AI VIET NAM – AIO COURSE 2024

VAE Model

Duc-Huy Tran, Phuc-Thinh Nguyen, Dinh-Vinh Nguyen

I Giới thiệu bài toán

AI tạo sinh (Generative AI) đang trở thành một thuật ngữ phổ biến và xuất hiện rộng rãi trong nhiều lĩnh vực. Tuy nhiên, khái niệm này thường bị hiểu sai hoặc nhầm lẫn với các mô hình ngôn ngữ lớn, chẳng hạn như mô hình ngôn ngữ lớn ChatGPT. Trên thực tế, AI tạo sinh không chỉ dừng lại ở việc xử lý và phân tích dữ liệu có sẵn mà còn có khả năng tạo ra dữ liệu mới từ đầu, bao gồm văn bản, hình ảnh, âm thanh và nhiều loại dữ liệu khác. Một trong những phương pháp cốt lõi của AI tạo sinh là **Variational AutoEncoder (VAE)**, một mô hình đóng vai trò quan trọng trong việc học biểu diễn dữ liệu và sinh dữ liệu mới. VAE không chỉ giúp chúng ta hiểu sâu hơn về quy trình tạo sinh mà còn là nền tảng của nhiều mô hình hiện đại, bao gồm các hệ thống tạo ảnh tiên tiến như Stable Diffusion.

Hình 1: Minh họa ảnh được tạo bởi Diffusion Model.Link

Autoencoder (AE) là một mô hình học sâu được sử dụng để tái tạo dữ liệu, với mục tiêu nén thông tin đầu vào và sau đó phục hồi nó từ không gian ẩn. Mặc dù AE có khả năng khôi phục lại dữ liệu ban đầu một cách hiệu quả nhưng nó lại không thể tạo ra những dữ liệu hoàn toàn mới chưa từng xuất hiện trong bộ dữ liệu huấn luyện. Nguyên nhân chính của vấn đề này là do AE chỉ tập trung vào việc ánh xạ giữa lớp mã hóa (encoder) và không gian ẩn (latent space) mà không học được mối quan hệ ngữ nghĩa sâu sắc trong dữ liệu. Điều này dẫn đến tình trạng các điểm dữ liệu tương tự nhau bị ánh xạ đến các vị trí cách xa nhau, trong khi các điểm dữ liệu khác biệt có thể nằm gần nhau trong không gian ẩn. Nhược điểm này khiến AE không thể sử dụng trong các tác vụ AI tạo sinh, nơi việc sáng tạo dữ

liệu mới là yêu cầu quan trọng. Để khắc phục hạn chế này, Variational Autoencoder (VAE) ra đời như một giải pháp mạnh mẽ. VAE không chỉ đơn thuần tái tạo dữ liệu mà còn có khả năng sinh ra dữ liệu mới thông qua việc học phân phối xác suất trong không gian ẩn. Điều này giúp VAE tạo ra những hình ảnh, âm thanh hoặc dữ liệu chưa từng có, mở ra nhiều ứng dụng sáng tạo trong AI tạo sinh. Trong bài viết này, chúng ta sẽ cùng khám phá nguyên lý hoạt động của VAE, ứng dụng thực tế của nó và cách nó đã góp phần định hình sự phát triển của AI tạo sinh.

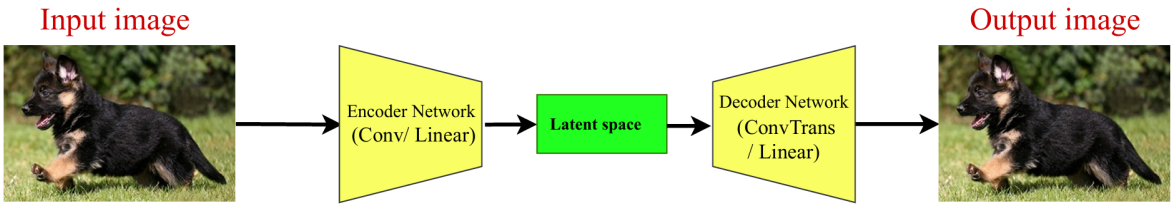
Bài viết được phân bổ theo bố cục sau:

1. Phần I: Giới thiệu bài toán.
2. Phần II: Kiến thức liên quan.
3. Phần III: Code triển khai.
4. Phần IV: Câu hỏi trắc nghiệm

II Kiến thức liên quan

1. Autoencoder

Autoencoder (AE) là một mô hình học sâu không giám sát, được thiết kế để học cách mã hóa và tái tạo lại dữ liệu đầu vào. Mô hình này có mục tiêu chính là giảm chiều dữ liệu và nén thông tin sao cho các đặc trưng quan trọng của dữ liệu vẫn được bảo tồn trong một không gian ẩn (latent space). AE hoạt động trên nguyên lý học biểu diễn của dữ liệu đầu vào và tái tạo lại dữ liệu từ biểu diễn này với sai số tối thiểu.



Hình 2: Kiến trúc của mô hình Autoencoder

Mô hình AE bao gồm ba phần chính:

- **Encoder**: Phần encoder có nhiệm vụ chuyển đổi dữ liệu đầu vào $x \in \mathbb{R}^n$ thành một biểu diễn ở không gian ẩn $z \in \mathbb{R}^q$, với $q \ll n$. Quá trình này thực hiện việc nén dữ liệu, giảm chiều của dữ liệu gốc nhằm giữ lại những đặc trưng quan trọng nhất. Việc chuyển đổi này giúp mô hình tập trung vào các đặc trưng quan trọng của dữ liệu, loại bỏ những thông tin không cần thiết và tạo ra một biểu diễn gọn gàng hơn.
- **Bottleneck (Latent Space):** Không gian ẩn là một không gian có kích thước nhỏ hơn so với đầu vào của mô hình, nơi mô hình lưu trữ các đặc trưng cốt lõi của dữ liệu. Không gian này đóng vai trò quan trọng trong việc tạo ra một biểu diễn gọn gàng và hiệu quả của dữ liệu. Khi kích thước của không gian ẩn càng nhỏ, thông tin được lưu trữ trong latent space càng trở nên quan trọng và giúp tránh được vấn đề overfitting khi model phải chọn lọc các thông tin quan trọng để lưu trữ.
- **Decoder**: Phần decoder nhận đầu vào là các đặc trưng từ không gian ẩn và tái tạo lại dữ liệu đầu vào ban đầu. Mục tiêu của decoder là khôi phục lại dữ liệu gốc một cách chính xác nhất có

thể, với sai số nhỏ nhất so với dữ liệu ban đầu. Quá trình này đóng vai trò quan trọng trong việc tái tạo thông tin từ biểu diễn ở không gian ẩn, đồng thời giúp mô hình hiểu rõ hơn về các đặc trưng đã được mã hóa trong quá trình encoder.

Hàm mất mát (Loss function) trong AE được thiết kế để đo sai số giữa dữ liệu gốc và dữ liệu tái tạo, đóng vai trò là mục tiêu tối ưu hóa của mô hình. Mục đích chính của AE là học cách mã hóa và giải mã dữ liệu sao cho dữ liệu được tái tạo giống với dữ liệu gốc nhất có thể. Để đạt được mục tiêu này, AE sử dụng *Reconstruction loss* để hướng dẫn mô hình tối ưu hóa việc tái tạo dữ liệu. Hai hàm mất mát phổ biến nhất trong AE là Mean Squared Error (MSE) và Binary Cross-Entropy (BCE).

- **Mean Squared Error (MSE)**: Được sử dụng khi dữ liệu đầu vào là các giá trị liên tục, ví dụ như grayscale hay ảnh RGB có pixel là các giá trị thực. Công thức của MSE được định nghĩa như sau:

$$L_{MSE} = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{N}\sum_{j=1}^{N}(x_{j,i} - \hat{x}_{j,i})^2 \tag{1}$$

  Trong đó:

  - M là số mẫu trong batch.
  - N là số chiều của dữ liệu.
  - $x_{j,i}$ là giá trị thực tại pixel j của mẫu i.
  - $\hat{x}_{j,i}$ là giá trị tái tạo tương ứng.

- **Binary Cross-Entropy (BCE)**: Được sử dụng khi dữ liệu đầu vào là các giá trị nhị phân hoặc đã chuẩn hóa về khoảng [0, 1], ví dụ như ảnh đen trắng. Công thức của BCE như sau:

$$L_{BCE} = -\frac{1}{M}\sum_{i=1}^{M}\frac{1}{N}\sum_{j=1}^{N}[x_{j,i}\log\hat{x}_{j,i} + (1 - x_{j,i})\log(1 - \hat{x}_{j,i})] \tag{2}$$

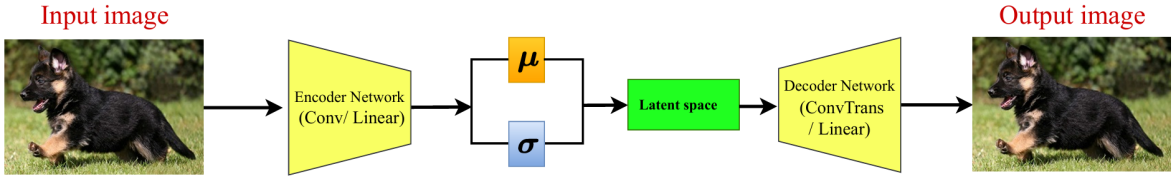
  Trong đó:

  - $x_{j,i}$ là giá trị tại pixel j của mẫu i.
  - $\hat{x}_{j,i}$ là giá trị tái tạo pixel của chính mẫu đó.

AE được ứng dụng phổ biến trong nén dữ liệu, khử nhiễu, và giảm chiều dữ liệu, nơi dữ liệu đầu vào (ví dụ: ảnh, văn bản) được mã hóa thành biểu diễn nén (latent space) để giảm kích thước mà vẫn giữ được thông tin quan trọng trước khi truyền đi. Sau đó bộ giải mã (decoder) ở nơi nhận khôi phục dữ liệu gần giống với bản gốc, giúp tiết kiệm bộ nhớ và tăng tốc độ xử lý. Tuy có nhiều lợi ích nhưng AE có một hạn chế lớn đó là việc không gian tiềm ẩn (latent space) không có cấu trúc xác suất, nghĩa là mô hình chỉ tối ưu hóa reconstruction loss mà không đảm bảo rằng không gian ẩn có tính liên tục và đầy đủ. Điều này có thể dẫn đến việc các điểm dữ liệu tương tự có thể bị ánh xạ đến các vị trí xa nhau trong không gian ẩn. Thêm vào đó, điểm yếu này khiến AE không thể sinh ra dữ liệu mới một cách hợp lý, vì mô hình chỉ học cách sao chép dữ liệu đầu vào thay vì học một phân phối tổng quát của dữ liệu. Để khắc phục vấn đế này, Variational Autoencoder (VAE) đã được đề xuất. VAE không chỉ có khả năng tái tạo dữ liệu mà còn mô hình hóa không gian tiềm ẩn như một phân phối xác suất liên tục. Điều này được thực hiện bằng cách thêm thành phần *KL-divergence loss* nhằm đảm bảo phân phối ẩn của dữ liệu được điều chỉnh theo một phân phối chuẩn (thường là phân phối Gaussian). Nhờ đó, VAE có thể sinh ra dữ liệu mới bằng cách lấy mẫu từ không gian tiềm ẩn đã học, mở ra nhiều ứng dụng trong AI tạo sinh như tạo ảnh, tổng hợp giọng nói hay mô phỏng dữ liệu.

2. Variational Autoencoder

Variational Autoencoder (VAE) là một mô hình học sâu sinh (generative deep learning model) được thiết kế để học các biểu diễn tiềm ẩn (latent representations) của dữ liệu một cách không giám sát. Không giống như các phương pháp giảm chiều đơn thuần, VAE học một không gian tiềm ẩn có cấu trúc và có thể lấy mẫu để tạo ra dữ liệu mới tương tự như dữ liệu huấn luyện. VAE kết hợp các kỹ thuật từ suy luận biến phân (variational inference) và mạng nơ-ron sâu (deep neural networks) để để tối ưu hóa không gian ẩn. Thay vì chỉ ánh xạ dữ liệu đầu vào thành một vector cố định, VAE ánh xạ dữ liệu vào một phân phối xác suất, cho phép sinh ra các dữ liệu mới từ không gian tiềm ẩn theo một cách có kiểm soát.



Hình 3: Kiến trúc của mô hình VAE

VAE có cấu trúc tương tự như AE, nhưng có một số khác biệt quan trọng trong cách encoder hoạt động và cấu trúc của không gian ẩn:

- **Encoder (Parametric Encoder)**: Thay vì tạo ra một vector mã hóa đơn lẻ như AE, encoder của VAE tạo ra hai vector: vector trung bình (mean-μ) và vector độ lệch chuẩn (deviation-σ). Mô hình sau đó sử dụng hai vector này để xác định phân phối Gaussian.
- **Latent space (Probabilistic Latent Space)**: VAE không có một "bottleneck"cố định như AE. Thay vào đó, mô hình ép buộc các biểu diễn trong không gian ẩn phải tuân theo một phân phối xác suất cụ thể. Điều này có tác dụng điều chuẩn (regularization), tạo cấu trúc và tính liên tục cho không gian ẩn.
- **Reparameterization Trick**: Để huấn luyện mô hình VAE bằng gradient descent, cần phải lấy mẫu từ phân phối xác suất được xác định bởi encoder. Tuy nhiên, thao tác lấy mẫu là không khả vi (non-differentiable). Để giải quyết vấn đề này, VAE sử dụng một kỹ thuật gọi là "reparameterization trick":

  $$z = \mu + \sigma \odot \epsilon \tag{3}$$

  Trong đó:

  - μ là vector trung bình.
  - σ là vector độ lệch chuẩn.
  - ϵ là biến ngẫu nhiên được lấy mẫu từ một phân phối cố định.

  Kỹ thuật này cho phép biểu diễn biến ngẫu nhiên z như một hàm khả vi của các tham số encoder và một biến ngẫu nhiên độc lập, sau z đó được truyền cho decoder.
- **Decoder**: Decoder nhận một mẫu từ không gian ẩn và tái tạo lại dữ liệu đầu vào ban đầu, tương tự như trong AE.

Hàm mất mát (Loss function) của VAE bao gồm hai thành phần chính: Reconstruction Loss và Regularization Loss. VAE khác biệt với AE với việc có thêm thành phần Regularization Loss để đảm bảo không gian tiềm ẩn có cấu trúc và liên tục. Điều này cho phép VAE tạo ra các mẫu dữ liệu mới từ

không gian tiềm ẩn một cách hiệu quả. Công thức tổng thể của hàm loss được gọi là Evidence Lower Bound (ELBO:

$$L(\theta, \phi) = \mathbb{E}_{q_\phi(z|x)} \left[\log p_\theta(x \mid z)\right] - \mathrm{KL}\left(q_\phi(z \mid x) \parallel p(z)\right) \tag{4}$$

Trong đó:

- ϕ: các tham số của encoder.
- θ: các tham số của decoder.
- $\mathbb{E}q\phi(z \mid x) \left[\log p_\theta(x \mid z)\right]$: là Reconstruction Loss của VAE, đo lường khả năng tái tạo dữ liệu x từ các mẫu z được lấy từ phân phối $q_\phi(z \mid x)$. Khác với Reconstruction Loss của AE, VAE tính giá trị trung bình của log-likelihood $\log p_\theta(x \mid z)$ trên phân phối $q_\phi(z \mid x)$. Điều này cho phép tích hợp tính bất định của z trong quá trình tái tạo, từ đó mô hình không chỉ cố gắng khôi phục lại x mà còn phản ánh được xác suất sinh ra x theo nhiều khả năng khác nhau của z.
- $\mathrm{KL}\left(\boldsymbol{q_\phi(z \mid x) \parallel p(z)}\right)$: là Regularization Loss, đo lường sự khác biệt giữa phân phối hậu nghiệm (posterior) gần đúng $q_\phi(z|x)$ (do encoder ước tính) và phân phối tiên nghiệm (prior) $p(z)$ (thường sử dụng Gaussian chuẩn). Regularization Loss đảm bảo rằng không gian tiềm ẩn có cấu trúc liên tục và có tính tổng quát cao, giúp VAE sinh ra các mẫu dữ liệu mới từ không gian này một cách hiệu quả. Đây là điểm khác biệt chính so với AE, vì AE không có thành phần điều chuẩn này, dẫn đến không gian tiềm ẩn của AE thường thiếu cấu trúc rõ ràng.

VAE được sử dụng rộng rãi trong nhiều lĩnh vực nhờ khả năng học biểu diễn tiềm ẩn và tạo dữ liệu mới. Một trong những ứng dụng phổ biến nhất của VAE là tạo dữ liệu tổng hợp, chẳng hạn như hình ảnh, video, hoặc văn bản, dựa trên các mẫu đã học từ dữ liệu huấn luyện. Điều này được ứng dụng trong việc tạo hình ảnh phong cách nghệ thuật, nội dung video mới hoặc tổng hợp văn bản tự nhiên trong các chatbot. Ngoài ra, VAE còn được sử dụng để phát hiện bất thường trong dữ liệu, ví dụ như phát hiện gian lận tài chính hoặc lỗi hệ thống công nghiệp.

Mặc dù mạnh mẽ nhưng VAE vẫn có một số hạn chế đáng chú ý. Một trong những điểm yếu lớn nhất là chất lượng tái tạo thấp khi hình ảnh được tạo ra thường bị mờ. Điều này xuất phát từ sự đánh đổi giữa hai thành phần chính trong hàm mất mát: *Reconstruction Loss* và *KL Divergence*. Hai thành phần này thường trái ngược nhau, dẫn đến sự đánh đổi (trade-off) khiến kết quả tái tạo không sắc nét như các mô hình khác như GAN. Chính điều này đã khiến GAN trở thành lựa chọn phổ biến hơn trong nhiều ứng dụng tạo hình ảnh. Ngoài ra, VAE thiếu cơ chế áp đặt ràng buộc lên dữ liệu khiến cho không thể kiểm soát được chính xác được dữ liệu cụ thể được tạo ra. Một số biến thể như Conditional VAE (CVAE), Beta-VAE (B-VAE), và Vector Quantized VAE (VQ-VAE) đã được phát triển để khắc phục vấn đề bằng cách cải thiện chất lượng hình ảnh và kiểm soát tốt hơn không gian tiềm ẩn.

III Code triển khai

1. Import các thư viện cần thiết

Phần này nhập các thư viện quan trọng để xây dựng mô hình deep learning.

- Thư viện hỗ trợ tính toán tensor và xây dựng mô hình học sâu.
- Thư viện xử lý mảng số học hiệu quả.
- Các module xây dựng mạng nơ-ron nhân tạo.
- Các thuật toán tối ưu hóa mô hình.
- Công cụ vẽ đồ thị và trực quan hóa dữ liệu.
- Hỗ trợ hiển thị thanh tiến trình khi huấn luyện.
- Công cụ hiển thị kiến trúc mô hình.
- Tiền xử lý ảnh và tải tập dữ liệu phổ biến.
- Công cụ tải dữ liệu theo batch và chia tập dữ liệu.

```
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import torch.nn.functional as F

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

2. Xác định thiết bị tính toán

Phần này kiểm tra xem GPU có khả dụng không và đặt thiết bị tính toán phù hợp.

- Nếu GPU khả dụng, sử dụng để tăng tốc tính toán.
- Nếu không có GPU, sử dụng CPU.
- Hiển thị thiết bị đang được sử dụng.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

3. Khai báo siêu tham số và tải dữ liệu

Phần này thiết lập các siêu tham số và chuẩn bị dữ liệu để huấn luyện mô hình.

- Định nghĩa các siêu tham số như kích thước batch, kích thước ảnh, số kênh màu, kích thước không gian tiềm ẩn và số epoch huấn luyện.
- Áp dụng phép biến đổi dữ liệu, bao gồm thay đổi kích thước ảnh và chuyển đổi thành tensor.
- Tải bộ dữ liệu MNIST, một tập dữ liệu chữ số viết tay, từ thư viện và áp dụng các phép biến đổi cần thiết.

```
# Hyperparameters
batch_size = 256
img_size = 28  # original image size is 28x28
channels = 1  # grayscale image
latent_dim = 2
num_epochs = 500  # training epochs

transform = transforms.Compose([
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
])

dataset = datasets.MNIST(
    root="./data",      # Data storage directory
    train=True,
    transform=transform,
    download=True,
)
```

4. Chia dữ liệu và tạo DataLoader

Phần này chia tập dữ liệu thành tập huấn luyện và tập kiểm tra, sau đó tạo bộ nạp dữ liệu để huấn luyện mô hình.

- Chia dữ liệu thành hai phần: 80% cho huấn luyện và 20% cho kiểm tra.
- Sử dụng phương thức chia ngẫu nhiên để đảm bảo mẫu huấn luyện và kiểm tra không trùng nhau.
- Tạo DataLoader để tải dữ liệu theo từng batch, hỗ trợ tăng tốc bằng cách sử dụng nhiều luồng xử lý.

```
# Split dataset into training (80%) and validation (20%)
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
print(f"Training samples: {len(train_dataset)}, Validation samples: {len(val_dataset)}")

# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=2)
```

5. Định nghĩa mô hình VAE

Phần này xây dựng mô hình VAE (Variational Autoencoder) để mã hóa và giải mã ảnh MNIST.

- Bộ mã hóa (Encoder): Biến đổi ảnh đầu vào thành một không gian tiềm ẩn có số chiều nhỏ hơn.
- Lấy mẫu (Reparameterization): Tạo biến ngẫu nhiên từ không gian tiềm ẩn theo phân phối chuẩn.
- Bộ giải mã (Decoder): Chuyển không gian tiềm ẩn trở lại ảnh gốc bằng cách sử dụng các lớp chuyển vị tích chập.
- Hàm truyền qua (Forward): Điều phối luồng dữ liệu qua từng thành phần của mô hình.

```python
# Define the Variational Autoencoder (VAE) in PyTorch
class VAE(nn.Module):
    def __init__(self, channels, latent_dim):
        super(VAE, self).__init__()
        # Encoder
        self.conv1 = nn.Conv2d(channels, 16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, stride=2, padding=1)

        self.flatten_dim = 64 * 7 * 7
        self.fc_mu = nn.Linear(self.flatten_dim, latent_dim)
        self.fc_logvar = nn.Linear(self.flatten_dim, latent_dim)

        # Decoder
        self.fc_decode = nn.Linear(latent_dim, self.flatten_dim)
        self.deconv1 = nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, output_padding=1)
        self.deconv2 = nn.ConvTranspose2d(32, 16, kernel_size=3, stride=2, padding=1, output_padding=1)
        self.conv_final = nn.Conv2d(16, channels, kernel_size=3, padding=1)

    def encode(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = x.view(-1, self.flatten_dim)
        mu = self.fc_mu(x)
        logvar = self.fc_logvar(x)
        return mu, logvar

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

    def decode(self, z):
        x = F.relu(self.fc_decode(z))
        x = x.view(-1, 64, 7, 7)
        x = F.relu(self.deconv1(x))
        x = F.relu(self.deconv2(x))
        x = torch.sigmoid(self.conv_final(x))
        return x

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
```

```
        recon_x = self.decode(z)
        return recon_x, mu, logvar
```

6. Hàm mất mát của VAE

Hàm mất mát trong VAE bao gồm hai thành phần:

- Mất mát tái tạo (Reconstruction Loss): Đo lường sự khác biệt giữa ảnh đầu vào và ảnh được tái tạo bằng lỗi trung bình bình phương (MSE).
- Độ đo KL (KL Divergence): Điều chỉnh phân phối tiềm ẩn về phân phối chuẩn, giúp mô hình sinh ra dữ liệu hợp lý hơn.
- Tổng mất mát: Kết hợp hai thành phần trên, trong đó mất mát tái tạo được nhân với hệ số B để cân bằng tác động.

```python
def loss_function(recon_x, x, mu, log_var, B=1000):
    # Flatten tensors for MSE calculation
    recon_x_flat = recon_x.view(recon_x.size(0), -1)
    x_flat = x.view(x.size(0), -1)

    # Calculate MSE (per element average)
    mse_out = F.mse_loss(recon_x_flat, x_flat, reduction='mean')

    # Scale by input dimensions
    reconstruction_loss = mse_out * x.shape[1] * x.shape[2] * x.shape[3]

    # Calculate KL divergence
    kl_loss = -0.5 * torch.sum(1 + log_var - mu.pow(2) - log_var.exp(), axis=1)

    # Compute final loss (adding B*recon_loss + kl_loss)
    total_loss = B * reconstruction_loss + torch.mean(kl_loss)

    return total_loss, reconstruction_loss, torch.mean(kl_loss)
```

7. Khởi tạo mô hình và bộ tối ưu

Trước khi huấn luyện, cần khởi tạo các thành phần chính:

- Mô hình VAE: Định nghĩa mạng autoencoder biến thể với số kênh đầu vào và kích thước không gian tiềm ẩn.
- Thiết bị tính toán: Đưa mô hình lên GPU (nếu có) để tăng tốc xử lý.
- Bộ tối ưu hóa: Sử dụng Adam với tốc độ học $1e-3$ và trọng số phân rã $1e-5$ để cập nhật tham số mô hình một cách hiệu quả.

```python
# Create model, optimizer, etc.
model = VAE(channels, latent_dim).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-5)
```

8. Huấn luyện mô hình và ghi log

Trong phần này, mô hình sẽ được huấn luyện qua nhiều epoch, và các thông tin quan trọng sẽ được ghi lại vào tệp log để theo dõi quá trình.

- Tạo thư mục lưu log: Nếu thư mục chưa tồn tại, nó sẽ được tạo tự động.
- Tạo tệp log: Đặt tên tệp theo thời gian hiện tại để dễ quản lý.
- Vòng lặp huấn luyện:
  - Tính toán lỗi tổng, lỗi tái tạo và lỗi KL.
  - Cập nhật tham số mô hình bằng thuật toán tối ưu.
  - Hiển thị tiến trình huấn luyện theo từng epoch.
  - Ghi lại các giá trị lỗi vào tệp log sau mỗi epoch.

```python
import os
from datetime import datetime

# Create a log directory if it doesn't exist
log_dir = "logs"
os.makedirs(log_dir, exist_ok=True)

# Create a log file with timestamp
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
log_file = os.path.join(log_dir, f"training_log_{timestamp}.txt")

# Open the log file
with open(log_file, "w") as f:
    f.write(f"Training started at {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
    f.write(f"Model: VAE with latent_dim={latent_dim}\n")
    f.write(f"Batch size: {batch_size}, Image size: {img_size}x{img_size}\n")
    f.write(f"Total epochs: {num_epochs}\n\n")
    f.write("Epoch,Avg_Loss,Recon_Loss,KL_Loss\n")

# Training loop with logging
model.train()
for epoch in range(num_epochs):
    train_loss = 0.0
    epoch_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}", leave=False)
    for data, _ in epoch_bar:
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data)
        loss, recon_loss, kl_loss = loss_function(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
        epoch_bar.set_postfix(loss=loss.item())

    # Calculate average loss
    avg_loss = train_loss / len(train_loader.dataset)

    # Print epoch summary
    print(f"Epoch {epoch+1}/{num_epochs} Loss per sample: {avg_loss:.4f}  "
          f"Recon Loss: {recon_loss.item():.4f}  KL Loss: {kl_loss.item():.4f}")

    # Save to log file
```

```
    with open(log_file, "a") as f:
        f.write(f"{epoch+1},{avg_loss:.6f},{recon_loss.item():.6f},{kl_loss.item():.6f}\n")

# Log training completion
with open(log_file, "a") as f:
    f.write(f"\nTraining completed at {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")

print(f"Training log saved to {log_file}")
```

9. Kiểm tra và trực quan hóa ảnh tái tạo

Sau khi huấn luyện, ta sẽ kiểm tra chất lượng mô hình bằng cách tái tạo ảnh từ tập validation.

- Chuyển mô hình sang chế độ đánh giá để đảm bảo không có cập nhật tham số.
- Chọn một số ảnh từ tập validation và đưa vào mô hình để tạo ảnh tái tạo.
- Lưu ý: Không tính toán gradient trong bước này để tiết kiệm bộ nhớ.

```
# After training, visualize the reconstruction on validation images
model.eval()
with torch.no_grad():
    data_iter = iter(val_loader)
    images, _ = next(data_iter)
    images = images.to(device)
    recon_images, _, _ = model(images)
```

10. Trực quan hóa ảnh gốc và ảnh tái tạo

Sau khi mô hình được huấn luyện, ta sẽ trực quan hóa ảnh gốc và ảnh tái tạo để đánh giá chất lượng tái tạo của mô hình.

- Chọn 10 ảnh từ tập validation để hiển thị.
- Hiển thị ảnh gốc và ảnh tái tạo theo từng cặp.
- Dùng Matplotlib để vẽ ảnh với kích thước phù hợp.

```
# Plot original and reconstructed images side by side
n = 10  # number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original image: (C,H,W) -> (H,W,C)
    orig = images[i].cpu().permute(1, 2, 0).numpy()
    recon = recon_images[i].cpu().permute(1, 2, 0).numpy()

    ax = plt.subplot(2, n, i + 1)
    plt.imshow(orig)
    plt.title("Original")
    plt.axis('off')

    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(recon)
    plt.title("Reconstructed")
    plt.axis('off')

plt.show()
```

IV Câu hỏi trắc nghiệm

Câu 1. Mục tiêu chính của Autoencoder (AE) là gì?

A. Phân loại hình ảnh.
B. Giảm nhiễu dữ liệu bằng cách học đặc trưng quan trọng.
C. Nén dữ liệu vào một không gian tiềm ẩn và tái tạo lại dữ liệu đầu vào.
D. Phát hiện các đối tượng trong ảnh.

Câu 2. Trong Autoencoder, mạng encoder có nhiệm vụ gì?

A. Tái tạo dữ liệu đầu vào từ véc-tơ tiềm ẩn.
B. Mã hóa dữ liệu đầu vào thành một không gian tiềm ẩn có kích thước nhỏ hơn.
C. So sánh dữ liệu đầu vào và đầu ra để tính toán sai số.
D. Lấy mẫu từ không gian tiềm ẩn để sinh dữ liệu mới.

Câu 3. So với Principal Component Analysis (PCA), Autoencoder có ưu điểm nào?

A. Autoencoder có thể học được các biểu diễn phi tuyến tính trong dữ liệu.
B. Autoencoder luôn hoạt động chính xác hơn PCA.
C. Autoencoder không cần huấn luyện mô hình trước khi sử dụng.
D. Autoencoder chỉ có thể áp dụng cho dữ liệu hình ảnh.

Câu 4. Trong VAE, thành phần nào giúp mô hình sinh dữ liệu mới thay vì chỉ sao chép đầu vào?

A. Hàm mất mát MSE (Mean Squared Error).
B. Cơ chế lấy mẫu lại (Reparameterization Trick).
C. Lớp convolutional đầu vào.
D. Hàm kích hoạt ReLU.

Câu 5. Trong VAE, hàm mất mát bao gồm những thành phần nào?

A. MSE Loss và Entropy Loss.
B. Reconstruction Loss và KL Divergence Loss.
C. KL Divergence Loss và Cross-Entropy Loss.
D. Cross-Entropy Loss và L1 Regularization.

Câu 6. Trong mạng encoder của VAE, đầu ra gồm những gì?

A. Một véc-tơ duy nhất biểu diễn ảnh đầu vào.
B. Hai véc-tơ: trung bình (mu) và phương sai (logvar).
C. Một ma trận trọng số và một véc-tơ độ lệch.
D. Một ảnh tái tạo có cùng kích thước với đầu vào.

Câu 7. Reparameterization Trick trong VAE được sử dụng để làm gì?

A. Giúp lan truyền gradient qua bước lấy mẫu ngẫu nhiên.
B. Giảm kích thước của mạng nơ-ron.

C. Tăng độ chính xác của mô hình bằng cách thêm tham số.

D. Biến dữ liệu thành ảnh có kích thước nhỏ hơn.

Câu 8. KL Divergence trong hàm mất mát của VAE có vai trò gì?

A. Điều chỉnh ảnh đầu ra sao cho gần với ảnh đầu vào.

B. Cân bằng giữa học đặc trưng và nén dữ liệu.

C. Tối ưu hóa mạng decoder.

D. Giảm nhiễu trong dữ liệu đầu vào.

Câu 9. So với Autoencoder (AE) thông thường, Variational Autoencoder (VAE) có điểm khác biệt nào chính?

A. VAE sử dụng mạng GAN để sinh dữ liệu.

B. VAE mã hóa đầu vào thành phân phối xác suất thay vì một véc-tơ cụ thể.

C. VAE sử dụng kiến trúc mạng CNN thay vì MLP.

D. VAE không thể tái tạo ảnh từ đầu vào.

Câu 10. Nếu mô hình VAE được huấn luyện với giá trị KL Loss quá cao, điều gì có thể xảy ra?

A. Mô hình có thể học tốt hơn và giảm sai số tái tạo.

B. Mô hình sẽ sinh ra các ảnh đa dạng hơn.

C. Mô hình có thể bị "underfitting"do véc-tơ tiềm ẩn bị bó hẹp quá mức.

D. Mô hình sẽ không bị ảnh hưởng.