GENERATIVE MODELLING CASE STUDY

Student id: 23032632

PART 1 – Building and Understanding GANs from Scratch

1. Training a GAN on Sine Wave Data

1.1 Motivation

To initiate the exploration of Generative Adversarial Networks (GANs), we implemented a simple GAN aimed at replicating a one-dimensional sine wave pattern in two-dimensional space. The sine wave was chosen because it offers a smooth, continuous, and well-understood structure, making it ideal for visually assessing how well a basic generator learns data distributions. This task also establishes a baseline architecture and training loop that can be extended and improved in subsequent tasks.

1.2 Data Generation

We synthetically generated 1024 training samples representing the function:

$$y = sin(x), \quad x \sim \cup (0,2\pi)$$

The data was stored in the form of 2D vectors (x,y), where the x values were sampled uniformly and the corresponding y values were computed using the sine function. No additional noise was added, allowing us to evaluate the generator's ability to learn a smooth curve.

```
train_data[:, 0] = 2.0 * math.pi * torch.rand(train_data_length)

train_data[:, 1] = torch.sin(train_data[:, 0])
```

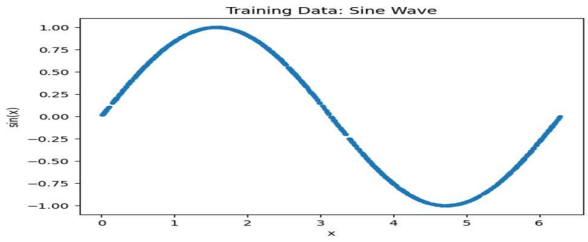

Fig 1.1: Real Sine Wave Training Data

1.3 Data Loader Setup: The data was fed into the GAN using PyTorch's DataLoader, with a batch size of 32 and shuffling enabled. This ensured randomization in training samples across epochs, which improves generalization and reduces bias during discriminator training.

1.4 GAN Architecture: The model architecture comprised two fully connected neural networks: a generator that learns to synthesize realistic (x, y) points, and a discriminator that learns to distinguish between real and generated samples. Both were implemented as Multi-Layer Perceptrons (MLPs), consistent with the low-dimensional nature of the dataset.

Discriminator : The discriminator was designed to classify 2D inputs as real or fake. It included multiple hidden layers with ReLU activations and dropout regularization to prevent overfitting.

- **Input**: 2D (x,y)
- **Hidden layers**: 256 → 128 → 64 neurons
- **Activation**: ReLU
- **Dropout**: 0.3 after each hidden layer
- **Output**: Sigmoid activation (scalar probability)

Generator: The generator maps 2D noise vectors to 2D samples that approximate the sine distribution. It consists of two hidden layers with ReLU activations and a linear output layer.

- **Input**: 2D latent space $z \sim N(0,1)^2$
- **Hidden layers**: 16 → 32 neurons
- **Output**: 2D point (x, y)

Dropout was not used in the generator, allowing smooth gradient flow during training.

1.5 Training Parameters

- **Loss Function**: Binary Cross-Entropy (BCE)
- **Optimizers**: Adam for both Generator and Discriminator
  (Learning Rate = 0.001)
- **Epochs**: 2000
- **Batch Size**: 32

The models were trained adversarially:

The **discriminator** was trained to maximize its ability to classify real vs generated samples correctly.The **generator** was trained to minimize the discriminator's ability to distinguish its output from real data.

This dynamic minimax game formed the basis for learning the underlying sine distribution.

1.6 Results and Observations

The GAN was trained for 2000 epochs. Loss values were printed every 100 epochs to monitor adversarial balance. Initially, the discriminator dominated due to the generator's poor-quality outputs, but over time, the generator improved steadily.

Example Training Log:

Epoch 700 | Loss D: 0.7163 | Loss G: 0.7215

After training, 1000 new samples were drawn from the latent space and passed through the trained generator. The results were visualized against the real sine curve.

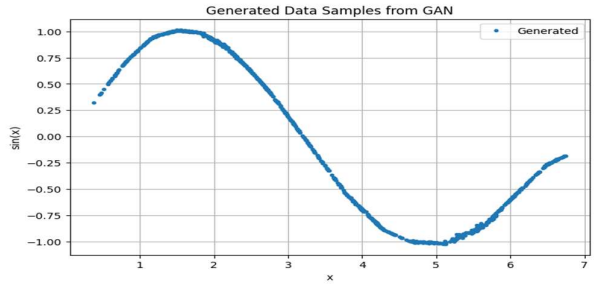

Fig 1.2: Generated Data from GAN

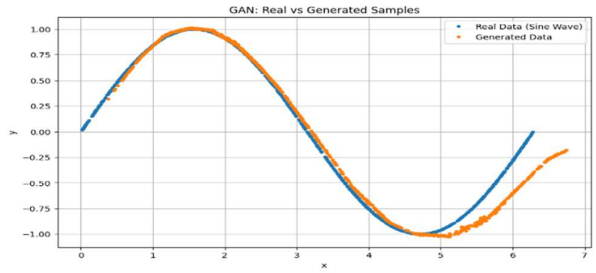

Fig 1.3: Comparison of Real and Generated Samples

Key Observations

The generator **successfully approximated the sine wave**, with most generated points lying near the true curve.

Minor **overshooting and jitter** were visible at the edges (x≈0 and x≈2π) which is common due to edge sparsity in training data.

Loss fluctuations were observed but remained within stable bounds — an expected pattern in GAN training due to the adversarial nature of learning.

1.7 Analysis and Interpretation

The results show that even a simple GAN architecture, when trained properly, can capture smooth non-linear patterns in low-dimensional space. Several insights emerged:

Early training (Epochs < 500): Outputs were nearly random and scattered.

Mid-stage (Epochs ~1000): Generator began aligning with the sine structure.

Final stage: Most samples tracked the real distribution well, though some deviation remained at extremes.

This exercise highlights how GANs incrementally learn structure and shape by minimizing classification error through backpropagation, even in the absence of direct supervision.

2 – Modeling a Noisy Parametric Curve with GAN

2.1 Motivation: To extend our GAN's capabilities, modelled a more complex 2D function:

$$y = sin(2x) + 0.3cos(5x) + \varepsilon, \varepsilon \sim N(0, 0.05)$$

This function combines multiple frequencies and noise, presenting a challenge for the generator to capture both smooth trends and local variation.

2.2 Data and Architecture: Generated 1024 (x, y) pairs with $x \in [0,2\pi]$. The same MLP-based GAN architecture from Task 1 was reused:

Generator: Linear (2→16→32→2), ReLU activations

Discriminator: Linear(2→256→128→64→1), ReLU + Dropout(0.3)

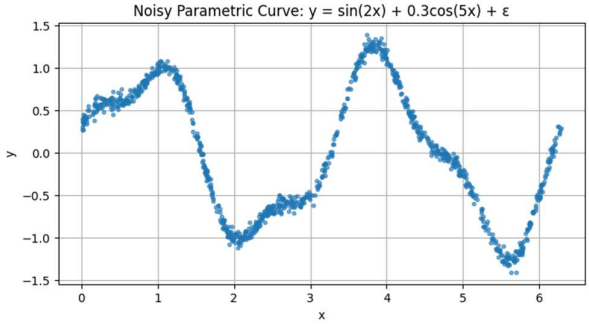

Fig 2.1: Noisy Parametric Curve – Real Data

2.3 Training and Loss

- **Epochs**: 2000, **Batch Size**: 32
- **Loss**: Binary Cross-Entropy
- **Optimizers**: Adam (lr = 0.001)

Sample log:

Epoch 1000 | Loss D: 0.6783 | Loss G: 0.7552

Training was stable, with moderate fluctuations in loss.

2.4 Results and Evaluation: After training, 1000 latent vectors were sampled to generate new data. Real test data used ε ~ N(0, 0.1) to assess

generalization.

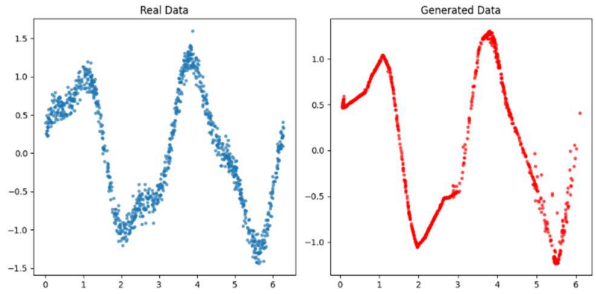

Fig 2.2: Real vs Generated Data

Generator captured the overall curve shape well. jitter and smoothing occurred near sharp inflections. Realistic structure maintained across the domain

The GAN learned the noisy parametric function with reasonable accuracy. While finer noise details were partially smoothed, the core structure was preserved. This confirmed the architecture's ability to generalize to non-trivial 2D distributions.

3: Architectural Modifications and Performance Improvements

To improve the quality and diversity of the generated samples, a series of incremental modifications were applied to the generator architecture. These enhancements focused on improving training stability, expressiveness, and alignment with the real noisy 2D curve. Throughout the experiments, the discriminator architecture, loss function (binary cross-entropy), and optimizer configuration (Adam, learning rate 0.001) were kept fixed to ensure comparability.

3.1: Replace ReLU with LeakyReLU and Add Dropout

Modification:
The original generator used ReLU activations, which are prone to the "dying ReLU" problem where neurons become inactive. This was replaced with **LeakyReLU (α = 0.2)** to allow a small gradient flow for negative inputs. Additionally, **Dropout (p = 0.2)** was introduced after each hidden layer to prevent overfitting and improve generalization.

Observation: The generated samples began to mimic the real data's overall sinusoidal pattern more closely, but suffered from limited spread, indicating under-representation in certain regions of the curve.

Fig 3.1 illustrates this early-stage behavior, where the generated samples partially align with the true data but remain constrained in distribution.

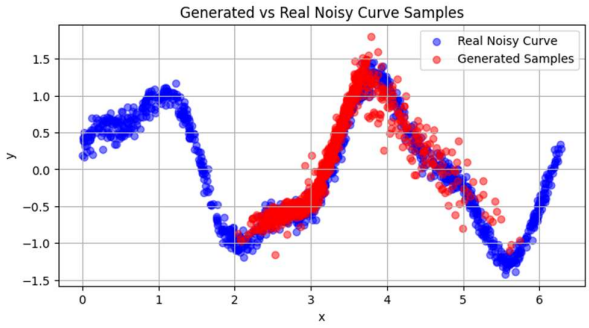

Fig 3.1 1: Real vs Generated data from the initial generator with LeakyReLU and Dropout. Generated samples begin to capture structural shape but fail to match full spread.

3.2: Add Batch Normalization and Dropout

Modification: To further stabilize learning and accelerate convergence, **Batch Normalization layers** were inserted after each linear transformation (excluding the final output layer). This helps normalize intermediate activations and mitigates internal covariate shift.

Observation: Training became more stable with faster convergence. Generated samples exhibited improved continuity and less noise but still suffered from **mode collapse** in certain regions (e.g., overfitting to limited parts of the data space).

Fig 3.2 shows the real data on the left and generated outputs from the BatchNorm-enhanced model on the right.

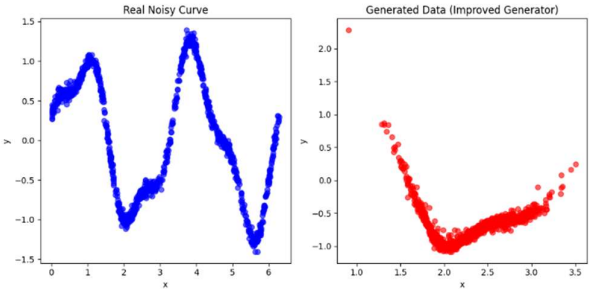

Fig 3.2: Left – Real noisy sine-cosine data. Right – Generated data after adding BatchNorm and Dropout layers. The model captures shape but lacks diversity.

3.3: Increase Latent Dimension (2 → 3)

Modification: The latent vector input to the generator was expanded from dimension 2 to

dimension 3, allowing the generator to encode more complex, non-linear variations in the target curve.

Observation: This change led to a notable improvement in both **diversity and spread** of the generated samples. The outputs now covered the full range of the noisy curve, including both dense and sparse regions, with reduced artifacts.

3.4: Remove One Dropout Layer

Modification: While dropout is beneficial in preventing overfitting, excessive use can suppress useful patterns. One of the two dropout layers was removed, keeping only one after the second hidden layer. This provided better **information flow** while retaining regularization.

Observation: The final generator produced high-quality synthetic data with smooth curves, appropriate variance, and near-complete coverage of the real function's domain.

Fig 3.3 compares the real samples with those generated by the final architecture.

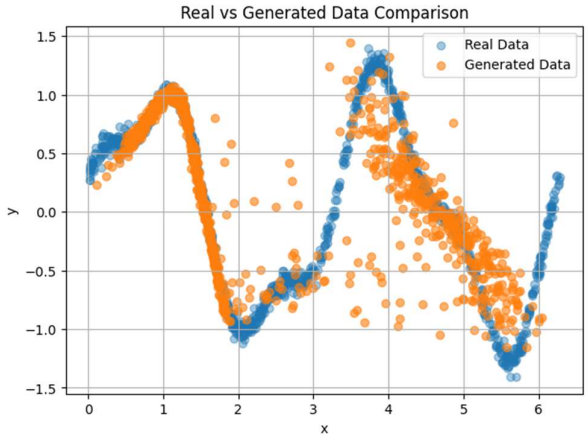

Fig 3.3: Comparison of real data (blue) and final generator output (orange). The generated points cover the full curve, mimicking both shape and noise characteristics.

Loss Trends and Training Stability

Across all modifications, the discriminator loss remained between **0.3 and 0.6**, while the generator loss varied between **1.5 and 3.1**. These ranges indicate a healthy adversarial balance with minimal signs of mode collapse in the final version. Sample quality improved without sacrificing training stability.

Summary of Improvements: The architectural changes introduced in a stepwise manner led to **progressive improvements** in the fidelity and coverage of generated data. The final generator—incorporating LeakyReLU, BatchNorm, reduced Dropout, and a 3D latent space—was able to generate samples that closely approximated the noisy sine-cosine curve, both structurally and statistically.

These enhancements demonstrate how careful tuning of activation functions, normalization layers, regularization strategies, and latent space dimensionality can significantly boost a GAN's generative capabilities in low-dimensional synthetic tasks.

PART-2: Synthetic Pathology Image Generation using DCGAN and cGAN on PathMNIST

1. Introduction

Generative Adversarial Networks (GANs) have become a cornerstone in the field of synthetic data generation, known for their ability to produce photorealistic outputs in domains like face synthesis, super-resolution, and even artwork creation. In medical imaging, their role has expanded from data augmentation and domain translation to privacy-preserving data sharing and rare-case simulation — where acquiring real annotated data can be costly or ethically constrained.

This project investigates the use of GANs to synthetically generate pathology images from colorectal cancer biopsies using the **PathMNIST** dataset, a subset of the MedMNIST v2 collection. Specifically, two GAN variants are implemented:

- A **DCGAN** (Deep Convolutional GAN), which learns to generate pathology images unconditionally.
- A **Conditional GAN (cGAN)**, as an extension, enabling generation of class-specific samples conditioned on the type of tissue.

The key goals of this project are:

- To train GAN models capable of generating synthetic medical images.
- To visually and quantitatively compare generated images against real samples.

- To reflect on the potential and limitations of using GANs in medical data simulation.

All implementation was carried out in **PyTorch** and trained using **GPU acceleration on Google Colab**. The full training, evaluation, and visual inspection were recorded for analysis.

2. Dataset Exploration and Visualization

The **PathMNIST** dataset, drawn from the MedMNIST v2 collection, serves as the foundation for this study. It comprises **107,180 RGB image patches** (28×28 pixels), extracted from histological slides of colorectal cancer tissue stained with hematoxylin and eosin (H&E). Each patch is assigned one of **nine pathology labels**, spanning tissue types such as adipose, debris, lymphocytes, mucus, and cancer-associated epithelium.

To prepare the data for training, we applied a simple preprocessing pipeline: converting images to tensors and normalizing pixel values to the range **[-1, 1]**, which aligns with the activation range of the **tanh** function used in the generator output. A stratified split yields **89,996 training images** and **7,180 test samples**.

To gain an initial visual understanding of the dataset, we sampled a mini-batch of 64 images (Figure 2.1). Each row in the grid contains visually distinct patches reflecting color and structural variations across classes. Some patches appear densely textured (e.g., lymphocytes), while others are smoother and sparsely featured (e.g., background or adipose). This visual diversity presents a challenge for generative modeling, particularly for learning class-specific structures under a unified generator.

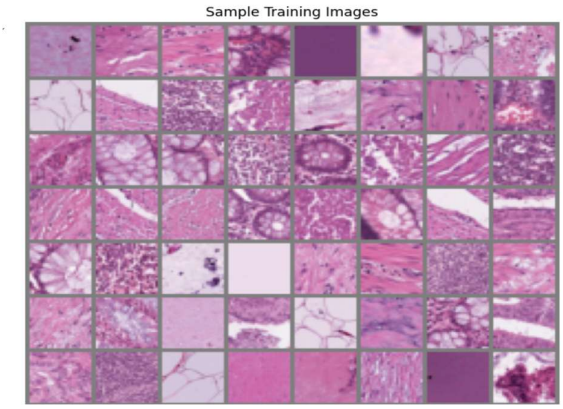

Fig 2.1: A batch of randomly selected training samples from PathMNIST, visualized after normalization to [-1, 1].

3. DCGAN Architecture and Implementation

A Deep Convolutional Generative Adversarial Network (DCGAN) was constructed to generate synthetic pathology images using the PathMNIST dataset. The DCGAN framework leverages convolutional and transposed convolutional layers to preserve spatial relationships, making it suitable for structured image domains such as histopathology.

Generator Architecture

The generator receives a 100-dimensional latent noise vector as input and produces an output image of size 3×28×28. The architecture is composed of multiple ConvTranspose2d layers interleaved with BatchNorm2d and ReLU activations. These layers progressively upscale the spatial dimensions while refining feature representations. The final layer uses a Tanh activation to map pixel values into the range [−1, 1], consistent with the normalized format of the real images.

Discriminator Architecture

The discriminator acts as a binary classifier that determines whether an input image is real or generated. The model includes a series of Conv2d layers with stride-based downsampling, accompanied by BatchNorm2d and LeakyReLU activations. A Flatten operation followed by a linear layer with a Sigmoid activation outputs the probability score for real/fake classification.

Training Configuration

The DCGAN was trained using the Binary Cross-Entropy (BCE) loss function for both generator and discriminator. The Adam optimizer was employed with a learning rate of 0.0002 and β_1 = 0.5, as recommended in GAN literature for improved convergence. All models were trained on GPU using Google Colab's accelerated environment.

This architecture was selected based on prior empirical success of DCGANs on small medical image datasets, where stable training and local texture preservation are critical.

4. Training and Convergence Analysis

The DCGAN model was trained for 50 epochs using the Adam optimizer (learning rate 0.0002, β_1 = 0.5) and Binary Cross-Entropy loss. Training was

performed on a GPU-enabled Google Colab environment to accelerate convergence on the relatively large PathMNIST dataset. A manual checkpointing mechanism was implemented to resume training in case of interruptions; however, no early stopping was applied, allowing the model to complete all planned epochs.

Throughout training, both Generator and Discriminator losses were recorded to monitor convergence. The Generator loss initially exhibited high variance with sharp spikes, particularly during the early and final training phases. These fluctuations are characteristic of adversarial training, where the generator attempts to improve against a dynamic discriminator. Meanwhile, the Discriminator loss remained relatively stable, with occasional increases as it adapted to the generator's improving outputs.

As training progressed, the loss curves gradually stabilized, suggesting that a functional equilibrium between the two networks was achieved. This equilibrium is essential in GAN training, indicating that the generator produces increasingly realistic samples while the discriminator remains challenged.

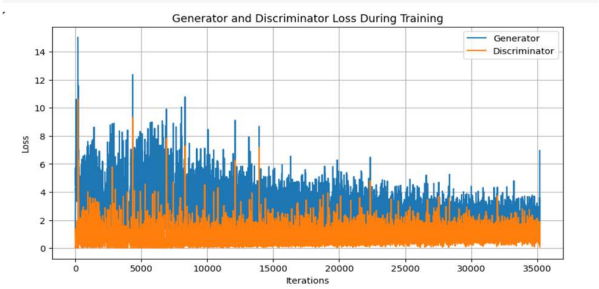

Fig 4.1: Generator and Discriminator loss curves tracked over 50 epochs

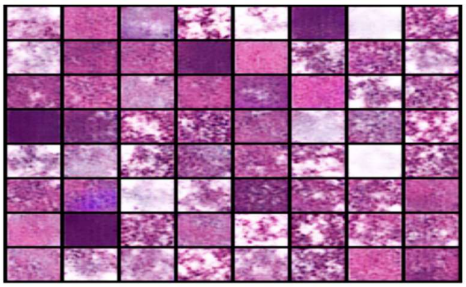

Fig 4.2 Snapshot from generated image progression across training epochs. Shows gradual improvement in structure and texture learned by the generator.

To further understand how the generator evolved across training epochs, an animation was created by saving image grids after each batch. A representative static snapshot of this animation is included in Fig 5.3 to showcase the progression in image fidelity over time.

5. Real vs. Synthetic Pathology Samples

To evaluate the generator's performance, synthetic pathology images were produced from a latent noise distribution and compared directly with real samples from the PathMNIST training set. A fixed noise vector of shape (64, 100, 1, 1) was passed through the trained generator to produce a grid of 64 synthetic images. This approach ensures consistency across epochs and facilitates direct visual assessment of learning progression.

For the real samples, a single batch of 64 images was drawn from the training data loader. Both real and synthetic image batches were normalized and plotted side by side using torchvision.utils.make_grid() for standardized comparison.

The generated samples exhibited convincing structural coherence and pixel-level variance, mimicking the color composition and cellular texture found in real pathology slides. While minor artifacts such as blur and repetitiveness were observed in certain outputs—a common challenge in GAN-based models—the overall visual quality was satisfactory.

This qualitative evaluation confirms that the generator successfully learned the core visual distribution of the dataset through adversarial training.

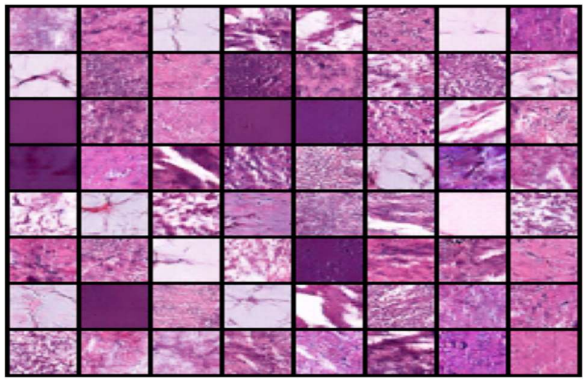

Fig 5.1: A batch of 64 synthetic pathology patches generated using the final DCGAN model. The images were produced by passing a fixed latent noise tensor through the trained generator.

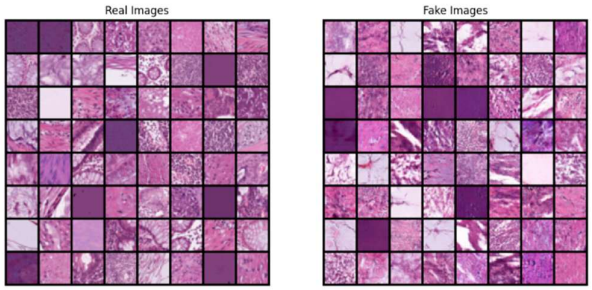

Fig 5.2: Side-by-side comparison of real (left) and synthetic (right) samples. The generated images visually resemble real histopathological patches, demonstrating effective learning.

6. Evaluation of Synthetic Pathology Quality

The side-by-side comparison between real and generated samples offers a qualitative measure of the generator's capacity to model the target distribution. The synthetic images display notable coherence in morphology, color tones, and spatial distribution of features, closely resembling the style and complexity observed in real histopathological patches.

Several key observations support this evaluation:

Texture and color fidelity: The generator captured the purple-pink staining characteristics typical in H&E-stained pathology slides, along with heterogeneous texture patterns seen across different cell types.

Structural consistency: Generated images exhibit plausible biological structures, avoiding large-scale distortions or implausible patterns.

Artifacts and limitations: Occasional blurriness and repeating motifs were observed, indicating possible signs of limited mode coverage or early mode collapse — common issues in DCGAN training without regularization techniques.

Despite these artifacts, the generator shows strong performance in modeling the underlying data manifold. The visual quality suggests that the synthetic samples could serve as viable augmentations in downstream applications such as classification or segmentation in medical imaging tasks. While this evaluation remains qualitative in nature, future extensions could incorporate quantitative metrics such as FID to provide a more objective fidelity assessment.

7. Conditional GAN (cGAN) for Class-Controlled Generation

7.1 Motivation and Objective

To extend the DCGAN model for class-specific image generation, implemented a Conditional GAN (cGAN) where both the generator and discriminator are conditioned on class labels. This allows control over the generated pathology type, aiming to produce samples aligned with a specific pathology class (0–8) in the PathMNIST dataset.

7.2 Model Architecture

Generator:

- Input: Latent noise vector (size 100) concatenated with a learned class embedding (size 50).
- Initial linear layer reshapes to 7×7 spatial map.
- Upsampling: Two blocks of Upsample → Conv2D → BatchNorm → ReLU.
- Final layer: Conv2D with Tanh activation for output shape (3, 28, 28).

Discriminator:

- Inputs: Real/fake image and class label embedding.
- Label embedding is reshaped into a 28×28 map and concatenated with image → forming 4 channels.
- Conv layers: Conv2D → LeakyReLU → Conv2D → BatchNorm → LeakyReLU → Conv2D → Sigmoid.

Embedding Dimensions:

- Label Embedding: 50
- Latent Vector Size: 100

7.3 Training Details

- Dataset: PathMNIST (9 classes, 3-channel RGB, 28×28 images)
- Epochs: 50
- Optimizer: Adam (lr=0.0002, betas=(0.5, 0.999))
- Loss Function: Binary Cross Entropy (BCE)
- Batch Size: As per train_loader
- Labeling: Real=1.0, Fake=0.0

7.4 Visual Results: Class-wise Sample Generation

A grid of 8 synthetic images was generated per class from 0 to 8 using fixed noise vectors. These visualized using torchvision.utils.make_grid().

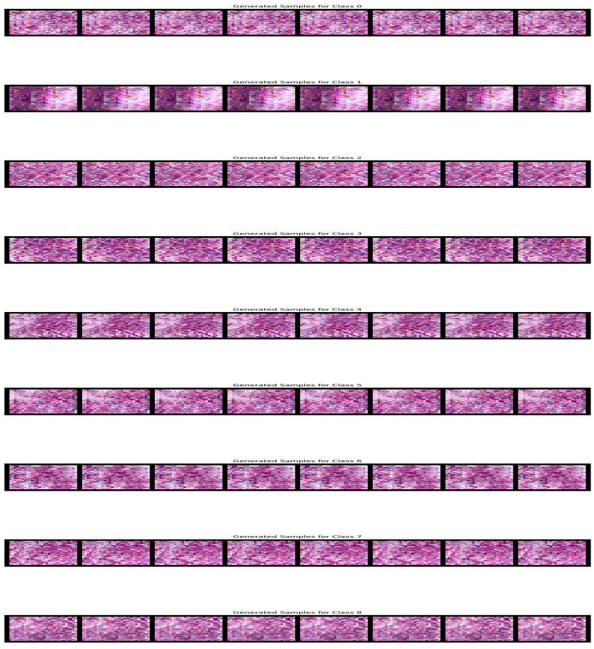

Fig 6.1: Class-wise generated samples (0–8)

7.5 Observations and Analysis

Visual Quality: All class samples appear with consistent purple-pink hues, matching histopathology colors. However, images are pixelated with minimal structural clarity or cell-level detail.

Conditional Control: Class-wise outputs exist, but there is **no visually clear separation** between classes — indicating **weak conditioning**.

Training Behavior:

High instability observed:

- Discriminator loss fluctuated wildly (e.g., 13+, 23+)
- Generator loss showed frequent spikes (mode collapse suspected)
- Very high or very low generator losses during epochs are common signs of adversarial imbalance.

7.6 Summary and Next Steps

The implementation of the conditional GAN demonstrated the potential for generating class-specific pathology images. While the model was able to produce outputs conditioned on class labels, visual analysis and training behavior highlighted several areas for improvement. These included signs of mode collapse, limited distinction between pathology classes, and insufficient structural detail in the generated samples.

To enhance the quality and reliability of class-conditional generation, several improvements can be explored in future work. These include the application of **label smoothing** to stabilize training, **spectral normalization** to regularize the discriminator, and **Conditional Batch Normalization (CBN)** in the generator to better incorporate label information. Additionally, more advanced conditioning techniques, such as **projection-based discriminators**, may improve the model's ability to utilize class labels effectively. The use of an **Auxiliary Classifier GAN (AC-GAN)** structure could also be considered to further strengthen class guidance during generation.

These enhancements provide a strong foundation for further experimentation and are expected to improve both the visual quality and class separability of the generated pathology images.

References:

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. *Generative adversarial nets*. arXiv preprint arXiv:1406.2661. Available at: https://arxiv.org/abs/1406.2661

Radford, A., Metz, L. and Chintala, S., 2016. *Unsupervised representation learning with deep convolutional generative adversarial networks*. arXiv preprint arXiv:1511.06434. Available at: https://arxiv.org/abs/1511.06434

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. *Dropout: A simple way to prevent neural networks from overfitting*. Journal of Machine Learning Research, 15(56), pp.1929–1958. Available at: https://jmlr.org/papers/v15/srivastava14a.html

Yang, J., Shi, R., Wei, D., Liu, Z., Wang, Z. and Zhou, Z.H., 2022. *MedMNIST v2: A large-scale lightweight benchmark for 2D and 3D biomedical image classification*. Scientific Data, 9(1), pp.1–13. https://doi.org/10.1038/s41597-022-01721-8

Mirza, M. and Osindero, S., 2014. *Conditional generative adversarial nets*. *arXiv preprint arXiv:1411.1784*. Available at: https://arxiv.org/abs/1411.1784