APEGAN JSMA

November 16, 2020

[1]:
```
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
```

```
Using TensorFlow backend.
```

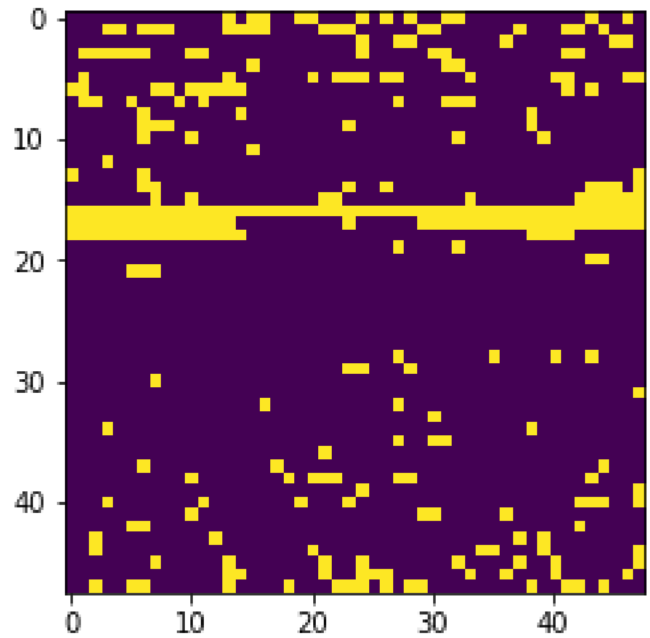
[2]:
```
x_clean = np.load('./SELECTED_DATA/X_CLEAN.npy')
x_adv = np.load('./SELECTED_DATA/X_ADV.npy')
x_label = np.load('./SELECTED_DATA/X_LABEL_1D.npy').astype('int')
```

[3]:
```
x_label[5]
```

[3]:
```
1
```

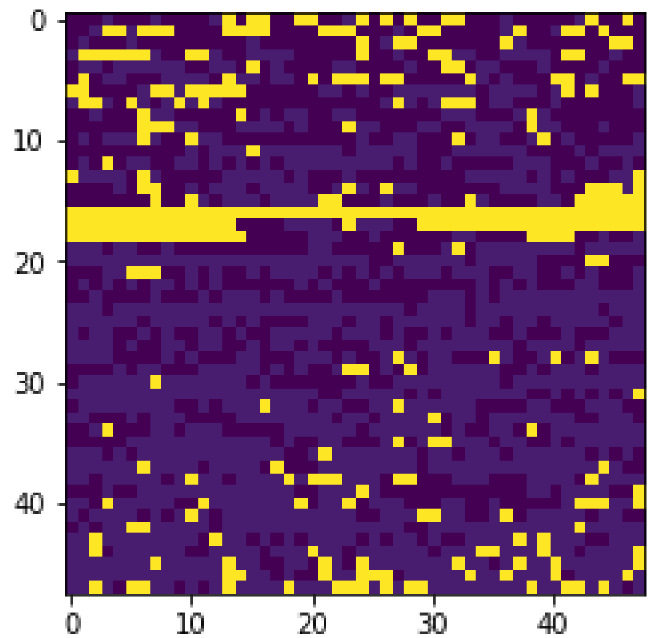
[4]:
```
plt.imshow((x_clean[5]))
```

[4]: <matplotlib.image.AxesImage at 0x1b9ed573d68>




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

[5]: <matplotlib.image.AxesImage at 0x1b9ed85b780>



1 DEFINE LOSS FUNCS AND APE GAN

[6]:
```
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)
```

[7]:
```
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

[24]:
```
epochs=50 # original 500
batch_size=256
```

[25]:
```
GAN, G, D = APEGAN([48,48,1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 48, 48, 1)         0
_________________________________________________________________
sequential_4 (Sequential)    (None, 48, 48, 1)         149889
_________________________________________________________________
sequential_5 (Sequential)    (None, 1)                 380673
=================================================================
Total params: 530,562
Trainable params: 149,377
Non-trainable params: 381,185
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 24, 24, 64)        640
_________________________________________________________________
batch_normalization_7 (Batch (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_8 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 24, 24, 64)        73792
_________________________________________________________________
batch_normalization_9 (Batch (None, 24, 24, 64)        256
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 48, 48, 1)         577
_________________________________________________________________
```

```
activation_3 (Activation)    (None, 48, 48, 1)         0
=================================================================
Total params: 149,889
Trainable params: 149,377
Non-trainable params: 512

_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 24, 24, 64)        640

_________________________________________________________________
batch_normalization_10 (Batc (None, 24, 24, 64)        256

_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 24, 24, 64)        0

_________________________________________________________________
conv2d_9 (Conv2D)            (None, 12, 12, 128)       73856

_________________________________________________________________
batch_normalization_11 (Batc (None, 12, 12, 128)       512

_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 12, 12, 128)       0

_________________________________________________________________
conv2d_10 (Conv2D)           (None, 6, 6, 256)         295168

_________________________________________________________________
batch_normalization_12 (Batc (None, 6, 6, 256)         1024

_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)   (None, 6, 6, 256)         0

_________________________________________________________________
flatten_2 (Flatten)          (None, 9216)              0

_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9217

_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0
=================================================================
Total params: 380,673
Trainable params: 0
Non-trainable params: 380,673

_________________________________________________________________
```

---

3 Set Params and RUN GAN

[28]:
```python
epochs=50 # original 500
batch_size=34
N = x_clean.shape[0]
```

[29]:
```
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\erikCopy\lib\site-
packages\keras\engine\training.py:975: 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 [6.643388721742667, 0.007630397, -0.022482593]
(34, 48, 48, 1)
Epoch number: 1 ; Loss [3.035663517192006, 0.017105281, -0.028847422]
(34, 48, 48, 1)
Epoch number: 2 ; Loss [1.6127283573150635, 0.012622592, -0.034341734]
(34, 48, 48, 1)
Epoch number: 3 ; Loss [1.9965812861919403, 0.023161652, -0.03671871]
(34, 48, 48, 1)
Epoch number: 4 ; Loss [2.6091552674770355, 0.013194317, -0.04140177]
(34, 48, 48, 1)
Epoch number: 5 ; Loss [1.870964527130127, 0.0132703455, -0.042581204]
(34, 48, 48, 1)
Epoch number: 6 ; Loss [1.4679377377033234, 0.010142392, -0.0418421]
(34, 48, 48, 1)
Epoch number: 7 ; Loss [1.2403725981712341, 0.013729751, -0.041151177]
(34, 48, 48, 1)
Epoch number: 8 ; Loss [1.0004899092018604, 0.019667909, -0.030291123]
(34, 48, 48, 1)
Epoch number: 9 ; Loss [0.6957384645938873, 0.0077204457, -0.040615175]
(34, 48, 48, 1)
Epoch number: 10 ; Loss [1.1669419705867767, 0.004159948, -0.038544778]
(34, 48, 48, 1)
Epoch number: 11 ; Loss [1.056589514017105, 0.005380317, -0.041386988]
```

```
(34, 48, 48, 1)
Epoch number: 12 ; Loss [0.8718676567077637, 0.0041082986, -0.038207915]
(34, 48, 48, 1)
Epoch number: 13 ; Loss [1.0671941637992859, 0.005577448, -0.044025093]
(34, 48, 48, 1)
Epoch number: 14 ; Loss [0.4028325527906418, 0.003258165, -0.034334864]
(34, 48, 48, 1)
Epoch number: 15 ; Loss [0.4004450663924217, 0.003166555, -0.040521804]
(34, 48, 48, 1)
Epoch number: 16 ; Loss [0.5166155397891998, 0.0016107651, -0.038523167]
(34, 48, 48, 1)
Epoch number: 17 ; Loss [0.36729342490434647, 0.005929934, -0.044343136]
(34, 48, 48, 1)
Epoch number: 18 ; Loss [0.3630184382200241, 0.007594651, -0.046721466]
(34, 48, 48, 1)
Epoch number: 19 ; Loss [0.2826797068119049, 0.00262951, -0.042316046]
(34, 48, 48, 1)
Epoch number: 20 ; Loss [0.45237143337726593, 0.0011067981, -0.042044267]
(34, 48, 48, 1)
Epoch number: 21 ; Loss [0.164405919611454, 0.0004437312, -0.032223184]
(34, 48, 48, 1)
Epoch number: 22 ; Loss [0.18212591484189034, 0.0010821102, -0.033281174]
(34, 48, 48, 1)
Epoch number: 23 ; Loss [0.11300947517156601, 0.0010495874, -0.029069975]
(34, 48, 48, 1)
Epoch number: 24 ; Loss [0.12660507299005985, 0.0006177977, -0.039005548]
(34, 48, 48, 1)
Epoch number: 25 ; Loss [0.09863054007291794, 0.0002596744, -0.03930391]
(34, 48, 48, 1)
Epoch number: 26 ; Loss [0.09588372893631458, 0.0005442033, -0.04193219]
(34, 48, 48, 1)
Epoch number: 27 ; Loss [0.15343614295125008, 0.00053434825, -0.043116912]
(34, 48, 48, 1)
Epoch number: 28 ; Loss [0.04930759780108929, 0.00079317705, -0.036612097]
(34, 48, 48, 1)
Epoch number: 29 ; Loss [0.28881484270095825, 0.00038269933, -0.04891691]
(34, 48, 48, 1)
Epoch number: 30 ; Loss [0.03699955902993679, 0.002273878, -0.040184215]
(34, 48, 48, 1)
Epoch number: 31 ; Loss [0.058464540168643, 0.0009459348, -0.038068935]
(34, 48, 48, 1)
Epoch number: 32 ; Loss [0.037951091304421425, 0.0022182795, -0.04373854]
(34, 48, 48, 1)
Epoch number: 33 ; Loss [0.10702563729137182, 0.00035065957, -0.034056697]
(34, 48, 48, 1)
Epoch number: 34 ; Loss [0.05924844369292259, 0.000110575056, -0.040650044]
(34, 48, 48, 1)
Epoch number: 35 ; Loss [0.019676608964800835, 3.1704843e-05, -0.035874847]
```

```
(34, 48, 48, 1)
Epoch number: 36 ; Loss [0.049741748720407486, 0.000101401834, -0.040650208]
(34, 48, 48, 1)
Epoch number: 37 ; Loss [0.08628794364631176, 0.00015114539, -0.050914258]
(34, 48, 48, 1)
Epoch number: 38 ; Loss [0.0233500967733562, 0.00022509896, -0.042461447]
(34, 48, 48, 1)
Epoch number: 39 ; Loss [0.009017219068482518, 7.4232405e-05, -0.036981013]
(34, 48, 48, 1)
Epoch number: 40 ; Loss [0.019093520939350128, 0.00021755118, -0.03858347]
(34, 48, 48, 1)
Epoch number: 41 ; Loss [0.01046544685959816, 0.00010616802, -0.037457705]
(34, 48, 48, 1)
Epoch number: 42 ; Loss [0.0655067004263401, 0.0006886539, -0.044961467]
(34, 48, 48, 1)
Epoch number: 43 ; Loss [0.010515751782804728, 0.0011731377, -0.04211011]
(34, 48, 48, 1)
Epoch number: 44 ; Loss [0.01796754403039813, 9.332397e-05, -0.04445447]
(34, 48, 48, 1)
Epoch number: 45 ; Loss [0.014877557288855314, 5.1130348e-05, -0.044714287]
(34, 48, 48, 1)
Epoch number: 46 ; Loss [0.003233722411096096, 3.90551e-05, -0.04215876]
(34, 48, 48, 1)
Epoch number: 47 ; Loss [0.08437887113541365, 7.3698306e-05, -0.045423422]
(34, 48, 48, 1)
Epoch number: 48 ; Loss [0.004097515717148781, 0.00012905398, -0.043533586]
(34, 48, 48, 1)
Epoch number: 49 ; Loss [0.010630580596625805, 0.00012033416, -0.044579383]
```

---

4 Classifier Load

[30]:
```
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
```

[31]:
```
F = keras.models.load_model("./SELECTED_DATA/network.h5py")
```

[32]:
```
test_labels = to_categorical(np.load('./SELECTED_DATA/X_LABEL_1D.npy').
↪astype('int'))
```

5 Purify the Stuff

[33]:
```
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:0.6833, rct acc:0.3167
```

[35]:
```
F.evaluate(clean.reshape(-1,48,48),test_labels)#[N//5*4:])
```

```
4768/4768 [==============================] - 1s 154us/step
```

[35]:
```
[0.015147521063660416, 1.0]
```

[37]:
```
F.evaluate(adv.reshape(-1,48,48),test_labels)#[N//5*4:])
```

```
4768/4768 [==============================] - 1s 119us/step
```

[37]:
```
[4.619444339867406, 0.6833053691275168]
```

[38]:
```
F.evaluate(purified.reshape(-1,48,48),(test_labels))#[N//5*4:])
```

```
4768/4768 [==============================] - 1s 117us/step
```

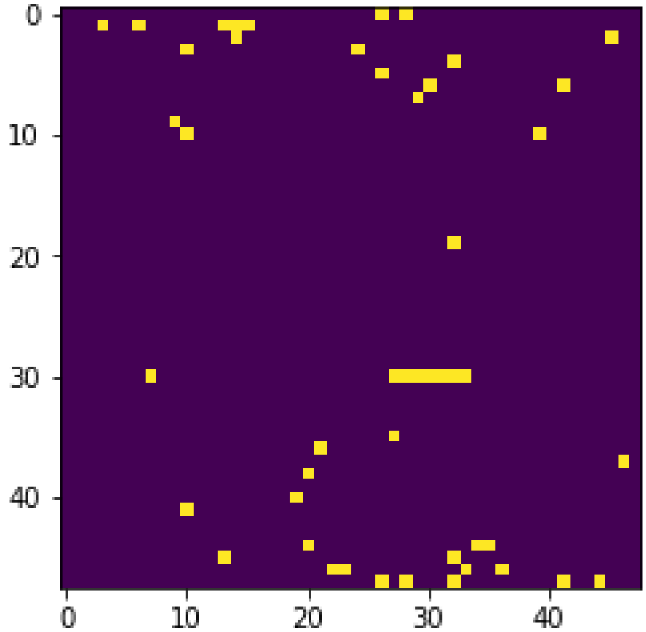
[38]:
```
[10.891552448272705, 0.31669463087248323]
```

[39]:
```
clean[0].shape
```

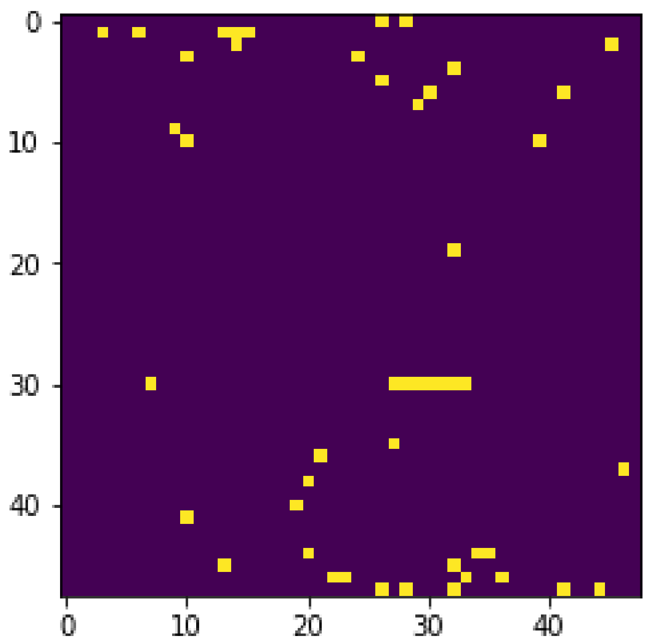
[39]:
```
(48, 48, 1)
```

[40]:
```
for k in range(5):
    plt.imshow((clean[k].reshape(48,48)).astype(np.int))
    print("CLEAN")
    plt.show()
    plt.imshow((adv[k].reshape(48,48)).astype(np.int))
    print("ADV")
    plt.show()
    plt.imshow((purified[k].reshape(48,48)).astype(np.int))
    print("PURIFIED")
    plt.show()
    print("==================")
```

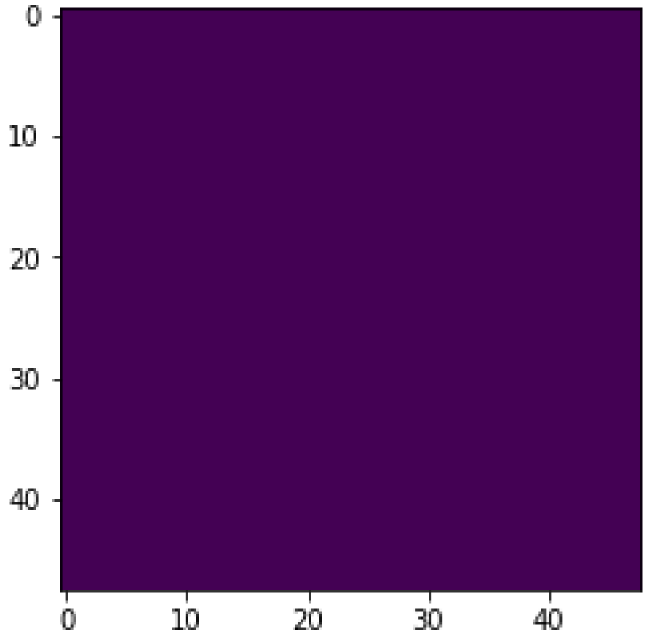
CLEAN



ADV

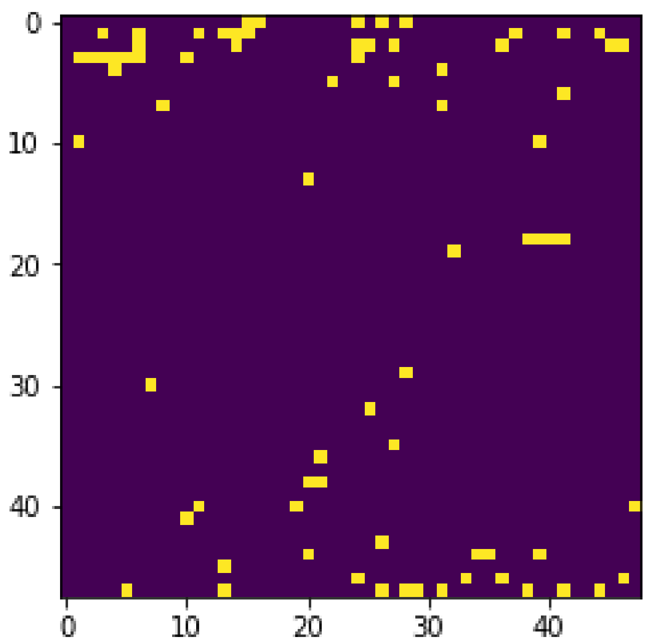


PURIFIED

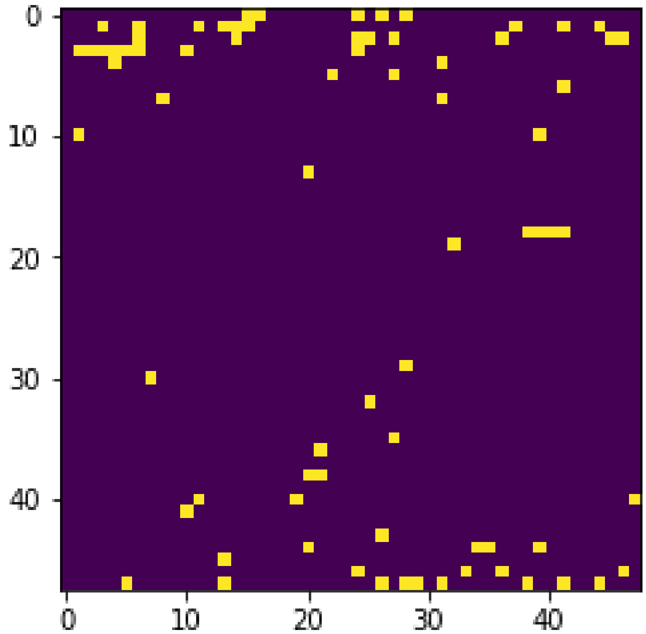


==================

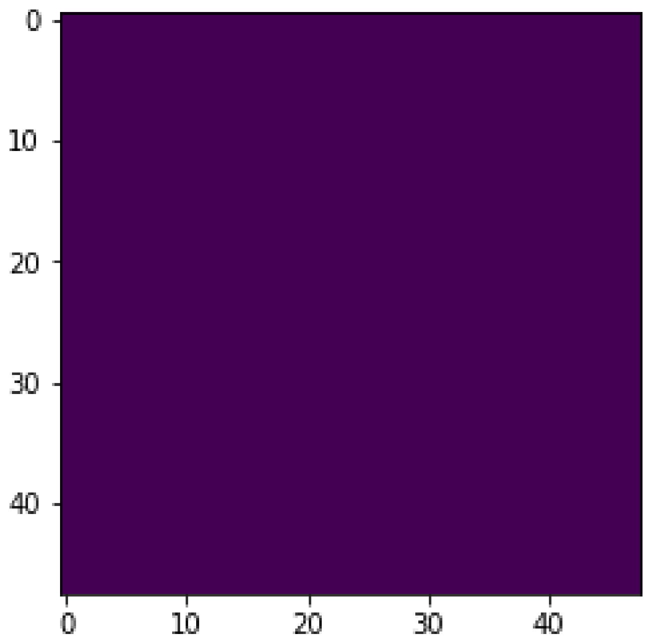
CLEAN



ADV

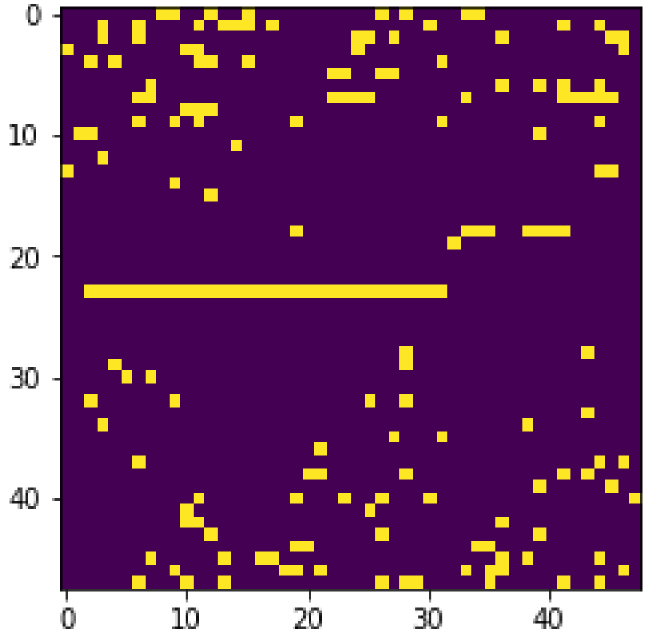


PURIFIED

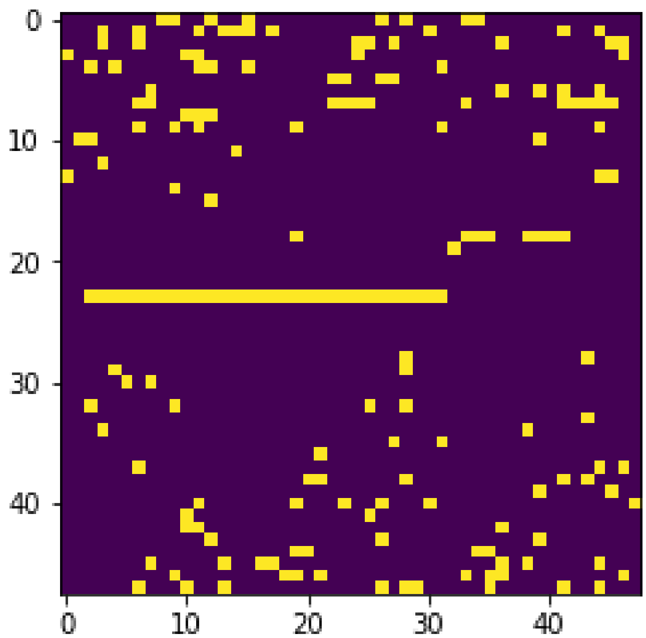


===================

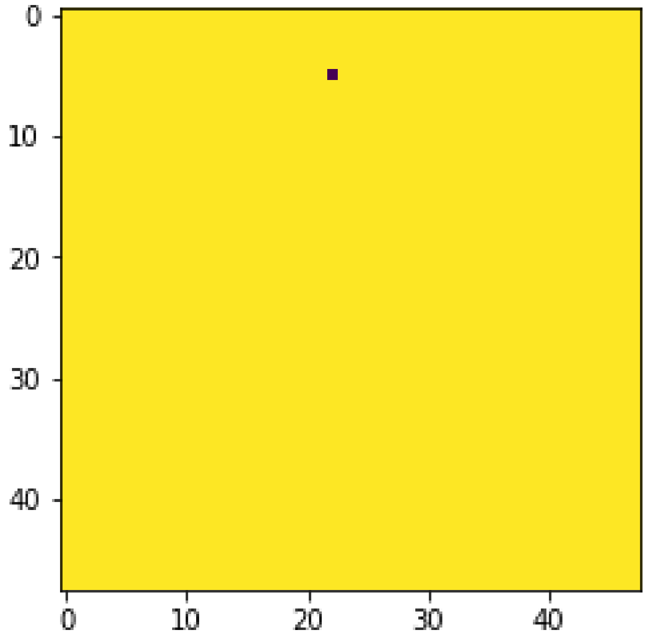
CLEAN



ADV

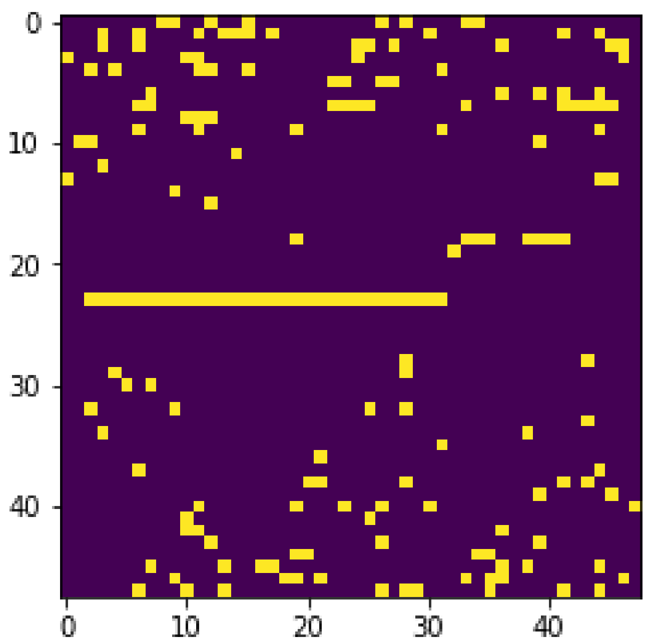


PURIFIED

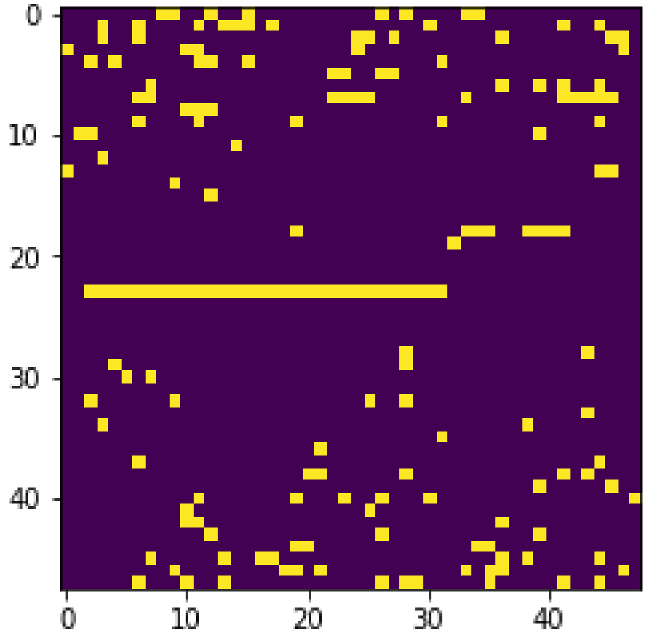


===================

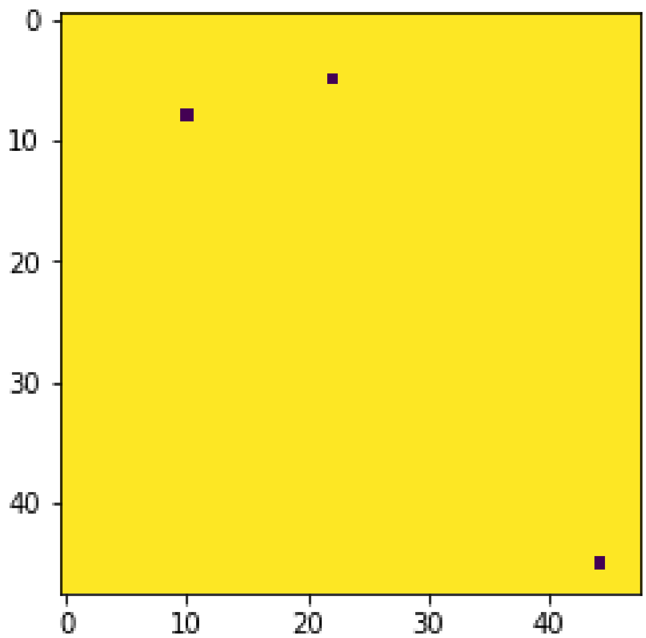
CLEAN



ADV

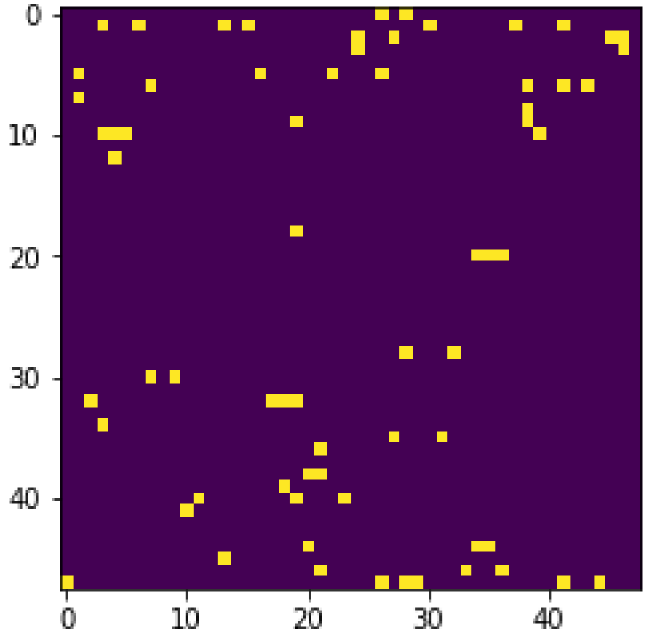


PURIFIED

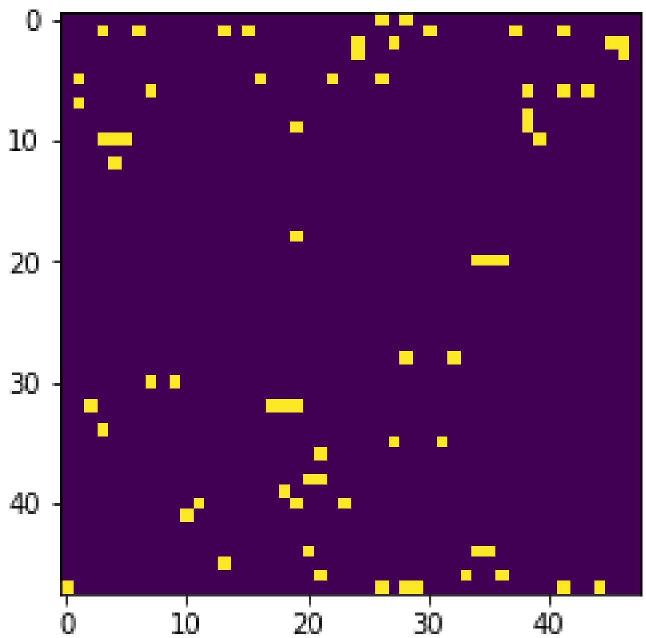


===================

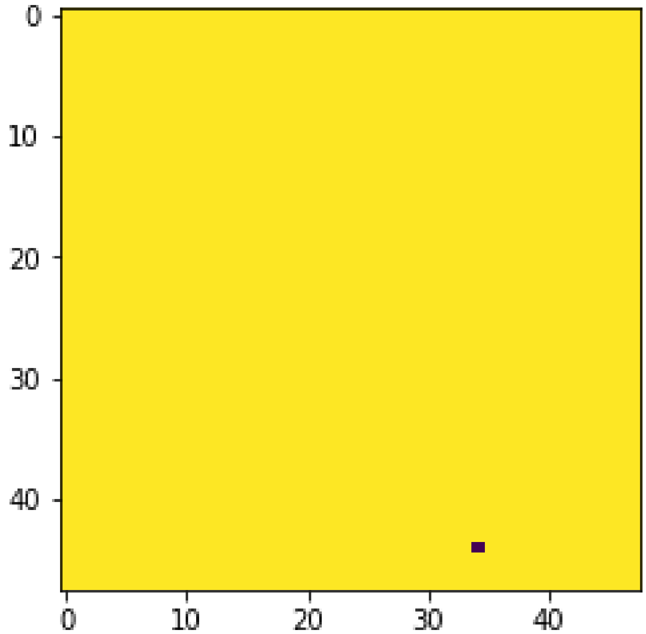
CLEAN



ADV



PURIFIED



===================

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

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

```
[46]: (array([1], dtype=int64), array([4768], dtype=int64))
```

```
[ ]:
```

```
[ ]:
```