UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
BACHELOR DATA SCIENCE (B3)

ICT3.017: INTRODUCTION TO DEEP LEARNING

Midterm Report

The variational autoencoder for image denoising

Group 45

Group Info

Full name	Student ID
Pham Tien Dat	22BI13080
Bui Ngoc Linh	22BI13246
Nguyen Dong San	22BI13388
Bui Thanh Vinh	22BI13469
La Thi Thuy Chi	BA12-026
Le Duc Anh	BI12-035
Nghiem Phu Khang	BI12-204

Hanoi, October 2024

Table of Contents

I. Introduction

1. Overview of Autoencoders and Variational Autoencoders (VAEs)

Autoencoders are a form of neural network that learns a compact, lower-dimensional representation of input data. They are commonly used for unsupervised learning tasks such as dimensionality reduction and feature extraction. A typical autoencoder consists of two parts: an encoder that compresses the input data into a latent space and a decoder that reconstructs the data from the latent representation.

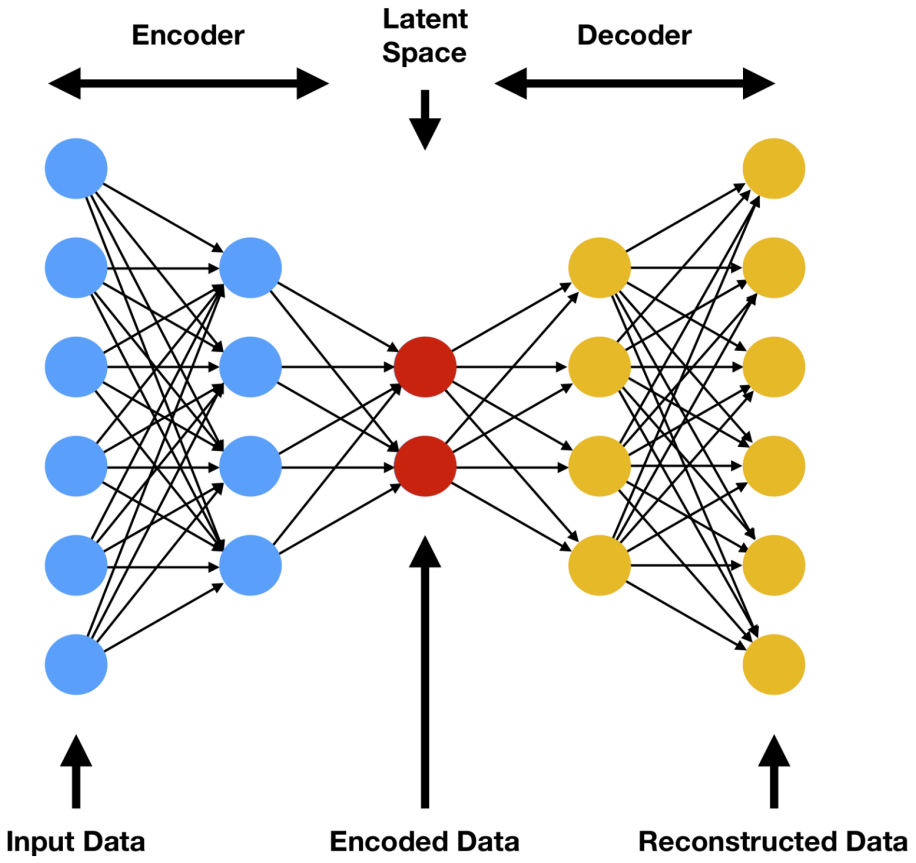

Figure 1: Autoencoder architecture

Variational Autoencoders (VAEs) build on classic autoencoders by taking a probabilistic approach to the latent space. Instead of learning a fixed latent vector for each input, VAEs learn a distribution (usually Gaussian) over the latent space, enabling better generalization and production of new data samples. The main innovation in VAEs is the reparameterization method, which allows backpropagation via stochastic layers.

2. Problem statement

This project addresses the problem of image denoising using the MNIST dataset. The objective is to reconstruct clean images of handwritten digits from noisy, corrupted versions. We employ VAEs to learn the latent structure of the data, allowing us to reconstruct clean, noise-free images, improving the overall visual clarity and utility of the images.

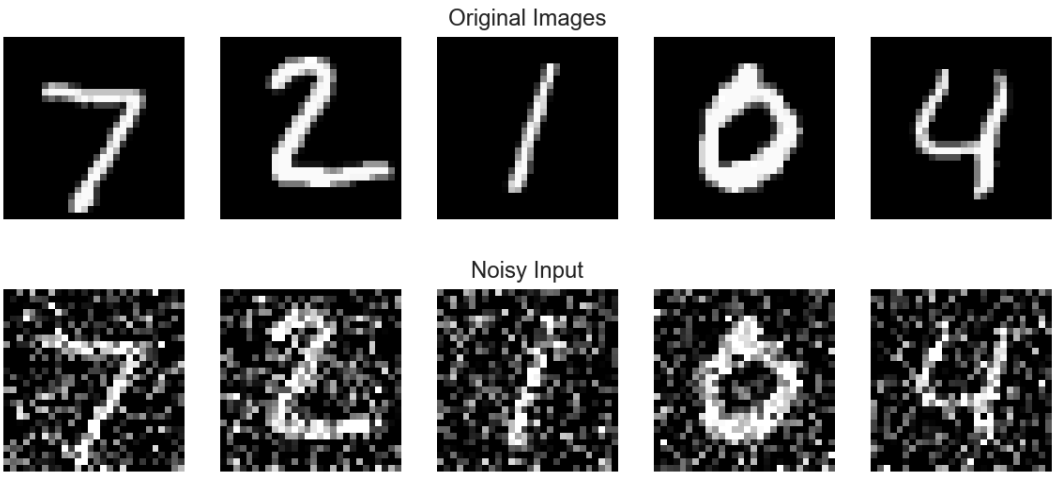

Figure 2: Example of denoise with autoencoders

3. Goal of the study

This study is to investigate the effectiveness of VAEs for image denoising. Image denoising is the process of removing noise from images to enhance their visual quality or prepare them for further processing tasks. VAEs, with their ability to capture meaningful latent representations, are ideal for this task. In this research, we focus on understanding how VAEs can learn to reconstruct clean images from noisy input data using latent space and probabilistic modeling. We aim to build a VAE model and test its performance on a noisy dataset to demonstrate its capabilities in image denoising. The results will be analyzed both visually and numerically, and we will explore the strengths and limitations of the VAE approach compared to some traditional techniques.

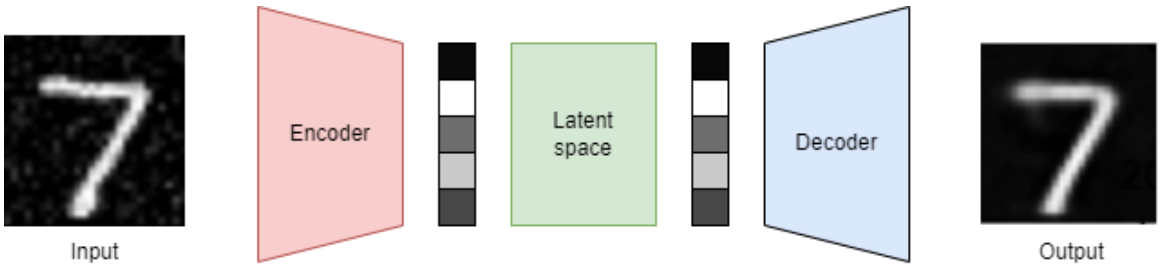

Figure 3: Overview of the VAE

II. Case study

1. Dataset

We choose the MNIST dataset, a simple and well-known dataset that is often used for testing models in deep learning. The dataset has 60,000 training images and 10,000 test images which are 28x28 grayscale images of handwritten digits ranging from 0 to 9.

Figure 4: Sample images from MNIST test dataset

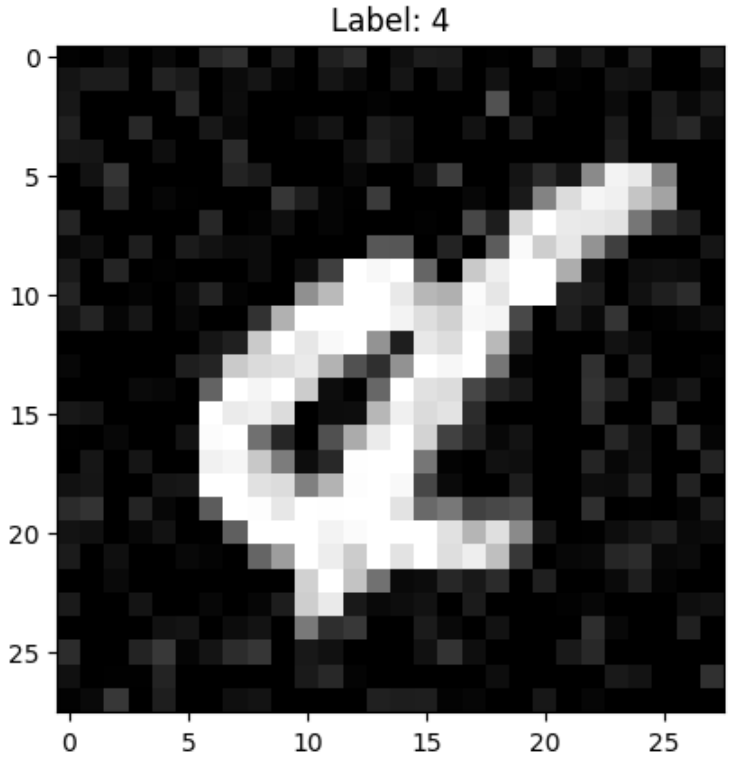

Figure 5: One example extracted from the dataset

2. Deep Learning Model

Components

A VAE model consists of two steps.

- **Encoder**: converts input to a latent distribution with two vectors: mean $\boldsymbol{\mu}$ and log-variance ***log(var)***.

```
        # Encoder layers
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc21 = nn.Linear(hidden_dim, latent_dim)  # Mean
        self.fc22 = nn.Linear(hidden_dim, latent_dim)  # Log-variance
```

Block 1: Code of the encoder layer

- **Decoder**: takes a sample latent vector z that is taken from the learnt distribution and reconstructs the input data from it. In the case of denoising, the decoder reconstructs a clean image from the compressed noisy image.

```
        # Decoder layers
        self.fc3 = nn.Linear(latent_dim, hidden_dim)
        self.fc4 = nn.Linear(hidden_dim, input_dim)
```

Block 2: Code of the decoder layer

There are two key components of the VAE Latent Structure.

- **Reparameterization Trick**: this technique allows us to train the model using backpropagation by making it differentiable. We take a sample from a normal distribution and adjust it based on the encoder's learnt mean and variance.

```
    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std  # Return reparameterized latent vector
```

Block 3: Function of the reparameterization trick

- **Latent Space**: preserves important details while removing the noise from the images, the latent space. This allows the VAE to recreate denoised images.

Loss Function

```
# Define the loss function for VAE
def vae_loss_function(recon_x, x, mu, logvar):
    # Binary Cross-Entropy for reconstruction loss
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')

    # KL Divergence loss
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    return BCE + KLD
```

Block 4: Function to calculate losses of the VAE model

The loss function of a VAE is composed of two terms.

- **Reconstruction Loss**: this is the binary cross-entropy between the original clean image and the reconstructed image. It calculates the error between these two images and ensures that the reconstructed image is as close as possible to the original.
- **KL Divergence**: this regularizes the latent space by encouraging it to follow a normal distribution then helping to improve the generalization of the model.

Training Process

The model was trained for 100 epochs using the Adam optimizer with a learning rate of 0.001. The training loop iterates over mini-batches of size 64, updating the model parameters after each batch.

```
# Initialize the model, optimizer, and move model to device (GPU/CPU)
model = VAE().to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

Block 5: Intialize the model

The training process includes three steps.

- **Zeroing gradients** before the forward pass.
- **Forward pass**: the input is passed through the encoder, reparameterized, and decoded to produce the output image.
- **Backward pass**: gradients are computed using backpropagation, and the optimizer updates the model weights.

```
# Training loop with 100 epochs and loss tracking
def train_and_plot(model, train_loader, optimizer, num_epochs=100):
    train_losses = []

    for epoch in range(1, num_epochs + 1):  # 100 epochs
        model.train()
        train_loss = 0
        for batch_idx, (data, _) in enumerate(train_loader):
            data = data.to(device)
            optimizer.zero_grad()

            # Forward pass
            recon_batch, mu, logvar = model(data)

            # Compute loss and backpropagate
            loss = vae_loss_function(recon_batch, data, mu, logvar)
            loss.backward()
            train_loss += loss.item()
            optimizer.step()

        # Average loss per epoch
        avg_loss = train_loss / len(train_loader.dataset)
        train_losses.append(avg_loss)
        print(f'Epoch {epoch}, Loss: {avg_loss}')
```

Block 6: Function to train the model

The training loss is tracked, printed after each epoch, and plotted into a graph to monitor the model's learning progress.

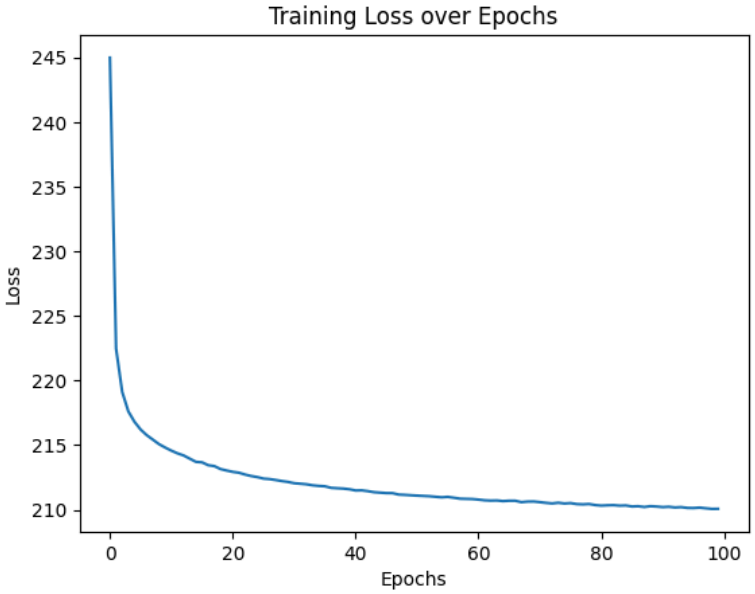

Figure 6: Loss curve over epochs

Results Analysis

The results indicate that the VAE was able to denoise the MNIST images effectively. After 100 epochs, the model successfully learned to reconstruct clean images from noisy inputs, as demonstrated by the output images.

- Training Loss: the training loss consistently decreased during the first 50 epochs, indicating that the model was learning. After 50 epochs, the loss began to plateau, showing that the model had likely converged.
- Denoising Performance: the VAE successfully removed noise from the noisy MNIST digits. While the reconstructed images were slightly blurry, they retained key digit features, proving the model's effectiveness.

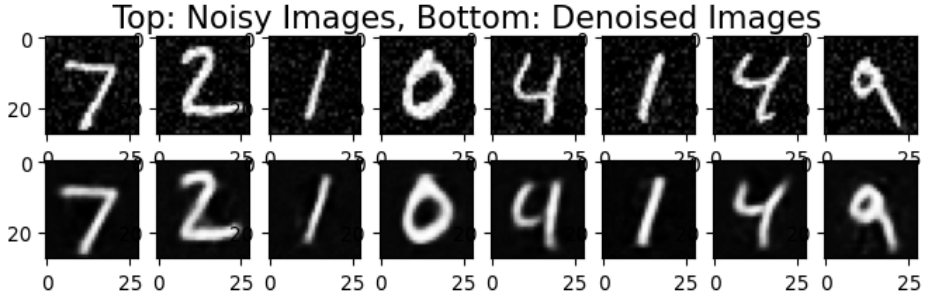

Figure 7: Visualization of the results on the test dataset

PSNR Testing Process

After training the VAE model, the **test_model()** function was called to compute the PSNR for the denoised images. The following steps were followed:

```
# Function to calculate MSE (with shape matching)
def calculate_mse(original, denoised):
    # Flatten the input images to the same shape as the output from the VAE (784)
    original = original.view(-1, 784)  # Flatten input images to match VAE output
    denoised = denoised.view(-1, 784)  # Ensure the denoised images are flattened
    # Calculate MSE loss between original and denoised images
    mse = F.mse_loss(denoised, original)
    return mse.item()
# Function to calculate PSNR
def calculate_psnr(mse, max_pixel_value=1.0):
        psnr = 10 * math.log10(max_pixel_value**2 / mse)    # Use math.log10() for scalar
    return psnr
# PSNR Testing Function
def test_model(model, test_loader):
    model.eval()  # Set the model to evaluation mode
    psnr_total = 0
    with torch.no_grad():  # No gradient computation in test mode
        for batch_idx, (data, _) in enumerate(test_loader):
            data = data.to(device)
```

```
            recon_batch, _, _ = model(data)  # Forward pass to get
denoised images

            # Calculate MSE for the batch (after reshaping)

            mse = calculate_mse(data, recon_batch)

            # Calculate PSNR based on the MSE

            psnr = calculate_psnr(mse)

            psnr_total += psnr

    # Average PSNR across the test dataset

    avg_psnr = psnr_total / len(test_loader)
```

1. **Evaluation Mode**: The model was set to **evaluation mode** using model.eval(), ensuring that the model did not update weights and was ready for testing.
2. **Forward Pass**: For each batch in the test dataset, a forward pass was performed through the VAE to get the denoised (reconstructed) images.
3. **MSE and PSNR Calculation**: For each batch, the **Mean Squared Error (MSE)** between the original and denoised images was computed, followed by the **PSNR** calculation.
4. **Average PSNR**: The PSNR values for each batch were averaged to get the final **PSNR score**, which represents the overall denoising performance of the VAE model.

III. Conclusion

In this study, we implemented a **Variational Autoencoder (VAE)** model to address the problem of image denoising using the MNIST dataset. Through the process of adding Gaussian noise to the images and training the VAE to reconstruct them, we observed the model's ability to recover the clean version of these noisy inputs. The combination of the **reparameterization trick** and the **latent space** allows the VAE to capture key features from the input data, leading to effective noise removal.

The results of this experiment demonstrate that VAEs are a powerful tool for image-denoising tasks. The model consistently reduced the training loss during the initial epochs and successfully learned to denoise the images, despite some slight blurriness in the

final outputs. This is expected in VAEs, as the probabilistic nature of the model can lead to smoother outputs due to the averaging effect of the latent space representation. However, the critical features of the handwritten digits were preserved, showing that the VAE effectively captured the underlying structure of the data.

The **Binary Cross-Entropy** and **KL Divergence** losses provided a well-balanced objective, ensuring that the reconstructions were accurate and that the latent space followed a regularized distribution. Although we trained the model for 100 epochs, further fine-tuning, or increasing the network's capacity, may help achieve even sharper reconstructions.

Overall, this project highlights the potential of VAEs for real-world denoising applications. While more advanced architectures or hybrid models could further improve results, this study serves as a solid foundation for exploring the role of probabilistic models in image denoising and other related tasks in computer vision. In future work, it would be beneficial to experiment with different noise levels, more complex datasets, or alternative autoencoder architectures, such as ***Denoising Autoencoders (DAE)*** or ***Convolutional Variational Autoencoders (CVAE)***, to enhance performance and generalization.

IV. References

- Denoising and Variational Autoencoders
- https://www.google.com/url?q=https%3A%2F%2Fmedium.com%2F%40aniketp2009%2Fimage-denoising-using-variational-autoencoders-e2cda0c336d2