Latent Space Representation using AE & VAE

Objective

The objective of this module is to design and evaluate Autoencoder (AE) and Variational Autoencoder (VAE) models on the MNIST dataset. The focus is to:

- Implement AE and VAE architectures.
- Visualize learned latent spaces using PCA/t-SNE.
- Compare reconstruction quality of both models.
- Explore latent space arithmetic to observe generative capabilities.

INTRODUCTION

Autoencoders are unsupervised neural network models that aim to learn efficient data encodings by compressing the input into a latent representation and then reconstructing it. Variational Autoencoders (VAEs), a probabilistic extension of AEs, not only learn a compressed representation but also impose a distribution on the latent space, enabling generation of new data samples.

Model Architectures

Autoencoder (AE)

- **Encoder:** Fully connected layers: 784 (28x28) → 128 → 2 (latent vector)
- **Decoder:** Fully connected layers: 2 → 128 → 784
- **Activation Functions:** ReLU in hidden layers, Sigmoid in the output layer
- **Loss Function:** Binary Cross-Entropy (BCE)

Variational Autoencoder (VAE)

- **Encoder:** 784 → 128 → [μ, log(σ^2)]

- **Sampling Function:** $z = \mu + \sigma \cdot \varepsilon$, where $\varepsilon \sim \mathbb{N}(0, 1)$ (Reparameterization Trick)
- **Decoder:** $2 \rightarrow 128 \rightarrow 784$
- **Loss Function:** Total Loss = BCE + KL Divergence (to encourage latent space to follow a standard normal distribution)

Training Results

AE Loss Curve

The AE model typically converges quickly due to the deterministic nature of encoding.

The BCE loss decreases steadily as the model learns to reconstruct digits.

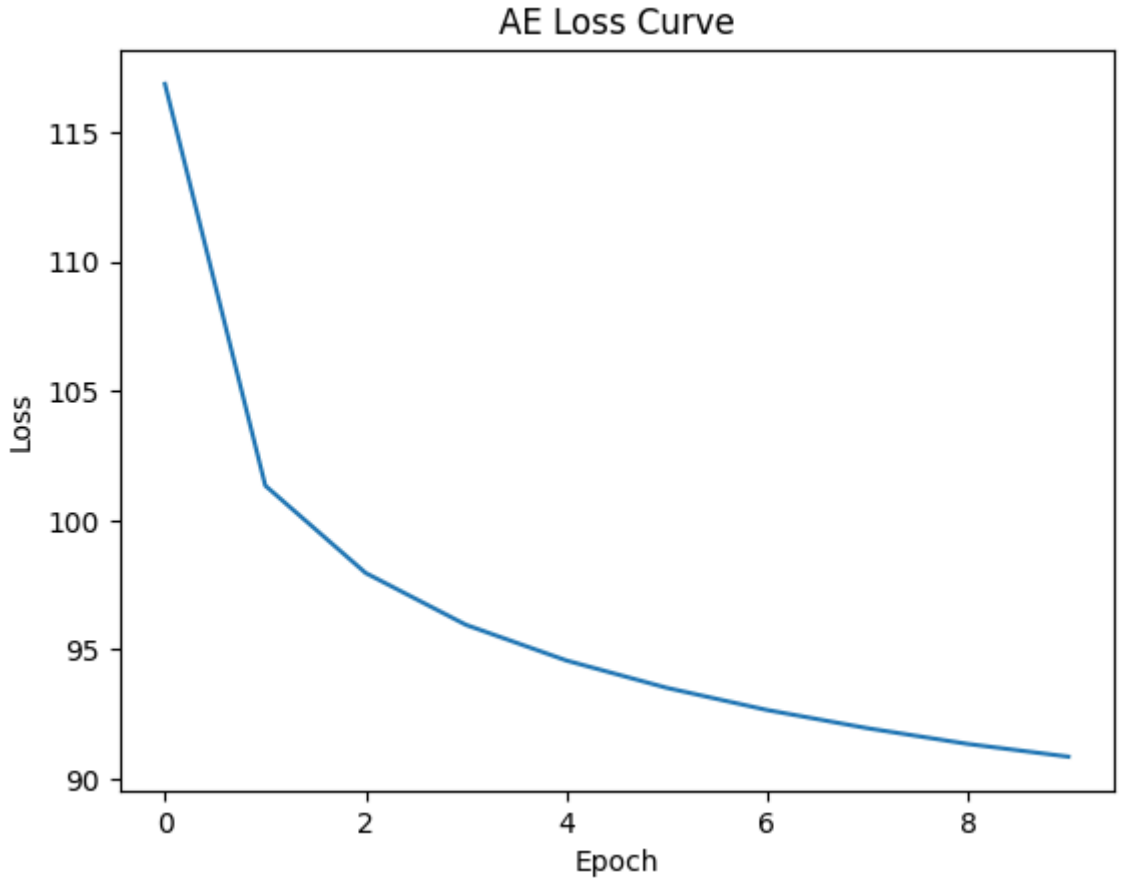
CODE

```
#AE
```

```
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

# Autoencoder architecture
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, 2)
        )
        self.decoder = nn.Sequential(
            nn.Linear(2, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid()
```

```
        )

    def forward(self, x):
        z = self.encoder(x)
        x_hat = self.decoder(z)
        return x_hat

# Data loading
transform = transforms.ToTensor()
train_data = datasets.MNIST(root='./data', train=True,
download=True, transform=transform)
test_data = datasets.MNIST(root='./data', train=False,
download=True, transform=transform)

train_loader = DataLoader(train_data, batch_size=128,
shuffle=True)
test_loader = DataLoader(test_data, batch_size=128,
shuffle=False)

device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
model = Autoencoder().to(device)

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

# Training loop
losses = []
for epoch in range(10):
    epoch_loss = 0
    for images, _ in train_loader:
        images = images.to(device)
        output = model(images)
        loss = criterion(output, images.view(-1, 28*28))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    losses.append(epoch_loss)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss:.2f}")

torch.save(model.state_dict(), "ae_model.pth")
```

```
# Save loss curve
plt.plot(losses)
plt.title("AE Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
plt.close()
```



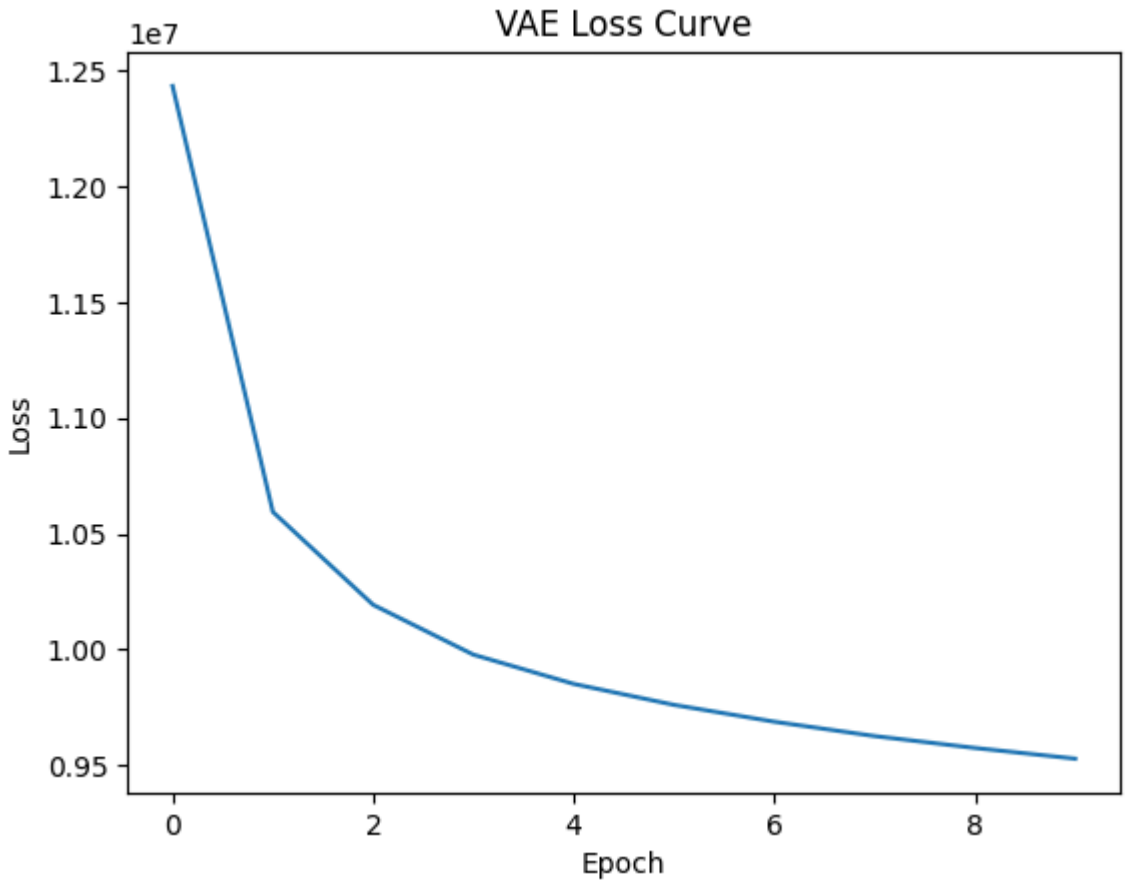
VAE Loss Curve

- Includes reconstruction loss (BCE) and KL divergence.
- The loss may appear higher initially due to the regularization effect of KL divergence.
- Over time, VAE balances reconstruction accuracy and latent distribution regularity.

CODE

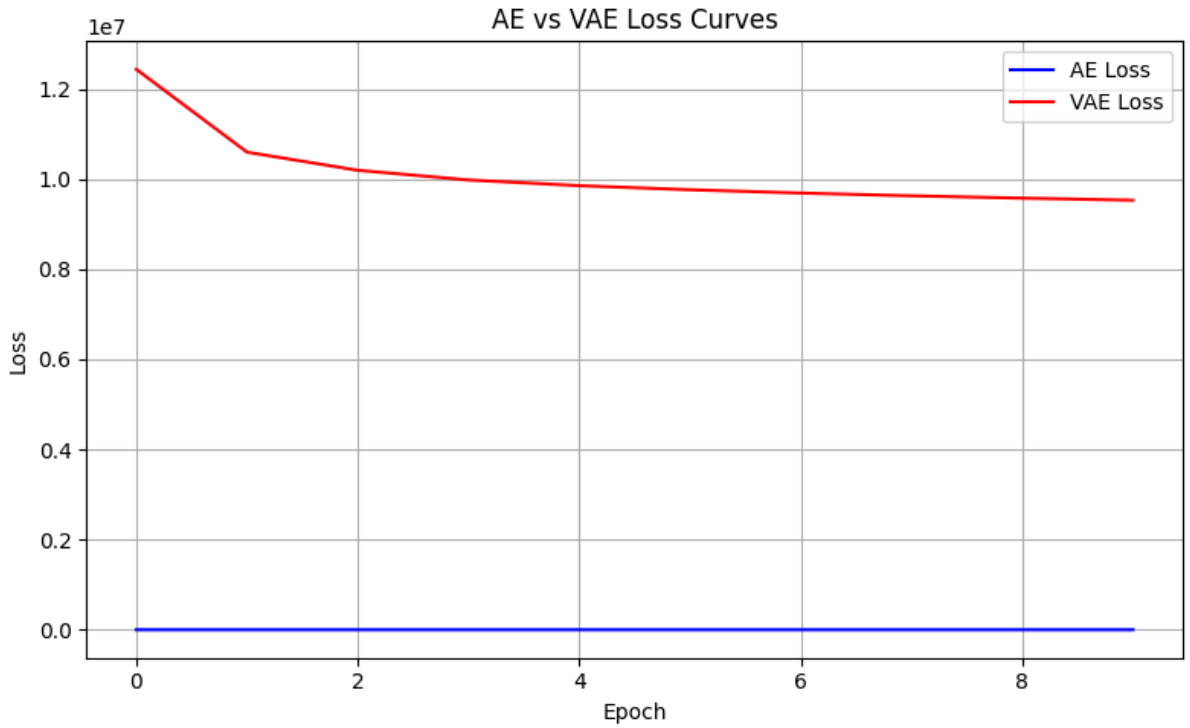
```python
#VAE
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        self.encoder_fc = nn.Linear(784, 128)
        self.mu = nn.Linear(128, 2)
        self.logvar = nn.Linear(128, 2)
        self.decoder = nn.Sequential(
            nn.Linear(2, 128),
            nn.ReLU(),
            nn.Linear(128, 784),
            nn.Sigmoid()
        )

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

    def forward(self, x):
        x = x.view(-1, 784)
        h = torch.relu(self.encoder_fc(x))
        mu = self.mu(h)
        logvar = self.logvar(h)
        z = self.reparameterize(mu, logvar)
        x_recon = self.decoder(z)
        return x_recon, mu, logvar

def loss_function(recon_x, x, mu, logvar):
    BCE = nn.functional.binary_cross_entropy(recon_x, x.view(-1, 784),
reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD

# Data loading
transform = transforms.ToTensor()
```

```
train_data = datasets.MNIST(root='./data', train=True, download=True,
transform=transform)
test_data = datasets.MNIST(root='./data', train=False, download=True,
transform=transform)

train_loader = DataLoader(train_data, batch_size=128, shuffle=True)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
vae = VAE().to(device)
optimizer = optim.Adam(vae.parameters(), lr=1e-3)

# Training loop
vae_losses = []
for epoch in range(10):
    epoch_loss = 0
    for images, _ in train_loader:
        images = images.to(device)
        recon_batch, mu, logvar = vae(images)
        loss = loss_function(recon_batch, images, mu, logvar)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    vae_losses.append(epoch_loss)
    print(f"Epoch {epoch+1}, VAE Loss: {epoch_loss:.2f}")

torch.save(vae.state_dict(), "vae_model.pth")

# Save loss curve
plt.plot(vae_losses)
plt.title("VAE Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
plt.close()
```



Comparison Between AE and VAE Loss Curves

Autoencoder (AE) loss typically decreases smoothly, as it only minimizes reconstruction error. In contrast, Variational Autoencoder (VAE) loss includes both reconstruction loss and KL divergence, making the curve more complex. Early in training, VAE's KL term may dominate, slowing total loss reduction. Over time, both losses converge, but VAE balances compression with a regularized latent space, often resulting in higher total loss but better generative capability.



Latent Space Visualization

Method: t-SNE (t-distributed Stochastic Neighbor Embedding)

- t-SNE reduces high-dimensional latent vectors to 2D for visualization.
- Each point is color-coded based on the true digit class (0–9).

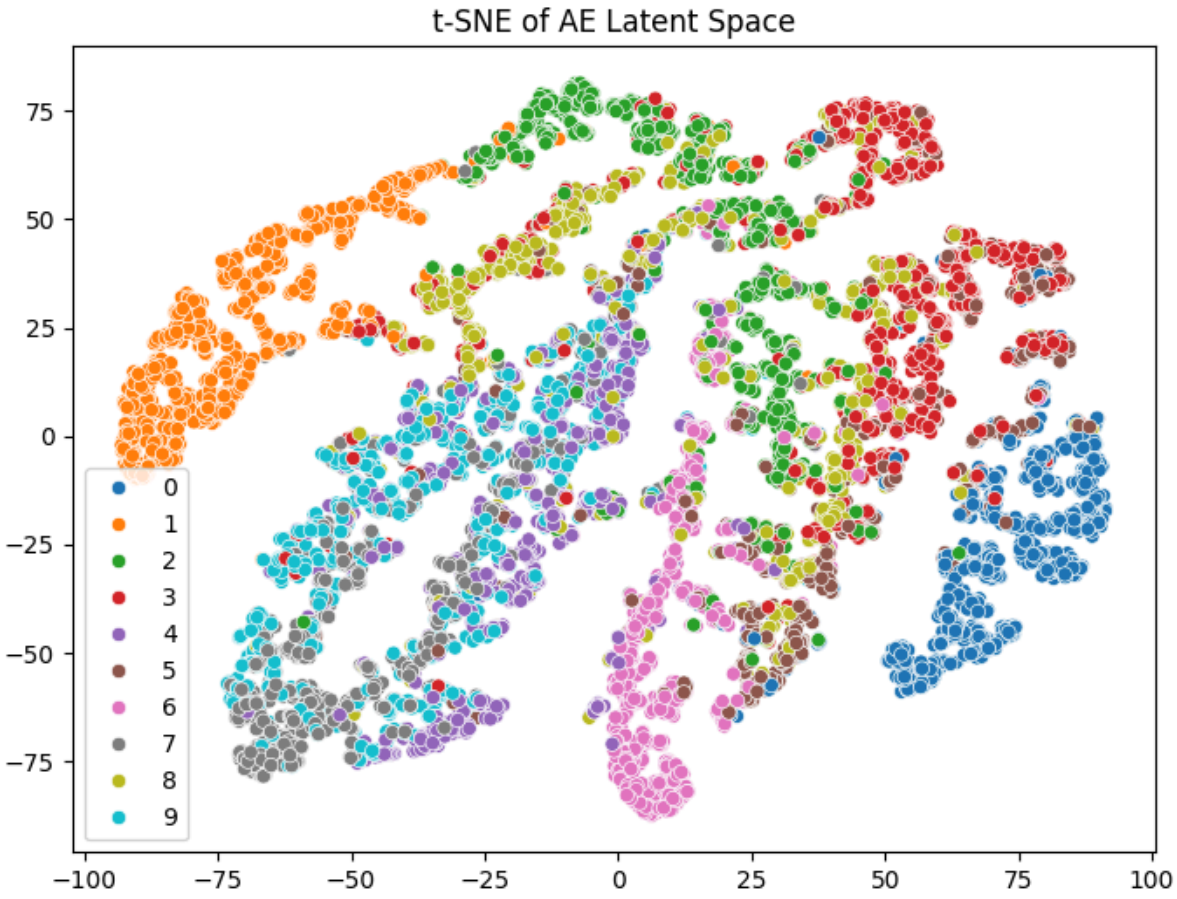
AE Latent Space

- Clusters form, but overlap is common between similar digits (e.g., 3 & 8).

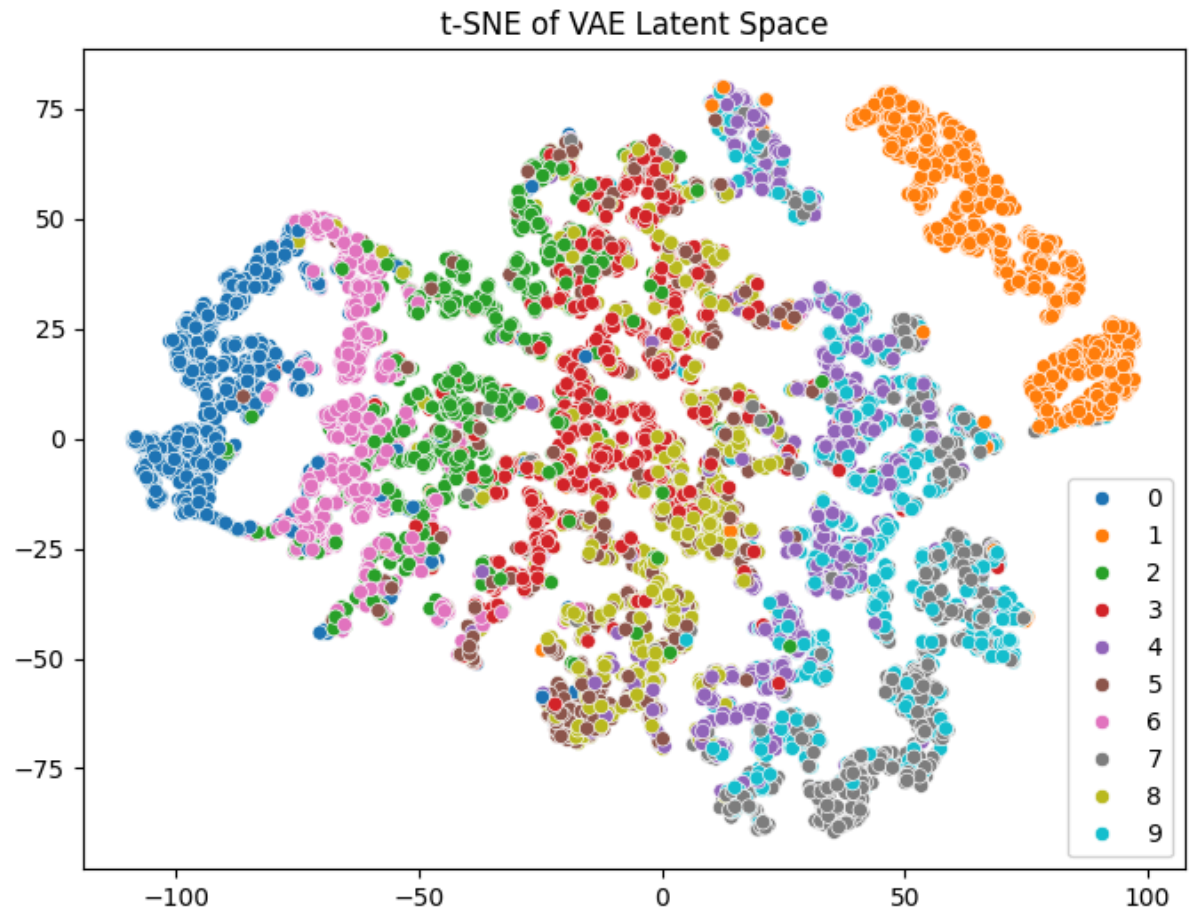
VAE Latent Space

- More structured, smooth, and separable clusters
- Easier to interpret and interpolate between different digits

t-SNE AE PLOT

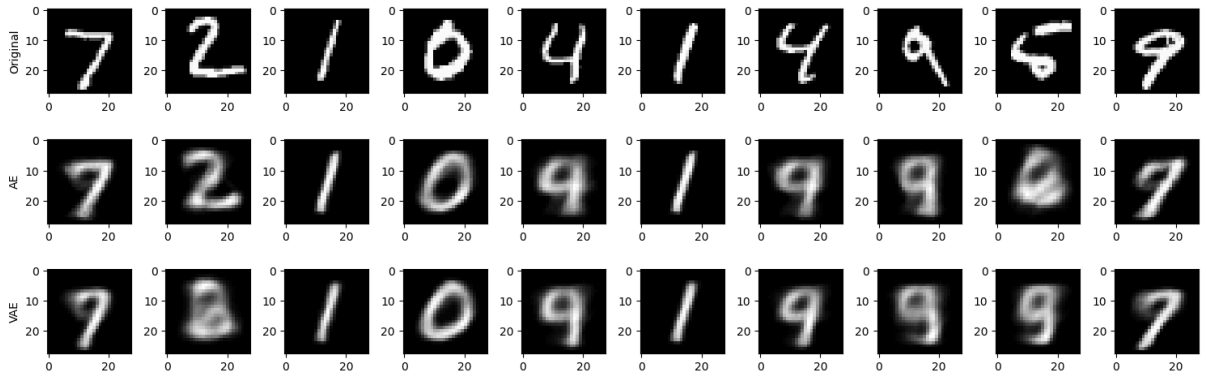


t-SNE VAE PLOT



Reconstruction Comparison

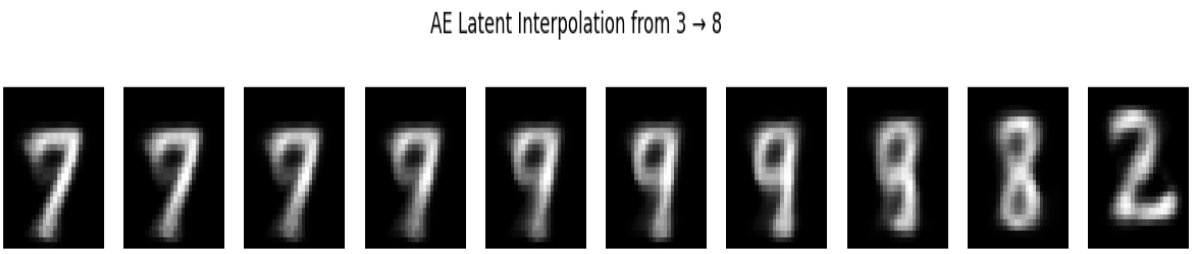
AE tends to reconstruct sharper digits. VAE reconstructions are blurrier but more diverse due to its generative nature.

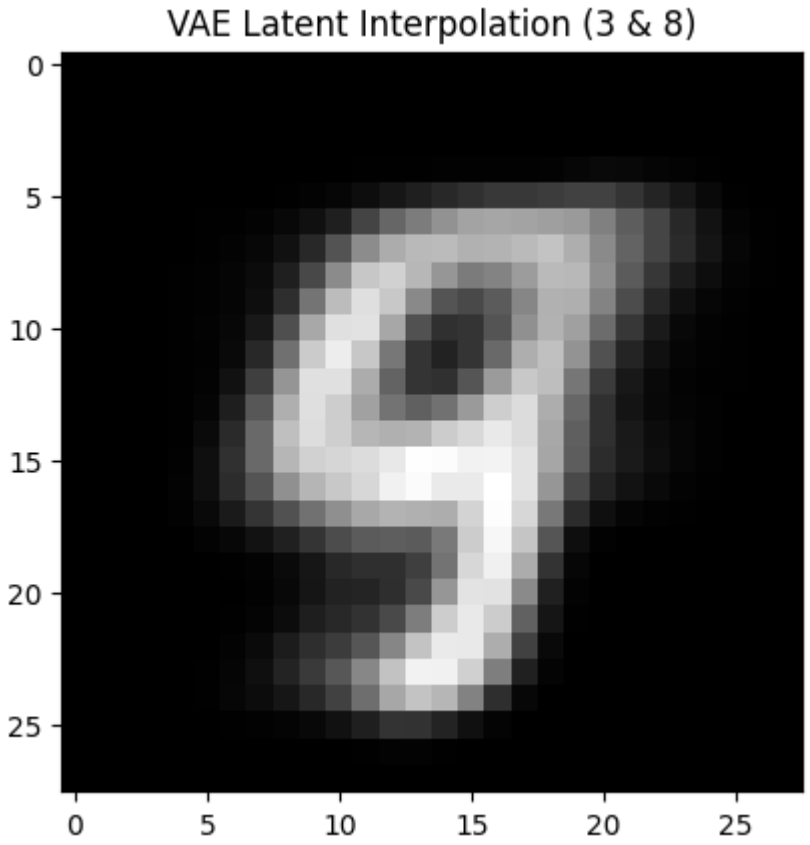


Latent Vector Arithmetic (Bonus)

Task: Interpolate between digits "3" and "8"

- AE latent space lacks meaningful interpolation.
- VAE latent vectors allow semantic blending: digit shapes morph smoothly.





VAE better captures semantic relationships in latent space.

Key Learnings

- AE compresses data but does not model variability or semantics well.
- VAE introduces stochasticity, enabling generation and smoother transitions.
- t-SNE reveals how structured or chaotic the latent spaces are.
- Latent arithmetic in VAE showcases understanding of digit concepts (e.g., transitioning from 3 → 8).

FUTURE WORK

- Test with higher latent dimensions (e.g., 10, 20) for richer encodings.
- Apply VAE to complex datasets like CIFAR-10 or CelebA.

- Explore models like:
    - β-VAE: Encourages disentangled latent factors.
    - Conditional VAE (CVAE): Generate data conditioned on labels.
    - Denoising AE: Adds robustness to noise.
- Integrate attention mechanisms or transformer-based encoders.

Author

- **Name:** Dudekula Prathimabi
- **Roll Number / ID (if applicable):** 228U1A3310
- **Email: dprathimabi2004@gmail.com**