White-box attack

May 26, 2025

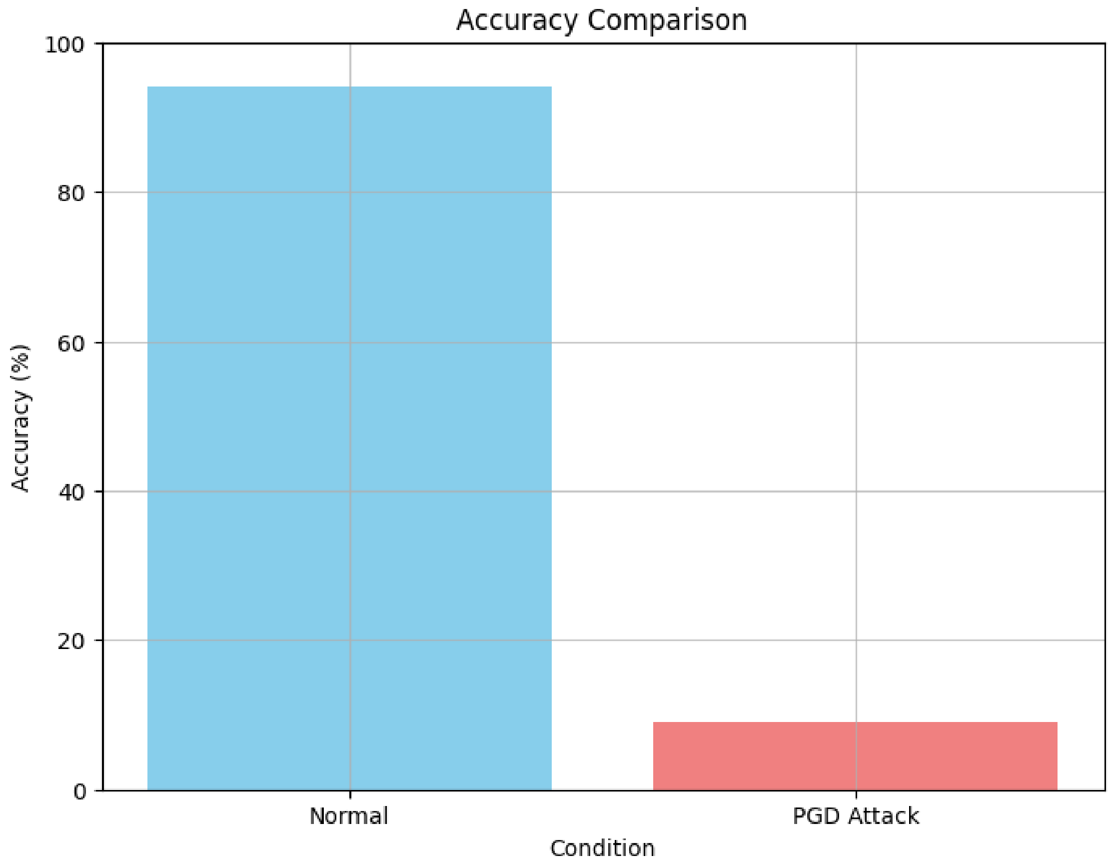
[6]:
```
#
!pip install pytorchcv matplotlib

import torch
import torch.nn as nn
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from pytorchcv.model_provider import get_model as ptcv_get_model
import matplotlib.pyplot as plt

#       GPU   CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#    ResNet-20    CIFAR-10
model = ptcv_get_model("resnet20_cifar10", pretrained=True)
model = model.to(device)  #
model.eval()  #       Dropout

#  CIFAR-10
transform = transforms.Compose([
    transforms.ToTensor(),  #    Tensor   [0, 1]
    transforms.Normalize(mean=[0.4914, 0.4822, 0.4465],  #
                         std=[0.2023, 0.1994, 0.2010])
])
#    CIFAR-10
testset = datasets.CIFAR10(root='./data', train=False, download=True,␣
 ↪transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=100,␣
 ↪shuffle=False, num_workers=2)

#  PGD
def pgd_attack(model, images, labels, eps=8/255, alpha=2/255, iters=10):
    images = images.clone().detach().to(device)  #
    labels = labels.to(device)
    original_images = images.clone().detach()  #
    loss_fn = nn.CrossEntropyLoss()  #
```

```
    for i in range(iters):  #
        images.requires_grad = True  #
        outputs = model(images)      #
        loss = loss_fn(outputs, labels)  #
        model.zero_grad()            #
        loss.backward()              #
        #
        adv_images = images + alpha * images.grad.sign()
        #       L∞
        eta = torch.clamp(adv_images - original_images, min=-eps, max=eps)
        #     [0, 1]
        images = torch.clamp(original_images + eta, min=0, max=1).detach()

    return images

#
model.eval()  #
correct_normal = 0
total_normal = 0
with torch.no_grad():  #
    for images, labels in testloader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = outputs.max(1)  #
        total_normal += labels.size(0)
        correct_normal += (predicted == labels).sum().item()  #

accuracy_normal = 100 * correct_normal / total_normal  #

#  PGD
correct_adv = 0
total_adv = 0
for images, labels in testloader:
    adv_images = pgd_attack(model, images, labels)  #
    with torch.no_grad():
        outputs = model(adv_images)
        _, predicted = outputs.max(1)
        total_adv += labels.size(0)
        correct_adv += (predicted.cpu() == labels).sum().item()  #

accuracy_adv = 100 * correct_adv / total_adv  #

#
labels_acc = ['Normal', 'PGD Attack']
accuracies = [accuracy_normal, accuracy_adv]
plt.figure(figsize=(8, 6))
```

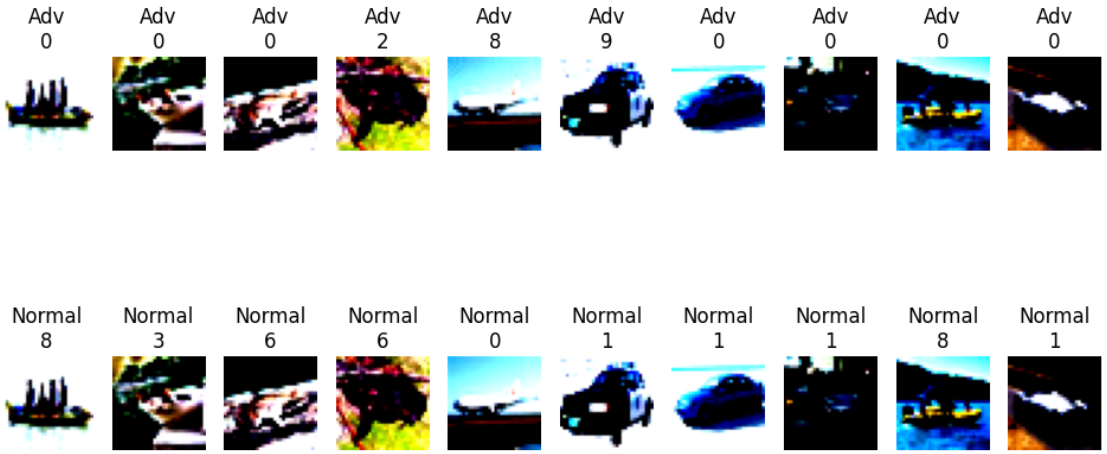
```
plt.bar(labels_acc, accuracies, color=['skyblue', 'lightcoral'])
plt.xlabel('Condition')  # X
plt.ylabel('Accuracy (%)')  # Y
plt.title('Accuracy Comparison')  #
plt.ylim(0, 100)
plt.grid(True, alpha=0.7)
plt.show()

#
plt.figure(figsize=(12, 6))
for i in range(10):  #   10
    ax = plt.subplot(2, 10, i + 1)
    plt.imshow(adv_images[i].cpu().permute(1, 2, 0))  #
    plt.title(f"Adv\n{predicted[i].item()}")
    plt.axis('off')

    ax = plt.subplot(2, 10, i + 11)
    plt.imshow(images[i].cpu().permute(1, 2, 0))  #
    plt.title(f"Normal\n{labels[i].item()}")
    plt.axis('off')

plt.show()

#
print(f"   : {accuracy_normal:.2f}%")
print(f"PGD    : {accuracy_adv:.2f}%")
```

```
/bin/sh: 1: pip: not found
Files already downloaded and verified
```


Adv
0
0
20
0
25




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```


Adv 0
Adv 0
Adv 0
Adv 2
Adv 8
Adv 9
Adv 0
Adv 0
Adv 0
Adv 0
Normal 8
Normal 3
Normal 6
Normal 6
Normal 0
Normal 1
Normal 1
Normal 1
Normal 8
Normal 1

: 94.02%

PGD : 9.09%

[ ]:

[ ]:

[ ]: