1. 环境设置，和相关数据的下载

|  |
| --- |
| 1. # set up environment 2. !pip install pytorchcv 3. !pip install imgaug 4. # download 5. !wget https://github.com/DanielLin94144/ML-attack-dataset/files/8167812/data.zip 6. # unzip 7. !unzip ./data.zip 8. !rm ./data.zip 9. import torch 10. import torch.nn as nn 11. device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu') 12. batch\_size = 8 |

2.全局设置

|  |
| --- |
| # the mean and std are the calculated statistics from cifar\_10 dataset  cifar\_10\_mean = (0.491, 0.482, 0.447) # mean for the three channels of cifar\_10 images  cifar\_10\_std = (0.202, 0.199, 0.201) # std for the three channels of cifar\_10 images  # convert mean and std to 3-dimensional tensors for future operations  mean = torch.tensor(cifar\_10\_mean).to(device).view(3, 1, 1)  std = torch.tensor(cifar\_10\_std).to(device).view(3, 1, 1)  epsilon = 8/255/std  root = './data' # directory for storing benign images  # benign images: images which do not contain adversarial perturbations  # adversarial images: images which include adversarial perturbations |

3.数据的下载

|  |
| --- |
| import os  import glob  import shutil  import numpy as np  from PIL import Image  from torchvision.transforms import transforms  from torch.utils.data import Dataset, DataLoader  transform = transforms.Compose([      transforms.ToTensor(),      transforms.Normalize(cifar\_10\_mean, cifar\_10\_std)  ])  class AdvDataset(Dataset):      def \_\_init\_\_(self, data\_dir, transform):          self.images = []          self.labels = []          self.names = []          '''          data\_dir          ├── class\_dir          │   ├── class1.png          │   ├── ...          │   ├── class20.png          '''          for i, class\_dir in enumerate(sorted(glob.glob(f'{data\_dir}/\*'))):              images = sorted(glob.glob(f'{class\_dir}/\*'))              self.images += images              self.labels += ([i] \* len(images))              self.names += [os.path.relpath(imgs, data\_dir) for imgs in images]          self.transform = transform      def \_\_getitem\_\_(self, idx):          image = self.transform(Image.open(self.images[idx]))          label = self.labels[idx]          return image, label      def \_\_getname\_\_(self):          return self.names      def \_\_len\_\_(self):          return len(self.images)  adv\_set = AdvDataset(root, transform=transform)  adv\_names = adv\_set.\_\_getname\_\_()  adv\_loader = DataLoader(adv\_set, batch\_size=batch\_size, shuffle=False)  print(f'number of images = {adv\_set.\_\_len\_\_()}') |

4.原本图像分类设置函数

|  |
| --- |
| # to evaluate the performance of model on benign images  def epoch\_benign(model, loader, loss\_fn):      model.eval()      train\_acc, train\_loss = 0.0, 0.0      for x, y in loader:          x, y = x.to(device), y.to(device)          yp = model(x)          loss = loss\_fn(yp, y)          train\_acc += (yp.argmax(dim=1) == y).sum().item()          train\_loss += loss.item() \* x.shape[0]      return train\_acc / len(loader.dataset), train\_loss / len(loader.dataset) |

5.攻击函数的定义

|  |
| --- |
| #定义FGSM一次就达到目的的攻击方式的对应的函数  # perform fgsm attack  def fgsm(model, x, y, loss\_fn, epsilon=epsilon):      x\_adv = x.detach().clone() # initialize x\_adv as original benign image x      x\_adv.requires\_grad = True # need to obtain gradient of x\_adv, thus set required grad      loss = loss\_fn(model(x\_adv), y) # calculate loss      loss.backward() # calculate gradient      # fgsm: use gradient ascent on x\_adv to maximize loss      grad = x\_adv.grad.detach()      x\_adv = x\_adv + epsilon \* grad.sign() #直接一锤定音，grad.sign就是正取1，负取-1，一拳超人      return x\_adv    #总共有fgsm ， ifgsm ，mifgsm 这3种model，最开始只要理解fgsm的工作方式就好了  # alpha and num\_iter can be decided by yourself ，  alpha = 0.8/255/std #其实我觉得，按照李宏毅老师的讲法，这个alpha应该是8/255/std也就是和epsilon的大小一致才对  #哦！原来这里是ifgsm,所以会有20个iter，所以alpha的数值 只取到 epsilon的1/10  def ifgsm(model, x, y, loss\_fn, epsilon=epsilon, alpha=alpha, num\_iter=20): #参数是model，x图像，y标签，总共20个iter      x\_adv = x      # write a loop of num\_iter to represent the iterative times      for i in range(num\_iter):          # x\_adv = fgsm(model, x\_adv, y, loss\_fn, alpha) # call fgsm with (epsilon = alpha) to obtain new x\_adv          x\_adv = x\_adv.detach().clone()          x\_adv.requires\_grad = True # need to obtain gradient of x\_adv, thus set required grad          loss = loss\_fn(model(x\_adv), y) # calculate loss          loss.backward() # calculate gradient          # fgsm: use gradient ascent on x\_adv to maximize loss          grad = x\_adv.grad.detach()          x\_adv = x\_adv + alpha \* grad.sign() #得到攻击之后的图像x\_adv',计算方法和李宏毅老师讲到的一致            x\_adv = torch.max(torch.min(x\_adv, x+epsilon), x-epsilon) # clip new x\_adv back to [x-epsilon, x+epsilon]      return x\_adv      import torch  import torch.nn.functional as F  # 定义PGD攻击函数 ，您将输入图像限制在0到1之间的范围内  def pgd(model, x, y,loss\_fn ,epsilon=epsilon, alpha = alpha, num\_iter =20): #参数：model，x图像，y标签      x\_adv = x      # 循环迭代进行PGD攻击      for i in range(num\_iter):          x\_adv = x\_adv.detach().clone()  #相当于创建一个和原始对象无关的样本          x\_adv.requires\_grad = True # need to obtain gradient of x\_adv, thus set required grad          loss = loss\_fn(model(x\_adv), y) # calculate loss          loss.backward() # calculate gradient          model.zero\_grad() #清空grad,            # 根据梯度计算扰动并更新图像          with torch.no\_grad():              grad = x\_adv.grad.detach()              x\_adv = x\_adv + alpha \* grad.sign() #x\_adv就是image,original\_image就是x                x\_adv = torch.clamp(x\_adv,0,1) #利用torch.clamp函数 将每个像素的值控制在0-1之内，因为上面那个"+"可能超界                #计算diff，并且让x\_adv 就是x 和 diff之和:              diff = x\_adv - x              diff = torch.clamp(diff,-epsilon,epsilon) #将diff控制在-epsilon 和 +epsilon之间              x\_adv = x + diff            #产生最终的x\_adv              #这里和那个github有一些区别就是，最后的x\_adv是否需要clamp回到0-1之间呢                # 对新生成的图像重新计算梯度              x\_adv.requires\_grad = True        return x\_adv        # 执行PGD攻击生成对抗样本  #adversarial\_image = pgd\_attack(model, image, label, epsilon, alpha, num\_steps)    #让我再看看这个mifgsm到底是个啥东西呢，也就是要额外加上一个momentum的处理 和 decay是吧  #其实，这个mifgsm就是再ifgsm的基础上，类似于原来gradient descend可能遇到的问题一样，都要借助前一次的momentum动量进行处理  def mifgsm(model, x, y, loss\_fn, epsilon=epsilon, alpha=alpha, num\_iter=20, decay=1.0):      x\_adv = x      # initialze momentum tensor      momentum = torch.zeros\_like(x).detach().to(device) #一开始设定的momentum的数值就是和x的大小完全一样的全0      # write a loop of num\_iter to represent the iterative times      for i in range(num\_iter):          x\_adv = x\_adv.detach().clone()          x\_adv.requires\_grad = True # need to obtain gradient of x\_adv, thus set required grad          loss = loss\_fn(model(x\_adv), y) # calculate loss          loss.backward() # calculate gradient          # TODO: Momentum calculation          # grad = .....          #。。。有待进一步添加计算momentum的代码，应该会用到参数里面的decay这个参数          x\_adv = x\_adv + alpha \* grad.sign()          x\_adv = torch.max(torch.min(x\_adv, x+epsilon), x-epsilon) # clip new x\_adv back to [x-epsilon, x+epsilon]      return x\_adv |

6.生成adverserial图像

|  |
| --- |
| # perform adversarial attack and generate adversarial examples  #调用上述定义的 adv\_attack的方法，并且生成对应的 图片的攻击后 图像  def gen\_adv\_examples(model, loader, attack, loss\_fn):      model.eval()      adv\_names = []      train\_acc, train\_loss = 0.0, 0.0      for i, (x, y) in enumerate(loader):          x, y = x.to(device), y.to(device)          x\_adv = attack(model, x, y, loss\_fn) # obtain adversarial examples ，获取到攻击后的图像          yp = model(x\_adv)       #攻击后的图像的预测结果yp          loss = loss\_fn(yp, y)    #和原来label之间的loss          train\_acc += (yp.argmax(dim=1) == y).sum().item()          train\_loss += loss.item() \* x.shape[0]          #上面已经用model生成了我们需要的x\_adv的攻击后图像，下面只是对这张图像进行 反向还原罢了          # store adversarial examples          adv\_ex = ((x\_adv) \* std + mean).clamp(0, 1) # to 0-1 scale          adv\_ex = (adv\_ex \* 255).clamp(0, 255) # 0-255 scale          adv\_ex = adv\_ex.detach().cpu().data.numpy().round() # round to remove decimal part，利用round函数取出小数点          adv\_ex = adv\_ex.transpose((0, 2, 3, 1)) # transpose (bs, C, H, W) back to (bs, H, W, C)          adv\_examples = adv\_ex if i == 0 else np.r\_[adv\_examples, adv\_ex]      return adv\_examples, train\_acc / len(loader.dataset), train\_loss / len(loader.dataset)    #将adv攻击之后的图片 和 对应的攻击需要 绑定的label存储到对应的文件路径种  # create directory which stores adversarial examples  def create\_dir(data\_dir, adv\_dir, adv\_examples, adv\_names):      if os.path.exists(adv\_dir) is not True:          \_ = shutil.copytree(data\_dir, adv\_dir)      for example, name in zip(adv\_examples, adv\_names):          im = Image.fromarray(example.astype(np.uint8)) # image pixel value should be unsigned int          im.save(os.path.join(adv\_dir, name)) |

7.利用pretrain好的分类model和loss\_fn

|  |
| --- |
| from pytorchcv.model\_provider import get\_model as ptcv\_get\_model  model = ptcv\_get\_model('resnet110\_cifar10', pretrained=True).to(device)  loss\_fn = nn.CrossEntropyLoss()  benign\_acc, benign\_loss = epoch\_benign(model, adv\_loader, loss\_fn)  print(f'benign\_acc = {benign\_acc:.5f}, benign\_loss = {benign\_loss:.5f}') |

8.生成攻击图像，并进行存储

|  |
| --- |
| #调用fgsm生成adv\_图片，并且存储到对应的文件中  adv\_examples, fgsm\_acc, fgsm\_loss = gen\_adv\_examples(model, adv\_loader, fgsm, loss\_fn)  print(f'fgsm\_acc = {fgsm\_acc:.5f}, fgsm\_loss = {fgsm\_loss:.5f}')    create\_dir(root, 'fgsm', adv\_examples, adv\_names) |

|  |
| --- |
| #调用pdg生成adv\_图片，并且存储到对应的文件中  adv\_examples, pdg\_acc, pdg\_loss = gen\_adv\_examples(model, adv\_loader, pgd, loss\_fn)  print(f'pgd\_acc = {fgsm\_acc:.5f}, pgd\_loss = {fgsm\_loss:.5f}')    create\_dir(root, 'pgd', adv\_examples, adv\_names)   #adv\_names在data部分就已经确定了，谢谢 |

|  |
| --- |
| #调用ifgsm生成adv\_图片，并且存储到对应的文件中  adv\_examples, ifgsm\_acc, ifgsm\_loss = gen\_adv\_examples(model, adv\_loader, ifgsm, loss\_fn)  print(f'ifgsm\_acc = {ifgsm\_acc:.5f}, ifgsm\_loss = {ifgsm\_loss:.5f}')    create\_dir(root, 'ifgsm', adv\_examples, adv\_names) |

9.可视化的训练结果

|  |
| --- |
| #这里做的事情，就是输出利用这个训练好的model，分别输入攻击前 和 攻击后的图片， 然后得到的各自的pred的概率  import matplotlib.pyplot as plt    classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']    plt.figure(figsize=(10, 20))  cnt = 0  for i, cls\_name in enumerate(classes): #10个种类，每个种类展示1.png那个图片攻击前后的识别结果      path = f'{cls\_name}/{cls\_name}1.png'        # benign image      cnt += 1      plt.subplot(len(classes), 4, cnt)      im = Image.open(f'./data/{path}')   #这一段 和 下面一段的唯一的区别 只是打开的图片的文件不同罢了      logit = model(transform(im).unsqueeze(0).to(device))[0]      predict = logit.argmax(-1).item()      prob = logit.softmax(-1)[predict].item()      plt.title(f'benign: {cls\_name}1.png\n{classes[predict]}: {prob:.2%}')      plt.axis('off')      plt.imshow(np.array(im))        # adversarial image      cnt += 1      plt.subplot(len(classes), 4, cnt)      im = Image.open(f'./ifgsm/{path}')      logit = model(transform(im).unsqueeze(0).to(device))[0]      predict = logit.argmax(-1).item()      prob = logit.softmax(-1)[predict].item()      plt.title(f'adversarial: {cls\_name}1.png\n{classes[predict]}: {prob:.2%}')      plt.axis('off')      plt.imshow(np.array(im))    plt.tight\_layout()  plt.show() |

Gpd

|  |
| --- |
| #这里做的事情，就是输出利用这个训练好的model，分别输入攻击前 和 攻击后的图片， 然后得到的各自的pred的概率  import matplotlib.pyplot as plt    classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']    plt.figure(figsize=(10, 20))  cnt = 0  for i, cls\_name in enumerate(classes): #10个种类，每个种类展示1.png那个图片攻击前后的识别结果      path = f'{cls\_name}/{cls\_name}1.png'        # benign image      cnt += 1      plt.subplot(len(classes), 4, cnt)      im = Image.open(f'./data/{path}')   #这一段 和 下面一段的唯一的区别 只是打开的图片的文件不同罢了      logit = model(transform(im).unsqueeze(0).to(device))[0]      predict = logit.argmax(-1).item()      prob = logit.softmax(-1)[predict].item()      plt.title(f'benign: {cls\_name}1.png\n{classes[predict]}: {prob:.2%}')      plt.axis('off')      plt.imshow(np.array(im))        # adversarial image      cnt += 1      plt.subplot(len(classes), 4, cnt)      im = Image.open(f'./pgd/{path}')      logit = model(transform(im).unsqueeze(0).to(device))[0]      predict = logit.argmax(-1).item()      prob = logit.softmax(-1)[predict].item()      plt.title(f'adversarial: {cls\_name}1.png\n{classes[predict]}: {prob:.2%}')      plt.axis('off')      plt.imshow(np.array(im))    plt.tight\_layout()  plt.show() |

10.测试结果：

|  |
| --- |
| #dog版本  # original image  path = f'dog/dog2.png'  im = Image.open(f'./data/{path}')  print(im.size)  print(im.mode)  logit = model(transform(im).unsqueeze(0).to(device))[0]  predict = logit.argmax(-1).item()  prob = logit.softmax(-1)[predict].item()  plt.title(f'benign: dog2.png\n{classes[predict]}: {prob:.2%}')  plt.axis('off')  plt.imshow(np.array(im))  plt.tight\_layout()  plt.show()  # adversarial image  adv\_im = Image.open(f'./fgsm/{path}')  logit = model(transform(adv\_im).unsqueeze(0).to(device))[0]  predict = logit.argmax(-1).item()  prob = logit.softmax(-1)[predict].item()  plt.title(f'adversarial: dog2.png\n{classes[predict]}: {prob:.2%}')  plt.axis('off')  plt.imshow(np.array(adv\_im))  plt.tight\_layout()  plt.show() |

|  |
| --- |
| #cat版本  # original image  path = f'cat/cat2.png'  im = Image.open(f'./data/{path}')  print(im.size)  print(im.mode)  logit = model(transform(im).unsqueeze(0).to(device))[0]  predict = logit.argmax(-1).item()  prob = logit.softmax(-1)[predict].item()  plt.title(f'benign: cat2.png\n{classes[predict]}: {prob:.2%}')  plt.axis('off')  plt.imshow(np.array(im))  plt.tight\_layout()  plt.show()  # adversarial image  adv\_im = Image.open(f'./ifgsm/{path}')  logit = model(transform(adv\_im).unsqueeze(0).to(device))[0]  predict = logit.argmax(-1).item()  prob = logit.softmax(-1)[predict].item()  plt.title(f'adversarial: cat2.png\n{classes[predict]}: {prob:.2%}')  plt.axis('off')  plt.imshow(np.array(adv\_im))  plt.tight\_layout()  plt.show() |

|  |
| --- |
| #airplane版本  # original image  path = f'airplane/airplane2.png'  im = Image.open(f'./data/{path}')  print(im.size)  print(im.mode)  logit = model(transform(im).unsqueeze(0).to(device))[0]  predict = logit.argmax(-1).item()  prob = logit.softmax(-1)[predict].item()  plt.title(f'benign: airplane2.png\n{classes[predict]}: {prob:.2%}')  plt.axis('off')  plt.imshow(np.array(im))  plt.tight\_layout()  plt.show()  # adversarial image  adv\_im = Image.open(f'./pgd/{path}')  logit = model(transform(adv\_im).unsqueeze(0).to(device))[0]  predict = logit.argmax(-1).item()  prob = logit.softmax(-1)[predict].item()  plt.title(f'adversarial: airplane2.png\n{classes[predict]}: {prob:.2%}')  plt.axis('off')  plt.imshow(np.array(adv\_im))  plt.tight\_layout()  plt.show() |