

# 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  $\eta$ .

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**

```
1 from tensorflow.keras.applications.resnet50 import ResNet50
2 from tensorflow.keras.applications.resnet50 import decode_predictions
3 from tensorflow.keras.applications.resnet50 import preprocess_input
4
5 import tensorflow as tf
6 from keras.preprocessing import image
7 import numpy as np
8 import matplotlib.pyplot as plt
9 import os
10 from os import listdir
11 from PIL import Image as PImage
12
13 img_width, img_height = 224, 224
14
15 model_pretrained = ResNet50(weights='imagenet',
16                             include_top=True, input_shape=(img_height, img_width, 3))
17
18 # Mounting Google Drive
19 from google.colab import drive
20 drive.mount('/content/drive')
21
22 # Insert correct path of your image below
23 img_path = ('/content/drive/My Drive/ImagesData/flower1.jpg')
24 img = image.load_img(img_path, target_size=(img_width, img_height))
25 img_data = image.img_to_array(img)
26 img_data1 = np.expand_dims(img_data, axis=0)
27 img_data = preprocess_input(img_data1)
28
29
30 #predict the result
31 cnn_feature = model_pretrained.predict(img_data, verbose=0)
32 # decode the results into a list of tuples (class, description, probability)
33 label = decode_predictions(cnn_feature)
34 label = label[0][0]
35
36 plt.imshow(img)
37 stringprint = "% .1f" % round(label[2]*100,1)
38 plt.title(label[1] + " " + str(stringprint) + "%", fontsize=20)
```

```

39 plt.axis('off')
40 plt.show()
41
42 # decode the results into a list of tuples (class, description, probability)
43 # (one such list for each sample in the batch)
44 print('Predicted:', decode_predictions(cnn_feature, top=3)[0])
45
46 # Insert correct path of your image folder below
47 folder_path = '/content/drive/My Drive/ImagesData/'
48 images = os.listdir(folder_path)
49 fig = plt.figure(figsize=(16,20))
50 i=0
51 rows=6
52 columns=4
53
54 for image1 in images:
55     i+=1
56     fimg = image.load_img(folder_path+image1, target_size=(img_width, img_height))
57     fimg_data = image.img_to_array(fimg)
58     fimg_data = np.expand_dims (fimg_data, axis=0)
59     fimg_data = preprocess_input(fimg_data)
60     cnn_feature = model_pretrained.predict(fimg_data, verbose=0)
61     label = decode_predictions (cnn_feature)
62     label = label[0][0]
63
64     fig.add_subplot(rows,columns,i)
65     fig.subplots_adjust(hspace=.5)
66
67     plt.imshow(fimg)
68     stringprint = "% .1f" % round(label[2]*100,1)
69     plt.title(label[1] + " " + str(stringprint) + "*", fontsize=20)
70     plt.axis('off')
71
72 plt.show()
73
74
75 #Attack begins.....
76
77 loss_object = tf.keras.losses.CategoricalCrossentropy()
78
79 def create_adversarial_pattern(input_image, input_label):
80     with tf.GradientTape() as tape:
81         tape.watch(input_image)
82         prediction = model_pretrained(input_image)
83         loss = loss_object(input_label, prediction)
84
85     # Get the gradients of the loss w.r.t to the input image.
86     gradient = tape.gradient(loss, input_image)
87     # Get the sign of the gradients to create the perturbation
88     signed_grad = tf.sign(gradient)
89     return signed_grad
90
91 # Get the input label of the image.
92
93 input_image_index = 985
94 label = tf.one_hot(input_image_index, cnn_feature.shape[-1])
95 label = tf.reshape(label, (1, cnn_feature.shape[-1]))
96
97 perturbations = create_adversarial_pattern(tf.convert_to_tensor(img_data), label)
98 plt.imshow(perturbations[0] * 0.5 + 0.5); # To change [-1, 1] to [0,1]
99
100
101 def display_images(image, description):
102     cnn_pfeature = model_pretrained.predict(image, verbose=0)
103     label = decode_predictions (cnn_pfeature, top=1)[0][0]
104
105     confidence = "% .2f" % round(label[2]*100,1)
106     plt.figure()
107     plt.imshow(image[0]*0.5+0.5)
108
109     plt.title('{} \n {} : {}% Confidence'.format(description,
110                                                 label[1], str(confidence)))
111
112     plt.show()
113

```

```

114 eta = [0, 11, 34, 56, 75]
115 descriptions = [('eta = {:.3f}'.format(eps) if eps else 'Input')
116     for eps in eta]
117
118 for i, eps in enumerate(eta):
119     adv_x = tf.convert_to_tensor(img_data) + eps*perturbations
120     adv_x = tf.clip_by_value(adv_x, -1, 1)
121     display_images(adv_x, descriptions[i])
122
123 print('Predicted:', decode_predictions(cnn_feature, top=4)[0])

```

Listing 1: FGSM Attack

## Output:



```

✓ [109] # 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])

```

```
Predicted: [('n11939491', 'daisy', 0.9924114), ('n02319095', 'sea_urchin', 0.0018165808), ('n04525038', 'velvet', 0.0015322213)]
```

Figure 1: Prediction without Perturbation

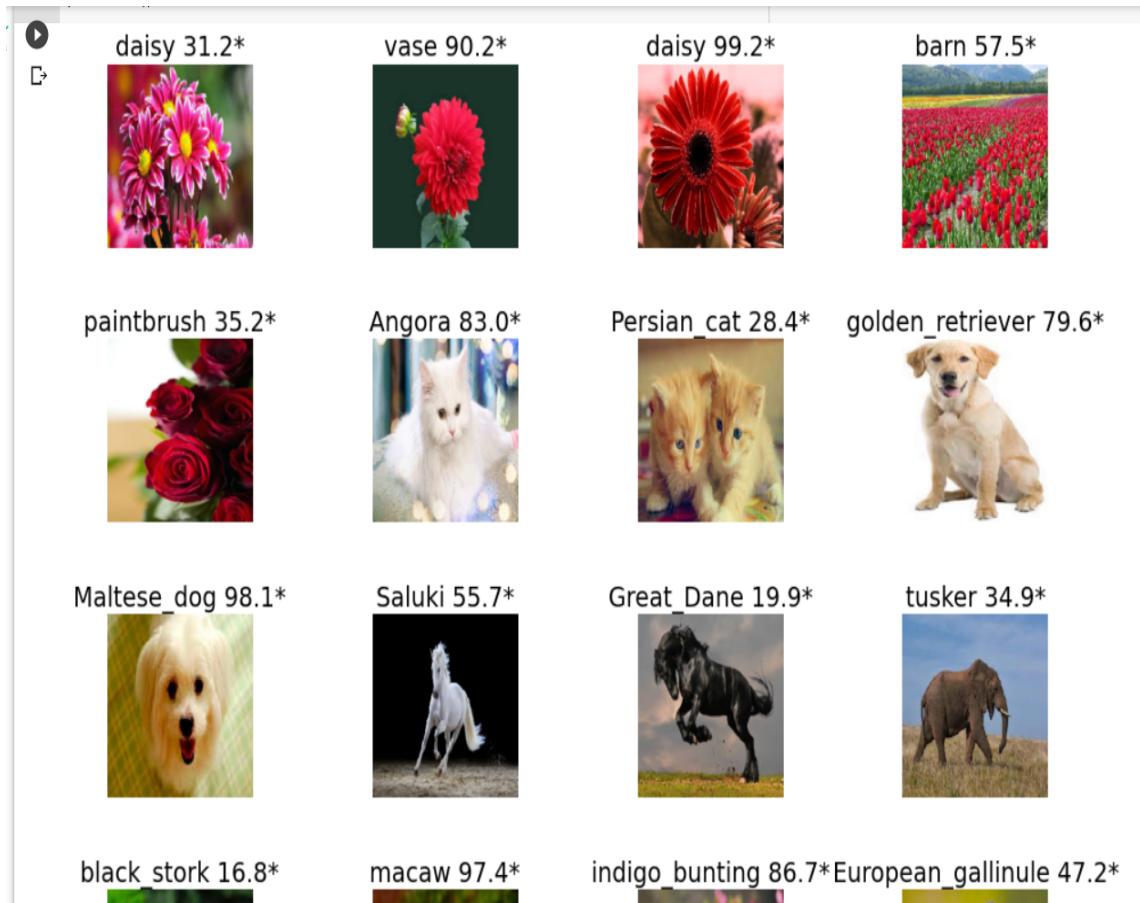


Figure 2: Prediction of whole image folder

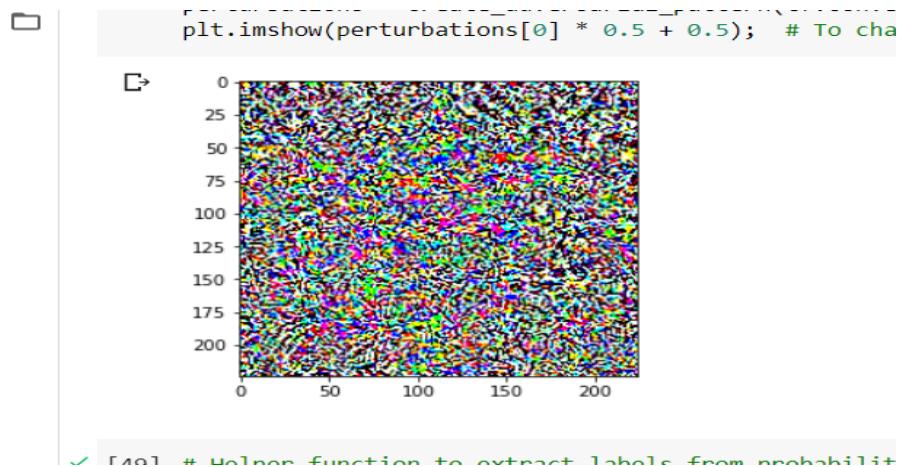
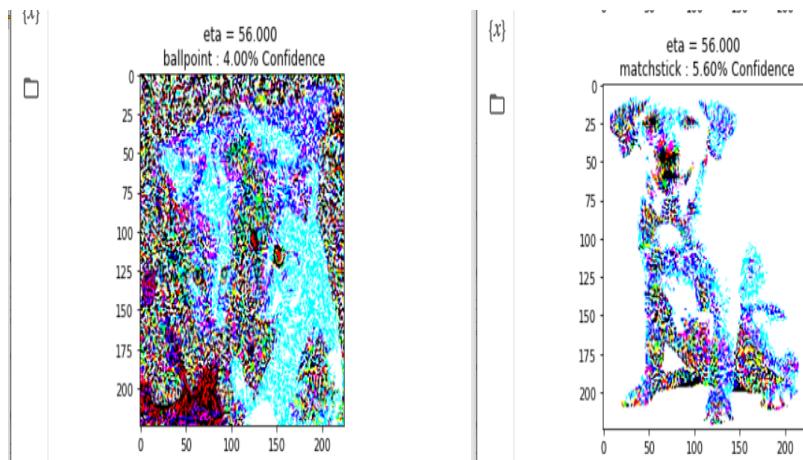
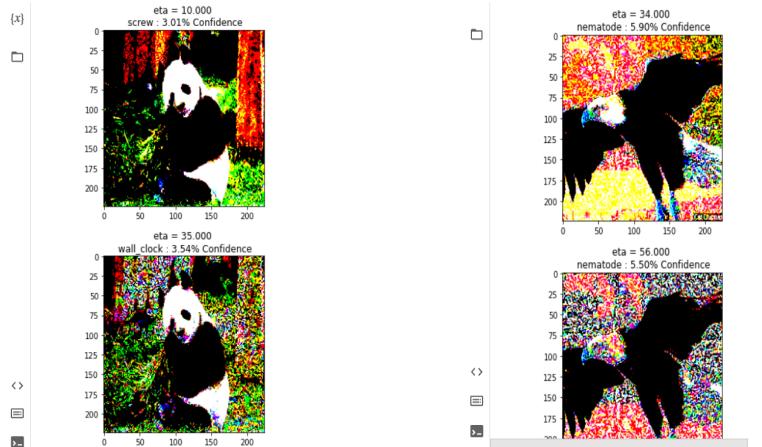
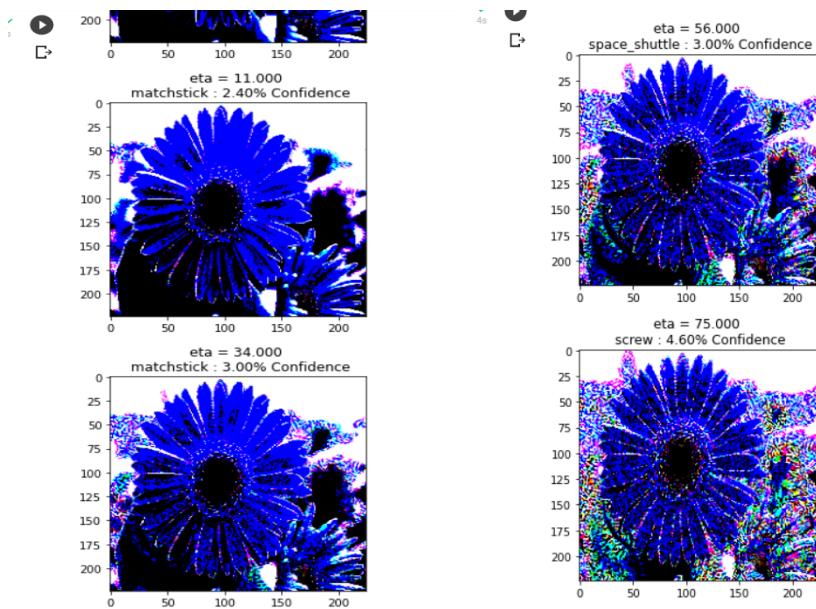
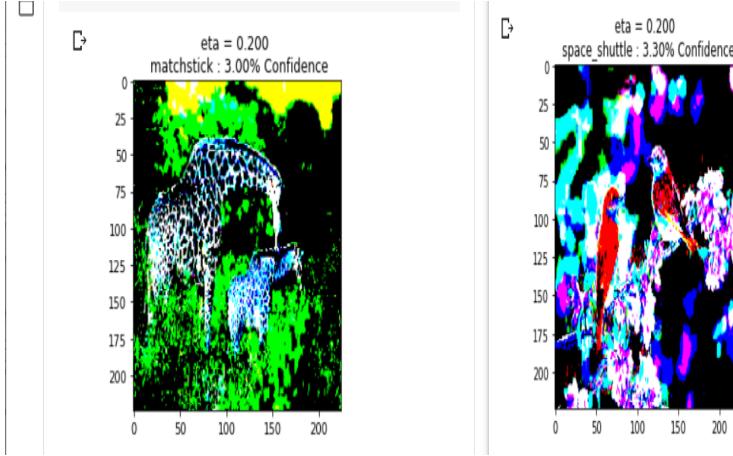


Figure 3: Perturbation Image





### 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  $\eta$ .

### Answer

```

1 pip install foolbox
2
3 import warnings
4 warnings.filterwarnings('ignore')
5
6 from foolbox.models import TensorFlowModel
7 fooled_model = TensorFlowModel(model_pretrained, bounds=(-255, 255))
8
9 from foolbox.criteria import TargetedMisclassification
10
11 orig_class = 985
12 target_class = 24
13 target_label = tf.convert_to_tensor([target_class])
14
15 criterion = TargetedMisclassification(target_label)
16
17 from foolbox.attacks import PGD
18 eps = 37
19 attack_pgd = PGD()
20 x_adv_pgd = attack_pgd.run(fooled_model, tf.convert_to_tensor(img_data), criterion, epsilon=
    eps)
21
22 def display_pgd_images(image, eps):
23     cnn_pfeature = model_pretrained.predict(image, verbose=0)
24     label = decode_predictions(cnn_pfeature, top=1)[0][0]
25
26     confidence = "% .2f" % round(label[2]*100,1)
27     plt.figure()
28     plt.imshow(image[0]*0.5+0.5)
29
30     plt.title('eta = {} \n {} : {}% Confidence'.format(eps,
31                                         label[1], str(confidence)))
32     plt.show()
33
34 x_adv_pgd = tf.clip_by_value(x_adv_pgd, -1, 1)
35 display_pgd_images(x_adv_pgd, eps)
36

```

```

37 # Insert correct path of your image folder below
38 from foolbox.attacks import PGD
39 folder_path = '/content/drive/My Drive/ImagesData1/'
40 images = os.listdir(folder_path)
41 fig = plt.figure(figsize=(16,20))
42 i=0
43 rows=5
44 columns=2
45
46 for image1 in images:
47     i+=1
48     fimg = image.load_img(folder_path+image1, target_size=(img_width, img_height))
49     fimg_data = image.img_to_array(fimg)
50     fimg_data = np.expand_dims (fimg_data, axis=0)
51     fimg_data = preprocess_input(fimg_data)
52     cnn_feature = model_pretrained.predict(fimg_data, verbose=0)
53     label = decode_predictions (cnn_feature)
54     label = label [0] [0]
55
56     eps = 57
57     attack_pgd = PGD()
58     x_adv_pgd = attack_pgd.run(fooled_model, tf.convert_to_tensor(fimg_data), criterion,
59                               epsilon=eps)
60
61     display_pgd_images(x_adv_pgd, eps)
62
63     model_pgd = model_pretrained.predict(x_adv_pgd, verbose=0)
64     print('Predicted:', decode_predictions(model_pgd, top=4) [0])

```

Listing 2: PGD attack

## Output:



Figure 4: PGD attack with  $\epsilon = 37$

```

eta = 57
leatherback_turtle : 100.00% Confidence
0 25 50 75 100 125 150 175 200

model_pgd = model_pretrained.predict(x_adv_pgd, verbose=0)
print('Predicted:', decode_predictions(model_pgd, top=4)[0])
Predicted: [('n01665541', 'leatherback_turtle', 0.99999213), ('n02097209', 'standard_schnauzer', 4.7078734e-07), ('n03388043', 'fountain', 4.6198494e-07), ('n03724870', 'mask', 3.39325

```

Figure 5: PGD attack with  $\text{eps} = 57$

```

+ Code + text
[ ] Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
<Figure size 115x1440 with 0 Axes>
eta = 57
leatherback_turtle : 100.00% Confidence
0 25 50 75 100 125 150 175 200

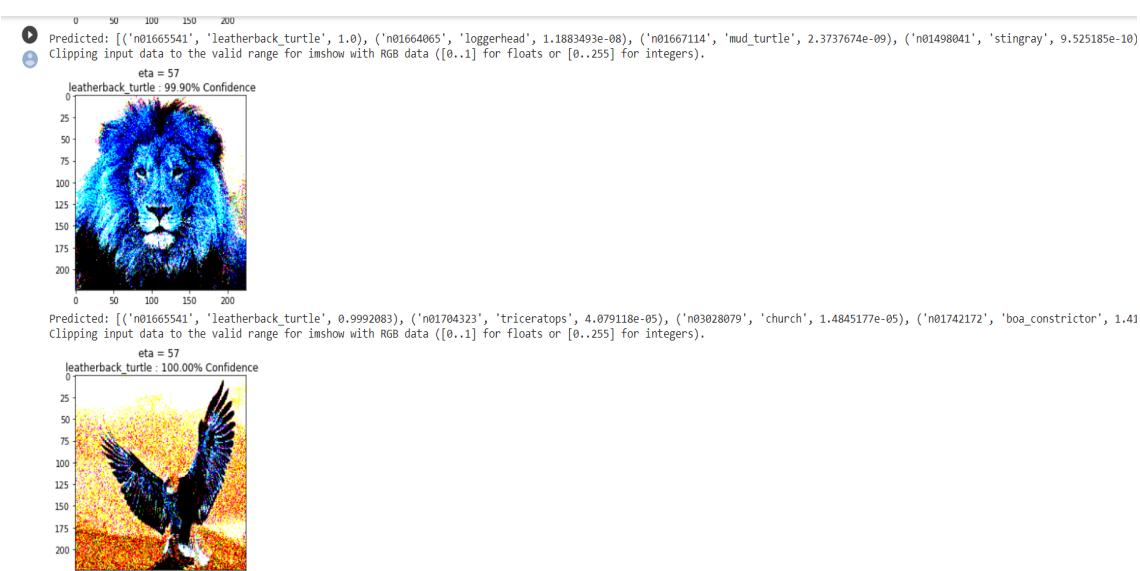
Predicted: [('n01665541', 'leatherback_turtle', 0.99998605), ('n01667778', 'terrapin', 1.0217989e-06), ('n01685808', 'whiptail', 4.57888e-07), ('n01698640', 'American_alligator', 2.
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
eta = 57
leatherback_turtle : 100.00% Confidence
0 25 50 75 100 125 150 175 200

```

```

0 50 100 150 200
Predicted: [('n01665541', 'leatherback_turtle', 1.0), ('n04200210', 'shovel', 1.1926624e-10), ('n03388043', 'fountain', 7.153679e-11), ('n02177972', 'weevil', 5.672631e-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
0 25 50 75 100 125 150 175 200
Predicted: [('n01665541', 'leatherback_turtle', 0.9999846), ('n03724870', 'mask', 6.145416e-07), ('n02676566', 'acoustic_guitar', 5.461334e-07), ('n03388043', 'fountain', 5.314875e-0
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
0 25 50 75 100 125 150 175 200

```



### 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

$$x^{k+1} = \operatorname{argmin}_{0 \leq x \leq 1} \delta J(x(k); w)^T x$$

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.  $\alpha$  : Gradient descent step size
- ii.  $\lambda$  : Regularization parameter

Comment your observation.

### Answer

code:

```
1 !pip install adversarial-robustness-toolbox
2
3 from art.estimators.classification import TensorFlowV2Classifier
4 from art.attacks.evasion import CarliniL2Method
5
6 import warnings
7 warnings.filterwarnings('ignore')
8
9 classifier = TensorFlowV2Classifier(model=model_pretrained, nb_classes=1000, input_shape=(
10     img_width, img_height, 1), loss_object=loss_object, clip_values=(0, 1), channels_first=False)
11
12 from art.attacks.evasion.carlini import CarliniL2Method
13 attack_cw = CarliniL2Method(classifier=classifier, max_iter=10, learning_rate=0.01)
14
15 x_test_adv = attack_cw.generate(img_data1);
16
17 predictions = classifier.predict(x_test_adv)
18 print(np.argmax(predictions, axis=1))
19
20 label = decode_predictions(classifier.predict(x_test_adv))
21 label = label[0][0]
22
23 predictions = classifier.predict(x_test_adv)
24 confidence = "% .2f" % round(label[2]*100,1)
25
26 plt.imshow(x_test_adv[0])
27 plt.title(label[1] + " " + format(confidence) + "%", fontsize=20)
28 plt.axis('off')
29 plt.show()
```

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

Listing 3: Carlini Wagner Attack

**Output:**

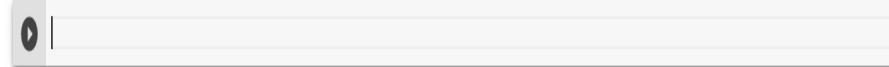
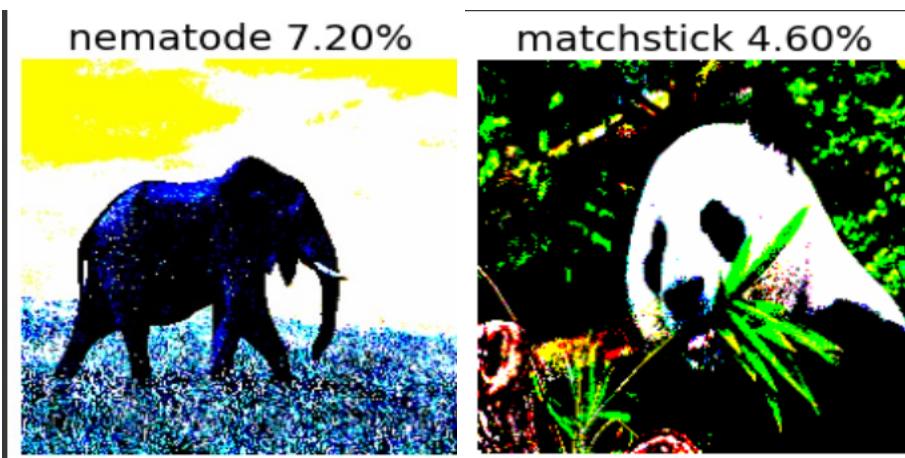
```
] Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
sea_urchin 48.10%
  
[80] print('Predicted:', decode_predictions(predictions, top=3)[0])
Predicted: [('n02319095', 'sea_urchin', 0.48072076), ('n11939491', 'daisy', 0.19630097), ('n07730033', 'cardoon', 0.061766487)]  

```

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.