Answer1

December 8, 2024

[1]:
```
import torch
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader, Subset
from torchvision import datasets
import torch.nn as nn
import matplotlib.pyplot as plt
```

[2]:
```
usps_train = datasets.USPS(root="./data", train=True, transform=ToTensor(),␣
↪download=True)
```

[3]:
```
device = 'cuda'
```

[4]:
```
class_indices = {i: [] for i in range(10)}
for idx, (_, label) in enumerate(usps_train):
    if len(class_indices[label]) < 500:
        class_indices[label].append(idx)

# Collect indices for the final dataset
final_indices = []
for indices in class_indices.values():
    final_indices.extend(indices)

# Subset the dataset
train_dataset = Subset(usps_train, final_indices)

# DataLoader
minibatches = DataLoader(train_dataset, batch_size=500, shuffle=True)

print(f"Number of samples in training dataset: {len(train_dataset)}")
```

```
Number of samples in training dataset: 5000
```

[5]:
```
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.l1 = nn.Linear(10, 128)
        self.l2 = nn.Linear(128, 16*16)
        self.a = nn.ReLU()
```

```
        self.s = nn.Sigmoid()
    def forward(self, X):
        X = self.a(self.l1(X))
        X = self.s(self.l2(X))
        return X

class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.l1 = nn.Linear(16*16, 128)
        self.l2 = nn.Linear(128, 1)
        self.a = nn.ReLU()
        self.s = nn.Sigmoid()
    def forward(self, X):
        X = self.a(self.l1(X))
        X = self.s(self.l2(X))
        return X
```

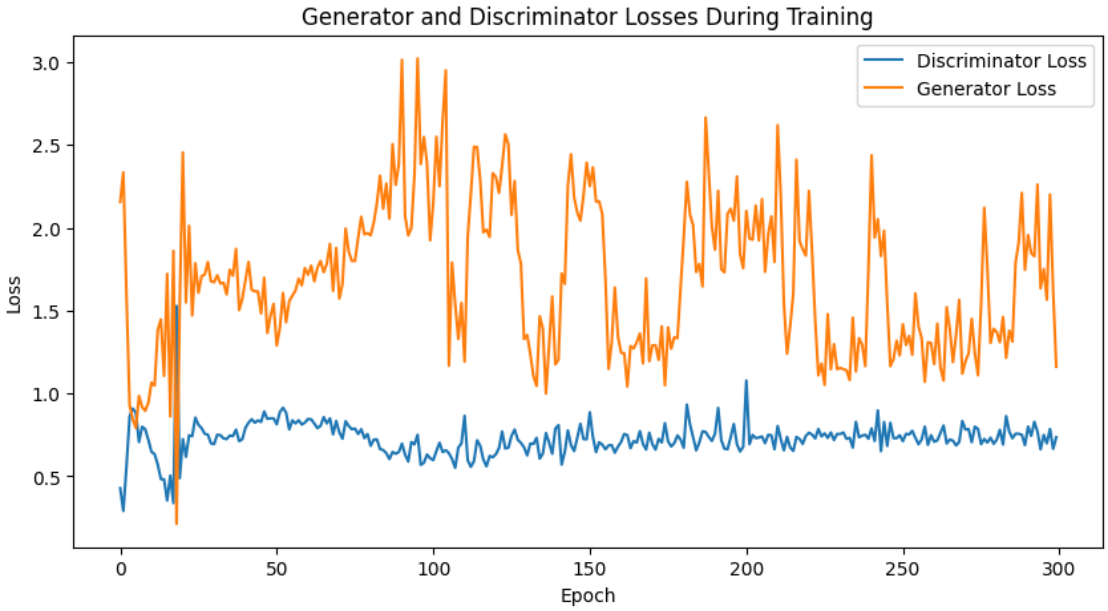
[6]:
```
G = Generator().to(device)
D = Discriminator().to(device)
loss_fn = nn.BCELoss()
lr = 0.15
optimizerG = torch.optim.SGD(G.parameters(), lr=lr)
optimizerD = torch.optim.SGD(D.parameters(), lr=lr)
epochs = 300

CE_D = torch.zeros(epochs)
CE_G = torch.zeros(epochs)

for start in range(epochs):
    for X, Y in minibatches:
        #loss acumalation for real images
        D.zero_grad()
        X_real, Y_real = nn.Flatten()(X), torch.ones((500, 1))
        X_real, Y_real = X_real.to(device), Y_real.to(device)
        outD = D(X_real)
        loss_real = loss_fn(outD, Y_real)
        #loss acumlation for fake images
        z = torch.randn(500, 10).to(device)
        X_fake, Y_fake = G(z), torch.zeros((500, 1))
        X_fake, Y_fake = X_fake.to(device), Y_fake.to(device)
        outD = D(X_fake)
        loss_fake = loss_fn(outD, Y_fake)
        #Gradient descent part for Discriminator
        lossD = loss_real + loss_fake
        lossD.backward()
        optimizerD.step()
```

```
        #Training of the Generator
        G.zero_grad()
        z = torch.randn(500, 10).to(device)
        Y = torch.ones((500, 1)).to(device)
        outG = G(z)
        outD = D(outG)
        lossG = loss_fn(outD, Y)
        lossG.backward()
        optimizerG.step()
    CE_D[start] = lossD
    CE_G[start] = lossG
```

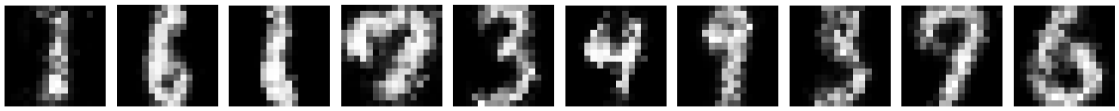
[7]:
```
plt.figure(figsize=(10, 5))
plt.plot(CE_D.detach().cpu().numpy(), label="Discriminator Loss")
plt.plot(CE_G.detach().cpu().numpy(), label="Generator Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Generator and Discriminator Losses During Training")
plt.legend()
plt.show()
```



[8]:
```
generated = G(torch.randn(10, 10).to(device))
generated = generated.view(10, 16, 16)

plt.figure(figsize=(15, 3))  # Adjusted figure size for better visualization
for i in range(10):
```

```
    plt.subplot(1, 10, i + 1)
    plt.imshow(generated[i].cpu().detach().numpy(), cmap="gray")
    plt.axis("off")  # Turn off axis for cleaner display
plt.tight_layout()
plt.show()
```



Answer2

December 8, 2024

[1]:
```
import torch
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader, Subset
from torchvision import datasets
import torch.nn as nn
import matplotlib.pyplot as plt
```

[2]:
```
usps_train = datasets.USPS(root="./data", train=True, transform=ToTensor(),␣
 ↪download=True)
```

[3]:
```
device = 'cuda'
```

[4]:
```
class_indices = {i: [] for i in range(10)}
for idx, (_, label) in enumerate(usps_train):
    if len(class_indices[label]) < 500:
        class_indices[label].append(idx)

# Collect indices for the final dataset
final_indices = []
for indices in class_indices.values():
    final_indices.extend(indices)

# Subset the dataset
train_dataset = Subset(usps_train, final_indices)

# DataLoader
minibatches = DataLoader(train_dataset, batch_size=500, shuffle=True)

print(f"Number of samples in training dataset: {len(train_dataset)}")
```

```
Number of samples in training dataset: 5000
```

[5]:
```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=8, kernel_size=4,␣
 ↪stride=2)
```

```
        self.conv2 = nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3,␣
 ↪stride=2)
        self.conv3 = nn.Conv2d(in_channels=16, out_channels=1, kernel_size=3,␣
 ↪stride=2)
        self.a = nn.Tanh()
        self.s = nn.Sigmoid()

    def forward(self, X):
        X = self.a(self.conv1(X))
        X = self.a(self.conv2(X))
        return self.s(self.conv3(X))[:, :, 0, 0]

class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.deconv1 = nn.ConvTranspose2d(in_channels=10, out_channels=16,␣
 ↪kernel_size=3, stride=1)
        self.bn1 = nn.BatchNorm2d(16)
        self.deconv2 = nn.ConvTranspose2d(in_channels=16, out_channels=8,␣
 ↪kernel_size=3, stride=2)
        self.bn2 = nn.BatchNorm2d(8)
        self.deconv3 = nn.ConvTranspose2d(in_channels=8, out_channels=1,␣
 ↪kernel_size=4, stride=2)
        self.a = nn.Tanh()

    def forward(self, X):
        X = self.a(self.bn1(self.deconv1(X)))
        X = self.a(self.bn2(self.deconv2(X)))
        X = self.a(self.deconv3(X))
        return X
```

[6]:
```
G = Generator().to(device)
D = Discriminator().to(device)

loss_fn = nn.BCELoss()
optimizerG = torch.optim.SGD(G.parameters(), lr=0.15)
optimizerD = torch.optim.SGD(D.parameters(), lr=0.15)

epochs = 300
CE_D = torch.zeros(epochs)
CE_G = torch.zeros(epochs)

CE_D = torch.zeros(epochs)
CE_G = torch.zeros(epochs)

for epoch in range(epochs):  # Fixed 'epochs' variable conflict
    for X, _ in minibatches:
```

```
        # Loss accumulation for real images
        D.zero_grad()
        X_real = X.to(device)
        Y_real = torch.ones(500).to(device)  # Match size of discriminator␣
↪output
        outD_real = D(X_real).squeeze()  # Ensure the output is [500]
        loss_real = loss_fn(outD_real, Y_real)

        # Loss accumulation for fake images
        z = torch.randn(500, 10, 1, 1).to(device)
        X_fake = G(z)
        Y_fake = torch.zeros(500).to(device)  # Match size of discriminator␣
↪output
        outD_fake = D(X_fake).squeeze()  # Ensure the output is [500]
        loss_fake = loss_fn(outD_fake, Y_fake)

        # Gradient descent part for Discriminator
        lossD = loss_real + loss_fake
        lossD.backward()
        optimizerD.step()

        # Training of the Generator
        G.zero_grad()
        z = torch.randn(500, 10, 1, 1).to(device)
        Y = torch.ones(500).to(device)  # Generator tries to fool the␣
↪discriminator
        outG = G(z)
        outD_fake_for_G = D(outG).squeeze()  # Ensure the output is [500]
        lossG = loss_fn(outD_fake_for_G, Y)
        lossG.backward()
        optimizerG.step()

    # Log the losses
    CE_D[epoch] = lossD.item()
    CE_G[epoch] = lossG.item()
    print(f"Epoch {epoch + 1}/5, Loss_D: {CE_D[epoch]:.4f}, Loss_G:␣
↪{CE_G[epoch]:.4f}")
```

```
Epoch 1/5, Loss_D: 1.2513, Loss_G: 0.8574
Epoch 2/5, Loss_D: 0.9193, Loss_G: 0.9029
Epoch 3/5, Loss_D: 1.5685, Loss_G: 1.4606
Epoch 4/5, Loss_D: 1.3883, Loss_G: 0.7935
Epoch 5/5, Loss_D: 1.3884, Loss_G: 0.6962
Epoch 6/5, Loss_D: 1.3695, Loss_G: 0.6923
Epoch 7/5, Loss_D: 1.3863, Loss_G: 0.6747
Epoch 8/5, Loss_D: 1.3698, Loss_G: 0.7114
Epoch 9/5, Loss_D: 1.4258, Loss_G: 0.6821
```

```
Epoch 10/5, Loss_D: 1.3442, Loss_G: 0.7249
Epoch 11/5, Loss_D: 1.3957, Loss_G: 0.6880
Epoch 12/5, Loss_D: 1.3917, Loss_G: 0.6919
Epoch 13/5, Loss_D: 1.3646, Loss_G: 0.6820
Epoch 14/5, Loss_D: 1.4296, Loss_G: 0.4888
Epoch 15/5, Loss_D: 1.3848, Loss_G: 0.6668
Epoch 16/5, Loss_D: 1.3609, Loss_G: 0.6816
Epoch 17/5, Loss_D: 1.4151, Loss_G: 0.4406
Epoch 18/5, Loss_D: 1.3633, Loss_G: 0.6622
Epoch 19/5, Loss_D: 1.3823, Loss_G: 0.5409
Epoch 20/5, Loss_D: 1.3627, Loss_G: 0.6435
Epoch 21/5, Loss_D: 1.3782, Loss_G: 0.5855
Epoch 22/5, Loss_D: 1.3848, Loss_G: 0.5990
Epoch 23/5, Loss_D: 1.3816, Loss_G: 0.6048
Epoch 24/5, Loss_D: 1.3776, Loss_G: 0.5965
Epoch 25/5, Loss_D: 1.3715, Loss_G: 0.6085
Epoch 26/5, Loss_D: 1.3803, Loss_G: 0.6021
Epoch 27/5, Loss_D: 1.3956, Loss_G: 0.5627
Epoch 28/5, Loss_D: 1.3776, Loss_G: 0.6014
Epoch 29/5, Loss_D: 1.3892, Loss_G: 0.5768
Epoch 30/5, Loss_D: 1.3791, Loss_G: 0.5911
Epoch 31/5, Loss_D: 1.3852, Loss_G: 0.5798
Epoch 32/5, Loss_D: 1.3743, Loss_G: 0.5958
Epoch 33/5, Loss_D: 1.3730, Loss_G: 0.5987
Epoch 34/5, Loss_D: 1.3770, Loss_G: 0.5861
Epoch 35/5, Loss_D: 1.3768, Loss_G: 0.5990
Epoch 36/5, Loss_D: 1.3772, Loss_G: 0.5946
Epoch 37/5, Loss_D: 1.3834, Loss_G: 0.5783
Epoch 38/5, Loss_D: 1.3647, Loss_G: 0.6242
Epoch 39/5, Loss_D: 1.3685, Loss_G: 0.6262
Epoch 40/5, Loss_D: 1.4248, Loss_G: 0.5845
Epoch 41/5, Loss_D: 1.3459, Loss_G: 0.6572
Epoch 42/5, Loss_D: 1.3650, Loss_G: 0.6574
Epoch 43/5, Loss_D: 1.3743, Loss_G: 0.6091
Epoch 44/5, Loss_D: 1.3561, Loss_G: 0.6897
Epoch 45/5, Loss_D: 1.4284, Loss_G: 0.6637
Epoch 46/5, Loss_D: 1.3613, Loss_G: 0.6127
Epoch 47/5, Loss_D: 1.3520, Loss_G: 0.7150
Epoch 48/5, Loss_D: 1.4904, Loss_G: 0.7516
Epoch 49/5, Loss_D: 1.3602, Loss_G: 0.7108
Epoch 50/5, Loss_D: 1.3799, Loss_G: 1.0669
Epoch 51/5, Loss_D: 1.3668, Loss_G: 0.7083
Epoch 52/5, Loss_D: 1.3552, Loss_G: 0.8154
Epoch 53/5, Loss_D: 1.3525, Loss_G: 0.7339
Epoch 54/5, Loss_D: 1.3717, Loss_G: 0.9669
Epoch 55/5, Loss_D: 1.3657, Loss_G: 0.9365
Epoch 56/5, Loss_D: 1.3422, Loss_G: 0.9598
Epoch 57/5, Loss_D: 1.3076, Loss_G: 1.1100
```

```
Epoch 58/5, Loss_D: 1.2580, Loss_G: 1.2688
Epoch 59/5, Loss_D: 1.2936, Loss_G: 1.2003
Epoch 60/5, Loss_D: 1.0670, Loss_G: 2.4761
Epoch 61/5, Loss_D: 1.3173, Loss_G: 1.0264
Epoch 62/5, Loss_D: 1.0819, Loss_G: 1.9965
Epoch 63/5, Loss_D: 1.1251, Loss_G: 2.5587
Epoch 64/5, Loss_D: 1.2496, Loss_G: 1.9687
Epoch 65/5, Loss_D: 1.2288, Loss_G: 2.2692
Epoch 66/5, Loss_D: 1.6370, Loss_G: 1.7244
Epoch 67/5, Loss_D: 1.2139, Loss_G: 1.0658
Epoch 68/5, Loss_D: 1.1044, Loss_G: 0.9514
Epoch 69/5, Loss_D: 1.1391, Loss_G: 1.0113
Epoch 70/5, Loss_D: 1.0380, Loss_G: 1.0734
Epoch 71/5, Loss_D: 1.6448, Loss_G: 2.2069
Epoch 72/5, Loss_D: 1.4438, Loss_G: 0.8124
Epoch 73/5, Loss_D: 1.1667, Loss_G: 0.7803
Epoch 74/5, Loss_D: 1.0227, Loss_G: 1.4668
Epoch 75/5, Loss_D: 0.7743, Loss_G: 1.4057
Epoch 76/5, Loss_D: 2.2650, Loss_G: 4.5505
Epoch 77/5, Loss_D: 0.7346, Loss_G: 1.5829
Epoch 78/5, Loss_D: 0.6584, Loss_G: 2.2176
Epoch 79/5, Loss_D: 0.7663, Loss_G: 1.8077
Epoch 80/5, Loss_D: 0.3434, Loss_G: 2.8086
Epoch 81/5, Loss_D: 0.4892, Loss_G: 1.7794
Epoch 82/5, Loss_D: 0.3765, Loss_G: 2.6814
Epoch 83/5, Loss_D: 0.2863, Loss_G: 2.4509
Epoch 84/5, Loss_D: 0.8831, Loss_G: 1.4525
Epoch 85/5, Loss_D: 0.3866, Loss_G: 3.4847
Epoch 86/5, Loss_D: 0.3169, Loss_G: 3.4332
Epoch 87/5, Loss_D: 0.3135, Loss_G: 2.5352
Epoch 88/5, Loss_D: 0.5741, Loss_G: 6.9118
Epoch 89/5, Loss_D: 0.2860, Loss_G: 2.2650
Epoch 90/5, Loss_D: 0.1409, Loss_G: 3.1247
Epoch 91/5, Loss_D: 1.1456, Loss_G: 0.7316
Epoch 92/5, Loss_D: 0.1952, Loss_G: 2.7444
Epoch 93/5, Loss_D: 0.2732, Loss_G: 2.4155
Epoch 94/5, Loss_D: 0.1927, Loss_G: 2.8358
Epoch 95/5, Loss_D: 0.0733, Loss_G: 5.0344
Epoch 96/5, Loss_D: 0.1405, Loss_G: 3.5935
Epoch 97/5, Loss_D: 0.0636, Loss_G: 4.1508
Epoch 98/5, Loss_D: 0.0193, Loss_G: 5.3525
Epoch 99/5, Loss_D: 0.1054, Loss_G: 4.2215
Epoch 100/5, Loss_D: 0.0572, Loss_G: 4.2389
Epoch 101/5, Loss_D: 0.0209, Loss_G: 5.9352
Epoch 102/5, Loss_D: 0.1640, Loss_G: 4.8878
Epoch 103/5, Loss_D: 0.0173, Loss_G: 6.3602
Epoch 104/5, Loss_D: 0.1897, Loss_G: 6.7575
Epoch 105/5, Loss_D: 0.0149, Loss_G: 5.9525
```

```
Epoch 106/5, Loss_D: 0.0112, Loss_G: 7.8454
Epoch 107/5, Loss_D: 0.0268, Loss_G: 5.2503
Epoch 108/5, Loss_D: 0.0183, Loss_G: 5.2904
Epoch 109/5, Loss_D: 0.0169, Loss_G: 6.7224
Epoch 110/5, Loss_D: 0.0166, Loss_G: 9.6179
Epoch 111/5, Loss_D: 0.0484, Loss_G: 4.7709
Epoch 112/5, Loss_D: 0.0053, Loss_G: 8.4304
Epoch 113/5, Loss_D: 0.0193, Loss_G: 5.2699
Epoch 114/5, Loss_D: 0.0254, Loss_G: 4.7166
Epoch 115/5, Loss_D: 0.0172, Loss_G: 4.8911
Epoch 116/5, Loss_D: 0.0438, Loss_G: 4.3580
Epoch 117/5, Loss_D: 0.0069, Loss_G: 8.5209
Epoch 118/5, Loss_D: 0.0190, Loss_G: 4.9402
Epoch 119/5, Loss_D: 0.0093, Loss_G: 6.0606
Epoch 120/5, Loss_D: 0.0168, Loss_G: 5.2709
Epoch 121/5, Loss_D: 0.0110, Loss_G: 5.7668
Epoch 122/5, Loss_D: 0.0115, Loss_G: 5.5222
Epoch 123/5, Loss_D: 0.0314, Loss_G: 4.9777
Epoch 124/5, Loss_D: 0.0329, Loss_G: 4.9307
Epoch 125/5, Loss_D: 0.0048, Loss_G: 6.9163
Epoch 126/5, Loss_D: 0.0215, Loss_G: 5.2360
Epoch 127/5, Loss_D: 0.0033, Loss_G: 6.8125
Epoch 128/5, Loss_D: 0.0073, Loss_G: 5.7900
Epoch 129/5, Loss_D: 0.0048, Loss_G: 6.4691
Epoch 130/5, Loss_D: 0.0091, Loss_G: 5.6759
Epoch 131/5, Loss_D: 0.0042, Loss_G: 6.9506
Epoch 132/5, Loss_D: 0.0139, Loss_G: 5.3156
Epoch 133/5, Loss_D: 0.0155, Loss_G: 5.1536
Epoch 134/5, Loss_D: 0.0079, Loss_G: 6.0229
Epoch 135/5, Loss_D: 0.0100, Loss_G: 6.1403
Epoch 136/5, Loss_D: 0.0072, Loss_G: 6.3298
Epoch 137/5, Loss_D: 0.0107, Loss_G: 5.7312
Epoch 138/5, Loss_D: 0.0116, Loss_G: 5.3548
Epoch 139/5, Loss_D: 0.0114, Loss_G: 5.2346
Epoch 140/5, Loss_D: 0.0079, Loss_G: 5.5755
Epoch 141/5, Loss_D: 0.0057, Loss_G: 6.3335
Epoch 142/5, Loss_D: 0.0051, Loss_G: 6.4232
Epoch 143/5, Loss_D: 0.0037, Loss_G: 7.4224
Epoch 144/5, Loss_D: 0.0056, Loss_G: 7.0824
Epoch 145/5, Loss_D: 0.0118, Loss_G: 5.5106
Epoch 146/5, Loss_D: 0.0026, Loss_G: 7.9610
Epoch 147/5, Loss_D: 0.0131, Loss_G: 5.5520
Epoch 148/5, Loss_D: 0.0070, Loss_G: 6.4049
Epoch 149/5, Loss_D: 0.0068, Loss_G: 5.7071
Epoch 150/5, Loss_D: 0.0082, Loss_G: 5.7396
Epoch 151/5, Loss_D: 0.0029, Loss_G: 8.5752
Epoch 152/5, Loss_D: 0.0042, Loss_G: 6.7313
Epoch 153/5, Loss_D: 0.0052, Loss_G: 7.6288
```

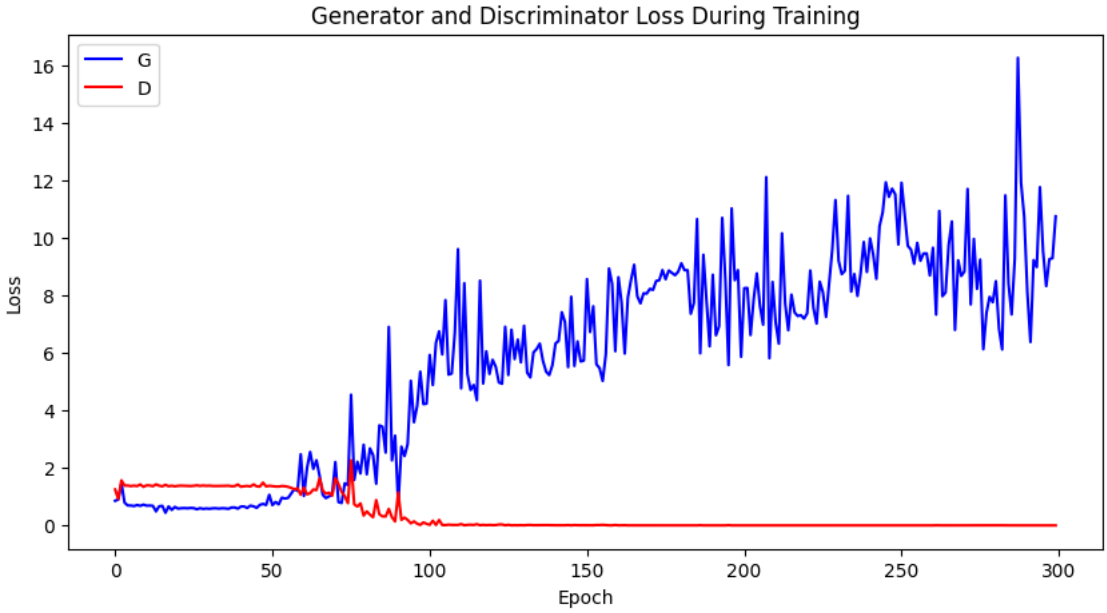
```
Epoch 154/5, Loss_D: 0.0141, Loss_G: 5.6089
Epoch 155/5, Loss_D: 0.0130, Loss_G: 5.4777
Epoch 156/5, Loss_D: 0.0193, Loss_G: 5.0280
Epoch 157/5, Loss_D: 0.0108, Loss_G: 5.9647
Epoch 158/5, Loss_D: 0.0038, Loss_G: 8.9435
Epoch 159/5, Loss_D: 0.0028, Loss_G: 8.4096
Epoch 160/5, Loss_D: 0.0153, Loss_G: 6.0637
Epoch 161/5, Loss_D: 0.0028, Loss_G: 8.6399
Epoch 162/5, Loss_D: 0.0052, Loss_G: 7.7585
Epoch 163/5, Loss_D: 0.0076, Loss_G: 5.9825
Epoch 164/5, Loss_D: 0.0022, Loss_G: 7.9352
Epoch 165/5, Loss_D: 0.0020, Loss_G: 8.5416
Epoch 166/5, Loss_D: 0.0015, Loss_G: 9.0738
Epoch 167/5, Loss_D: 0.0013, Loss_G: 7.9789
Epoch 168/5, Loss_D: 0.0012, Loss_G: 7.7298
Epoch 169/5, Loss_D: 0.0012, Loss_G: 8.0801
Epoch 170/5, Loss_D: 0.0009, Loss_G: 8.0709
Epoch 171/5, Loss_D: 0.0010, Loss_G: 8.2352
Epoch 172/5, Loss_D: 0.0009, Loss_G: 8.1986
Epoch 173/5, Loss_D: 0.0006, Loss_G: 8.5079
Epoch 174/5, Loss_D: 0.0006, Loss_G: 8.5375
Epoch 175/5, Loss_D: 0.0004, Loss_G: 8.8890
Epoch 176/5, Loss_D: 0.0005, Loss_G: 8.5697
Epoch 177/5, Loss_D: 0.0004, Loss_G: 8.8738
Epoch 178/5, Loss_D: 0.0005, Loss_G: 8.7922
Epoch 179/5, Loss_D: 0.0004, Loss_G: 8.7107
Epoch 180/5, Loss_D: 0.0004, Loss_G: 8.8304
Epoch 181/5, Loss_D: 0.0004, Loss_G: 9.1243
Epoch 182/5, Loss_D: 0.0004, Loss_G: 8.8865
Epoch 183/5, Loss_D: 0.0004, Loss_G: 8.8827
Epoch 184/5, Loss_D: 0.0011, Loss_G: 7.3664
Epoch 185/5, Loss_D: 0.0017, Loss_G: 7.7443
Epoch 186/5, Loss_D: 0.0013, Loss_G: 10.6623
Epoch 187/5, Loss_D: 0.0076, Loss_G: 5.9901
Epoch 188/5, Loss_D: 0.0011, Loss_G: 9.4223
Epoch 189/5, Loss_D: 0.0013, Loss_G: 7.8142
Epoch 190/5, Loss_D: 0.0036, Loss_G: 6.2364
Epoch 191/5, Loss_D: 0.0011, Loss_G: 8.7257
Epoch 192/5, Loss_D: 0.0044, Loss_G: 6.6184
Epoch 193/5, Loss_D: 0.0027, Loss_G: 6.9353
Epoch 194/5, Loss_D: 0.0014, Loss_G: 10.7071
Epoch 195/5, Loss_D: 0.0010, Loss_G: 8.5494
Epoch 196/5, Loss_D: 0.0088, Loss_G: 5.5831
Epoch 197/5, Loss_D: 0.0007, Loss_G: 11.0357
Epoch 198/5, Loss_D: 0.0008, Loss_G: 8.5337
Epoch 199/5, Loss_D: 0.0010, Loss_G: 8.8855
Epoch 200/5, Loss_D: 0.0046, Loss_G: 5.8700
Epoch 201/5, Loss_D: 0.0012, Loss_G: 8.2583
```

```
Epoch 202/5, Loss_D: 0.0013, Loss_G: 8.2653
Epoch 203/5, Loss_D: 0.0027, Loss_G: 6.6275
Epoch 204/5, Loss_D: 0.0014, Loss_G: 7.8765
Epoch 205/5, Loss_D: 0.0011, Loss_G: 8.7727
Epoch 206/5, Loss_D: 0.0016, Loss_G: 7.6417
Epoch 207/5, Loss_D: 0.0023, Loss_G: 6.9881
Epoch 208/5, Loss_D: 0.0011, Loss_G: 12.1254
Epoch 209/5, Loss_D: 0.0058, Loss_G: 5.8209
Epoch 210/5, Loss_D: 0.0011, Loss_G: 8.4752
Epoch 211/5, Loss_D: 0.0036, Loss_G: 7.0446
Epoch 212/5, Loss_D: 0.0068, Loss_G: 6.3261
Epoch 213/5, Loss_D: 0.0015, Loss_G: 10.1697
Epoch 214/5, Loss_D: 0.0018, Loss_G: 7.7367
Epoch 215/5, Loss_D: 0.0070, Loss_G: 6.7955
Epoch 216/5, Loss_D: 0.0024, Loss_G: 8.0334
Epoch 217/5, Loss_D: 0.0019, Loss_G: 7.4159
Epoch 218/5, Loss_D: 0.0017, Loss_G: 7.2896
Epoch 219/5, Loss_D: 0.0017, Loss_G: 7.3186
Epoch 220/5, Loss_D: 0.0021, Loss_G: 7.2001
Epoch 221/5, Loss_D: 0.0015, Loss_G: 7.3806
Epoch 222/5, Loss_D: 0.0017, Loss_G: 8.8727
Epoch 223/5, Loss_D: 0.0012, Loss_G: 7.5703
Epoch 224/5, Loss_D: 0.0032, Loss_G: 7.0262
Epoch 225/5, Loss_D: 0.0019, Loss_G: 8.4848
Epoch 226/5, Loss_D: 0.0016, Loss_G: 8.1207
Epoch 227/5, Loss_D: 0.0042, Loss_G: 7.2578
Epoch 228/5, Loss_D: 0.0014, Loss_G: 8.4316
Epoch 229/5, Loss_D: 0.0006, Loss_G: 9.6262
Epoch 230/5, Loss_D: 0.0006, Loss_G: 11.3273
Epoch 231/5, Loss_D: 0.0004, Loss_G: 9.2156
Epoch 232/5, Loss_D: 0.0005, Loss_G: 8.7472
Epoch 233/5, Loss_D: 0.0004, Loss_G: 8.8656
Epoch 234/5, Loss_D: 0.0003, Loss_G: 11.4740
Epoch 235/5, Loss_D: 0.0007, Loss_G: 8.1393
Epoch 236/5, Loss_D: 0.0005, Loss_G: 8.7506
Epoch 237/5, Loss_D: 0.0007, Loss_G: 7.9874
Epoch 238/5, Loss_D: 0.0005, Loss_G: 8.7829
Epoch 239/5, Loss_D: 0.0003, Loss_G: 9.8680
Epoch 240/5, Loss_D: 0.0005, Loss_G: 8.8221
Epoch 241/5, Loss_D: 0.0003, Loss_G: 9.9893
Epoch 242/5, Loss_D: 0.0004, Loss_G: 9.4279
Epoch 243/5, Loss_D: 0.0008, Loss_G: 8.5835
Epoch 244/5, Loss_D: 0.0003, Loss_G: 10.4151
Epoch 245/5, Loss_D: 0.0002, Loss_G: 10.8926
Epoch 246/5, Loss_D: 0.0004, Loss_G: 11.9436
Epoch 247/5, Loss_D: 0.0002, Loss_G: 11.4351
Epoch 248/5, Loss_D: 0.0002, Loss_G: 11.7223
Epoch 249/5, Loss_D: 0.0003, Loss_G: 11.5250
```

```
Epoch 250/5, Loss_D: 0.0004, Loss_G: 9.7778
Epoch 251/5, Loss_D: 0.0003, Loss_G: 11.9282
Epoch 252/5, Loss_D: 0.0002, Loss_G: 10.8509
Epoch 253/5, Loss_D: 0.0002, Loss_G: 9.7310
Epoch 254/5, Loss_D: 0.0002, Loss_G: 9.6030
Epoch 255/5, Loss_D: 0.0003, Loss_G: 9.1040
Epoch 256/5, Loss_D: 0.0002, Loss_G: 9.8355
Epoch 257/5, Loss_D: 0.0003, Loss_G: 9.2118
Epoch 258/5, Loss_D: 0.0003, Loss_G: 9.4649
Epoch 259/5, Loss_D: 0.0002, Loss_G: 9.4675
Epoch 260/5, Loss_D: 0.0005, Loss_G: 8.7015
Epoch 261/5, Loss_D: 0.0004, Loss_G: 9.6665
Epoch 262/5, Loss_D: 0.0048, Loss_G: 7.3385
Epoch 263/5, Loss_D: 0.0005, Loss_G: 10.9444
Epoch 264/5, Loss_D: 0.0010, Loss_G: 7.9796
Epoch 265/5, Loss_D: 0.0012, Loss_G: 8.1115
Epoch 266/5, Loss_D: 0.0013, Loss_G: 9.7707
Epoch 267/5, Loss_D: 0.0008, Loss_G: 10.5804
Epoch 268/5, Loss_D: 0.0020, Loss_G: 6.8051
Epoch 269/5, Loss_D: 0.0004, Loss_G: 9.2292
Epoch 270/5, Loss_D: 0.0009, Loss_G: 8.6834
Epoch 271/5, Loss_D: 0.0005, Loss_G: 8.8412
Epoch 272/5, Loss_D: 0.0003, Loss_G: 11.7134
Epoch 273/5, Loss_D: 0.0009, Loss_G: 7.6921
Epoch 274/5, Loss_D: 0.0004, Loss_G: 9.9729
Epoch 275/5, Loss_D: 0.0007, Loss_G: 8.2331
Epoch 276/5, Loss_D: 0.0004, Loss_G: 9.2530
Epoch 277/5, Loss_D: 0.0039, Loss_G: 6.1342
Epoch 278/5, Loss_D: 0.0011, Loss_G: 7.3943
Epoch 279/5, Loss_D: 0.0008, Loss_G: 7.9578
Epoch 280/5, Loss_D: 0.0007, Loss_G: 7.7700
Epoch 281/5, Loss_D: 0.0005, Loss_G: 8.5016
Epoch 282/5, Loss_D: 0.0016, Loss_G: 6.8473
Epoch 283/5, Loss_D: 0.0039, Loss_G: 6.1262
Epoch 284/5, Loss_D: 0.0003, Loss_G: 11.4898
Epoch 285/5, Loss_D: 0.0011, Loss_G: 8.4304
Epoch 286/5, Loss_D: 0.0023, Loss_G: 7.3441
Epoch 287/5, Loss_D: 0.0004, Loss_G: 9.2789
Epoch 288/5, Loss_D: 0.0009, Loss_G: 16.2790
Epoch 289/5, Loss_D: 0.0004, Loss_G: 11.9312
Epoch 290/5, Loss_D: 0.0003, Loss_G: 10.7473
Epoch 291/5, Loss_D: 0.0008, Loss_G: 8.1161
Epoch 292/5, Loss_D: 0.0036, Loss_G: 6.3841
Epoch 293/5, Loss_D: 0.0007, Loss_G: 9.2374
Epoch 294/5, Loss_D: 0.0003, Loss_G: 8.9915
Epoch 295/5, Loss_D: 0.0003, Loss_G: 11.7749
Epoch 296/5, Loss_D: 0.0003, Loss_G: 9.5412
Epoch 297/5, Loss_D: 0.0005, Loss_G: 8.3316
```

```
Epoch 298/5, Loss_D: 0.0003, Loss_G: 9.2667
Epoch 299/5, Loss_D: 0.0004, Loss_G: 9.3035
Epoch 300/5, Loss_D: 0.0003, Loss_G: 10.7615
```

[7]:
```
plt.figure(figsize=(10,5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(CE_G.numpy(), label="G", color='blue')
plt.plot(CE_D.numpy(), label="D", color='red')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[8]:
```
G.eval()  # Set the generator to evaluation mode
generated = G(torch.randn(10, 10, 1, 1).to(device))

plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(1, 10, i + 1)
    # Remove channel dimension by indexing [0]
    plt.imshow(generated[i, 0].cpu().detach().numpy(), cmap='gray')
    plt.axis('off')  # Hide axes for better visualization
plt.show()
```

[ ]: