Variational Autoencoder (VAE) for Fashion MNIST

RA2211056010039

ADHIDEV MD

B.TECH CSE WITH DATA SCIENCE

Introduction

A Variational Autoencoder (VAE) is a generative model that learns to encode input data into a lower-dimensional latent space while enforcing a probabilistic structure. The model consists of an encoder, a decoder, and a reparameterization trick to allow backpropagation through the latent space. This report details the implementation of a VAE trained on the Fashion MNIST dataset, covering model architecture, training, and results.

Dataset Description

Fashion MNIST consists of grayscale images (28x28 pixels) representing 10 different clothing categories, including shirts, trousers, and sneakers. The dataset serves as a benchmark for machine learning models in image generation and classification tasks.

```
transform = transforms.Compose([transforms.ToTensor()])

train_dataset = datasets.FashionMNIST(root='./data', train=True,
transform=transform, download=True)

test_dataset = datasets.FashionMNIST(root='./data', train=False,
transform=transform, download=True)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

Model Architecture

The VAE model consists of:

- **Encoder:** A neural network that maps input images to a latent space, outputting both a mean (μ) and log variance ($\log\sigma^2$) for each dimension.
- **Reparameterization Trick:** Sampling from a Gaussian distribution using the computed mean and variance to allow gradient-based optimization.

- **Decoder:** A neural network that reconstructs images from the sampled latent space representation.

The model was implemented using PyTorch with the following structure:

- Fully connected layers for encoding and decoding.
- Activation functions (ReLU, Sigmoid) for non-linearity.
- Binary cross-entropy loss for reconstruction combined with KL divergence to regularize the latent space.

```
class VAE(nn.Module):
    def _init_(self, latent_dim):
        super(VAE, self).__init__()

        # Encoder
        self.fc1 = nn.Linear(28*28, 400)
        self.fc2_mu = nn.Linear(400, latent_dim)
        self.fc2_logvar = nn.Linear(400, latent_dim)

        # Decoder
        self.fc3 = nn.Linear(latent_dim, 400)
        self.fc4 = nn.Linear(400, 28*28)

    def encode(self, x):
        h = F.relu(self.fc1(x))
        mu = self.fc2_mu(h)
        logvar = self.fc2_logvar(h)

        return mu, logvar


    def reparameterize(self, mu, logvar):

        std = torch.exp(0.5 * logvar)

        eps = torch.randn_like(std)

        return mu + eps * std


    def decode(self, z):

        h = F.relu(self.fc3(z))

        return torch.sigmoid(self.fc4(h))


    def forward(self, x):

        mu, logvar = self.encode(x.view(-1, 28*28))

        z = self.reparameterize(mu, logvar)

        recon_x = self.decode(z)

        return recon_x, mu, logvar
```

Training Setup

- **Optimizer:** Adam optimizer with a learning rate of 0.001.
- **Batch Size:** 128.
- **Latent Dimension:** 10.
- **Epochs:** 10.

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

model = VAE(latent_dim).to(device)

optimizer = optim.Adam(model.parameters(), lr=learning_rate)



losses = []

for epoch in range(epochs):

    model.train()

    train_loss = 0

    for batch_idx, (data, _) in enumerate(train_loader):

        data = data.to(device)

        optimizer.zero_grad()

        recon_batch, mu, logvar = model(data)

        loss = vae_loss(recon_batch, data, mu, logvar)

        loss.backward()

        train_loss += loss.item()

        optimizer.step()



    avg_loss = train_loss / len(train_loader.dataset)

    losses.append(avg_loss)

    print(f"Epoch {epoch+1}, Loss: {avg_loss:.4f}")
```

- **Loss Function:** The VAE loss comprises the sum of reconstruction loss and KL divergence:
  where BCE (Binary Cross-Entropy) ensures accurate reconstructions and KL divergence regularizes the latent space.

```
def vae_loss(recon_x, x, mu, logvar):

    recon_loss = F.binary_cross_entropy(recon_x, x.view(-1, 28*28),
reduction='sum')

    kl_div = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    return recon_loss + kl_div
```

Results

Training Loss

The loss curve steadily decreases, indicating successful model convergence. The balance between reconstruction and KL divergence was tuned to ensure meaningful latent representations.

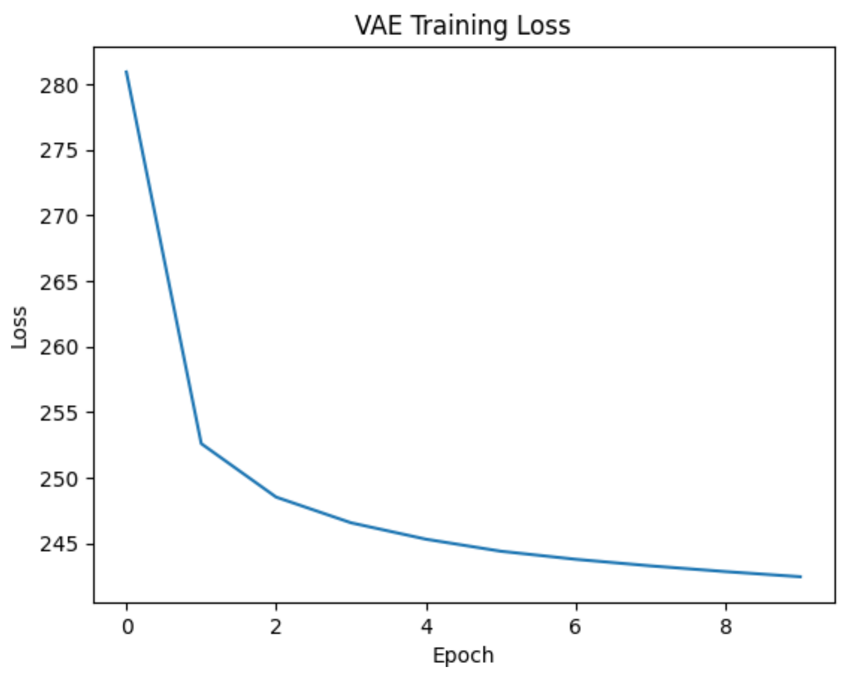
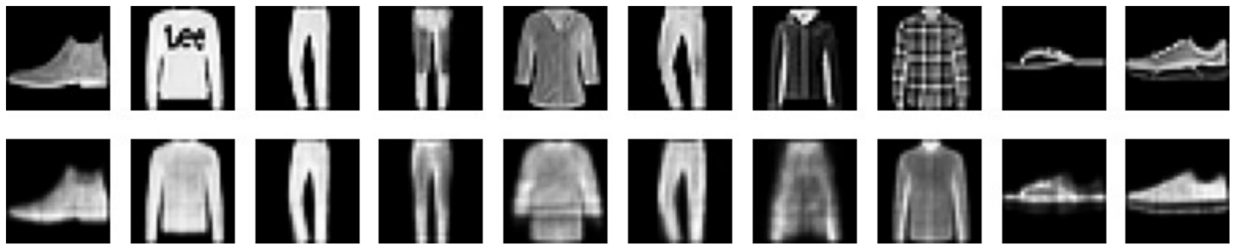


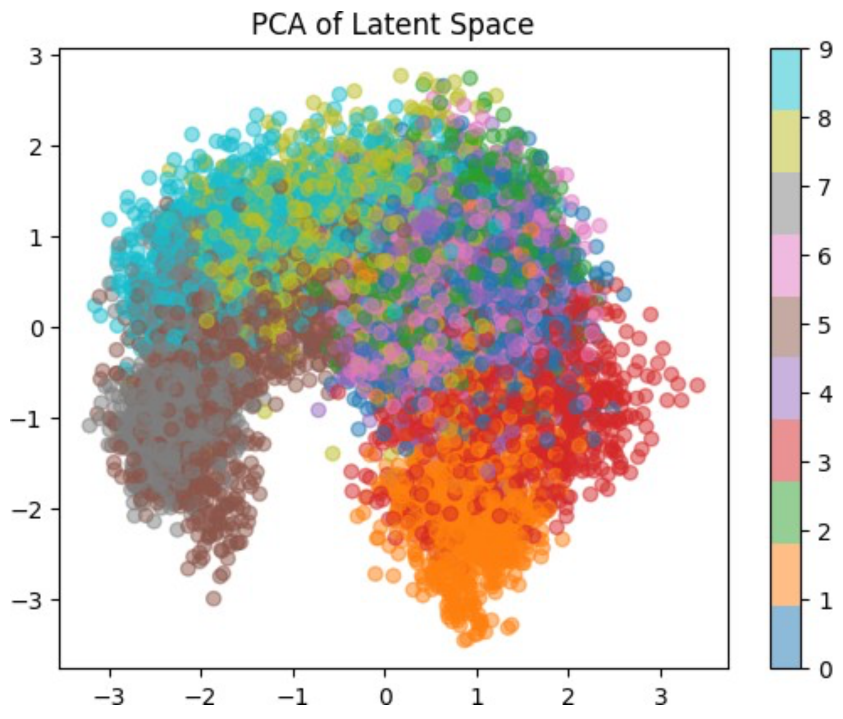
Image Reconstruction

Side-by-side comparisons of original and reconstructed images show that the VAE effectively captures the main features of clothing items, albeit with some blurring due to the probabilistic nature of the latent space.



Latent Space Visualization

We applied PCA and t-SNE to visualize the learned latent representations. The results show that clothing categories are well-clustered, indicating that the model successfully captures meaningful latent structure.



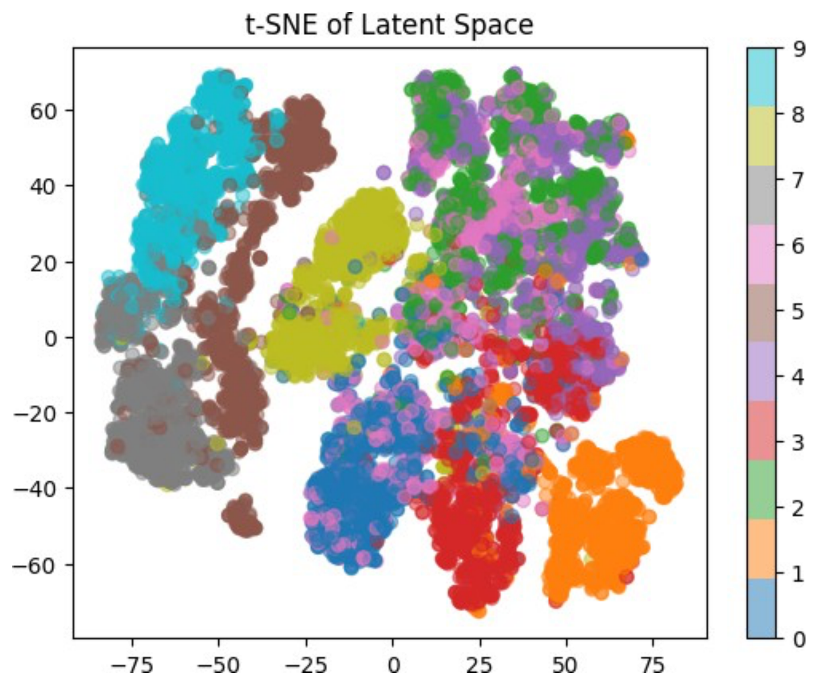
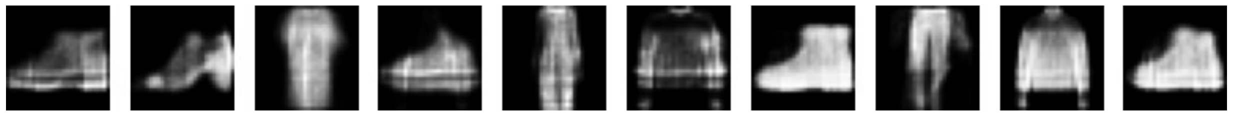


Image Generation

Sampling from the latent space generated realistic yet slightly blurry images. Increasing the latent dimension could improve quality by allowing more expressive representations.



Conclusion

This implementation of a VAE successfully learns meaningful latent representations of Fashion MNIST. While the model can generate new images, further improvements can be made by increasing training epochs, adjusting hyperparameters, or using more advanced architectures, such as convolutional VAEs.