```
import warnings
```

```
warnings.filterwarnings("ignore")
```

```
Start coding or generate with AI.
```

```
!pip install dash --upgrade
```

```
Collecting dash
  Downloading dash-2.18.2-py3-none-any.whl.metadata (10 kB)
Collecting Flask<3.1,>=1.0.4 (from dash)
  Downloading flask-3.0.3-py3-none-any.whl.metadata (3.2 kB)
Collecting Werkzeug<3.1 (from dash)
  Downloading werkzeug-3.0.6-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.11/dist-packages
Collecting dash-html-components==2.0.0 (from dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl.metadata (3.8 kB)
Collecting dash-core-components==2.0.0 (from dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl.metadata (2.9 kB)
Collecting dash-table==5.0.0 (from dash)
  Downloading dash_table-5.0.0-py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from
Collecting retrying (from dash)
  Downloading retrying-1.3.4-py3-none-any.whl.metadata (6.9 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: itsdangerous>=2.1.2 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: blinker>=1.6.2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.11/dist-packages (fr
Downloading dash-2.18.2-py3-none-any.whl (7.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 84.0 MB/s eta 0:00:00
Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Downloading flask-3.0.3-py3-none-any.whl (101 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.7/101.7 kB 7.7 MB/s eta 0:00:00
Downloading werkzeug-3.0.6-py3-none-any.whl (227 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 228.0/228.0 kB 21.4 MB/s eta 0:00:00
Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Installing collected packages: dash-table, dash-html-components, dash-core-components, W
```

```
  Attempting uninstall: Werkzeug
    Found existing installation: Werkzeug 3.1.3
    Uninstalling Werkzeug-3.1.3:
      Successfully uninstalled Werkzeug-3.1.3
  Attempting uninstall: Flask
    Found existing installation: Flask 3.1.0
    Uninstalling Flask-3.1.0:
      Successfully uninstalled Flask-3.1.0
Successfully installed Flask-3.0.3 Werkzeug-3.0.6 dash-2.18.2 dash-core-components-2.0.0
```

Start coding or generate with AI.

```
!pip install dash-daq
```

```
Collecting dash-daq
  Downloading dash_daq-0.5.0.tar.gz (642 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 642.7/642.7 kB 35.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: dash>=1.6.1 in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.11/
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.11/
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: retrying in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: itsdangerous>=2.1.2 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: blinker>=1.6.2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (frc
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.11/dist-packages (fr
Building wheels for collected packages: dash-daq
  Building wheel for dash-daq (setup.py) ... done
  Created wheel for dash-daq: filename=dash_daq-0.5.0-py3-none-any.whl size=669692 sha25
  Stored in directory: /root/.cache/pip/wheels/9a/e1/a3/ef7c3fa914e4df214fdcb64529c44669
Successfully built dash-daq
Installing collected packages: dash-daq
Successfully installed dash-daq-0.5.0
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
#  Imports and Setup
```

```
import torch
```

```
import random
```

```
import pickle
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import os
```

```
from torchvision import datasets, transforms
```

```
from torch.utils.data import DataLoader, TensorDataset
```

```
import torch.nn as nn
```

```
import torch.optim as optim
```

```
import torch.nn.functional as F
```

```
import dash
```

```
from dash import dcc, html, Input, Output
```

```
import plotly.express as px
```

```
import dash_daq as daq
```

```
import pandas as pd
```

```
import base64
```

```
from io import BytesIO
```

```
from tensorflow.keras.datasets import mnist
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
#  Data loading and Preprocessing
```

```
# Load MNIST dataset
```

```
transform = transforms.ToTensor()
```

```
mnist_train = datasets.MNIST(root="./data", train=True, download=True, transform=transform)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to
100%|██████████| 9.91M/9.91M [00:10<00:00, 903kB/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to
100%|██████████| 28.9k/28.9k [00:00<00:00, 134kB/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to .
100%|██████████| 1.65M/1.65M [00:06<00:00, 244kB/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw
```

```
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to .
100%|██████████| 4.54k/4.54k [00:00<00:00, 11.7MB/s]Extracting ./data/MNIST/raw/t10k-lab
```

```
Start coding or generate with AI.
```

```
# Define labeled data size (10% labeled, 90% unlabeled)
```

```
labeled_ratio = 0.1
```

```
num_labeled = int(labeled_ratio * len(mnist_train))
```

```
Start coding or generate with AI.
```

```
# Shuffle dataset indices# Shuffle dataset indices
```

```
indices = list(range(len(mnist_train)))
```

```
random.shuffle(indices)
```

```
Start coding or generate with AI.
```

```
# Split into labeled and unlabeled datasets
```

```
labeled_indices = indices[:num_labeled]
```

```
unlabeled_indices = indices[num_labeled:]
```

```
Start coding or generate with AI.
```

```
# Extract labeled and unlabeled data
```

```

```

```
labeled_data = [(mnist_train[i][0], mnist_train[i][1]) for i in labeled_indices]
```

```
unlabeled_data = [mnist_train[i][0] for i in unlabeled_indices]  # Remove labels
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
# Save labeled & unlabeled data
```

```
with open("labeled_data.pkl", "wb") as f:
    pickle.dump(labeled_data, f)
```

```
Start coding or generate with AI.
```

```
with open("unlabeled_data.pkl", "wb") as f:
    pickle.dump(unlabeled_data, f)
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
# Convert to tensors
```

```
labeled_images = torch.stack([img for img, _ in labeled_data])
```

```
labeled_labels = torch.tensor([label for _, label in labeled_data])
```

```
unlabeled_images = torch.stack(unlabeled_data)
```

```
Start coding or generate with AI.
```

```
# Save cleaned & processed data
```

```
torch.save((labeled_images, labeled_labels), "labeled_mnist.pt")
```

```
torch.save(unlabeled_images, "unlabeled_mnist.pt")
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
# Data Validation
```

```
# Check dataset size
```

```
print(f"Total samples: {len(mnist_train)}")
```

```
Total samples: 60000
```

```
Start coding or generate with AI.
```

```
# Check shape of images
```

```
sample_img, sample_label = mnist_train[0]
```

```
print(f"Image Shape: {sample_img.shape} (Should be 1x28x28)")
```

```
Image Shape: torch.Size([1, 28, 28]) (Should be 1x28x28)
```

```
Start coding or generate with AI.
```

```
# Check data type
```

```
print(f"Data Type: {sample_img.dtype}")
```

```
Data Type: torch.float32
```

```
Start coding or generate with AI.
```

```
# Check pixel value range
```

```

```

```
print(f"Min Pixel Value: {sample_img.min()}, Max Pixel Value: {sample_img.max()}")
```

```
Min Pixel Value: 0.0, Max Pixel Value: 1.0
```

Start coding or generate with AI.

```
# Check for missing or corrupt images
```

```
corrupt_count = sum(1 for img, _ in mnist_train if img is None)
```

```
print(f"Corrupt Samples: {corrupt_count}")
```

```
Corrupt Samples: 0
```

Start coding or generate with AI.

```
# Check class distribution in labeled data
```

```
label_counts = {i: 0 for i in range(10)}
for _, label in mnist_train:
    label_counts[label] += 1
```

Start coding or generate with AI.

```
print("Label Distribution:")
for digit, count in label_counts.items():
    print(f"Digit {digit}: {count} samples")
```

```
Label Distribution:
Digit 0: 5923 samples
Digit 1: 6742 samples
Digit 2: 5958 samples
Digit 3: 6131 samples
Digit 4: 5842 samples
Digit 5: 5421 samples
Digit 6: 5918 samples
Digit 7: 6265 samples
Digit 8: 5851 samples
Digit 9: 5949 samples
```

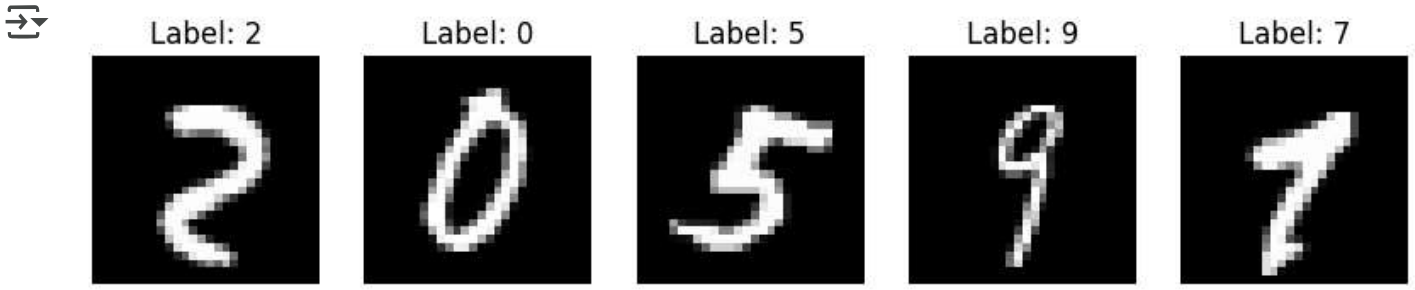
Start coding or generate with AI.

Start coding or generate with AI.

```
# Visualize random samples
```

```
fig, axes = plt.subplots(1, 5, figsize=(10, 2))
for i, ax in enumerate(axes):
    img, label = mnist_train[random.randint(0, len(mnist_train) - 1)]
    ax.imshow(img.squeeze(), cmap="gray")
    ax.set_title(f"Label: {label}")
    ax.axis("off")

plt.show()
```




```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

## MAIN MODEL

```
# # Load processed datasets
```

```
labeled_images, labeled_labels = torch.load("/content/labeled_mnist.pt")
```

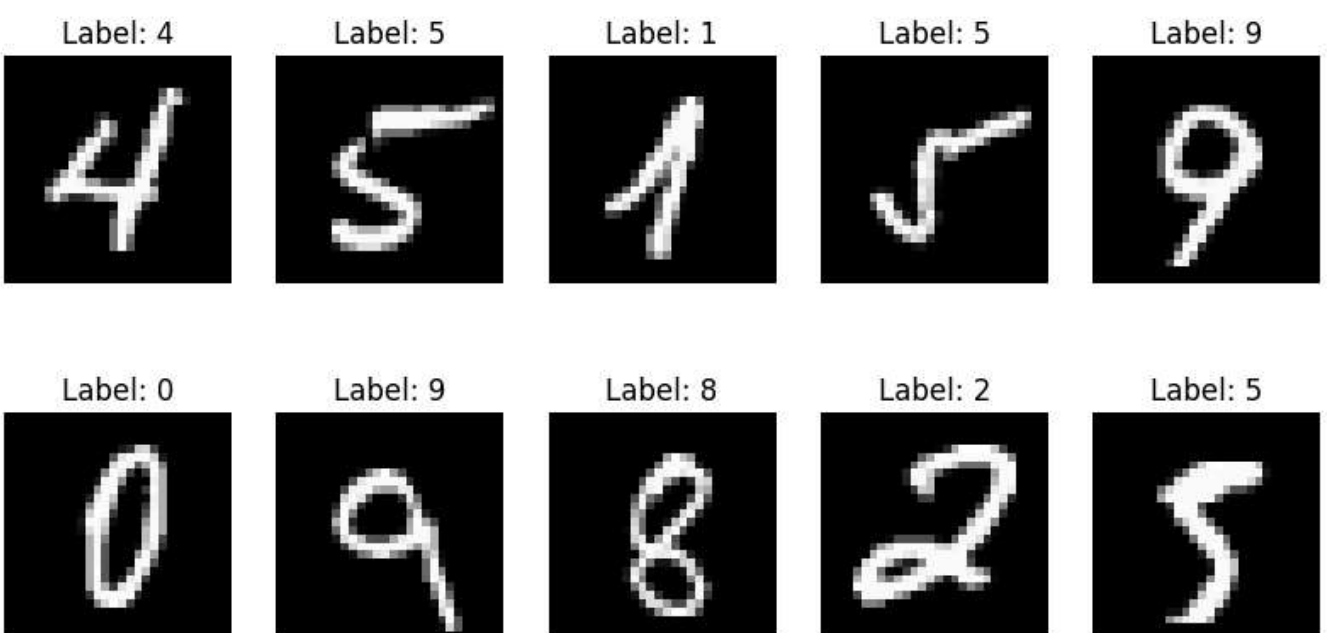
```
unlabeled_images = torch.load("/content/unlabeled_mnist.pt")
```

```
Start coding or generate with AI.
```

```
# Visualize some labeled samples
```

```
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(labeled_images[i].squeeze(), cmap="gray")
    ax.set_title(f"Label: {labeled_labels[i].item()}")
```

```
    ax.axis("off")
plt.show()
```




```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
# Ensure batch sizes match
```

```
batch_size = 64
```

```
labeled_loader = DataLoader(TensorDataset(labeled_images, labeled_labels), batch_size=batch_
```

```
unlabeled_loader = DataLoader(TensorDataset(unlabeled_images), batch_size=batch_size, shuffl
```

```
Start coding or generate with AI.
```

```
# Define Generator
```

```
class Generator(nn.Module):
    def __init__(self, latent_dim=100):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(latent_dim, 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)
```

```
Start coding or generate with AI.
```

```
# Define Discriminator
```

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.feature_extractor = nn.Sequential(
            nn.Linear(28 * 28, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU()
        )
        self.classifier = nn.Linear(256, 11)  # 10 classes + 1 for fake data

    def forward(self, x):
        features = self.feature_extractor(x.view(-1, 28 * 28))
        return self.classifier(features)
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

## Initialize models

```
generator = Generator()
```

```
discriminator = Discriminator()
```

```
Start coding or generate with AI.
```

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

```
generator.to(device)
```

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=784, bias=True)
    (5): Tanh()
  )
)
```

```
discriminator.to(device)
```

```
Discriminator(
  (feature_extractor): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): ReLU()
  )
  (classifier): Linear(in_features=256, out_features=11, bias=True)
)
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
# Optimizers
```

```
g_optimizer = optim.Adam(generator.parameters(), lr=0.0002, betas=(0.5, 0.999))
```

```
d_optimizer = optim.Adam(discriminator.parameters(), lr=0.0002, betas=(0.5, 0.999))
```

```
Start coding or generate with AI.
```

```
# Loss function
```

```
gan_loss = nn.CrossEntropyLoss()
```

```
Start coding or generate with AI.
```

```
# Create results directory
```

```
os.makedirs("results", exist_ok=True)
```

```
def save_generated_images(epoch):
    """Generate and save a batch of fake images for visualization."""
    generator.eval()
    with torch.no_grad():
        z = torch.randn(16, 100, device=device)
        fake_images = generator(z).cpu()

    fig, axes = plt.subplots(4, 4, figsize=(4, 4))
    for i, ax in enumerate(axes.flat):
        ax.imshow(fake_images[i].squeeze(), cmap="gray")
        ax.axis("off")

    plt.savefig(f"results/generated_epoch_{epoch}.png")
    plt.close()
    generator.train()
```

```
def train_sgan(epochs=10):
    generator.train()
    discriminator.train()

    for epoch in range(epochs):
        for (real_images, real_labels), (unlabeled_images,) in zip(labeled_loader, unlabelec
            real_images, real_labels = real_images.to(device), real_labels.to(device)
            unlabeled_images = unlabeled_images.to(device)
            batch_size = real_images.size(0)

            # Generate fake images
            z = torch.randn(batch_size, 100, device=device)
            fake_images = generator(z)

            # Train Discriminator
            d_optimizer.zero_grad()

            real_preds = discriminator(real_images)
            fake_preds = discriminator(fake_images.detach())
            unlabeled_preds = discriminator(unlabeled_images)

            # Compute discriminator losses
            real_loss = gan_loss(real_preds[:, :-1], F.one_hot(real_labels, num_classes=10).
            fake_loss = gan_loss(fake_preds[:, -1].unsqueeze(1), torch.zeros((batch_size, 1)
            unlabeled_loss = gan_loss(unlabeled_preds[:, -1].unsqueeze(1), torch.zeros((batc

            d_loss = real_loss + fake_loss + unlabeled_loss
            d_loss.backward()
```

```
            d_optimizer.step()

            # Train Generator
            g_optimizer.zero_grad()
            fake_preds = discriminator(fake_images)

            # Instead of training the generator to minimize "fake class,"
            # we push it towards being classified as real classes
            g_loss = gan_loss(fake_preds[:, :-1], torch.ones_like(fake_preds[:, :-1]) / 10)
            g_loss.backward()
            g_optimizer.step()

        print(f"Epoch {epoch+1}/{epochs}, D Loss: {d_loss.item():.4f}, G Loss: {g_loss.item(
        save_generated_images(epoch+1)

    # Save models
    torch.save(generator.state_dict(), "results/generator.pth")
    torch.save(discriminator.state_dict(), "results/discriminator.pth")
    print("Training complete! Models saved.")
```

```
Start coding or generate with AI.
```

```
# Start Training
```

```
train_sgan(epochs=10)
```

```
Epoch 1/10, D Loss: 0.6812, G Loss: 2.3026
Epoch 2/10, D Loss: 0.3689, G Loss: 2.3026
Epoch 3/10, D Loss: 0.3223, G Loss: 2.3026
Epoch 4/10, D Loss: 0.3476, G Loss: 2.3026
Epoch 5/10, D Loss: 0.3572, G Loss: 2.3026
Epoch 6/10, D Loss: 0.2376, G Loss: 2.3026
Epoch 7/10, D Loss: 0.2452, G Loss: 2.3026
Epoch 8/10, D Loss: 0.1042, G Loss: 2.3026
Epoch 9/10, D Loss: 0.1540, G Loss: 2.3026
Epoch 10/10, D Loss: 0.2011, G Loss: 2.3026
Training complete! Models saved.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

## Evalution of the model

```
# Load trained models
```

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

```
Start coding or generate with AI.
```

```
generator = torch.load("results/generator.pth", map_location=device)
```

```
discriminator = torch.load("results/discriminator.pth", map_location=device)
```

```
Start coding or generate with AI.
```

```
# Initialize models before loading state_dict
```

```
generator_model = Generator().to(device)
```

```
discriminator_model = Discriminator().to(device)
```

```
Start coding or generate with AI.
```

```
generator_model.load_state_dict(torch.load("results/generator.pth", map_location=
```

```
<All keys matched successfully>
```

```
discriminator_model.load_state_dict(torch.load("results/discriminator.pth", map_location=dev
```

```
<All keys matched successfully>
```

```
Start coding or generate with AI.
```

```
generator_model.eval()
```

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=784, bias=True)
    (5): Tanh()
```

```
    )
  )
```

```
discriminator_model.eval()
```

```
Discriminator(
  (feature_extractor): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): ReLU()
  )
  (classifier): Linear(in_features=256, out_features=11, bias=True)
)
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
# Generate synthetic images
```

```
z = torch.randn(16, 100, device=device)
```

```
fake_images = generator_model(z).cpu().detach()
```

```
Start coding or generate with AI.
```

```
# Load real images
```

```
labeled_images, labeled_labels = torch.load("/content/labeled_mnist.pt")
```

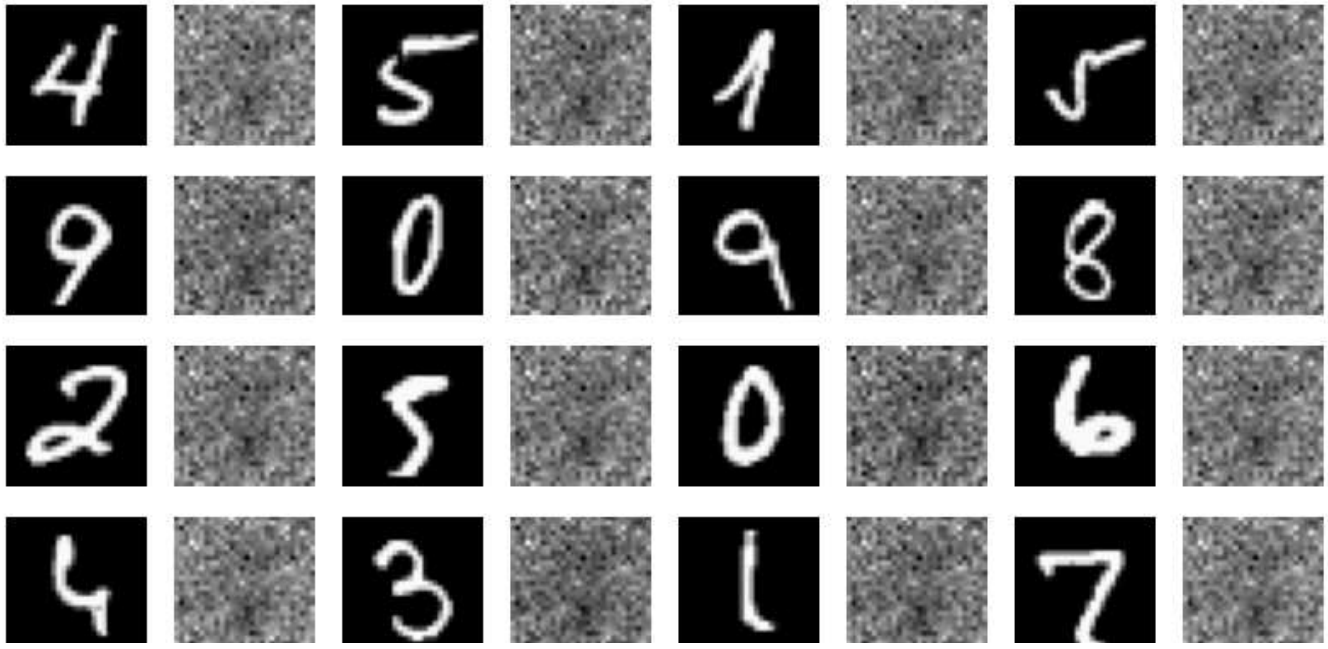
```
real_images = labeled_images[:16].cpu().detach()
```

```
Start coding or generate with AI.
```

```
# Compare real and generated images
```

```
fig, axes = plt.subplots(4, 8, figsize=(10, 5))
for i in range(16):
    axes[i//4, 2*(i%4)].imshow(real_images[i].squeeze(), cmap='gray')
    axes[i//4, 2*(i%4)].axis('off')
    axes[i//4, 2*(i%4)+1].imshow(fake_images[i].squeeze(), cmap='gray')
    axes[i//4, 2*(i%4)+1].axis('off')
```

```
plt.suptitle("Left: Real | Right: Generated")
plt.show()
```




Start coding or generate with AI.

Start coding or generate with AI.

```
# Evaluate Discriminator
```

```
real_preds = discriminator_model(real_images.to(device))
```

```
fake_preds = discriminator_model(fake_images.to(device))
```

Start coding or generate with AI.

```
print("Real Image Predictions:\n", real_preds.softmax(dim=1)[:, :-1].mean(dim=0))
print("\nFake Image Predictions:\n", fake_preds.softmax(dim=1)[:, -1].mean().item())
```

```
Real Image Predictions:
 tensor([0.1197, 0.1031, 0.0685, 0.0658, 0.1015, 0.2012, 0.0657, 0.0654, 0.0657,
        0.1248], device='cuda:0', grad_fn=<MeanBackward1>)

Fake Image Predictions:
 0.10253884643316269
```

```
Start coding or generate with AI.
```

```
# Save generated images and evaluation results
```

```
torch.save(fake_images, "generated_images.pth")
```

```
results = {"real_preds": real_preds.cpu().detach(), "fake_preds": fake_preds.cpu().detach()}
```

```
with open("discriminator_results.pkl", "wb") as f:
    pickle.dump(results, f)
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

## Dash App for Visualization

```
# Load MNIST dataset
```

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```
Start coding or generate with AI.
```

```
def get_sample_image(index):
    """Convert a sample image to base64 for displaying in Dash."""
    fig, ax = plt.subplots()
    ax.imshow(train_images[index], cmap='gray')
    ax.axis('off')
    buf = BytesIO()
    plt.savefig(buf, format='png')
    buf.seek(0)
    encoded_image = base64.b64encode(buf.getvalue()).decode('utf-8')
```

```
    plt.close()
    return f'{encoded_image}'
```

```
Start coding or generate with AI.
```

```
# Generate a DataFrame for visualization
```

```
labels_df = pd.DataFrame({'Label': train_labels})
```

```
Start coding or generate with AI.
```

```
# Initialize Dash app
```

```
app = dash.Dash(__name__)
```

```
Start coding or generate with AI.
```

```
app.layout = html.Div([
    html.H1("MNIST Dataset Dashboard"),

    html.Div([
        html.Label("Select Sample Image Index:"),
        dcc.Slider(0, len(train_images) - 1, step=1, value=random.randint(0, len(
                   id='image-slider')
    ]),

    html.Div([html.Img(id='image-display', style={'width': '200px', 'height': '20

    html.H3("Label Distribution"),
    dcc.Graph(id='label-distribution'),

    html.Label("Select Number of GAN-Generated Images:"),
    dcc.Slider(1, 10, step=1, value=5, id='gan-slider'),

    html.Div(id='gan-images', style={'display': 'flex', 'flex-wrap': 'wrap'})
])

@app.callback(
    Output('image-display', 'src'),
    Input('image-slider', 'value')
)
def update_image(index):
    return get_sample_image(index)

@app.callback(
    Output('label-distribution', 'figure'),
```

```
    Input('image-slider', 'value')
)
def update_distribution(_):
    fig = px.histogram(labels_df, x='Label', nbins=10, title='Label Distribution'
    return fig

@app.callback(
    Output('gan-images', 'children'),
    Input('gan-slider', 'value')
)
def generate_gan_images(n):
    return [html.Img(src=get_sample_image(random.randint(0, len(train_images) - 1
                     style={'width': '100px', 'height': '100px', 'margin': '5px'}
            for _ in range(n)]
```

Start coding or generate with AI.

Start coding or generate with AI.

```
if __name__ == '__main__':
    app.run_server(debug=True)
```