第一次作业　测试 VAE 拓展模型生成效果

生成式人工智能

吴天阳 *4124136039* 人工智能 *B2480*

1　应用实践

使用 CVAE 模型在 FashionMNIST 数据集上进行训练，以下是各模块代码

1.1　模型定义

本实验使用 MLP 与卷积分别搭建两个模型．MLP 模型以展平后的 784 维 MNIST 图像向量与 10 维 one-hot 标签向量直接拼接作为输入，编码器通过 512 维隐层进行特征融合后，并行输出潜在空间的均值与对数方差参数．解码阶段将 64 维潜在向量与条件标签拼接，经相同维度的全连接层映射后，通过 Sigmoid 激活重构 28×28 像素图像．该架构通过向量级拼接实现条件信息的显式融合，编码阶段采用单隐层结构简化特征提取流程，解码过程通过全连接网络的全局感知特性实现图像整体结构的生成．

卷积架构 CVAE 模型输入为 28×28 灰度图像与 10 维 one-hot 标签的融合信息：编码阶段将标签张量沿空间维度扩展至与图像同尺寸后拼接于通道维度，形成 11 通道的复合输入，经过两次步长为 2 的卷积下采样（$32 \to 64$ 通道）获得 7×7 特征图，最终通过全连接层输出潜在空间的均值与对数方差．解码阶段将潜在向量与条件标签拼接后，经全连接层恢复至 $64 \times 7 \times 7$ 特征维度，通过两层转置卷积完成上采样重构，输出层采用 Sigmoid 激活确保像素值在 $[0, 1]$ 区间．

```python
import torch
import torch.nn as nn
import torch.nn.functional as F


# Variational Autoencoder
class CVAE(nn.Module):

    def __init__(self, latent_size=64):
        super(CVAE, self).__init__()
        self.labels = 10  # Number of labels
        self.latent_size = latent_size  # Dimension of latent space
        # Encoder layers: input is flattened image and label
        self.fc1 = nn.Linear(28 * 28 + self.labels, 512)  # Encoder input layer
        self.fc2 = nn.Linear(512, latent_size)              # Mean of latent space
        self.fc3 = nn.Linear(512, latent_size)              # Log variance of latent space

        # Decoder layers: input is concatenation of latent variable and label
        self.fc4 = nn.Linear(latent_size + self.labels, 512)  # Decoder input layer
```

```
        self.fc5 = nn.Linear(512, 28 * 28)                    # Decoder output layer

    # Encoder part, input and output are consistent with ConvCVAE
    def encode(self, x, y):
        # x: (batch, 1, 28, 28), y: (batch, 10)
        x_flat = x.view(x.size(0), -1)  # Flatten the image
        x_concat = torch.cat([x_flat, y], dim=1)
        hidden = F.relu(self.fc1(x_concat))
        mu = self.fc2(hidden)
        log_var = self.fc3(hidden)
        return mu, log_var

    # Reparameterization trick
    def reparameterize(self, mu, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mu + eps * std

    # Decoder part, input and output are consistent with ConvCVAE
    def decode(self, z, y):
        # z: (batch, latent_size), y: (batch, 10)
        z_concat = torch.cat([z, y], dim=1)
        hidden = F.relu(self.fc4(z_concat))
        recon_x = torch.sigmoid(self.fc5(hidden))
        return recon_x

    # Forward pass, interface identical to ConvCVAE
    def forward(self, x, y):
        mu, log_var = self.encode(x, y)
        z = self.reparameterize(mu, log_var)
        recon_x = self.decode(z, y)
        return recon_x, mu, log_var


class ConvCVAE(nn.Module):

    def __init__(self, latent_size=64):
        super(ConvCVAE, self).__init__()
        self.labels = 10
        # Encoder: image channels (1) concat one-hot label (10) --> 11 channels
        self.encoder_conv = nn.Sequential(
            nn.Conv2d(11, 32, kernel_size=4, stride=2, padding=1),  # 28x28 -> 14x14
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1),  # 14x14 -> 7x7
            nn.ReLU())
        self.fc_mu = nn.Linear(64 * 7 * 7, latent_size)
        self.fc_logvar = nn.Linear(64 * 7 * 7, latent_size)
```

```
67
68          # Decoder: concatenated latent vector with label (latent_size+10)
            ↪  -> feature map
69          self.fc_decode = nn.Linear(latent_size + 10, 64 * 7 * 7)
70          self.decoder_deconv = nn.Sequential(
71              nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2, padding=1),
                ↪  # 7x7 -> 14x14
72              nn.ReLU(),
73              nn.ConvTranspose2d(32, 1, kernel_size=4, stride=2, padding=1),
                ↪  # 14x14 -> 28x28
74              nn.Sigmoid())
75
76      def encode(self, x, y):
77          # x shape: (batch, 1, 28, 28), y shape: (batch, 10)
78          # Expand y spatially and concatenate with x along channel
            ↪  dimension.
79          y_expanded = y.view(y.size(0), y.size(1), 1, 1).expand(-1, -1,
            ↪  x.size(2), x.size(3))
80          x_cat = torch.cat([x, y_expanded], dim=1)  # Now: (batch, 11, 28,
            ↪  28)
81          conv_out = self.encoder_conv(x_cat)  # (batch, 64, 7, 7)
82          conv_out = conv_out.view(conv_out.size(0), -1)
83          mu = self.fc_mu(conv_out)
84          logvar = self.fc_logvar(conv_out)
85          return mu, logvar
86
87      def reparameterize(self, mu, logvar):
88          std = torch.exp(0.5 * logvar)
89          eps = torch.randn_like(std)
90          return mu + eps * std
91
92      def decode(self, z, y):
93          # Concatenate latent vector with label.
94          z_cat = torch.cat([z, y], dim=1)
95          fc_out = self.fc_decode(z_cat)
96          fc_out = fc_out.view(-1, 64, 7, 7)
97          recon_x = self.decoder_deconv(fc_out)
98          return recon_x
99
100     def forward(self, x, y):
101         mu, logvar = self.encode(x, y)
102         z = self.reparameterize(mu, logvar)
103         recon_x = self.decode(z, y)
104         return recon_x, mu, logvar
```

1.2 训练结果

分别训练两个模型各 50 个 epochs，进行模型图像生成，重构，以及平滑度测试，测试结果如下

1.2.1 MLP 模型

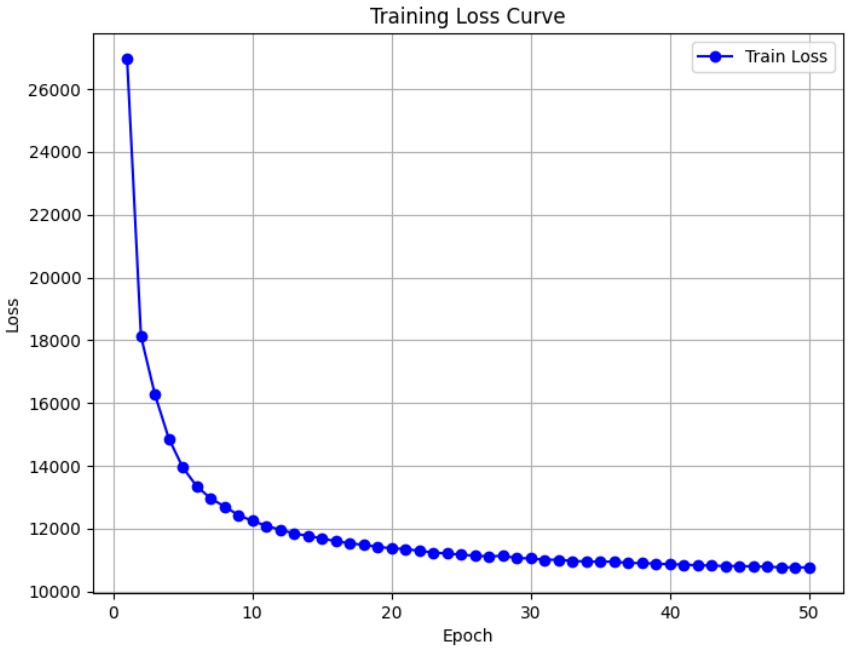
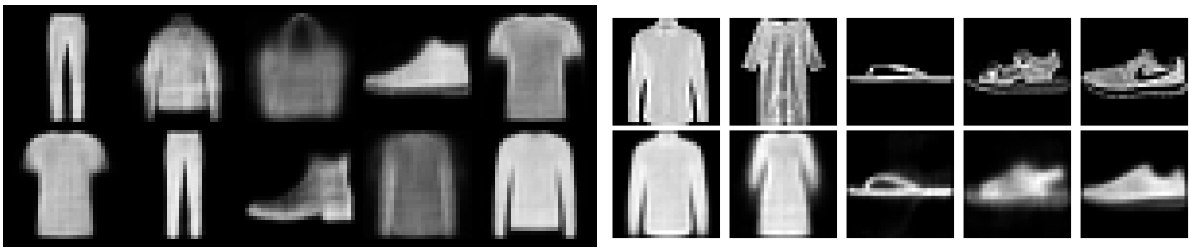


图 1: MLP 训练损失函数曲线



(a) 图像生成 (b) 图像重构

图 2: 重构图像效果

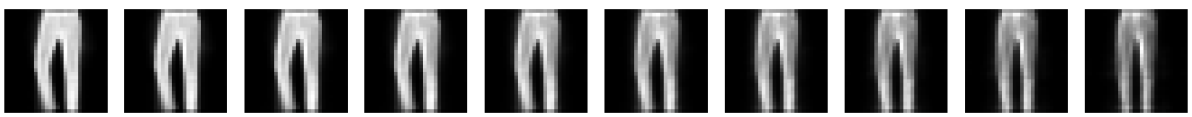


图 3: 隐空间连续差值采样

1.2.2 卷积模型

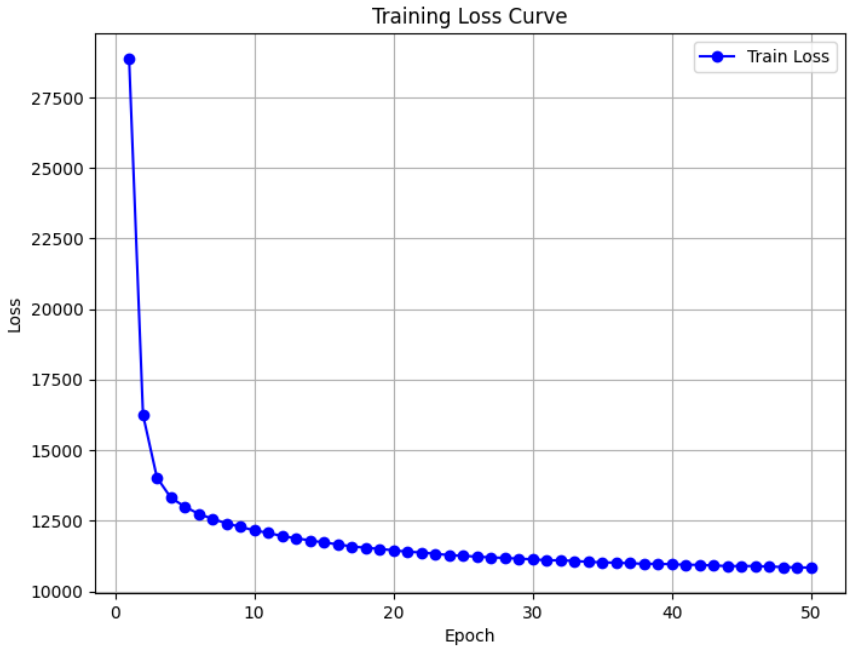
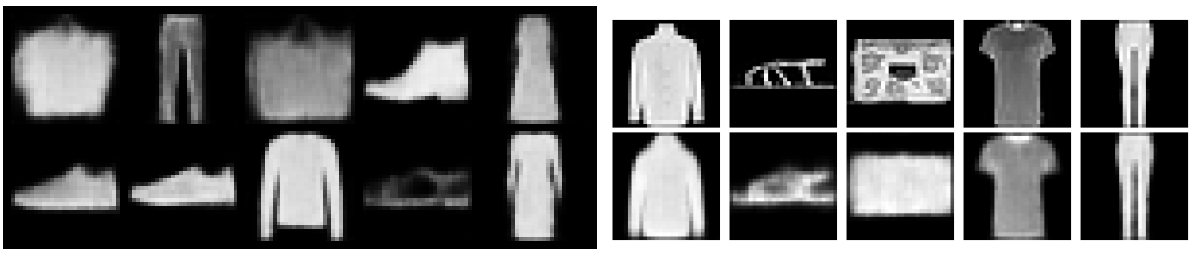


图 4: CNN 训练损失函数曲线



(a) 图像生成 (b) 图像重构

图 5: 重构图像效果

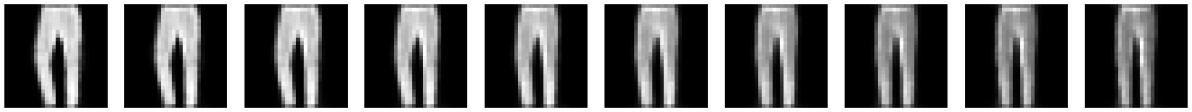


图 6: 隐空间连续差值采样

可以发现卷积结构 CVAE 拥有与 MLP 结构相同的效果，而模型大小从 MLP 的 3.5M 下降到 2.8M，更加轻量化.

2 理论分析

2.1 先验分布与隐空间结构化

VAE 假设潜在变量 z 服从标准高斯先验，这一选择具有多重理论意义：

- 几何正则性：各向同性高斯分布强制潜在空间满足局部连续性，使得解码器的微小扰动对应生成结果的平滑过渡
- 计算可行性：高斯分布的 KL 散度具有闭合解，极大简化优化过程
- 信息瓶颈作用：通过 KL 项约束隐变量携带信息量，防止模型退化为确定性自编码器

2.2 近似后验分布与推断网络

编码器作为高斯近似后验的参数化形式，其设计隐含以下理论权衡：

- 表达能力限制：假设虽简化计算，但忽略变量间相关性，导致后验坍塌风险
- 变分间隙分析：真实后验与近似后验的 KL 散度构成 ELBO 目标的下界间隙，直接影响模型容量
- 重参数化技巧：通过实现梯度回传，本质是路径导数在连续分布场景的应用

2.3 高斯假设的深层影响

(1) 解码器输出建模

对于图像数据，常假设或伯努利分布．这种选择导致：

- 像素独立性假设：忽略像素间空间相关性，生成图像常出现模糊现象
- 似然函数局限性：L2 重构损失与人类感知差异的不匹配性，促使后续研究转向对抗训练（如 VAE-GAN）

(2) 隐变量非线性纠缠

虽然高斯先验强制隐空间局部线性，但解码器的深度非线性变换导致：

- 流形学习视角：VAE 本质学习从潜空间到数据流形的可微映射，高斯先验对应流形上的均匀测度
- 解耦表示困境：标准高斯先验难以自然产生解耦表征，需引入解耦正则项（如 β-VAE）