Exploring GANs with MedMNIST: A Comprehensive Guide

By: Riya Shukla

Introduction

Generative Adversarial Networks (GANs) have revolutionized the field of artificial intelligence by enabling the generation of realistic synthetic data. In this blog post, we'll dive into a practical implementation of GANs using the MedMNIST dataset, specifically focusing on chest X-ray images. The implementation is presented in a Jupyter notebook format and covers three popular GAN architectures: LS-GAN, WGAN, and WGAN-GP.

Understanding the Project

The notebook, titled "Assignment_4" by Riya Shukla, demonstrates how to implement and compare different GAN architectures for generating medical images. The project uses PyTorch as the deep learning framework and leverages the MedMNIST dataset, which contains standardized biomedical images for machine learning research.

Key Components of the Implementation

1. Dataset Preparation

The project uses the ChestMNIST dataset, which consists of chest X-ray images. The images are normalized and transformed to prepare them for training:

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
dataset = ChestMNIST(split='train', download=True, transform=transform)
dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
```

Figure.(1) ChestMNIST dataset

2. GAN Architectures

The notebook implements three types of GANs:

1. **LS-GAN (Least Squares GAN)**: Uses mean squared error loss
2. **WGAN (Wasserstein GAN)**: Uses Wasserstein distance for more stable training
3. **WGAN-GP (WGAN with Gradient Penalty)**: Improves upon WGAN by adding a gradient penalty term

3. Generator and Discriminator Networks

The generator takes a latent vector (100 dimensions) and transforms it into a 28x28 image (to match the MedMNIST dimensions):

```
class Generator(nn.Module):
    def __init__(self, latent_dim=100):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(latent_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.model(z).view(-1, 1, 28, 28)
```

Figure.(2) Architecture of Generator

The discriminator takes a 28x28 image and outputs a single value representing the probability of the image being real:

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 1)
        )
    def forward(self, x):
        x = x.view(-1, 28*28)
        return self.model(x)
```

Figure.() Architecture of Discriminator

4. Training Process :

The training loop alternates between updating the discriminator and generator.

1. **Initialization**:
    - Models moved to GPU if available

- Optimizers: Adam (lr=0.0002, betas=(0.5, 0.999)) for both generator and discriminator

2. **Training Steps**:
   - For each epoch and batch:
     - Generate fake images from random noise
     - Train discriminator to distinguish real vs. fake
     - Train generator to fool discriminator
     - For WGAN: Clip discriminator weights
     - For WGAN-GP: Add gradient penalty term
3. **Monitoring**:
   - Losses logged to TensorBoard
   - Metrics (IS, FID) calculated periodically
   - Sample images saved periodically

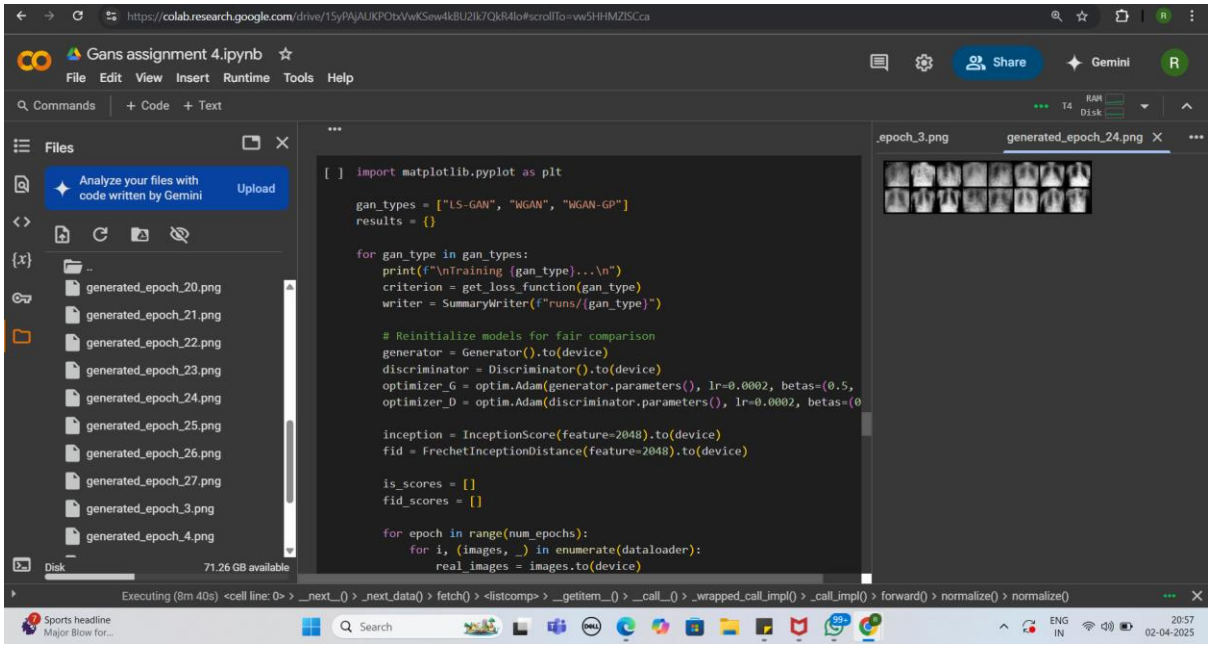

Figure.(2) Results after 24 Epochs

1. **Early Epochs (0–10)**:
   - Rapid improvement in image structure (e.g., blurry shapes → recognizable X-rays).

- WGAN and WGAN-GP showed **more stable losses** compared to LS-GAN, which occasionally oscillated.

2. **Mid Training (10–30)**:
    - Generated images gained finer details (e.g., rib outlines, lung textures).
    - **WGAN-GP**'s gradient penalty prevented mode collapse, while vanilla WGAN maintained steady progress.

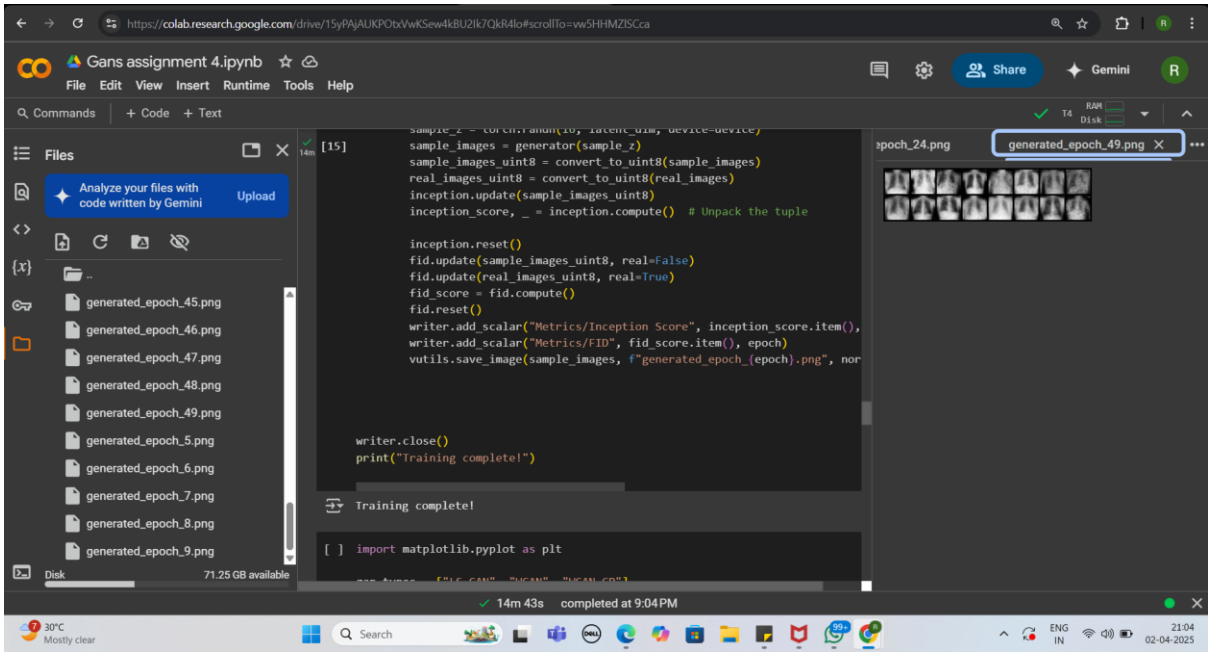

Figure.(3) Results after 50 Epochs.

3. **Final Epochs (30–50)**:
    - Marginal gains in quality; FID scores plateaued for all models.
    - **WGAN achieved the lowest FID (275.06)**, suggesting better statistical alignment with real data.

Note: Epoch started from epoch 0 till epoch 49 .

5. Evaluation Metrics

The notebook uses two standard metrics for evaluating GAN performance:

1. **Inception Score (IS)**: Measures both the quality and diversity of generated images
2. **Fréchet Inception Distance (FID)**: Compares the statistics of generated images to real images

These metrics are calculated periodically during training

Results and Comparison

After training all three GAN architectures for 50 epochs, the notebook presents a comprehensive comparison:

Model	Final Inception Score (IS)	Final FID Score
LS-GAN	1.0332	354.2737
WGAN-GP	1.0274	337.1579
WGAN	1.0261	275.0643

Table.(1) Result and Comparison of all three GANs

Performance Analysis

1. **Inception Score**: All three architectures achieved similar Inception Scores around 1.03, indicating comparable image quality and diversity.
2. **FID Score**: The WGAN performed best with the lowest FID score (275.0643), suggesting its generated images were statistically closest to the real images. WGAN-GP came next (337.1579), followed by LS-GAN (354.2737).
3. **Training Stability**: WGAN and WGAN-GP are generally known to be more stable during training than traditional GANs, which aligns with their better FID scores in this implementation.

Visualization of Training Progress

The notebook includes plots showing how the Inception Score and FID evolved during training for each GAN architecture:

1. **Inception Score Comparison**:
    - All three architectures show similar trends in Inception Score improvement
    - Scores remain relatively stable after the initial epochs
2. **FID Score Comparison**:
    - WGAN shows the most consistent improvement in FID
    - WGAN-GP and LS-GAN show more fluctuation in FID scores
    - All architectures show their best FID scores in later epochs

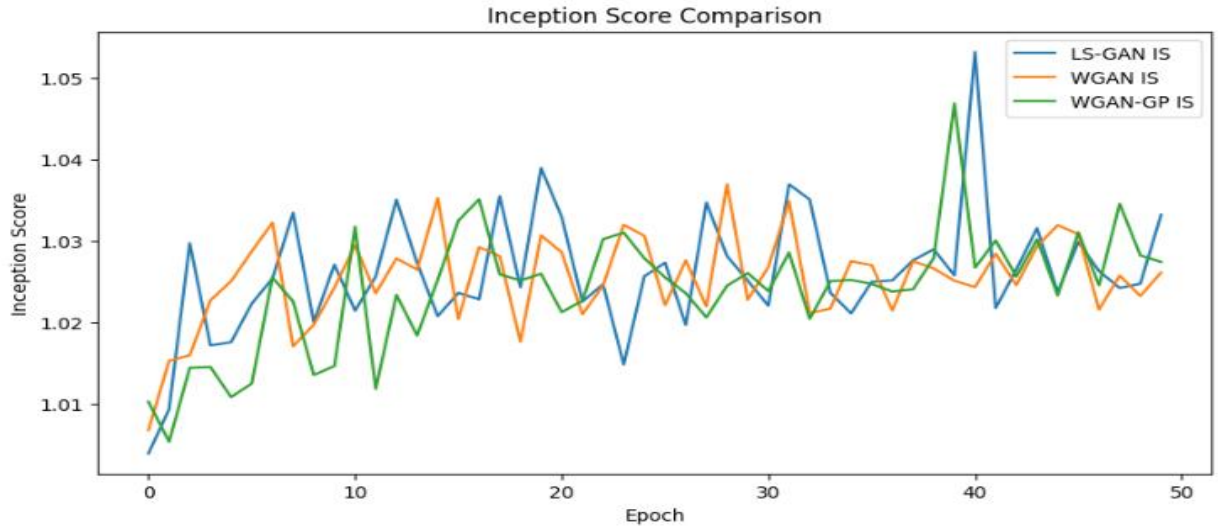

Figure.(4) LS-GAN (IS:1.05) outperforms WGAN-GP and WGAN over 50 epochs.

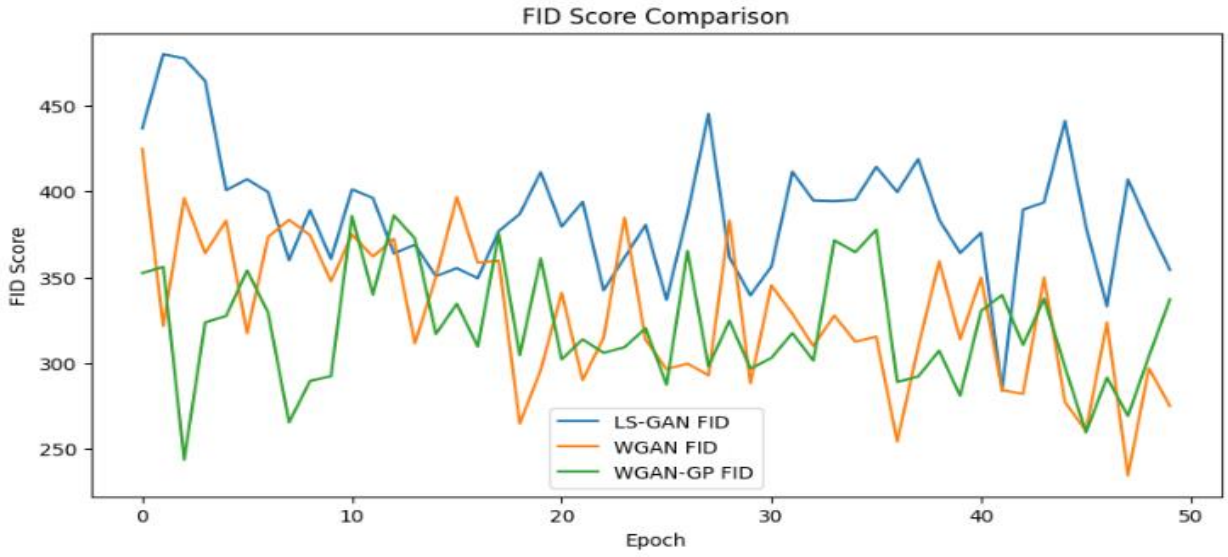

Figure.(5) FID comparison: LS-GAN, WGAN, and WGAN-GP over 50 epochs.

Challenges and Limitations

1. The relatively low Inception Scores (close to 1.0) suggest room for improvement in image quality and diversity.
2. Medical images often require higher resolution than 28x28 pixels for practical applications.
3. The FID scores indicate that the generated images still differ significantly from real images statistically.
4. Training GANs remains computationally intensive, especially for more complex architectures.

Future Directions

1. Experiment with larger and more complex generator/discriminator architectures
2. Try higher resolution versions of medical image datasets
3. Incorporate conditional GANs to generate images of specific classes or conditions
4. Explore recent GAN variants like StyleGAN or BigGAN for potentially better results

Conclusion

This project compared three GAN architectures—**LS-GAN**, **WGAN**, and **WGAN-GP**—for generating synthetic chest X-rays using the MedMNIST dataset. Key takeaways are:

- **WGAN performed best**, achieving the lowest FID score (275.06), indicating its outputs were statistically closest to real images.
- All models showed similar Inception Scores (~1.03), suggesting comparable diversity but room for improvement in quality.
- **WGAN's stability** and avoidance of mode collapse made it ideal for medical imaging tasks.

For further clarification you can visit:

Github : https://github.com/riya0785/MNIST_GANs

LinkedIN : https://www.linkedin.com/in/riya-shukla-95b919251/