1. Variational Autoencoder (VAE) on MNIST

1.1. Introduction

Variational Autoencoders (VAEs) are generative models that learn a continuous latent representation of data while enforcing a probabilistic structure on the latent space. Unlike classical autoencoders, VAEs model the latent variables as probability distributions and allow sampling to generate new data.

In this lab, a VAE is implemented and trained on the MNIST dataset of handwritten digits using KNIME Analytics Platform with a Python integration based on PyTorch.

1.2. Dataset Preparation

1.2.1. MNIST Dataset

The MNIST dataset consists of 70,000 grayscale images of handwritten digits from 0 to 9. Each image has a resolution of 28×28 pixels.

The dataset is loaded directly using the PyTorch `torchvision` library.

- Images are normalized to the range $[0, 1]$
- Data is split into 80% training and 20% testing
- Labels are not used for training, since the task is unsupervised

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, random_split

# Transformation: normalisation des pixels dans [0,1]
transform = transforms.ToTensor()

# Chargement du dataset MNIST
dataset = datasets.MNIST(
    root="./data",
    train=True,
    download=True,
    transform=transform
)

# Partitionnement 80% entraînement / 20% test
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Création des DataLoaders
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
```

1.2.2. Sample Visualization

The following figure shows randomly selected samples from the MNIST dataset.

Figure 1: Sample images from the MNIST dataset

1.2.3. KNIME Workflow Implementation

The implementation of the Variational Autoencoder was initially attempted using KNIME's native Keras Deep Learning nodes.

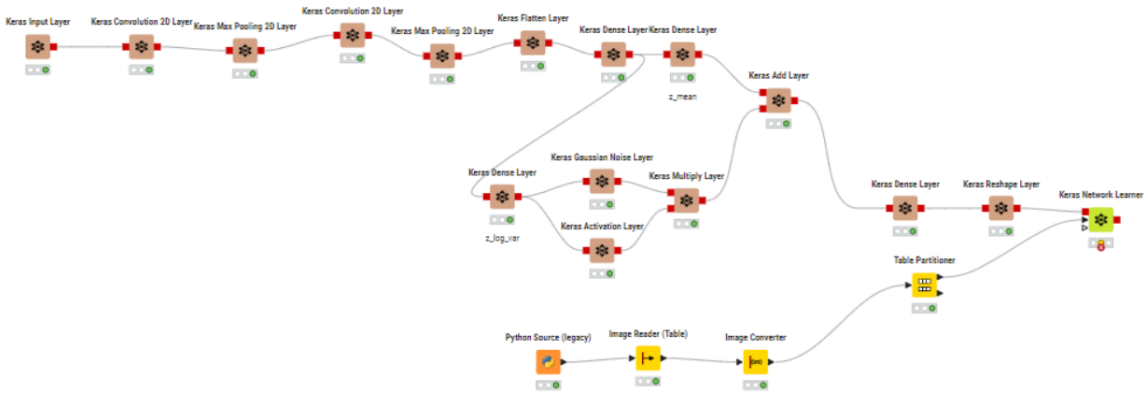

Figure 2: Initial attempt to implement the VAE using KNIME Keras nodes

However, due to the complexity of the VAE architecture—especially the reparameterization trick and the KL divergence loss—it was not possible to correctly implement the model using only graphical Keras nodes. Several connections and custom operations required by the VAE are not directly supported in the standard KNIME Keras workflow.

As a result, a Python-based solution was adopted using the **Python Source (legacy)** node, allowing full control over the PyTorch implementation of the VAE.

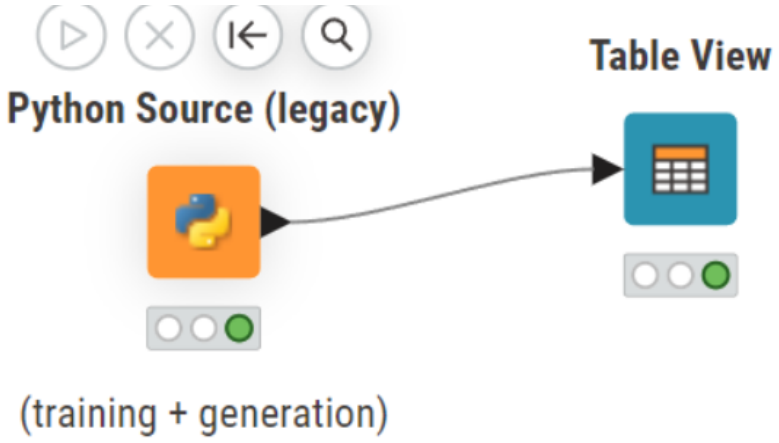

Figure 3: Final KNIME workflow using Python Source (legacy) for VAE training and generation

This approach ensures correct implementation of the encoder, decoder, loss function, training loop, and image generation while maintaining integration within the KNIME Analytics Platform.

1.3. Variational Autoencoder Architecture

1.3.1. Encoder Network

The encoder maps the input image to a latent probability distribution. Each image is flattened into a 784-dimensional vector.

The encoder outputs:

- Mean vector (μ)
- Log-variance vector ($\log \sigma^2$)

These parameters define a Gaussian distribution in the latent space.

```
class Encoder(nn.Module):
    def __init__(self, latent_dim):
        super().__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc_mu = nn.Linear(256, latent_dim)
        self.fc_logvar = nn.Linear(256, latent_dim)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        mu = self.fc_mu(x)
        logvar = self.fc_logvar(x)
        return mu, logvar
```

The ReLU activation function introduces non-linearity and improves learning capacity.

1.3.2. Reparameterization Trick

To allow backpropagation through stochastic sampling, the reparameterization trick is used:

$z = \mu + \sigma \odot \varepsilon$, where $\varepsilon \quad N(0, I)$

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

This separates randomness from the network parameters, enabling gradient-based optimization.

1.3.3. 3.3 Decoder Network

The decoder reconstructs the original image from the latent vector z.

```
class Decoder(nn.Module):
    def __init__(self, latent_dim):
        super().__init__()
        self.fc1 = nn.Linear(latent_dim, 256)
        self.fc2 = nn.Linear(256, 784)

    def forward(self, z):
        z = torch.relu(self.fc1(z))
        return torch.sigmoid(self.fc2(z))
```

The sigmoid activation ensures that output pixel values remain in the range $[0, 1]$.

1.4. Loss Function and Training

1.4.1. VAE Loss Function

The total VAE loss consists of two components:

- Reconstruction loss (Binary Cross-Entropy)
- KL divergence loss

The KL divergence regularizes the latent space to follow a standard normal distribution $N(0, I)$.

```
def vae_loss(recon_x, x, mu, logvar, beta=1.0):
    recon_loss = nn.functional.binary_cross_entropy(
        recon_x, x, reduction='sum'
    )
    kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return recon_loss + beta * kl_loss
```

The parameter β controls the trade-off between reconstruction quality and latent space regularization.

1.4.2. Training Configuration

- Optimizer: Adam
- Learning rate: 0.001
- Batch size: 128
- Epochs: 30
- Latent dimension: 2 (for visualization)

The training loss evolution is shown below.

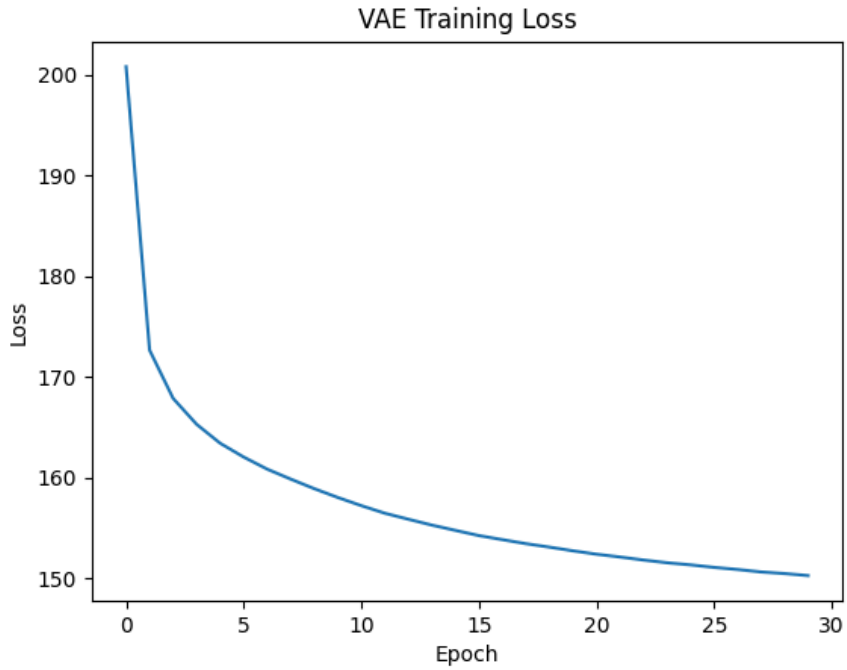

Figure 4: Training loss over epochs

1.5. Evaluation Results

1.5.1. Image Reconstruction

Test images are passed through the encoder and decoder to evaluate reconstruction quality.

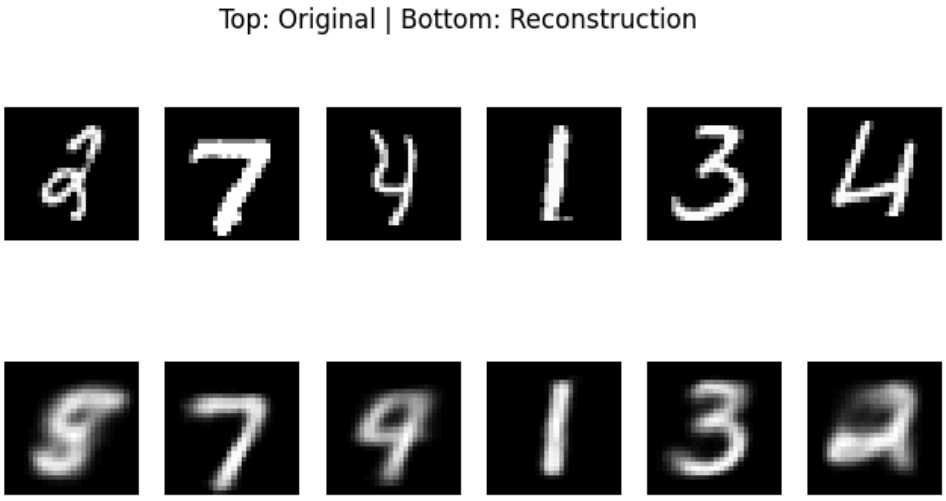

Figure 5: Top: original images — Bottom: reconstructed images

The model successfully preserves the main digit structure while introducing minor blurring, which is expected in VAEs due to probabilistic sampling.

1.5.2. Latent Space Visualization

With a 2-dimensional latent space, encoded test samples are visualized and colored by digit label.

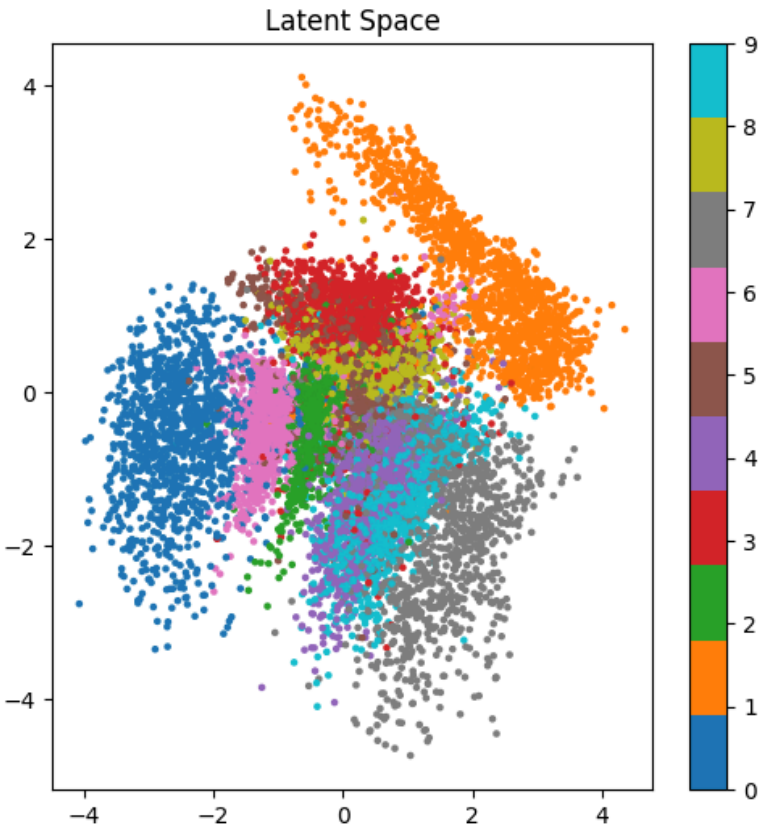

Figure 6: 2D latent space representation colored by digit label

Digits with similar shapes tend to cluster together, indicating a meaningful latent representation.

1.6. Image Generation

New images are generated by sampling latent vectors from a standard normal distribution and decoding them.

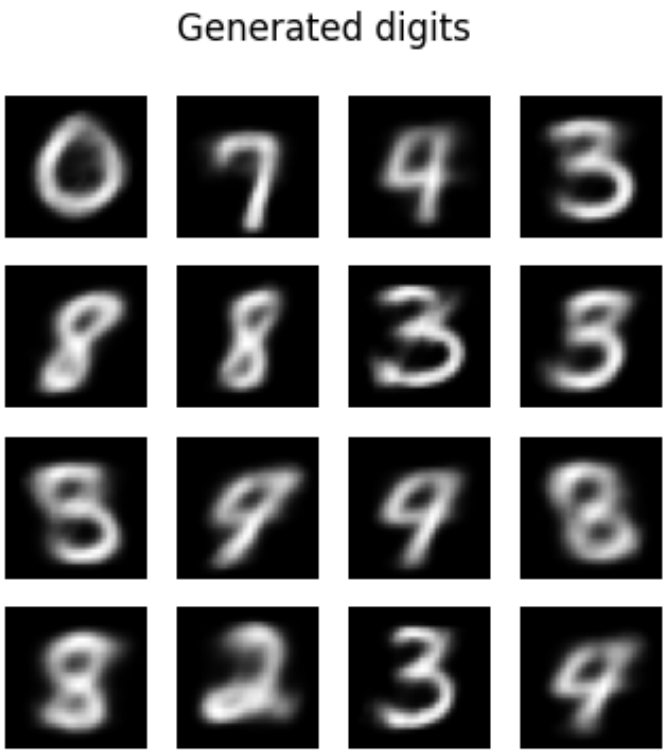

Figure 7: New digits generated from random latent vectors

The generated digits are diverse and visually coherent, demonstrating the generative capability of the VAE.

1.7. Experiments and Discussion

Several variations can be explored:

- Increasing latent dimensionality (5, 10, 20) improves reconstruction quality
- Lower β emphasizes reconstruction accuracy
- Higher β enforces smoother latent spaces but reduces detail

This highlights the trade-off between fidelity and regularization inherent in VAEs.

1.8. Conclusion

In this lab, a Variational Autoencoder was successfully implemented and evaluated on the MNIST dataset. The model learned a structured latent space, produced meaningful reconstructions, and generated new handwritten digit samples.

VAEs provide a powerful framework for unsupervised learning and generative modeling.