```
import torch
import torchvision.datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

import sys
sys.path.append("/content/drive/My Drive/Colab Notebooks/AdversarialExample/advGAN")
import ipynb_importer

import models
from models import MNIST_target_net


use_cuda=True
image_nc=1
batch_size = 128

gen_input_nc = image_nc


# Define what device we are using
print("CUDA Available: ",torch.cuda.is_available())
device = torch.device("cuda" if (use_cuda and torch.cuda.is_available()) else "cpu")
```

```
CUDA Available:  True
```

```
# load the pretrained model
pretrained_model = "/content/drive/My Drive/Colab Notebooks/AdversarialExample/advGAN/MNIST_target_model.pth"
target_model = MNIST_target_net().to(device)
target_model.load_state_dict(torch.load(pretrained_model))
target_model.eval()
```

```
MNIST_target_net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=1024, out_features=200, bias=True)
  (fc2): Linear(in_features=200, out_features=200, bias=True)
  (logits): Linear(in_features=200, out_features=10, bias=True)
)
```

```
# load the generator of adversarial examples
pretrained_generator_path = '/content/drive/My Drive/Colab Notebooks/AdversarialExample/advGAN/models/netG_epoch_60.pth'
pretrained_G = models.Generator(gen_input_nc, image_nc).to(device)
pretrained_G.load_state_dict(torch.load(pretrained_generator_path))
pretrained_G.eval()
```

```
Generator(
  (encoder): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1))
    (1): InstanceNorm2d(8, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (2): ReLU()
    (3): Conv2d(8, 16, kernel_size=(3, 3), stride=(2, 2))
    (4): InstanceNorm2d(16, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (5): ReLU()
    (6): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2))
    (7): InstanceNorm2d(32, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (8): ReLU()
  )
  (bottle_neck): Sequential(
    (0): ResnetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
        (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
        (6): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): ResnetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
        (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
        (6): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (2): ResnetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
        (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
```

```
            (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
            (6): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          )
        )
        (3): ResnetBlock(
          (conv_block): Sequential(
            (0): ReflectionPad2d((1, 1, 1, 1))
            (1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
            (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (3): ReLU(inplace=True)
            (4): ReflectionPad2d((1, 1, 1, 1))
            (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
            (6): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          )
        )
      )
      (decoder): Sequential(
        (0): ConvTranspose2d(32, 16, kernel_size=(3, 3), stride=(2, 2), bias=False)
        (1): InstanceNorm2d(16, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
        (2): ReLU()
        (3): ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(2, 2), bias=False)
        (4): InstanceNorm2d(8, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
        (5): ReLU()
        (6): ConvTranspose2d(8, 1, kernel_size=(6, 6), stride=(1, 1), bias=False)
        (7): Tanh()
      )
    )
```

```
# test adversarial examples in MNIST training dataset
mnist_dataset = torchvision.datasets.MNIST('./dataset', train=True, transform=transforms.ToTensor(), download=False)
train_dataloader = DataLoader(mnist_dataset, batch_size=batch_size, shuffle=False, num_workers=1)
num_correct = 0
for i, data in enumerate(train_dataloader, 0):
    test_img, test_label = data
    test_img, test_label = test_img.to(device), test_label.to(device)
    perturbation = pretrained_G(test_img)
    perturbation = torch.clamp(perturbation, -0.3, 0.3)
    adv_img = perturbation + test_img
    adv_img = torch.clamp(adv_img, 0, 1)
```

```
        pred_lab = torch.argmax(target_model(adv_img),1)
        num_correct += torch.sum(pred_lab==test_label,0)

    print('MNIST training dataset:')
    print('num_correct: ', num_correct.item())
    print('accuracy of adv imgs in training set: %f\n'%(num_correct.item()/len(mnist_dataset)))
```

```
MNIST training dataset:
num_correct:  155
accuracy of adv imgs in training set: 0.002583
```

```
    # test adversarial examples in MNIST testing dataset
    mnist_dataset_test = torchvision.datasets.MNIST('./dataset', train=False, transform=transforms.ToTensor(), download=False
    test_dataloader = DataLoader(mnist_dataset_test, batch_size=batch_size, shuffle=False, num_workers=1)
    num_correct = 0
    for i, data in enumerate(test_dataloader, 0):
        test_img, test_label = data
        test_img, test_label = test_img.to(device), test_label.to(device)
        perturbation = pretrained_G(test_img)
        perturbation = torch.clamp(perturbation, -0.3, 0.3)
        adv_img = perturbation + test_img
        adv_img = torch.clamp(adv_img, 0, 1)
        pred_lab = torch.argmax(target_model(adv_img),1)
        num_correct += torch.sum(pred_lab==test_label,0)

    print('num_correct: ', num_correct.item())
    print('accuracy of adv imgs in testing set: %f\n'%(num_correct.item()/len(mnist_dataset_test)))
```

```
num_correct:  54
accuracy of adv imgs in testing set: 0.005400
```