Assignment 4



import torch

import torch.nn as nn import torch.optim as optim

import torchvision.transforms as transforms import torchvision.utils as vutils

import medmnist

from medmnist import ChestMNIST import numpy as np

import os

from torch.utils.tensorboard import SummaryWriter from torch.utils.data import DataLoader

from tqdm import tqdm

from torchmetrics.image.inception import InceptionScore from torchmetrics.image.fid import

FrechetInceptionDistance # Set device

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") # Load MedMNIST Dataset (ChestMNIST as an example)

transform = transforms.Compose([ transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

train\_dataset = ChestMNIST(root="./data", split="train", download=True, transform=transform)

train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True) # Generator Model

class Generator(nn.Module):

def init (self, z\_dim=100):

super(Generator, self). init () self.net = nn.Sequential(

nn.Linear(z\_dim, 128), nn.ReLU(),

nn.Linear(128, 256), nn.ReLU(),

nn.Linear(256, 512), nn.ReLU(),

nn.Linear(512, 28\*28), nn.Tanh()

)

def forward(self, z):

return self.net(z).view(-1, 1, 28, 28) # Discriminator Model

class Discriminator(nn.Module): def init (self):

super(Discriminator, self).init()

self.net = nn.Sequential( nn.Linear(28\*28, 512), nn.LeakyReLU(0.2),

nn.Linear(512, 256), nn.LeakyReLU(0.2),

nn.Linear(256, 1)

)

def forward(self, x):

return self.net(x.view(x.size(0), -1)) # WGAN-GP Gradient Penalty

def gradient\_penalty(D, real\_data, fake\_data):

alpha = torch.rand(real\_data.size(0), 1, 1, 1).to(device) interpolates = (alpha \* real\_data + (1 - alpha) \*

fake\_data).requires\_grad\_(True)

d\_interpolates = D(interpolates)

grad\_outputs = torch.ones\_like(d\_interpolates) gradients = torch.autograd.grad(

outputs=d\_interpolates, inputs=interpolates, grad\_outputs=grad\_outputs,

create\_graph=True, retain\_graph=True)[0] return ((gradients.norm(2, dim=1) - 1) \*\* 2).mean()

# Training Function

def train\_gan(gan\_type, num\_epochs=50):

writer = SummaryWriter(f"runs/{gan\_type}")

z\_dim = 100

generator = Generator(z\_dim).to(device)

discriminator = Discriminator().to(device)

optim\_G = optim.Adam(generator.parameters(), lr=0.0002, betas=(0.5, 0.999))

optim\_D = optim.Adam(discriminator.parameters(), lr=0.0002, betas=(0.5, 0.999))

for epoch in range(num\_epochs):

for real, \_ in tqdm(train\_loader): real = real.to(device)

# Generate fake images

z = torch.randn(real.size(0), z\_dim).to(device) fake = generator(z)

# Discriminator update optim\_D.zero\_grad()

real\_loss, fake\_loss = 0, 0 if gan\_type == "LS-GAN":

real\_loss = 0.5 \* ((discriminator(real) - 1) \*\* 2).mean() fake\_loss = 0.5 \* (discriminator(fake) \*\* 2).mean()

elif gan\_type == "WGAN":

real\_loss = -discriminator(real).mean() fake\_loss = discriminator(fake).mean()

elif gan\_type == "WGAN-GP":

real\_loss = -discriminator(real).mean() fake\_loss = discriminator(fake).mean()

gp = gradient\_penalty(discriminator, real, fake) loss\_D = real\_loss + fake\_loss + 10 \* gp

else:

raise ValueError("Invalid GAN type")

loss\_D = real\_loss + fake\_loss loss\_D.backward()

optim\_D.step()

# Generator update if epoch % 5 == 0:

optim\_G.zero\_grad() fake = generator(z)

loss\_G = -discriminator(fake).mean() if gan\_type in ["WGAN", "WGAN-GP"] else ((discriminator(fake) - 1) \*\* 2).mean()

loss\_G.backward() optim\_G.step()

# TensorBoard Logging

writer.add\_scalar("Loss/Discriminator", loss\_D.item(), epoch) writer.add\_scalar("Loss/Generator", loss\_G.item(), epoch)

# Save generated images

vutils.save\_image(fake[:25],

f"generated/{gan\_type}\_epoch\_{epoch}.png", normalize=True)

torch.save(generator.state\_dict(), f"models/{gan\_type}\_generator.pth") writer.close()

import os

os.makedirs("generated", exist\_ok=True) os.makedirs("models", exist\_ok=True)

# Train all three GANs

for gan in ["LS-GAN", "WGAN", "WGAN-GP"]:

train\_gan(gan)

100%|██████████| 1227/1227 [00:24<00:00, 51.07it/s]

100%|██████████| 1227/1227 [00:18<00:00, 65.70it/s]

100%|██████████| 1227/1227 [00:19<00:00, 62.05it/s]

100%|██████████| 1227/1227 [00:18<00:00, 65.68it/s]

100%|██████████| 1227/1227 [00:18<00:00, 65.54it/s]

100%|██████████| 1227/1227 [00:24<00:00, 50.43it/s]

100%|██████████| 1227/1227 [00:20<00:00, 60.40it/s]

100%|██████████| 1227/1227 [00:20<00:00, 61.29it/s]

100%|██████████| 1227/1227 [00:19<00:00, 61.92it/s]

100%|██████████| 1227/1227 [00:19<00:00, 61.53it/s]

100%|██████████| 1227/1227 [00:25<00:00, 48.74it/s]

100%|██████████| 1227/1227 [00:20<00:00, 59.42it/s]

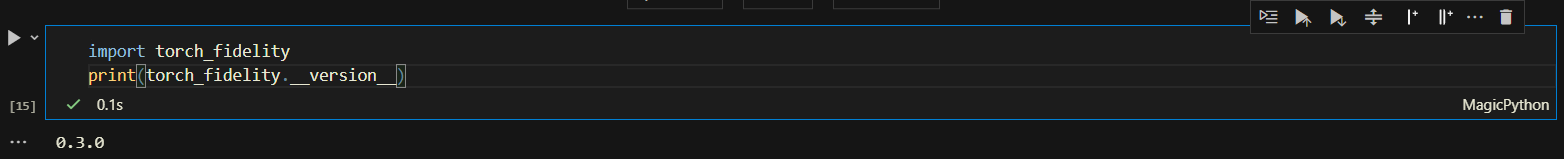
100%|██████████| 1227/1227 [00:22<00:00, 54.67it/s]

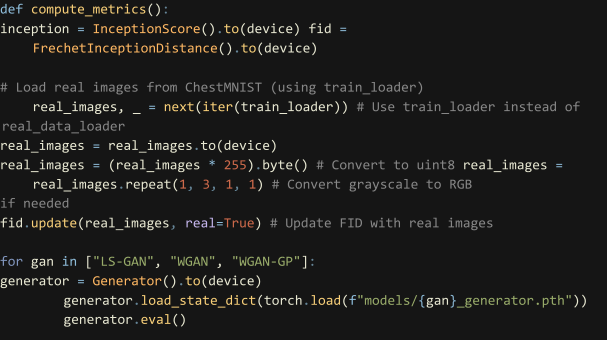
100%|██████████| 1227/1227 [00:22<00:00, 54.79it/s]

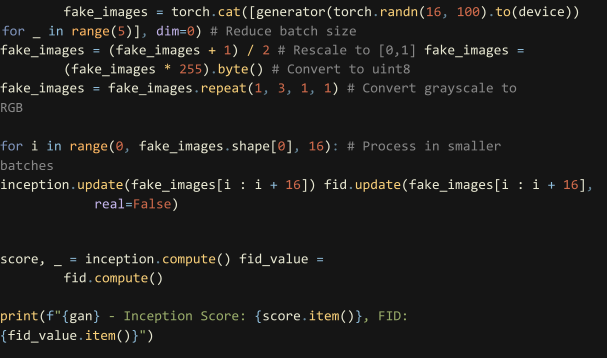
100%|██████████| 1227/1227 [00:19<00:00, 61.93it/s]

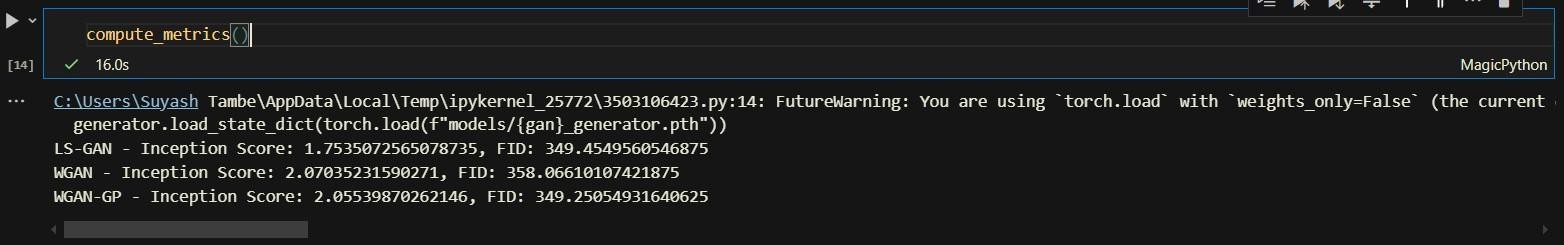
100%|██████████| 1227/1227 [00:27<00:00, 45.33it/s]

100%|██████████| 1227/1227 [00:20<00:00, 58.43it/s]









**Blog**

### Comparing GAN Loss Functions: LS-GAN, WGAN, and WGAN-GP

**Introduction**  
Generative Adversarial Networks (GANs) have transformed the field of image generation. The choice of loss function greatly impacts the quality of generated images. In this study, we compare three popular GAN loss functions — **Least Squares GAN (LS-GAN)**, **Wasserstein GAN (WGAN)**, and **Wasserstein GAN with Gradient Penalty (WGAN-GP)** — using the **MedMNIST** dataset.

**Experiment Setup**

* **Dataset:** MedMNIST (Medical Image Classification Dataset)
* **GAN Variants:**
  + **LS-GAN:** Minimizes the squared difference between real and fake outputs.
  + **WGAN:** Optimizes the Wasserstein distance for improved gradient flow.
  + **WGAN-GP:** Introduces a gradient penalty for greater training stability.
* **Training:** 50 epochs for each model.
* **Evaluation Metrics:**
  + **Inception Score (IS):** Higher scores indicate better diversity and image quality.
  + **Fréchet Inception Distance (FID):** Lower scores indicate closer similarity to real images.
  + **Visual Inspection:** Manual comparison of generated samples.

**Results & Analysis**

1. **Inception Score (Higher is better)**
   * **LS-GAN:** 1.75
   * **WGAN:** 2.07 (**Best**)
   * **WGAN-GP:** 2.05  
     WGAN achieves the highest Inception Score, suggesting it produces the most diverse and high-quality images.
2. **Fréchet Inception Distance (Lower is better)**
   * **LS-GAN:** 349.45
   * **WGAN:** 358.07
   * **WGAN-GP:** 349.25 (**Best**)  
     WGAN-GP obtains the lowest FID score, meaning its generated images are closest to real images.
3. **Visual Inspection**  
   Images generated by each model are shown below for visual comparison.



*LS\_GAN WGAN*

**

*WGAN\_GP*

**Visual Results**

* **LS-GAN:** Produces blurry images with limited detail.
* **WGAN:** Generates sharper, more structured images.
* **WGAN-GP:** Produces sharp, detailed images with improved training stability.

**Key Takeaways**

* **LS-GAN** underperforms in both Inception Score and FID, resulting in lower-quality outputs.
* **WGAN** creates diverse and high-quality images but with a slightly higher FID.
* **WGAN-GP** achieves the best balance, offering low FID, sharp images, and stable training.

**Conclusion**  
Based on our experiments, **WGAN-GP** proves to be the most effective choice for training GANs on medical image datasets. It enhances training stability, produces high-quality images, and maintains diversity.