```python
import torch
import torchvision.transforms as transforms
import torch.nn as nn
import torchvision.datasets as dset
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
import torchvision.utils as vutils
import torch.optim as optim
import time
from tqdm import tqdm

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

```python
# Hyperparameters
image_size = 64
batch_size = 256
nz = 100  # Latent vector size
num_epochs = 20
lr = 0.0002
beta1 = 0.5  # Beta1 hyperparameter for Adam optimizer

# Define dataset transformation
transform = transforms.Compose([
    transforms.Resize((image_size, image_size)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

dataset = dset.ImageFolder(root="data\img_align_celeba",
transform=transform)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True,
num_workers=4)
```

```
<>:1: SyntaxWarning: invalid escape sequence '\i'
<>:1: SyntaxWarning: invalid escape sequence '\i'
C:\Users\Suyash Tambe\AppData\Local\Temp\
ipykernel_27800\2267843485.py:1: SyntaxWarning: invalid escape
sequence '\i'
  dataset = dset.ImageFolder(root="data\img_align_celeba",
transform=transform)
```

```python
# Define the Generator
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(nz, 512, 4, 1, 0, bias=False),
```

```python
            nn.BatchNorm2d(512),
            nn.ReLU(True),

            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(True),

            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),

            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),

            nn.ConvTranspose2d(64, 3, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, input):
        return self.main(input)

# Define the Discriminator
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(3, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(512, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)

# Initialize models
netG = Generator().to(device)
```

```
netD = Discriminator().to(device)

# Loss and optimizers
criterion = nn.BCELoss()
optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1,
0.999))
optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1,
0.999))

# Create fixed noise for image generation
fixed_noise = torch.randn(64, nz, 1, 1, device=device)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
netD.to(device)
netG.to(device)
criterion.to(device)

BCELoss()
```

Train DCGAN

```
real_label = 1.
fake_label = 0.

print("Starting Training...")
start_time = time.time()

G_losses = []
D_losses = []
img_list = []

Starting Training...

for epoch in range(num_epochs):
    for i, (real_images, _) in enumerate(dataloader):
        real_images = real_images.to(device)
        batch_size = real_images.size(0)

        # Update Discriminator: max log(D(x)) + log(1 - D(G(z)))
        netD.zero_grad()
        label = torch.full((batch_size,), real_label,
dtype=torch.float, device=device)

        output = netD(real_images).view(-1)
        errD_real = criterion(output, label)
        errD_real.backward()

        noise = torch.randn(batch_size, nz, 1, 1, device=device)
        fake_images = netG(noise)
        label.fill_(fake_label)
```

```python
        output = netD(fake_images.detach()).view(-1)
        errD_fake = criterion(output, label)
        errD_fake.backward()
        optimizerD.step()

        # Update Generator: min log(1 - D(G(z))) <=> max log(D(G(z)))
        netG.zero_grad()
        label.fill_(real_label)

        output = netD(fake_images).view(-1)
        errG = criterion(output, label)
        errG.backward()
        optimizerG.step()

        # Store losses for visualization
        G_losses.append(errG.item())
        D_losses.append(errD_real.item() + errD_fake.item())

        # Print progress
        if i % 100 == 0:
            print(f"Epoch [{epoch}/{num_epochs}] | Batch
[{i}/{len(dataloader)}] | D Loss: {errD_real.item() +
errD_fake.item():.4f} | G Loss: {errG.item():.4f}")

    # Save generated images every epoch
    with torch.no_grad():
        fake = netG(fixed_noise).detach().cpu()
    img_list.append(vutils.make_grid(fake, padding=2, normalize=True))

print("Training Complete Time Taken:", round((time.time() -
start_time) / 60, 2), "minutes")
```

```
Epoch [0/20] | Batch [0/215] | D Loss: 1.3756 | G Loss: 2.5450
Epoch [0/20] | Batch [100/215] | D Loss: 1.5769 | G Loss: 3.9990
Epoch [0/20] | Batch [200/215] | D Loss: 1.3657 | G Loss: 6.7274
Epoch [1/20] | Batch [0/215] | D Loss: 0.8003 | G Loss: 3.6828
Epoch [1/20] | Batch [100/215] | D Loss: 1.2486 | G Loss: 5.0917
Epoch [1/20] | Batch [200/215] | D Loss: 1.1288 | G Loss: 5.6316
Epoch [2/20] | Batch [0/215] | D Loss: 0.4727 | G Loss: 3.3179
Epoch [2/20] | Batch [100/215] | D Loss: 0.6919 | G Loss: 5.1794
Epoch [2/20] | Batch [200/215] | D Loss: 0.5908 | G Loss: 4.2242
Epoch [3/20] | Batch [0/215] | D Loss: 1.5105 | G Loss: 7.7666
Epoch [3/20] | Batch [100/215] | D Loss: 0.8457 | G Loss: 3.5241
Epoch [3/20] | Batch [200/215] | D Loss: 1.4321 | G Loss: 5.7661
Epoch [4/20] | Batch [0/215] | D Loss: 0.5884 | G Loss: 2.6612
Epoch [4/20] | Batch [100/215] | D Loss: 0.6020 | G Loss: 2.6435
Epoch [4/20] | Batch [200/215] | D Loss: 0.4881 | G Loss: 2.7254
Epoch [5/20] | Batch [0/215] | D Loss: 0.6012 | G Loss: 2.1209
Epoch [5/20] | Batch [100/215] | D Loss: 0.4776 | G Loss: 2.6640
Epoch [5/20] | Batch [200/215] | D Loss: 0.9638 | G Loss: 4.4273
```

```
Epoch [6/20] | Batch [0/215] | D Loss: 0.4093 | G Loss: 3.4601
Epoch [6/20] | Batch [100/215] | D Loss: 0.5808 | G Loss: 3.4717
Epoch [6/20] | Batch [200/215] | D Loss: 0.5578 | G Loss: 2.2784
Epoch [7/20] | Batch [0/215] | D Loss: 0.5431 | G Loss: 4.1268
Epoch [7/20] | Batch [100/215] | D Loss: 0.5233 | G Loss: 4.8694
Epoch [7/20] | Batch [200/215] | D Loss: 0.5443 | G Loss: 3.0236
Epoch [8/20] | Batch [0/215] | D Loss: 0.7652 | G Loss: 1.9277
Epoch [8/20] | Batch [100/215] | D Loss: 0.5713 | G Loss: 1.8814
Epoch [8/20] | Batch [200/215] | D Loss: 0.9439 | G Loss: 6.0227
Epoch [9/20] | Batch [0/215] | D Loss: 0.4279 | G Loss: 1.5815
Epoch [9/20] | Batch [100/215] | D Loss: 0.5365 | G Loss: 2.4530
Epoch [9/20] | Batch [200/215] | D Loss: 0.6560 | G Loss: 4.8223
Epoch [10/20] | Batch [0/215] | D Loss: 0.7223 | G Loss: 5.0424
Epoch [10/20] | Batch [100/215] | D Loss: 1.6646 | G Loss: 1.4956
Epoch [10/20] | Batch [200/215] | D Loss: 0.5047 | G Loss: 3.6161
Epoch [11/20] | Batch [0/215] | D Loss: 0.3108 | G Loss: 2.8082
Epoch [11/20] | Batch [100/215] | D Loss: 0.4099 | G Loss: 2.6913
Epoch [11/20] | Batch [200/215] | D Loss: 1.1880 | G Loss: 6.2122
Epoch [12/20] | Batch [0/215] | D Loss: 0.6358 | G Loss: 1.6787
Epoch [12/20] | Batch [100/215] | D Loss: 0.5521 | G Loss: 2.0994
Epoch [12/20] | Batch [200/215] | D Loss: 0.4645 | G Loss: 1.5683
Epoch [13/20] | Batch [0/215] | D Loss: 0.5354 | G Loss: 3.3806
Epoch [13/20] | Batch [100/215] | D Loss: 0.3122 | G Loss: 3.0757
Epoch [13/20] | Batch [200/215] | D Loss: 0.2518 | G Loss: 2.8160
Epoch [14/20] | Batch [0/215] | D Loss: 0.6501 | G Loss: 1.3759
Epoch [14/20] | Batch [100/215] | D Loss: 1.1764 | G Loss: 1.5394
Epoch [14/20] | Batch [200/215] | D Loss: 0.5564 | G Loss: 2.4358
Epoch [15/20] | Batch [0/215] | D Loss: 1.0553 | G Loss: 0.7434
Epoch [15/20] | Batch [100/215] | D Loss: 0.3139 | G Loss: 3.5183
Epoch [15/20] | Batch [200/215] | D Loss: 0.4293 | G Loss: 3.0397
Epoch [16/20] | Batch [0/215] | D Loss: 0.3790 | G Loss: 2.7854
Epoch [16/20] | Batch [100/215] | D Loss: 1.2360 | G Loss: 0.5778
Epoch [16/20] | Batch [200/215] | D Loss: 1.2909 | G Loss: 5.1489
Epoch [17/20] | Batch [0/215] | D Loss: 0.3831 | G Loss: 2.8593
Epoch [17/20] | Batch [100/215] | D Loss: 0.6994 | G Loss: 1.3661
Epoch [17/20] | Batch [200/215] | D Loss: 0.3509 | G Loss: 2.2374
Epoch [18/20] | Batch [0/215] | D Loss: 0.3072 | G Loss: 3.2009
Epoch [18/20] | Batch [100/215] | D Loss: 0.5192 | G Loss: 2.3658

# Display Generated Images
def show_generated_images():
    real_images, _ = next(iter(dataloader))
    real_images = real_images[:64]

    # Generate fake images
    with torch.no_grad():
        fake_images = netG(fixed_noise).detach().cpu()

    fig, axes = plt.subplots(2, 1, figsize=(8, 8))
```

```
    # Show real images
    axes[0].imshow(np.transpose(vutils.make_grid(real_images,
padding=2, normalize=True), (1, 2, 0)))
    axes[0].set_title("Real Images")
    axes[0].axis("off")

    # Show fake images
    axes[1].imshow(np.transpose(vutils.make_grid(fake_images,
padding=2, normalize=True), (1, 2, 0)))
    axes[1].set_title("Generated Images")
    axes[1].axis("off")

    plt.show()

# Save generated images
show_generated_images()

# Save models
torch.save(netG.state_dict(), "generator.pth")
torch.save(netD.state_dict(), "discriminator.pth")

# Load models
netG.load_state_dict(torch.load("generator.pth", map_location=device))
netD.load_state_dict(torch.load("discriminator.pth",
map_location=device))
```

https://github.com/suyashtambe/Gan-s