Name : Kiran Shinde

PRN : 202201040091

Div : A

Batch : GEN AI 2

Practical Experiment 2 :

1. Build a simple AE model for Dimensionality Reduction and Denoising.

Code :

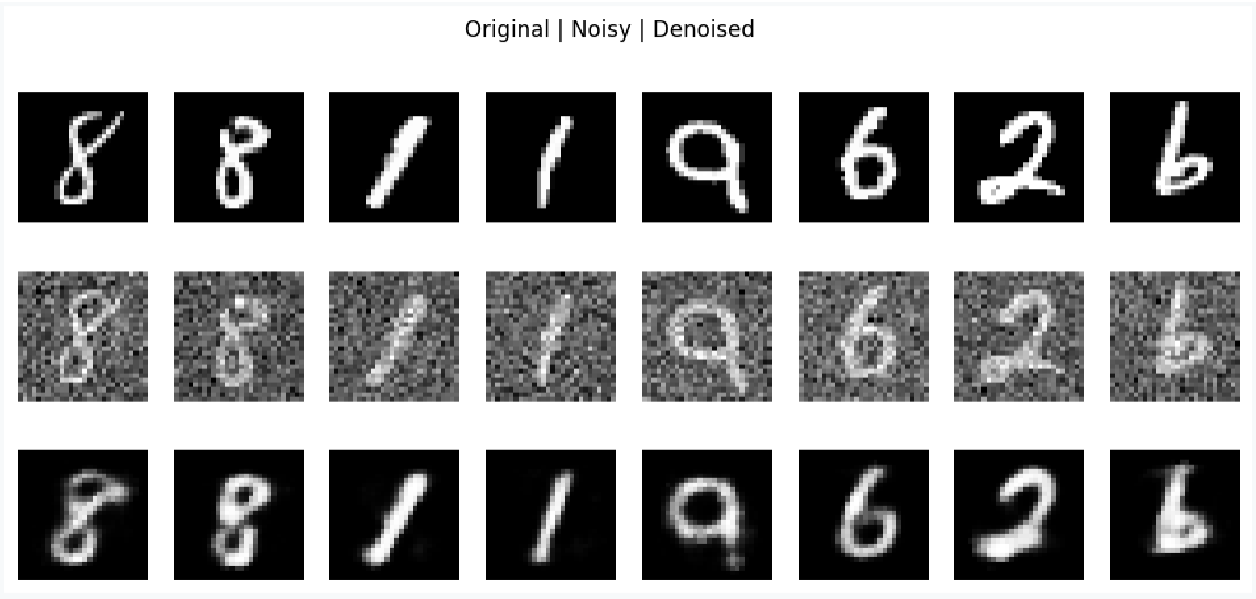
```
# ✅ Practical 1: Autoencoder with MNIST

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

  Load MNIST
transform = transforms.ToTensor()
train_dataset = datasets.MNIST(root="./data", train=True, download=True,
transform=transform)
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)

  Autoencoder
class Autoencoder(nn.Module):
  def__init__(self):
    super(Autoencoder, self).__init__()
    self.encoder = nn.Sequential(
      nn.Flatten(),
      nn.Linear(28*28, 128),
      nn.ReLU(),
```

```
            nn.Linear(128, 32)    # compressed representation
        )
        self.decoder = nn.Sequential(
            nn.Linear(32, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid(),
            nn.Unflatten(1, (1, 28, 28))
        )
    def forward(self, x):
        z = self.encoder(x)
        return self.decoder(z)

device = "cuda" if torch.cuda.is_available() else "cpu" model
= Autoencoder().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

   Train AE
for epoch in range(2):      increase epochs for better results
    for imgs, _ in train_loader:
        noisy_imgs = imgs + 0.3*torch.randn_like(imgs)       add noise
        noisy_imgs, imgs = noisy_imgs.to(device), imgs.to(device)

        outputs = model(noisy_imgs)
        loss = criterion(outputs, imgs)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print(f"Epoch [{epoch+1}/2], Loss: {loss.item():.4f}")

#Visualize original, noisy, denoised
imgs, _ = next(iter(train_loader))
```

```
noisy_imgs = imgs + 0.3*torch.randn_like(imgs)
with torch.no_grad():
    outputs = model(noisy_imgs.to(device)).cpu()

fig, axes = plt.subplots(3, 8, figsize=(12,5))
for i in range(8):
    axes[0,i].imshow(imgs[i][0], cmap="gray"); axes[0,i].axis("off")
    axes[1,i].imshow(noisy_imgs[i][0], cmap="gray"); axes[1,i].axis("off")
    axes[2,i].imshow(outputs[i][0], cmap="gray"); axes[2,i].axis("off")
plt.suptitle("Original | Noisy | Denoised")
plt.show()
```

Output :

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 60.3MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.70MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 14.8MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 5.14MB/s]
Epoch [1/2], Loss: 0.0283
Epoch [2/2], Loss: 0.0196
```

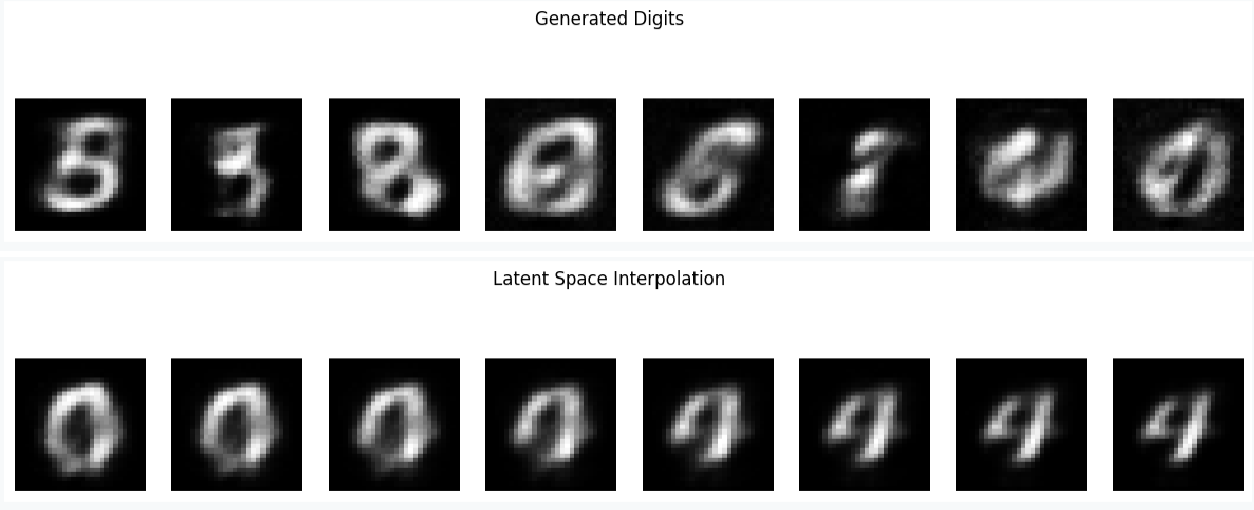


2. Generate realistic faces or interpolate between facial features for creative applications in entertainment and design using VAE.

```
# ✅ Practical 2: VAE with MNIST

import torch.nn.functional as F

  VAE Model
class VAE(nn.Module):
   def _init_(self, latent_dim=16):
      super(VAE, self)._init_()
      self.fc1 = nn.Linear(28*28, 128)
      self.fc21 = nn.Linear(128, latent_dim)      mean
      self.fc22 = nn.Linear(128, latent_dim)      logvar
      self.fc3 = nn.Linear(latent_dim, 128)
      self.fc4 = nn.Linear(128, 28*28)

   def encode(self, x):
      h1 = F.relu(self.fc1(x))
      return self.fc21(h1), self.fc22(h1)

   def reparameterize(self, mu, logvar):
      std = torch.exp(0.5*logvar)
      eps = torch.randn_like(std)
      return mu + eps*std

   def decode(self, z):
      h3 = F.relu(self.fc3(z))
      return torch.sigmoid(self.fc4(h3))

   def forward(self, x):
      mu, logvar = self.encode(x.view(-1, 28*28))
      z = self.reparameterize(mu, logvar)
      return self.decode(z), mu, logvar
```

```
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 28*28), reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD

vae = VAE(latent_dim=16).to(device)
optimizer = optim.Adam(vae.parameters(), lr=1e-3)

    Train VAE
for epoch in range(2):
    for imgs, _ in train_loader:
        imgs = imgs.to(device)
        recon, mu, logvar = vae(imgs)
        loss = loss_function(recon, imgs, mu, logvar)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print(f"Epoch [{epoch+1}/6], Loss: {loss.item():.4f}")

#Generate new digits
with torch.no_grad():
    z = torch.randn(8, 16).to(device)
    samples = vae.decode(z).cpu().view(-1, 1, 28, 28)

fig, axes = plt.subplots(1, 8, figsize=(15,3))
for i in range(8):
    axes[i].imshow(samples[i][0], cmap="gray")
    axes[i].axis("off")
plt.suptitle("Generated Digits")
plt.show()

    Interpolation
with torch.no_grad():
```

```
    z1, z2 = torch.randn(1, 16).to(device), torch.randn(1, 16).to(device)
    interpolations = []
    for alpha in torch.linspace(0, 1, 8):
        z = (1-alpha)*z1 + alpha*z2
        interpolations.append(vae.decode(z).cpu().view(28,28))
fig, axes = plt.subplots(1, 8, figsize=(15,3))
for i in range(8):
    axes[i].imshow(interpolations[i], cmap="gray")
    axes[i].axis("off")
plt.suptitle("Latent Space Interpolation")
plt.show()
```



Code Link :

https://colab.research.google.com/drive/1akrpMdsJnsegwUWrkC2exwu6pgZUuD_j?usp=sharing