IT581 Adversarial Machine Learning

Lab Assignment - 03

Group 05

2021 17013: Shaikh Faizan Ahmed
2021 17014: Sonam Bharti

Question 1.

In this assignment you will implement how to perform adversarial attacks using Keras and TensorFlow. Implement **The Fast Gradient Sign Method (FGSM)**,
a. The one parameter you need to tune which is η.

These attacks exploits the gradients of a neural network to build an adversarial image. The result is an output image that, according to the human eye, looks identical to the original, but makes the neural network make an incorrect prediction.
Take 10 sample images for classification using ResNet-50 and show the original class before attack and new class after attack with different hyper parameters value for all 3 attacks given above.
Comment your observation.

Answer

We are asked to implement **FGSM attack**

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import decode_predictions
from tensorflow.keras.applications.resnet50 import preprocess_input

import tensorflow as tf
from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import os
from os import listdir
from PIL import Image as PImage

img_width, img_height = 224, 224

model_pretrained = ResNet50(weights='imagenet',
                            include_top=True, input_shape=(img_height, img_width, 3))

# Mounting Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Insert correct path of your image below
img_path = ('/content/drive/My Drive/ImagesData/flower1.jpg')
img = image.load_img(img_path, target_size=(img_width, img_height))
img_data = image.img_to_array(img)
img_data1 = np.expand_dims (img_data, axis=0)
img_data = preprocess_input(img_data1)


#predict the result
cnn_feature = model_pretrained.predict(img_data, verbose=0)
# decode the results into a list of tuples (class, description, probability)
label = decode_predictions(cnn_feature)
label = label[0][0]

plt.imshow(img)
stringprint ="%.1f" % round(label[2]*100,1)
plt.title(label[1] + " " + str(stringprint) + "%", fontsize=20)
```

```
plt.axis('off')
plt.show()

# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(cnn_feature, top=3)[0])

# Insert correct path of your image folder below
folder_path = '/content/drive/My Drive/ImagesData/'
images = os.listdir(folder_path)
fig = plt.figure(figsize=(16,20))
i=0
rows=6
columns=4

for image1 in images:
   i+=1
   fimg = image.load_img(folder_path+image1, target_size=(img_width, img_height))
   fimg_data = image.img_to_array(fimg)
   fimg_data = np.expand_dims (fimg_data, axis=0)
   fimg_data = preprocess_input(fimg_data)
   cnn_feature = model_pretrained.predict(fimg_data, verbose=0)
   label = decode_predictions (cnn_feature)
   label = label[0][0]

   fig.add_subplot(rows,columns,i)
   fig.subplots_adjust(hspace=.5)

   plt.imshow(fimg)
   stringprint ="%.1f" % round(label[2]*100,1)
   plt.title(label[1] + " " + str(stringprint) + "*", fontsize=20)
   plt.axis('off')

plt.show()


#Attack begins.....

loss_object = tf.keras.losses.CategoricalCrossentropy()

def create_adversarial_pattern(input_image, input_label):
  with tf.GradientTape() as tape:
    tape.watch(input_image)
    prediction = model_pretrained(input_image)
    loss = loss_object(input_label, prediction)

  # Get the gradients of the loss w.r.t to the input image.
  gradient = tape.gradient(loss, input_image)
  # Get the sign of the gradients to create the perturbation
  signed_grad = tf.sign(gradient)
  return signed_grad

# Get the input label of the image.

input_image_index = 985
label = tf.one_hot(input_image_index, cnn_feature.shape[-1])
label = tf.reshape(label, (1, cnn_feature.shape[-1]))

perturbations = create_adversarial_pattern(tf.convert_to_tensor(img_data), label)
plt.imshow(perturbations[0] * 0.5 + 0.5);  # To change [-1, 1] to [0,1]


def display_images(image, description):
  cnn_pfeature = model_pretrained.predict(image, verbose=0)
  label = decode_predictions (cnn_pfeature, top=1)[0][0]

  confidence ="%.2f" % round(label[2]*100,1)
  plt.figure()
  plt.imshow(image[0]*0.5+0.5)

  plt.title('{} \n {} : {}% Confidence'.format(description,
                                               label[1], str(confidence)))
  plt.show()


```

```
eta = [0, 11, 34, 56, 75]
descriptions = [('eta = {:0.3f}'.format(eps) if eps else 'Input')
                for eps in eta]

for i, eps in enumerate(eta):
  adv_x = tf.convert_to_tensor(img_data) + eps*perturbations
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])

print('Predicted:', decode_predictions(cnn_feature, top=4)[0])
```

Listing 1: FGSM Attack

Output:

Figure 1: Prediction without Perturbation

Figure 2: Prediction of whole image folder

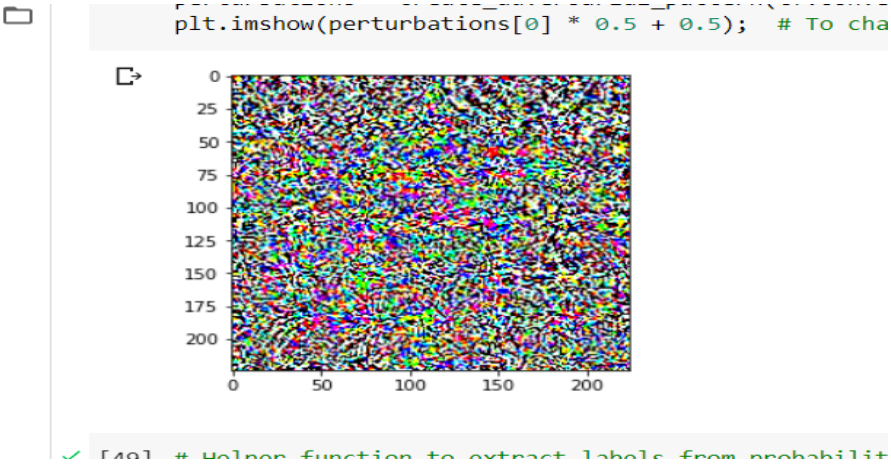
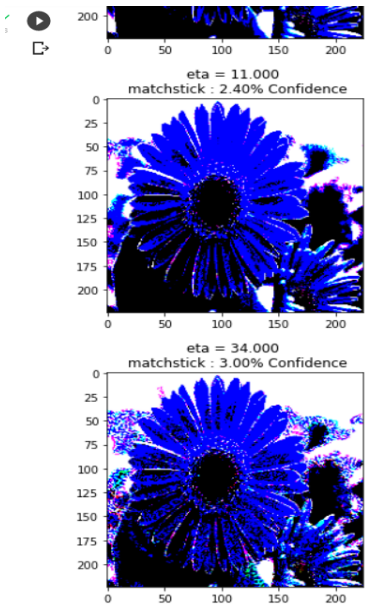
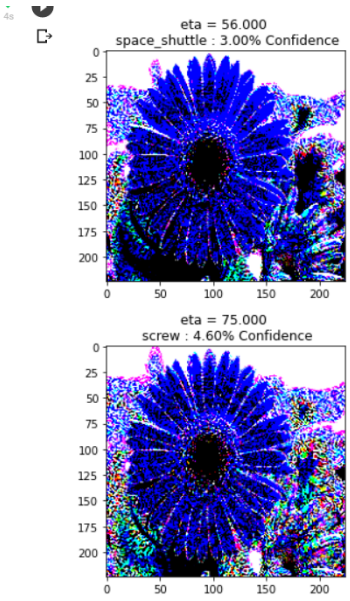
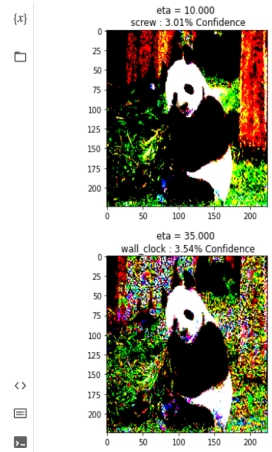
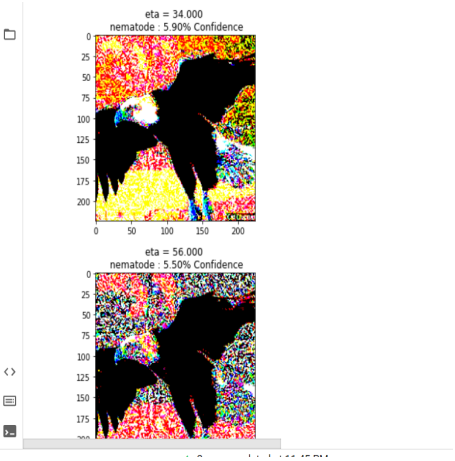


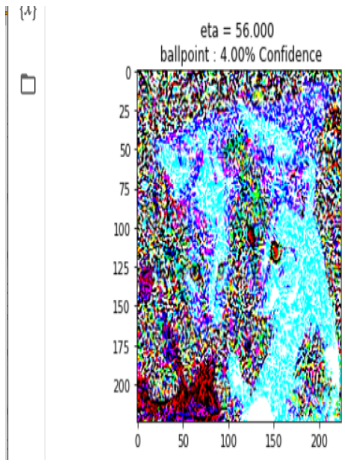
Figure 3: Perturbation Image

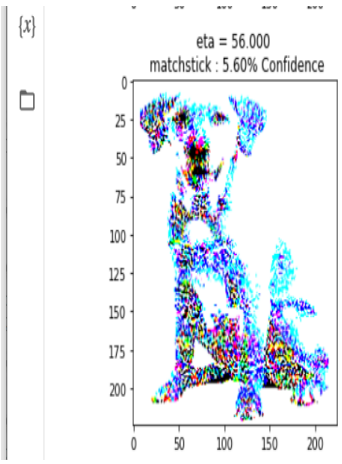

eta = 11.000
matchstick : 2.40% Confidence
eta = 34.000
matchstick : 3.00% Confidence

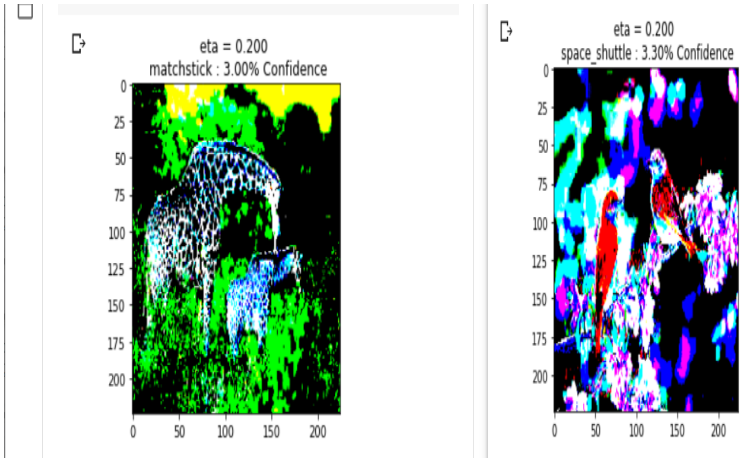

eta = 56.000
space_shuttle : 3.00% Confidence
eta = 75.000
screw : 4.60% Confidence


eta = 10.000
screw : 3.01% Confidence
eta = 35.000
wall_clock : 3.54% Confidence


eta = 34.000
nematode : 5.90% Confidence
eta = 56.000
nematode : 5.50% Confidence


eta = 56.000
ballpoint : 4.00% Confidence


eta = 56.000
matchstick : 5.60% Confidence



Observation / Justification:

Figure 2 shows all the prediction without an attack to 10 images. Figure 3 shows the perturbation Image which is added in each image to perform the FGSM attack.
As seen in the figure 4, images are been miss classified after the FGSM attack

Question 2.

Follow the same instruction as in Question 1. and
Implement **The Iterative Fast Gradient Sign Method (PGD)**,
a. The one parameter you need to tune which is η.

Answer

```
pip install foolbox

import warnings
warnings.filterwarnings('ignore')

from foolbox.models import TensorFlowModel
fooled_model = TensorFlowModel(model_pretrained, bounds=(-255, 255))

from foolbox.criteria import TargetedMisclassification

orig_class = 985
target_class = 24
target_label = tf.convert_to_tensor([target_class])

criterion = TargetedMisclassification(target_label)

from foolbox.attacks import PGD
eps = 37
attack_pgd = PGD()
x_adv_pgd = attack_pgd.run(fooled_model, tf.convert_to_tensor(img_data), criterion, epsilon=
    eps)

def display_pgd_images(image, eps):
  cnn_pfeature = model_pretrained.predict(image, verbose=0)
  label = decode_predictions (cnn_pfeature, top=1)[0][0]

  confidence ="%.2f" % round(label[2]*100,1)
  plt.figure()
  plt.imshow(image[0]*0.5+0.5)

  plt.title('eta = {} \n {} : {}% Confidence'.format(eps,
                                                     label[1], str(confidence)))
  plt.show()

x_adv_pgd = tf.clip_by_value(x_adv_pgd,-1,1)
display_pgd_images(x_adv_pgd, eps)

```

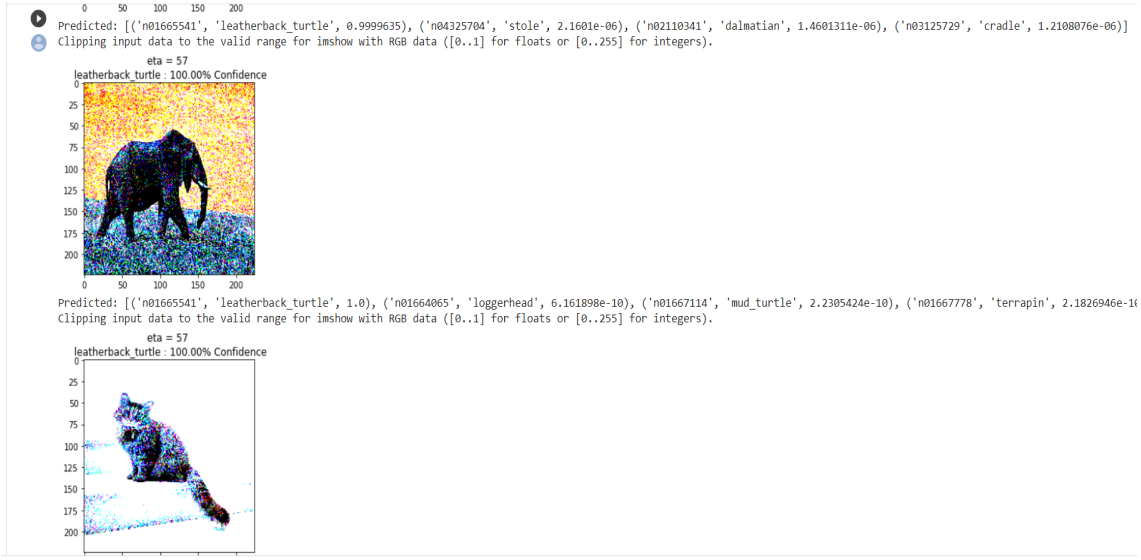
```
# Insert correct path of your image folder below
from foolbox.attacks import PGD
folder_path = '/content/drive/My Drive/ImagesData1/'
images = os.listdir(folder_path)
fig = plt.figure(figsize=(16,20))
i=0
rows=5
columns=2

for image1 in images:
    i+=1
    fimg = image.load_img(folder_path+image1, target_size=(img_width, img_height))
    fimg_data = image.img_to_array(fimg)
    fimg_data = np.expand_dims (fimg_data, axis=0)
    fimg_data = preprocess_input(fimg_data)
    cnn_feature = model_pretrained.predict(fimg_data, verbose=0)
    label = decode_predictions (cnn_feature)
    label = label[0][0]

    eps = 57
    attack_pgd = PGD()
    x_adv_pgd = attack_pgd.run(fooled_model, tf.convert_to_tensor(fimg_data), criterion,
     epsilon=eps)

    display_pgd_images(x_adv_pgd, eps)

    model_pgd = model_pretrained.predict(x_adv_pgd, verbose=0)
    print('Predicted:', decode_predictions(model_pgd, top=4)[0])
```

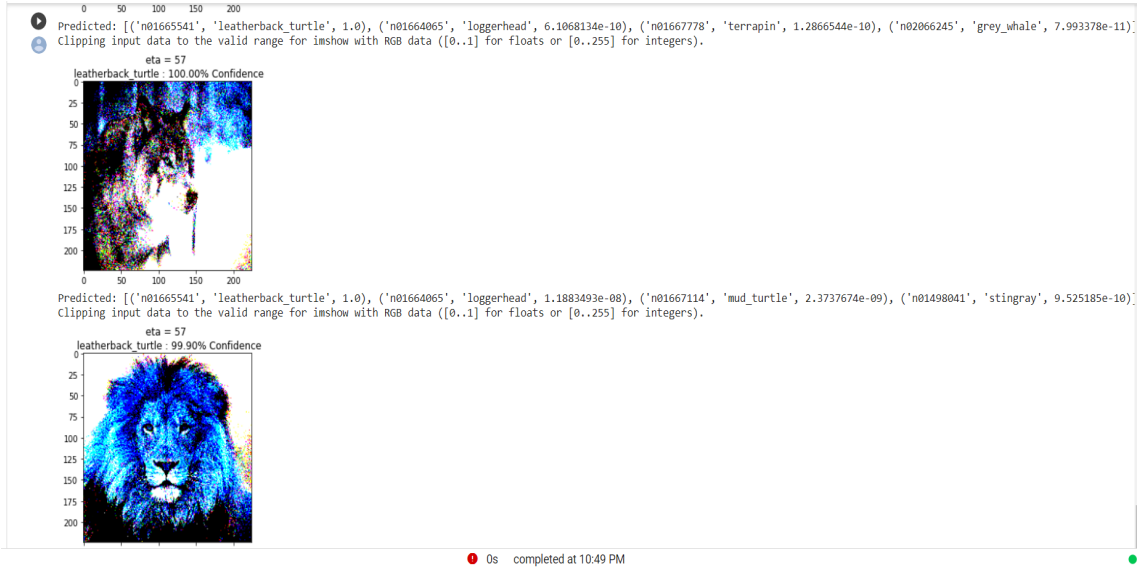
Listing 2: PGD attack

Output:

Figure 4: PGD attack with eps = 37

Figure 5: PGD attack with eps = 57


Predicted: [('n01665541', 'leatherback_turtle', 0.9999635), ('n04325704', 'stole', 2.1601e-06), ('n02110341', 'dalmatian', 1.4601311e-06), ('n03125729', 'cradle', 1.2108076e-06)]
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
eta = 57
leatherback_turtle : 100.00% Confidence
Predicted: [('n01665541', 'leatherback_turtle', 1.0), ('n01664065', 'loggerhead', 6.161898e-10), ('n01667114', 'mud_turtle', 2.2305424e-10), ('n01667778', 'terrapin', 2.1826946e-10
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
eta = 57
leatherback_turtle : 100.00% Confidence
0s completed at 10:49 PM


Predicted: [('n01665541', 'leatherback_turtle', 1.0), ('n01664065', 'loggerhead', 6.1068134e-10), ('n01667778', 'terrapin', 1.2866544e-10), ('n02066245', 'grey_whale', 7.993378e-11)]
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
eta = 57
leatherback_turtle : 100.00% Confidence
Predicted: [('n01665541', 'leatherback_turtle', 1.0), ('n01664065', 'loggerhead', 1.1883493e-08), ('n01667114', 'mud_turtle', 2.3737674e-09), ('n01498041', 'stingray', 9.525185e-10)]
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
eta = 57
leatherback_turtle : 99.90% Confidence
0s completed at 10:49 PM


Predicted: [('n01665541', 'leatherback_turtle', 1.0), ('n01664065', 'loggerhead', 1.1883493e-08), ('n01667114', 'mud_turtle', 2.3737674e-09), ('n01498041', 'stingray', 9.525185e-10)]
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
eta = 57
leatherback_turtle : 99.90% Confidence
Predicted: [('n01665541', 'leatherback_turtle', 0.9992083), ('n01704323', 'triceratops', 4.079118e-05), ('n03028079', 'church', 1.4845177e-05), ('n01742172', 'boa_constrictor', 1.41
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
eta = 57
leatherback_turtle : 100.00% Confidence

Definition:

The PGD attack is a white-box attack which means the attacker has access to the model gradients i.e. the attacker has a copy of your model's weights. This threat model gives the attacker much more power than black box attacks as they can specifically craft their attack to fool your model without having to rely on transfer attacks that often result in human-visible perturbations. PGD can be considered the most "complete" white-box adversary as it lifts any constraints on the amount of time and effort the attacker can put into finding the best attack.

Observation / Justification:

We performed a targeted attack and all the 10 images are classified as leatherback_turtle with 100% confidence.
By using the optimization

$$x^{k+1} = argmin_{0<=x<=1}\delta J(x(k); w)^T x$$

such that

$$||x = x^{(k)}||_\infty$$

Question 3.

Follow the same instruction as in Question 1. and
Implement **Carlini Wagner Attack (C-W attack)**
a. The two parameters you need to tune are
i. α : $Gradientdescentstepsize$
ii. λ : $Regularizationparameter$
Comment your observation.

Answer

code:

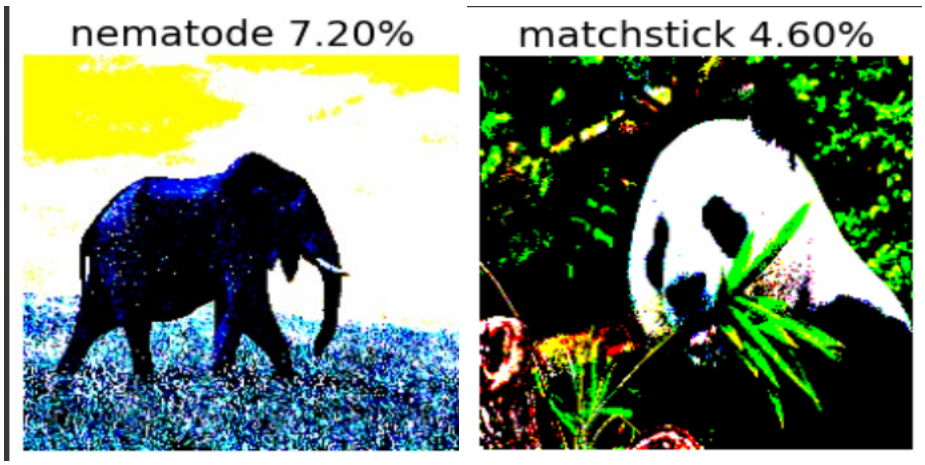
```
!pip install adversarial-robustness-toolbox

from art.estimators.classification import TensorFlowV2Classifier
from art.attacks.evasion import CarliniL2Method

import warnings
warnings.filterwarnings('ignore')

classifier = TensorFlowV2Classifier(model=model_pretrained, nb_classes=1000, input_shape=(
    img_width, img_height, 1), loss_object=loss_object,clip_values=(0, 1), channels_first=
    False)

from art.attacks.evasion.carlini import CarliniL2Method
attack_cw = CarliniL2Method(classifier=classifier,max_iter=10,learning_rate=0.01)

x_test_adv = attack_cw.generate(img_data1);

predictions = classifier.predict(x_test_adv)
print(np.argmax(predictions, axis=1))

label = decode_predictions(classifier.predict(x_test_adv))
label = label[0][0]

predictions = classifier.predict(x_test_adv)

confidence ="%.2f" % round(label[2]*100,1)

plt.imshow(x_test_adv[0])
plt.title(label[1] + " " + format(confidence) + "%", fontsize=20)
plt.axis('off')
plt.show()

```

```
print('Predicted:', decode_predictions(predictions, top=3)[0])
```

Listing 3: Carlini Wagner Attack

Output:

Figure 6: Carlini Wagner Attack Prediction



Observation / Justification:

We use adversarial-robustness-toolbox, CarliniL2Method library to perform CW attack. As shown all 10 images are miss-classified.