APE GAN FGSM PURE MAL

November 14, 2020

[2]:
```
print("STARTING")
```

```
STARTING
```

[3]:
```
import numpy as np
from sklearn.model_selection import train_test_split
import random
import numpy as np
import keras
import tensorflow as tf
from keras.datasets import mnist
from keras.utils import np_utils  # utilities for one-hot encoding of ground␣
↪truth values

import os

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, Conv1D
from keras.optimizers import Adam
from keras import backend as K

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

%reload_ext autoreload
%autoreload 2
```

```
Using TensorFlow backend.
```

[4]:
```
X_TEST = np.load('./DATA/X_CLEAN_ONLY_MAL.npy')
Y_TEST = np.load('./DATA/X_LABEL_ONLY_MAL.npy')
coeff = np.load('./DATA/ORIGINAL/coeff_features.npy')
X_N_TEST = np.load('./DATA/X_ADV_ONLY_MAL.npy')
```

1 Classifier Import and test

Building a classifier to evaluate the denoising

[6]:
```
np.unique(Y_TEST,return_counts=True)
```

[6]:
```
(array([1]), array([3799], dtype=int64))
```

[7]:
```
modelClassifier = keras.models.load_model('./modelClassifierFGSM.h5')
modelClassifier.summary()
```

```
WARNING:tensorflow:From C:\Users\Pitch\.conda\envs\tf1-gpu\lib\site-
packages\tensorflow_core\python\ops\resource_variable_ops.py:1630: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From C:\Users\Pitch\.conda\envs\tf1-gpu\lib\site-
packages\tensorflow_core\python\ops\nn_impl.py:183: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\Pitch\.conda\envs\tf1-gpu\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 2948)              8693652
_________________________________________________________________
dense_8 (Dense)              (None, 128)               377472
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 129
=================================================================
Total params: 9,071,253
Trainable params: 9,071,253
Non-trainable params: 0
_________________________________________________________________
```

[9]:
```
modelClassifier.evaluate(X_TEST, Y_TEST)
```

```
3799/3799 [==============================] - 2s 556us/step
```

[9]:
```
[0.04282340146973827, 1.0]
```

[10]:
```
modelClassifier.evaluate(X_N_TEST, Y_TEST)
```

```
3799/3799 [==============================] - 2s 450us/step
```

[10]:
```
[299.8835265824091, 0.0]
```

1.0.1 Accuracy of classifier

- Clean data = 100%
- FGSM attacked data = 0%

But remeber this is just malware

[11]:
```
weightsmodelClassifier, biasesmodelClassifier = modelClassifier.layers[0].
 ↪get_weights()
modelClassifier.layers[0].get_weights()[1].shape
```

[11]:
```
(2948,)
```

2 Ben and Mal Cols and Pure Malware and Benign Arrays

Pure Malware = all malware columns will be one

Pure Ben = all Ben cols will be one

This gives us the 2 extreme cases to compare with one another during GAN training. All Ben being the most benign and all mal being the worst

[12]:
```
mal_col_index = []
ben_col_index = []
for i in range(len(coeff)):
    if coeff[i] > 0:
        mal_col_index.append(i)
    elif coeff[i] < 0:
        ben_col_index.append(i)
    else:
        print("DANGER")
print(len(mal_col_index),len(ben_col_index))
```

```
1513 1435
```

[13]:
```
ALL_BEN_APK = np.zeros(X_TEST[0].shape)
for i in ben_col_index:
    ALL_BEN_APK[i] = 1
np.unique(ALL_BEN_APK,return_counts=True)
```

[13]:
```
(array([0., 1.]), array([1513, 1435], dtype=int64))
```

[14]:
```
ALL_MAL_APK = np.zeros(X_TEST[0].shape)
for i in mal_col_index:
    ALL_MAL_APK[i] = 1
np.unique(ALL_MAL_APK,return_counts=True)
```

[14]:
```
(array([0., 1.]), array([1435, 1513], dtype=int64))
```

3 APE GAN DEF

[15]:
```
def MANHATTAN(y_true, y_pred):
    return K.sum( K.abs( y_true - y_pred),axis=1,keepdims=True) + 1e-10

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

def WMOD(y_true, y_pred):
    return K.abs(1 - K.mean(y_true * y_pred))

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

[16]:
```
def generator(input_dims):
    model = Sequential()
    model.add(Dense(512, input_shape = input_dims, activation='relu'))
    #model.add(Dense(2048, input_shape = input_dims, activation='relu'))
    #model.add(Dense(1024, activation='relu'))
    #model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(4, activation='relu'))
    model.add(Dense(2, activation='relu'))
    model.add(Dense(1, activation='relu'))
    model.add(Activation('tanh'))
    return model

def discriminator(input_dims):
    model = Sequential()
    model.add(Dense(512, input_shape = input_dims, activation='relu'))
    #model.add(Dense(2048, input_shape = input_dims, activation='relu'))
    #model.add(Dense(1024, activation='relu'))
    #model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
```

```
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(4, activation='relu'))
    model.add(Dense(2, activation='relu'))
    model.add(Dense(1, activation='relu'))
    model.add(Activation('sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy')
    return model

def APEGAN(input_dims):
    G = generator(input_dims)
    print("========================\n\nGENERATOR\n\n")
    print(G.summary())
    print("==============\n\n")

    D = discriminator(input_dims)
    print("========================\n\nDISCRIMINATOR\n\n")
    print(D.summary())
    print("==============\n\n")

    ipt = Input(input_dims)
    print("========================\n\nINPUT TENSOR\n\n")
    print(ipt)
    print("==============\n\n")

    purified = G(ipt)
    print("========================\n\nPURIFIED TENSOR\n\n")
    print(purified)
    print("==============\n\n")

    D.trainable = False

    judge = D(purified)
    print("========================\n\nJUDGE TENSOR\n\n")
    print(judge)
    print("==============\n\n")

    GAN = Model(ipt, [judge,purified])
    print("========================\n\nGAN BASIC\n\n")
    print(GAN.summary())
    print("==============\n\n")

    GAN.compile(optimizer='adam',
                loss=['binary_crossentropy',WGAN],
```

```
                loss_weights=[0.02, 0.9])

    print("========================\n\nGAN AFTER COMPILE\n\n")
    print(GAN.summary())
    print("==============\n\n")

    return GAN,G,D
```

4 APE GAN RUN

[17]:
```
epochs=10 # original 500
batch_size=128

N = X_TEST.shape[0]
x_clean = X_TEST.copy()
x_adv = X_N_TEST.copy()
x_label = Y_TEST.copy()
```

[18]:
```
GAN, G, D = APEGAN([1,X_TEST.shape[1],1])
# GAN,G,D = APEGAN([X_TEST.shape[1]])
```

```
========================


GENERATOR


Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1, 2948, 512)      1024
_________________________________________________________________
dense_2 (Dense)              (None, 1, 2948, 256)      131328
_________________________________________________________________
dense_3 (Dense)              (None, 1, 2948, 128)      32896
_________________________________________________________________
dense_4 (Dense)              (None, 1, 2948, 64)       8256
_________________________________________________________________
dense_5 (Dense)              (None, 1, 2948, 32)       2080
_________________________________________________________________
dense_6 (Dense)              (None, 1, 2948, 16)       528
_________________________________________________________________
dense_7 (Dense)              (None, 1, 2948, 8)        136
_________________________________________________________________
dense_8 (Dense)              (None, 1, 2948, 4)        36
_________________________________________________________________
```

```
dense_9 (Dense)              (None, 1, 2948, 2)       10
_________________________________________________________________
dense_10 (Dense)             (None, 1, 2948, 1)       3
_________________________________________________________________
activation_1 (Activation)    (None, 1, 2948, 1)       0
=================================================================
Total params: 176,297
Trainable params: 176,297
Non-trainable params: 0
_________________________________________________________________
None
==============


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

DISCRIMINATOR


Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape             Param #
=================================================================
dense_11 (Dense)             (None, 1, 2948, 512)     1024
_________________________________________________________________
dense_12 (Dense)             (None, 1, 2948, 256)     131328
_________________________________________________________________
dense_13 (Dense)             (None, 1, 2948, 128)     32896
_________________________________________________________________
dense_14 (Dense)             (None, 1, 2948, 64)      8256
_________________________________________________________________
dense_15 (Dense)             (None, 1, 2948, 32)      2080
_________________________________________________________________
dense_16 (Dense)             (None, 1, 2948, 16)      528
_________________________________________________________________
dense_17 (Dense)             (None, 1, 2948, 8)       136
_________________________________________________________________
dense_18 (Dense)             (None, 1, 2948, 4)       36
_________________________________________________________________
dense_19 (Dense)             (None, 1, 2948, 2)       10
_________________________________________________________________
dense_20 (Dense)             (None, 1, 2948, 1)       3
_________________________________________________________________
activation_2 (Activation)    (None, 1, 2948, 1)       0
=================================================================
Total params: 176,297
Trainable params: 176,297
Non-trainable params: 0
```

```
----------------------------------------------------------------
None
==============


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

INPUT TENSOR


Tensor("input_1:0", shape=(?, 1, 2948, 1), dtype=float32)
==============


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

PURIFIED TENSOR


Tensor("sequential_1/activation_1/Tanh:0", shape=(?, 1, 2948, 1), dtype=float32)
==============


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

JUDGE TENSOR


Tensor("sequential_2/activation_2/Sigmoid:0", shape=(?, 1, 2948, 1),
dtype=float32)
==============


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

GAN BASIC


Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1, 2948, 1)        0
_________________________________________________________________
sequential_1 (Sequential)    (None, 1, 2948, 1)        176297
_________________________________________________________________
sequential_2 (Sequential)    (None, 1, 2948, 1)        176297
=================================================================
```

```
Total params: 352,594
Trainable params: 176,297
Non-trainable params: 176,297
_________________________________________________________________
None
==============


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

GAN AFTER COMPILE


Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1, 2948, 1)        0
_________________________________________________________________
sequential_1 (Sequential)    (None, 1, 2948, 1)        176297
_________________________________________________________________
sequential_2 (Sequential)    (None, 1, 2948, 1)        176297
=================================================================
Total params: 352,594
Trainable params: 176,297
Non-trainable params: 176,297
_________________________________________________________________
None
==============
```

[19]:

```
scalarloss = [0,0,0]
LossHistory = [0]*epochs
for cur_epoch in range(epochs):

    print("\n\nEPOCH",cur_epoch,"\n")

    idx = np.random.randint(0, N*4//5, size=batch_size)
    x_clean_batch = x_clean[idx,].reshape(-1,1,2948,1)
    x_adv_batch = x_adv[idx,].reshape(-1,1,2948,1)

    ALL_MAL_APK_BATCH = np.tile(ALL_MAL_APK,(batch_size,1))
    ALL_BEN_APK_BATCH = np.tile(ALL_BEN_APK,(batch_size,1))

    print("\n=====\nSHAPE of XCLEAN")
    print(x_clean_batch.shape)
```

```
    scalarloss[0] = D.train_on_batch(x_clean_batch, ALL_MAL_APK_BATCH.
↪reshape(-1,1,2948,1))/2

    print("\n=====\nNP ONES TRAIN ON BATCH DISCRIMINATOR")
    print("1 "+str(scalarloss))

    scalarloss[0] += D.train_on_batch(x_adv_batch, ALL_BEN_APK_BATCH.
↪reshape(-1,1,2948,1))/2

    print("\n=====\nNP ZEROS TRAIN ON BATCH DISCRIMINATOR")
    print("2 "+str(scalarloss))

    GAN.train_on_batch(x_adv_batch, [ ALL_MAL_APK_BATCH.reshape(-1,1,2948,1),␣
↪x_clean_batch])

    scalarloss[1:] = GAN.train_on_batch(x_adv_batch, [ ALL_MAL_APK_BATCH.
↪reshape(-1,1,2948,1), x_clean_batch])[1:]

#     LossHistory.append(scalarloss)
    LossHistory[cur_epoch] = scalarloss
    print("\n=====\nLOSS HISTORY")
    print(LossHistory)

    print("\n=====\nTRAIN ON BATCH GAN")
    print("Epoch number:",cur_epoch,"; Loss",scalarloss)
    print("\n EPOCHING \n\n\n\n\n")
```

```
EPOCH 0


=====
SHAPE of XCLEAN
(128, 1, 2948, 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'
```

```
=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0, 0]
```

```
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'


=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0, 0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], 0, 0, 0, 0, 0, 0, 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 0 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING







EPOCH 1


=====
SHAPE of XCLEAN
(128, 1, 2948, 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'


=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
```

```
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
0, 0, 0, 0, 0, 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 1 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING









EPOCH 2



=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], 0, 0, 0, 0, 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 2 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING
```

```
EPOCH 3


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0], 0,
0, 0, 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 3 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING






EPOCH 4


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]
```

```
=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], 0, 0, 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 4 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING




EPOCH 5


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0], 0,
0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 5 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING
```

```
EPOCH 6


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 6 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING




EPOCH 7


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]
```

```
=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0], 0,
0]

=====
TRAIN ON BATCH GAN
Epoch number: 7 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING




EPOCH 8


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], 0]
```

```
=====
TRAIN ON BATCH GAN
Epoch number: 8 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING







EPOCH 9


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0]]

=====
TRAIN ON BATCH GAN
Epoch number: 9 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING
```

5 Evaluation

We use the Generator of the trained APEGAN to purify data samples and then test it out with the trained classifier

[20]:
```
F = keras.models.load_model('./modelClassifierFGSM.h5')
print("========================\n\nKERAS LOAD MODEL\n\n")
print(F.summary())
print("==============\n\n")

clean = X_TEST.copy().reshape(-1,1,X_TEST.shape[1],1)
adv = X_N_TEST.copy().reshape(-1,1,X_TEST.shape[1],1)
label = Y_TEST.copy()

print("========================\n\nCLEAN\n\n")
print(clean)
print("==============\n\n")

print("========================\n\nADV\n\n")
print(adv)
print("==============\n\n")

purified = G.predict(adv)
print("========================\n\nG Predict ADV - PURIFIED\n\n")
print(purified)
print("==============\n\n")
```

```
========================

KERAS LOAD MODEL


Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 2948)              8693652
_________________________________________________________________
dense_8 (Dense)              (None, 128)               377472
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 129
=================================================================
Total params: 9,071,253
Trainable params: 9,071,253
Non-trainable params: 0
_________________________________________________________________
None
==============
```

```
========================

CLEAN


[[[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]


 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]


 [[[0.]
   [1.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]


 …


 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]


 [[[0.]
   [0.]
```

```
  [0.]
  …
  [0.]
  [0.]
  [0.]]]


 [[[0.]
  [0.]
  [0.]
  …
  [0.]
  [0.]
  [0.]]]]
==============


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

ADV


[[[[0.]
   [0.]
   [0.]
   …
   [0.]
   [1.]
   [1.]]]


 [[[1.]
   [1.]
   [0.]
   …
   [1.]
   [1.]
   [0.]]]


 [[[0.]
   [1.]
   [0.]
   …
   [1.]
   [1.]
   [1.]]]
```

```
 …

 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [1.]
   [1.]]]

 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [1.]
   [1.]]]

 [[[1.]
   [0.]
   [0.]
   …
   [1.]
   [1.]
   [1.]]]]
==============

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

G Predict ADV - PURIFIED

[[[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]

 [[[0.]
   [0.]
```

```
   [0.]
   …
   [0.]
   [0.]
   [0.]]]

 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]

 …

 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]

 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]

 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]]
==============
```

[21]:
```
adv.reshape(-1,adv.shape[2]).shape
```

[21]: (3799, 2948)

[22]:
```
FPredAdv = F.predict(adv.reshape(-1,adv.shape[2]))
print("========================\n\nF Predict ADV\n\n")
print(FPredAdv)
print("==============\n\n")
```

```
========================

F Predict ADV


[[0.0000000e+00]
 [0.0000000e+00]
 [0.0000000e+00]
 …
 [1.3551286e-06]
 [0.0000000e+00]
 [0.0000000e+00]]
==============
```

[23]:
```
adv_pdt = np.argmax(FPredAdv, axis=1)
print("========================\n\nF Predict ADV FUNC- ADV_PDT\n\n")
print(adv_pdt)
print("==============\n\n")
print(np.unique(adv_pdt,return_counts=True))
```

```
========================

F Predict ADV FUNC- ADV_PDT


[0 0 0 … 0 0 0]
==============


(array([0], dtype=int64), array([3799], dtype=int64))
```

[24]:
```
FPredClean = F.predict(clean.reshape(-1,adv.shape[2]))
print("========================\n\nF Predict CLEAN\n\n")
print(FPredClean)
print("==============\n\n")
clean_pdt = np.argmax(FPredClean, axis=1)
```

```python
print("========================\n\nF Predict CLEAN FUNC- ADV_PDT\n\n")
print(clean_pdt)
print("==============\n\n")
print(np.unique(clean_pdt,return_counts=True))
```

```
========================

F Predict CLEAN


[[0.98784703]
 [0.9990308 ]
 [0.79184556]
 …
 [0.9997898 ]
 [0.987847  ]
 [0.9801514 ]]
==============



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

F Predict CLEAN FUNC- ADV_PDT


[0 0 0 … 0 0 0]
==============


(array([0], dtype=int64), array([3799], dtype=int64))
```

[25]:

```python
FPredPur = F.predict(purified.reshape(-1,adv.shape[2]))
print("========================\n\nF Predict PURIFIED \n\n")
print(FPredPur)
print("==============\n\n")
```

```
========================

F Predict PURIFIED


[[0.05129439]
 [0.05129439]
 [0.05129439]
 …
 [0.05129439]
 [0.05129439]
 [0.05129439]]
```

```
==============
```

```
[26]: purified_pdt = np.argmax(FPredPur, axis=1)
print("========================\n\nF Predict PURIFIED FUNC - PURIFIED_PDT\n\n")
print(purified_pdt)
print("==============\n\n")
print(np.unique(purified_pdt,return_counts=True))
```

```
========================

F Predict PURIFIED FUNC - PURIFIED_PDT


[0 0 0 … 0 0 0]
==============


(array([0], dtype=int64), array([3799], dtype=int64))
```

```
[27]: print("========================\n\nLABEL\n\n")
print(label)
print("==============\n\n")
print(np.unique(label,return_counts=True))
```

```
========================

LABEL


[1 1 1 … 1 1 1]
==============


(array([1]), array([3799], dtype=int64))
```

```
[28]: print(' adv acc: {:.10f},\n rct acc: {:.10f},\n\n\n SIMILARITY: {:.10f}'.
 ↪format( np.mean(adv_pdt==label),
                                    np.mean(purified_pdt==label), np.
 ↪mean(adv_pdt==purified_pdt)))
```

```
 adv acc: 0.0000000000,
 rct acc: 0.0000000000,


 SIMILARITY: 1.0000000000
```

[29]:
```
F.evaluate(clean.reshape(-1,2948), label)
```

```
3799/3799 [==============================] - 2s 520us/step
```

[29]:
```
[0.04282340146973827, 1.0]
```

[30]:
```
F.evaluate(adv.reshape(-1,2948), label)
```

```
3799/3799 [==============================] - 2s 443us/step
```

[30]:
```
[299.8835265824091, 0.0]
```

[31]:
```
F.evaluate(purified.reshape(-1,2948), label)
```

```
3799/3799 [==============================] - 2s 447us/step
```

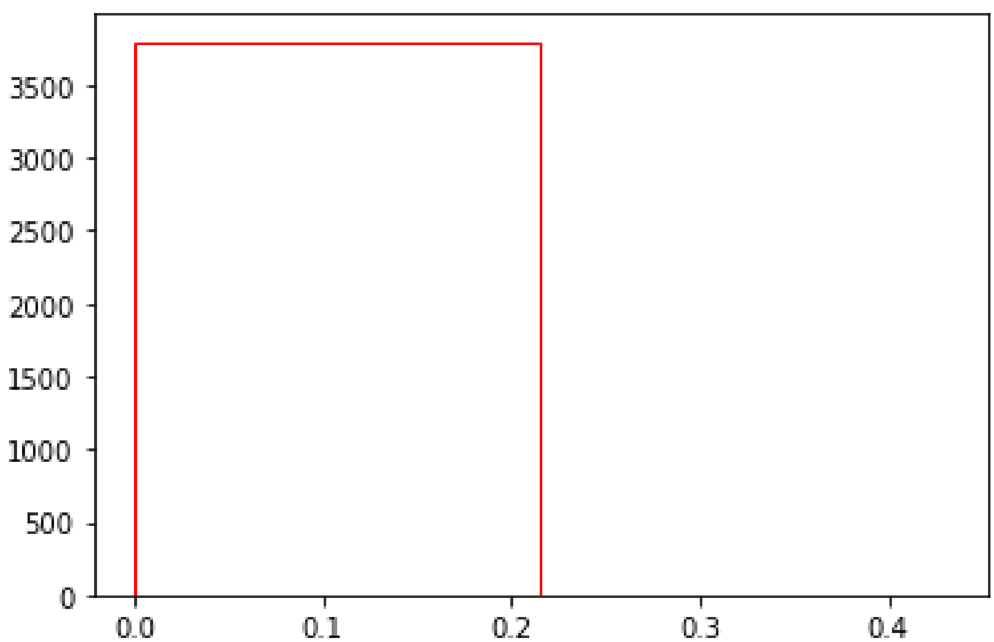
[31]:
```
[2.970173834310955, 0.0]
```

[32]:
```
FPredAdv.shape
```

[32]:
```
(3799, 1)
```

[33]:
```
import matplotlib.pyplot as plt
from scipy import stats
```

[34]:
```
plt.hist(FPredAdv,bins=2,color='white', edgecolor='red')
plt.show()
```



[35]:
```
StatsDesc = ['nobs','min,max','mean','var','skewness','kurtosis']
for i in range(len(stats.describe(FPredAdv))):
    print("===========")
    print(StatsDesc[i])
    print(stats.describe(FPredAdv)[i])
```

```
===========
nobs
3799
===========
min,max
(array([0.], dtype=float32), array([0.4306298], dtype=float32))
===========
mean
[0.00035468]
===========
var
[0.00013957]
===========
skewness
[35.453857]
===========
kurtosis
[1260.1925]
```

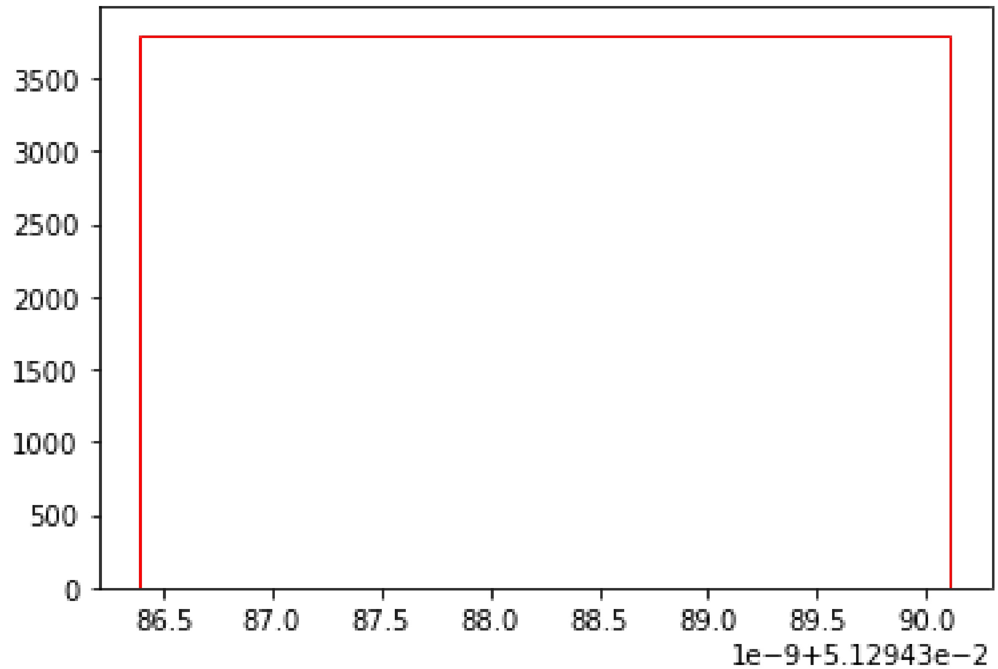
[36]:
```
print(np.unique(purified,return_counts=True))
print(purified.shape)
c = ccc = 0
for i in range(len(purified)):
    for j in range(purified.shape[2]):
        if purified[i][0][j][0]==0:
            c+=1
        else:
            ccc+=1

print(c,ccc)
```

```
(array([0.], dtype=float32), array([11199452], dtype=int64))
(3799, 1, 2948, 1)
11199452 0
```

[37]:
```
ccc/15621
```

[37]: 0.0

[38]:
```
plt.hist(FPredPur,bins=2,color='white', edgecolor='red')
plt.show()
```



[39]:
```
StatsDesc = ['nobs','min,max','mean','var','skewness','kurtosis']
for i in range(len(stats.describe(FPredPur))):
    print("===========")
    print(StatsDesc[i])
    print(stats.describe(FPredPur)[i])
```

```
===========
nobs
3799
===========
min,max
(array([0.05129439], dtype=float32), array([0.05129439], dtype=float32))
===========
mean
[0.05129439]
===========
var
[1.0961918e-20]
===========
skewness
[35.585575]
===========
```

```
kurtosis
[1263.3259]
```

6 Conclusions FGSM PURE MAL

Given that FGSM Pure MAL is also converting all to 0. We were stuck here for quite some time and then began going over to other methods. The next section discusses about those methods

[ ]:

```

```