IT581 Adversarial Machine Learning

Lab Assignment - 5

Group 05

2021 17013: Shaikh Faizan Ahmed
2021 17014: Sonam Bharti

Question 1.

In this exercise use **Uniform noise** and Correlated Gaussian noise and compare the probability of a successful attack for all three cases.
Write code for this and show result for the sample images. You can use any classification technique and data to perform this task.
Comment your observation.

Answer

code

```
# Import libraries
!pip install adversarial-robustness-toolbox

import cv2
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import os
import sys
import PIL
from PIL import Image
from matplotlib.pyplot import figure

import torch.nn as nn
from PIL import Image
from torchvision import transforms
import torch.optim as optim
import torchvision.models as models
from art.estimators.classification import PyTorchClassifier
import warnings
warnings.filterwarnings('ignore')
import scipy.signal

from tensorflow.keras.applications.resnet50 import decode_predictions

# Softmax function
def softmax_activation(inputs):
    inputs = inputs.tolist()
    exp_values = np.exp(inputs - np.max(inputs))

    # Normalize
    probabilities = exp_values / np.sum(exp_values)
    return probabilities

# image transformation

preprocess = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor()])

model_resnet50 = models.resnet50(pretrained=True)

criterion = nn.CrossEntropyLoss()
```

```

# Create the ART classifier

classifier = PyTorchClassifier(
    model=model_resnet50,
    loss=criterion,
    input_shape=(3, 224, 224),
    nb_classes=1000,
    device_type='gpu')
```

Now We will classify the original Image.

```
img = cv2.imread('/content/panda.jpg')

color_coverted = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
original_image = Image.fromarray(color_coverted)

input_tensor = preprocess(original_image)
input_batch = input_tensor.unsqueeze(0).numpy().astype(np.float32)
input = input_batch[0].transpose((1,2,0))


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

preds = classifier.predict(input_batch)

accuracy = np.max(softmax_activation(preds), axis=1)
accuracy = round(accuracy[0], 2)

plt.imshow(input)
plt.title(label[1] + " " + format(accuracy * 100) + "%", fontsize=20)
plt.axis('off')
plt.show()
```

The given below figure represents the classification of original image or say, clean image.

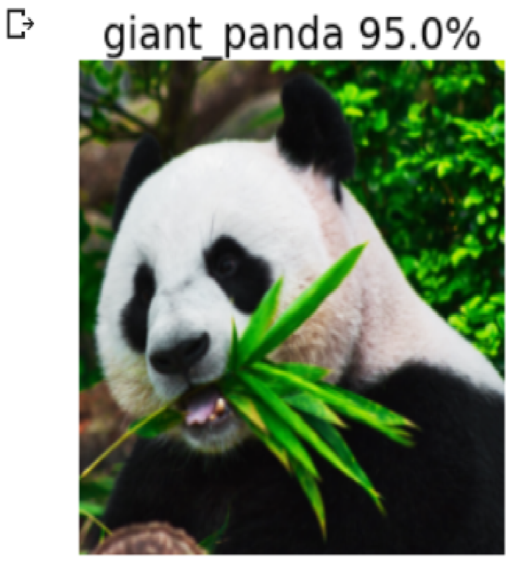
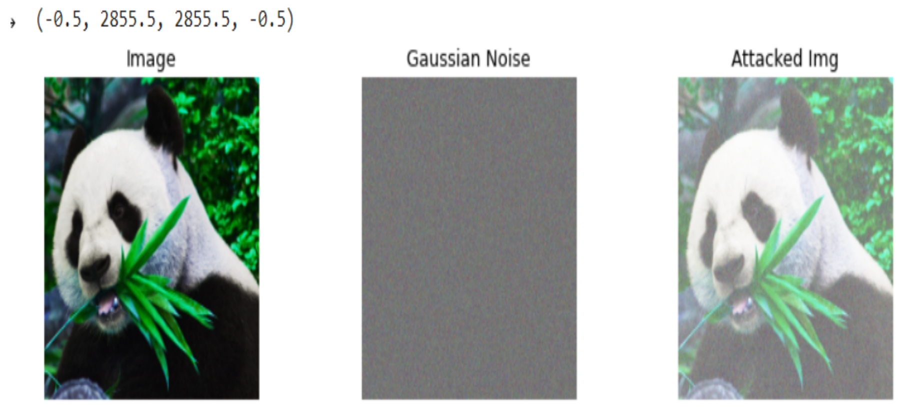


Figure 1:

Gaussian Noise Attack

```python
figure(figsize=(12, 10), dpi=80)

plt.subplot(331)
plt.imshow(img)
plt.title('Image')
plt.axis('off')


# Generate Gaussian noise
gauss = np.random.normal(1,20,img.size)
gauss = gauss.reshape(img.shape[0],img.shape[1],img.shape[2]).astype('uint8')

plt.subplot(332)
plt.imshow(gauss)
plt.title('Gaussian Noise')
plt.axis('off')


# Add the Gaussian noise to the image
img_gauss = cv2.add(img,gauss)

plt.subplot(333)
plt.imshow(img_gauss)
plt.title('Attacked Img')
plt.axis('off')
```

Output



```
(-0.5, 2855.5, 2855.5, -0.5)
```

Figure 2:

Now Classifying the Adversarial Image, we get the below output.

```python
color_coverted = cv2.cvtColor(img_gauss, cv2.COLOR_BGR2RGB)
adv_image = Image.fromarray(color_coverted)

input_tensor = preprocess(adv_image)
input_batch1 = input_tensor.unsqueeze(0).numpy().astype(np.float32)
adv_input = input_batch1[0].transpose((1,2,0))

label = decode_predictions(classifier.predict(input_batch1))
label = label[0][0]

preds = classifier.predict(input_batch1)

accuracy = np.max(softmax_activation(preds), axis=1)
accuracy = round(accuracy[0], 2)

plt.imshow(adv_input)
plt.title(label[1] + " " + format(accuracy * 100) + "%", fontsize=20)
plt.axis('off')
plt.show()
```

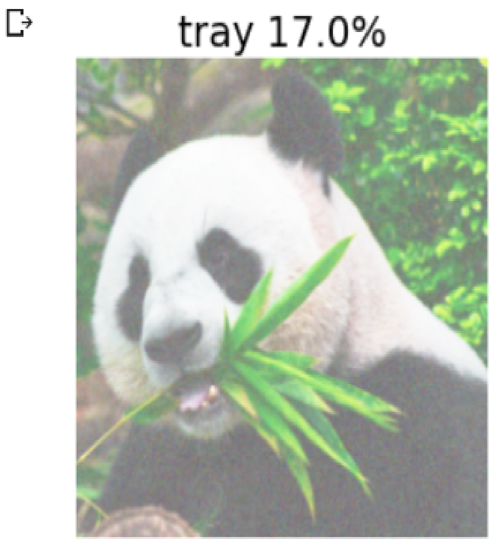
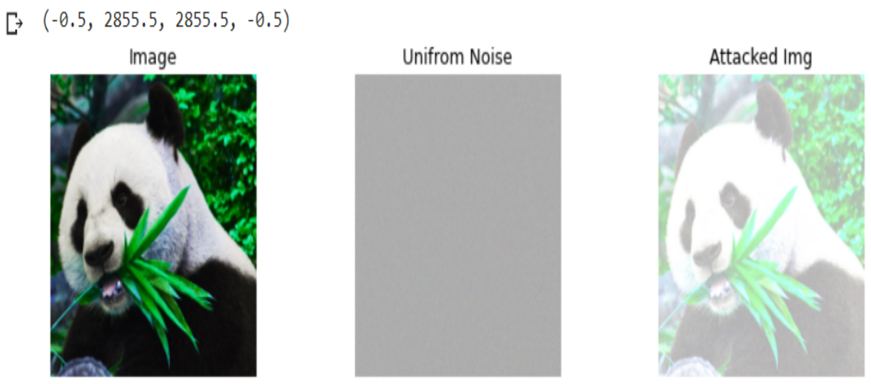


Figure 3:

Uniform Noise Attack

```
figure(figsize=(12, 10), dpi=80)

plt.subplot(331)
plt.imshow(img)
plt.title('Image')
plt.axis('off')

#plt.resize(img,height=33,width=45)

uniform = np.random.uniform(100,250,img.size)
uniform_noise = uniform.reshape(img.shape[0],img.shape[1],img.shape[2]).astype('uint8')
#uniform_noise = cv2.randu(uniform_noise,0,255)
plt.subplot(332)
plt.imshow(uniform_noise)
plt.title('Unifrom Noise')
plt.axis('off')

#uniform_noise = (uniform_noise*0.5).astype(np.uint8)
noisy_image2 = cv2.add(img,uniform_noise)

plt.subplot(333)
plt.imshow(noisy_image2)
plt.title('Attacked Img')
plt.axis('off')
```



```
(-0.5, 2855.5, 2855.5, -0.5)
```

Figure 4:

Now Classifying the Adversarial Image, we get the below output.

```
color_coverted = cv2.cvtColor(noisy_image2, cv2.COLOR_BGR2RGB)
adv_image = Image.fromarray(color_coverted)

input_tensor = preprocess(adv_image)
input_batch2 = input_tensor.unsqueeze(0).numpy().astype(np.float32)
adv_input = input_batch2[0].transpose((1,2,0))

label = decode_predictions(classifier.predict(input_batch2))
label = label[0][0]

preds = classifier.predict(input_batch2)

accuracy = np.max(softmax_activation(preds), axis=1)
accuracy = round(accuracy[0], 2)

plt.imshow(adv_input)
plt.title(label[1] + " " + format(accuracy * 100) + "%", fontsize=20)
plt.axis('off')
plt.show()
```

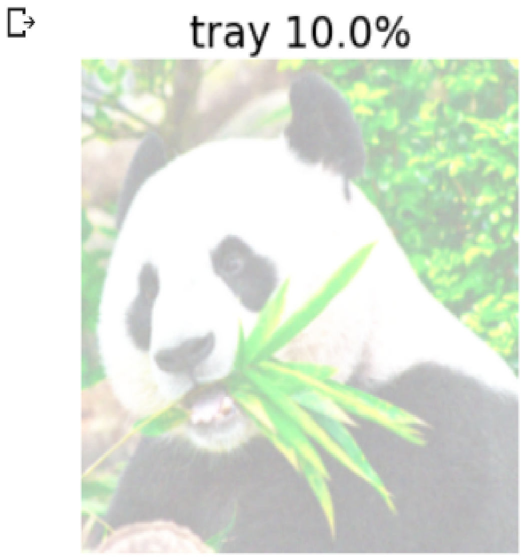


Figure 5:

Correlated Gaussian noise

```
figure(figsize=(12, 10), dpi=80)

plt.subplot(331)
plt.imshow(img)
plt.title('Image')
plt.axis('off')


# Generate correlated Gaussian noise
mean = [40, 1000, 90]
cov = [[10, 0, 0], [0, 3000, 0],[0, 0, 4000]]
cov_gauss = np.random.multivariate_normal(mean, cov,8156736)

cgauss_noise = cov_gauss.reshape(2856,2856,3).astype('uint8')
plt.subplot(332)
plt.imshow(cgauss_noise)
plt.title('Correlated Gaussian Noise')
plt.axis('off')

# Add the correlated Gaussian noise to the image
noisy_image3 = cv2.add(img,cgauss_noise)
plt.subplot(333)
plt.imshow(noisy_image3)
plt.title('Attacked Img')
plt.axis('off')
```

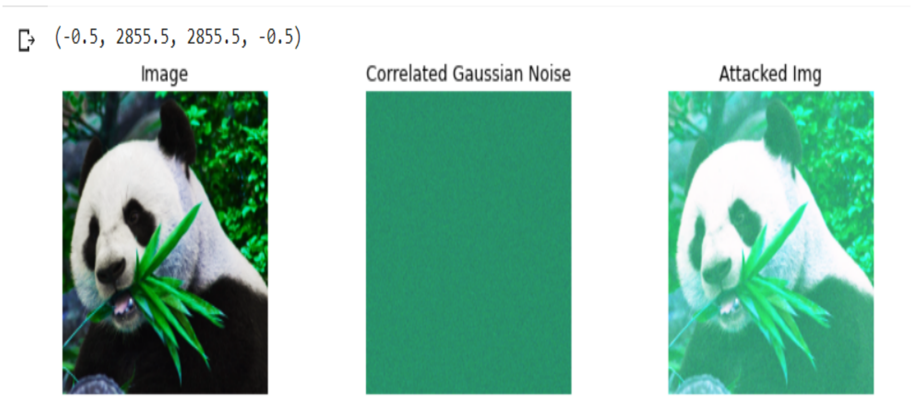

```
(-0.5, 2855.5, 2855.5, -0.5)
```

Figure 6:

Now Classifying the Adversarial Image, we get the below output.

```
color_coverted = cv2.cvtColor(noisy_image3, cv2.COLOR_BGR2RGB)
adv_image = Image.fromarray(color_coverted)

input_tensor = preprocess(adv_image)
input_batch3 = input_tensor.unsqueeze(0).numpy().astype(np.float32)
adv_input = input_batch3[0].transpose((1,2,0))

label = decode_predictions(classifier.predict(input_batch3))
label = label[0][0]

preds = classifier.predict(input_batch3)

accuracy = np.max(softmax_activation(preds), axis=1)
accuracy = round(accuracy[0], 2)

plt.imshow(adv_input)
plt.title(label[1] + " " + format(accuracy * 100) + "%", fontsize=20)
plt.axis('off')
plt.show()
```

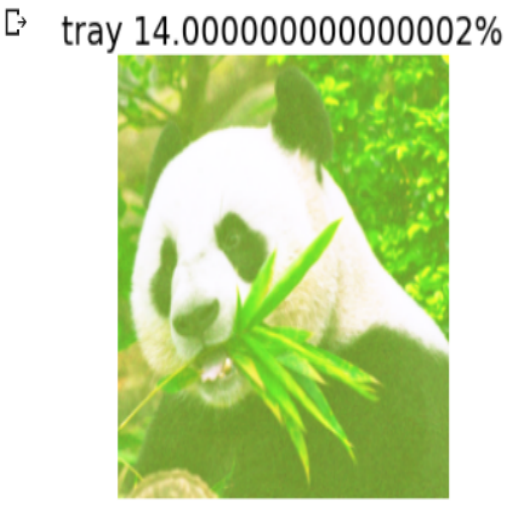


Figure 7:

Observation

For same dimension of same image (say, our first image of Panda) the all three attack have different probability to misclassify. Uniform noise has the least probability while Gaussian noise has the highest probability to misclassify the image and the correlated gaussian noise has average probability.
Here, we have observed that for different dimensions of images accuracy/probability is also changing. When we checked it with lower dimension image (such as eagle) and the output is given below.

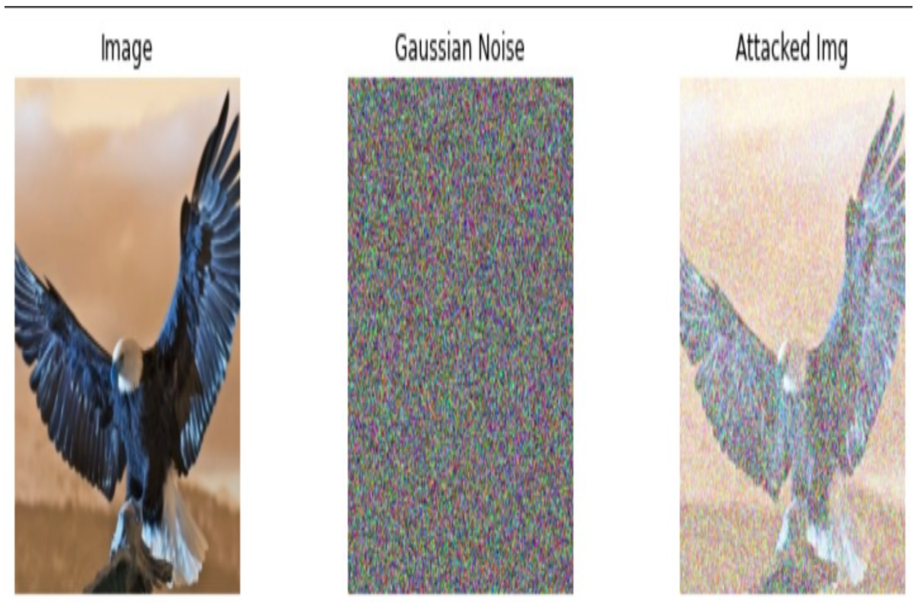


Figure 8:

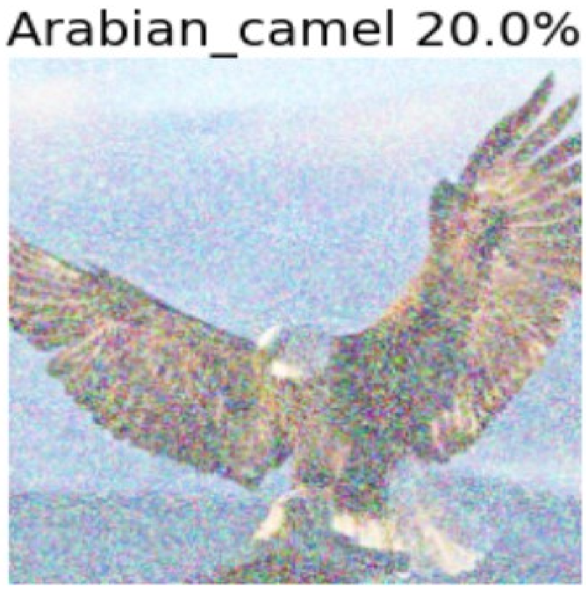


Figure 9:

Hence, We can conclude that for higher dimensions the probability is decreasing to perturbe the image.

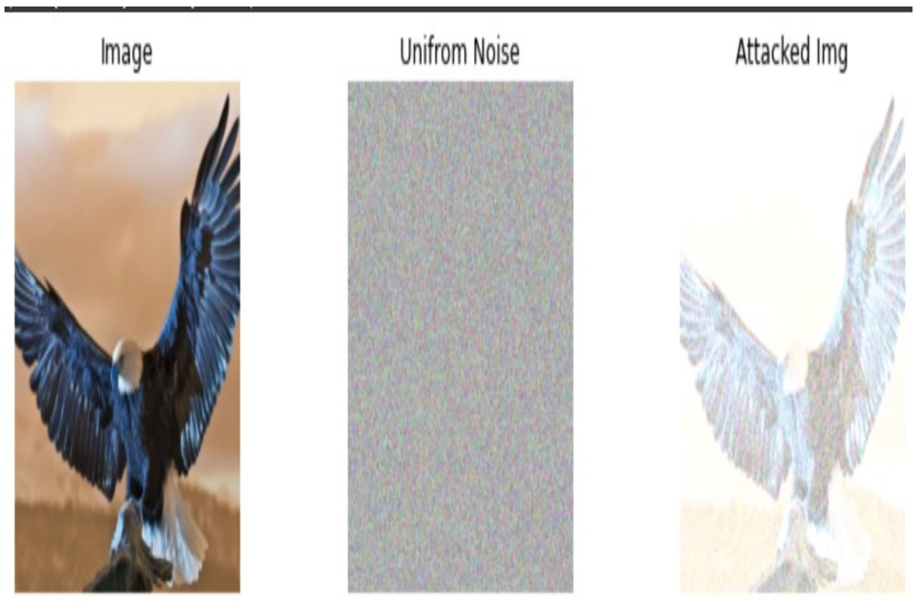


Figure 10:

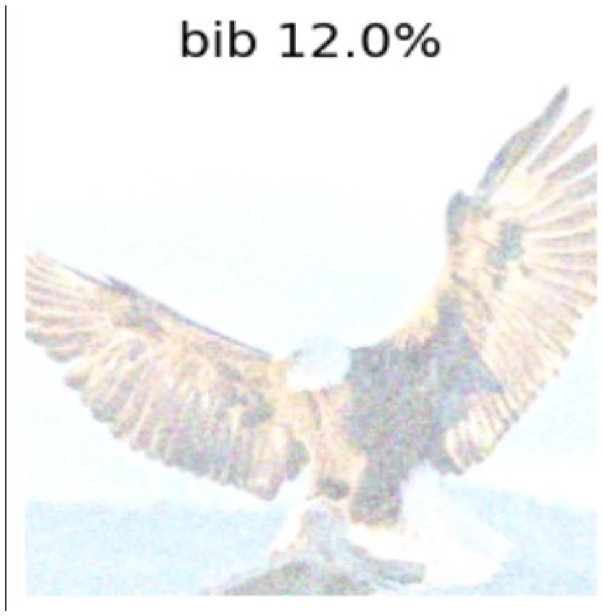


Figure 11:

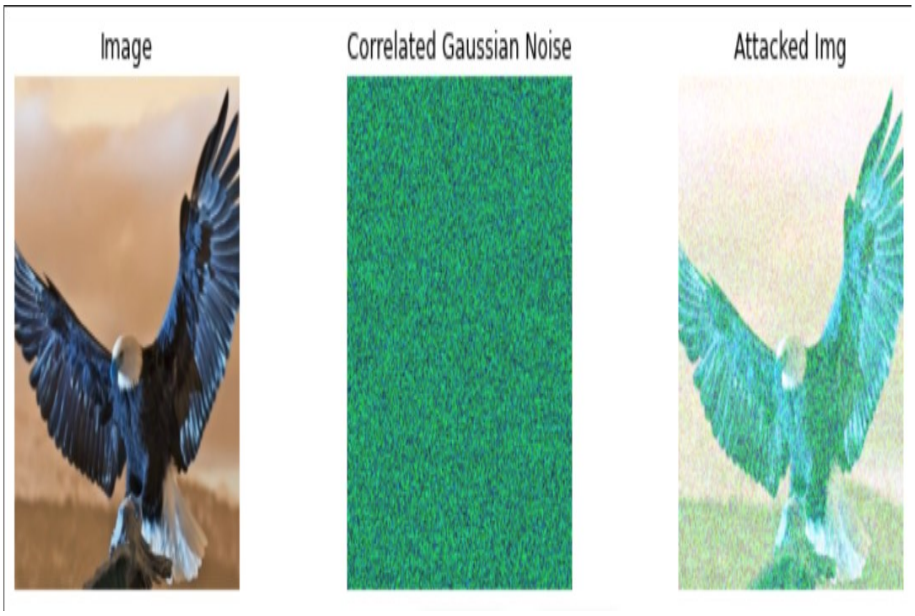


Figure 12:

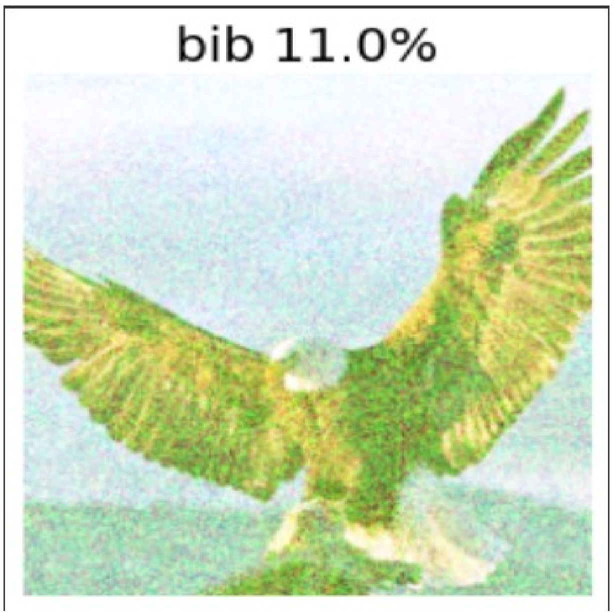


Figure 13: