Black-box attack

May 26, 2025

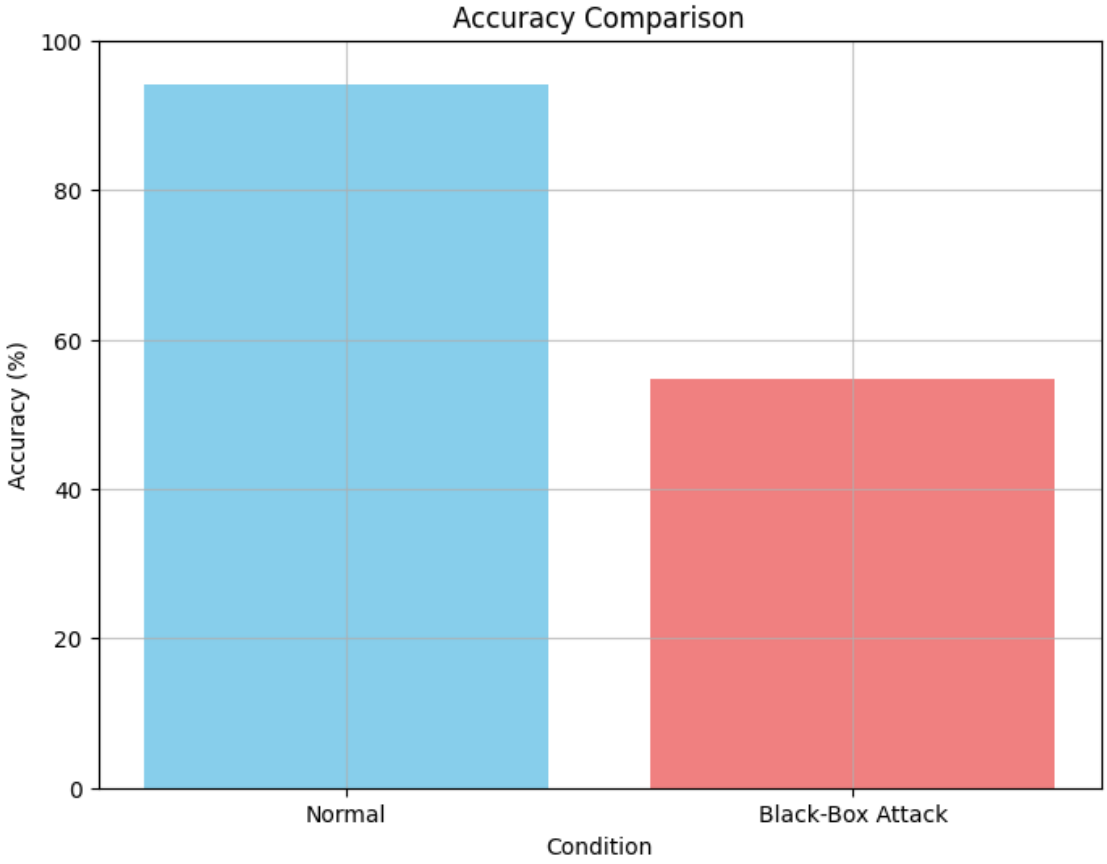
[3]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from pytorchcv.model_provider import get_model as ptcv_get_model
import matplotlib.pyplot as plt

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

#    ResNet-20
target_model = ptcv_get_model("resnet20_cifar10", pretrained=True)
target_model = target_model.to(device)
target_model.eval()  #

#    LeNet
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)  #  3   6    5x5
        self.pool = nn.MaxPool2d(2, 2)   #       2x2  2
        self.conv2 = nn.Conv2d(6, 16, 5) #     6   16
        self.fc1 = nn.Linear(16*5*5, 120)#     16*5*5  120
        self.fc2 = nn.Linear(120, 84)    #     120  84
        self.fc3 = nn.Linear(84, 10)     #    84  10 CIFAR-10

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  #    +ReLU+
        x = self.pool(F.relu(self.conv2(x)))  #    +ReLU+
        x = x.view(-1, 16*5*5)                #
        x = F.relu(self.fc1(x))               #   +ReLU
        x = F.relu(self.fc2(x))               #   +ReLU
        x = self.fc3(x)                       #
        return x
```

```
#
surrogate = LeNet().to(device)
optimizer = optim.Adam(surrogate.parameters(), lr=0.001)  #  Adam
loss_fn = nn.CrossEntropyLoss()  #

#  CIFAR-10
transform = transforms.Compose([
    transforms.ToTensor(),  #    Tensor
    transforms.Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2023, 0.1994, 0.
 ↪2010])  #
])
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)

#
surrogate.train()
for epoch in range(10):  #  10 epoch
    for images, labels in testloader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()  #
        outputs = surrogate(images)  #
        loss = loss_fn(outputs, labels)  #
        loss.backward()  #
        optimizer.step()  #

#  FGSM
def fgsm_attack(model, images, labels, eps=8/255):
    images = images.clone().detach().to(device)
    labels = labels.to(device)
    images.requires_grad = True  #
    outputs = model(images)
    loss = loss_fn(outputs, labels)
    model.zero_grad()
    loss.backward()
    adv_images = images + eps * images.grad.sign()  #
    adv_images = torch.clamp(adv_images, 0, 1)  #     [0,1]
    return adv_images

#
correct_normal = 0
total_normal = 0
with torch.no_grad():
    for images, labels in testloader:
        images, labels = images.to(device), labels.to(device)
        outputs = target_model(images)
```

```
        _, predicted = outputs.max(1)
        total_normal += labels.size(0)
        correct_normal += (predicted == labels).sum().item()

accuracy_normal = 100 * correct_normal / total_normal

#
correct_bb = 0
total_bb = 0
target_model.eval()
surrogate.eval()
for images, labels in testloader:
    adv_images = fgsm_attack(surrogate, images, labels)  #
    with torch.no_grad():
        outputs = target_model(adv_images.to(device))  #
        _, predicted = outputs.max(1)
        total_bb += labels.size(0)
        correct_bb += (predicted.cpu() == labels).sum().item()

accuracy_bb = 100 * correct_bb / total_bb

#
labels_acc = ['Normal', 'Black-Box Attack']
accuracies = [accuracy_normal, accuracy_bb]
plt.figure(figsize=(8, 6))
plt.bar(labels_acc, accuracies, color=['skyblue', 'lightcoral'])
plt.xlabel('Condition')
plt.ylabel('Accuracy (%)')
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):
    #
    ax = plt.subplot(2, 10, i + 1)
    plt.imshow(adv_images[i].cpu().detach().permute(1, 2, 0).numpy())
    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).numpy())
    plt.title(f"Normal\n{labels[i].item()}")
    plt.axis('off')
```

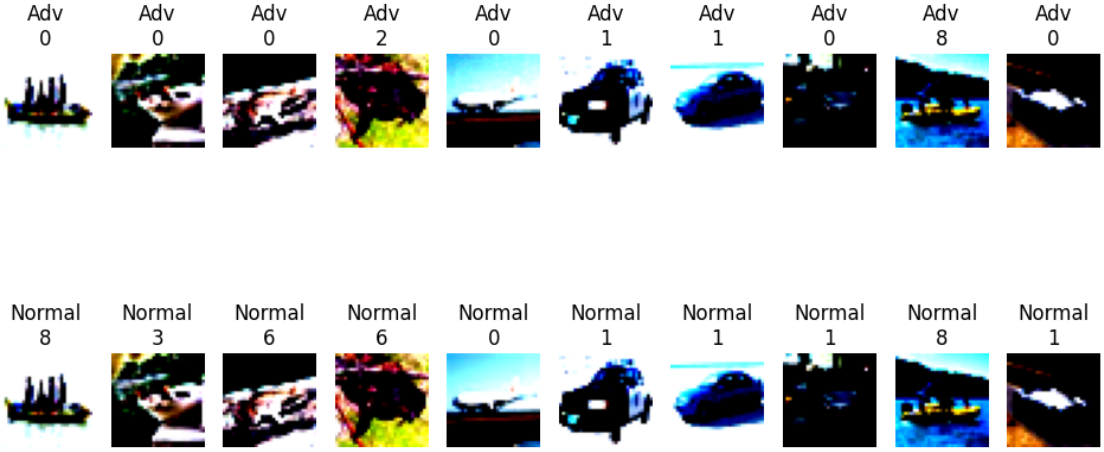
```
plt.show()

#
print(f"   : {accuracy_normal:.2f}%")
print(f"     : {accuracy_bb:.2f}%")
print(f"       : {100 * (1 - correct_bb / total_bb):.2f}%")
```

```
Files already downloaded and verified
```




```
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).
```




```
: 94.02%
  : 54.74%
     : 45.26%
```

[ ]: