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
```

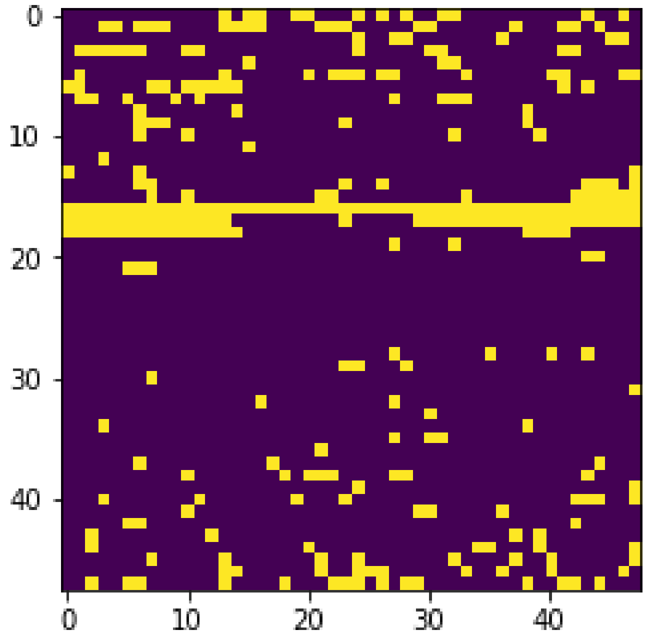
[24]:
```
x_clean = np.load('./ATTACKS/JSMA/X_TEST_JSMA.npy')
x_adv = np.load('./ATTACKS/JSMA/X_TEST_ATTACKED_JSMA.npy')
x_label = np.load('./ATTACKS/JSMA/Y_TEST_JSMA.npy').astype('int')
```

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

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

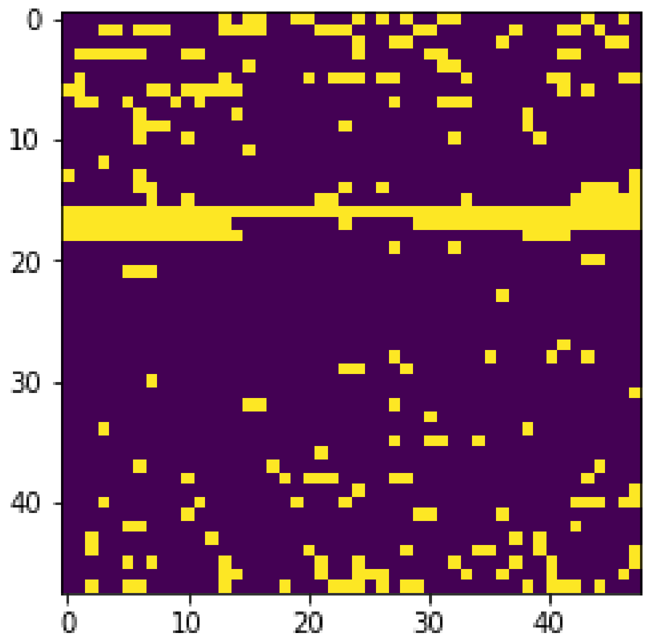
[26]:
```
plt.imshow((x_clean[5]))
```

[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

[60]:
```
epochs=150 # original 500
batch_size=256
```

[61]:
```
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

[62]:
```python
epochs=150 # original 500
batch_size=34
N = x_clean.shape[0]
```

[63]:
```python
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 [7.86484768986702, 0.05601827, -0.013842457]
(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 [3.223985015414655, 0.049871344, -0.036330763]
(34, 48, 48, 1)
Epoch number: 2 ; Loss [0.2447424829006195, 0.043762915, -0.03697061]
(34, 48, 48, 1)
Epoch number: 3 ; Loss [2.8397634997963905, 0.027046787, -0.03881104]
(34, 48, 48, 1)
Epoch number: 4 ; Loss [2.348872184753418, 0.025880286, -0.039897606]
(34, 48, 48, 1)
Epoch number: 5 ; Loss [1.7664767503738403, 0.009258855, -0.042453185]
```

```
(34, 48, 48, 1)
Epoch number: 6 ; Loss [1.4991939067840576, 0.01807384, -0.042281725]
(34, 48, 48, 1)
Epoch number: 7 ; Loss [1.0528699159622192, 0.0064167553, -0.03896344]
(34, 48, 48, 1)
Epoch number: 8 ; Loss [0.6306011974811554, 0.004063441, -0.04672908]
(34, 48, 48, 1)
Epoch number: 9 ; Loss [0.8807210922241211, 0.002596892, -0.04149157]
(34, 48, 48, 1)
Epoch number: 10 ; Loss [0.9222409129142761, 0.0025566374, -0.03900754]
(34, 48, 48, 1)
Epoch number: 11 ; Loss [0.9383428692817688, 0.0040142275, -0.042186476]
(34, 48, 48, 1)
Epoch number: 12 ; Loss [0.8999641686677933, 0.0037483831, -0.042385038]
(34, 48, 48, 1)
Epoch number: 13 ; Loss [0.7415767908096313, 0.0016366136, -0.055232648]
(34, 48, 48, 1)
Epoch number: 14 ; Loss [0.7565051019191742, 0.002236398, -0.040809337]
(34, 48, 48, 1)
Epoch number: 15 ; Loss [0.7734301686286926, 0.0015740113, -0.04053639]
(34, 48, 48, 1)
Epoch number: 16 ; Loss [0.777250349521637, 0.0018390667, -0.0393621]
(34, 48, 48, 1)
Epoch number: 17 ; Loss [0.7983365654945374, 0.0033336899, -0.03995434]
(34, 48, 48, 1)
Epoch number: 18 ; Loss [0.7374239563941956, 0.0028860485, -0.035118725]
(34, 48, 48, 1)
Epoch number: 19 ; Loss [0.7075166404247284, 0.0056606694, -0.035749555]
(34, 48, 48, 1)
Epoch number: 20 ; Loss [0.7812702655792236, 0.0049098395, -0.044611003]
(34, 48, 48, 1)
Epoch number: 21 ; Loss [0.8000511229038239, 0.0007304241, -0.038869534]
(34, 48, 48, 1)
Epoch number: 22 ; Loss [0.7166752219200134, 0.001282447, -0.04734288]
(34, 48, 48, 1)
Epoch number: 23 ; Loss [0.7553698420524597, 0.0011751965, -0.052057914]
(34, 48, 48, 1)
Epoch number: 24 ; Loss [0.5744025707244873, 0.0011435283, -0.042825434]
(34, 48, 48, 1)
Epoch number: 25 ; Loss [0.6199201792478561, 0.00057868403, -0.041809306]
(34, 48, 48, 1)
Epoch number: 26 ; Loss [0.6172942221164703, 0.00034658544, -0.046519313]
(34, 48, 48, 1)
Epoch number: 27 ; Loss [0.6903990209102631, 0.0007170049, -0.042329103]
(34, 48, 48, 1)
Epoch number: 28 ; Loss [0.5417914241552353, 0.0017000891, -0.046591964]
(34, 48, 48, 1)
Epoch number: 29 ; Loss [0.570267528295517, 0.0005626886, -0.043799404]
```

```
(34, 48, 48, 1)
Epoch number: 30 ; Loss [0.5155835151672363, 0.0013757912, -0.033646982]
(34, 48, 48, 1)
Epoch number: 31 ; Loss [0.7575516402721405, 0.0013492098, -0.043970227]
(34, 48, 48, 1)
Epoch number: 32 ; Loss [0.6759389042854309, 0.0015431675, -0.039051548]
(34, 48, 48, 1)
Epoch number: 33 ; Loss [0.5223501175642014, 0.0005824081, -0.036288347]
(34, 48, 48, 1)
Epoch number: 34 ; Loss [0.641716718673706, 0.00087997643, -0.042303562]
(34, 48, 48, 1)
Epoch number: 35 ; Loss [0.522760346531868, 0.00061323657, -0.0414999]
(34, 48, 48, 1)
Epoch number: 36 ; Loss [0.5542566180229187, 0.0008152327, -0.03754979]
(34, 48, 48, 1)
Epoch number: 37 ; Loss [0.5342595279216766, 0.00055182277, -0.03749458]
(34, 48, 48, 1)
Epoch number: 38 ; Loss [0.6911451518535614, 0.0011129265, -0.047892723]
(34, 48, 48, 1)
Epoch number: 39 ; Loss [0.6133788079023361, 0.00067005027, -0.045620263]
(34, 48, 48, 1)
Epoch number: 40 ; Loss [0.5246322154998779, 0.00071650534, -0.048708007]
(34, 48, 48, 1)
Epoch number: 41 ; Loss [0.6390285491943359, 0.0027248592, -0.05327394]
(34, 48, 48, 1)
Epoch number: 42 ; Loss [0.5936098098754883, 0.0007359107, -0.03722434]
(34, 48, 48, 1)
Epoch number: 43 ; Loss [0.5263519883155823, 0.0007748858, -0.046465516]
(34, 48, 48, 1)
Epoch number: 44 ; Loss [0.5602746307849884, 0.00058529514, -0.04661291]
(34, 48, 48, 1)
Epoch number: 45 ; Loss [0.5908735543489456, 0.0005292305, -0.03876124]
(34, 48, 48, 1)
Epoch number: 46 ; Loss [0.5556635707616806, 0.0005139162, -0.049011305]
(34, 48, 48, 1)
Epoch number: 47 ; Loss [0.6653226017951965, 0.0032081343, -0.039519694]
(34, 48, 48, 1)
Epoch number: 48 ; Loss [0.5690702944993973, 0.00043195774, -0.05460092]
(34, 48, 48, 1)
Epoch number: 49 ; Loss [0.4903230369091034, 0.00038219048, -0.039377864]
(34, 48, 48, 1)
Epoch number: 50 ; Loss [0.651767760515213, 0.0004546022, -0.039867744]
(34, 48, 48, 1)
Epoch number: 51 ; Loss [0.49485622346401215, 0.002500495, -0.046032235]
(34, 48, 48, 1)
Epoch number: 52 ; Loss [0.4811054617166519, 0.00037995816, -0.03590232]
(34, 48, 48, 1)
Epoch number: 53 ; Loss [0.5501068830490112, 0.00050985586, -0.03721642]
```

```
(34, 48, 48, 1)
Epoch number: 54 ; Loss [0.5618027448654175, 0.00068530417, -0.041166525]
(34, 48, 48, 1)
Epoch number: 55 ; Loss [0.5961918234825134, 0.0003620031, -0.04556085]
(34, 48, 48, 1)
Epoch number: 56 ; Loss [0.6143903434276581, 0.00049963547, -0.04082096]
(34, 48, 48, 1)
Epoch number: 57 ; Loss [0.5054880529642105, 0.0003360914, -0.042123023]
(34, 48, 48, 1)
Epoch number: 58 ; Loss [0.5793071687221527, 0.0014807017, -0.03928193]
(34, 48, 48, 1)
Epoch number: 59 ; Loss [0.5472579151391983, 0.00023728935, -0.048101664]
(34, 48, 48, 1)
Epoch number: 60 ; Loss [0.693994402885437, 0.00028315597, -0.037955668]
(34, 48, 48, 1)
Epoch number: 61 ; Loss [0.5059803426265717, 0.0008542517, -0.037862837]
(34, 48, 48, 1)
Epoch number: 62 ; Loss [0.5799429267644882, 0.0003547905, -0.039460193]
(34, 48, 48, 1)
Epoch number: 63 ; Loss [0.4660877287387848, 0.0014754841, -0.037967592]
(34, 48, 48, 1)
Epoch number: 64 ; Loss [0.5712213814258575, 0.0003868316, -0.04374573]
(34, 48, 48, 1)
Epoch number: 65 ; Loss [0.5958122611045837, 0.0009377429, -0.049743112]
(34, 48, 48, 1)
Epoch number: 66 ; Loss [0.5266004502773285, 0.0041450537, -0.04390988]
(34, 48, 48, 1)
Epoch number: 67 ; Loss [0.47383467853069305, 0.0015270733, -0.048218198]
(34, 48, 48, 1)
Epoch number: 68 ; Loss [0.4304245859384537, 0.0012762122, -0.048940655]
(34, 48, 48, 1)
Epoch number: 69 ; Loss [0.49990835785865784, 0.0006208713, -0.031667814]
(34, 48, 48, 1)
Epoch number: 70 ; Loss [0.42486031353473663, 0.0015899534, -0.04259921]
(34, 48, 48, 1)
Epoch number: 71 ; Loss [0.46914172172546387, 0.0008322167, -0.044461697]
(34, 48, 48, 1)
Epoch number: 72 ; Loss [0.7921487092971802, 0.00040753908, -0.044640765]
(34, 48, 48, 1)
Epoch number: 73 ; Loss [0.3780480697751045, 0.0004267849, -0.03805977]
(34, 48, 48, 1)
Epoch number: 74 ; Loss [0.43439480662345886, 0.0005432751, -0.040882677]
(34, 48, 48, 1)
Epoch number: 75 ; Loss [0.41709819436073303, 0.0002646335, -0.035317495]
(34, 48, 48, 1)
Epoch number: 76 ; Loss [0.49194830656051636, 0.0002544677, -0.035611004]
(34, 48, 48, 1)
Epoch number: 77 ; Loss [0.33888307213783264, 0.0001766485, -0.043108433]
```

```
(34, 48, 48, 1)
Epoch number: 78 ; Loss [0.2748115733265877, 0.00015100875, -0.036385287]
(34, 48, 48, 1)
Epoch number: 79 ; Loss [0.43273159861564636, 0.00051635737, -0.03999743]
(34, 48, 48, 1)
Epoch number: 80 ; Loss [0.44282011687755585, 0.00011554623, -0.03781623]
(34, 48, 48, 1)
Epoch number: 81 ; Loss [0.303708091378212, 0.00039801985, -0.041972045]
(34, 48, 48, 1)
Epoch number: 82 ; Loss [0.5957076251506805, 0.00033013732, -0.040764272]
(34, 48, 48, 1)
Epoch number: 83 ; Loss [0.3546561896800995, 9.2549795e-05, -0.031949367]
(34, 48, 48, 1)
Epoch number: 84 ; Loss [0.46535928547382355, 0.00019633844, -0.037142072]
(34, 48, 48, 1)
Epoch number: 85 ; Loss [0.42723236978054047, 0.00015579507, -0.037384603]
(34, 48, 48, 1)
Epoch number: 86 ; Loss [0.4446816146373749, 0.00017489634, -0.036719587]
(34, 48, 48, 1)
Epoch number: 87 ; Loss [0.47875529527664185, 0.00017338197, -0.041708533]
(34, 48, 48, 1)
Epoch number: 88 ; Loss [0.4503537118434906, 0.00020797605, -0.03901714]
(34, 48, 48, 1)
Epoch number: 89 ; Loss [0.4084818363189697, 0.00019858476, -0.044549882]
(34, 48, 48, 1)
Epoch number: 90 ; Loss [0.2576989680528641, 7.892893e-05, -0.042154845]
(34, 48, 48, 1)
Epoch number: 91 ; Loss [0.46189427375793457, 0.0005405128, -0.041350737]
(34, 48, 48, 1)
Epoch number: 92 ; Loss [0.4255378693342209, 0.00034950615, -0.039490968]
(34, 48, 48, 1)
Epoch number: 93 ; Loss [0.3963019400835037, 9.340554e-05, -0.047192816]
(34, 48, 48, 1)
Epoch number: 94 ; Loss [0.3017963469028473, 0.00019525303, -0.036542226]
(34, 48, 48, 1)
Epoch number: 95 ; Loss [0.39550676941871643, 0.00029866965, -0.038697492]
(34, 48, 48, 1)
Epoch number: 96 ; Loss [0.6158477365970612, 8.846281e-05, -0.045344148]
(34, 48, 48, 1)
Epoch number: 97 ; Loss [0.3470371812582016, 0.00019170612, -0.041391477]
(34, 48, 48, 1)
Epoch number: 98 ; Loss [0.400258332490921, 0.00015873885, -0.039068762]
(34, 48, 48, 1)
Epoch number: 99 ; Loss [0.6083909571170807, 0.0001425895, -0.050784238]
(34, 48, 48, 1)
Epoch number: 100 ; Loss [0.4979783296585083, 9.072145e-05, -0.055513315]
(34, 48, 48, 1)
Epoch number: 101 ; Loss [0.46248855255544186, 0.00012163299, -0.039285947]
```

```
(34, 48, 48, 1)
Epoch number: 102 ; Loss [0.253824919462204, 0.00013811179, -0.029154776]
(34, 48, 48, 1)
Epoch number: 103 ; Loss [0.46466317400336266, 5.8764857e-05, -0.039401848]
(34, 48, 48, 1)
Epoch number: 104 ; Loss [0.6902834624052048, 6.157075e-05, -0.04188802]
(34, 48, 48, 1)
Epoch number: 105 ; Loss [0.3439755216240883, 0.00013766481, -0.037755925]
(34, 48, 48, 1)
Epoch number: 106 ; Loss [0.26332996040582657, 7.854234e-05, -0.038086448]
(34, 48, 48, 1)
Epoch number: 107 ; Loss [0.42238781601190567, 0.00010525045, -0.040204916]
(34, 48, 48, 1)
Epoch number: 108 ; Loss [0.26960157603025436, 9.5405245e-05, -0.05328491]
(34, 48, 48, 1)
Epoch number: 109 ; Loss [0.32599424570798874, 7.470118e-05, -0.03584257]
(34, 48, 48, 1)
Epoch number: 110 ; Loss [0.41600441187620163, 0.00018715042, -0.04465538]
(34, 48, 48, 1)
Epoch number: 111 ; Loss [0.30610182881355286, 8.783272e-05, -0.040473666]
(34, 48, 48, 1)
Epoch number: 112 ; Loss [0.4329855479300022, 5.494049e-05, -0.0460107]
(34, 48, 48, 1)
Epoch number: 113 ; Loss [0.2747863493859768, 6.8039386e-05, -0.044936173]
(34, 48, 48, 1)
Epoch number: 114 ; Loss [0.32238323241472244, 0.00011696031, -0.035203096]
(34, 48, 48, 1)
Epoch number: 115 ; Loss [0.3129628002643585, 8.536498e-05, -0.037909236]
(34, 48, 48, 1)
Epoch number: 116 ; Loss [0.373031385242939, 6.85145e-05, -0.039389405]
(34, 48, 48, 1)
Epoch number: 117 ; Loss [0.4133535921573639, 0.00014631478, -0.038369145]
(34, 48, 48, 1)
Epoch number: 118 ; Loss [0.40620070695877075, 8.5720356e-05, -0.048408423]
(34, 48, 48, 1)
Epoch number: 119 ; Loss [0.3723102807998657, 6.503501e-05, -0.04417259]
(34, 48, 48, 1)
Epoch number: 120 ; Loss [0.28697607666254044, 5.838729e-05, -0.039913878]
(34, 48, 48, 1)
Epoch number: 121 ; Loss [0.16318894177675247, 8.099321e-05, -0.042528912]
(34, 48, 48, 1)
Epoch number: 122 ; Loss [0.3177166283130646, 6.095185e-05, -0.04805269]
(34, 48, 48, 1)
Epoch number: 123 ; Loss [0.4216760993003845, 5.2964366e-05, -0.036838662]
(34, 48, 48, 1)
Epoch number: 124 ; Loss [0.16276038065552711, 0.00012642234, -0.039594326]
(34, 48, 48, 1)
Epoch number: 125 ; Loss [0.30001694709062576, 0.00015067623, -0.045845505]
```

```
(34, 48, 48, 1)
Epoch number: 126 ; Loss [0.3159421682357788, 8.377814e-05, -0.04405999]
(34, 48, 48, 1)
Epoch number: 127 ; Loss [0.7380866035819054, 0.000115568655, -0.041189607]
(34, 48, 48, 1)
Epoch number: 128 ; Loss [0.19947421550750732, 0.00022573447, -0.03359672]
(34, 48, 48, 1)
Epoch number: 129 ; Loss [0.593092842027545, 0.00012695853, -0.050259676]
(34, 48, 48, 1)
Epoch number: 130 ; Loss [0.4632520489394665, 8.6987224e-05, -0.048982248]
(34, 48, 48, 1)
Epoch number: 131 ; Loss [0.2951035760343075, 0.00015277314, -0.038140092]
(34, 48, 48, 1)
Epoch number: 132 ; Loss [1.3468485102057457, 0.00021912198, -0.039492607]
(34, 48, 48, 1)
Epoch number: 133 ; Loss [0.27994532138109207, 0.000148436, -0.042249247]
(34, 48, 48, 1)
Epoch number: 134 ; Loss [0.38383947126567364, 0.00023928641, -0.037336674]
(34, 48, 48, 1)
Epoch number: 135 ; Loss [0.8314497424289584, 0.00018903865, -0.044723235]
(34, 48, 48, 1)
Epoch number: 136 ; Loss [0.3223050646483898, 0.00020184602, -0.037157353]
(34, 48, 48, 1)
Epoch number: 137 ; Loss [0.5648700147867203, 6.118857e-05, -0.04761727]
(34, 48, 48, 1)
Epoch number: 138 ; Loss [0.22406186629086733, 7.050854e-05, -0.033979833]
(34, 48, 48, 1)
Epoch number: 139 ; Loss [0.36801600083708763, 0.000110156485, -0.040207267]
(34, 48, 48, 1)
Epoch number: 140 ; Loss [0.17570526897907257, 0.00014425929, -0.046686657]
(34, 48, 48, 1)
Epoch number: 141 ; Loss [0.5422162413597107, 5.5128505e-05, -0.039224993]
(34, 48, 48, 1)
Epoch number: 142 ; Loss [0.43735361844301224, 0.00012901326, -0.03238475]
(34, 48, 48, 1)
Epoch number: 143 ; Loss [0.40306900441646576, 0.00010627197, -0.04109961]
(34, 48, 48, 1)
Epoch number: 144 ; Loss [0.8714688383042812, 0.00017198664, -0.04319277]
(34, 48, 48, 1)
Epoch number: 145 ; Loss [0.1854732260107994, 8.59092e-05, -0.04278505]
(34, 48, 48, 1)
Epoch number: 146 ; Loss [0.25317271798849106, 8.454021e-05, -0.045846887]
(34, 48, 48, 1)
Epoch number: 147 ; Loss [0.25281124375760555, 0.00015258107, -0.040233474]
(34, 48, 48, 1)
Epoch number: 148 ; Loss [0.9492258876562119, 3.7874634e-05, -0.03621388]
(34, 48, 48, 1)
Epoch number: 149 ; Loss [0.3592187911272049, 4.8324204e-05, -0.045297593]
```

4 Classifier Load

[64]:
```
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
```

[65]:
```
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
_________________________________________________________________
```

[66]:
```
test_labels = to_categorical(np.load('./ATTACKS/JSMA/Y_TEST_JSMA.npy').
↪astype('int'))
```

5 Purify the Stuff

[67]:
```
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:0.3366
```

[68]:
```
F.evaluate(clean.reshape(-1,48,48),test_labels)#[N//5*4:])
```

```
5000/5000 [==============================] - 1s 155us/step
```

[68]:
```
[0.19247934680879117, 0.9476000070571899]
```

[69]:
```
F.evaluate(adv.reshape(-1,48,48),test_labels)#[N//5*4:])
```

```
5000/5000 [==============================] - 1s 110us/step
```

[69]:
```
[0.8315977921485901, 0.6453999876976013]
```

[70]:
```
F.evaluate(purified.reshape(-1,48,48),(test_labels))#[N//5*4:])
```

```
5000/5000 [==============================] - 1s 101us/step
```

[70]:
```
[6.785995056152344, 0.33739998936653137]
```

[71]:
```
clean[0].shape
```

[71]:
```
(48, 48, 1)
```

[72]:
```
"DONE"
```

[72]:
```
'DONE'
```

[73]:
```
np.unique(np.argmax(F.predict(adv.reshape(-1,48,48)),axis=1),return_counts=True)
```

[73]:
```
(array([0, 1], dtype=int64), array([4894,  106], dtype=int64))
```

[74]:
```
np.unique(np.argmax(F.predict(purified.
 ↪reshape(-1,48,48)),axis=1),return_counts=True)
```

[74]:
```
(array([0, 1], dtype=int64), array([  48, 4952], dtype=int64))
```

6 Conclusion

In JSMA, training for 150 epochs made it such that almost all were classified as MALWARE, ehich is the reason behinf the 33%, we had 33% MAL and if all predict as MAL, its 33 %

[ ]:
```

```