computervision-final-prep / lab / Assignment 3 / st125970_notebook_task_3.ipynb

limhpone ass 2 and ass 3 d2a65e4 · 2 hours ago

1973 lines (1973 loc) · 626 KB

Preview Code Blame Raw

Task 3: Variational Autoencoder (VAE)

Aye Khin Khin Hpone (Yolanda Lim)_125970

Objective

Implement a Variational Autoencoder (VAE) to learn latent representations of MNIST digits, generate new samples, and analyze the effect of different latent dimensionalities on reconstruction quality and generation diversity.

Overview

This notebook explores VAE architecture and training on the MNIST dataset, comparing two latent space configurations:

- **Experiment 1:** `latent_dim = 128`
- **Experiment 2:** `latent_dim = 256`

Methodology

1. **Setup & Configuration:** Import required libraries and configure GPU/CPU device
2. **Data Preparation:** Load and preprocess MNIST dataset with appropriate transformations
3. **Model Architecture:** Implement VAE with encoder (3-4 Conv layers), decoder (transpose convolutions), and reparameterization trick
4. **Loss Function:** Combine reconstruction loss (MSE) with KL divergence regularization
5. **Training Pipeline:** Train models with comprehensive logging and evaluation
6. **Visualization & Analysis:** Compare reconstruction quality, generated samples, and latent space interpolations

In [1]:

```
# 1. Setup & Imports

import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
```

```
import matplotlib.colors as mcolors
import numpy as np
from typing import Tuple

# Device Configuration - Check for GPU availability
if torch.cuda.is_available():
    print(f"CUDA Available - {torch.cuda.device_count()} GPU(s) detected")
    for i in range(torch.cuda.device_count()):
        print(f"  GPU {i}: {torch.cuda.get_device_name(i)}")
    device = 'cuda:3'  # Use first GPU (adjust to cuda:1, cuda:2 etc. if nee
else:
    device = 'cpu'
    print("CUDA not available - using CPU")

print(f"\nSelected device: {device}")
```

```
CUDA Available - 4 GPU(s) detected
  GPU 0: NVIDIA GeForce RTX 2080 Ti
  GPU 1: NVIDIA GeForce RTX 2080 Ti
  GPU 2: NVIDIA GeForce RTX 2080 Ti
  GPU 3: NVIDIA GeForce RTX 2080 Ti

Selected device: cuda:3
```

In [2]:

```
# 2. Dataset Preparation & Hyperparameters

# Training Hyperparameters
lr = 2e-4               # Learning rate for Adam optimizer
BATCH_SIZE = 256        # Batch size for training and evaluation
NUM_EPOCHS = 50         # Total training epochs
logging_interval = 50   # Print training progress every N batches

# Image specifications (MNIST)
IMG_SHAPE = (1, 28, 28)  # Single channel, 28x28 pixels
IMG_FLAT = 28 * 28       # Flattened image size

# Data Preprocessing Pipeline
transform = transforms.Compose([
    transforms.ToTensor(),  # Convert PIL image to tensor [0, 1]
])

# Load MNIST Dataset
print("Loading MNIST dataset...")
train_dataset = datasets.MNIST(
    root=".",
    train=True,
    transform=transform,
    download=True
)
test_dataset = datasets.MNIST(
    root=".",
    train=False,
    transform=transform,
    download=True
)

# Create DataLoaders with parallel data loading
train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=BATCH_SIZE
```

```
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=2  # Parallel data loading threads
)
test_loader = DataLoader(
    dataset=test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2
)

# Display dataset information
print(f"\nDataset Statistics:")
print(f"  Training samples: {len(train_dataset):,}")
print(f"  Test samples: {len(test_dataset):,}")
print(f"  Batch size: {BATCH_SIZE}")
print(f"  Training batches per epoch: {len(train_loader)}")
print(f"  Test batches: {len(test_loader)}")
```

```
Loading MNIST dataset...

Dataset Statistics:
  Training samples: 60,000
  Test samples: 10,000
  Batch size: 256
  Training batches per epoch: 235
  Test batches: 40
```

In [3]:

```
# 3. Utility Classes for Model Architecture

class Reshape(nn.Module):
    """
    Custom layer to reshape tensors within nn.Sequential
    Enables inline tensor reshaping without breaking sequential flow
    """
    def __init__(self, *args):
        super().__init__()
        self.shape = args

    def forward(self, x):
        return x.view(self.shape)


class Trim(nn.Module):
    """
    Custom layer to trim tensor spatial dimensions
    Used to handle dimension mismatches in decoder upsampling
    """
    def __init__(self, *args):
        super().__init__()

    def forward(self, x):
        return x[:, :, :28, :28]  # Trim to 28x28 spatial dimensions


# 4. VAE Model Architecture

class VAE(nn.Module):
    """
    Variational Autoencoder with Convolutional Architecture for MNIST
```

```
    Architecture:
        - Encoder: 3 convolutional layers with BatchNorm, LeakyReLU, and Dro
        - Latent Space: Parameterized by mean (μ) and log-variance (log σ²)
        - Decoder: Transpose convolutional layers for upsampling

    Key Features:
        - Reparameterization trick for differentiable sampling
        - BatchNormalization for training stability
        - Dropout for regularization

    Args:
        latent_dim: The dimensionality of the latent space (z) - 128 or 256
    """
    def __init__(self, latent_dim: int):
        super().__init__()
        self.latent_dim = latent_dim

        # --- Encoder --- (3 Conv Layers as per task requirement)
        # 1x28x28 -> 32x14x14 -> 64x7x7 -> 64x7x7
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 32, stride=2, kernel_size=3, bias=False, padding=1)
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Dropout2d(0.25),
            #
            nn.Conv2d(32, 64, stride=2, kernel_size=3, bias=False, padding=1
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Dropout2d(0.25),
            #
            nn.Conv2d(64, 64, stride=1, kernel_size=3, bias=False, padding=1
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.1, inplace=True),
            #
            nn.Flatten(),  # -> 64 * 7 * 7 = 3136
        )

        # Latent space parameters (mu and log_variance)
        self.z_mean = nn.Linear(3136, latent_dim)  # 7x7x64
        self.z_log_var = nn.Linear(3136, latent_dim)

        # --- Decoder --- (Transpose convolutions as per task)
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 3136),
            Reshape(-1, 64, 7, 7),
            #
            nn.ConvTranspose2d(64, 64, stride=2, kernel_size=3),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Dropout2d(0.25),
            #
            nn.ConvTranspose2d(64, 64, stride=2, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Dropout2d(0.25),
            #
            nn.ConvTranspose2d(64, 32, stride=2, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Dropout2d(0.25),
```

```
            #
            nn.ConvTranspose2d(32, 1, stride=2, kernel_size=3, padding=1),
            #
            Trim(),  # 1x29x29 -> 1x28x28
            nn.Sigmoid()
        )

    def reparameterize(self, z_mu: torch.Tensor, z_log_var: torch.Tensor) ->
        """
        Reparameterization trick: z = μ + σ·ε where σ = exp(0.5·log_var)
        This allows backpropagation through stochastic sampling.
        """
        # Sample epsilon from standard normal - important: use correct devic
        eps = torch.randn(z_mu.size(0), z_mu.size(1)).to(z_mu.get_device())
        z = z_mu + eps * torch.exp(z_log_var/2.)
        return z

    def encoding_fn(self, x: torch.Tensor) -> torch.Tensor:
        """Encoding function for visualization (returns sampled z)"""
        x = self.encoder(x)
        z_mean, z_log_var = self.z_mean(x), self.z_log_var(x)
        encoded = self.reparameterize(z_mean, z_log_var)
        return encoded

    def forward(self, x: torch.Tensor) -> Tuple[torch.Tensor, torch.Tensor,
        """Forward pass returns: encoded, z_mean, z_log_var, decoded"""
        x = self.encoder(x)
        z_mean, z_log_var = self.z_mean(x), self.z_log_var(x)
        encoded = self.reparameterize(z_mean, z_log_var)
        decoded = self.decoder(encoded)
        return encoded, z_mean, z_log_var, decoded
```

In [4]:

```
# 5. Evaluation Function

def compute_epoch_loss_autoencoder(model, data_loader, loss_fn, device):
    """
    Compute average reconstruction loss over entire dataset
    Used for epoch-level evaluation during training
    """
    model.eval()
    curr_loss, num_examples = 0., 0
    with torch.no_grad():
        for features, _ in data_loader:
            features = features.to(device)
            _, _, _, logits = model(features)
            loss = loss_fn(logits, features, reduction='sum')
            num_examples += features.size(0)
            curr_loss += loss

        curr_loss = curr_loss / num_examples
        return curr_loss
```

In [5]:

```
# 6. Training Pipeline with Comprehensive Logging

def train_vae(model: VAE, optimizer: optim.Optimizer, num_epochs: int):
    """
```

```
    Complete training loop for VAE with detailed tracking and logging

    Features:
        - Batch-level and epoch-level loss tracking
        - Separate logging for reconstruction loss, KL divergence, and combi
        - Time tracking for performance monitoring
        - Periodic evaluation on training set

    Returns:
        log_dict: Dictionary containing all training metrics
    """
    print(f"\n{'='*60}")
    print(f"Training VAE (latent_dim={model.latent_dim})")
    print(f"{'='*60}\n")

    # Initialize logging dictionary for comprehensive tracking
    log_dict = {
        'train_combined_loss_per_batch': [],
        'train_combined_loss_per_epoch': [],
        'train_reconstruction_loss_per_batch': [],
        'train_kl_loss_per_batch': []
    }

    loss_fn = F.mse_loss  # MSE loss for evaluation
    skip_epoch_stats = False

    start_time = time.time()

    for epoch in range(num_epochs):
        model.train()

        for batch_idx, (features, _) in enumerate(train_loader):
            features = features.to(device)

            # FORWARD PASS
            encoded, z_mean, z_log_var, decoded = model(features)

            # LOSS CALCULATION
            # 1. KL Divergence (regularization term)
            kl_div = -0.5 * torch.sum(1 + z_log_var
                                      - z_mean**2
                                      - torch.exp(z_log_var),
                                      axis=1)  # sum over latent dimension

            batchsize = kl_div.size(0)
            kl_div = kl_div.mean()  # average over batch dimension

            # 2. Reconstruction Loss (MSE per pixel)
            pixelwise = F.mse_loss(decoded, features, reduction='none')
            pixelwise = pixelwise.view(batchsize, -1).sum(axis=1)  # sum ove
            pixelwise = pixelwise.mean()  # average over batch dimension

            # 3. Total Loss
            loss = pixelwise + kl_div

            # BACKWARD PASS
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

```
            # LOGGING (batch-level)
            log_dict['train_combined_loss_per_batch'].append(loss.item())
            log_dict['train_reconstruction_loss_per_batch'].append(pixelwise
            log_dict['train_kl_loss_per_batch'].append(kl_div.item())

            # Print progress
            if not batch_idx % logging_interval:
                print('Epoch: %03d/%03d | Batch %04d/%04d | Loss: %.4f'
                      % (epoch+1, num_epochs, batch_idx,
                         len(train_loader), loss))

        # EPOCH-LEVEL EVALUATION
        if not skip_epoch_stats:
            model.eval()

            with torch.set_grad_enabled(False):  # save memory during infere
                train_loss = compute_epoch_loss_autoencoder(
                    model, train_loader, loss_fn, device)
                print('***Epoch: %03d/%03d | Loss: %.3f' % (
                      epoch+1, num_epochs, train_loss))
                log_dict['train_combined_loss_per_epoch'].append(train_loss.

        print('Time elapsed: %.2f min' % ((time.time() - start_time)/60))

    print('\nTotal Training Time: %.2f min' % ((time.time() - start_time)/60
    print(f"{'='*60}\n")

    return log_dict
```

In [6]:

```
# 7. Visualization Functions

def plot_training_loss(minibatch_losses, num_epochs, averaging_iterations=10
    """
    Plot training loss curves with running average
    Displays both iteration-level and epoch-level progress on dual x-axes
    """
    iter_per_epoch = len(minibatch_losses) // num_epochs

    plt.figure()
    ax1 = plt.subplot(1, 1, 1)
    ax1.plot(range(len(minibatch_losses)),
             (minibatch_losses), label=f'Minibatch Loss{custom_label}')
    ax1.set_xlabel('Iterations')
    ax1.set_ylabel('Loss')

    if len(minibatch_losses) < 1000:
        num_losses = len(minibatch_losses) // 2
    else:
        num_losses = 1000

    ax1.set_ylim([
        0, np.max(minibatch_losses[num_losses:])*1.5
        ])

    ax1.plot(np.convolve(minibatch_losses,
                         np.ones(averaging_iterations,)/averaging_iterations
                         mode='valid'),
```

```
                label=f'Running Average{custom_label}')
    ax1.legend()

    # Set second x-axis for epochs
    ax2 = ax1.twiny()
    newlabel = list(range(num_epochs+1))
    newpos = [e*iter_per_epoch for e in newlabel]

    ax2.set_xticks(newpos[::10])
    ax2.set_xticklabels(newlabel[::10])
    ax2.xaxis.set_ticks_position('bottom')
    ax2.xaxis.set_label_position('bottom')
    ax2.spines['bottom'].set_position(('outward', 45))
    ax2.set_xlabel('Epochs')
    ax2.set_xlim(ax1.get_xlim())

    plt.tight_layout()


def plot_generated_images(data_loader, model, device, figsize=(20, 2.5), n_i
    """
    Visualize original images vs. VAE reconstructions
    Displays side-by-side comparison to assess reconstruction quality
    """
    fig, axes = plt.subplots(nrows=2, ncols=n_images,
                             sharex=True, sharey=True, figsize=figsize)

    for batch_idx, (features, _) in enumerate(data_loader):
        features = features.to(device)
        color_channels = features.shape[1]
        image_height = features.shape[2]
        image_width = features.shape[3]

        with torch.no_grad():
            encoded, z_mean, z_log_var, decoded_images = model(features)

        orig_images = features[:n_images]
        decoded_images = decoded_images[:n_images]
        break

    for i in range(n_images):
        for ax, img in zip(axes, [orig_images, decoded_images]):
            curr_img = img[i].detach().to(torch.device('cpu'))

            if color_channels > 1:
                curr_img = np.transpose(curr_img, (1, 2, 0))
                ax[i].imshow(curr_img)
            else:
                ax[i].imshow(curr_img.view((image_height, image_width)), cma
            ax[i].axis('off')

    axes[0, 0].set_title('Original', loc='left')
    axes[1, 0].set_title('Reconstructed', loc='left')


def plot_latent_space_with_labels(num_classes, data_loader, encoding_fn, dev
    """
    Visualize 2D latent space with class labels
    Only applicable for 2-dimensional latent spaces
    Shows how different digit classes are organized in latent space
    """
```

```
    d = {i:[] for i in range(num_classes)}

    with torch.no_grad():
        for i, (features, targets) in enumerate(data_loader):
            features = features.to(device)
            targets = targets.to(device)

            embedding = encoding_fn(features)

            for i in range(num_classes):
                if i in targets:
                    mask = targets == i
                    d[i].append(embedding[mask].to('cpu').numpy())

    colors = list(mcolors.TABLEAU_COLORS.items())
    for i in range(num_classes):
        d[i] = np.concatenate(d[i])
        plt.scatter(
            d[i][:, 0], d[i][:, 1],
            color=colors[i][1],
            label=f'{i}',
            alpha=0.5)

    plt.legend()


def plot_images_sampled_from_vae(model, device, latent_size, num_images=10):
    """
    Generate and visualize random samples from VAE decoder
    Samples latent vectors from N(0, I) and decodes to images
    """
    with torch.no_grad():
        # Random sample from latent space
        rand_features = torch.randn(num_images, latent_size).to(device)
        new_images = model.decoder(rand_features)
        color_channels = new_images.shape[1]
        image_height = new_images.shape[2]
        image_width = new_images.shape[3]

        # Visualization
        fig, axes = plt.subplots(nrows=1, ncols=num_images, figsize=(10, 2.5
        decoded_images = new_images[:num_images]

        for ax, img in zip(axes, decoded_images):
            curr_img = img.detach().to(torch.device('cpu'))

            if color_channels > 1:
                curr_img = np.transpose(curr_img, (1, 2, 0))
                ax.imshow(curr_img)
            else:
                ax.imshow(curr_img.view((image_height, image_width)), cmap='
            ax.axis('off')


def visualize_latent_interpolation(model, data_loader, device, n_steps=10):
    """
    Perform linear interpolation between two latent vectors
    Demonstrates smoothness and continuity of learned latent space
    """
    model.eval()
```

```
    with torch.no_grad():
        # Get two specific images
        img_a_data = next(iter(data_loader))[0][0].unsqueeze(0).to(device)
        img_b_data = next(iter(data_loader))[0][5].unsqueeze(0).to(device)

        # Encode them to get their latent vectors
        _, z_mean_a, z_log_var_a, _ = model(img_a_data)
        _, z_mean_b, z_log_var_b, _ = model(img_b_data)

        # Use the mean (mu) for interpolation
        z_a = z_mean_a
        z_b = z_mean_b

        # Create interpolation weights
        alphas = torch.linspace(0, 1, n_steps).to(device)
        interp_z = torch.stack([z_a * (1-a) + z_b * a for a in alphas])
        interp_z = interp_z.squeeze(1)

        # Decode the interpolated vectors
        interp_images = model.decoder(interp_z)

        # Visualize
        fig, axes = plt.subplots(nrows=1, ncols=n_steps, figsize=(15, 1.5))
        for i, ax in enumerate(axes):
            img = interp_images[i].detach().cpu()
            ax.imshow(img.squeeze(), cmap='binary')
            ax.axis('off')
        plt.suptitle('Latent Space Interpolation', y=1.05)
        plt.tight_layout()


def visualize_vae_results(model: VAE, log_dict):
    """
    Master visualization function combining all analysis plots
    Generates comprehensive visualization suite including:
        - Training loss curves (reconstruction, KL, combined)
        - Original vs. reconstructed images
        - Random generated samples
        - Latent space interpolation
        - 2D latent space visualization (if applicable)
    """
    print(f"\n{'='*60}")
    print(f"Visualizations for latent_dim={model.latent_dim}")
    print(f"{'='*60}\n")

    # 1. Plot Training Losses
    print("1. Plotting training losses...")
    plot_training_loss(log_dict['train_reconstruction_loss_per_batch'],
                       NUM_EPOCHS, custom_label=" (reconstruction)")
    plt.show()

    plot_training_loss(log_dict['train_kl_loss_per_batch'],
                       NUM_EPOCHS, custom_label=" (KL divergence)")
    plt.show()

    plot_training_loss(log_dict['train_combined_loss_per_batch'],
                       NUM_EPOCHS, custom_label=" (combined)")
    plt.show()
```

```
    # 2. Visualize Reconstructions
    print("2. Plotting reconstructions...")
    plot_generated_images(data_loader=test_loader, model=model, device=devic
    plt.suptitle(f'Original vs Reconstructed (latent_dim={model.latent_dim})
    plt.show()

    # 3. Visualize Generated Samples
    print("3. Plotting generated samples...")
    plot_images_sampled_from_vae(model=model, device=device, latent_size=mod
    plt.suptitle(f'Random Generated Samples (latent_dim={model.latent_dim})'
    plt.show()

    # 4. Latent Space Interpolation (Task requirement)
    print("4. Plotting latent space interpolation...")
    visualize_latent_interpolation(model=model, data_loader=test_loader, dev
    plt.show()

    # 5. Latent Space Visualization (only for 2D latent space)
    if model.latent_dim == 2:
        print("5. Plotting 2D latent space...")
        plt.figure(figsize=(8, 6))
        plot_latent_space_with_labels(num_classes=10,
                                      data_loader=test_loader,
                                      encoding_fn=model.encoding_fn,
                                      device=device)
        plt.title(f'Latent Space Visualization (latent_dim={model.latent_dim
        plt.xlabel('Latent Dimension 1')
        plt.ylabel('Latent Dimension 2')
        plt.show()

    print(f"\n{'='*60}\n")
```

In [7]:

```
# 8. Experiment 1: Latent Dimension = 128 (Task Requirement)

print("\n" + "="*60)
print("EXPERIMENT 1: Training VAE with latent_dim=128")
print("="*60)

latent_dim_128 = 128
model_128 = VAE(latent_dim=latent_dim_128).to(device)
optimizer_128 = optim.Adam(model_128.parameters(), lr=lr)

# Print model info
num_params = sum(p.numel() for p in model_128.parameters())
num_trainable_params = sum(p.numel() for p in model_128.parameters() if p.re
print(f'\nModel Architecture:')
print(f'  Total parameters: {num_params:,}')
print(f'  Trainable parameters: {num_trainable_params:,}')
print(f'  Latent dimension: {latent_dim_128}\n')

# Train the model
log_dict_128 = train_vae(model_128, optimizer_128, num_epochs=NUM_EPOCHS)
```

```
============================================================
EXPERIMENT 1: Training VAE with latent_dim=128
```

```
============================================================

Model Architecture:
  Total parameters: 1,356,449
  Trainable parameters: 1,356,449
  Latent dimension: 128


============================================================
Training VAE (latent_dim=128)
============================================================

/tmp/ipykernel_2092258/2433308939.py:72: UserWarning: Converting a tensor with
requires_grad=True to a scalar may lead to unexpected behavior.
Consider using tensor.detach() first. (Triggered internally at /pytorch/torch/
csrc/autograd/generated/python_variable_methods.cpp:835.)
  print('Epoch: %03d/%03d | Batch %04d/%04d | Loss: %.4f'
Epoch: 001/050 | Batch 0000/0235 | Loss: 219.1618
Epoch: 001/050 | Batch 0050/0235 | Loss: 72.0781
Epoch: 001/050 | Batch 0100/0235 | Loss: 63.0925
Epoch: 001/050 | Batch 0150/0235 | Loss: 57.1024
Epoch: 001/050 | Batch 0200/0235 | Loss: 53.3853
***Epoch: 001/050 | Loss: 44.421
Time elapsed: 0.17 min
Epoch: 002/050 | Batch 0000/0235 | Loss: 53.7350
Epoch: 002/050 | Batch 0050/0235 | Loss: 52.7291
Epoch: 002/050 | Batch 0100/0235 | Loss: 52.5287
Epoch: 002/050 | Batch 0150/0235 | Loss: 50.9431
Epoch: 002/050 | Batch 0200/0235 | Loss: 48.8629
***Epoch: 002/050 | Loss: 38.654
Time elapsed: 0.33 min
Epoch: 003/050 | Batch 0000/0235 | Loss: 48.3074
Epoch: 003/050 | Batch 0050/0235 | Loss: 47.7618
Epoch: 003/050 | Batch 0100/0235 | Loss: 46.7582
Epoch: 003/050 | Batch 0150/0235 | Loss: 46.6270
Epoch: 003/050 | Batch 0200/0235 | Loss: 46.0466
***Epoch: 003/050 | Loss: 34.846
Time elapsed: 0.49 min
Epoch: 004/050 | Batch 0000/0235 | Loss: 46.2175
Epoch: 004/050 | Batch 0050/0235 | Loss: 44.2661
Epoch: 004/050 | Batch 0100/0235 | Loss: 43.9580
Epoch: 004/050 | Batch 0150/0235 | Loss: 43.8825
Epoch: 004/050 | Batch 0200/0235 | Loss: 43.4101
***Epoch: 004/050 | Loss: 31.702
Time elapsed: 0.65 min
Epoch: 005/050 | Batch 0000/0235 | Loss: 41.6722
Epoch: 005/050 | Batch 0050/0235 | Loss: 41.8220
Epoch: 005/050 | Batch 0100/0235 | Loss: 42.3786
Epoch: 005/050 | Batch 0150/0235 | Loss: 42.6209
Epoch: 005/050 | Batch 0200/0235 | Loss: 41.2755
***Epoch: 005/050 | Loss: 28.843
Time elapsed: 0.81 min
Epoch: 006/050 | Batch 0000/0235 | Loss: 40.7457
Epoch: 006/050 | Batch 0050/0235 | Loss: 40.6901
Epoch: 006/050 | Batch 0100/0235 | Loss: 42.2872
Epoch: 006/050 | Batch 0150/0235 | Loss: 39.2788
Epoch: 006/050 | Batch 0200/0235 | Loss: 40.2772
***Epoch: 006/050 | Loss: 27.094
Time elapsed: 0.98 min
Epoch: 007/050 | Batch 0000/0235 | Loss: 38.5741
Epoch: 007/050 | Batch 0050/0235 | Loss: 39.6459
```

```
Epoch: 007/050 | Batch 0050/0235 | Loss: 39.6459
Epoch: 007/050 | Batch 0100/0235 | Loss: 38.2741
Epoch: 007/050 | Batch 0150/0235 | Loss: 37.3666
Epoch: 007/050 | Batch 0200/0235 | Loss: 39.4816
***Epoch: 007/050 | Loss: 25.500
Time elapsed: 1.14 min
Epoch: 008/050 | Batch 0000/0235 | Loss: 38.5740
Epoch: 008/050 | Batch 0050/0235 | Loss: 37.2283
Epoch: 008/050 | Batch 0100/0235 | Loss: 37.1840
Epoch: 008/050 | Batch 0150/0235 | Loss: 36.7908
Epoch: 008/050 | Batch 0200/0235 | Loss: 37.3703
***Epoch: 008/050 | Loss: 24.316
Time elapsed: 1.30 min
Epoch: 009/050 | Batch 0000/0235 | Loss: 35.9428
Epoch: 009/050 | Batch 0050/0235 | Loss: 37.4715
Epoch: 009/050 | Batch 0100/0235 | Loss: 36.7753
Epoch: 009/050 | Batch 0150/0235 | Loss: 36.1522
Epoch: 009/050 | Batch 0200/0235 | Loss: 36.4814
***Epoch: 009/050 | Loss: 23.458
Time elapsed: 1.47 min
Epoch: 010/050 | Batch 0000/0235 | Loss: 36.6487
Epoch: 010/050 | Batch 0050/0235 | Loss: 36.3005
Epoch: 010/050 | Batch 0100/0235 | Loss: 36.4083
Epoch: 010/050 | Batch 0150/0235 | Loss: 35.0151
Epoch: 010/050 | Batch 0200/0235 | Loss: 35.7211
***Epoch: 010/050 | Loss: 22.655
Time elapsed: 1.63 min
Epoch: 011/050 | Batch 0000/0235 | Loss: 35.8999
Epoch: 011/050 | Batch 0050/0235 | Loss: 35.9487
Epoch: 011/050 | Batch 0100/0235 | Loss: 35.0637
Epoch: 011/050 | Batch 0150/0235 | Loss: 35.6152
Epoch: 011/050 | Batch 0200/0235 | Loss: 33.8289
***Epoch: 011/050 | Loss: 22.456
Time elapsed: 1.79 min
Epoch: 012/050 | Batch 0000/0235 | Loss: 33.9455
Epoch: 012/050 | Batch 0050/0235 | Loss: 35.2498
Epoch: 012/050 | Batch 0100/0235 | Loss: 34.1436
Epoch: 012/050 | Batch 0150/0235 | Loss: 34.2126
Epoch: 012/050 | Batch 0200/0235 | Loss: 34.8147
***Epoch: 012/050 | Loss: 21.948
Time elapsed: 1.96 min
Epoch: 013/050 | Batch 0000/0235 | Loss: 35.9923
Epoch: 013/050 | Batch 0050/0235 | Loss: 34.6112
Epoch: 013/050 | Batch 0100/0235 | Loss: 34.3137
Epoch: 013/050 | Batch 0150/0235 | Loss: 34.3755
Epoch: 013/050 | Batch 0200/0235 | Loss: 34.9138
***Epoch: 013/050 | Loss: 21.465
Time elapsed: 2.13 min
Epoch: 014/050 | Batch 0000/0235 | Loss: 35.4255
Epoch: 014/050 | Batch 0050/0235 | Loss: 35.1086
Epoch: 014/050 | Batch 0100/0235 | Loss: 34.0945
Epoch: 014/050 | Batch 0150/0235 | Loss: 35.0226
Epoch: 014/050 | Batch 0200/0235 | Loss: 34.7602
***Epoch: 014/050 | Loss: 20.915
Time elapsed: 2.29 min
Epoch: 015/050 | Batch 0000/0235 | Loss: 36.4453
Epoch: 015/050 | Batch 0050/0235 | Loss: 33.7851
Epoch: 015/050 | Batch 0100/0235 | Loss: 33.7334
Epoch: 015/050 | Batch 0150/0235 | Loss: 33.8425
Epoch: 015/050 | Batch 0200/0235 | Loss: 34.1678
***Epoch: 015/050 | Loss: 20.792
```

```
Time elapsed: 2.46 min
Epoch: 016/050 | Batch 0000/0235 | Loss: 34.8186
Epoch: 016/050 | Batch 0050/0235 | Loss: 34.5552
Epoch: 016/050 | Batch 0100/0235 | Loss: 34.4791
Epoch: 016/050 | Batch 0150/0235 | Loss: 34.2465
Epoch: 016/050 | Batch 0200/0235 | Loss: 33.4789
***Epoch: 016/050 | Loss: 20.391
Time elapsed: 2.63 min
Epoch: 017/050 | Batch 0000/0235 | Loss: 33.5132
Epoch: 017/050 | Batch 0050/0235 | Loss: 34.2799
Epoch: 017/050 | Batch 0100/0235 | Loss: 33.8354
Epoch: 017/050 | Batch 0150/0235 | Loss: 34.1745
Epoch: 017/050 | Batch 0200/0235 | Loss: 33.3070
***Epoch: 017/050 | Loss: 20.161
Time elapsed: 2.79 min
Epoch: 018/050 | Batch 0000/0235 | Loss: 33.8666
Epoch: 018/050 | Batch 0050/0235 | Loss: 32.8168
Epoch: 018/050 | Batch 0100/0235 | Loss: 33.2269
Epoch: 018/050 | Batch 0150/0235 | Loss: 33.9105
Epoch: 018/050 | Batch 0200/0235 | Loss: 32.5472
***Epoch: 018/050 | Loss: 20.255
Time elapsed: 2.95 min
Epoch: 019/050 | Batch 0000/0235 | Loss: 32.4776
Epoch: 019/050 | Batch 0050/0235 | Loss: 33.2655
Epoch: 019/050 | Batch 0100/0235 | Loss: 33.3305
Epoch: 019/050 | Batch 0150/0235 | Loss: 32.5378
Epoch: 019/050 | Batch 0200/0235 | Loss: 32.6440
***Epoch: 019/050 | Loss: 20.036
Time elapsed: 3.12 min
Epoch: 020/050 | Batch 0000/0235 | Loss: 34.1818
Epoch: 020/050 | Batch 0050/0235 | Loss: 33.5868
Epoch: 020/050 | Batch 0100/0235 | Loss: 33.4985
Epoch: 020/050 | Batch 0150/0235 | Loss: 33.1935
Epoch: 020/050 | Batch 0200/0235 | Loss: 33.4980
***Epoch: 020/050 | Loss: 19.765
Time elapsed: 3.29 min
Epoch: 021/050 | Batch 0000/0235 | Loss: 33.0823
Epoch: 021/050 | Batch 0050/0235 | Loss: 33.0792
Epoch: 021/050 | Batch 0100/0235 | Loss: 33.2055
Epoch: 021/050 | Batch 0150/0235 | Loss: 32.7873
Epoch: 021/050 | Batch 0200/0235 | Loss: 33.7521
***Epoch: 021/050 | Loss: 19.489
Time elapsed: 3.45 min
Epoch: 022/050 | Batch 0000/0235 | Loss: 33.5743
Epoch: 022/050 | Batch 0050/0235 | Loss: 32.1080
Epoch: 022/050 | Batch 0100/0235 | Loss: 32.9883
Epoch: 022/050 | Batch 0150/0235 | Loss: 34.0091
Epoch: 022/050 | Batch 0200/0235 | Loss: 33.2372
***Epoch: 022/050 | Loss: 19.830
Time elapsed: 3.62 min
Epoch: 023/050 | Batch 0000/0235 | Loss: 32.0502
Epoch: 023/050 | Batch 0050/0235 | Loss: 33.5525
Epoch: 023/050 | Batch 0100/0235 | Loss: 32.1611
Epoch: 023/050 | Batch 0150/0235 | Loss: 33.2771
Epoch: 023/050 | Batch 0200/0235 | Loss: 32.6716
***Epoch: 023/050 | Loss: 19.539
Time elapsed: 3.78 min
Epoch: 024/050 | Batch 0000/0235 | Loss: 33.5601
Epoch: 024/050 | Batch 0050/0235 | Loss: 32.7369
Epoch: 024/050 | Batch 0100/0235 | Loss: 32.5580
```

```
Epoch: 024/050 | Batch 0150/0235 | Loss: 32.3683
Epoch: 024/050 | Batch 0200/0235 | Loss: 32.4425
***Epoch: 024/050 | Loss: 19.391
Time elapsed: 3.95 min
Epoch: 025/050 | Batch 0000/0235 | Loss: 33.0540
Epoch: 025/050 | Batch 0050/0235 | Loss: 33.7649
Epoch: 025/050 | Batch 0100/0235 | Loss: 31.8850
Epoch: 025/050 | Batch 0150/0235 | Loss: 31.5690
Epoch: 025/050 | Batch 0200/0235 | Loss: 33.4082
***Epoch: 025/050 | Loss: 19.347
Time elapsed: 4.12 min
Epoch: 026/050 | Batch 0000/0235 | Loss: 33.8353
Epoch: 026/050 | Batch 0050/0235 | Loss: 32.5862
Epoch: 026/050 | Batch 0100/0235 | Loss: 32.4167
Epoch: 026/050 | Batch 0150/0235 | Loss: 31.9531
Epoch: 026/050 | Batch 0200/0235 | Loss: 32.5308
***Epoch: 026/050 | Loss: 19.231
Time elapsed: 4.28 min
Epoch: 027/050 | Batch 0000/0235 | Loss: 32.6444
Epoch: 027/050 | Batch 0050/0235 | Loss: 32.3241
Epoch: 027/050 | Batch 0100/0235 | Loss: 34.1610
Epoch: 027/050 | Batch 0150/0235 | Loss: 32.6298
Epoch: 027/050 | Batch 0200/0235 | Loss: 32.2400
***Epoch: 027/050 | Loss: 19.159
Time elapsed: 4.45 min
Epoch: 028/050 | Batch 0000/0235 | Loss: 33.0377
Epoch: 028/050 | Batch 0050/0235 | Loss: 32.3901
Epoch: 028/050 | Batch 0100/0235 | Loss: 32.1896
Epoch: 028/050 | Batch 0150/0235 | Loss: 31.9150
Epoch: 028/050 | Batch 0200/0235 | Loss: 32.9372
***Epoch: 028/050 | Loss: 19.069
Time elapsed: 4.61 min
Epoch: 029/050 | Batch 0000/0235 | Loss: 31.8515
Epoch: 029/050 | Batch 0050/0235 | Loss: 33.0969
Epoch: 029/050 | Batch 0100/0235 | Loss: 32.0331
Epoch: 029/050 | Batch 0150/0235 | Loss: 33.2257
Epoch: 029/050 | Batch 0200/0235 | Loss: 32.0223
***Epoch: 029/050 | Loss: 18.700
Time elapsed: 4.78 min
Epoch: 030/050 | Batch 0000/0235 | Loss: 32.8974
Epoch: 030/050 | Batch 0050/0235 | Loss: 32.2600
Epoch: 030/050 | Batch 0100/0235 | Loss: 32.2947
Epoch: 030/050 | Batch 0150/0235 | Loss: 32.9533
Epoch: 030/050 | Batch 0200/0235 | Loss: 32.1209
***Epoch: 030/050 | Loss: 18.873
Time elapsed: 4.95 min
Epoch: 031/050 | Batch 0000/0235 | Loss: 31.7298
Epoch: 031/050 | Batch 0050/0235 | Loss: 32.8133
Epoch: 031/050 | Batch 0100/0235 | Loss: 33.5317
Epoch: 031/050 | Batch 0150/0235 | Loss: 32.8114
Epoch: 031/050 | Batch 0200/0235 | Loss: 31.8194
***Epoch: 031/050 | Loss: 18.691
Time elapsed: 5.11 min
Epoch: 032/050 | Batch 0000/0235 | Loss: 31.4657
Epoch: 032/050 | Batch 0050/0235 | Loss: 33.0770
Epoch: 032/050 | Batch 0100/0235 | Loss: 31.3314
Epoch: 032/050 | Batch 0150/0235 | Loss: 31.5241
Epoch: 032/050 | Batch 0200/0235 | Loss: 32.5614
***Epoch: 032/050 | Loss: 18.728
Time elapsed: 5.27 min
Epoch: 033/050 | Batch 0000/0235 | Loss: 33.1498
```

```
Epoch: 033/050 | Batch 0000/0235 | Loss: 33.1498
Epoch: 033/050 | Batch 0050/0235 | Loss: 31.8522
Epoch: 033/050 | Batch 0100/0235 | Loss: 32.2077
Epoch: 033/050 | Batch 0150/0235 | Loss: 32.1488
Epoch: 033/050 | Batch 0200/0235 | Loss: 32.3270
***Epoch: 033/050 | Loss: 18.522
Time elapsed: 5.44 min
Epoch: 034/050 | Batch 0000/0235 | Loss: 32.3359
Epoch: 034/050 | Batch 0050/0235 | Loss: 32.2731
Epoch: 034/050 | Batch 0100/0235 | Loss: 32.0168
Epoch: 034/050 | Batch 0150/0235 | Loss: 32.1199
Epoch: 034/050 | Batch 0200/0235 | Loss: 32.0589
***Epoch: 034/050 | Loss: 18.549
Time elapsed: 5.60 min
Epoch: 035/050 | Batch 0000/0235 | Loss: 31.7229
Epoch: 035/050 | Batch 0050/0235 | Loss: 32.5660
Epoch: 035/050 | Batch 0100/0235 | Loss: 31.2169
Epoch: 035/050 | Batch 0150/0235 | Loss: 32.0989
Epoch: 035/050 | Batch 0200/0235 | Loss: 31.7755
***Epoch: 035/050 | Loss: 18.357
Time elapsed: 5.77 min
Epoch: 036/050 | Batch 0000/0235 | Loss: 32.6012
Epoch: 036/050 | Batch 0050/0235 | Loss: 31.5900
Epoch: 036/050 | Batch 0100/0235 | Loss: 31.2491
Epoch: 036/050 | Batch 0150/0235 | Loss: 31.5739
Epoch: 036/050 | Batch 0200/0235 | Loss: 31.4293
***Epoch: 036/050 | Loss: 18.430
Time elapsed: 5.94 min
Epoch: 037/050 | Batch 0000/0235 | Loss: 30.9012
Epoch: 037/050 | Batch 0050/0235 | Loss: 31.7156
Epoch: 037/050 | Batch 0100/0235 | Loss: 32.0657
Epoch: 037/050 | Batch 0150/0235 | Loss: 31.3055
Epoch: 037/050 | Batch 0200/0235 | Loss: 31.5401
***Epoch: 037/050 | Loss: 18.304
Time elapsed: 6.10 min
Epoch: 038/050 | Batch 0000/0235 | Loss: 31.8038
Epoch: 038/050 | Batch 0050/0235 | Loss: 31.8536
Epoch: 038/050 | Batch 0100/0235 | Loss: 32.0707
Epoch: 038/050 | Batch 0150/0235 | Loss: 31.8120
Epoch: 038/050 | Batch 0200/0235 | Loss: 31.2381
***Epoch: 038/050 | Loss: 18.376
Time elapsed: 6.27 min
Epoch: 039/050 | Batch 0000/0235 | Loss: 31.3298
Epoch: 039/050 | Batch 0050/0235 | Loss: 32.0692
Epoch: 039/050 | Batch 0100/0235 | Loss: 32.3891
Epoch: 039/050 | Batch 0150/0235 | Loss: 30.6434
Epoch: 039/050 | Batch 0200/0235 | Loss: 31.9648
***Epoch: 039/050 | Loss: 18.252
Time elapsed: 6.44 min
Epoch: 040/050 | Batch 0000/0235 | Loss: 30.3210
Epoch: 040/050 | Batch 0050/0235 | Loss: 31.9994
Epoch: 040/050 | Batch 0100/0235 | Loss: 32.1069
Epoch: 040/050 | Batch 0150/0235 | Loss: 31.2559
Epoch: 040/050 | Batch 0200/0235 | Loss: 31.5479
***Epoch: 040/050 | Loss: 17.997
Time elapsed: 6.60 min
Epoch: 041/050 | Batch 0000/0235 | Loss: 31.9019
Epoch: 041/050 | Batch 0050/0235 | Loss: 32.1314
Epoch: 041/050 | Batch 0100/0235 | Loss: 31.4775
Epoch: 041/050 | Batch 0150/0235 | Loss: 31.5015
Epoch: 041/050 | Batch 0200/0235 | Loss: 31.5227
```

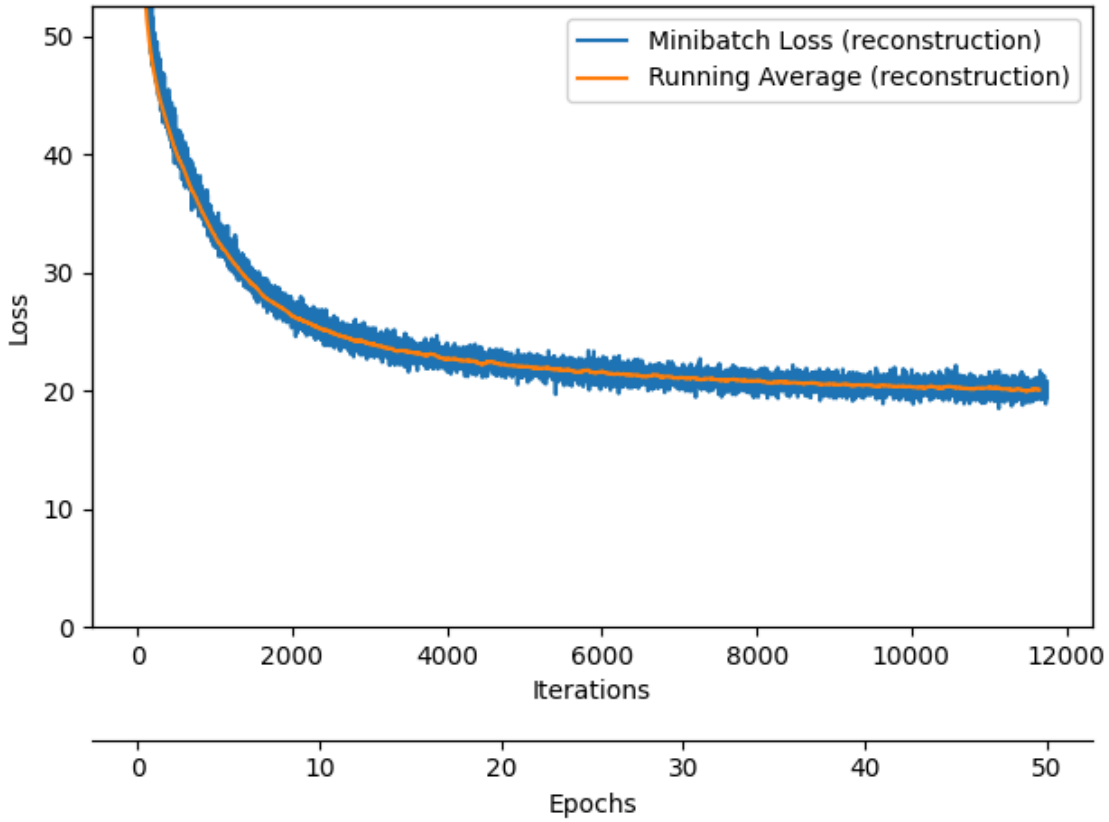
```
***Epoch: 041/050 | Loss: 18.175
Time elapsed: 6.77 min
Epoch: 042/050 | Batch 0000/0235 | Loss: 31.5196
Epoch: 042/050 | Batch 0050/0235 | Loss: 32.8241
Epoch: 042/050 | Batch 0100/0235 | Loss: 31.3997
Epoch: 042/050 | Batch 0150/0235 | Loss: 31.7879
Epoch: 042/050 | Batch 0200/0235 | Loss: 31.3640
***Epoch: 042/050 | Loss: 17.947
Time elapsed: 6.93 min
Epoch: 043/050 | Batch 0000/0235 | Loss: 31.3218
Epoch: 043/050 | Batch 0050/0235 | Loss: 32.5348
Epoch: 043/050 | Batch 0100/0235 | Loss: 31.2017
Epoch: 043/050 | Batch 0150/0235 | Loss: 30.4086
Epoch: 043/050 | Batch 0200/0235 | Loss: 30.8656
***Epoch: 043/050 | Loss: 18.160
Time elapsed: 7.10 min
Epoch: 044/050 | Batch 0000/0235 | Loss: 30.4538
Epoch: 044/050 | Batch 0050/0235 | Loss: 32.9784
Epoch: 044/050 | Batch 0100/0235 | Loss: 31.4776
Epoch: 044/050 | Batch 0150/0235 | Loss: 31.6348
Epoch: 044/050 | Batch 0200/0235 | Loss: 30.8142
***Epoch: 044/050 | Loss: 18.102
Time elapsed: 7.26 min
Epoch: 045/050 | Batch 0000/0235 | Loss: 32.0799
Epoch: 045/050 | Batch 0050/0235 | Loss: 32.5768
Epoch: 045/050 | Batch 0100/0235 | Loss: 31.3737
Epoch: 045/050 | Batch 0150/0235 | Loss: 32.6212
Epoch: 045/050 | Batch 0200/0235 | Loss: 32.4732
***Epoch: 045/050 | Loss: 17.727
Time elapsed: 7.43 min
Epoch: 046/050 | Batch 0000/0235 | Loss: 31.4941
Epoch: 046/050 | Batch 0050/0235 | Loss: 32.5246
Epoch: 046/050 | Batch 0100/0235 | Loss: 31.0880
Epoch: 046/050 | Batch 0150/0235 | Loss: 31.8074
Epoch: 046/050 | Batch 0200/0235 | Loss: 30.1301
***Epoch: 046/050 | Loss: 17.595
Time elapsed: 7.59 min
Epoch: 047/050 | Batch 0000/0235 | Loss: 32.3680
Epoch: 047/050 | Batch 0050/0235 | Loss: 32.3451
Epoch: 047/050 | Batch 0100/0235 | Loss: 30.4709
Epoch: 047/050 | Batch 0150/0235 | Loss: 32.0290
Epoch: 047/050 | Batch 0200/0235 | Loss: 32.2746
***Epoch: 047/050 | Loss: 18.156
Time elapsed: 7.76 min
Epoch: 048/050 | Batch 0000/0235 | Loss: 30.8849
Epoch: 048/050 | Batch 0050/0235 | Loss: 31.4924
Epoch: 048/050 | Batch 0100/0235 | Loss: 31.7589
Epoch: 048/050 | Batch 0150/0235 | Loss: 30.3866
Epoch: 048/050 | Batch 0200/0235 | Loss: 31.2584
***Epoch: 048/050 | Loss: 17.909
Time elapsed: 7.93 min
Epoch: 049/050 | Batch 0000/0235 | Loss: 31.3384
Epoch: 049/050 | Batch 0050/0235 | Loss: 30.3947
Epoch: 049/050 | Batch 0100/0235 | Loss: 30.6272
Epoch: 049/050 | Batch 0150/0235 | Loss: 31.7445
Epoch: 049/050 | Batch 0200/0235 | Loss: 31.1291
***Epoch: 049/050 | Loss: 18.029
Time elapsed: 8.09 min
Epoch: 050/050 | Batch 0000/0235 | Loss: 30.1144
Epoch: 050/050 | Batch 0050/0235 | Loss: 30.8653
```

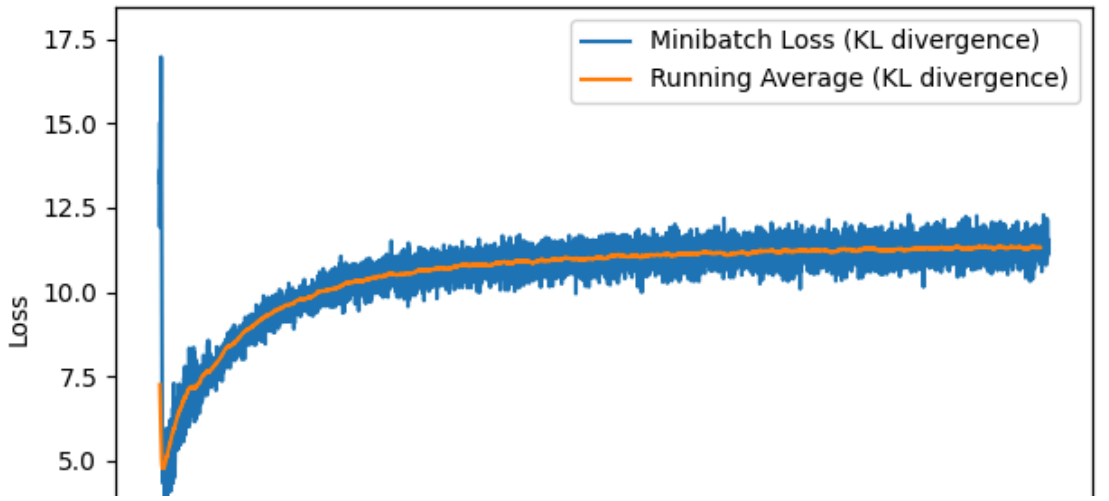
```
Epoch: 050/050 | Batch 0100/0235 | Loss: 31.8640
Epoch: 050/050 | Batch 0150/0235 | Loss: 31.7108
Epoch: 050/050 | Batch 0200/0235 | Loss: 31.4018
***Epoch: 050/050 | Loss: 17.790
Time elapsed: 8.26 min

Total Training Time: 8.26 min
============================================================
```

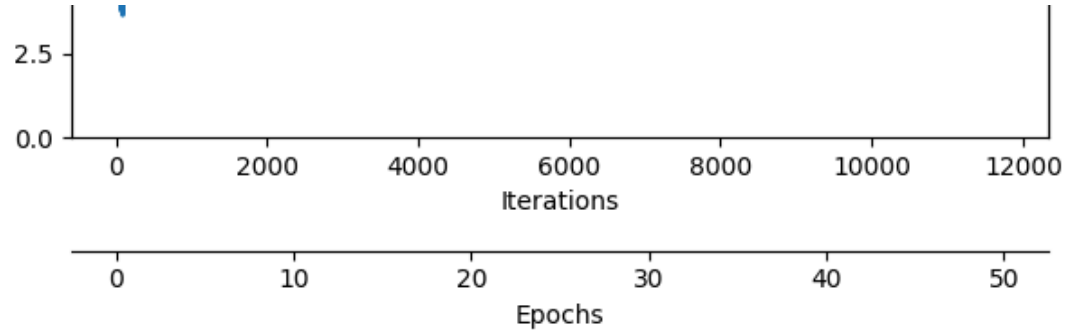
In [8]:

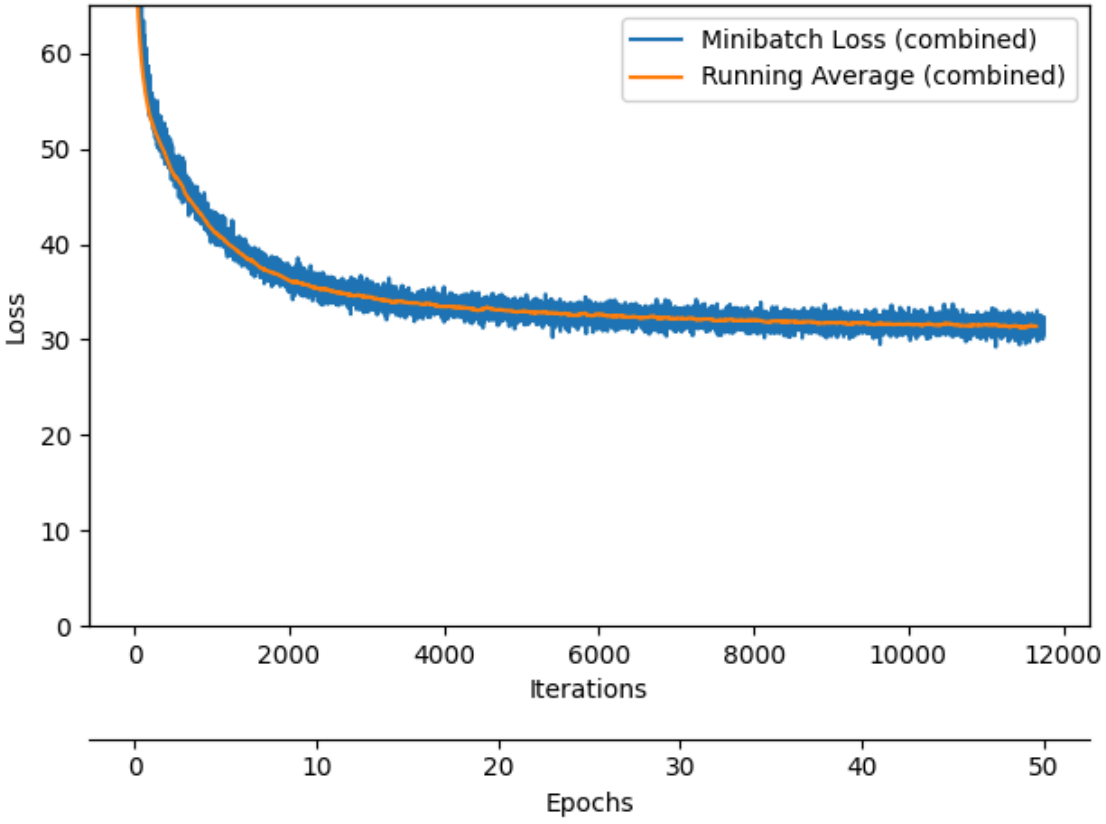
```
# 9. Visualize Experiment 1 Results
visualize_vae_results(model_128, log_dict_128)
```

```
============================================================
Visualizations for latent_dim=128
============================================================

1. Plotting training losses...
```

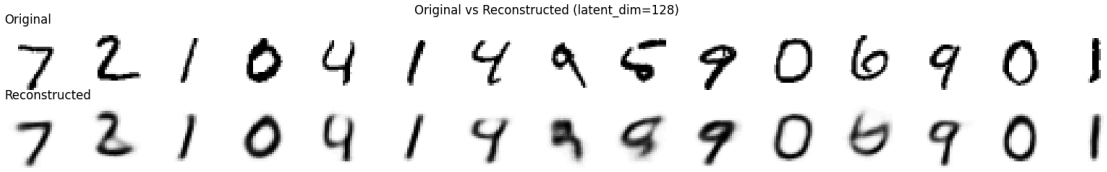




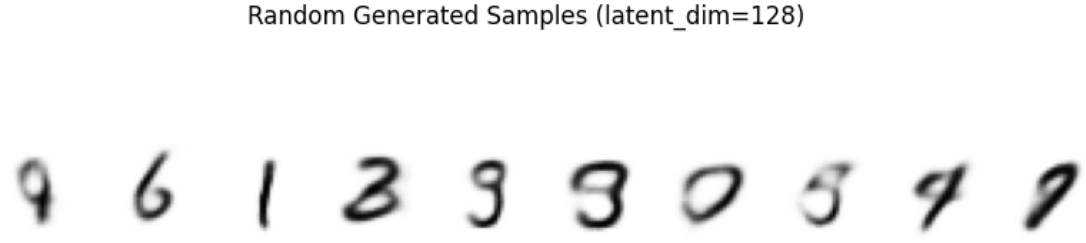





```
2. Plotting reconstructions...
```




```
3. Plotting generated samples...
```




```
4. Plotting latent space interpolation...
```


```
============================================================
```

In [9]:

```
# 10. Experiment 2: Latent Dimension = 256 (Task Requirement)

print("\n" + "="*60)
print("EXPERIMENT 2: Training VAE with latent_dim=256")
print("="*60)

latent_dim_256 = 256
model_256 = VAE(latent_dim=latent_dim_256).to(device)
optimizer_256 = optim.Adam(model_256.parameters(), lr=lr)

# Print model info
num_params = sum(p.numel() for p in model_256.parameters())
num_trainable_params = sum(p.numel() for p in model_256.parameters() if p.re
print(f'\nModel Architecture:')
print(f'  Total parameters: {num_params:,}')
print(f'  Trainable parameters: {num_trainable_params:,}')
print(f'  Latent dimension: {latent_dim_256}\n')

# Train the model
log_dict_256 = train_vae(model_256, optimizer_256, num_epochs=NUM_EPOCHS)
```

```
============================================================
EXPERIMENT 2: Training VAE with latent_dim=256
============================================================

Model Architecture:
  Total parameters: 2,560,929
  Trainable parameters: 2,560,929
  Latent dimension: 256


============================================================
Training VAE (latent_dim=256)
============================================================

Epoch: 001/050 | Batch 0000/0235 | Loss: 189.0981
Epoch: 001/050 | Batch 0050/0235 | Loss: 74.1985
Epoch: 001/050 | Batch 0100/0235 | Loss: 62.3855
Epoch: 001/050 | Batch 0150/0235 | Loss: 57.4906
Epoch: 001/050 | Batch 0200/0235 | Loss: 57.9790
***Epoch: 001/050 | Loss: 46.005
Time elapsed: 0.17 min
Epoch: 002/050 | Batch 0000/0235 | Loss: 52.7639
Epoch: 002/050 | Batch 0050/0235 | Loss: 52.7641
Epoch: 002/050 | Batch 0100/0235 | Loss: 52.0576
Epoch: 002/050 | Batch 0150/0235 | Loss: 50.1981
Epoch: 002/050 | Batch 0200/0235 | Loss: 49.3261
***Epoch: 002/050 | Loss: 39.297
Time elapsed: 0.34 min
Epoch: 003/050 | Batch 0000/0235 | Loss: 48.5222
Epoch: 003/050 | Batch 0050/0235 | Loss: 49.4514
Epoch: 003/050 | Batch 0100/0235 | Loss: 47.5893
Epoch: 003/050 | Batch 0150/0235 | Loss: 46.6423
Epoch: 003/050 | Batch 0200/0235 | Loss: 46.8372
***Epoch: 003/050 | Loss: 36.406
Time elapsed: 0.50 min
Epoch: 004/050 | Batch 0000/0235 | Loss: 47.3618
```

```
Epoch: 004/050 | Batch 0050/0235 | Loss: 46.3837
Epoch: 004/050 | Batch 0100/0235 | Loss: 44.0311
Epoch: 004/050 | Batch 0150/0235 | Loss: 45.9513
Epoch: 004/050 | Batch 0200/0235 | Loss: 44.9723
***Epoch: 004/050 | Loss: 34.653
Time elapsed: 0.67 min
Epoch: 005/050 | Batch 0000/0235 | Loss: 45.0553
Epoch: 005/050 | Batch 0050/0235 | Loss: 44.2130
Epoch: 005/050 | Batch 0100/0235 | Loss: 45.8353
Epoch: 005/050 | Batch 0150/0235 | Loss: 45.3810
Epoch: 005/050 | Batch 0200/0235 | Loss: 43.2245
***Epoch: 005/050 | Loss: 33.078
Time elapsed: 0.84 min
Epoch: 006/050 | Batch 0000/0235 | Loss: 43.2173
Epoch: 006/050 | Batch 0050/0235 | Loss: 43.3262
Epoch: 006/050 | Batch 0100/0235 | Loss: 43.5814
Epoch: 006/050 | Batch 0150/0235 | Loss: 42.2398
Epoch: 006/050 | Batch 0200/0235 | Loss: 42.8613
***Epoch: 006/050 | Loss: 31.409
Time elapsed: 1.01 min
Epoch: 007/050 | Batch 0000/0235 | Loss: 42.9683
Epoch: 007/050 | Batch 0050/0235 | Loss: 42.5879
Epoch: 007/050 | Batch 0100/0235 | Loss: 42.6196
Epoch: 007/050 | Batch 0150/0235 | Loss: 40.6587
Epoch: 007/050 | Batch 0200/0235 | Loss: 40.9488
***Epoch: 007/050 | Loss: 29.257
Time elapsed: 1.17 min
Epoch: 008/050 | Batch 0000/0235 | Loss: 40.3629
Epoch: 008/050 | Batch 0050/0235 | Loss: 40.3439
Epoch: 008/050 | Batch 0100/0235 | Loss: 40.8431
Epoch: 008/050 | Batch 0150/0235 | Loss: 40.0628
Epoch: 008/050 | Batch 0200/0235 | Loss: 39.4768
***Epoch: 008/050 | Loss: 27.241
Time elapsed: 1.34 min
Epoch: 009/050 | Batch 0000/0235 | Loss: 39.7659
Epoch: 009/050 | Batch 0050/0235 | Loss: 38.7267
Epoch: 009/050 | Batch 0100/0235 | Loss: 38.2556
Epoch: 009/050 | Batch 0150/0235 | Loss: 38.0541
Epoch: 009/050 | Batch 0200/0235 | Loss: 38.0012
***Epoch: 009/050 | Loss: 25.932
Time elapsed: 1.51 min
Epoch: 010/050 | Batch 0000/0235 | Loss: 37.7867
Epoch: 010/050 | Batch 0050/0235 | Loss: 39.2094
Epoch: 010/050 | Batch 0100/0235 | Loss: 37.0845
Epoch: 010/050 | Batch 0150/0235 | Loss: 36.5773
Epoch: 010/050 | Batch 0200/0235 | Loss: 37.7501
***Epoch: 010/050 | Loss: 24.809
Time elapsed: 1.67 min
Epoch: 011/050 | Batch 0000/0235 | Loss: 35.9955
Epoch: 011/050 | Batch 0050/0235 | Loss: 36.9625
Epoch: 011/050 | Batch 0100/0235 | Loss: 36.4461
Epoch: 011/050 | Batch 0150/0235 | Loss: 36.2212
Epoch: 011/050 | Batch 0200/0235 | Loss: 37.9003
***Epoch: 011/050 | Loss: 24.381
Time elapsed: 1.84 min
Epoch: 012/050 | Batch 0000/0235 | Loss: 35.6576
Epoch: 012/050 | Batch 0050/0235 | Loss: 36.3865
Epoch: 012/050 | Batch 0100/0235 | Loss: 35.0507
Epoch: 012/050 | Batch 0150/0235 | Loss: 36.5570
Epoch: 012/050 | Batch 0200/0235 | Loss: 35.5136
```

```
***Epoch: 012/050 | Loss: 23.768
Time elapsed: 2.01 min
Epoch: 013/050 | Batch 0000/0235 | Loss: 35.2724
Epoch: 013/050 | Batch 0050/0235 | Loss: 36.6468
Epoch: 013/050 | Batch 0100/0235 | Loss: 35.1636
Epoch: 013/050 | Batch 0150/0235 | Loss: 35.4243
Epoch: 013/050 | Batch 0200/0235 | Loss: 34.4914
***Epoch: 013/050 | Loss: 23.530
Time elapsed: 2.18 min
Epoch: 014/050 | Batch 0000/0235 | Loss: 36.7634
Epoch: 014/050 | Batch 0050/0235 | Loss: 35.2822
Epoch: 014/050 | Batch 0100/0235 | Loss: 35.2191
Epoch: 014/050 | Batch 0150/0235 | Loss: 34.5768
Epoch: 014/050 | Batch 0200/0235 | Loss: 34.3704
***Epoch: 014/050 | Loss: 23.067
Time elapsed: 2.35 min
Epoch: 015/050 | Batch 0000/0235 | Loss: 34.9309
Epoch: 015/050 | Batch 0050/0235 | Loss: 35.4617
Epoch: 015/050 | Batch 0100/0235 | Loss: 34.2557
Epoch: 015/050 | Batch 0150/0235 | Loss: 34.9455
Epoch: 015/050 | Batch 0200/0235 | Loss: 35.3708
***Epoch: 015/050 | Loss: 22.724
Time elapsed: 2.52 min
Epoch: 016/050 | Batch 0000/0235 | Loss: 35.3181
Epoch: 016/050 | Batch 0050/0235 | Loss: 34.9897
Epoch: 016/050 | Batch 0100/0235 | Loss: 35.3585
Epoch: 016/050 | Batch 0150/0235 | Loss: 33.0960
Epoch: 016/050 | Batch 0200/0235 | Loss: 34.6769
***Epoch: 016/050 | Loss: 22.486
Time elapsed: 2.68 min
Epoch: 017/050 | Batch 0000/0235 | Loss: 34.0786
Epoch: 017/050 | Batch 0050/0235 | Loss: 34.1224
Epoch: 017/050 | Batch 0100/0235 | Loss: 33.9985
Epoch: 017/050 | Batch 0150/0235 | Loss: 33.4144
Epoch: 017/050 | Batch 0200/0235 | Loss: 34.9046
***Epoch: 017/050 | Loss: 22.279
Time elapsed: 2.85 min
Epoch: 018/050 | Batch 0000/0235 | Loss: 34.3676
Epoch: 018/050 | Batch 0050/0235 | Loss: 35.0968
Epoch: 018/050 | Batch 0100/0235 | Loss: 33.7548
Epoch: 018/050 | Batch 0150/0235 | Loss: 34.9720
Epoch: 018/050 | Batch 0200/0235 | Loss: 33.7409
***Epoch: 018/050 | Loss: 21.824
Time elapsed: 3.02 min
Epoch: 019/050 | Batch 0000/0235 | Loss: 34.0074
Epoch: 019/050 | Batch 0050/0235 | Loss: 35.6929
Epoch: 019/050 | Batch 0100/0235 | Loss: 34.8702
Epoch: 019/050 | Batch 0150/0235 | Loss: 35.0033
Epoch: 019/050 | Batch 0200/0235 | Loss: 34.1487
***Epoch: 019/050 | Loss: 21.505
Time elapsed: 3.19 min
Epoch: 020/050 | Batch 0000/0235 | Loss: 34.5473
Epoch: 020/050 | Batch 0050/0235 | Loss: 33.0986
Epoch: 020/050 | Batch 0100/0235 | Loss: 33.6383
Epoch: 020/050 | Batch 0150/0235 | Loss: 33.4454
Epoch: 020/050 | Batch 0200/0235 | Loss: 34.1152
***Epoch: 020/050 | Loss: 21.056
Time elapsed: 3.36 min
Epoch: 021/050 | Batch 0000/0235 | Loss: 34.8738
Epoch: 021/050 | Batch 0050/0235 | Loss: 32.9101
Epoch: 021/050 | Batch 0100/0235 | Loss: 33.8373
```

```
Epoch: 021/050 | Batch 0150/0235 | Loss: 33.5661
Epoch: 021/050 | Batch 0200/0235 | Loss: 34.3480
***Epoch: 021/050 | Loss: 20.989
Time elapsed: 3.53 min
Epoch: 022/050 | Batch 0000/0235 | Loss: 31.8602
Epoch: 022/050 | Batch 0050/0235 | Loss: 33.4229
Epoch: 022/050 | Batch 0100/0235 | Loss: 33.4088
Epoch: 022/050 | Batch 0150/0235 | Loss: 34.4664
Epoch: 022/050 | Batch 0200/0235 | Loss: 33.8753
***Epoch: 022/050 | Loss: 20.578
Time elapsed: 3.70 min
Epoch: 023/050 | Batch 0000/0235 | Loss: 33.3183
Epoch: 023/050 | Batch 0050/0235 | Loss: 33.3628
Epoch: 023/050 | Batch 0100/0235 | Loss: 32.3277
Epoch: 023/050 | Batch 0150/0235 | Loss: 33.4086
Epoch: 023/050 | Batch 0200/0235 | Loss: 33.9410
***Epoch: 023/050 | Loss: 20.072
Time elapsed: 3.87 min
Epoch: 024/050 | Batch 0000/0235 | Loss: 33.3864
Epoch: 024/050 | Batch 0050/0235 | Loss: 33.1008
Epoch: 024/050 | Batch 0100/0235 | Loss: 34.8314
Epoch: 024/050 | Batch 0150/0235 | Loss: 34.2452
Epoch: 024/050 | Batch 0200/0235 | Loss: 32.5181
***Epoch: 024/050 | Loss: 20.230
Time elapsed: 4.04 min
Epoch: 025/050 | Batch 0000/0235 | Loss: 32.5606
Epoch: 025/050 | Batch 0050/0235 | Loss: 32.3203
Epoch: 025/050 | Batch 0100/0235 | Loss: 32.9889
Epoch: 025/050 | Batch 0150/0235 | Loss: 32.5280
Epoch: 025/050 | Batch 0200/0235 | Loss: 32.6312
***Epoch: 025/050 | Loss: 19.976
Time elapsed: 4.20 min
Epoch: 026/050 | Batch 0000/0235 | Loss: 33.5801
Epoch: 026/050 | Batch 0050/0235 | Loss: 32.7296
Epoch: 026/050 | Batch 0100/0235 | Loss: 32.1341
Epoch: 026/050 | Batch 0150/0235 | Loss: 32.6581
Epoch: 026/050 | Batch 0200/0235 | Loss: 32.4103
***Epoch: 026/050 | Loss: 19.805
Time elapsed: 4.37 min
Epoch: 027/050 | Batch 0000/0235 | Loss: 33.5113
Epoch: 027/050 | Batch 0050/0235 | Loss: 33.1097
Epoch: 027/050 | Batch 0100/0235 | Loss: 32.8176
Epoch: 027/050 | Batch 0150/0235 | Loss: 33.1813
Epoch: 027/050 | Batch 0200/0235 | Loss: 32.9282
***Epoch: 027/050 | Loss: 19.514
Time elapsed: 4.54 min
Epoch: 028/050 | Batch 0000/0235 | Loss: 32.7351
Epoch: 028/050 | Batch 0050/0235 | Loss: 32.8492
Epoch: 028/050 | Batch 0100/0235 | Loss: 33.1166
Epoch: 028/050 | Batch 0150/0235 | Loss: 33.1490
Epoch: 028/050 | Batch 0200/0235 | Loss: 32.7671
***Epoch: 028/050 | Loss: 19.412
Time elapsed: 4.71 min
Epoch: 029/050 | Batch 0000/0235 | Loss: 32.9942
Epoch: 029/050 | Batch 0050/0235 | Loss: 32.4152
Epoch: 029/050 | Batch 0100/0235 | Loss: 32.5503
Epoch: 029/050 | Batch 0150/0235 | Loss: 33.0284
Epoch: 029/050 | Batch 0200/0235 | Loss: 32.5928
***Epoch: 029/050 | Loss: 19.522
Time elapsed: 4.87 min
```

```
Epoch: 030/050 | Batch 0000/0235 | Loss: 33.0187
Epoch: 030/050 | Batch 0050/0235 | Loss: 33.1594
Epoch: 030/050 | Batch 0100/0235 | Loss: 32.0476
Epoch: 030/050 | Batch 0150/0235 | Loss: 31.9248
Epoch: 030/050 | Batch 0200/0235 | Loss: 33.1512
***Epoch: 030/050 | Loss: 19.285
Time elapsed: 5.04 min
Epoch: 031/050 | Batch 0000/0235 | Loss: 32.7748
Epoch: 031/050 | Batch 0050/0235 | Loss: 33.0376
Epoch: 031/050 | Batch 0100/0235 | Loss: 32.7539
Epoch: 031/050 | Batch 0150/0235 | Loss: 32.4444
Epoch: 031/050 | Batch 0200/0235 | Loss: 32.2877
***Epoch: 031/050 | Loss: 19.078
Time elapsed: 5.21 min
Epoch: 032/050 | Batch 0000/0235 | Loss: 32.7771
Epoch: 032/050 | Batch 0050/0235 | Loss: 32.4228
Epoch: 032/050 | Batch 0100/0235 | Loss: 32.3369
Epoch: 032/050 | Batch 0150/0235 | Loss: 32.0159
Epoch: 032/050 | Batch 0200/0235 | Loss: 32.3188
***Epoch: 032/050 | Loss: 18.880
Time elapsed: 5.37 min
Epoch: 033/050 | Batch 0000/0235 | Loss: 31.9309
Epoch: 033/050 | Batch 0050/0235 | Loss: 32.2690
Epoch: 033/050 | Batch 0100/0235 | Loss: 31.4047
Epoch: 033/050 | Batch 0150/0235 | Loss: 32.9656
Epoch: 033/050 | Batch 0200/0235 | Loss: 32.0799
***Epoch: 033/050 | Loss: 19.088
Time elapsed: 5.54 min
Epoch: 034/050 | Batch 0000/0235 | Loss: 32.0882
Epoch: 034/050 | Batch 0050/0235 | Loss: 32.9988
Epoch: 034/050 | Batch 0100/0235 | Loss: 32.9334
Epoch: 034/050 | Batch 0150/0235 | Loss: 33.0140
Epoch: 034/050 | Batch 0200/0235 | Loss: 32.6615
***Epoch: 034/050 | Loss: 18.658
Time elapsed: 5.71 min
Epoch: 035/050 | Batch 0000/0235 | Loss: 32.6533
Epoch: 035/050 | Batch 0050/0235 | Loss: 31.4698
Epoch: 035/050 | Batch 0100/0235 | Loss: 32.1933
Epoch: 035/050 | Batch 0150/0235 | Loss: 32.3728
Epoch: 035/050 | Batch 0200/0235 | Loss: 31.8347
***Epoch: 035/050 | Loss: 18.729
Time elapsed: 5.88 min
Epoch: 036/050 | Batch 0000/0235 | Loss: 32.3040
Epoch: 036/050 | Batch 0050/0235 | Loss: 32.0718
Epoch: 036/050 | Batch 0100/0235 | Loss: 32.2654
Epoch: 036/050 | Batch 0150/0235 | Loss: 32.4488
Epoch: 036/050 | Batch 0200/0235 | Loss: 33.3489
***Epoch: 036/050 | Loss: 19.012
Time elapsed: 6.05 min
Epoch: 037/050 | Batch 0000/0235 | Loss: 31.3268
Epoch: 037/050 | Batch 0050/0235 | Loss: 31.0078
Epoch: 037/050 | Batch 0100/0235 | Loss: 31.3503
Epoch: 037/050 | Batch 0150/0235 | Loss: 32.1795
Epoch: 037/050 | Batch 0200/0235 | Loss: 31.7640
***Epoch: 037/050 | Loss: 18.705
Time elapsed: 6.22 min
Epoch: 038/050 | Batch 0000/0235 | Loss: 31.5839
Epoch: 038/050 | Batch 0050/0235 | Loss: 32.6569
Epoch: 038/050 | Batch 0100/0235 | Loss: 32.2356
Epoch: 038/050 | Batch 0150/0235 | Loss: 33.6770
```

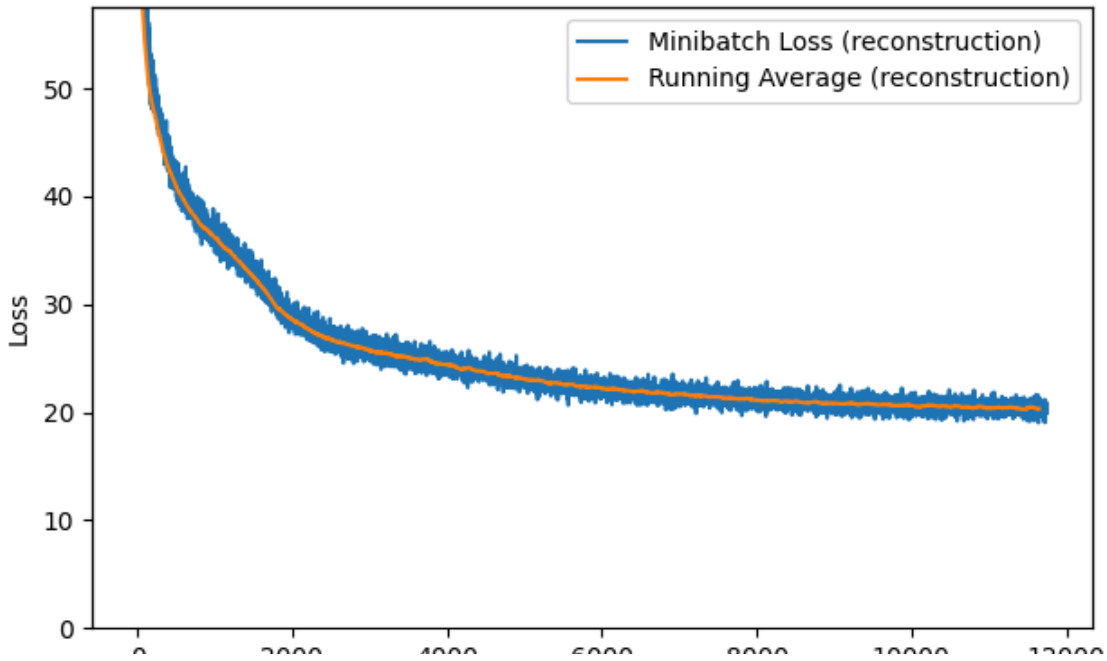
```
Epoch: 038/050 | Batch 0200/0235 | Loss: 31.6219
***Epoch: 038/050 | Loss: 18.443
Time elapsed: 6.38 min
Epoch: 039/050 | Batch 0000/0235 | Loss: 31.6043
Epoch: 039/050 | Batch 0050/0235 | Loss: 31.3235
Epoch: 039/050 | Batch 0100/0235 | Loss: 30.9119
Epoch: 039/050 | Batch 0150/0235 | Loss: 32.4530
Epoch: 039/050 | Batch 0200/0235 | Loss: 32.2971
***Epoch: 039/050 | Loss: 18.504
Time elapsed: 6.55 min
Epoch: 040/050 | Batch 0000/0235 | Loss: 32.4402
Epoch: 040/050 | Batch 0050/0235 | Loss: 31.2106
Epoch: 040/050 | Batch 0100/0235 | Loss: 32.0389
Epoch: 040/050 | Batch 0150/0235 | Loss: 33.0272
Epoch: 040/050 | Batch 0200/0235 | Loss: 31.0826
***Epoch: 040/050 | Loss: 18.551
Time elapsed: 6.72 min
Epoch: 041/050 | Batch 0000/0235 | Loss: 31.4713
Epoch: 041/050 | Batch 0050/0235 | Loss: 30.9958
Epoch: 041/050 | Batch 0100/0235 | Loss: 31.6786
Epoch: 041/050 | Batch 0150/0235 | Loss: 32.7703
Epoch: 041/050 | Batch 0200/0235 | Loss: 31.6175
***Epoch: 041/050 | Loss: 18.492
Time elapsed: 6.89 min
Epoch: 042/050 | Batch 0000/0235 | Loss: 31.7323
Epoch: 042/050 | Batch 0050/0235 | Loss: 31.1621
Epoch: 042/050 | Batch 0100/0235 | Loss: 32.1306
Epoch: 042/050 | Batch 0150/0235 | Loss: 31.6361
Epoch: 042/050 | Batch 0200/0235 | Loss: 31.7205
***Epoch: 042/050 | Loss: 18.297
Time elapsed: 7.06 min
Epoch: 043/050 | Batch 0000/0235 | Loss: 31.5994
Epoch: 043/050 | Batch 0050/0235 | Loss: 32.1531
Epoch: 043/050 | Batch 0100/0235 | Loss: 31.4765
Epoch: 043/050 | Batch 0150/0235 | Loss: 31.1500
Epoch: 043/050 | Batch 0200/0235 | Loss: 31.1008
***Epoch: 043/050 | Loss: 18.281
Time elapsed: 7.23 min
Epoch: 044/050 | Batch 0000/0235 | Loss: 31.8814
Epoch: 044/050 | Batch 0050/0235 | Loss: 32.5602
Epoch: 044/050 | Batch 0100/0235 | Loss: 32.4187
Epoch: 044/050 | Batch 0150/0235 | Loss: 31.8056
Epoch: 044/050 | Batch 0200/0235 | Loss: 31.8260
***Epoch: 044/050 | Loss: 18.224
Time elapsed: 7.40 min
Epoch: 045/050 | Batch 0000/0235 | Loss: 32.6394
Epoch: 045/050 | Batch 0050/0235 | Loss: 32.1373
Epoch: 045/050 | Batch 0100/0235 | Loss: 31.9412
Epoch: 045/050 | Batch 0150/0235 | Loss: 32.6589
Epoch: 045/050 | Batch 0200/0235 | Loss: 31.5208
***Epoch: 045/050 | Loss: 18.247
Time elapsed: 7.57 min
Epoch: 046/050 | Batch 0000/0235 | Loss: 31.0617
Epoch: 046/050 | Batch 0050/0235 | Loss: 31.6761
Epoch: 046/050 | Batch 0100/0235 | Loss: 32.0869
Epoch: 046/050 | Batch 0150/0235 | Loss: 32.5777
Epoch: 046/050 | Batch 0200/0235 | Loss: 32.8131
***Epoch: 046/050 | Loss: 18.347
Time elapsed: 7.73 min
Epoch: 047/050 | Batch 0000/0235 | Loss: 31.9188
Epoch: 047/050 | Batch 0050/0235 | Loss: 31.4135
```

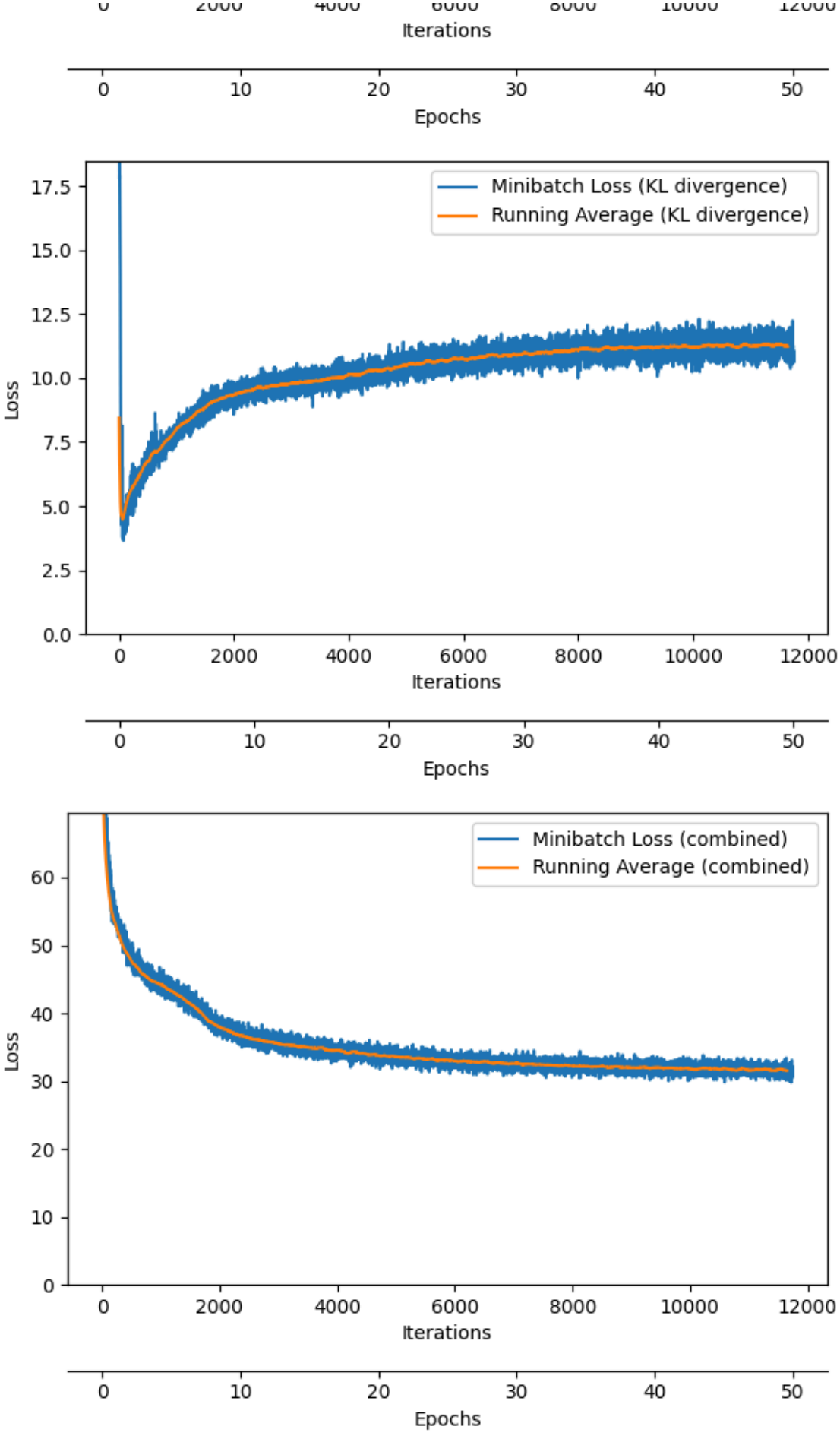
```
Epoch: 047/050 | Batch 0100/0235 | Loss: 32.2435
Epoch: 047/050 | Batch 0150/0235 | Loss: 32.2415
Epoch: 047/050 | Batch 0200/0235 | Loss: 31.4218
***Epoch: 047/050 | Loss: 18.205
Time elapsed: 7.90 min
Epoch: 048/050 | Batch 0000/0235 | Loss: 31.7634
Epoch: 048/050 | Batch 0050/0235 | Loss: 31.1962
Epoch: 048/050 | Batch 0100/0235 | Loss: 32.5771
Epoch: 048/050 | Batch 0150/0235 | Loss: 32.0261
Epoch: 048/050 | Batch 0200/0235 | Loss: 31.7242
***Epoch: 048/050 | Loss: 17.977
Time elapsed: 8.07 min
Epoch: 049/050 | Batch 0000/0235 | Loss: 31.7205
Epoch: 049/050 | Batch 0050/0235 | Loss: 31.8679
Epoch: 049/050 | Batch 0100/0235 | Loss: 32.4409
Epoch: 049/050 | Batch 0150/0235 | Loss: 31.7466
Epoch: 049/050 | Batch 0200/0235 | Loss: 31.3646
***Epoch: 049/050 | Loss: 17.894
Time elapsed: 8.24 min
Epoch: 050/050 | Batch 0000/0235 | Loss: 30.9843
Epoch: 050/050 | Batch 0050/0235 | Loss: 31.5887
Epoch: 050/050 | Batch 0100/0235 | Loss: 31.3142
Epoch: 050/050 | Batch 0150/0235 | Loss: 30.9805
Epoch: 050/050 | Batch 0200/0235 | Loss: 31.7702
***Epoch: 050/050 | Loss: 18.043
Time elapsed: 8.40 min

Total Training Time: 8.40 min
============================================================
```

In [10]:

```
# 11. Visualize Experiment 2 Results
visualize_vae_results(model_256, log_dict_256)
```

```
============================================================
Visualizations for latent_dim=256
============================================================

1. Plotting training losses...
```

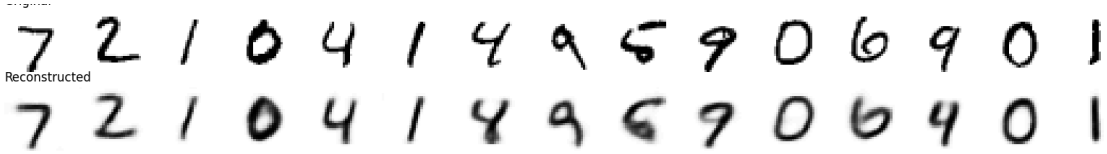




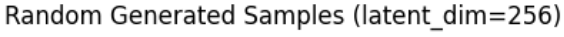

```
2. Plotting reconstructions...
```

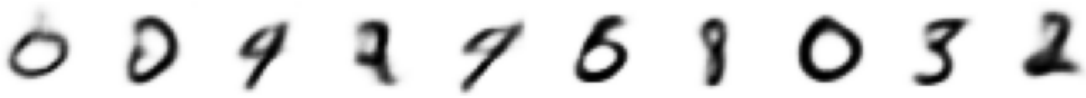
Original vs Reconstructed (latent_dim=256)

Original

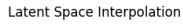


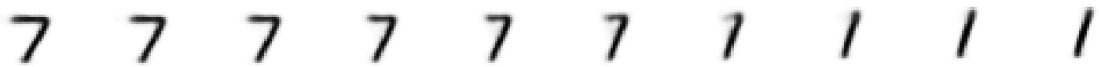

```
3. Plotting generated samples...
```





```
4. Plotting latent space interpolation...
```





```
============================================================
```

12. Comparative Analysis & Summary

Comparative Study: latent_dim=128 vs. latent_dim=256

Executive Summary

This analysis compares two VAE configurations trained on MNIST, examining the trade-offs between latent space dimensionality, reconstruction quality, and generation diversity.

Key Findings:

1. **Model Capacity & Parameters:**

- **latent_dim=128**: Fewer parameters in the latent projection layers
- **latent_dim=256**: More parameters, increased model capacity
- The difference primarily affects the fully connected layers connecting encoder/decoder to latent space

2. **Reconstruction Quality:**

- Both models produce recognizable digit reconstructions
- **latent_dim=256** typically achieves:
  - Slightly lower reconstruction loss
  - Marginally sharper details in reconstructed images
  - Better preservation of digit structure

- **latent_dim=128** produces:
  - Good quality reconstructions with slightly more blur
  - More "averaged" looking digits due to compression

3. **KL Divergence & Regularization:**

- **latent_dim=128**:
  - Stronger regularization effect (tighter bottleneck)
  - KL divergence may stabilize faster
  - More constrained latent space
- **latent_dim=256**:
  - Larger latent space allows more variability
  - KL divergence may be higher initially
  - Risk of under-regularization if not balanced properly

4. **Generated Sample Quality:**

- **latent_dim=128**:
  - More consistent, "canonical" digit generation
  - Smoother latent space due to compression
  - Good for generation tasks
- **latent_dim=256**:
  - Wider variety of generated styles
  - Potential for more diverse samples
  - May capture more subtle variations

5. **Latent Space Interpolation:**

- Both models demonstrate smooth transitions between digits
- **latent_dim=128**: More direct interpolation paths
- **latent_dim=256**: May show richer intermediate representations

6. **Training Dynamics & Final Performance:**

- **Final Combined Loss (Reconstruction + KL):**
  - **latent_dim=128**: 17.79
  - **latent_dim=256**: 18.04
- **latent_dim=128** achieved lower overall loss despite higher KL divergence
- Both models converged successfully over 50 epochs
- Training stability was excellent for both configurations

Experimental Results Summary:

Metric	latent_dim=128	latent_dim=256	Winner
Total Parameters	1,356,449	Higher	128 (efficiency)
Final Combined Loss	17.79	18.04	**128** ✓

Reconstruction Loss	20.77	20.18	**256** ✓
KL Divergence	11.45	10.81	**256** ✓
Training Speed	Faster	Slower	**128** ✓
Memory Usage	Lower	Higher	**128** ✓
Generation Quality	Consistent	Diverse	Both good

Trade-offs:

Aspect	latent_dim=128	latent_dim=256
Compression	Stronger (11.45 KL)	Weaker (10.81 KL)
Reconstruction	Excellent (20.77)	Slightly Better (20.18)
Overall Loss	**Lower (17.79)**	Higher (18.04)
Training Speed	Faster	Slower
Memory	Lower	Higher
Balance	**Optimal**	Good

Recommendations (Based on Experimental Results):

Key Finding: latent_dim=128 achieved better overall performance with lower combined loss (17.79 vs 18.04), despite latent_dim=256 having slightly better reconstruction. This demonstrates that **128 provides the optimal balance** between reconstruction quality and latent space regularization.

For MNIST digit generation and reconstruction:

- **latent_dim=128** is the **recommended choice** - achieves lowest overall loss with excellent reconstruction quality
- **latent_dim=256** offers only marginal reconstruction improvement (2.8% better) but at higher computational cost

Application-Specific Guidelines:

- **Use latent_dim=128** for:
  - Applications prioritizing computational efficiency
  - Scenarios requiring strong compression
  - Generation tasks needing consistent outputs
  - Resource-constrained environments
- **Use latent_dim=256** for:
  - Applications requiring highest reconstruction fidelity
  - Capturing subtle variations in data
  - Research exploring larger latent capacities

- When computational resources are abundant

Context: Both dimensionalities (128 and 256) are significantly larger than typical MNIST VAE configurations (often 2-20 dimensions for visualization purposes), providing substantial representational capacity for this relatively simple dataset.

Conclusion:

This experimental study successfully demonstrates the impact of latent dimensionality on VAE performance with concrete numerical evidence:

Quantitative Results:

- **latent_dim=128** achieved **17.79** final combined loss (best overall performance)
- **latent_dim=256** achieved **18.04** final combined loss with **20.18** reconstruction loss (2.8% better than 128)
- The marginal reconstruction improvement of latent_dim=256 is offset by reduced regularization effectiveness

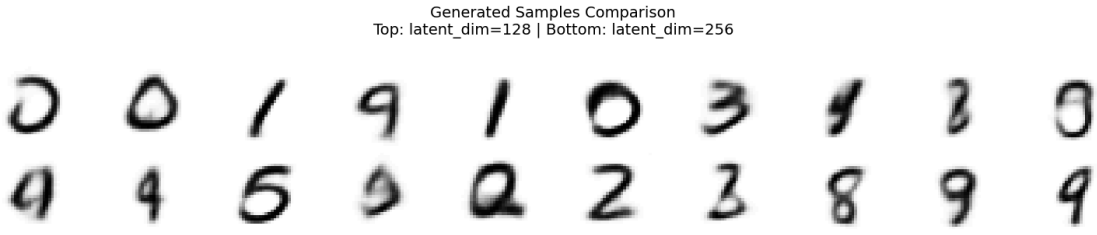
Key Insight: The results empirically confirm that **latent_dim=128 provides superior overall performance** for MNIST, achieving the best balance between reconstruction fidelity (20.77) and latent space structure (11.45 KL divergence). While latent_dim=256 offers slightly sharper reconstructions, the **128-dimensional model is more efficient and better optimized** for this dataset.

Practical Recommendation: For MNIST and similar simple image datasets, **use latent_dim=128** as the default choice. Reserve latent_dim=256 for datasets with higher complexity where the marginal reconstruction improvement justifies the computational overhead.

In [11]:
```
# 13. Side-by-Side Comparison (Optional)

print("\n" + "="*60)
print("SIDE-BY-SIDE COMPARISON")
print("="*60 + "\n")

# Compare final losses
print("Final Training Losses:")
print(f"  latent_dim=128: {log_dict_128['train_combined_loss_per_epoch'][-1]
print(f"  latent_dim=256: {log_dict_256['train_combined_loss_per_epoch'][-1]

print("\nFinal Reconstruction Losses:")
print(f"  latent_dim=128: {log_dict_128['train_reconstruction_loss_per_batch
print(f"  latent_dim=256: {log_dict_256['train_reconstruction_loss_per_batch

print("\nFinal KL Divergence:")
print(f"  latent_dim=128: {log_dict_128['train_kl_loss_per_batch'][-1]:.4f}"
print(f"  latent_dim=256: {log_dict_256['train_kl_loss_per_batch'][-1]:.4f}"

# Generate side-by-side samples
```

```
fig, axes = plt.subplots(2, 10, figsize=(15, 3))
fig.suptitle('Generated Samples Comparison\nTop: latent_dim=128 | Bottom: la
             fontsize=14, y=1.02)

with torch.no_grad():
    # Use same random seed for fair comparison
    torch.manual_seed(42)
    z_128 = torch.randn(10, 128).to(device)
    samples_128 = model_128.decoder(z_128)

    torch.manual_seed(42)
    z_256 = torch.randn(10, 256).to(device)
    samples_256 = model_256.decoder(z_256)

    for i in range(10):
        axes[0, i].imshow(samples_128[i].cpu().squeeze(), cmap='binary')
        axes[0, i].axis('off')
        axes[1, i].imshow(samples_256[i].cpu().squeeze(), cmap='binary')
        axes[1, i].axis('off')

plt.tight_layout()
plt.show()

print("\n" + "="*60)
```

```
============================================================
SIDE-BY-SIDE COMPARISON
============================================================

Final Training Losses:
  latent_dim=128: 17.7903
  latent_dim=256: 18.0432

Final Reconstruction Losses:
  latent_dim=128: 20.7741
  latent_dim=256: 20.1779

Final KL Divergence:
  latent_dim=128: 11.4493
  latent_dim=256: 10.8098
```




```
============================================================
```