Machine Learning

LABORATORY: GAN In Class

NAME:	STUDENT ID#:

Objectives:

- Understand the training dynamics of a **Generative Adversarial Network (GAN).**
- Implement the adversarial loss used to train **GANs.**
- Train a GAN to generate handwritten digit images using the **MNIST** dataset.
- Apply **PyTorch** operations to manually train both a generator and a discriminator.
- Visualize generated results and observe how outputs evolve over training.

Instructions:

- In this assignment, you will complete the training loop for a **Generative Adversarial Network (GAN)** using the code template provided in class.
- Your task is to:
- Train a **generator** that learns to produce handwritten digit images from random noise.
- Train a **discriminator** that learns to distinguish between real MNIST images and generated (fake) ones.
- Implement the GAN loss using **binary cross-entropy** to update both models.
- Train using **PyTorch** — do not use higher-level wrappers (like nn.GAN libraries).
- Visualize the output to observe how GAN works. Please compare the input image, the fake image, and the generated image.

Code Template.

Step	Procedure

1	#Load Dataset
	import torch
	import torch.nn as nn
	import torch.optim as optim
	from torchvision import datasets, transforms
	from torch.utils.data import DataLoader
	import matplotlib.pyplot as plt
	import numpy as np
	import torchvision
	
	# Load MNIST
	transform = transforms.Compose([
	transforms.ToTensor(),
	transforms.Normalize((0.5,), (0.5,)),
	])
	dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)

Lecture: Prof. Hsien-I Lin
TA: Satrio Sanjaya and Muhammad Ahsan

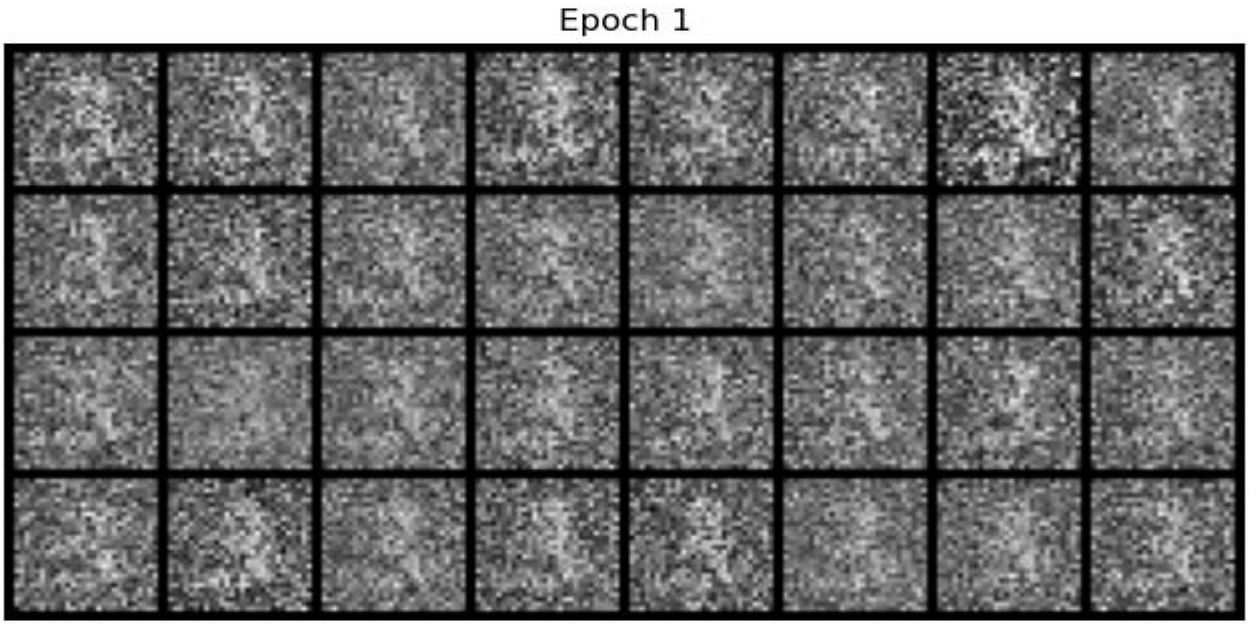
	loader = DataLoader(dataset, batch_size=64, shuffle=True)
	
	target_digit = 3 # you can change this based on your last digit student number
	
	# Reload MNIST and filter to only target_digit
	dataset = MNIST(root='./data', train=True, transform=transform, download=True)
	filtered_indices = [i for i, (_, label) in enumerate(dataset) if label == target_digit]
	filtered_dataset = Subset(dataset, filtered_indices[:5000]) # limit to 5000 samples
	
	loader = DataLoader(filtered_dataset, batch_size=64, shuffle=True)
	
	# Visualize filtered samples
	examples = next(iter(loader))[0][:32]
	examples = examples * 0.5 + 0.5
	grid = torchvision.utils.make_grid(examples, nrow=8, padding=2, normalize=True)
	plt.figure(figsize=(8, 8))
	plt.imshow(np.transpose(grid, (1, 2, 0)))
	plt.title(f"Real MNIST Digit '{target_digit}' Only (Before Training)")
	plt.axis("off")
	plt.show()

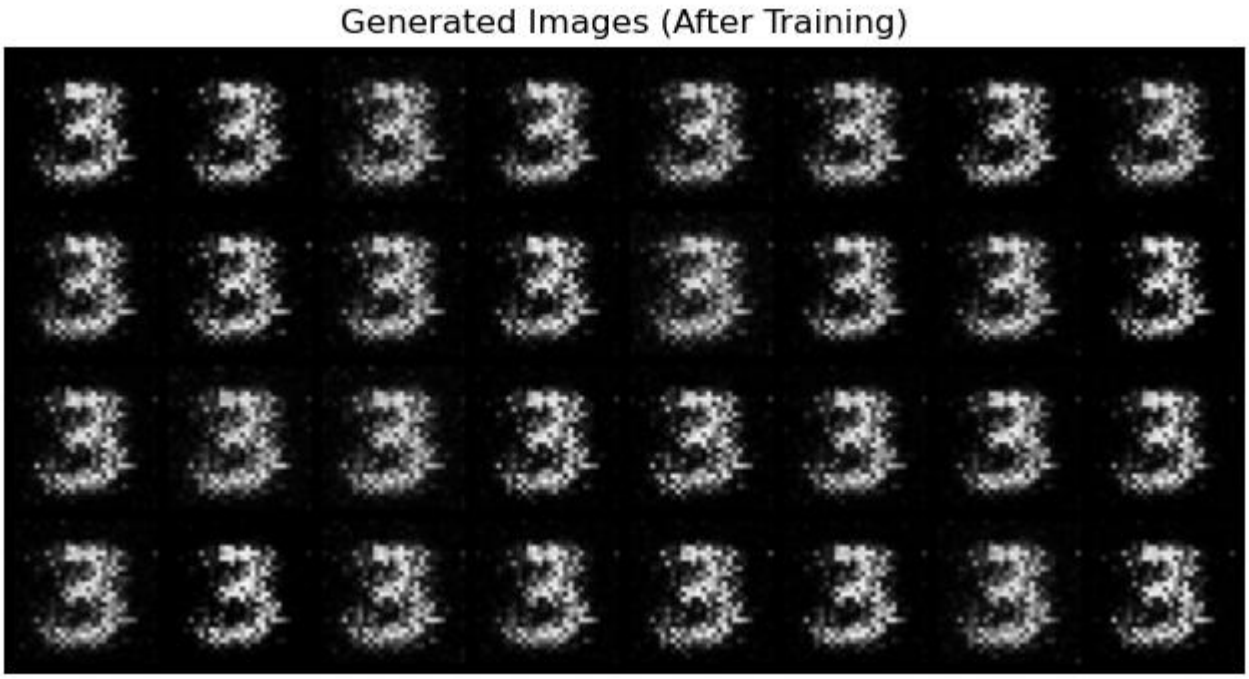
2	# ====== Generator and Discriminator Definitions ======
	# Define the Generator
	class Generator(nn.Module):
	  def __init__(self, z_dim=100, img_dim=784):
	  super().__init__()
	  self.gen = nn.Sequential(
	  # Define your generator architecture here
	  )
	
	  def forward(self, x):
	  return self.gen(x)
	
	# Define the Discriminator
	class Discriminator(nn.Module):
	  def __init__(self, img_dim=784):
	  super().__init__()
	  self.disc = nn.Sequential(
	  # Define your discriminator architecture here
	  )
	
	  def forward(self, x):
	  return self.disc(x)

3	# ========== Training Setup ==========
	# Initialize networks and optimizers, you can adjust the parameters
	z_dim = 100
	lr = 0.0002

	gen = Generator(z_dim) disc = Discriminator() criterion = nn.BCELoss() opt_gen = optim.Adam(gen.parameters(), lr) opt_disc = optim.Adam(disc.parameters(), lr)
4	# Write training loop with GAN adversarial loss here # TODO: implement training loop with real/fake labels, forward passes, and optim steps # Example training loop (pseudo-code): # 1. Loop over epochs and batches: # for epoch in range(num_epochs): # for batch_idx, (real, _) in enumerate(loader): # 2. Flatten and move real images to the device: # real = real.view(-1, 784).to(device) # batch_size = real.size(0) # 3. Generate fake images from noise: # noise = torch.randn(batch_size, z_dim).to(device) # fake = gen(noise) # 4. Train Discriminator # 5. Train Generator # 6. Print epoch loss values: # print(f"Epoch [{epoch+1}/{num_epochs}] Loss D: {loss_disc:.4f}, Loss G: {loss_gen:.4f}") # 7. Visualize generated samples after training

Example Output:





Grading Assignment & Submission (30% Max)

Implementation:

1. **(10%) GAN Training Loop Implementation**
   Implement the training loop using PyTorch:
   a. Generator: generates fake images from noise
   b. Discriminator: classifies real vs. fake images
   c. Loss: Binary Cross Entropy for both models
2. **(10%) Code Runs Without Errors + Successful Training Output**
   a. The model runs end-to-end without errors
   b. Trains using the MNIST dataset
   c. Displays GAN loss values per epoch
   d. Outputs generated samples as images
3. **(5%) Generated Style Focus: Use Last Digit of Student ID as Target Style**
   a. Train your GAN only on a **single digit** (e.g., if ID ends in 7 → use class '7')
   b. The generator should produce fake images that resemble that digit

Question:

1. **(5%) Briefly discuss your results.**

Submission:

1. Report: Answer all conceptual questions. Include screenshots of your results in the last pages of this PDF File.
2. Code: Submit your complete Python script in either .py or .ipynb format.
3. Upload both your report and code to the E3 system (**Lab8 In Class Assignment**). Name your files correctly:
   a. Report: StudentID_Lab8_InClass.pdf
   b. Code: StudentID_Lab8_ InClass.py or StudentID_Lab8_InClass.ipynb
4. Deadline: 16:20 PM

5. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.

Answer:

Using device: cuda

Epoch [1/100] Loss D: 0.3916, Loss G: 1.7560

Epoch [2/100] Loss D: 0.4097, Loss G: 2.3016

Epoch [3/100] Loss D: 0.8713, Loss G: 0.8978

Epoch [4/100] Loss D: 0.8476, Loss G: 1.1233

Epoch [5/100] Loss D: 0.9619, Loss G: 1.0258

Epoch [6/100] Loss D: 0.9699, Loss G: 0.9953

Epoch [7/100] Loss D: 1.0854, Loss G: 0.9541

Epoch [8/100] Loss D: 1.1460, Loss G: 0.9418

Epoch [9/100] Loss D: 1.1809, Loss G: 0.9173

Epoch [10/100] Loss D: 1.1853, Loss G: 0.9406

Epoch [11/100] Loss D: 1.2259, Loss G: 0.9021

Epoch [12/100] Loss D: 1.2095, Loss G: 0.9051

Epoch [13/100] Loss D: 1.2200, Loss G: 0.9092

Epoch [14/100] Loss D: 1.2387, Loss G: 0.9054

Epoch [15/100] Loss D: 1.2475, Loss G: 0.8743

Epoch [16/100] Loss D: 1.2561, Loss G: 0.8721

Epoch [17/100] Loss D: 1.2465, Loss G: 0.8741

Epoch [18/100] Loss D: 1.2632, Loss G: 0.8686

Epoch [19/100] Loss D: 1.2446, Loss G: 0.8886

Epoch [20/100] Loss D: 1.2423, Loss G: 0.9082

Epoch [21/100] Loss D: 1.2529, Loss G: 0.8987

Epoch [22/100] Loss D: 1.2581, Loss G: 0.8836

Epoch [23/100] Loss D: 1.2563, Loss G: 0.9018

Epoch [24/100] Loss D: 1.2735, Loss G: 0.8742

Epoch [25/100] Loss D: 1.2754, Loss G: 0.8855

Epoch [26/100] Loss D: 1.2751, Loss G: 0.8767

Epoch [27/100] Loss D: 1.2748, Loss G: 0.8747

Epoch [28/100] Loss D: 1.2801, Loss G: 0.8713

Epoch [29/100] Loss D: 1.2717, Loss G: 0.8824

Epoch [30/100] Loss D: 1.2787, Loss G: 0.8821

Epoch [31/100] Loss D: 1.2665, Loss G: 0.8896

Epoch [32/100] Loss D: 1.2775, Loss G: 0.8791

Epoch [33/100] Loss D: 1.2910, Loss G: 0.8801

Epoch [34/100] Loss D: 1.2817, Loss G: 0.8718

Epoch [35/100] Loss D: 1.2731, Loss G: 0.8857

Epoch [36/100] Loss D: 1.2794, Loss G: 0.8801

Epoch [37/100] Loss D: 1.2955, Loss G: 0.8694

Epoch [38/100] Loss D: 1.2974, Loss G: 0.8607

Epoch [39/100] Loss D: 1.3036, Loss G: 0.8657

Epoch [40/100] Loss D: 1.2968, Loss G: 0.8611

Epoch [41/100] Loss D: 1.3027, Loss G: 0.8533

Epoch [42/100] Loss D: 1.3036, Loss G: 0.8489

Epoch [43/100] Loss D: 1.2965, Loss G: 0.8553

Epoch [44/100] Loss D: 1.3038, Loss G: 0.8524

Epoch [45/100] Loss D: 1.3120, Loss G: 0.8561

Epoch [46/100] Loss D: 1.3123, Loss G: 0.8359

Epoch [47/100] Loss D: 1.3177, Loss G: 0.8320

Epoch [48/100] Loss D: 1.3047, Loss G: 0.8420

Epoch [49/100] Loss D: 1.3136, Loss G: 0.8490

Epoch [50/100] Loss D: 1.3104, Loss G: 0.8478

Epoch [51/100] Loss D: 1.3185, Loss G: 0.8375

Epoch [52/100] Loss D: 1.3164, Loss G: 0.8414

Epoch [53/100] Loss D: 1.3185, Loss G: 0.8340

Epoch [54/100] Loss D: 1.3162, Loss G: 0.8318

Epoch [55/100] Loss D: 1.3125, Loss G: 0.8478

Epoch [56/100] Loss D: 1.3124, Loss G: 0.8344

Epoch [57/100] Loss D: 1.3180, Loss G: 0.8370

Epoch [58/100] Loss D: 1.3241, Loss G: 0.8214

Epoch [59/100] Loss D: 1.3218, Loss G: 0.8311

Epoch [60/100] Loss D: 1.3118, Loss G: 0.8312

Epoch [61/100] Loss D: 1.3107, Loss G: 0.8420

Epoch [62/100] Loss D: 1.3154, Loss G: 0.8443

Epoch [63/100] Loss D: 1.3162, Loss G: 0.8449

Epoch [64/100] Loss D: 1.3062, Loss G: 0.8401

Epoch [65/100] Loss D: 1.3167, Loss G: 0.8453

Epoch [66/100] Loss D: 1.3114, Loss G: 0.8361

Epoch [67/100] Loss D: 1.3193, Loss G: 0.8422

Epoch [68/100] Loss D: 1.3137, Loss G: 0.8388

Epoch [69/100] Loss D: 1.3085, Loss G: 0.8434

Epoch [70/100] Loss D: 1.3088, Loss G: 0.8504

Epoch [71/100] Loss D: 1.3110, Loss G: 0.8511

Epoch [72/100] Loss D: 1.3133, Loss G: 0.8459

Epoch [73/100] Loss D: 1.3086, Loss G: 0.8473

Epoch [74/100] Loss D: 1.3135, Loss G: 0.8567

Epoch [75/100] Loss D: 1.3021, Loss G: 0.8471

Epoch [76/100] Loss D: 1.3035, Loss G: 0.8631

Epoch [77/100] Loss D: 1.3049, Loss G: 0.8528

Epoch [78/100] Loss D: 1.3038, Loss G: 0.8540

Epoch [79/100] Loss D: 1.3050, Loss G: 0.8490

Epoch [80/100] Loss D: 1.2990, Loss G: 0.8658

Epoch [81/100] Loss D: 1.3056, Loss G: 0.8578

Epoch [82/100] Loss D: 1.3067, Loss G: 0.8594

Epoch [83/100] Loss D: 1.3036, Loss G: 0.8608

Epoch [84/100] Loss D: 1.2969, Loss G: 0.8652

Epoch [85/100] Loss D: 1.2957, Loss G: 0.8702

Epoch [86/100] Loss D: 1.2951, Loss G: 0.8700

Epoch [87/100] Loss D: 1.3060, Loss G: 0.8556

Epoch [88/100] Loss D: 1.2944, Loss G: 0.8737

Epoch [89/100] Loss D: 1.2923, Loss G: 0.8771

Epoch [90/100] Loss D: 1.2994, Loss G: 0.8691

Epoch [91/100] Loss D: 1.2849, Loss G: 0.8813

Epoch [92/100] Loss D: 1.2833, Loss G: 0.8934

Epoch [93/100] Loss D: 1.2843, Loss G: 0.8972

Epoch [94/100] Loss D: 1.2762, Loss G: 0.8909

Epoch [95/100] Loss D: 1.2828, Loss G: 0.9020

Epoch [96/100] Loss D: 1.2790, Loss G: 0.9062

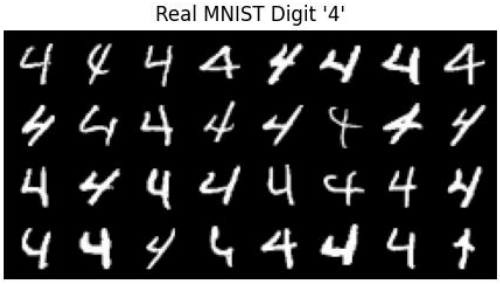
Epoch [97/100] Loss D: 1.2651, Loss G: 0.9154

Epoch [98/100] Loss D: 1.2766, Loss G: 0.9100

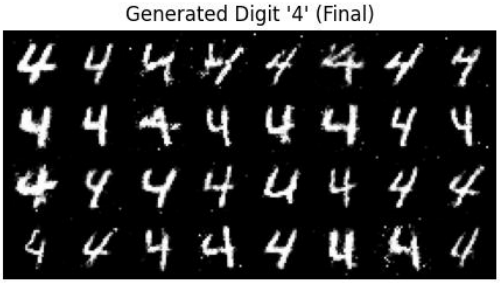
Epoch [99/100] Loss D: 1.2697, Loss G: 0.9028

Epoch [100/100] Loss D: 1.2819, Loss G: 0.9020

Real MNIST Digit '4'

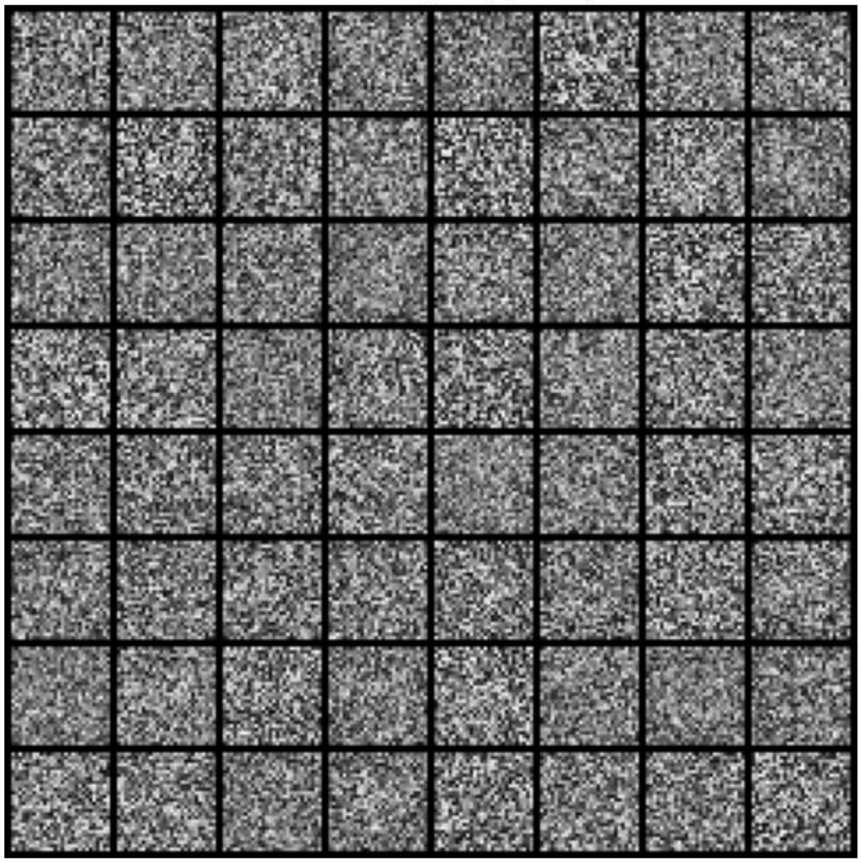


Generated Digit '4' (Final)

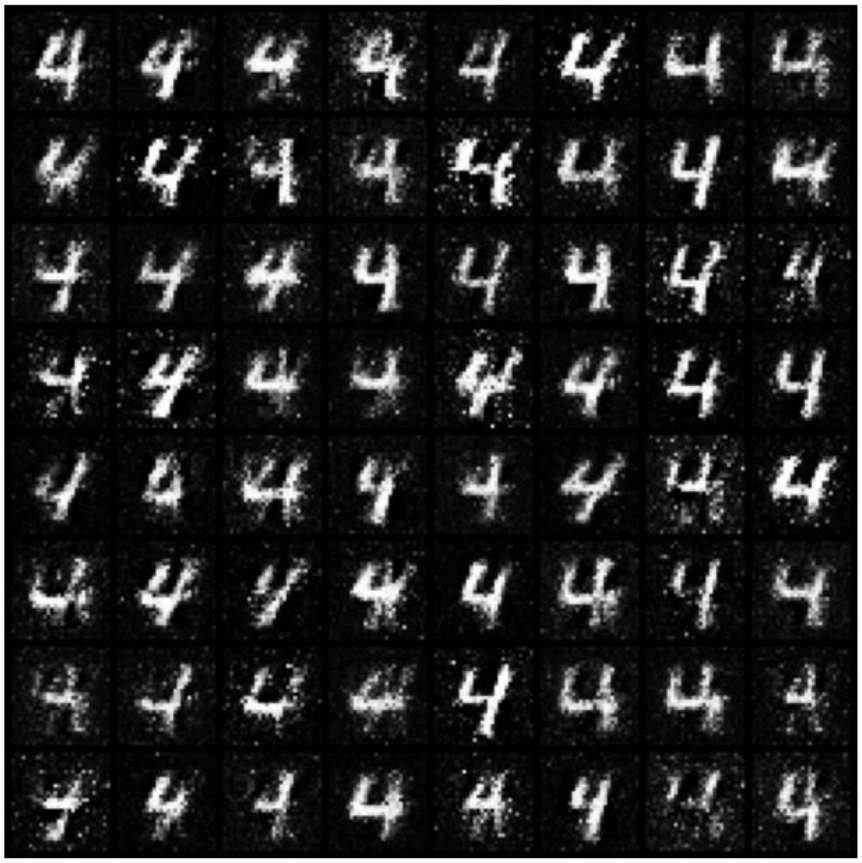


Training complete! Results saved to 'gan_results' directory.

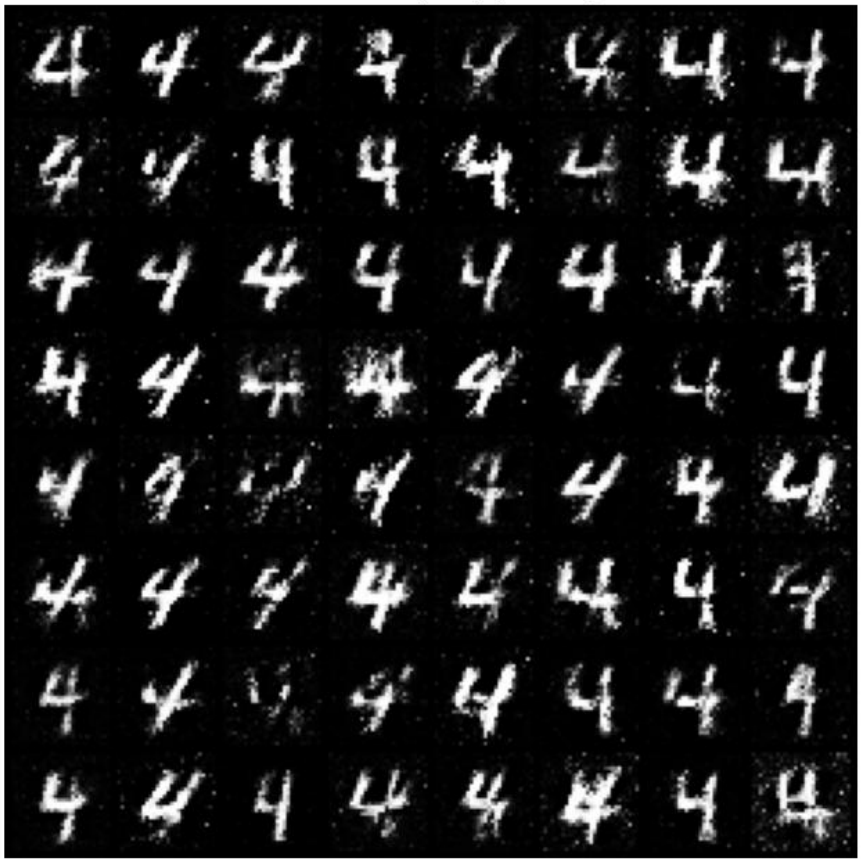
Generated Digit 4 (Epoch 1)



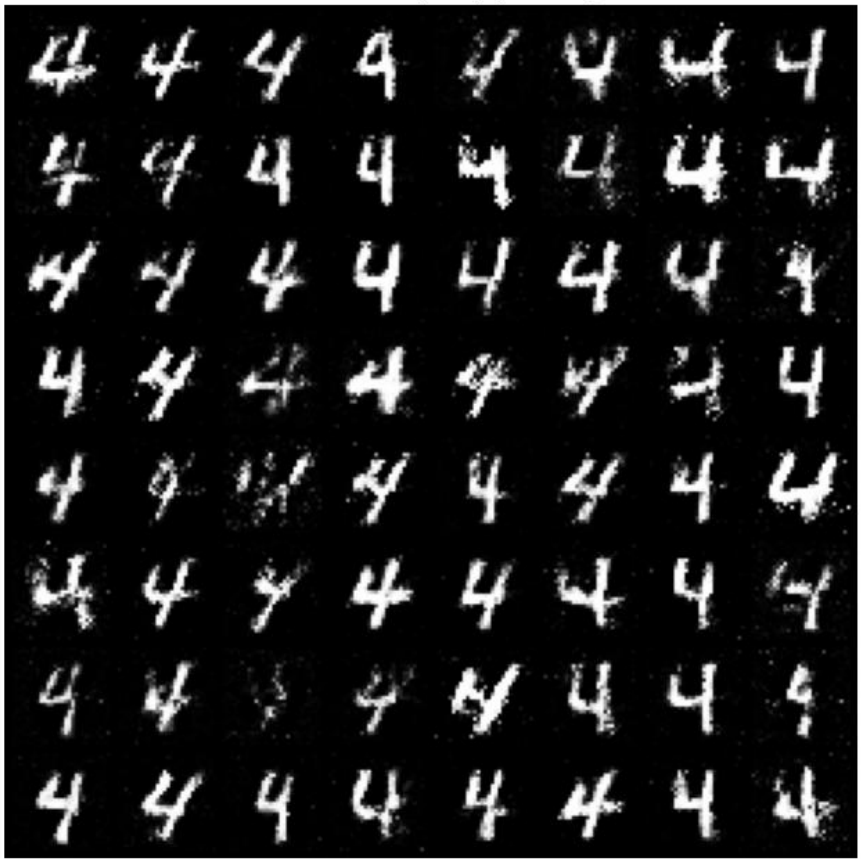
Generated Digit 4 (Epoch 10)



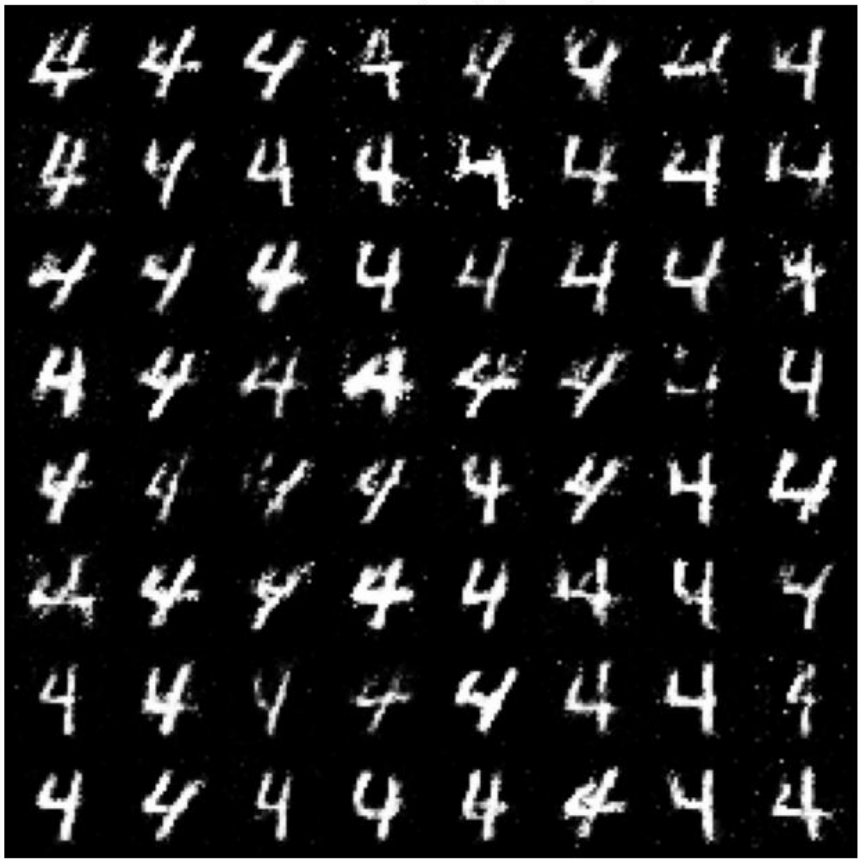
Generated Digit 4 (Epoch 20)



Generated Digit 4 (Epoch 30)



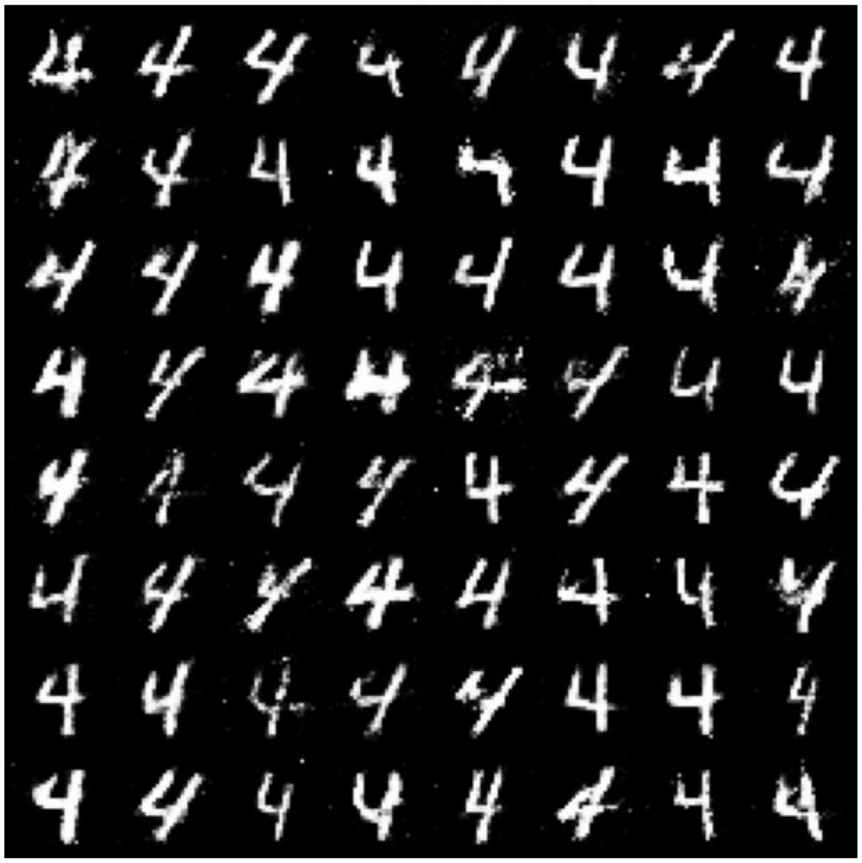
Generated Digit 4 (Epoch 40)



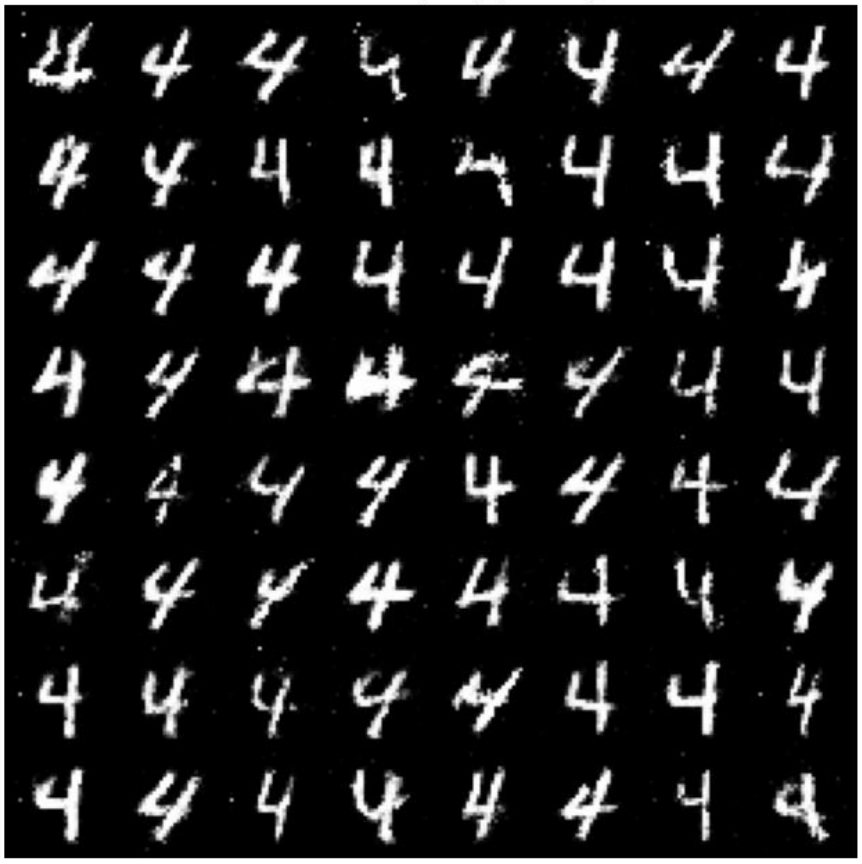
Generated Digit 4 (Epoch 50)

Generated Digit 4 (Epoch 60)

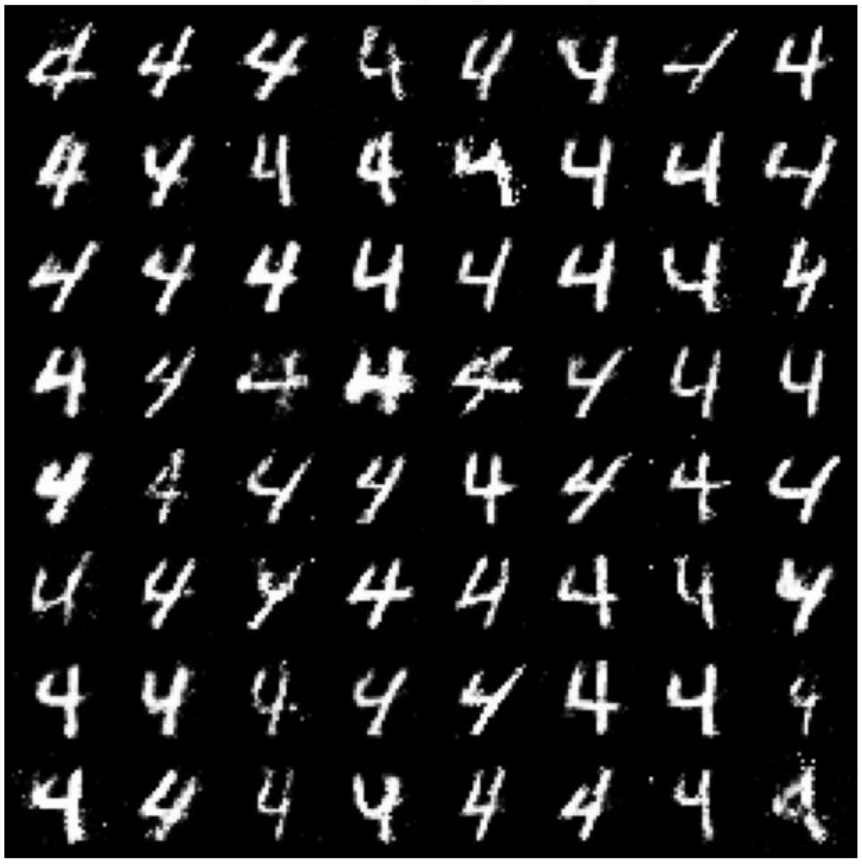
Generated Digit 4 (Epoch 70)

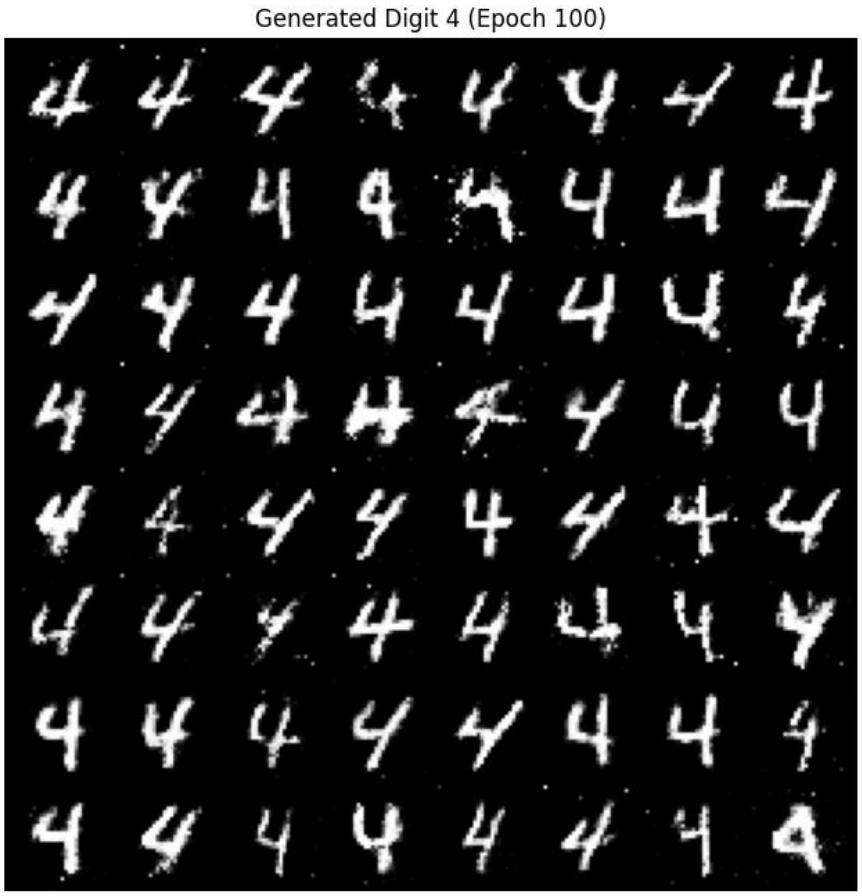


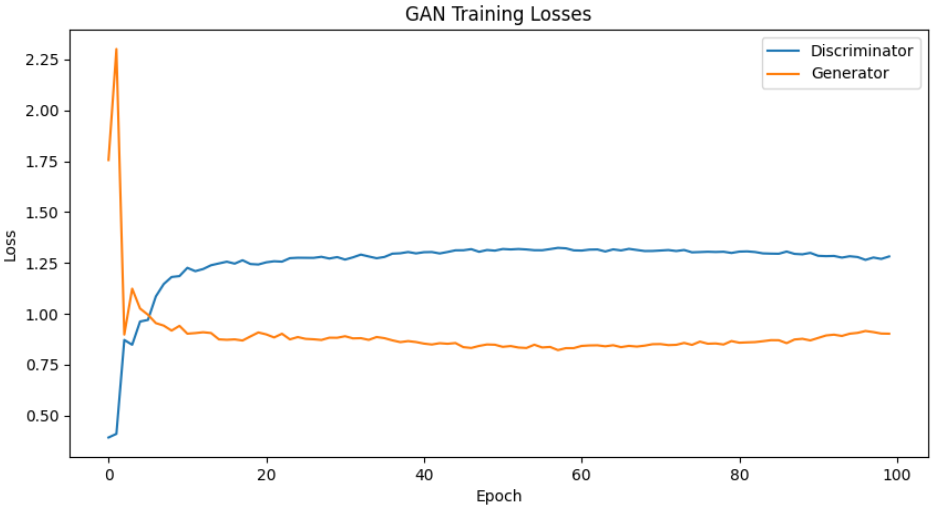
Generated Digit 4 (Epoch 80)



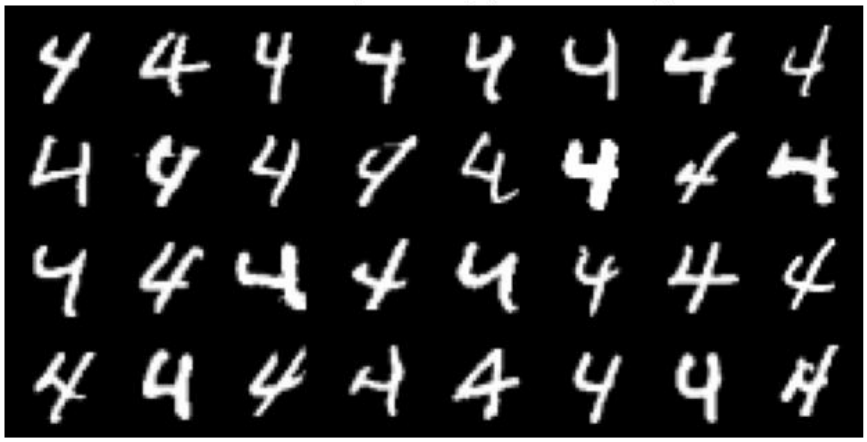
Generated Digit 4 (Epoch 90)



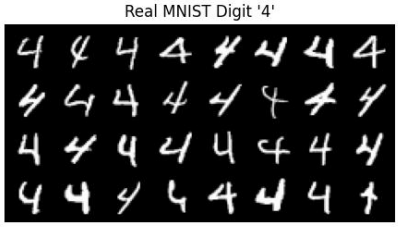

Generated Digit 4 (Epoch 100)


GAN Training Losses
Discriminator
Generator
Loss
Epoch
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0
20
40
60
80
100

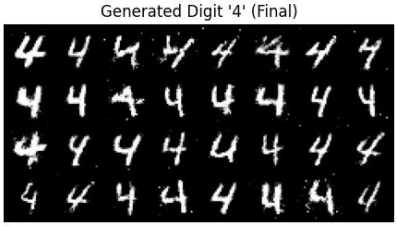
Real MNIST Digit '4' Only (Before Training)



Real MNIST Digit '4'



Generated Digit '4' (Final)



Question:

1. **Briefly discuss your results.**

Phase	What the samples look like	What the losses say	Interpretation
Epoch 1	Pure white-noise tiles	D ≈ 0.4, G ≈ 1.8	Discriminator still "blind," generator outputs noise.
Epoch 10	Clearly recognizable 4's, but dotted noise and broken strokes	D jumps to ≈ 1.18, G drops to ≈ 0.94	Generator has learned the coarse shape; discriminator starts to apply pressure.
Epoch 10 → 60	The background gets darker, strokes more continuous, progress slows	D hovers 1.25–1.30, G 0.83–0.88 (flat)	Classic plateau: the two networks reach a sub-optimal equilibrium.
Epoch 60 → 100	Quality more or less frozen; occasionally over-exposed strokes	D and G oscillate slightly around the same mean	Model is stuck in a "good-but-not-great" local optimum.

Real vs. generated (final epoch)

- **Shape accuracy** – Most generated digits have the correct vertical bar and diagonal bar, so a simple classifier would label them "4".
- **Texture differences** –
    - Background pepper noise that real MNIST images don't have.
    - Jagged, uneven stroke width; real digits are smoother.
- **Diversity** – Not a full mode collapse, but variation is limited (same slant angle, similar aspect ratio).

Loss curve

- Very sharp cross-over in the first few epochs (generator loss plunges, discriminator loss spikes) – expected in DCGAN warm-up.

- After ~10 epochs both losses flatten; the long horizontal stretch means extra epochs alone will not bring big gains.

Two-line takeaway

1. **Goal met**: after 100 epochs the DCGAN reliably produces recognizable digit "4" samples.
2. **Headroom left**: to remove noise and widen style diversity you will need training-stability tricks (e.g., WGAN-GP, spectral norm, label-smoothing, lower LR + EMA) rather than just "train longer."