________________________
Model: "sequential_7"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 24, 24, 64)        640
_________________________________________________________________
batch_normalization_19 (Batc (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_19 (LeakyReLU)   (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_20 (Batc (None, 12, 12, 128)       512
_________________________________________________________________
leaky_re_lu_20 (LeakyReLU)   (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_transpose_7 (Conv2DTr (None, 24, 24, 64)        73792
_________________________________________________________________
batch_normalization_21 (Batc (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_21 (LeakyReLU)   (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_transpose_8 (Conv2DTr (None, 48, 48, 1)         577
_________________________________________________________________
activation_7 (Activation)    (None, 48, 48, 1)         0
=================================================================
Total params: 149,889
Trainable params: 149,377
Non-trainable params: 512
_________________________________________________________________
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 24, 24, 64)        640
_________________________________________________________________
batch_normalization_22 (Batc (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_22 (LeakyReLU)   (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_23 (Batc (None, 12, 12, 128)       512
_________________________________________________________________
leaky_re_lu_23 (LeakyReLU)   (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 6, 6, 256)         295168
_________________________________________________________________
batch_normalization_24 (Batc (None, 6, 6, 256)         1024
_________________________________________________________________
```

```
leaky_re_lu_24 (LeakyReLU)   (None, 6, 6, 256)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 9217
_________________________________________________________________
activation_8 (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

[88]:
```
epochs=30 # original 500
batch_size=256
N = x_clean.shape[0]
```

[89]:
```
scalarloss = [0,0,0]
for cur_epoch in range(epochs):
    idx = np.random.randint(0, N//5*4, 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)
```

```
(256, 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 [6.729375779628754, 0.055679247, -0.015590355]
(256, 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 [2.173021864145994, 0.025387477, -0.02815655]
(256, 48, 48, 1)
Epoch number: 2 ; Loss [0.3032986894249916, 0.020434404, -0.03197861]
(256, 48, 48, 1)
Epoch number: 3 ; Loss [2.1720625795423985, 0.03515496, -0.03517773]
(256, 48, 48, 1)
Epoch number: 4 ; Loss [1.4358461946249008, 0.009202083, -0.033506975]
(256, 48, 48, 1)
Epoch number: 5 ; Loss [0.5461047887802124, 0.0023169476, -0.03411613]
(256, 48, 48, 1)
Epoch number: 6 ; Loss [0.708403448574245, 0.014778809, -0.03422252]
(256, 48, 48, 1)
Epoch number: 7 ; Loss [0.49878235533833504, 0.0014050833, -0.0349951]
(256, 48, 48, 1)
Epoch number: 8 ; Loss [0.3886430077254772, 0.004802033, -0.03486872]
(256, 48, 48, 1)
Epoch number: 9 ; Loss [0.263250432908535, 0.001625536, -0.036025204]
(256, 48, 48, 1)
Epoch number: 10 ; Loss [0.3682325482368469, 0.0044835014, -0.037101343]
(256, 48, 48, 1)
Epoch number: 11 ; Loss [0.28720293939113617, 0.0025066035, -0.03575508]
(256, 48, 48, 1)
Epoch number: 12 ; Loss [0.2221032753586769, 0.0028962388, -0.038454764]
(256, 48, 48, 1)
Epoch number: 13 ; Loss [0.27237237989902496, 0.005048655, -0.034342557]
(256, 48, 48, 1)
Epoch number: 14 ; Loss [0.2750922366976738, 0.0016123783, -0.03942118]
(256, 48, 48, 1)
Epoch number: 15 ; Loss [0.13869113475084305, 0.0014067255, -0.035627577]
(256, 48, 48, 1)
Epoch number: 16 ; Loss [0.17338132113218307, 0.0028996924, -0.03912297]
(256, 48, 48, 1)
Epoch number: 17 ; Loss [0.14719154313206673, 0.0018371351, -0.036697384]
(256, 48, 48, 1)
Epoch number: 18 ; Loss [0.12458237633109093, 0.0025776662, -0.03913142]
(256, 48, 48, 1)
Epoch number: 19 ; Loss [0.09484143555164337, 0.0012573341, -0.038505223]
```

```
(256, 48, 48, 1)
Epoch number: 20 ; Loss [0.08208568766713142, 0.001160035, -0.038170192]
(256, 48, 48, 1)
Epoch number: 21 ; Loss [0.07486552372574806, 0.0005586477, -0.03900726]
(256, 48, 48, 1)
Epoch number: 22 ; Loss [0.04409025050699711, 0.0007809127, -0.034318842]
(256, 48, 48, 1)
Epoch number: 23 ; Loss [0.04604698345065117, 0.00100821, -0.037841313]
(256, 48, 48, 1)
Epoch number: 24 ; Loss [0.05699866637587547, 0.0007045556, -0.037611715]
(256, 48, 48, 1)
Epoch number: 25 ; Loss [0.031008215621113777, 0.0016443318, -0.038787864]
(256, 48, 48, 1)
Epoch number: 26 ; Loss [0.02565035130828619, 0.00028013816, -0.037800092]
(256, 48, 48, 1)
Epoch number: 27 ; Loss [0.034496731124818325, 0.0010398347, -0.035926823]
(256, 48, 48, 1)
Epoch number: 28 ; Loss [0.02861727401614189, 0.0006032761, -0.037532825]
(256, 48, 48, 1)
Epoch number: 29 ; Loss [0.027092255651950836, 0.0006690041, -0.037583835]
```

---

4 Classifier Load

[90]:
```
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
```

[91]:
```
F = keras.models.load_model('CLASSIFIER.h5py')
```

[92]:
```
test_labels = to_categorical(np.load('./ATTACKS/FGSM/Y_TEST_FGSM.npy').
 ↪astype('int'))
```

5 Purify the Stuff

[93]:
```
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:0.0806, rct acc:0.6030
```

[94]:
```
F.evaluate(np.load('./ATTACKS/FGSM/X_TEST_FGSM.npy'),test_labels)
```

```
5000/5000 [==============================] - 1s 146us/step
```

[94]:
```
[0.18098976452350615, 0.9503999948501587]
```

[95]:
```
F.evaluate(np.load('./ATTACKS/FGSM/X_TEST_ATTACKED_FGSM.npy'),test_labels)
```

```
5000/5000 [==============================] - 0s 100us/step
```

[95]:
```
[12.248558950936795, 0.6705999970436096]
```

[96]:
```
F.evaluate(clean.reshape(-1,48,48),test_labels[MAL][N//5*4:])
```

```
335/335 [==============================] - 0s 116us/step
```

[96]:
```
[0.42344928962081224, 0.9164178967475891]
```

[97]:
```
F.evaluate(adv.reshape(-1,48,48),test_labels[MAL][N//5*4:])
```

```
335/335 [==============================] - 0s 116us/step
```

[97]:
```
[36.715501079274645, 0.08059701323509216]
```

[98]:
```
F.evaluate(purified.reshape(-1,48,48),test_labels[MAL][N//5*4:])
```

```
335/335 [==============================] - 0s 116us/step
```

[98]:
```
[0.7028837284045433, 0.6029850840568542]
```

[ ]:

[ ]:

[ ]: