Dataset Augmentation using Generative Adversarial Networks (G.A.N)

Farees Siddiqui [100-780-513], Conor Bradley [100-559-961], Nicholas Kissoon [100-742-790]

For this project we decided to tackle the problem of dataset augmentation, Very often when trying to construct a machine learning model we simply do not have enough training data. This problem can be solved with the help of a Generative Adverserial Network.

As the name implies this is a generative model, for our example we will be using this technique to generate new data for the MNIST dataset.

This model works similar to the Encoder Decoder model that we discussed in class, but instead of encoding some data to a lower dimension, this model is used to generate completely brad new data from random noise.

The model is constructed using 2 main parts, the Generator and the Discriminator

The generator tries to Generate data based on the training data

The discriminator is a classifier that outputs a probability of whether or not it thinks the output of the generator was taking from the original dataset or generated. We want this probability to be as close to 1 as possible as that signifies that the discriminator thinks the data was taken straight from the dataset

In [ ]:
```
import os
import torch
import torchvision
import imageio
import torch.nn as nn
import matplotlib.pyplot as plt
import time
from torchvision import transforms
from torch.utils.data import DataLoader
from IPython.display import Image
from torchvision.utils import save_image
```

In [ ]:
```
# Use GPU for training if available

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

```
True
```

Out[ ]:
```
'cuda'
```

In [ ]:
```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
dataset = torchvision.datasets.MNIST("./data", download=True, transform=transform)
```

In [ ]:
```
print(f'[dataset shape: {dataset.data.shape} and type: {type(dataset.data)}]\n[targ
```

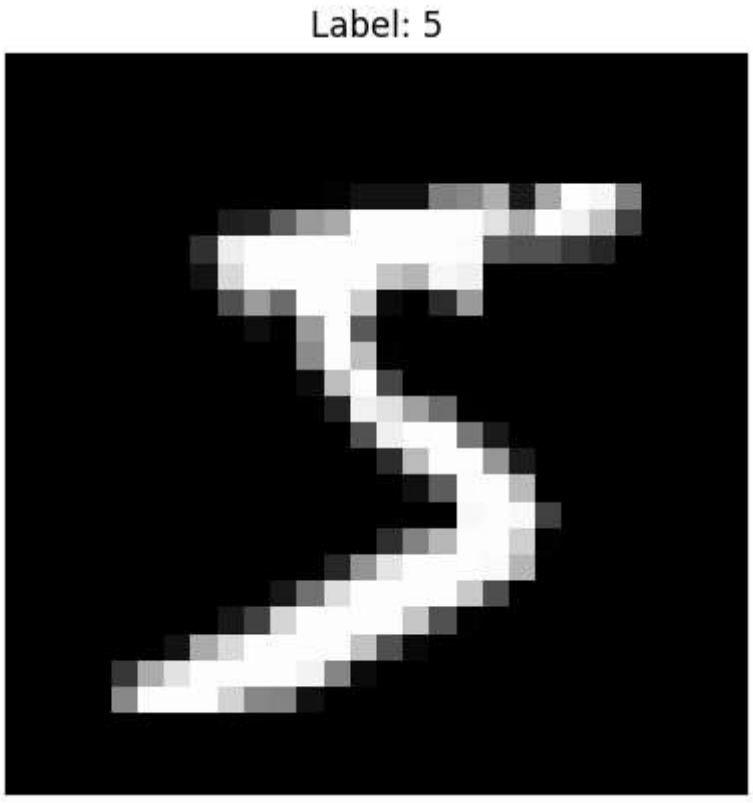
```
[dataset shape: torch.Size([60000, 28, 28]) and type: <class 'torch.Tensor'>]
[target shape: torch.Size([60000]) and type: <class 'torch.Tensor'>]
```

In [ ]:
```
# Looking at the data
x, y = dataset[0]
x.shape, type(y)
```

Out[ ]:
```
(torch.Size([1, 28, 28]), int)
```

In [ ]:
```
x.squeeze_()
plt.imshow(x, cmap='gray')
plt.xticks([])
plt.yticks([])
plt.title(f'Label: {y}')
```

Out[ ]:
```
Text(0.5, 1.0, 'Label: 5')
```



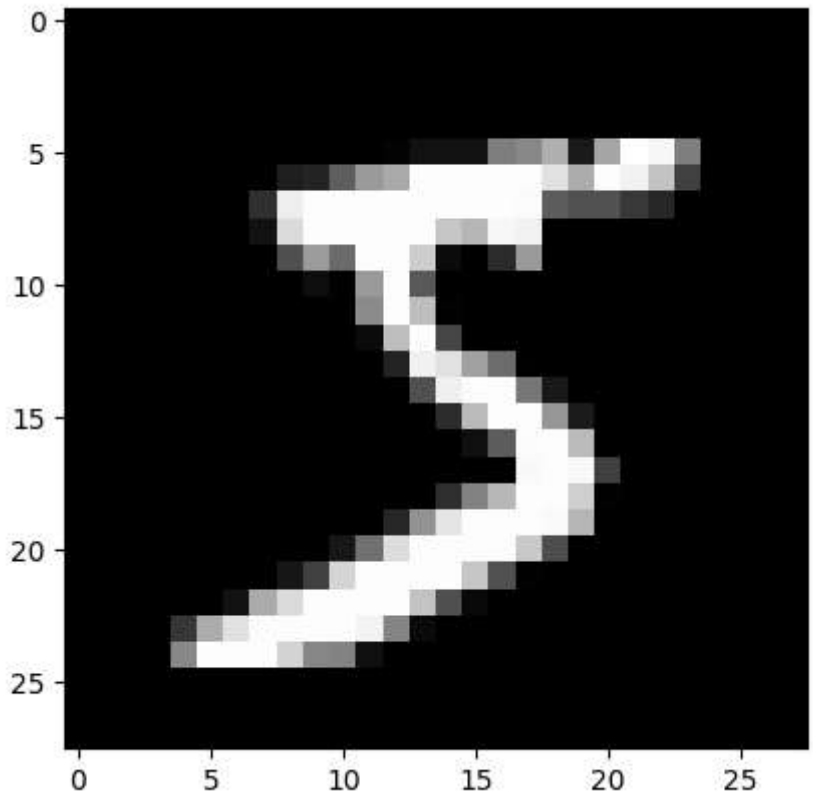
In [ ]:
```
img, label = dataset[0]
print(f'Label: {label}')
print(img[:, 10:15, 10:15])
torch.min(img), torch.max(img)
```

```
Label: 5
tensor([[[-0.9922,  0.2078,  0.9843, -0.2941, -1.0000],
         [-1.0000,  0.0902,  0.9843,  0.4902, -0.9843],
         [-1.0000, -0.9137,  0.4902,  0.9843, -0.4510],
         [-1.0000, -1.0000, -0.7255,  0.8902,  0.7647],
         [-1.0000, -1.0000, -1.0000, -0.3647,  0.8824]]])
```

Out[ ]:
```
(tensor(-1.), tensor(1.))
```

In [ ]:
```
def denorm(x):
    out = (x + 1) / 2
    return out.clamp(0, 1)

img_norm = denorm(img)
plt.imshow(img_norm[0], cmap='gray')
print(f'Label: {label}')
```

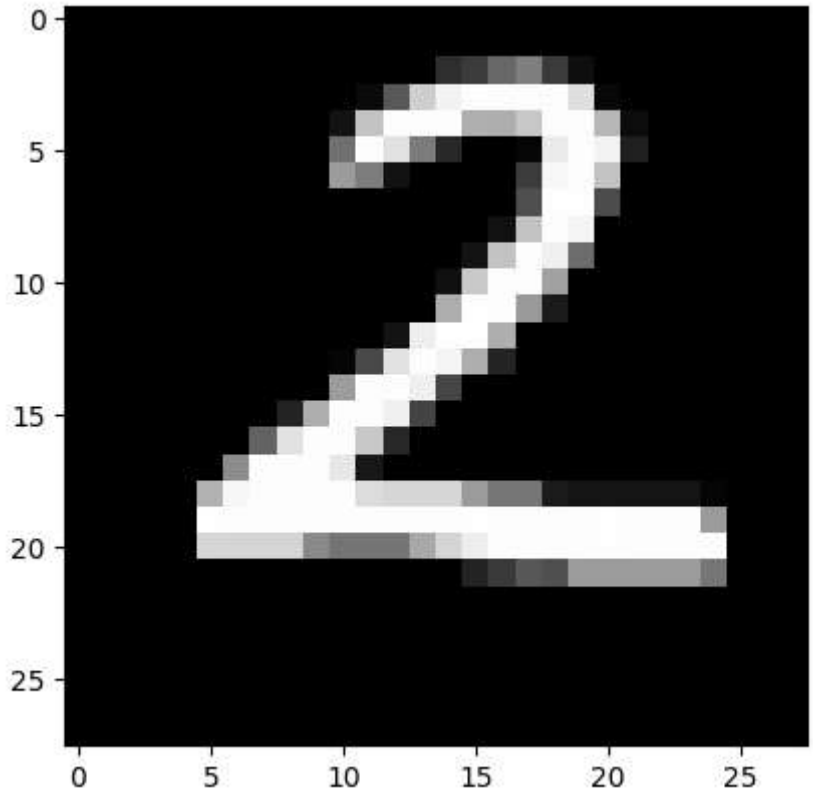
```
Label: 5
```



In [ ]:
```
batch_size = 100
dataloader = DataLoader(dataset, batch_size, shuffle=True)
```

In [ ]:
```
for img_batch, label_batch in dataloader:
    print('First batch:')
    print(img_batch.shape)
    plt.imshow(img_batch[0][0], cmap='gray')
    print(label_batch)
    break
```

```
First batch:
torch.Size([100, 1, 28, 28])
tensor([2, 6, 2, 5, 6, 1, 2, 7, 6, 4, 7, 8, 8, 3, 6, 3, 5, 3, 0, 9, 8, 6, 7, 6,
        8, 4, 8, 6, 9, 1, 3, 3, 5, 8, 1, 2, 0, 9, 2, 1, 8, 4, 4, 0, 0, 3, 0, 9,
        8, 4, 3, 4, 5, 4, 6, 8, 3, 1, 9, 5, 4, 0, 5, 7, 6, 9, 9, 0, 2, 8, 5, 3,
        9, 1, 7, 6, 1, 2, 5, 0, 1, 2, 4, 6, 1, 4, 0, 9, 9, 9, 9, 7, 9, 7, 9, 5,
        3, 1, 5, 1])
```



In [ ]:
```python
img_size = 784 # (28*28)
hidden_size = 256
```

In [ ]:
```python
class DiscriminatorModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = nn.Linear(img_size, hidden_size)
        self.leakyReLU = nn.LeakyReLU(0.2)
        self.linear2 = nn.Linear(hidden_size, hidden_size)
        self.fc = nn.Linear(hidden_size, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.linear1(x)
        x = self.leakyReLU(x)
        x = self.linear2(x)
        x = self.leakyReLU(x)
        x = self.fc(x)
        x = self.sigmoid(x)
        return x

discriminator = DiscriminatorModel()
```

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

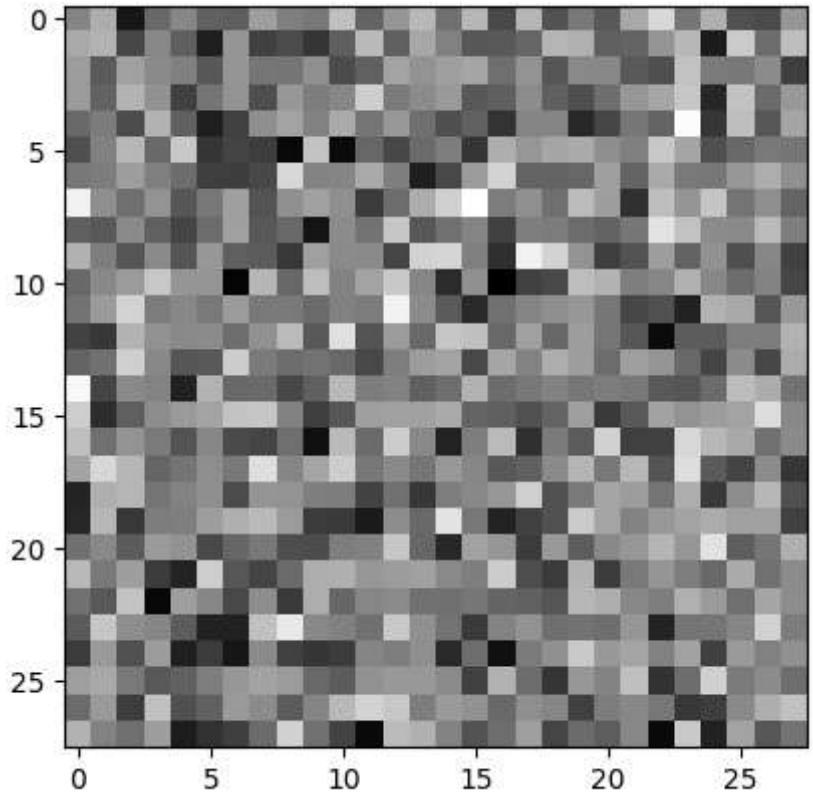
Out[ ]:
```
DiscriminatorModel(
  (linear1): Linear(in_features=784, out_features=256, bias=True)
  (leakyReLU): LeakyReLU(negative_slope=0.2)
  (linear2): Linear(in_features=256, out_features=256, bias=True)
  (fc): Linear(in_features=256, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
```

In [ ]:
```
latent_size = 64

class GeneratorModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = nn.Linear(latent_size, hidden_size)
        self.ReLU = nn.ReLU()
        self.linear2 = nn.Linear(hidden_size, hidden_size)
        self.fc = nn.Linear(hidden_size, img_size)
        self.tanh = nn.Tanh()

    def forward(self, x):
        x = self.linear1(x)
        x = self.ReLU(x)
        x = self.linear2(x)
        x = self.ReLU(x)
        x = self.fc(x)
        x = self.tanh(x)
        return x

generator = GeneratorModel()
generator = generator.to(device)
generator
```

Out[ ]:
```
GeneratorModel(
  (linear1): Linear(in_features=64, out_features=256, bias=True)
  (ReLU): ReLU()
  (linear2): Linear(in_features=256, out_features=256, bias=True)
  (fc): Linear(in_features=256, out_features=784, bias=True)
  (tanh): Tanh()
)
```

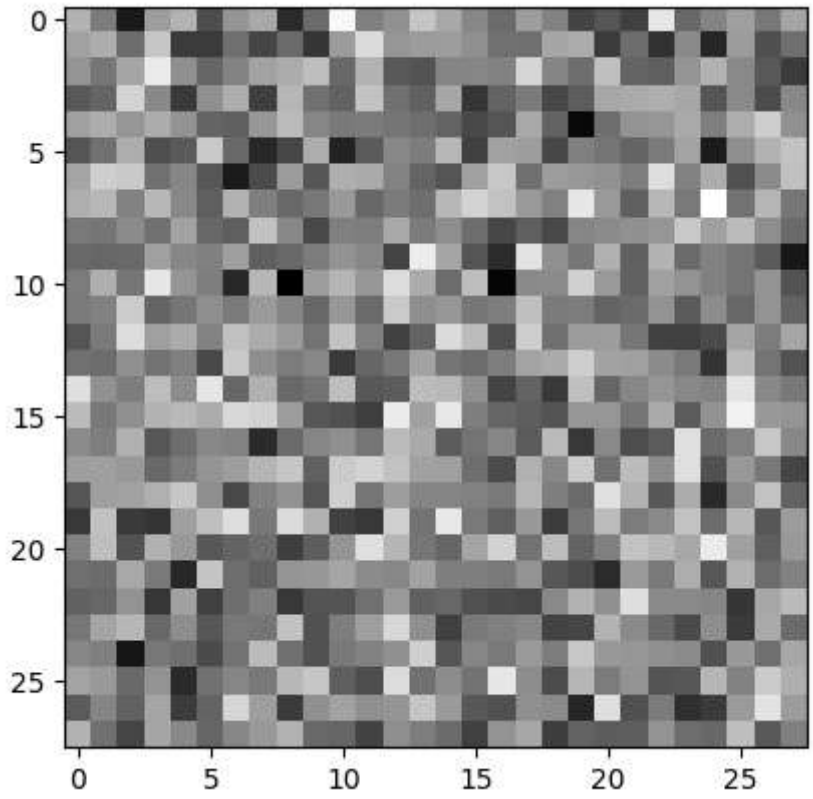
In [ ]:
```
z = torch.randn(2, latent_size).to(device)
Y = generator(z)
gen_imgs = denorm(Y.reshape((-1, 28, 28)).detach().cpu())
plt.imshow(gen_imgs[0], cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x2168afe8b20>
```



In [ ]:
```
plt.imshow(gen_imgs[1], cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x2168b041300>
```



In [ ]:
```
loss = nn.BCELoss()
d_optimizer = torch.optim.Adam(discriminator.parameters(), lr=0.0002)
```

```
g_optimizer = torch.optim.Adam(generator.parameters(), lr=0.0002)

# print the number of parameters in the discriminator and generator
print("[Discriminator] Number of parameters: %d" % sum(p.numel() for p in discrimin
print("[Generator] Number of parameters: %d" % sum(p.numel() for p in generator.par
```

```
[Discriminator] Number of parameters: 267009
[Generator] Number of parameters: 283920
```

In [ ]:
```
def reset_grad():
    d_optimizer.zero_grad()
    g_optimizer.zero_grad()
```

In [ ]:
```
def train_discriminator(images):
    real_labels = torch.ones(batch_size, 1).to(device)
    fake_labels = torch.zeros(batch_size, 1).to(device)

    outputs = discriminator(images)
    d_loss_real = loss(outputs, real_labels)
    real_score = outputs

    z = torch.randn(batch_size, latent_size).to(device)
    fake_images = generator(z)
    outputs = discriminator(fake_images)
    d_loss_fake = loss(outputs, fake_labels)
    fake_score = outputs

    d_loss = d_loss_real + d_loss_fake
    reset_grad
    d_loss.backward()
    d_optimizer.step()

    return d_loss, real_score, fake_score
```

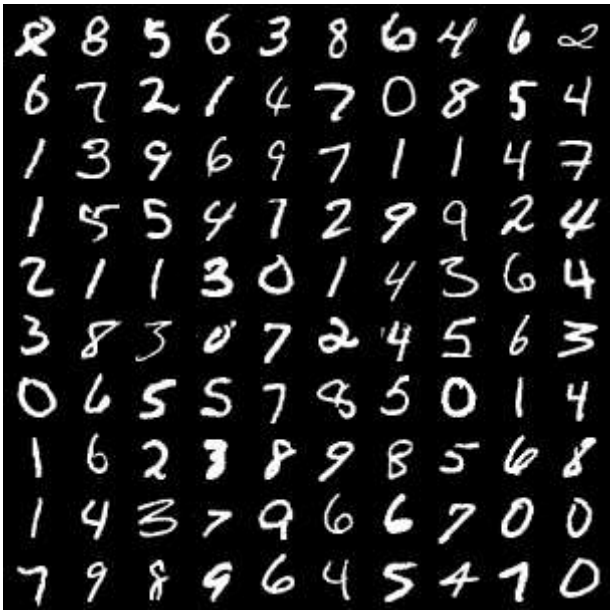
In [ ]:
```
def train_generator():
    z = torch.randn(batch_size, latent_size).to(device)
    fake_images = generator(z)
    labels = torch.ones(batch_size, 1).to(device)
    g_loss = loss(discriminator(fake_images), labels)

    reset_grad()
    g_loss.backward()
    g_optimizer.step()
    return g_loss, fake_images
```

In [ ]:
```
sample_dir = 'output'

if not os.path.exists(sample_dir):
    os.makedirs(sample_dir)
```

In [ ]:
```
for images, _ in dataloader:
    images = images.reshape(images.size(0), 1, 28, 28)
    save_image(denorm(images), os.path.join(sample_dir, 'real_images.png'), nrow=10
```

```
    break
Image(os.path.join(sample_dir, 'real_images.png'))
```

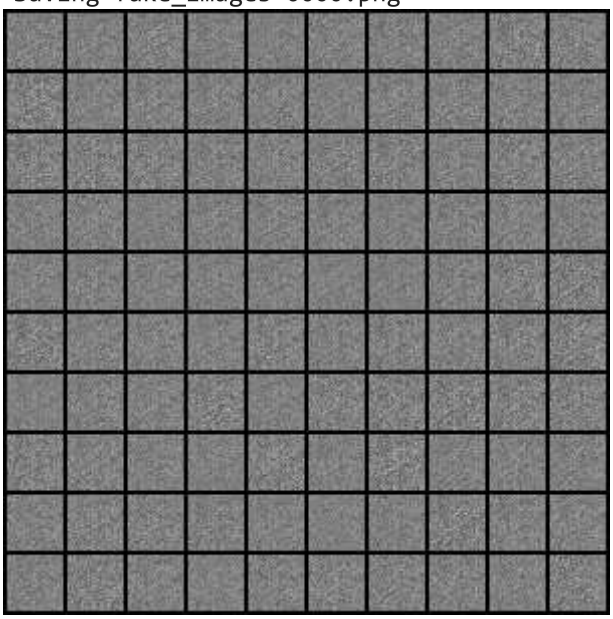
Out[ ]:



In [ ]:
```
sample_vectors = torch.randn(batch_size, latent_size).to(device)

def save_fake_images(index):
    fake_images = generator(sample_vectors)
    fake_images = fake_images.reshape(fake_images.size(0), 1, 28, 28)
    fake_fname = 'fake_images-{0:0=4d}.png'.format(index)
    print('Saving', fake_fname)
    save_image(denorm(fake_images), os.path.join(sample_dir, fake_fname), nrow=10)

# Save before training
save_fake_images(0)
Image(os.path.join(sample_dir, 'fake_images-0000.png'))
```

```
Saving fake_images-0000.png
```

Out[ ]:



In [ ]:
```
%%time
start0 = time.time()
```

```
num_epochs = 300
total_step = len(dataloader)
d_losses, g_losses, real_scores, fake_scores = [], [], [], []
for epoch in range(num_epochs):
    for i, (images, _) in enumerate(dataloader):
        images = images.reshape(batch_size, -1).to(device)

        # training the discriminator and generator
        d_loss, real_score, fake_score = train_discriminator(images)
        g_loss, fake_images = train_generator()

        if (i+1) % 200 == 0:
            d_losses.append(d_loss.item())
            g_losses.append(g_loss.item())
            real_scores.append(real_score.mean().item())
            fake_scores.append(fake_score.mean().item())
            print('Epoch [{}/{}], Step [{}/{}], d_loss: {:.4f}, g_loss: {:.4f}, Dis

    save_fake_images(epoch+1)
duration0 = time.time() - start0
print('Total time: {:.2f}s'.format(duration0))
```

```
Epoch [0/300], Step [200/600], d_loss: 1.0167, g_loss: 0.6986, Disc_real_image: 0.
93, Disc_gen_image: 0.60
Epoch [0/300], Step [400/600], d_loss: 0.8103, g_loss: 0.6062, Disc_real_image: 0.
92, Disc_gen_image: 0.51
Epoch [0/300], Step [600/600], d_loss: 0.8210, g_loss: 0.6207, Disc_real_image: 0.
98, Disc_gen_image: 0.54
Saving fake_images-0001.png
Epoch [1/300], Step [200/600], d_loss: 0.9384, g_loss: 0.6683, Disc_real_image: 0.
93, Disc_gen_image: 0.57
Epoch [1/300], Step [400/600], d_loss: 0.9560, g_loss: 0.7698, Disc_real_image: 0.
91, Disc_gen_image: 0.56
Epoch [1/300], Step [600/600], d_loss: 0.9075, g_loss: 0.6769, Disc_real_image: 0.
97, Disc_gen_image: 0.57
Saving fake_images-0002.png
Epoch [2/300], Step [200/600], d_loss: 0.8737, g_loss: 0.6851, Disc_real_image: 0.
94, Disc_gen_image: 0.55
Epoch [2/300], Step [400/600], d_loss: 1.1732, g_loss: 0.5163, Disc_real_image: 0.
88, Disc_gen_image: 0.64
Epoch [2/300], Step [600/600], d_loss: 1.1334, g_loss: 0.6517, Disc_real_image: 0.
92, Disc_gen_image: 0.64
Saving fake_images-0003.png
Epoch [3/300], Step [200/600], d_loss: 0.9726, g_loss: 0.5662, Disc_real_image: 0.
87, Disc_gen_image: 0.56
Epoch [3/300], Step [400/600], d_loss: 0.9049, g_loss: 0.6292, Disc_real_image: 0.
91, Disc_gen_image: 0.55
Epoch [3/300], Step [600/600], d_loss: 0.9795, g_loss: 0.5911, Disc_real_image: 0.
91, Disc_gen_image: 0.58
Saving fake_images-0004.png
Epoch [4/300], Step [200/600], d_loss: 0.9224, g_loss: 0.6099, Disc_real_image: 0.
92, Disc_gen_image: 0.56
Epoch [4/300], Step [400/600], d_loss: 0.9357, g_loss: 0.6073, Disc_real_image: 0.
90, Disc_gen_image: 0.55
Epoch [4/300], Step [600/600], d_loss: 1.0512, g_loss: 0.5921, Disc_real_image: 0.
88, Disc_gen_image: 0.59
Saving fake_images-0005.png
Epoch [5/300], Step [200/600], d_loss: 0.9543, g_loss: 0.6499, Disc_real_image: 0.
89, Disc_gen_image: 0.56
Epoch [5/300], Step [400/600], d_loss: 1.1121, g_loss: 0.5684, Disc_real_image: 0.
89, Disc_gen_image: 0.61
Epoch [5/300], Step [600/600], d_loss: 0.9892, g_loss: 0.6270, Disc_real_image: 0.
90, Disc_gen_image: 0.57
Saving fake_images-0006.png
Epoch [6/300], Step [200/600], d_loss: 0.9419, g_loss: 0.6346, Disc_real_image: 0.
88, Disc_gen_image: 0.53
Epoch [6/300], Step [400/600], d_loss: 1.0937, g_loss: 0.6300, Disc_real_image: 0.
89, Disc_gen_image: 0.61
Epoch [6/300], Step [600/600], d_loss: 0.9843, g_loss: 0.5974, Disc_real_image: 0.
90, Disc_gen_image: 0.57
Saving fake_images-0007.png
Epoch [7/300], Step [200/600], d_loss: 1.0004, g_loss: 0.6029, Disc_real_image: 0.
92, Disc_gen_image: 0.59
Epoch [7/300], Step [400/600], d_loss: 0.9049, g_loss: 0.6323, Disc_real_image: 0.
90, Disc_gen_image: 0.54
Epoch [7/300], Step [600/600], d_loss: 1.0154, g_loss: 0.6252, Disc_real_image: 0.
90, Disc_gen_image: 0.58
Saving fake_images-0008.png
```

```
Epoch [8/300], Step [200/600], d_loss: 1.0000, g_loss: 0.6462, Disc_real_image: 0.
92, Disc_gen_image: 0.59
Epoch [8/300], Step [400/600], d_loss: 1.0408, g_loss: 0.5969, Disc_real_image: 0.
90, Disc_gen_image: 0.59
Epoch [8/300], Step [600/600], d_loss: 1.1158, g_loss: 0.6489, Disc_real_image: 0.
88, Disc_gen_image: 0.62
Saving fake_images-0009.png
Epoch [9/300], Step [200/600], d_loss: 0.9723, g_loss: 0.6683, Disc_real_image: 0.
93, Disc_gen_image: 0.58
Epoch [9/300], Step [400/600], d_loss: 0.9948, g_loss: 0.6496, Disc_real_image: 0.
90, Disc_gen_image: 0.57
Epoch [9/300], Step [600/600], d_loss: 1.0387, g_loss: 0.7442, Disc_real_image: 0.
92, Disc_gen_image: 0.60
Saving fake_images-0010.png
Epoch [10/300], Step [200/600], d_loss: 1.1099, g_loss: 0.4886, Disc_real_image:
0.82, Disc_gen_image: 0.57
Epoch [10/300], Step [400/600], d_loss: 1.0688, g_loss: 0.6600, Disc_real_image:
0.91, Disc_gen_image: 0.60
Epoch [10/300], Step [600/600], d_loss: 0.9340, g_loss: 0.6235, Disc_real_image:
0.87, Disc_gen_image: 0.53
Saving fake_images-0011.png
Epoch [11/300], Step [200/600], d_loss: 0.9915, g_loss: 0.5673, Disc_real_image:
0.93, Disc_gen_image: 0.59
Epoch [11/300], Step [400/600], d_loss: 1.1003, g_loss: 0.6568, Disc_real_image:
0.89, Disc_gen_image: 0.61
Epoch [11/300], Step [600/600], d_loss: 0.9446, g_loss: 0.7207, Disc_real_image:
0.87, Disc_gen_image: 0.54
Saving fake_images-0012.png
Epoch [12/300], Step [200/600], d_loss: 1.0307, g_loss: 0.6211, Disc_real_image:
0.90, Disc_gen_image: 0.59
Epoch [12/300], Step [400/600], d_loss: 0.9915, g_loss: 0.5484, Disc_real_image:
0.89, Disc_gen_image: 0.57
Epoch [12/300], Step [600/600], d_loss: 0.8921, g_loss: 0.6252, Disc_real_image:
0.92, Disc_gen_image: 0.53
Saving fake_images-0013.png
Epoch [13/300], Step [200/600], d_loss: 1.0152, g_loss: 0.6062, Disc_real_image:
0.90, Disc_gen_image: 0.58
Epoch [13/300], Step [400/600], d_loss: 1.0189, g_loss: 0.6365, Disc_real_image:
0.89, Disc_gen_image: 0.58
Epoch [13/300], Step [600/600], d_loss: 0.8791, g_loss: 0.7697, Disc_real_image:
0.92, Disc_gen_image: 0.54
Saving fake_images-0014.png
Epoch [14/300], Step [200/600], d_loss: 0.9378, g_loss: 0.5973, Disc_real_image:
0.90, Disc_gen_image: 0.55
Epoch [14/300], Step [400/600], d_loss: 0.9263, g_loss: 0.6270, Disc_real_image:
0.89, Disc_gen_image: 0.55
Epoch [14/300], Step [600/600], d_loss: 0.9629, g_loss: 0.5689, Disc_real_image:
0.90, Disc_gen_image: 0.56
Saving fake_images-0015.png
Epoch [15/300], Step [200/600], d_loss: 0.9572, g_loss: 0.6570, Disc_real_image:
0.92, Disc_gen_image: 0.57
Epoch [15/300], Step [400/600], d_loss: 0.9564, g_loss: 0.6865, Disc_real_image:
0.90, Disc_gen_image: 0.56
Epoch [15/300], Step [600/600], d_loss: 1.0844, g_loss: 0.6362, Disc_real_image:
0.88, Disc_gen_image: 0.60
Saving fake_images-0016.png
```

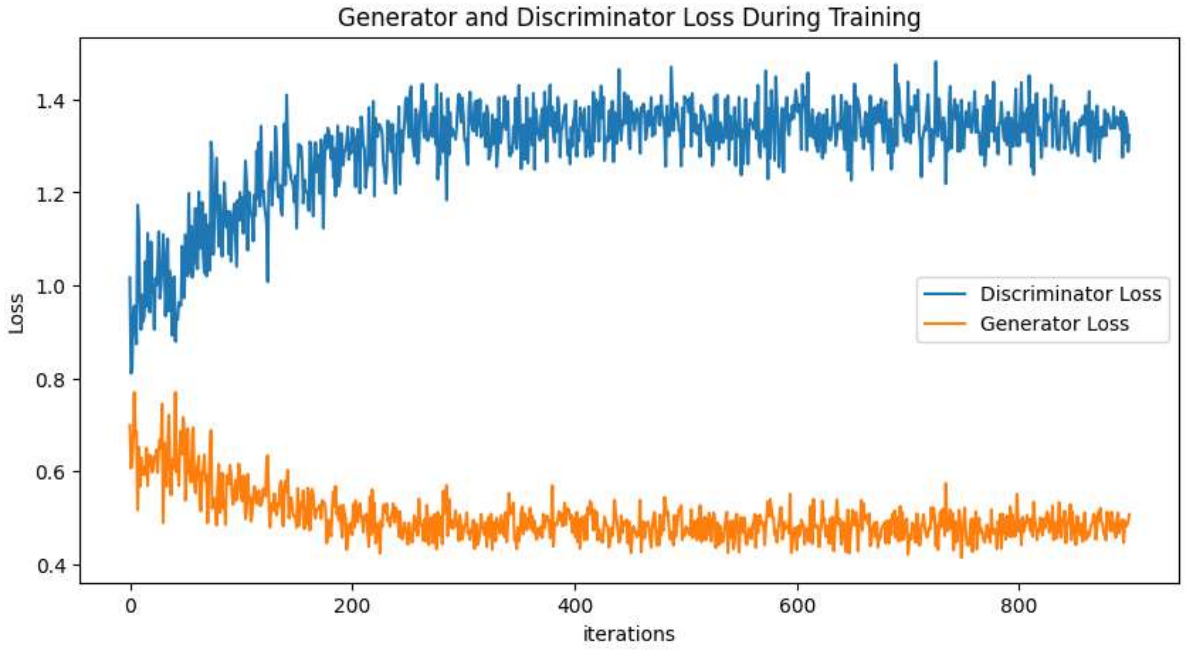
```
Epoch [16/300], Step [200/600], d_loss: 1.0185, g_loss: 0.7157, Disc_real_image:
0.88, Disc_gen_image: 0.57
Epoch [16/300], Step [400/600], d_loss: 0.9731, g_loss: 0.6854, Disc_real_image:
0.83, Disc_gen_image: 0.52
Epoch [16/300], Step [600/600], d_loss: 1.1091, g_loss: 0.5369, Disc_real_image:
0.84, Disc_gen_image: 0.59
Saving fake_images-0017.png
Epoch [17/300], Step [200/600], d_loss: 1.0456, g_loss: 0.6034, Disc_real_image:
0.83, Disc_gen_image: 0.56
Epoch [17/300], Step [400/600], d_loss: 1.0201, g_loss: 0.6913, Disc_real_image:
0.89, Disc_gen_image: 0.58
Epoch [17/300], Step [600/600], d_loss: 1.1982, g_loss: 0.5961, Disc_real_image:
0.88, Disc_gen_image: 0.64
Saving fake_images-0018.png
Epoch [18/300], Step [200/600], d_loss: 1.0280, g_loss: 0.6221, Disc_real_image:
0.86, Disc_gen_image: 0.57
Epoch [18/300], Step [400/600], d_loss: 1.1281, g_loss: 0.5762, Disc_real_image:
0.86, Disc_gen_image: 0.60
Epoch [18/300], Step [600/600], d_loss: 1.0171, g_loss: 0.6673, Disc_real_image:
0.89, Disc_gen_image: 0.58
Saving fake_images-0019.png
Epoch [19/300], Step [200/600], d_loss: 1.0304, g_loss: 0.6934, Disc_real_image:
0.90, Disc_gen_image: 0.58
Epoch [19/300], Step [400/600], d_loss: 1.1063, g_loss: 0.5797, Disc_real_image:
0.86, Disc_gen_image: 0.60
Epoch [19/300], Step [600/600], d_loss: 1.1661, g_loss: 0.5528, Disc_real_image:
0.83, Disc_gen_image: 0.61
Saving fake_images-0020.png
Epoch [20/300], Step [200/600], d_loss: 1.0486, g_loss: 0.6013, Disc_real_image:
0.87, Disc_gen_image: 0.58
Epoch [20/300], Step [400/600], d_loss: 1.0355, g_loss: 0.6135, Disc_real_image:
0.85, Disc_gen_image: 0.57
Epoch [20/300], Step [600/600], d_loss: 1.2014, g_loss: 0.6326, Disc_real_image:
0.85, Disc_gen_image: 0.63
Saving fake_images-0021.png
Epoch [21/300], Step [200/600], d_loss: 1.0791, g_loss: 0.5284, Disc_real_image:
0.81, Disc_gen_image: 0.56
Epoch [21/300], Step [400/600], d_loss: 1.1107, g_loss: 0.5136, Disc_real_image:
0.81, Disc_gen_image: 0.57
Epoch [21/300], Step [600/600], d_loss: 1.1791, g_loss: 0.5965, Disc_real_image:
0.85, Disc_gen_image: 0.62
Saving fake_images-0022.png
Epoch [22/300], Step [200/600], d_loss: 1.0655, g_loss: 0.6038, Disc_real_image:
0.84, Disc_gen_image: 0.57
Epoch [22/300], Step [400/600], d_loss: 1.0267, g_loss: 0.6351, Disc_real_image:
0.84, Disc_gen_image: 0.56
Epoch [22/300], Step [600/600], d_loss: 1.1524, g_loss: 0.5784, Disc_real_image:
0.86, Disc_gen_image: 0.61
Saving fake_images-0023.png
Epoch [23/300], Step [200/600], d_loss: 1.0199, g_loss: 0.5909, Disc_real_image:
0.85, Disc_gen_image: 0.56
Epoch [23/300], Step [400/600], d_loss: 1.1220, g_loss: 0.4888, Disc_real_image:
0.81, Disc_gen_image: 0.58
Epoch [23/300], Step [600/600], d_loss: 1.1311, g_loss: 0.5733, Disc_real_image:
0.80, Disc_gen_image: 0.56
Saving fake_images-0024.png
```

```
Epoch [24/300], Step [200/600], d_loss: 1.0326, g_loss: 0.6421, Disc_real_image:
0.82, Disc_gen_image: 0.53
Epoch [24/300], Step [400/600], d_loss: 1.3088, g_loss: 0.6878, Disc_real_image:
0.86, Disc_gen_image: 0.66
Epoch [24/300], Step [600/600], d_loss: 1.2808, g_loss: 0.5196, Disc_real_image:
0.87, Disc_gen_image: 0.66
Saving fake_images-0025.png
Epoch [25/300], Step [200/600], d_loss: 1.0664, g_loss: 0.5261, Disc_real_image:
0.86, Disc_gen_image: 0.58
Epoch [25/300], Step [400/600], d_loss: 1.1529, g_loss: 0.5416, Disc_real_image:
0.80, Disc_gen_image: 0.58
Epoch [25/300], Step [600/600], d_loss: 1.2151, g_loss: 0.5334, Disc_real_image:
0.87, Disc_gen_image: 0.64
Saving fake_images-0026.png
Epoch [26/300], Step [200/600], d_loss: 1.2741, g_loss: 0.4836, Disc_real_image:
0.80, Disc_gen_image: 0.62
Epoch [26/300], Step [400/600], d_loss: 1.1467, g_loss: 0.5722, Disc_real_image:
0.82, Disc_gen_image: 0.59
Epoch [26/300], Step [600/600], d_loss: 1.0837, g_loss: 0.6142, Disc_real_image:
0.85, Disc_gen_image: 0.58
Saving fake_images-0027.png
Epoch [27/300], Step [200/600], d_loss: 1.1936, g_loss: 0.5102, Disc_real_image:
0.83, Disc_gen_image: 0.61
Epoch [27/300], Step [400/600], d_loss: 1.1167, g_loss: 0.5237, Disc_real_image:
0.84, Disc_gen_image: 0.59
Epoch [27/300], Step [600/600], d_loss: 1.0623, g_loss: 0.5960, Disc_real_image:
0.84, Disc_gen_image: 0.57
Saving fake_images-0028.png
Epoch [28/300], Step [200/600], d_loss: 1.1423, g_loss: 0.5202, Disc_real_image:
0.83, Disc_gen_image: 0.60
Epoch [28/300], Step [400/600], d_loss: 1.2218, g_loss: 0.5370, Disc_real_image:
0.84, Disc_gen_image: 0.63
Epoch [28/300], Step [600/600], d_loss: 1.1667, g_loss: 0.4847, Disc_real_image:
0.84, Disc_gen_image: 0.61
Saving fake_images-0029.png
Epoch [29/300], Step [200/600], d_loss: 1.1259, g_loss: 0.5910, Disc_real_image:
0.87, Disc_gen_image: 0.61
Epoch [29/300], Step [400/600], d_loss: 1.1601, g_loss: 0.5848, Disc_real_image:
0.79, Disc_gen_image: 0.57
Epoch [29/300], Step [600/600], d_loss: 1.0672, g_loss: 0.5722, Disc_real_image:
0.85, Disc_gen_image: 0.57
Saving fake_images-0030.png
Epoch [30/300], Step [200/600], d_loss: 1.1582, g_loss: 0.6127, Disc_real_image:
0.87, Disc_gen_image: 0.62
Epoch [30/300], Step [400/600], d_loss: 1.0516, g_loss: 0.5955, Disc_real_image:
0.84, Disc_gen_image: 0.57
Epoch [30/300], Step [600/600], d_loss: 1.1093, g_loss: 0.5433, Disc_real_image:
0.80, Disc_gen_image: 0.57
Saving fake_images-0031.png
Epoch [31/300], Step [200/600], d_loss: 1.1544, g_loss: 0.5284, Disc_real_image:
0.82, Disc_gen_image: 0.59
Epoch [31/300], Step [400/600], d_loss: 1.1760, g_loss: 0.5544, Disc_real_image:
0.85, Disc_gen_image: 0.62
Epoch [31/300], Step [600/600], d_loss: 1.1087, g_loss: 0.5809, Disc_real_image:
0.83, Disc_gen_image: 0.58
Saving fake_images-0032.png
```

```
Epoch [32/300], Step [200/600], d_loss: 1.0405, g_loss: 0.5372, Disc_real_image:
0.84, Disc_gen_image: 0.56
Epoch [32/300], Step [400/600], d_loss: 1.1833, g_loss: 0.5520, Disc_real_image:
0.83, Disc_gen_image: 0.61
Epoch [32/300], Step [600/600], d_loss: 1.1412, g_loss: 0.6155, Disc_real_image:
0.84, Disc_gen_image: 0.60
Saving fake_images-0033.png
Epoch [33/300], Step [200/600], d_loss: 1.2004, g_loss: 0.5830, Disc_real_image:
0.82, Disc_gen_image: 0.61
Epoch [33/300], Step [400/600], d_loss: 1.1393, g_loss: 0.5563, Disc_real_image:
0.83, Disc_gen_image: 0.60
Epoch [33/300], Step [600/600], d_loss: 1.1896, g_loss: 0.5359, Disc_real_image:
0.84, Disc_gen_image: 0.62
Saving fake_images-0034.png
Epoch [34/300], Step [200/600], d_loss: 1.1120, g_loss: 0.5901, Disc_real_image:
0.80, Disc_gen_image: 0.56
Epoch [34/300], Step [400/600], d_loss: 1.1907, g_loss: 0.5702, Disc_real_image:
0.82, Disc_gen_image: 0.61
Epoch [34/300], Step [600/600], d_loss: 1.2683, g_loss: 0.4970, Disc_real_image:
0.84, Disc_gen_image: 0.64
Saving fake_images-0035.png
...
Epoch [295/300], Step [200/600], d_loss: 1.3842, g_loss: 0.4703, Disc_real_image:
0.75, Disc_gen_image: 0.65
Epoch [295/300], Step [400/600], d_loss: 1.3382, g_loss: 0.4882, Disc_real_image:
0.71, Disc_gen_image: 0.62
Epoch [295/300], Step [600/600], d_loss: 1.3516, g_loss: 0.4566, Disc_real_image:
0.72, Disc_gen_image: 0.63
Saving fake_images-0296.png
Epoch [296/300], Step [200/600], d_loss: 1.3631, g_loss: 0.5020, Disc_real_image:
0.74, Disc_gen_image: 0.63
Epoch [296/300], Step [400/600], d_loss: 1.3467, g_loss: 0.5103, Disc_real_image:
0.74, Disc_gen_image: 0.63
Epoch [296/300], Step [600/600], d_loss: 1.3549, g_loss: 0.4660, Disc_real_image:
0.76, Disc_gen_image: 0.64
Saving fake_images-0297.png
Epoch [297/300], Step [200/600], d_loss: 1.3352, g_loss: 0.4881, Disc_real_image:
0.74, Disc_gen_image: 0.62
Epoch [297/300], Step [400/600], d_loss: 1.3754, g_loss: 0.4779, Disc_real_image:
0.73, Disc_gen_image: 0.64
Epoch [297/300], Step [600/600], d_loss: 1.2755, g_loss: 0.4958, Disc_real_image:
0.76, Disc_gen_image: 0.62
Saving fake_images-0298.png
Epoch [298/300], Step [200/600], d_loss: 1.3719, g_loss: 0.4460, Disc_real_image:
0.74, Disc_gen_image: 0.64
Epoch [298/300], Step [400/600], d_loss: 1.3060, g_loss: 0.4926, Disc_real_image:
0.73, Disc_gen_image: 0.61
Epoch [298/300], Step [600/600], d_loss: 1.3610, g_loss: 0.4810, Disc_real_image:
0.74, Disc_gen_image: 0.64
Saving fake_images-0299.png
Epoch [299/300], Step [200/600], d_loss: 1.3478, g_loss: 0.4834, Disc_real_image:
0.73, Disc_gen_image: 0.63
Epoch [299/300], Step [400/600], d_loss: 1.2880, g_loss: 0.4906, Disc_real_image:
0.74, Disc_gen_image: 0.61
Epoch [299/300], Step [600/600], d_loss: 1.3232, g_loss: 0.5066, Disc_real_image:
0.73, Disc_gen_image: 0.62
```

```
Saving fake_images-0300.png
Total time: 4764.42s
CPU times: total: 1h 26min 39s
Wall time: 1h 19min 24s
```

In [ ]:
```
plt.figure(figsize=(10, 5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(d_losses,label="Discriminator Loss")
plt.plot(g_losses,label="Generator Loss")
plt.xlabel("iterations")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



In [ ]:
```
# save the model
torch.save(generator.state_dict(), './MNIST_Generator.pt')
torch.save(discriminator.state_dict(), './MNIST_Discriminator.pt')
```

In [ ]:
```
def load_images_from_dir(directory):
    image_list = []
    for filename in os.listdir(directory):
        if filename.endswith('.jpg') or filename.endswith('.png'):
            image = imageio.imread(os.path.join(directory, filename))
            image_list.append(image)
    return image_list

images = load_images_from_dir(sample_dir)
imageio.mimsave('output/out.gif', images, fps=20)
```

```
C:\Users\faree\AppData\Local\Temp\ipykernel_7360\456747677.py:5: DeprecationWarnin
g: Starting with ImageIO v3 the behavior of this function will switch to that of i
io.v3.imread. To keep the current behavior (and make this warning disappear) use `
import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(os.path.join(directory, filename))
```

In [ ]:
```
# Load the model parameters
generator.load_state_dict(torch.load('./MNIST_Generator.pt'))
discriminator.load_state_dict(torch.load('./MNIST_Discriminator.pt'))

# make a prediction
z = torch.randn(batch_size, latent_size).to(device)
fake_images = generator(z)
fake_images = fake_images.reshape(fake_images.size(0), 1, 28, 28)
save_image(denorm(fake_images), os.path.join(sample_dir, 'fake_images.png'), nrow=1
Image(os.path.join(sample_dir, 'fake_images.png'))
```

Out[ ]:

In [ ]: