```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.utils as vutils
from torch.utils.data import DataLoader
import medmnist
from medmnist import INFO
import numpy as np
from torch.utils.tensorboard import SummaryWriter
from torchmetrics.image.fid import FrechetInceptionDistance
import os
```

```
# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
print(device)
```

```
cuda
```

```
# Hyperparameters
image_size = 64
batch_size = 128
latent_dim = 100
num_epochs = 50
lr = 0.0002
beta1 = 0.5
lambda_gp = 10  # Gradient penalty coefficient for WGAN-GP

dataset_name = "pathmnist"
info = INFO[dataset_name]
data_flag = dataset_name
num_classes = len(info["label"])
```

```
# Dataset preparation
transform = transforms.Compose([
    transforms.Resize(image_size),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

dataset = medmnist.PathMNIST(split="train", download=True, transform=transform)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

```
Downloading https://zenodo.org/records/10519652/files/pathmnist.npz?download=1 to C:\Users\rushi\.medmnist\pathmnist.npz
100%|██████████| 205615438/205615438 [00:20<00:00, 9908975.87it/s]
```

```
# Define Generator class
class Generator(nn.Module):
    def __init__(self, latent_dim):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.ConvTranspose2d(latent_dim, 512, 4, 1, 0, bias=False),
            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, x):
        return self.model(x)
```

```
# Define Discriminator class (shared across all GANs)
class Discriminator(nn.Module):
```

```
        def __init__(self):
            super(Discriminator, self).__init__()
            self.model = 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, x):
            return self.model(x).view(-1, 1)


    # Initialize models
    generator = Generator(latent_dim).to(device)
    discriminator = Discriminator().to(device)



    # Loss functions for different GANs
    criterion_bce = nn.BCELoss()  # For Vanilla GAN
    criterion_mse = nn.MSELoss()  # For LS-GAN



    # Optimizers
    optimizer_g = optim.Adam(generator.parameters(), lr=lr, betas=(beta1, 0.999))
    optimizer_d = optim.Adam(discriminator.parameters(), lr=lr, betas=(beta1, 0.999))


    # TensorBoard setup
    writer = SummaryWriter(log_dir="./runs")


    # FID Score for evaluation
    fid = FrechetInceptionDistance(feature=64).to(device)
```

```
Downloading: "https://github.com/toshas/torch-fidelity/releases/download/v0.2.0/weights-inception-2015-12-05-6726825d.pth" to C:\Us
100%|██████████| 91.2M/91.2M [00:08<00:00, 11.2MB/s]
```

```
    # Training loop
    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)

            noise = torch.randn(batch_size, latent_dim, 1, 1, device=device)
            fake_images = generator(noise)

            # Vanilla GAN Loss
            real_labels = torch.ones_like(discriminator(real_images), device=device)
            fake_labels = torch.zeros_like(discriminator(fake_images), device=device)
            loss_real = criterion_bce(discriminator(real_images), real_labels)
            loss_fake = criterion_bce(discriminator(fake_images.detach()), fake_labels)
            loss_d_vanilla = loss_real + loss_fake

            # Least Squares GAN Loss
            loss_real_ls = criterion_mse(discriminator(real_images), real_labels)
            loss_fake_ls = criterion_mse(discriminator(fake_images.detach()), fake_labels)
            loss_d_ls = loss_real_ls + loss_fake_ls

            # WGAN Loss (without gradient penalty)
            loss_d_wgan = -torch.mean(discriminator(real_images)) + torch.mean(discriminator(fake_images.detach()))

            # Update Discriminator
            optimizer_d.zero_grad()
            loss_d_vanilla.backward(retain_graph=True)
            loss_d_ls.backward(retain_graph=True)
            loss_d_wgan.backward()
            optimizer_d.step()

            # Generator Loss (for all GANs)
            loss_g_vanilla = criterion_bce(discriminator(fake_images), real_labels)
```

```
            loss_g_ls = criterion_mse(discriminator(fake_images), real_labels)
            loss_g_wgan = -torch.mean(discriminator(fake_images))

            optimizer_g.zero_grad()
            loss_g_vanilla.backward(retain_graph=True)
            loss_g_ls.backward(retain_graph=True)
            loss_g_wgan.backward()
            optimizer_g.step()

        print(f"Epoch [{epoch}/{num_epochs}] D_Vanilla: {loss_d_vanilla.item():.4f}, D_LS: {loss_d_ls.item():.4f}, D_WGAN: {loss_d_wgan.iter

        # Compute FID
        fid.update((real_images * 255).byte(), real=True)
        fid.update((fake_images * 255).byte(), real=False)
        fid_score = fid.compute().item()
        writer.add_scalar("FID", fid_score, epoch)
        print(f"FID Score: {fid_score:.4f}")

    writer.close()
```

```
Epoch [0/50] D_Vanilla: 0.0002, D_LS: 0.0000, D_WGAN: -0.9998
FID Score: 2.4429
Epoch [1/50] D_Vanilla: 0.0517, D_LS: 0.0035, D_WGAN: -0.9502
FID Score: 2.5361
Epoch [2/50] D_Vanilla: 0.0118, D_LS: 0.0002, D_WGAN: -0.9883
FID Score: 2.2547
Epoch [3/50] D_Vanilla: 0.0395, D_LS: 0.0021, D_WGAN: -0.9616
FID Score: 2.0141
Epoch [4/50] D_Vanilla: 0.6008, D_LS: 0.1892, D_WGAN: -0.6262
FID Score: 1.8116
Epoch [5/50] D_Vanilla: 0.0237, D_LS: 0.0008, D_WGAN: -0.9766
FID Score: 1.9741
Epoch [6/50] D_Vanilla: 0.0018, D_LS: 0.0000, D_WGAN: -0.9982
FID Score: 2.0441
Epoch [7/50] D_Vanilla: 0.0010, D_LS: 0.0000, D_WGAN: -0.9990
FID Score: 2.5992
Epoch [8/50] D_Vanilla: 0.0111, D_LS: 0.0001, D_WGAN: -0.9889
FID Score: 2.2214
Epoch [9/50] D_Vanilla: 0.0207, D_LS: 0.0005, D_WGAN: -0.9796
FID Score: 2.0021
Epoch [10/50] D_Vanilla: 0.0144, D_LS: 0.0004, D_WGAN: -0.9858
FID Score: 2.1729
Epoch [11/50] D_Vanilla: 0.1121, D_LS: 0.0326, D_WGAN: -0.9163
FID Score: 2.1133
Epoch [12/50] D_Vanilla: 0.0194, D_LS: 0.0010, D_WGAN: -0.9811
FID Score: 2.1880
Epoch [13/50] D_Vanilla: 0.0212, D_LS: 0.0007, D_WGAN: -0.9791
FID Score: 2.1806
Epoch [14/50] D_Vanilla: 0.0986, D_LS: 0.0114, D_WGAN: -0.9078
FID Score: 2.0435
Epoch [15/50] D_Vanilla: 0.0082, D_LS: 0.0001, D_WGAN: -0.9919
FID Score: 2.0998
Epoch [16/50] D_Vanilla: 0.0025, D_LS: 0.0000, D_WGAN: -0.9975
FID Score: 2.1110
Epoch [17/50] D_Vanilla: 0.0876, D_LS: 0.0079, D_WGAN: -0.9167
FID Score: 2.1024
Epoch [18/50] D_Vanilla: 1.5374, D_LS: 0.4845, D_WGAN: -0.3774
FID Score: 1.9848
Epoch [19/50] D_Vanilla: 0.0645, D_LS: 0.0073, D_WGAN: -0.9396
FID Score: 1.7677
Epoch [20/50] D_Vanilla: 0.0402, D_LS: 0.0020, D_WGAN: -0.9609
FID Score: 1.6876
Epoch [21/50] D_Vanilla: 0.0066, D_LS: 0.0001, D_WGAN: -0.9934
FID Score: 1.6119
Epoch [22/50] D_Vanilla: 0.0451, D_LS: 0.0030, D_WGAN: -0.9565
FID Score: 1.4995
Epoch [23/50] D_Vanilla: 0.4876, D_LS: 0.1642, D_WGAN: -0.6659
FID Score: 1.5224
Epoch [24/50] D_Vanilla: 0.0064, D_LS: 0.0000, D_WGAN: -0.9936
FID Score: 1.4183
Epoch [25/50] D_Vanilla: 0.2078, D_LS: 0.0595, D_WGAN: -0.8538
FID Score: 1.4174
Epoch [26/50] D_Vanilla: 3.8260, D_LS: 0.9478, D_WGAN: -0.0266
FID Score: 1.4019
Epoch [27/50] D_Vanilla: 0.7029, D_LS: 0.2252, D_WGAN: -0.5887
FID Score: 1.3798
Epoch [28/50] D_Vanilla: 0.0294, D_LS: 0.0011, D_WGAN: -0.9712
FID Score: 1.3980
```

```
print ("Done")
```

Start coding or generate with AI.