Detailed Report: Implementation and Evaluation of a Variational Autoencoder (VAE) for Image Generation

1. Introduction

The goal of this project is to implement a Variational Autoencoder (VAE) for image generation and manipulation using PyTorch. VAEs are powerful generative models that learn a continuous latent representation of input data, enabling the generation of new image samples that share similar characteristics with the training images. In this project, the VAE is applied to images of size 150×150 pixels, and the implementation is structured in two main components:

- **vae.py:** Contains the VAE model architecture.
- **genpic.py:** Provides functionality to generate image variations from a trained VAE model.

This report details the steps taken in the implementation, explains the architecture and design choices, presents experimental results, and concludes with insights and potential future improvements.

2. Implementation Steps and Architecture

2.1. VAE Model Architecture (vae.py)

2.1.1. Encoder

- **Input Size**: 150×150 RGB images.
- **Architecture**: The encoder is built as a series of convolutional layers that progressively increase in channel depth. The channels evolve as follows: 32 → 64 → 128 → 256 → 512.
- **Activation and Normalization**: Each convolutional block is followed by BatchNorm2d and LeakyReLU activations, which help in stabilizing the learning process and enabling the network to capture complex patterns.
- **Latent Variable Computation**: After the convolutional layers, the feature maps are flattened and passed through linear layers to compute:
    - **Mean (μ)**
    - **Log variance (log_var)**

These are used in the reparameterization step.

2.1.2. Reparameterization Trick

To allow gradient flow through the sampling process, the reparameterization trick is employed:

python

CopyEdit

z = eps * std + mu

- **eps** is sampled from a standard normal distribution, N(0,1).
- **std** is computed as **exp(0.5 * log_var)**.

This method enables backpropagation through the stochastic sampling step by expressing the latent variable zzz as a deterministic function of μ\muμ, log_var\log_varlog_var, and the random variable ϵ\epsilonϵ.

2.1.3. Decoder

- **Input**: A latent vector of dimension 128.
- **Architecture**: The decoder reverses the encoder's architecture by using a linear layer to project the latent vector into a high-dimensional space, followed by a series of transposed convolution layers that upsample the image back to the original size.
- **Activation**: Intermediate activations are handled by LeakyReLU, while the final layer applies a sigmoid activation to constrain output pixel values between 0 and 1.

2.2. Image Generation Pipeline (genpic.py)

2.2.1. Loading and Preprocessing

- **Input Preparation**: Images are preprocessed using custom transforms (e.g., **celeb_transform**), which include resizing, center cropping, and conversion to tensor format.
- **Model Loading**: A trained VAE model is loaded from the checkpoint located at **./checkpoints_best/vae_model_20.pth**.

2.2.2. Generation Process

1. **Encoding**: The input image is encoded to extract the latent representation (μ and log_var).
2. **Sampling**: Several latent vectors are sampled using the reparameterization trick, with a variance scaling factor (typically 0.01) controlling the intensity of the variations.
3. **Decoding**: The sampled latent vectors are passed through the decoder to generate multiple image variations.
4. **Post-processing**: The generated images are processed with **celeb_transform1** (if needed) and then assembled into a grid.
5. **Output**: The final image grid is saved as **generated_images_grid2.jpg** in the project root directory.

2.3. Parameters and Hyperparameters

- **Image Size**: 150 pixels (height and width)
- **Latent Dimension**: 128
- **Hidden Dimensions in Encoder**: [32, 64, 128, 256, 512]
- **Number of Variations Generated**: 10
- **Variance Scaling Factor**: 0.01 (to control the spread in the latent space)

3. Experimental Results

3.1. Training and Checkpointing

- **Training Process**: The VAE was trained on a dataset of images with proper preprocessing applied. The training involved optimizing a loss function that combines reconstruction loss (to ensure generated images are similar to the input) and the Kullback-Leibler (KL) divergence (to regularize the latent space).
- **Checkpoint**: The best-performing model was saved as **./checkpoints_best/vae_model_20.pth** after 20 epochs.

3.2. Image Generation Outcomes

- **Quality of Generated Images**: By sampling multiple points from the latent space, the generator successfully produced 10 distinct image variations that preserved key characteristics of the input image.
- **Image Grid**: The final output, **generated_images_grid2.jpg**, displays the input alongside the generated variations in a well-organized grid format.
- **Observations**:
  - The reparameterization trick allowed smooth interpolation between images.
  - The variance scaling factor was critical in controlling the degree of variation; a small factor (0.01) ensured that the variations were not too drastic.

3.3. Experimental Analysis

- **Strengths**:
  - The model effectively captures the essential features of the input images.
  - The reconstruction quality is high, thanks to the balanced architecture and proper choice of activations.
- **Limitations**:
  - Fine-tuning the variance scaling factor remains crucial; too high a value may lead to unrealistic outputs.
  - Training on higher resolution images or larger datasets might require adjustments in the network architecture.