机器学习与数据挖掘

期末课程报告

(2023年秋季学期)

班级	学号	姓名
计科（人工智能与大数据实验班）	21307404	黄河锦

1. 研究问题的背景和动机

1.1 背景

1.2 动机

2. 当前解决该问题的主要方法

2.1 概述

2.2 自编码器

2.3 变分自编码器

3. 实验部分：使用的模型、算法或方法

3.1 基础的自编码器模型（AE）

3.1.1 自编码器架构

a) 编码器

b) 解码器

3.1.2 训练过程与优化

3.1.3 实验结果可视化

3.2 改进的自编码器模型：使用卷积神经网络架构（CNN-AE）

卷积自编码器架构

a) 卷积层编码器

b) 反卷积层解码器

3.3 变分自编码器模型（VAE）

3.3.1 模型结构与组件

a) 编码器:

b) 解码器:

3.3.2 训练算法

3.3.3 可视化

4. 实验结果及分析

4.1 基础自编码器模型

4.1.1 实验结果

4.1.2 探究不同的隐空间维度对模型性能的影响

4.1.3 基础自编码器模型的局限性及改进方法

4.2 卷积自编码器

实验结果

4.3 变分自编码器

实验结果

5. 结论

5.1 自编码器模型（AE）

5.2 变分自编码器模型（VAE）

5.3 课程报告总结

参考文献

1. 研究问题的背景和动机

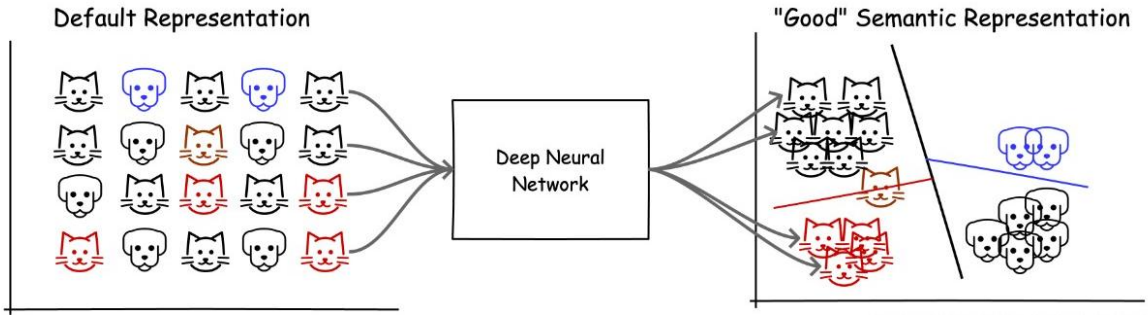
本课程报告选择的主题为 **表征学习** 。

1.1 背景

表征学习，也称为特征学习，是机器学习中的一个重要领域，它致力于自动发现数据中的表示方法。传统机器学习算法通常依赖于手工特征提取，这既耗时且效率低下。表征学习的目标是使机器能够自动学习到能够高效表示原始数据的特征，从而提高学习任务（如分类、回归等）的性能。

1.2 动机

随着深度学习的发展，表征学习在图像识别、语音识别和自然语言处理等诸多领域取得了突破性进展。自动提取的特征比手工设计的特征具有更好的泛化能力和更高的表达力。此外，随着数据量的增加，手工提取特征变得越来越不可行，因此发展有效的表征学习算法具有重要的实践意义。



2. 当前解决该问题的主要方法

2.1 概述

当前解决表征学习问题的主要方法包括：

- **自编码器（Autoencoders）**：通过无监督学习，使网络能够学习到输入数据的压缩表示。
- **变分自编码器（Variational Autoencoders, VAEs）**：一种生成模型，它不仅学习数据表示还能生成新的数据。
- **生成对抗网络（Generative Adversarial Networks, GANs）**：由生成器和判别器组成，能够生成高质量的数据。
- **卷积神经网络（Convolutional Neural Networks, CNNs）**：特别适用于图像数据，能够捕捉局部特征。
- **循环神经网络（Recurrent Neural Networks, RNNs）**：适用于时间序列数据或序列化文本数据。
- **Transformers**：依靠自注意力机制处理序列数据，已成为自然语言处理领域的主流模型。
- **图神经网络（Graph Neural Networks, GNNs）**：适用于图结构数据的表征学习。
- **对比学习（Contrastive Learning）**：最近的研究表明，通过对比损失促使不同类别的表示彼此远离，相同类别的表示彼此接近，可以学习到有用的特征表示。

在本课程报告的实验部分中，我们专注于自编码器（AutoEncoder）和变分自编码器（VAE）的实现，因此接下来我将相对详细的介绍这两部分的内容。

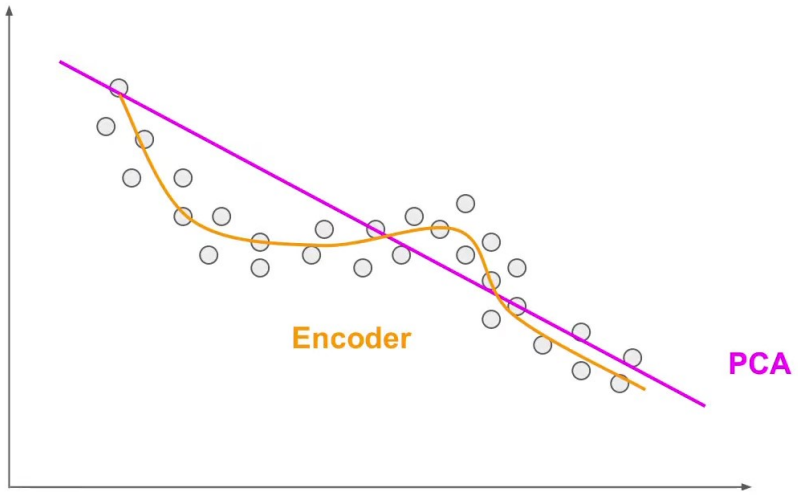
2.2 自编码器

自编码器是一种无监督的表征学习模型，通过编码器将输入数据转换为更低维度的表示，再通过解码器将其重构。关键进展之一是去噪自编码器，通过从输入数据中随机去除噪声，提高模型的鲁棒性和表示能力。Vincent等人（2008年）在图像去噪方面展示了去噪自编码器的有效性[1]。

简单来说，我们希望学习到数据的结构和模式，这显然是一个无监督学习的任务，因为原始的输入数据并没有标签。我们的基本假设是数据在不同维度之间有相关性，因此可以自编码器（auto-encoder）模型能够学习到数据的低维表示。

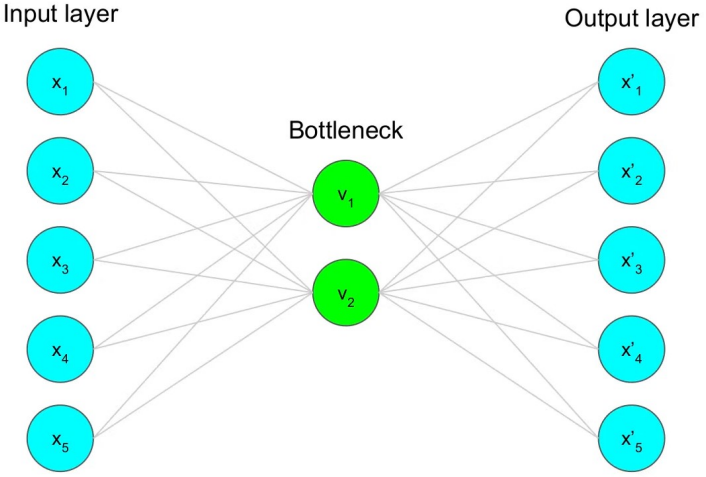
在机器学习的无监督学习领域，PCA降维是不可不提的经典方法。它基于这样一种思想，通过分析输入数据的主成分（principal component），将数据降维到协方差矩阵特征值较大的那些分量中。我们自然会思考，自编码器方法和PCA的区别是什么呢？实际上，从PCA降维的原理可以看到，PCA学习到的是数据特征中线性的相关性，而编码器可以学习到数据特征中非线性的相关性。Hinton等人（2006年）提出的降维方法说明了自编码器在学习非线性表示方面的潜力[2]。

两种方法的对比如下图所示：



因此，编码器可以学习到更复杂的数据结构信息，泛化能力也更强。总的来说，在自编码器模型中，编码器（Encoder）压缩数据到相对低维的隐空间中，然后由解码器（Decoder）重构数据。在训练阶段，将编码器和解码器定义为两个不同的神经网络，利用BP算法最小化重构误差 $E(x,\hat{x})$ 来训练自编码器的神经网络。

这时，从直观上看，我们希望模型对输入数据敏感，以便于重构；但是又不能过于敏感，避免过拟合。为此，我们可以引入正则化因子 $E(x,\hat{x})+reg$ 来平衡这一trade-off。Rifai等人（2011年）通过引入一个局部去噪标准来改进自编码器，提高了其泛化能力[3]。这样，网络既可以通过最小化 $E(x,\hat{x})$ 学习到数据的良好表示，又能通过正则化因子 reg 避免过拟合。其图示如下：

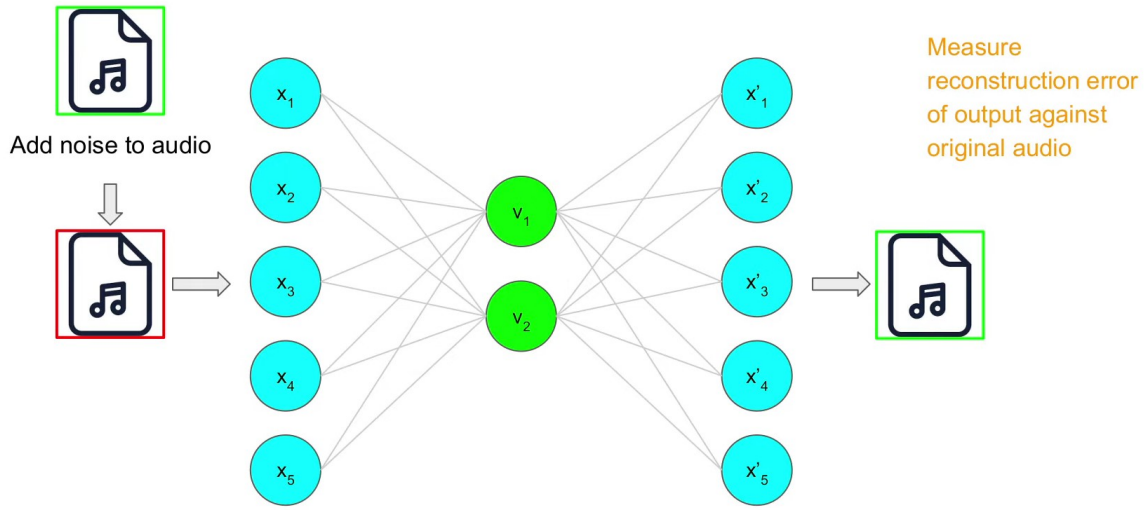


有了这个思想，我们可以结合深度模型很容易地将其扩展为深度自编码器（Deep-AE），其中解码器的网络结构是对编码器的一种镜像。Bengio等人（2007年）展示了通过贪婪逐层训练来有效地训练深度网络的方法[4]。

实际上，隐空间保持了输入数据最重要的特征，因此可以用于生成、去噪和异常检测等下游任务，则也是表征学习的一个重要作用。

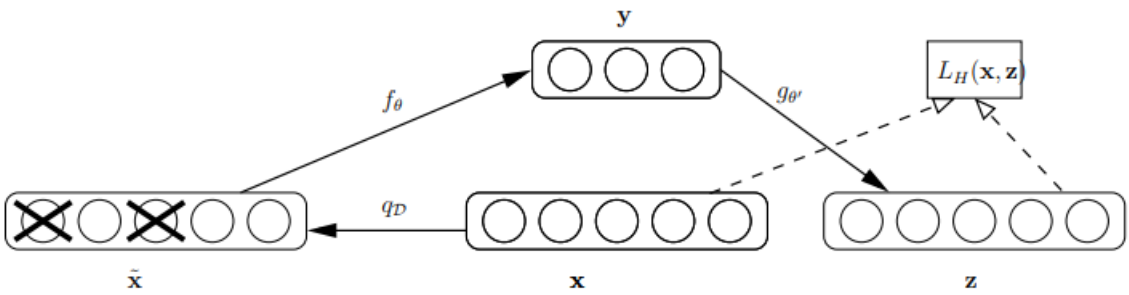
接下来我们以去噪任务为例，简单讨论一下自编码器模型的应用场景。首先我们应该知道，现实世界中采样到的许多原始数据都是带噪声的（corrupted），因此去噪一直是各领域和工业界关注的重点。更具体地说，如果我们希望对一段带噪声的音频或图像去噪，经典的学院派方法是利用傅里叶变换将时域信号转换为频域信号，然后使用相应的滤波器滤除噪点的频率值，再通过傅里叶逆变换还原去噪后的图像。但这一方法显然有很大的局限性，当噪声的模式较为复杂时很难达到较好的去噪效果。

这时，自编码器就派上用场了。我们可以让输入数据带上噪声，然后输入到编码器中，如下图所示：



由于在隐空间中学习到的是数据最重要的特征，因此隐空间中的低维向量可以将噪声信息滤除，再通过解码器重构后，我们就得到了去噪的音频或图像。从中我们也可以看到表征学习的重要性，即聚焦于数据最重要的特征，忽略不重要的信息（如噪声）。

其中一种去噪自编码器的模型示意图如下[5]：

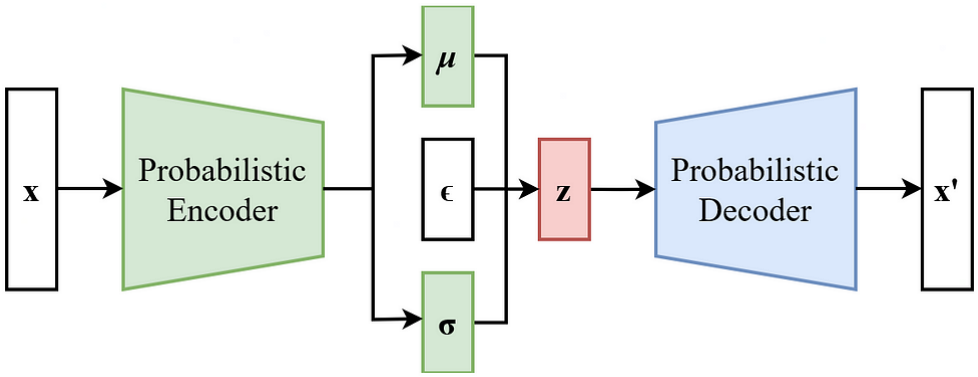


去噪自编码器已被证明可以有效地学习到数据中的有用表示，并忽略噪声。Vincent等人（2010年）对去噪自编码器进行了深入研究，强调了其在表征学习中的作用[5]。

2.3 变分自编码器

变分自编码器（VAEs）是一种深度学习模型，它结合了传统自编码器（*Vanilla Auto-encoder*）的框架和概率图模型的理论，由Kingma和Welling在2013年引入，标志着自编码器在生成模型领域的一大进步[6]。与传统自编码器的确定性映射不同，VAEs的核心在于编码过程中对数据点的概率分布进行建模，这一点使其成为了一种强大的生成模型。

VAEs的关键特性之一是其能够学习到平滑且连续的隐空间。在这个隐空间中，任意两点之间的插值可以通过解码器生成具有意义的中间数据。这种特性使得VAEs在多种任务上都非常有用，例如图像生成、风格转换以及作为表征学习的工具。其原理的图示如下：



VAEs的工作原理基于变分推断（variational inference），它旨在近似复杂分布的后验分布。具体来说，VAEs假设存在一个隐变量模型 $p_\theta(x|z)$，可以生成观测数据 x，其中 z 是潜在的隐变量。在编码阶段，VAEs将输入数据映射到隐变量的分布参数上，通常是均值 μ 和方差 σ^2。这可以表示为 $q_\phi(z|x)$，这是输入数据 x 的隐变量 z 的条件分布。然后，在解码阶段，它从这个概率分布中抽取样本 $z \sim q_\phi(z|x)$，并通过解码器 $p_\theta(x|z)$ 重构数据[6]。

这个过程中引入了KL散度（Kullback-Leibler divergence），一种衡量两个概率分布差异的方法，来作为正则项。这使得VAEs的优化目标不仅包括前面在自编码器模型中提到的重构误差，还包括了隐空间分布与先验分布之间的相似度。因此，VAEs的训练目标是最小化重构误差和KL散度的总和，这在文献中被称为证据下界（ELBO）[6]。ELBO可以表示为：

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta \cdot KL(q_\phi(z|x)||p(z))$$

其中，β 是一个权衡重构误差和KL散度的系数，$p(z)$ 是 z 的先验分布，通常假定为标准正态分布。ELBO的直观解释是，第一项鼓励模型准确地重构数据，而第二项鼓励编码后的潜在变量分布接近先验分布。

VAEs的这种特性使得它们在生成新样本时具有高度的灵活性。例如，可以通过操纵隐空间中的向量来控制生成数据的特定属性，这在条件生成模型中尤其有用。此外，由于VAEs的隐空间是连续的，因此可以通过遍历隐空间来探索新的数据点，从而生成具有丰富变化的数据样本。

VAEs已经被应用于许多领域，包括但不限于图像重构、图像去噪和异常检测。随着研究的深入，VAEs的变体也在不断涌现，进一步提高了模型的性能和应用范围。Rezende等人（2014年）介绍了一种名为正态分布变分自编码器的变体，它通过引入一种更复杂的后验分布来改进VAEs的表现[7]。Higgins等人（2017年）提出的β-VAEs则通过调整ELBO中的权重参数β，使模型能更好地学习因果关系并提高特征解耦能力[8]。

3. 实验部分：使用的模型、算法或方法

3.1 基础的自编码器模型（AE）

自编码器是一种神经网络架构，旨在通过无监督学习方式捕获输入数据的有效表示。在这个过程中，自编码器试图学习一个将输入数据映射到低维空间的函数，同时能够从这个低维表示中重构出原始数据。以下是对本项目中实现的自编码器模型的详细描述和分析，基础数据集为 MNIST 手写数字数据集。

3.1.1 自编码器架构

我们首先采用深度自编码器（Deep-AE）架构实现自编码器。它由两部分组成：编码器（Encoder）和解码器（Decoder）。编码器负责将输入数据映射到称为“隐空间”（latent space）的低维空间，而解码器则从这个隐空间中重构出与原始输入尽可能相似的输出。

a) 编码器

我们的编码器包括四个全连接层，每一层后都跟随一个ReLU激活函数。具体结构如下：

- 第一层接收 MNIST 手写数字数据集的 28×28 的图像向量（即784维的输入），输出128维的表示。
- 第二层将128维的表示压缩到64维。
- 第三层将表示再次压缩到12维。
- 最后一层输出4维的隐空间表征。

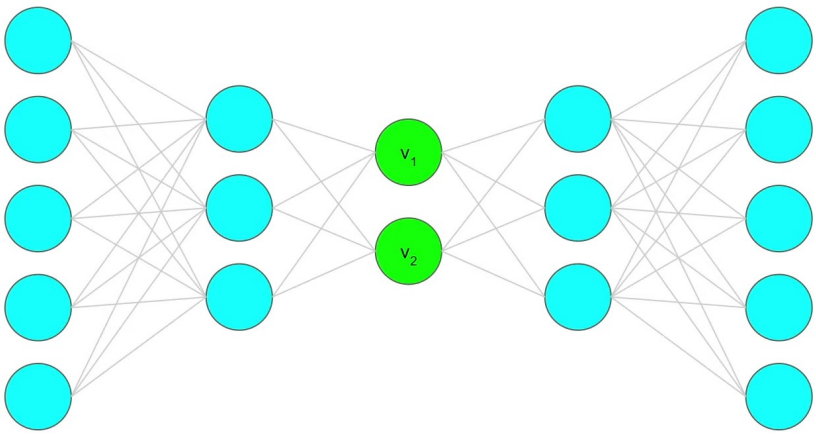
这个逐渐“收缩”的过程迫使网络学习输入数据中最重要的特征，从而实现数据压缩。

b) 解码器

解码器的结构是编码器的镜像。它以4维隐空间向量为输入，通过相反的映射过程逐步扩展到原始数据的大小。

- 第一层接收4维的编码表示，并将其扩展到12维。
- 第二层将12维表示扩展到64维。
- 第三层进一步扩展到128维。
- 最后一层重构出784维的输出向量，通过Sigmoid激活函数确保输出值在[0,1]区间内，与归一化的输入图像对应。

深度自编码器（Deep-AE）架构的示意图如下**（绿色部分为隐空间下的低维数据表征）**：



自编码器架构的 python 代码如下：

```python
import torch
import torch.nn as nn

# 定义自编码器
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        # 编码器
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
```

```
            nn.ReLU(),
            nn.Linear(64, 12),
            nn.ReLU(),
            nn.Linear(12, 4)  # 输出的隐空间表征维度为4
        )

        # 解码器
        self.decoder = nn.Sequential(
            nn.Linear(4, 12),
            nn.ReLU(),
            nn.Linear(12, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 28 * 28),
            nn.Sigmoid()  # 输出值在[0, 1]之间
        )

    def forward(self, x):
        x = self.encoder(x)     # 编码
        x = self.decoder(x)     # 解码
        return x
```

`torchsummary` 是一个用于查看PyTorch模型结构摘要的工具，它提供了一个 `summary` 函数，该函数可以显示模型的层次结构、每一层的输入和输出形状以及参数统计信息等。利用该函数显示出的模型结构如下：

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 128]         100,480
              ReLU-2               [-1, 1, 128]               0
            Linear-3                [-1, 1, 64]           8,256
              ReLU-4                [-1, 1, 64]               0
            Linear-5                [-1, 1, 12]             780
              ReLU-6                [-1, 1, 12]               0
            Linear-7                 [-1, 1, 4]              52
            Linear-8                [-1, 1, 12]              60
              ReLU-9                [-1, 1, 12]               0
           Linear-10                [-1, 1, 64]             832
             ReLU-11                [-1, 1, 64]               0
           Linear-12               [-1, 1, 128]           8,320
             ReLU-13               [-1, 1, 128]               0
           Linear-14               [-1, 1, 784]         101,136
          Sigmoid-15               [-1, 1, 784]               0
================================================================
```

3.1.2 训练过程与优化

我们采用均方误差（MSE）作为损失函数进行优化，因为MSE直接衡量重构图像与原始图像在像素级别的相似度，是图像重构任务中的常见选择。

训练过程中，我们使用了Adam优化器，这是一种广泛用于深度学习任务的优化算法，结合了动量法和RMSProp算法的优点，既能够像动量法那样加快学习速度，又能像RMSProp那样自适应调整每个参数的学习率。

在训练自编码器时，每处理完所有批次的数据（一个epoch），我们记录并打印出当前epoch的损失值。此外，我们还实施了定期的可视化检查，通过比较原图像与重构图像，直观展示模型的学习进度。

对应的 python 代码如下：

```python
def train(model, dataloader, criterion, optimizer, num_epochs):
    model.train()  # 将模型设置为训练模式
    for epoch in range(num_epochs):  # 迭代整个数据集num_epochs次
        for data in dataloader:  # 从数据加载器中逐批次取数据
            img, _ = data  # 获取图像数据，忽略标签
            img = img.view(img.size(0), -1).to(device)  # 将图像数据展成一维向量

            optimizer.zero_grad()  # 梯度清零
            output = model(img)  # 前向传播，计算模型的输出
            loss = criterion(output, img)  # 计算重构图像和原图像之间的MSE损失函数
            loss.backward()  # 反向传播，计算损失关于权重的梯度
            optimizer.step()  # 根据计算的梯度更新模型的参数
        # 每个epoch结束后，打印loss
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')
        if (epoch + 1) % 10 == 0:  # 每10个epoch执行一次可视化操作
            visualize_reconstruction(model, dataloader)  # 可视化重构图像和原图像
```

3.1.3 实验结果可视化

为了直观展示自编码器的性能，我们采用了三种可视化方法：

1. **损失曲线**：我们绘制了损失随epoch变化的曲线图，有助于观察模型的学习趋势和稳定性。
2. **重构质量对比**：通过将原始图像与重构图像进行对比展示，我们可以直观地评估模型重构的准确性。理想的重构图像应保留原始图像的主要特征，同时消除冗余信息。
3. **隐空间可视化**：我们利用PCA算法将隐空间的高维数据降维到2维，然后通过散点图展示不同类别的数据点。这有助于我们了解模型是否能够在隐空间中有效地区分不同类别的数据，以及模型是否学到数据在低维度下的较好表征。

3.2 改进的自编码器模型：使用卷积神经网络架构（CNN-AE）

卷积自编码器是自编码器的一种变体，它利用卷积神经网络（Convolutional Neural Networks, CNNs）来捕捉图像数据更为复杂的空间层次结构，从而更好地进行表征学习。基础数据集为 CIFAR10 十类别数据集，样本数量为 50,000。

卷积自编码器架构

卷积自编码器通过卷积层和反卷积层（或转置卷积层）来实现输入图像的编码和解码过程。我们的模型同样由两个主要部分组成：编码器和解码器。

a) 卷积层编码器

编码器通过一系列卷积层和激活函数逐步提取输入图像的特征。在我们的代码实现中，编码器包含了三个卷积层，每个卷积层后面都跟着一个ReLU激活函数：

1. 第一卷积层有16个3x3的卷积核，步长为2，使用padding，以减小特征图的空间大小，并引入非线性。
2. 第二卷积层将特征图的深度增加到32，同样使用3x3的卷积核和步长为2。
3. 第三卷积层进一步增加深度到64，使用7x7的卷积核，不使用步长和padding，以得到更抽象的高级特征表示。

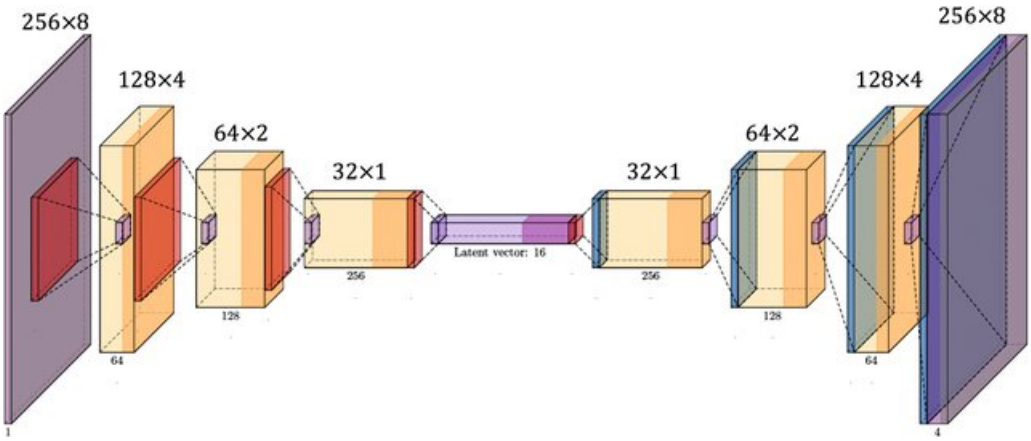
b) 反卷积层解码器

解码器结构镜像于编码器，它通过一系列转置卷积层和激活函数，将编码的特征图逐步重构为原始图像的大小和形状：

1. 第一转置卷积层使用7x7卷积核，将深度从64降低到32。
2. 第二转置卷积层采用3x3的卷积核、步长为2，并使用padding和output_padding，将深度降低至16。
3. 第三转置卷积层将16通道的特征图转换回原始的3通道图像，再次使用3x3的卷积核、步长为2，以及padding和output_padding。

最后，通过一个Sigmoid激活函数，确保输出图像的像素值在[0,1]之间，这对应于输入图像的归一化范围。

卷积自编码器（CNN-AE）架构的示意图如下：



卷积自编码器架构的 python 代码如下：

```python
class ConvAutoencoder(nn.Module):
    def __init__(self):
        super(ConvAutoencoder, self).__init__()
        # 卷积层编码器
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 16, 3, stride=2, padding=1),  # 输出形状: [batch_size, 16, 16, 16]
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, stride=2, padding=1),  # 输出形状: [batch_size, 32, 8, 8]
            nn.ReLU(),
            nn.Conv2d(32, 64, 7),  # 输出形状: [batch_size, 64, 2, 2]
        )
        # 隐空间的维度为 [batch_size, 64, 2, 2]，即隐空间共有64x2x2=256个维度
        # 反卷积层解码器
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(64, 32, 7),  # 输出形状: [batch_size, 32, 8, 8]
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output_padding=1),  # 输出形状: [batch_size, 16, 16, 16]
            nn.ReLU(),
            nn.ConvTranspose2d(16, 3, 3, stride=2, padding=1, output_padding=1),  # 输出形状: [batch_size, 3, 32, 32]
            nn.Sigmoid()
        )

```

```
    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

使用 `torchsummary.summary` 函数显示出的模型结构摘要如下：

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 16, 16]             448
              ReLU-2          [-1, 16, 16, 16]               0
            Conv2d-3            [-1, 32, 8, 8]           4,640
              ReLU-4            [-1, 32, 8, 8]               0
            Conv2d-5            [-1, 64, 2, 2]         100,416
   ConvTranspose2d-6            [-1, 32, 8, 8]         100,384
              ReLU-7            [-1, 32, 8, 8]               0
   ConvTranspose2d-8          [-1, 16, 16, 16]           4,624
              ReLU-9          [-1, 16, 16, 16]               0
  ConvTranspose2d-10           [-1, 3, 32, 32]             435
          Sigmoid-11           [-1, 3, 32, 32]               0
================================================================
```

*卷积自编码器的训练函数与优化方法总体上与 3.1.2 类似，故不再赘述。

3.3 变分自编码器模型（VAE）

VAE是一种强大的生成模型，它通过学习输入数据的概率分布来生成新的数据实例。基础数据集为MNIST 手写数字数据集，样本数量为 60,000。

3.3.1 模型结构与组件

VAE由两个主要组件组成：编码器和解码器。

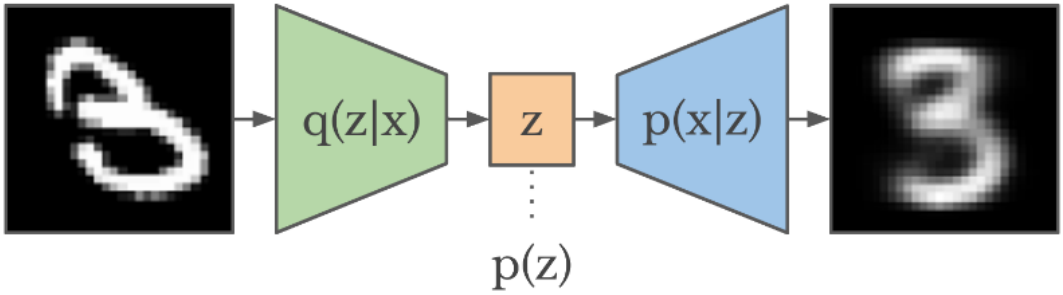
a) 编码器:

1. **输入层**: 接受一个一维784维的向量，代表MNIST数据集中28x28像素的展平图像。
2. **隐藏层**: 一个包含512个神经元的全连接层，激活函数为ReLU，负责提取输入数据的特征。
3. **输出层**: 两个全连接层，一个用于输出均值(`mean`)，另一个用于输出对数方差(`logvar`)。这两个输出定义了一个隐空间（latent space）中的多变量高斯分布。

b) 解码器:

1. **输入层**: 从隐空间中接受一个20维的向量（在此项目中`latent_dims`设为20）。
2. **隐藏层**: 同样使用一个包含512个神经元的全连接层，激活函数为ReLU。
3. **输出层**: 一个全连接层，将解码器的输出转换回784维的向量，并通过Sigmoid激活函数，确保输出像素值在0到1之间。

变分自编码器（VAE）架构的示意图如下，与传统AE模型的核心差异在于引入了概率图模型：



VAE 模型的 python 代码如下：

```python
class VAE(nn.Module):
    def __init__(self, input_dims, latent_dims):
        super(VAE, self).__init__()
        # 定义编码器的第一层，将输入数据映射到隐藏层
        self.fc1 = nn.Linear(input_dims, 512)
        # 定义编码器的第二层，分别输出均值向量和对数方差向量
        self.fc2_mean = nn.Linear(512, latent_dims)  # 均值向量
        self.fc2_logvar = nn.Linear(512, latent_dims)  # 对数方差向量
        # 定义解码器的第一层，从隐空间映射回隐藏层
        self.fc3 = nn.Linear(latent_dims, 512)
        # 定义解码器的第二层，从隐藏层映射到输出，即重建的输入数据
        self.fc4 = nn.Linear(512, input_dims)

    def encode(self, x):
        # 编码器的前向传播过程，使用ReLU激活函数
        h1 = F.relu(self.fc1(x))
        # 返回编码后的均值和对数方差向量
        return self.fc2_mean(h1), self.fc2_logvar(h1)

    # 重参数化技巧，以便进行反向传播
    def reparameterize(self, mean, logvar):
        std = torch.exp(0.5*logvar)  # 计算标准差
        eps = torch.randn_like(std)  # 生成标准正态分布随机数
        return mean + eps * std  # 返回重参数化后的样本点

    def decode(self, z):
        # 解码器的前向传播过程，使用ReLU激活函数
        h3 = F.relu(self.fc3(z))
        # 使用Sigmoid激活函数将输出限制在(0, 1)之间
        return torch.sigmoid(self.fc4(h3))

    def forward(self, x):
        # VAE的前向传播过程
        mean, logvar = self.encode(x.view(-1, 784))  # 编码输入数据并获取均值和对数方差
        z = self.reparameterize(mean, logvar)  # 重参数化以获得隐变量
        return self.decode(z), mean, logvar  # 返回重建数据、均值和对数方差
```

总的来说，VAE的核心是其概率编码和解码过程，以及如何通过优化重构和正则化损失来训练模型。

1. **概率编码**: 编码器输出的均值和对数方差描述了数据在隐空间中的分布。对数方差表示隐变量的不确定性，这是VAE区别于传统自编码器的关键特征。使用重参数化技巧，我们通过均值和方差产生随机样本，以此作为隐变量。
2. **解码过程**: 解码器部分接收来自编码器的隐变量，并尝试重构输入数据。这个过程涉及从隐变量映射回数据空间，目的是最小化重构误差。

模型结构摘要如下：

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                 [-1, 512]         401,920
            Linear-2                  [-1, 20]          10,260
            Linear-3                  [-1, 20]          10,260
            Linear-4                 [-1, 512]          10,752
            Linear-5                 [-1, 784]         402,192
================================================================
```

3.3.2 训练算法

VAE的训练涉及两个关键部分的优化：重构损失和KL散度（Kullback-Leibler divergence）。

1. **重构损失**: 使用二进制交叉熵（BCE）作为重构损失，它衡量重构图像与原始输入之间的相似度。
2. **KL散度**: KL散度衡量编码器产生的分布与标准正态分布之间的差异。这部分损失鼓励模型学习一个接近正态分布的隐空间，这有助于生成新的、多样化的数据实例。

对应的 python 代码如下：

```python
# 损失函数
def vae_loss(recon_x, x, mean, logvar):
    # 计算重构损失，即二元交叉熵损失，对每个像素进行求和
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
    # 计算KL散度，即隐空间的正则化项
    KLD = -0.5 * torch.sum(1 + logvar - mean.pow(2) - logvar.exp())
    # 总损失是重构损失和KL散度的和
    return BCE + KLD

# 训练函数
def train(model, dataloader, criterion, optimizer, num_epochs):
    model.train()  # 将模型设置为训练模式
    loss_history = []  # 用于记录每个epoch的loss

    for epoch in range(num_epochs):
        train_loss = 0
        for batch_idx, (data, _) in enumerate(dataloader):  # 从数据加载器中逐批次取数据
            data = data.to(device)
            optimizer.zero_grad()  # 梯度清零
            # 前向传播，获取重构数据、均值和对数方差
            recon_batch, mean, logvar = model(data)
            loss = criterion(recon_batch, data, mean, logvar)  # 计算损失
            loss.backward()                # 反向传播，计算梯度
            train_loss += loss.item()    # 将当前批次的损失累加到epoch的总损失上
            optimizer.step()               # 根据梯度更新模型参数

        # 计算并记录当前epoch的平均损失
        loss_history.append(train_loss / len(dataloader.dataset))
        # 打印当前epoch的损失信息
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {train_loss / len(dataloader.dataset):.4f}')
        if (epoch + 1) % 10 == 0:  # 每10个epoch可视化一次重构图像和原始图像
            visualize_reconstruction(model, dataloader, device)
```

```
33
34      # 绘制loss曲线
35      plt.figure(figsize=(10, 5))
36      plt.plot(loss_history, label='Training Loss')
37      plt.title('Loss History')
38      plt.xlabel('Epoch')
39      plt.ylabel('Loss')
40      plt.legend()
41      plt.show()
```

3.3.3 可视化

在 VAE 模型部分，我们利用VAE进行了几个应用实践和可视化分析：

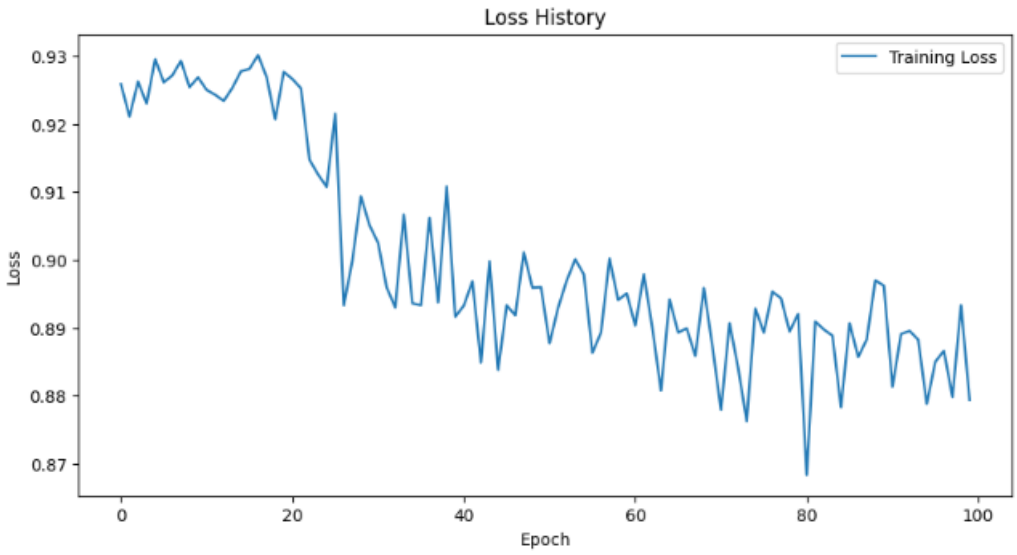
1. **重构**: 我们展示了原始图像与通过VAE重构的图像之间的比较，以展示模型学习重建数据的能力。
2. **隐空间可视化**: 使用PCA将高维隐空间减至二维，可视化不同数字类别在隐空间中的分布情况。
3. **隐空间插值**: 我们在隐空间中进行线性插值，展示了模型如何平滑地过渡并生成介于两个数字之间的图像。

4. 实验结果及分析

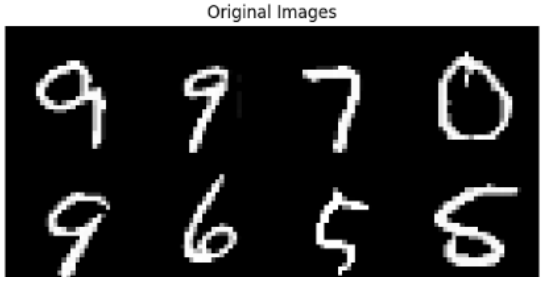
4.1 基础自编码器模型

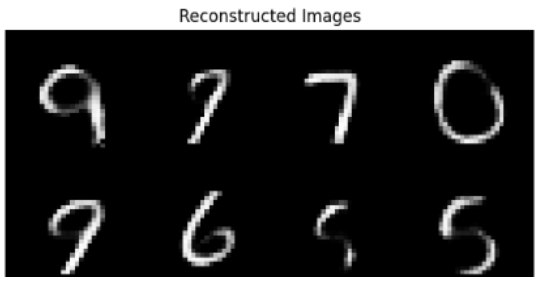
4.1.1 实验结果

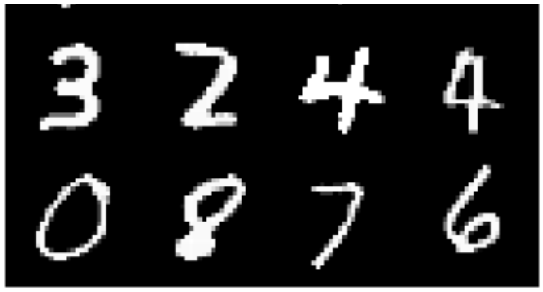
在这个实验中，隐空间的维度设置为 4，batch size 设置为 64，迭代次数 epoch 为 100，优化函数使用 Adam 函数，其默认学习率为0.001，其余均保持默认值。loss曲线如下图所示：

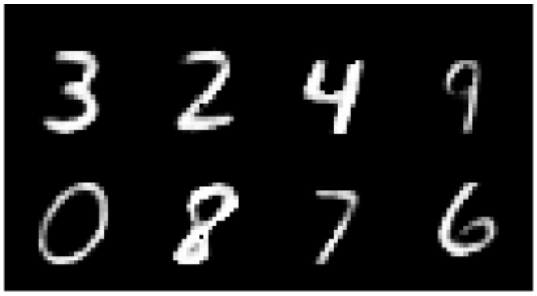


可以看到，基础自编码器模型的loss在震荡中下降，初步判断模型学习到了数据的信息。接下来，给出随机采样的16张原图像和重构后的灰度图像的可视化对比：

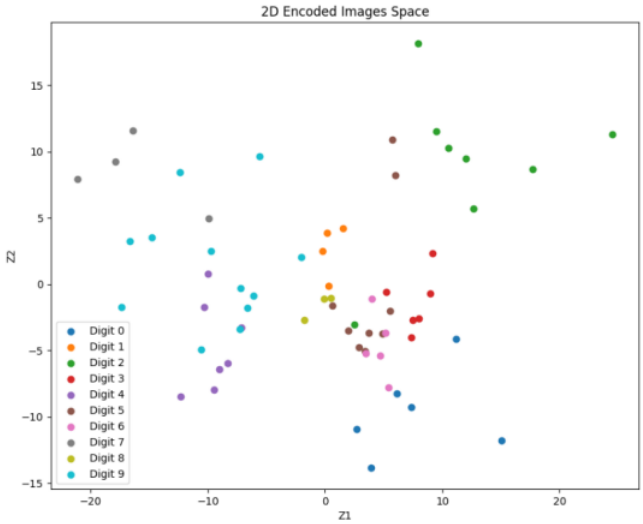








从图中可以看出，此时模型对原图像的重构效果尚可。接下来给出隐空间的低维数据表征的可视化图像，由于隐空间的维度为4，无法直接在二维平面上显示，因此我们将隐空间中的样本点进行PCA降维后再显示。图示如下：

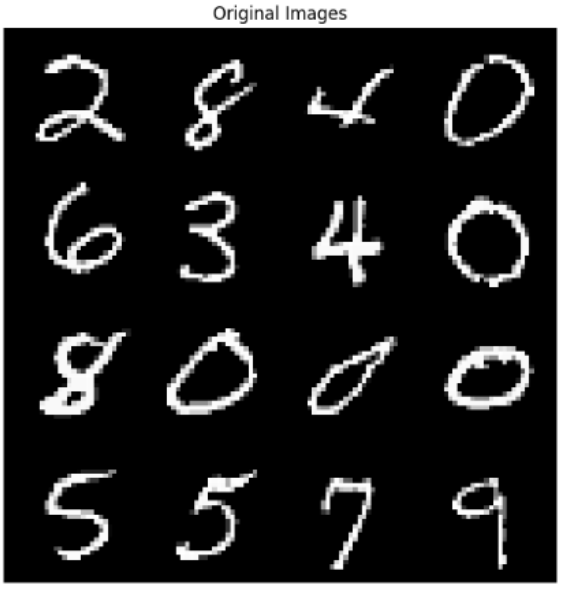


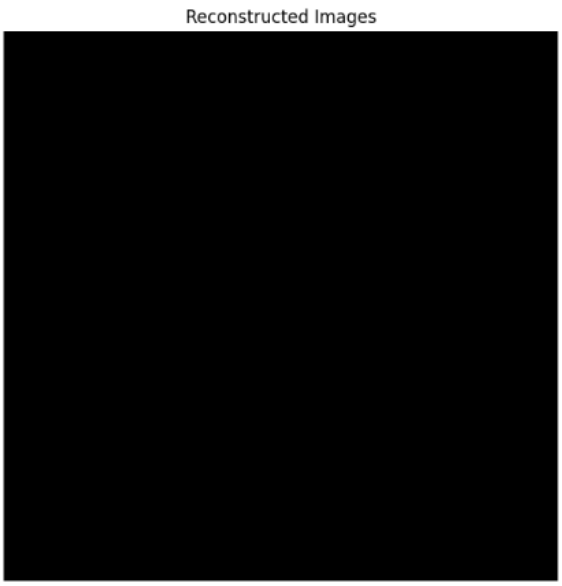
这幅图像很直观地显示了隐空间对数据较好的表征学习效果。不同的数字对应的样本点整体上有序地分簇分布在二维平面的特定区域，而非均匀散布。比如数字2（绿色点）相对集中在图像的右上角，数字1（橙色点）相对集中在图像的中间位置，等等。这说明此时模型已经在隐空间学习到了原来高维数据的低维有效表征，这也充分体现了自编码器在表征学习领域确实是一个可行的模型。

4.1.2 探究不同的隐空间维度对模型性能的影响

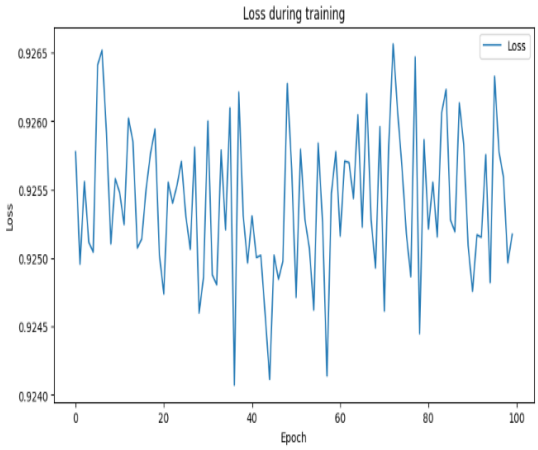
- **隐空间维度为 2**

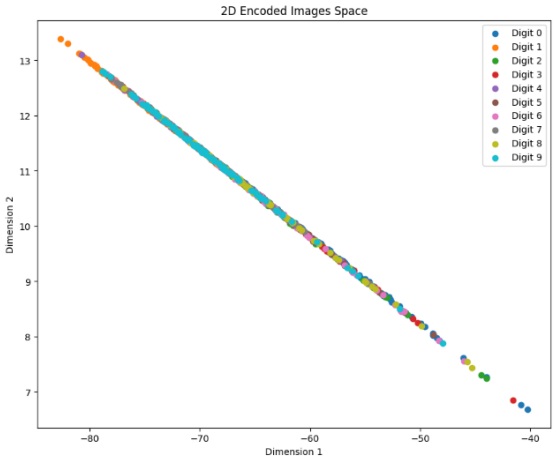
除了隐空间维度改为2之外，其余参数与 4.1.1 中的设置保持一致。在训练 100 个 epoch 后，随机采样的16张原图像和重构后的灰度图像的可视化对比如下：





可以看到，此时重构图像为一副全黑图像，表示当前模型并没有学习到任何有效的信息。我们可以进一步查看此时的loss曲线和隐空间二维散点图：

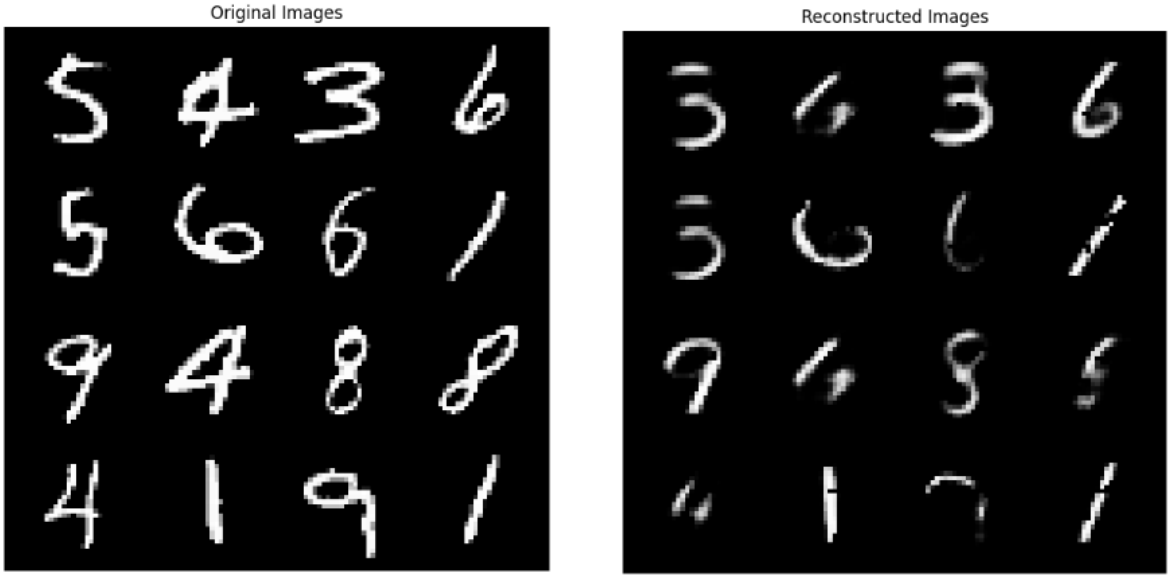




从图中可以看出，loss曲线只震荡而几乎没有下降，隐空间二维散点图中的样本点则几乎是全部分布在一条直线上，说明模型对这些数据的表示甚至没有解耦，因此可以看出二维的隐空间因为维度太低导致了信息的过度丢失而无法还原，且此时模型对数据的表示能力不足。

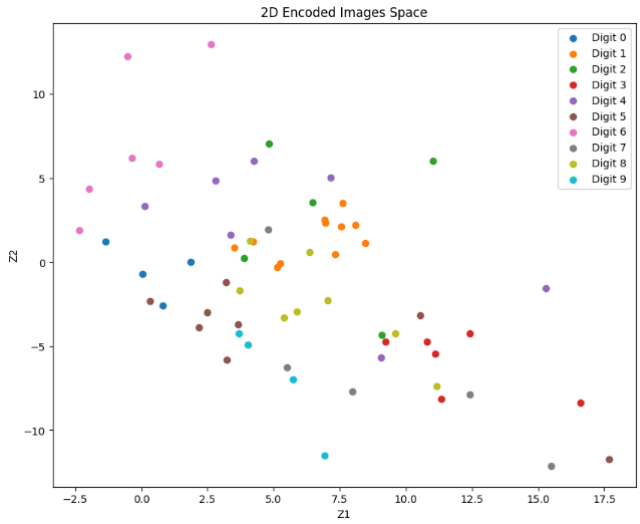
- **隐空间维度为 3**

和上面一样，除了隐空间维度改为3之外，其余参数与 4.1.1 中的设置保持一致。在训练 100 个 epoch 后，随机采样的16张原图像和重构后的灰度图像的可视化对比如下：



可以看到，从隐空间维度为3开始，模型在经过一定的训练论述后已经可以学习到原数据的信息，并可以进行一定程度的重构。与此同时对比 4.1.1 中隐空间维度为4的训练结果，隐空间维度为3的重构效果从直观上看略差一点。

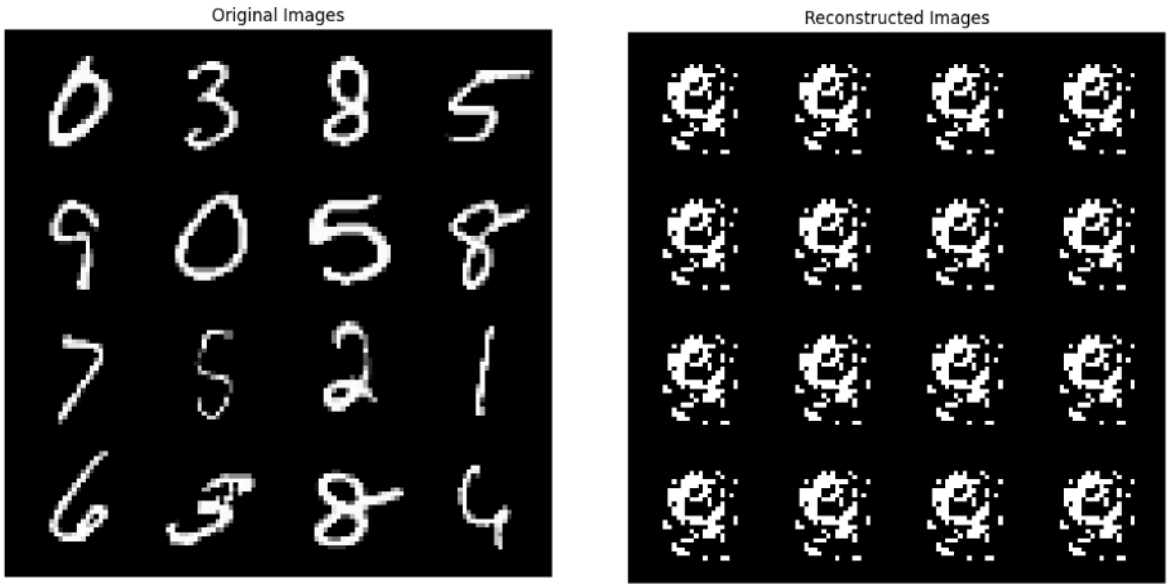
接下来给出此时PCA降维后的隐空间二维散点图：



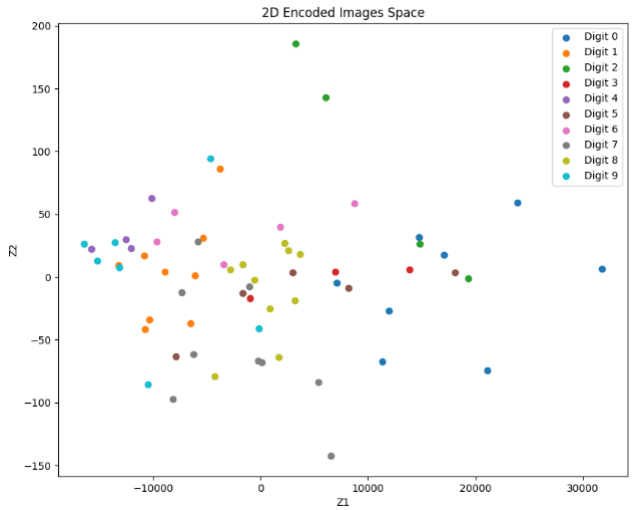
从整体上看，此时隐空间已经能够对不同的数字样本点进行有结构的表征，比如数字1（橙色点）位于图像的中间，数字6（粉色点）位于图像的左上角，等等。但也有一些本该属于同类的样本点较为分散，如数字2（绿色点）。因此总的来说，隐空间的维度至少为3才能学到数据的有效表征，且在一定范围内增加隐空间的维度会使表示能力增强（如对比隐空间的维度为4的实验结果）。

- **隐空间维度为 7**

假使我们继续增大隐空间的维度，当隐空间的维度为7时和上面一样，在训练 20 个 epoch 后，随机采样的16张原图像和重构后的灰度图像的可视化对比如下：



可以看到，此时模型学习到图像的噪声，而非数据中有用的特征，因此不能正确地重构。此时PCA降维后的隐空间二维散点图如下：



值得注意的是，上图的坐标尺度已经发生了数量级的变化，比如Z1轴的尺度为10k数量级，说明样本点已经完全随意地散布在二维平面，说明此时模型也没有学习到原数据任何有效的表示。但同时需要指出的是，隐空间的维度为7对应的模型有时也还是可以正常训练的，但从总体上来说，隐空间的维度太高会放大这种学习到图像噪声的风险，因此选择适当的隐空间维度对于确保自编码器模型能够学习到数据的有效表征是十分重要的。

4.1.3 基础自编码器模型的局限性及改进方法

- **神经网络架构的局限性**

上述在 4.1 中实现的基础自编码器模型采用了全连接网络作为编码器和镜像的解码器的网络架构，之所以还能有不错的实验结果，是因为 4.1 中的实验基于 MNIST 手写数字数据集，而该数据集中的样本的10类样本的特征均相对简单所以直接用全连接网络模型也能正常训练，但如果图像数据集中的数据样本的特征相对更复杂可能难以学习到数据的有效表征，比如下面是使用相同网络架构在 CIFAR10 数据集上训练 100 个 epoch 的重构结果：

对于左图和重构后的右图，可以看到模型丢失了原数据大量的细节，表示效果不佳。

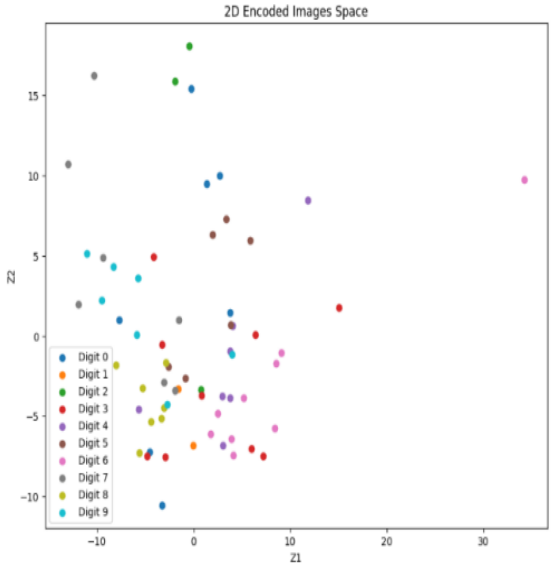
改进方法　在发现上述基础模型的局限性后，基于卷积神经网络能够更好处理图像数据的直觉，我尝试将原来的神经网络换成CNN，结果发现此时的loss曲线居然没有了之前的震荡现象而是能够直接单调下降，并且重构后的图像效果要比右上图好不少！关于将自编码器的神经网络架构改进为CNN的讨论将在 4.2 中进行。

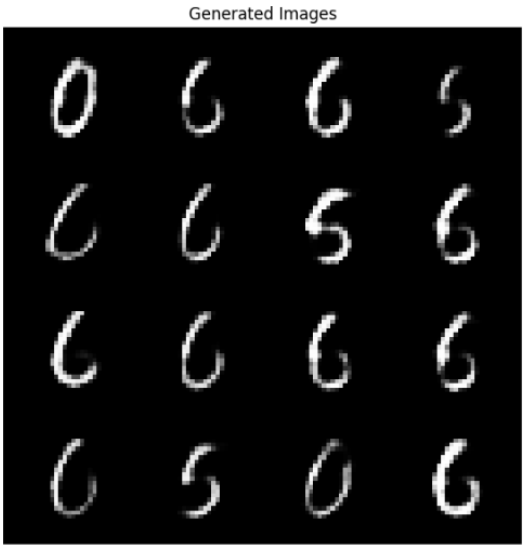
- **传统自编码器模型的局限性**

对 4.1 中PCA降维后的隐空间的二维散点图进行深入分析后，不难发现传统自编码器在生成任务中存在几个影响其性能的局限性：

1. **原点周围的不对称性**：在AE模型中，数据的隐空间表示在原点周围不对称。这种不对称性对于生成新点的采样构成了挑战，因为没有明确的策略来选择能够解码为有效或有意义数据的点。
2. **标签表示的不均匀性**：在隐空间中，一些类别或标签可能在较小区域内表示，而其他类别则占据较大区域。这种不一致性会导致表示中的不平衡，其中某些类型的数据变得过度表示，而其他类型则表示不足。因此，生成样本的多样性可能受到显著限制。
3. **数据点之间的间隙**：隐空间中存在一些没有包含任何数据点的区域，导致了间隙的产生。这些间隙在采样过程中可能会引起问题，因为从这些区域采样的任何新点在解码时可能不对应任何现实或连贯的数据。
4. **生成数据的质量**：由于前述的间隙和隐空间中多样性的缺乏，传统AE模型生成的一些图像或数据点可能质量较差。模型的生成可能不反映真实数据分布的输出，或者无法捕捉输入数据中固有的复杂性和变异性。

上述分析主要针对的是传统自编码器模型在生成新数据能力上的欠缺。下图给出了一个生成的示例：





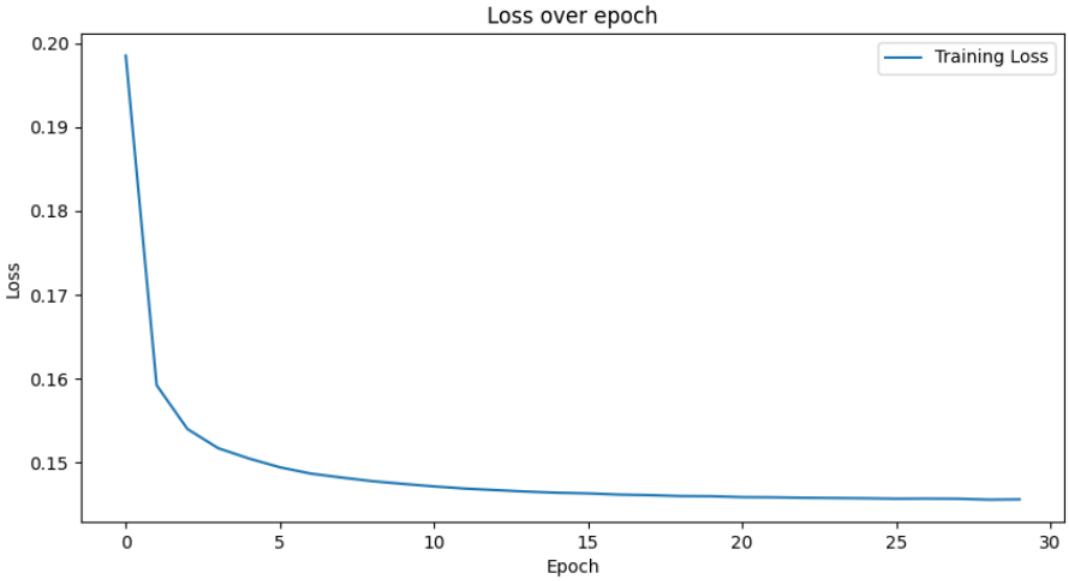
可以看到，正如上面的分析所指出的那样，传统自编码器生成的数据相对比较单一，且有些不是有效数据（不对应0-9的任何数字）。总体而言，这些局限性突显了对改进的生成模型的需求，改进后的模型应该能够产生更均匀和连续的隐空间表示，以确保生成数据的多样性和高质量输出，比如将会在 4.3 中进行讨论的 VAE 模型。

4.2 卷积自编码器

实验结果

- **loss 迭代曲线**

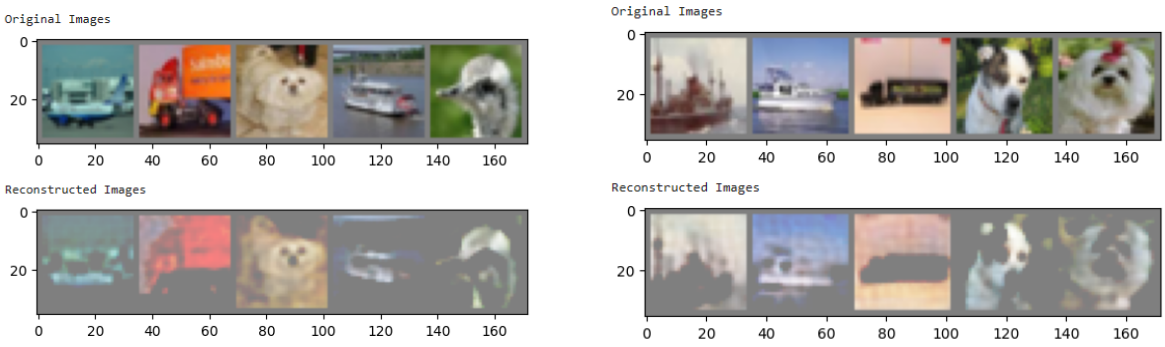
在这个实验中，隐空间的维度为 $64 \times 2 \times 2 = 256$，batch size 设置为 64，迭代次数 epoch 为 30，优化函数使用 Adam 函数，其默认学习率为0.001，其余均保持默认值。loss曲线如下图所示：



可以看到，卷积自编码器模型的loss曲线与 4.1.1 中的情形不同，损失函数几乎是单调下降的，说明模型相对稳定地学习到了数据的特征。

- **重构效果**

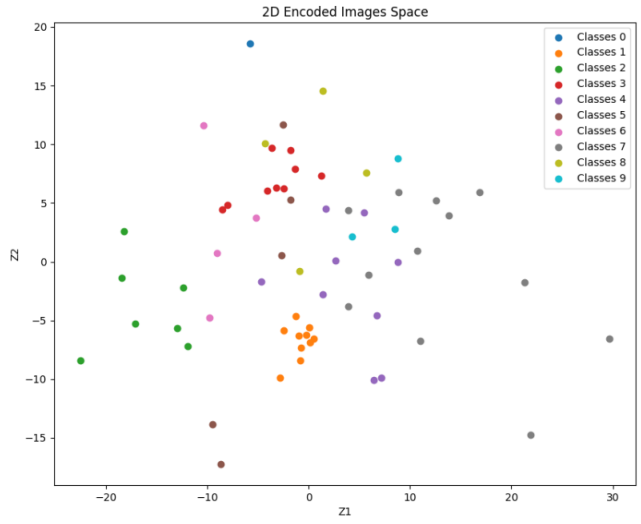
接下来，给出随机采样的10张原图像和重构后的RGB图像的可视化对比：



从视觉效果上来说，重构的质量也要比 4.1.1 中的情况好不少，除了与原图像在色差上有一定区别，在只训练 30 个 epoch 的情况下，细节还是得到了不错的还原。

- **隐空间二维散点图**

CIFAR10的10类别样本数据经过编码器编码到低维的隐空间后，经过PCA降维显示的二维分布如下：



图像体现出了分簇的聚类效果，说明此时模型学习到了有意义的数据表征。总的来说，上述实验结果表明在自编码器中引入卷积神经网络确实是一个不错的想法，适合对图像进行特征提取和低维表征，以供相应的下游任务使用。

4.3 变分自编码器

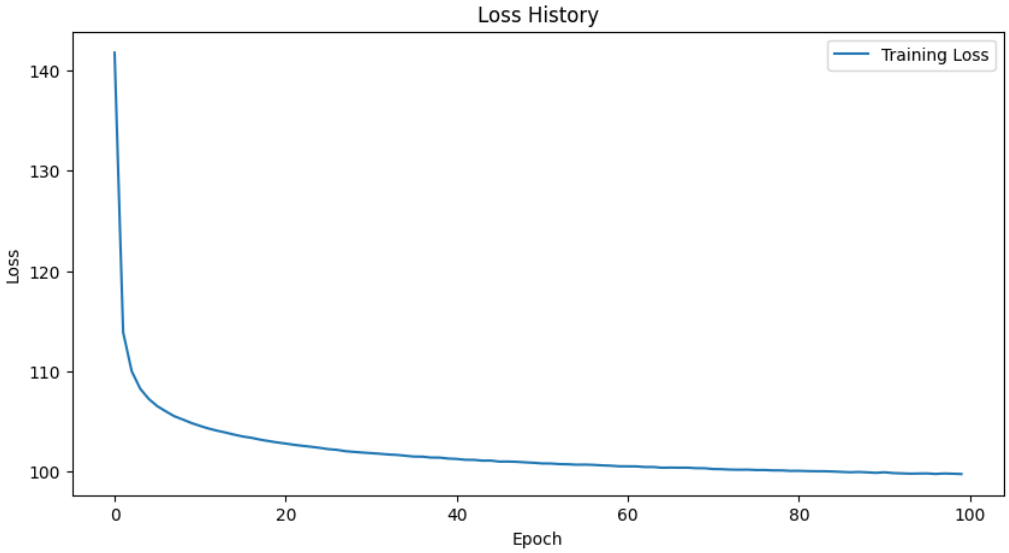
实验结果

- **loss 迭代曲线**

模型的超参数设置如下：

```
input_dims = 28*28
latent_dims = 20
batch_size = 64
epochs = 100
```

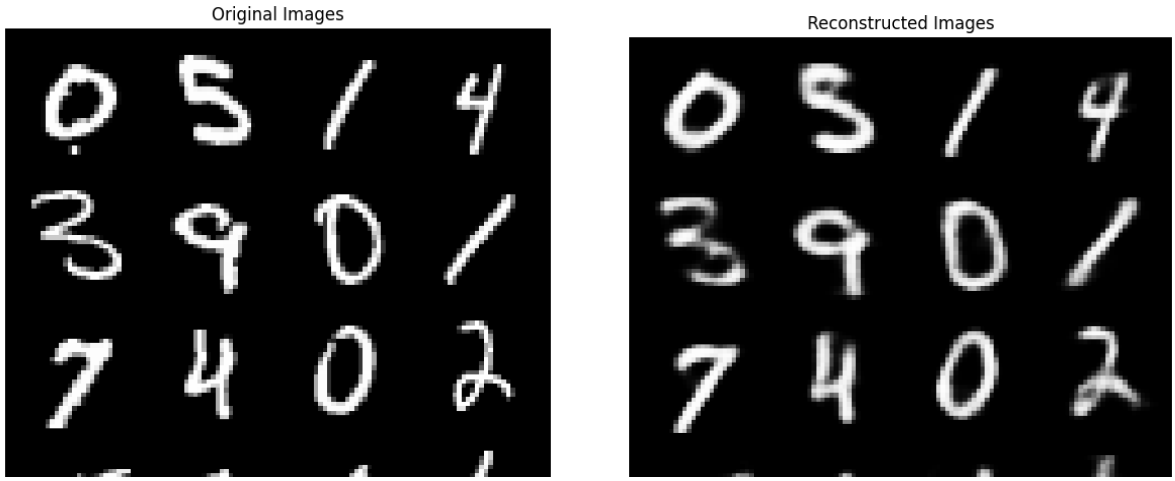
二进制交叉熵（BCE）的损失函数曲线如下图所示：



可以看到，类似于 4.2 中卷积自编码器模型的loss曲线，变分自编码器模型在训练过程中的损失函数的下降也呈现出单调下降的趋势，在训练初期loss下降较快，随后相对平滑地下降，可以初步判断模型有效地学习到了数据在隐空间中的低维表征。

- **重构效果**

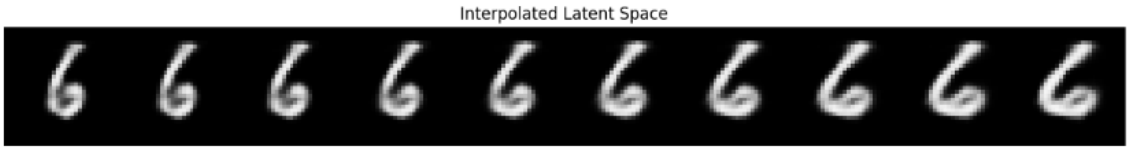
为了充分体现 VAE 模型的出色表现，我们只训练 10 个 epoch，观察其重构表现：

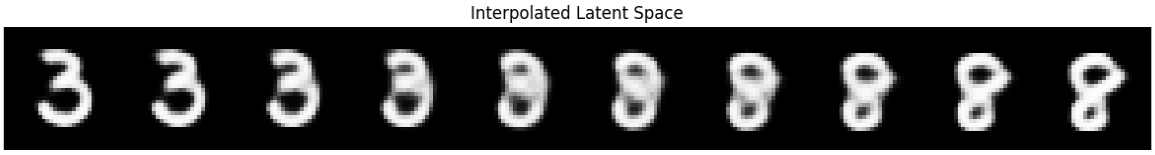


在初次看到这一重构效果时，我有点感到意外，因为 VAE 模型在很少的训练次数下就能已经能达到非常好的重构效果！如上图所示，重构后的图像在视觉上只相当于加了一个低通滤波器平滑噪声，其他的重要特征和细节信息悉数得以保留。而这相比 4.1.1 的传统 AE 模型在训练 100 个 epoch 后的重构结果也要好不少。

- **线性插值**

为了进一步体现 VAE 模型的隐空间是连续的，且可以通过遍历隐空间来探索新的数据点，从而生成具有丰富变化的数据样本这一特性，我尝试了一种线性插值的方法，用以插值生成新样本点。可视化的效果如下：





可以看到，插值的过程还是比较平滑的，很好地体现了隐空间的连续性。

5. 结论

下面的结论基于对实验结果及分析部分的总结。

5.1 自编码器模型（AE）

- **隐空间维度的选择**

理论上，在隐空间的维度选择方面，更低的维度会导致信息的更多丢失，而更高的维度能够保留更多信息。然而，这并不意味着维度越高越好，因为过高的维度可能会导致模型学习到噪声，而非数据中有用的特征。因此，选择一个合适的隐空间维度是实现有效表征学习的关键。

通过 4.1 中的讨论，在本实验中，自编码器隐空间的维度选择为3或4的模型表征能力较好。

- **神经网络架构的改进**

正如我们在 4.2 中看到的那样，对于相对复杂的图像数据，如CIFAR10中的数据样本，将神经网络架构改为卷积神经网络效果更好。

卷积自编码器作为一种强大的特征学习工具，其潜力在各种图像处理任务中得到了验证。在未来的研究中，我们还可以探究其在更复杂任务中的应用，如语义分割、图像超分辨率等领域。

5.2 变分自编码器模型（VAE）

通过训练VAE，我们能够学习到一个有意义的隐空间，其中相似的数字被映射到相近的区域。VAE的这种能力使得它非常适合进行无监督学习，数据生成和数据压缩等任务。未来，我们可以通过改进网络结构、调整超参数或利用更复杂的概率分布来进一步提高VAE的性能。

总的来说，VAE提供了一个强大的框架来学习复杂数据集的隐结构，同时提供了生成新数据实例的能力。通过本课程项目，我不仅理解了VAE的理论基础，还通过实践掌握了其应用和可视化方法，为未来在更广泛的问题和数据集上应用VAE奠定了基础。

5.3 课程报告总结

在本课程项目中，我们深入探索了表征学习的核心概念、方法。特别地，我们聚焦于自编码器（AE）和变分自编码器（VAE）模型，其中后者结合了自编码器架构和概率图模型的先进生成模型，它在多个方面提升了我们对数据的理解和处理能力。

通过理论学习和实验，我充分认识到了表征学习的重要性。表征学习作为机器学习领域的一个重要分支，为数据分析、特征提取、复杂系统建模和新领域的研究提供了丰富的工具和方法。未来，我期待将所学知识应用于更广泛的数据集和复杂的问题中，同时也期待这一领域的进一步研究和发展。

参考文献

[1] Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008). Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning (pp. 1096-1103). ACM.

[2] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507.

[3] Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In Proceedings of the 28th International Conference on International Conference on Machine Learning (pp. 833-840). Omnipress.

[4] Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Advances in neural information processing systems (pp. 153-160).

[5] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11(Dec), 3371-3408.

[6] Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. arXiv preprint arXiv:1312.6114.

[7] Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. arXiv preprint arXiv:1401.4082.

[8] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., ... & Lerchner, A. (2017). β-VAE: Learning basic visual concepts with a constrained variational framework. In 5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings.