```
import os
import torch
import torch.nn as nn
import numpy as np
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torchvision.utils as vutils

# ----------------------------
# Configuration
# ----------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
latent_dim = 100
img_shape = (3, 64, 64)
batch_size = 64
n_epochs = 50
save_dir = "./images"
os.makedirs(save_dir, exist_ok=True)
```

```
# Data Preparation
# ----------------------------
transform = transforms.Compose([
    transforms.Resize(64),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3),
])

dataset_path = '/root/.cache/kagglehub/datasets/spandan2/cats-faces-64x64-for-generative-models/versions/1'
dataset = torchvision.datasets.ImageFolder(root=dataset_path, transform=transform)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=2)
```

```
class Generator(nn.Module):
    def __init__(self, latent_dim, img_shape):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(latent_dim, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, int(np.prod(img_shape))),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.model(z)
        return img.view(z.size(0), *img_shape)

class Discriminator(nn.Module):
    def __init__(self, img_shape):
        super().__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(int(np.prod(img_shape)), 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, img):
        return self.model(img)

generator = Generator(latent_dim, img_shape).to(device)
discriminator = Discriminator(img_shape).to(device)

adversarial_loss = nn.BCELoss().to(device)
optimizer_G = torch.optim.Adam(generator.parameters(), lr=0.0002, betas=(0.5, 0.999))
optimizer_D = torch.optim.Adam(discriminator.parameters(), lr=0.0002, betas=(0.5, 0.999))
```

```
for epoch in range(n_epochs):
    g_loss_epoch = 0
    d_loss_epoch = 0

    for i, (imgs, _) in enumerate(dataloader):
        batch_size_curr = imgs.size(0)
        real = torch.ones(batch_size_curr, 1, device=device)
        fake = torch.zeros(batch_size_curr, 1, device=device)

        real_imgs = imgs.to(device)

        # Train Discriminator
        optimizer_D.zero_grad()

        real_validity = discriminator(real_imgs)
        d_real_loss = adversarial_loss(real_validity, real)

        z = torch.randn(batch_size_curr, latent_dim, device=device)
        fake_imgs = generator(z).detach()
        fake_validity = discriminator(fake_imgs)
        d_fake_loss = adversarial_loss(fake_validity, fake)

        d_loss = (d_real_loss + d_fake_loss) / 2
        d_loss.backward()
        optimizer_D.step()

        # Train Generator
        optimizer_G.zero_grad()

        gen_imgs = generator(z)
        g_loss = adversarial_loss(discriminator(gen_imgs), real)

        g_loss.backward()
        optimizer_G.step()

        g_loss_epoch += g_loss.item()
        d_loss_epoch += d_loss.item()

    avg_g_loss = g_loss_epoch / len(dataloader)
    avg_d_loss = d_loss_epoch / len(dataloader)
    print(f"[Epoch {epoch+1}/{n_epochs}] D Loss: {avg_d_loss:.4f} | G Loss: {avg_g_loss:.4f}")
    if (epoch + 1) % 10 == 0:
        with torch.no_grad():
            sample_noise = torch.randn(25, latent_dim, device=device)
            sample_imgs = generator(sample_noise)
            sample_imgs = (sample_imgs + 1) / 2  # Rescale to [0,1]
            vutils.save_image(sample_imgs, f"{save_dir}/epoch_{epoch+1}.png", nrow=5)

print("Training complete.")
```

```
[Epoch 1/50] D Loss: 0.4389 | G Loss: 0.9386
[Epoch 2/50] D Loss: 0.5159 | G Loss: 1.2910
[Epoch 3/50] D Loss: 0.5345 | G Loss: 1.4680
[Epoch 4/50] D Loss: 0.5778 | G Loss: 1.4922
[Epoch 5/50] D Loss: 0.5662 | G Loss: 1.4247
[Epoch 6/50] D Loss: 0.5649 | G Loss: 1.3875
[Epoch 7/50] D Loss: 0.5769 | G Loss: 1.3628
[Epoch 8/50] D Loss: 0.5852 | G Loss: 1.3226
[Epoch 9/50] D Loss: 0.5870 | G Loss: 1.2571
[Epoch 10/50] D Loss: 0.5927 | G Loss: 1.2692
[Epoch 11/50] D Loss: 0.5843 | G Loss: 1.2544
[Epoch 12/50] D Loss: 0.5867 | G Loss: 1.2442
[Epoch 13/50] D Loss: 0.5887 | G Loss: 1.2342
[Epoch 14/50] D Loss: 0.5900 | G Loss: 1.2227
[Epoch 15/50] D Loss: 0.5939 | G Loss: 1.2044
[Epoch 16/50] D Loss: 0.5895 | G Loss: 1.2042
[Epoch 17/50] D Loss: 0.6042 | G Loss: 1.1724
[Epoch 18/50] D Loss: 0.6169 | G Loss: 1.1289
[Epoch 19/50] D Loss: 0.6186 | G Loss: 1.1149
[Epoch 20/50] D Loss: 0.6189 | G Loss: 1.1078
[Epoch 21/50] D Loss: 0.6284 | G Loss: 1.0831
[Epoch 22/50] D Loss: 0.6271 | G Loss: 1.0683
[Epoch 23/50] D Loss: 0.6273 | G Loss: 1.0581
[Epoch 24/50] D Loss: 0.6275 | G Loss: 1.0727
[Epoch 25/50] D Loss: 0.6295 | G Loss: 1.0681
[Epoch 26/50] D Loss: 0.6308 | G Loss: 1.0510
[Epoch 27/50] D Loss: 0.6319 | G Loss: 1.0558
[Epoch 28/50] D Loss: 0.6312 | G Loss: 1.0447
[Epoch 29/50] D Loss: 0.6282 | G Loss: 1.0632
[Epoch 30/50] D Loss: 0.6285 | G Loss: 1.0637
```

```
[Epoch 31/50] D Loss: 0.6243 | G Loss: 1.0726
[Epoch 32/50] D Loss: 0.6195 | G Loss: 1.0827
[Epoch 33/50] D Loss: 0.6158 | G Loss: 1.0999
[Epoch 34/50] D Loss: 0.6119 | G Loss: 1.1101
[Epoch 35/50] D Loss: 0.6062 | G Loss: 1.1305
[Epoch 36/50] D Loss: 0.6010 | G Loss: 1.1457
[Epoch 37/50] D Loss: 0.5964 | G Loss: 1.1669
[Epoch 38/50] D Loss: 0.5906 | G Loss: 1.1938
[Epoch 39/50] D Loss: 0.5852 | G Loss: 1.2134
[Epoch 40/50] D Loss: 0.5783 | G Loss: 1.2505
[Epoch 41/50] D Loss: 0.5721 | G Loss: 1.2486
[Epoch 42/50] D Loss: 0.5680 | G Loss: 1.2826
[Epoch 43/50] D Loss: 0.5626 | G Loss: 1.3041
[Epoch 44/50] D Loss: 0.5585 | G Loss: 1.3261
[Epoch 45/50] D Loss: 0.5491 | G Loss: 1.3501
[Epoch 46/50] D Loss: 0.5434 | G Loss: 1.3704
[Epoch 47/50] D Loss: 0.5387 | G Loss: 1.3913
[Epoch 48/50] D Loss: 0.5329 | G Loss: 1.4248
[Epoch 49/50] D Loss: 0.5236 | G Loss: 1.4567
[Epoch 50/50] D Loss: 0.5219 | G Loss: 1.4742
Training complete.
```

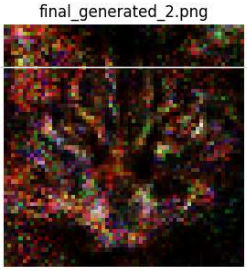
```
# Generate Exactly 5 Final Images
# ----------------------------
generator.eval()
with torch.no_grad():
    final_noise = torch.randn(5, latent_dim, device=device)
    final_imgs = generator(final_noise)
    final_imgs = (final_imgs + 1) / 2  # Rescale to [0,1]

    for idx in range(5):
        image_path = f"{save_dir}/final_generated_{idx+1}.png"
        vutils.save_image(final_imgs[idx], image_path)
        print(f"Saved final generated image: {image_path}")
```

```
Saved final generated image: ./images/final_generated_1.png
Saved final generated image: ./images/final_generated_2.png
Saved final generated image: ./images/final_generated_3.png
Saved final generated image: ./images/final_generated_4.png
Saved final generated image: ./images/final_generated_5.png
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Paths to the 5 generated images
image_paths = [
    "./images/final_generated_1.png",
    "./images/final_generated_2.png",
    "./images/final_generated_3.png",
    "./images/final_generated_4.png",
    "./images/final_generated_5.png"
]

# Load all images
images = [mpimg.imread(path) for path in image_paths]

# Create a figure with 5 subplots
fig, axes = plt.subplots(1, 5, figsize=(15, 3))

for ax, img, path in zip(axes, images, image_paths):
    ax.imshow(img)
    ax.set_title(path.split('/')[-1])  # Show filename as title
    ax.axis('off')  # Hide axes

plt.tight_layout()
plt.show()
```

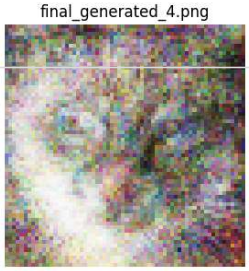
final_generated_1.png

final_generated_2.png



final_generated_3.png

final_generated_4.png



final_generated_5.png