|  |
| --- |
| 电子科技大学  **计算机专业类课程** |
| **实验报告** |
| **课程名称：人工智能综合实验II**  **学院专业：计算机科学与工程学院**  **学生姓名：崔若晨**  **学　　号：2021060902008**  **指导教师：石小爽** |
| **日　　期： 2023 年 9 月 19 日** |
|  |
| 电子科技大学计算机学院实验中心 |

**电 子 科 技 大 学**

**实 验 报 告**

**实验四**

# 一、实验室名称：

电子科技大学清水河校区主楼A2-413-1

# 二、实验项目名称：

人工智能实验II-4：基于自动编码器的手写数字生成

# 三、实验原理：

自动编码器是无监督学习方法中的一种结构。它通过编码器提取输入数据的隐含特征，再通过解码器根据隐含特征重构输入，实现了一种自监督学习的方法。其中，编码器和解码器通常为一个非线性映射函数，隐含特征往往比输入数据更加紧凑。因此，自动编码器能够实现数据降维、压缩和数据隐含分布的投影。随着深度神经网络的广泛应用，人们尝试使用神经网络搭建编码器和解码器，并取得了广泛成功。本次实验基于神经网络搭建自动编码器，以探究其在隐变量映射和图像降维与还原任务中体现出的各种特性。

# 四、实验目的：

通过使用 PyTorch 深度学习框架，搭建神经网络，实现自动编码器以进行手写数字生成等任务，并在此过程中掌握：

1. PyTorch 中全连接网络、卷积网络等的实现方法；
2. 自动编码器的基本框架和设计思路；
3. 探究自动编码器在图像降维和还原中体现的具体功能。

扩展掌握：

1. 自动编码器的隐变量可视化分析；
2. 变分自编码器的实现方式；
3. 对抗生成网络的实现方式。

# 五、实验内容：

* 1. 问题描述

设计自动编码器（AutoEncoder）生成MNIST和CIFAR数据集图像

* 1. 算法的概要设计与分析

AutoEncoder：将数据集中的数据输入Encoder中进行编码，然后将Encoder的结果输入的Decoder中进行解码，生成图片。Loss为生成图片和原图片基于loss函数的差异，然后通过反向传播算法更新网络参数。

VAE：将数据集中的数据输入Encoder编码为k个均值和k个方差，然后将k个对应的正态分布输入的Decoder中进行解码，生成图片。Loss为生成图片和原图片基于loss函数的差异加上0.1权重的先验分布和变分后验分布的KLloss，然后通过反向传播算法更新网络参数。

GAN：由生成器和判别器“左右互博”。我们把伪造的图片和真实的图片交给判别器，并训练判别器辨认哪些图片是伪造的，哪些是真实的。生成器则恰恰相反，它根据判别器的判断来调整自己，尽可能去欺骗判别器。具体操作时：根据随机噪声，生成器生成一组伪造图片。把伪造图片和真实图片交给判别器，并告诉它真实标签，训练进行二分类，判断图片是“真”还是“假”。生成器再生成一组图片，同样交给判别器，但这时我们训练去干扰的分类结果。

1. 核心算法的详细设计与实现

AutoEncoder:如图 1所示，AutoEncoder分为encoder和decoder两部分。Encoder对数据进行编码,通常先提取特征，然后对特征进行编码。常用的特征提取方式有线性层网络（MLP）和卷积神经网络（CNN），特征提取后通常用全连接层（FC）对特征进行编码。Decode根据Encoder决定解码方式。如果采用线性层网络，则Decoder也用线性层网络进行解码。如果Encoder采用卷积网络，则Decoder通过反卷积（ConvTranspose）对特征进行解码。

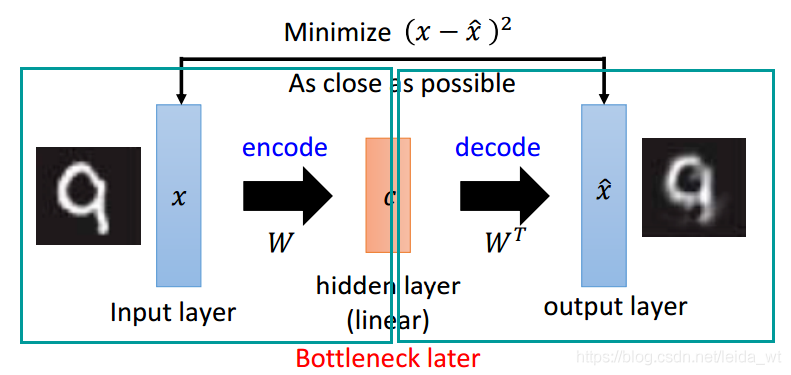


图 1 AutoEncoder结构图

反卷积操作如图 2所示。首先，对输入的feature map进行padding操作，得到新的feature map；然后，随机初始化一定尺寸的卷积核；最后，用随机初始化的一定尺寸的卷积核在新的feature map上进行卷积操作。当s=1时，对于原feature map不进行插值操作，只进行padding操作，这里的padding是卷积操作时设置的padding值，经过padding之后，再经过卷积核进行卷积操作就得到了输出结果。当s>1时，进行插值操作，feature map的元素之间被分隔开了，间隔(s-1)个插入元素0。插值之后的操作同s=1的操作一致。

|  |  |
| --- | --- |
| * 1. S=1 | * 1. S>1 |

图 2 反卷积（ConvTranspose）示意图

Decoder最后要使用Sigmoid函数，将输出固定在0~1之间，这是因为输出的是图像，其像素点大小需要在0~1之间。损失为Decoder生成的图片和真实图片之间的差异，常见的损失函数有MSEloss,binary\_cross\_entropy等。计算出损失后，使用反向传播算法同时对Encoder的参数和Decoder的参数进行更新。

VAE：变分自编码器。它能够学习一个函数/模型，使得输出数据的分布尽可能的逼近原始数据分布。相比AE，VAE不再把分布映射到固定值或变量上，而是映射到一个分布上。如图 3，网络模型中的bottle neck被分解成了两个向量：均值向量方差向量。VAE的Encoder用来计算均值和方差，均值和方差是用来描述正态分布的数字特征，不同的和取值对应不同位置、不同形状的正态分布。这里的均值和方差并不是确定的统计量而是两个函数，是由神经网络训练得到的。其中后验分布是先验分布的近似；是服从标准多元高斯分布的标准化随机变量，是服从高斯分布的随机变量，称为潜在变量或隐变量；是似然分布，即知道变量的分布后根据来估计的分布。当解码时，我们将从每个潜在分布中随机采样，生成一个向量作为解码器模型的输入。由于有了每个属性的概率分布，我们可以简单地从分布中抽取任何值来生成一个新的输出，而且自带过渡效果。

VAE是一种无监督的生成模型，其理论基础是建立在GMM高斯混合模型之上。GMM认为任何一个数据的分布，都可以看作是若干高斯分布的叠加。是由一个标准正态分布所产生的向量，的每一维度都代表了一种属性，且的每一个属性的分布都服从高维高斯分布，通过每一个属性的高维高斯分布来拟合输入样本的真实分布、对于每个采样，会有两个函数和，分别对应到高斯分布的均值和方差，然后在积分域上所有的高斯分布的累加就得到

原始分布的近似，即，整个网络就是把样本经过

降维表达成，再由根据低维特征重构原始输入，对应了一个完整的Encoder-Decoder过程。

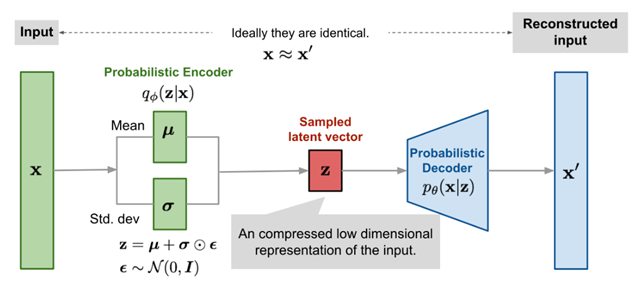


图 3 VAE结构图

我们的目的就是求解的分布。由于是已知的，未知而，于是我们真正要求解的是和两个函数的表达式。通常非常复杂，因而引入两个神经网络进行求解：第一个神经网络是Decoder，用于求解和两个函数，等价于求解。第二个神经网络是Encoder，用于求解，可以表示任何的分布。为了让尽可能接近，我们引入KL散度。KL散度是两个分布之间差异的度量其公式为：



通过最小化KL散度，使两个分布尽可能地相似。因而模型的损失函数除了binary\_cross\_entropy之外，还要加上KL散度，这样最小化损失函数既能是让生成的图片跟真实图片接近又能使生成的分布与原分布尽可能接近，从而提高模型的鲁棒性。

# 六、实验器材（设备、元器件）：

硬件平台：PC端

开发环境：操作系统Windows 11

测试环境：Anaconda 3，Python 3.11，PyTorch 2.01

# 七、实验步骤：

1. 网络框架结构搭建介绍

线性层AutoEncoder网络框架如图所示，Encoder由3个全连接层（FC）组成，Decoder也由3个全连接层（FC）组成。

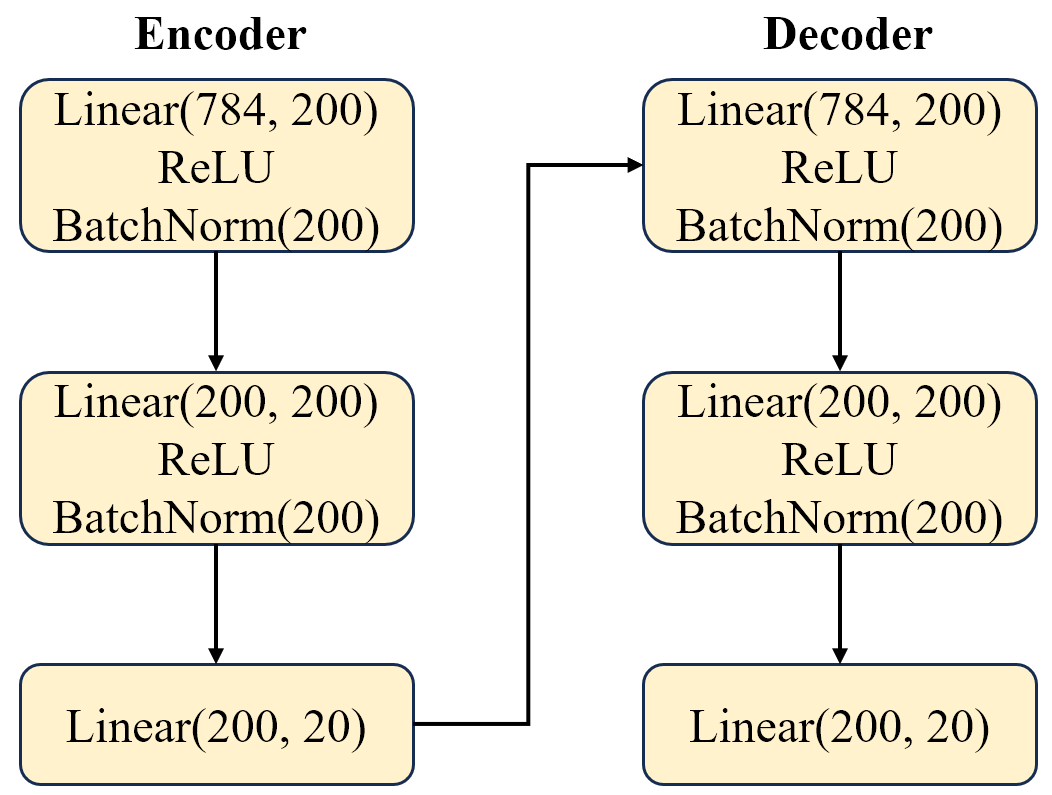


图 4 线性层AutoEncoder网络框架

卷积层 AutoEncoder 网络框架如图 41 所示，Encoder 由 4 个 3\*3 的卷积和 2 个 2\*2 的最大池化层组成，Decoder 由 4 个反卷积层组成。

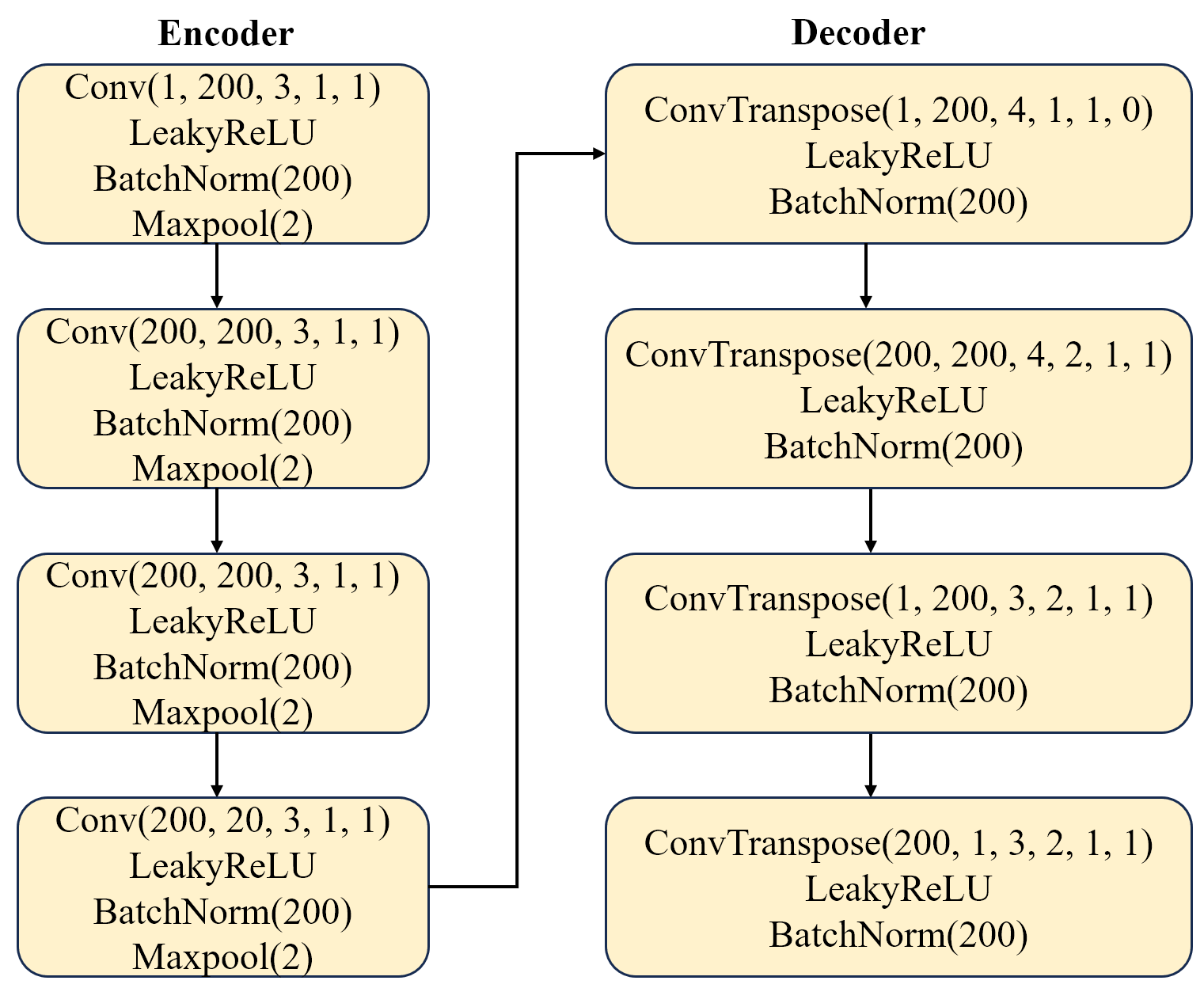


图 5 卷积层AutoEncoder网络框架

Cifar10 卷积 AutoEncoder 网络框架如图所示，Encoder 由 5 个 3\*3 的卷积层组成，Decoder 由 5 个反卷积层组成。

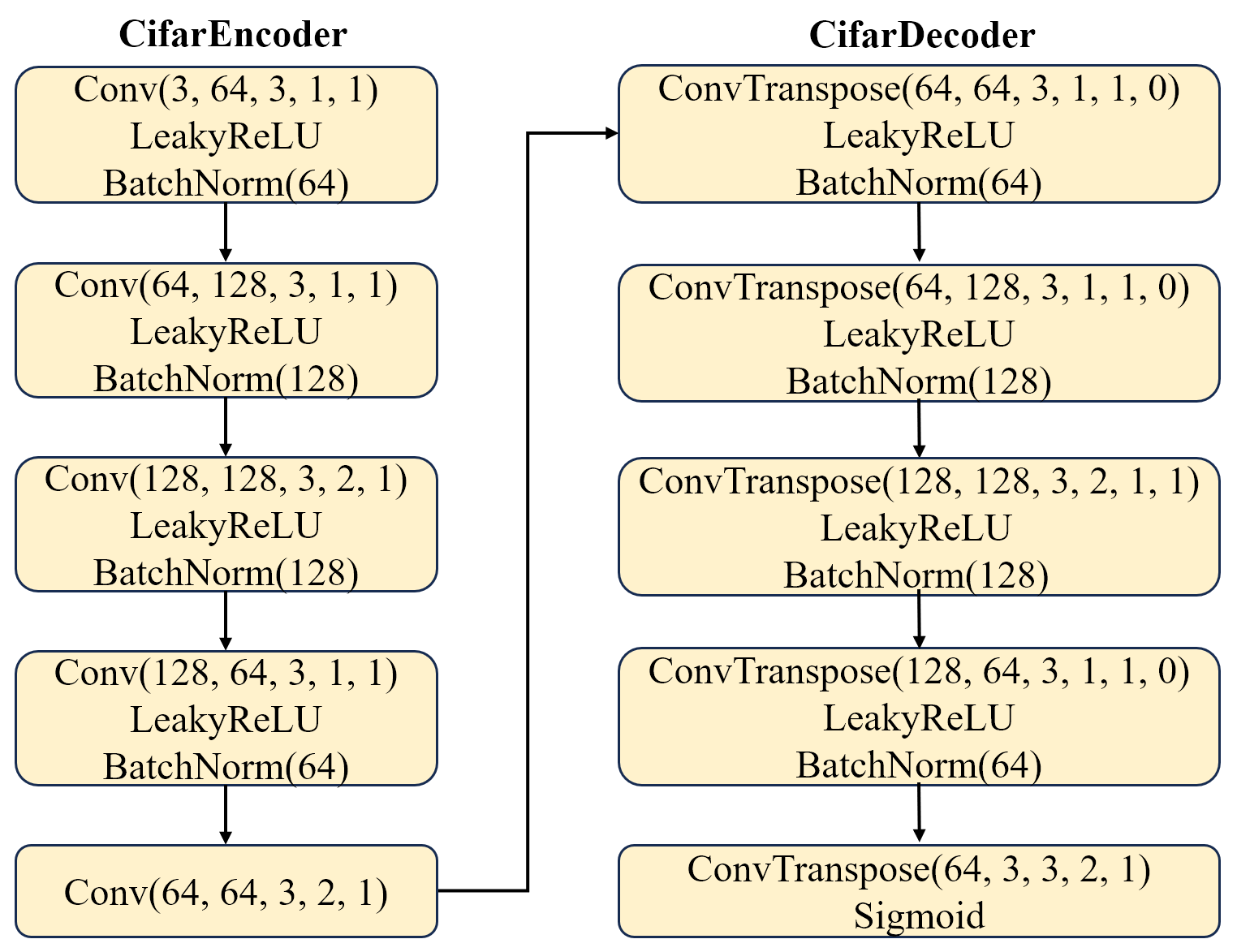
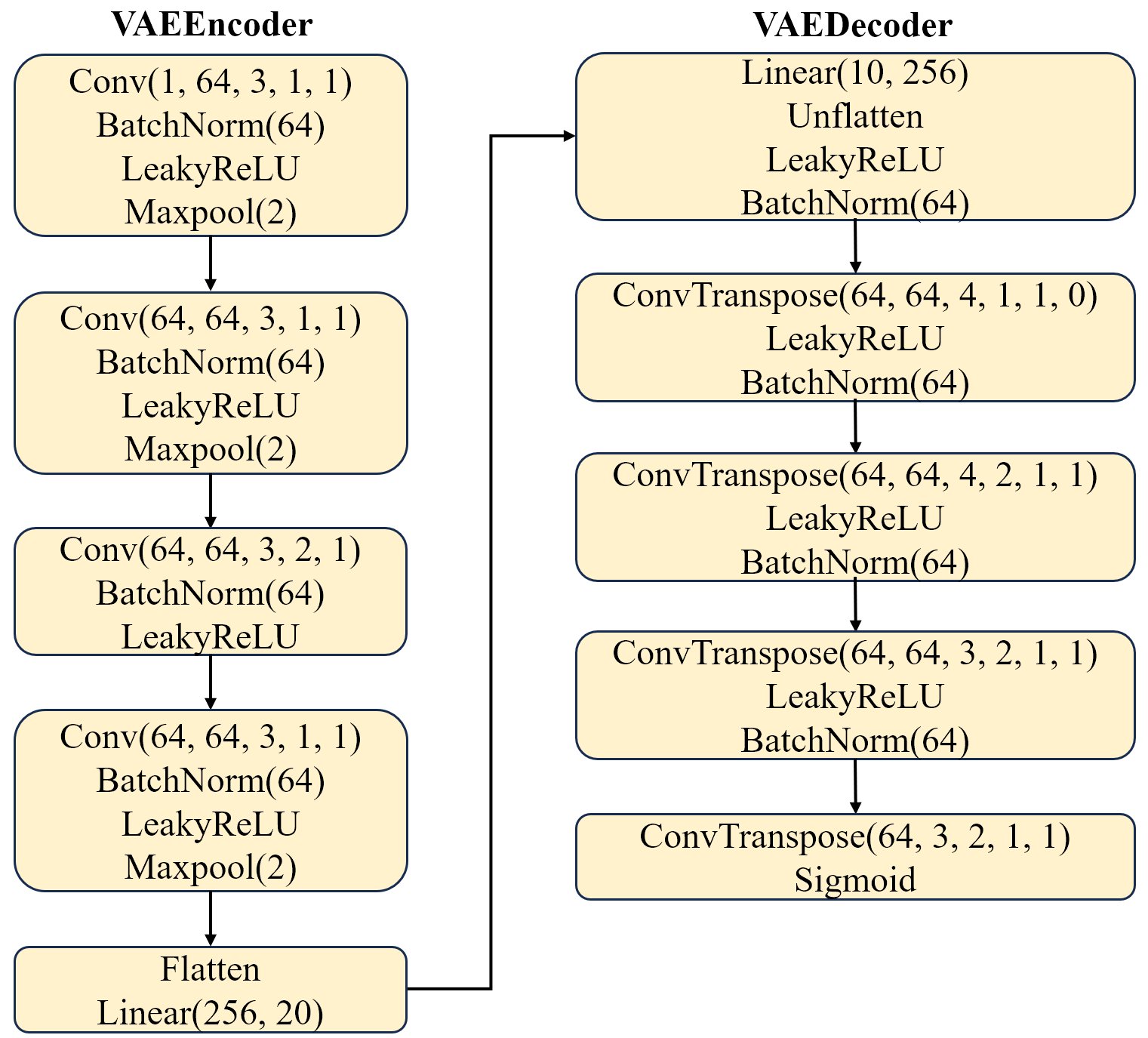


图 6 Cifar10卷积AutoEncoder网络框架

VAE 网络框架如图所示，Encoder 由 4 个 3\*3 的卷积层，3 个 2\*2 的最大池化层和 1 个全连接层组成，Decoder 由 1 个全连接层和 4 个反卷积层组成。

图 7 VAE网络框架

GAN 网络框架如图 44 所示，Discriminator 由 4 个 3\*3 的卷积层和 1 个全 连接层组成，Decoder 由 1 个全连接层和 4 个反卷积层组成。

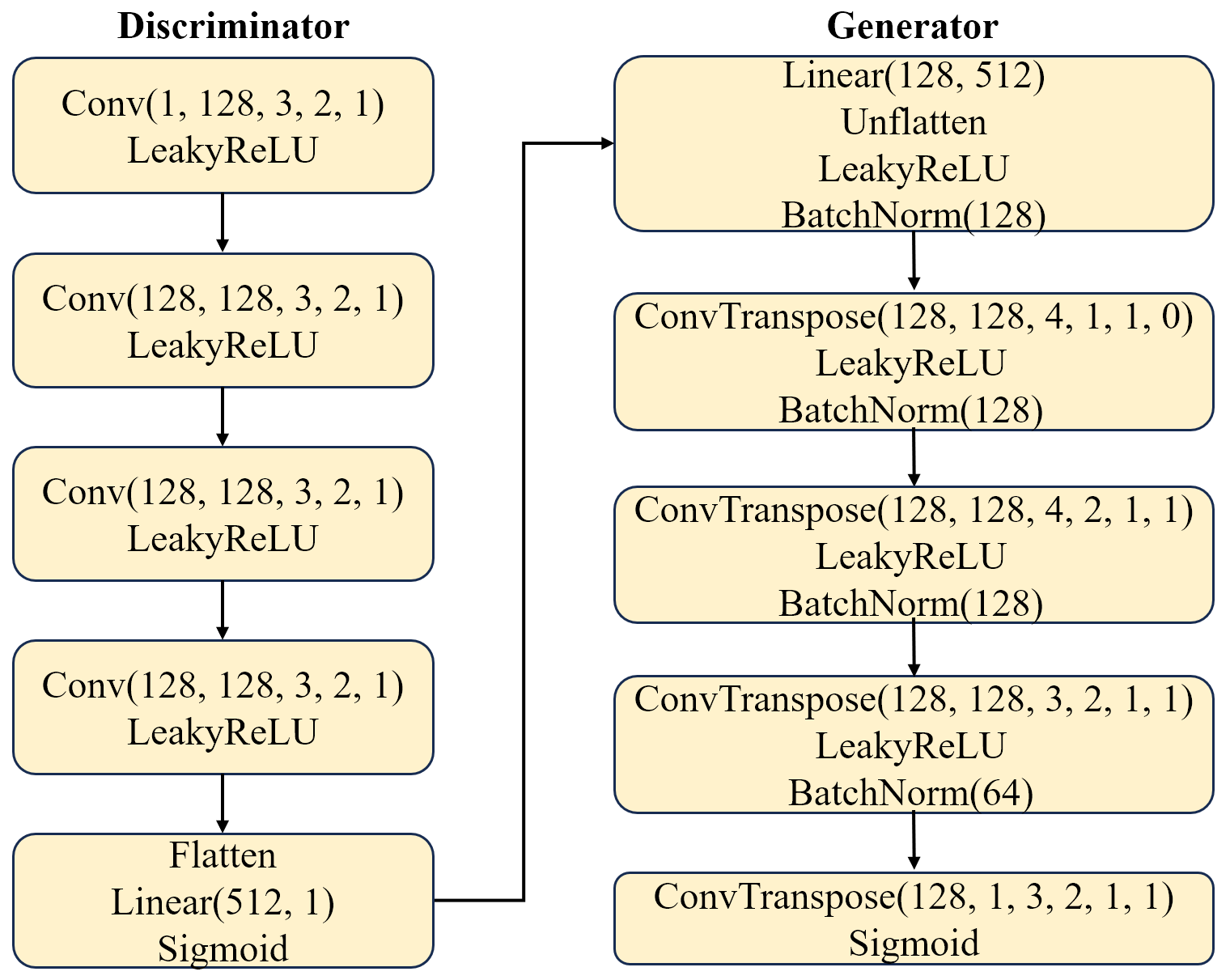


图8 GAN网络框架

1. 核心代码实现
2. *### 线性AutoEncoder*
3. class Encoder(nn.Module):
4. """ Vanilla encoder with 3 fc layers
5. Args:
6. dIn (int): Dim of input, if `x.shape = [N, C, H, W]`, then `dIn = C\*H\*W`
7. dHidden (int): Dim of intermediate FCs.
8. dZ (int): Dim of z.
10. Inputs:
11. x (torch.FloatTensor): [N, 1, 28, 28]. Input images.
13. Outputs:
14. torch.FloatTensor: [N, dZ]. Latent variable z.
15. """
16. def \_\_init\_\_(self, dIn: int, dHidden: int, dZ: int):
17. super().\_\_init\_\_()
18. *# [fc -> relu -> bn] \* 2 -> fc*
19. self.\_net = nn.Sequential(
20. *# Convert [N, C, H, W] to [N, C\*H\*W]*
21. nn.Flatten(),
22. nn.Linear(dIn, dHidden),
23. nn.ReLU(),
24. nn.BatchNorm1d(dHidden),
25. nn.Linear(dHidden, dHidden),
26. nn.ReLU(),
27. nn.BatchNorm1d(dHidden),
28. nn.Linear(dHidden, dZ)
29. )
31. def forward(self, x):
32. return self.\_net(x)
33. *# Encoder*
34. class Decoder(nn.Module):
35. """ Vanilla decoder with 3 fc layers
36. Args:
37. dZ (int): Dim of z.
38. dHidden (int): Dim of intermediate FCs.
39. hw (int): Height/width of x.
40. channel (int): Channel num of x.
42. Inputs:
43. z (torch.FloatTensor): [N, dZ]. Input latent variable z.
45. Outputs:
46. torch.FloatTensor: [N, channel, hw, hw]. xHat, restored x.
47. """
48. def \_\_init\_\_(self, dZ: int, dHidden: int, hw: int, channel: int):
49. super().\_\_init\_\_()
50. *# [fc -> relu -> bn] \* 2 -> fc*
51. self.\_net = nn.Sequential(
52. nn.Linear(dZ, dHidden),
53. nn.ReLU(),
54. nn.BatchNorm1d(dHidden),
55. nn.Linear(dHidden, dHidden),
56. nn.ReLU(),
57. nn.BatchNorm1d(dHidden),
58. nn.Linear(dHidden, hw \* hw),
59. *# Convert [N, C\*H\*W] to [N, C, H, W]*
60. nn.Unflatten(-1, (channel, hw, hw)),
61. *# Map output to [0, 1]*
62. nn.Sigmoid()
63. )
65. def forward(self, z):
66. return self.\_net(z)
67. *### 卷积AutoEncoder*
68. *# Convolutional Encoder*
69. class ConvEncoder(nn.Module):
70. """ Convolutional encoder with 3 conv layers
71. Args:
72. cIn (int): Channel of x.
73. cHidden (int): Channel of intermediate Convs.
74. cZ (int): Channel of z.
76. Inputs:
77. x (torch.FloatTensor): [N, 1, 28, 28]. Input x.
79. Outputs:
80. torch.FloatTensor: [N, cZ, 7, 7]. Latent variable z.
81. """
82. def \_\_init\_\_(self, cIn: int, cHidden: int, cZ: int):
83. super().\_\_init\_\_()
85. self.\_net = nn.Sequential(
86. *###### PUT YOUR CODE HERE ######*
87. nn.Conv2d(cIn, cHidden, 5, 1, 2),
88. nn.LeakyReLU(),
89. nn.BatchNorm2d(cHidden),
90. nn.Conv2d(cHidden, 2\*cHidden, 3, 1, 1),
91. nn.LeakyReLU(),
92. nn.BatchNorm2d(2\*cHidden),
93. nn.Conv2d(2\*cHidden, 2\*cHidden, 3, 2, 1),
94. nn.LeakyReLU(),
95. nn.BatchNorm2d(2\*cHidden),
96. nn.Conv2d(2\*cHidden, cHidden, 3, 1, 1),
97. nn.LeakyReLU(),
98. nn.BatchNorm2d(cHidden),
100. nn.Conv2d(cHidden, cZ, 3, 2, 1),
101. )
103. def forward(self, x):
104. return self.\_net(x)
105. *# Convolutional Decoder*
106. class ConvDecoder(nn.Module):
107. """ Convolutional decoder with 3 conv layers
108. Args:
109. cZ (int): Channel of z.
110. cHidden (int): Channel of intermediate Convs.
111. cX (int): Channel of xHat.
112. Inputs:
113. x (torch.FloatTensor): [N, cZ, 28, 28]. Input latent variable z.
115. Outputs:
116. torch.FloatTensor: [N, cX, 28, 28]. Outpus xHat.
117. """
118. def \_\_init\_\_(self, cZ: int, cHidden: int, cX: int):
119. super().\_\_init\_\_()
120. self.\_net = nn.Sequential(
121. *###### PUT YOUR CODE HERE ######*
122. nn.ConvTranspose2d(cZ, cHidden, 3, 1, 1, 0),
123. nn.LeakyReLU(),
124. nn.BatchNorm2d(cHidden),
125. nn.ConvTranspose2d(cHidden,2\*cHidden,3,1,1,0),
126. nn.LeakyReLU(),
127. nn.BatchNorm2d(2\*cHidden),
128. nn.ConvTranspose2d(2\*cHidden,2\*cHidden,3,2,1,1),
129. nn.LeakyReLU(),
130. nn.BatchNorm2d(2\*cHidden),
131. nn.ConvTranspose2d(2\*cHidden,cHidden,3,1,1, 0),
132. nn.LeakyReLU(),
133. nn.BatchNorm2d(cHidden),
134. nn.ConvTranspose2d(cHidden, cX, 3, 2, 1, 1),
135. nn.Sigmoid()
136. )
138. def forward(self, x):
139. return self.\_net(x)
140. *### GAN*
141. class Generator(nn.Module):
142. """ GAN generator
143. Args:
144. k (int): Number of samples from the Gaussian distribution.
145. cHidden (int): Channel of intermediate Convs.
146. cOut (int): Channel of xHat.
148. Inputs:
149. n (int): Batch size.
151. Outputs:
152. torch.FloatTensor: [n, cX, 28, 28]. xHat.
153. """
154. def \_\_init\_\_(self, k: int, cHidden: int, cX: int):
155. super().\_\_init\_\_()
156. self.\_k = k
157. self.\_net = nn.Sequential(\*[
158. *###### PUT YOUR CODE HERE ######*
159. nn.Linear(k, 4 \* cHidden),
160. nn.Unflatten(-1, (cHidden, 2, 2)),
161. nn.LeakyReLU(),
162. nn.BatchNorm2d(cHidden),
163. nn.ConvTranspose2d(cHidden,cHidden, 4, 1, 1, 0),
164. nn.LeakyReLU(),
165. nn.BatchNorm2d(cHidden),
166. nn.ConvTranspose2d(cHidden,cHidden, 4, 2, 1, 1),
167. nn.LeakyReLU(),
168. nn.BatchNorm2d(cHidden),
169. nn.ConvTranspose2d(cHidden,cHidden, 3, 2, 1, 1),
170. nn.LeakyReLU(),
171. nn.BatchNorm2d(cHidden),
173. nn.ConvTranspose2d(cHidden,cX, 3, 2, 1, 1),
174. nn.Sigmoid()
175. ])
177. def forward(self, n):
178. z = Normal(0.0, 1.0).sample((n, self.\_k))
179. return self.\_net(z)
180. class Discriminator(nn.Module):
181. """ GAN discriminator
182. Args:
183. cIn (int): Channel of x.
184. cHidden (int): Channel of intermediate Convs.
185. cOut (int): Channel of xHat.
187. Inputs:
188. x (torch.FloatTensor): [N, cIn, 28, 28]. Input x.
190. Outputs:
191. torch.FloatTensor: [n, 1]. Probability of x is a real image or not.
192. """
193. def \_\_init\_\_(self, cIn: int, cHidden: int):
194. super().\_\_init\_\_()
195. self.\_net = nn.Sequential(\*[
196. *###### PUT YOUR CODE HERE ######*
197. nn.Conv2d(cIn, cHidden, 3, 2, 1),
198. nn.LeakyReLU(),
199. nn.Conv2d(cHidden, cHidden, 3, 2, 1),
200. nn.LeakyReLU(),
201. nn.Conv2d(cHidden, cHidden, 3, 2, 1),
202. nn.LeakyReLU(),
203. nn.Conv2d(cHidden, cHidden, 3, 2, 1),
204. nn.LeakyReLU(),
206. nn.Flatten(),
207. nn.Linear(4 \* cHidden, 1),
208. nn.Sigmoid()
209. ])
211. def forward(self, x):
212. return self.\_net(x)

3. 训练执行流程介绍

AutoEncoder：每次训练从DataLoader中取一个 batch，计算训练集上的损失，每100次进行一次可视化操作，可视化操作为展示从训练集中取出的4张图片，展示通过这 4 张图片AutoEncoder生成的图片，绘制 loss曲线图。训练完毕后打印训练的总次数和最终的loss。

VAE：每100次迭代，会展示原始图像和解码后的图像，以及绘制损失曲线图以进行可视化。损失函数包括重构损失和KL散度损失。最后打印总训练次数和最终损失。

GAN：迭代进行GAN的训练，其中生成器和判别器交替训练。每100次迭代，会展示生成的图像和损失曲线图以进行可视化。最后打印总训练次数和最终损失。

具体的训练细节将在实验数据及结果分析中展开。

八、实验数据及结果分析：

1. 实验数据

MNIST数据集一共有70000张图片，其中60000张是训练集，10000张是测试集。每张图片是2828的0-9的手写数字图片组成。每个图片是黑底白字的形式，黑底用0表示，白字用0-1之间的浮点数表示，越接近1，颜色越白。如图 9所示。

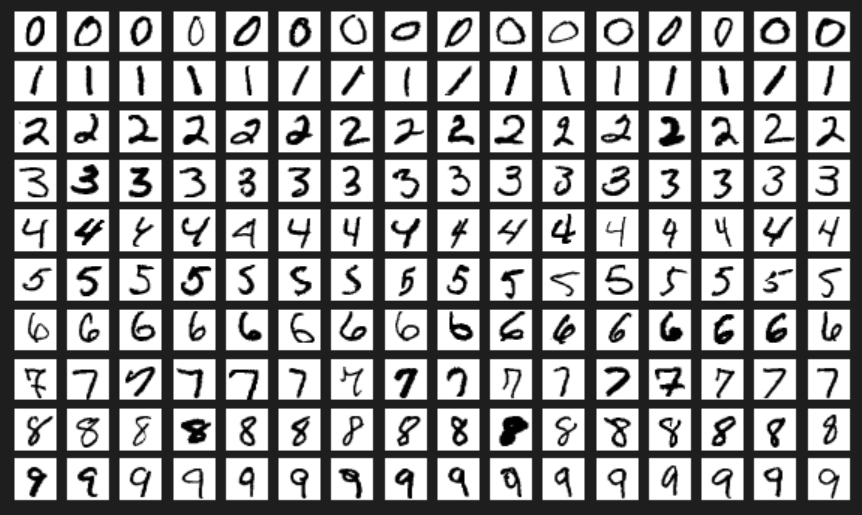


图 9 MNIST数据集

CIFAR-10 是一个包含60000张图片的数据集。如图 10 CIFAR10数据集，其中每张照片为32\*32的彩色照片，每个像素点包括RGB三个数值，数值范围 0 ~ 255。所有照片分属10个不同的类别，分别是 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'。其中训练集有50000张图片，测试集有10000张图片。

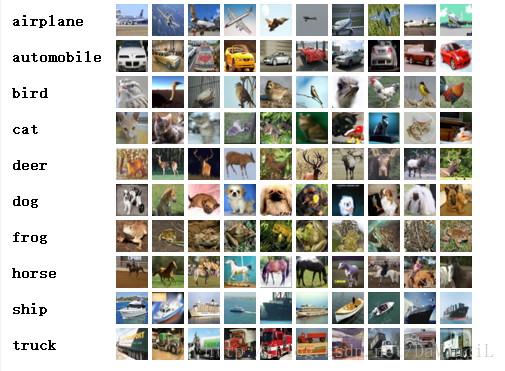


图 10 CIFAR10数据集

2. 结果分析

如图 11为使用线性层AutoEncoder生成结果和loss变化曲线。使用了Adam优化器，学习率为5e-4，一共训练了10000次，最终的loss为0.1330。

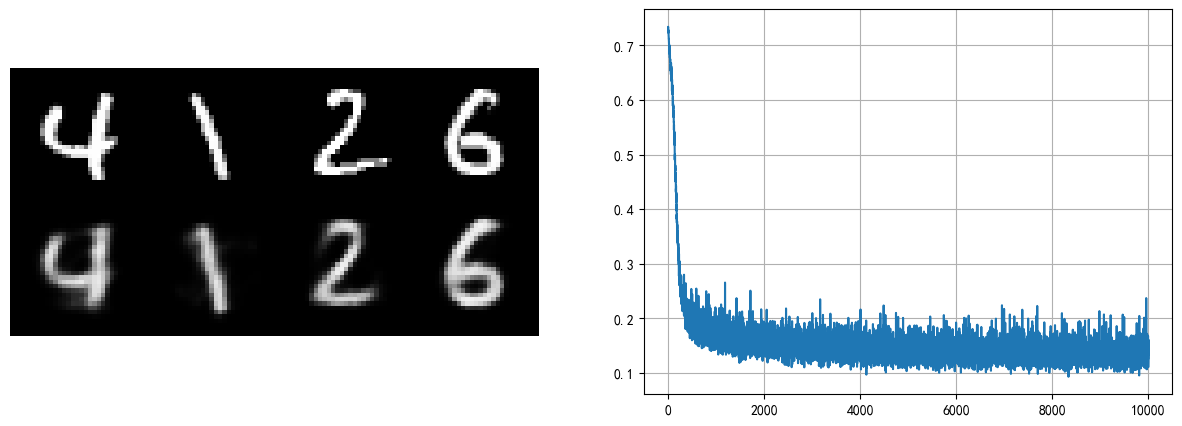


图 11 线性层AutoEncoder生成结果及loss曲线

图 12为使用卷积层AutoEncoder生成结果和loss变化曲线。使用了Adam优化器，学习率为1e-3，一共训练了1000次，最终的loss为0.0747，相比于线性层AutoEncoder，卷积层AutoEncoder生成图片效果更好，最终的loss也更低，并且从loss图中可以看出，卷积层AutoEncoder的loss下降更快，后面的波动也相对更平缓一些。

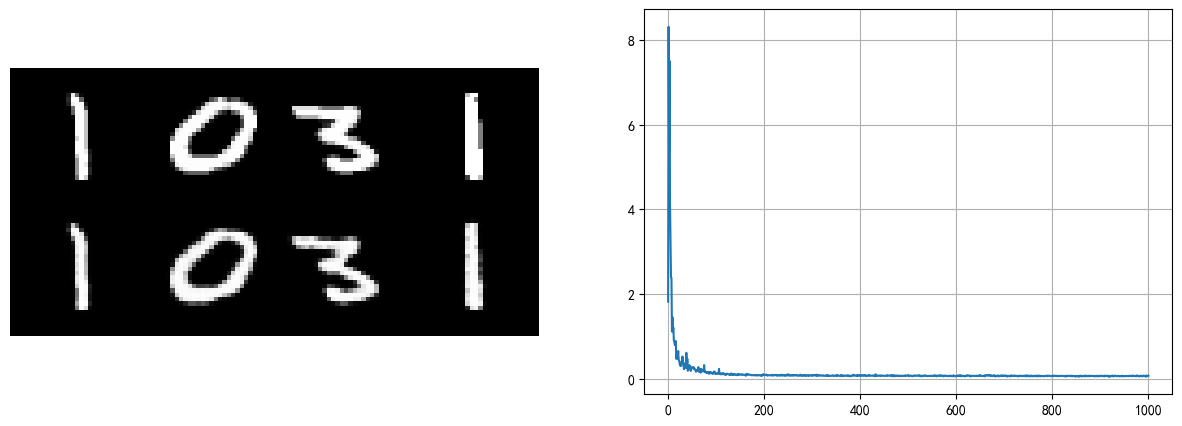


图 12 卷积层AutoEncoder生成结果及loss曲线

上面两个实验均使用了binary\_cross\_entropy作为损失函数，图 13为使用MSELoss后，卷积层AutoEncoder生成结果和loss变化曲线。仍然使用Adam优化器，学习率为1e-3，一共训练了1000次，最终的loss为0.0042。相比于使用binary\_cross\_entropy作为损失函数的AutoEncoder，使用MSELoss生成图片效果更好，最终的loss也更低。这主要AutoEncoder通过生成图片与训练图片之间的loss来更新网络参数。binary\_cross\_entropy更适合于分类任务，也就是它能更好的区分两个样本是否属于同一类。而MSELoss通过L2范数计算生成图片和训练图片之间的差异。显然，直接计算二者的差异比判断二者是否属于同一类的要求和精度会更高，因而使用MSELoss能够使网络参数更快，更好地更新，训练的效果也更好。

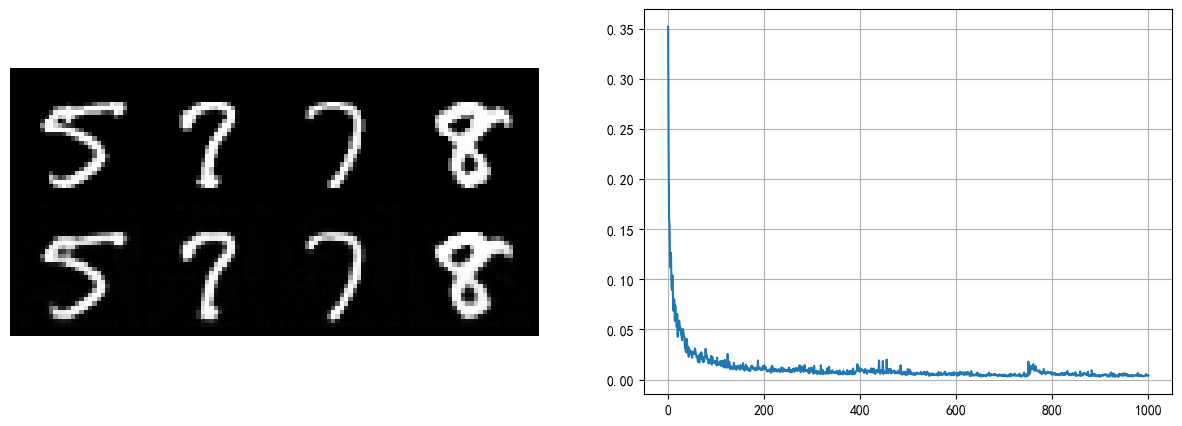


图 13 使用MSELoss卷积层AutoEncoder生成结果及loss曲线

图 14为卷积层AutoEncoder在CIFAR10生成结果及loss曲线，用Adam优化器，学习率为2e-4，一共训练了10000次，最终loss为0.0015。损失函数使用MSELoss。相比于MNIST数据集，CIFAR10的图片是3通道的，因而训练难度更大，生成的图片效果也比MNSIT数据集要差。

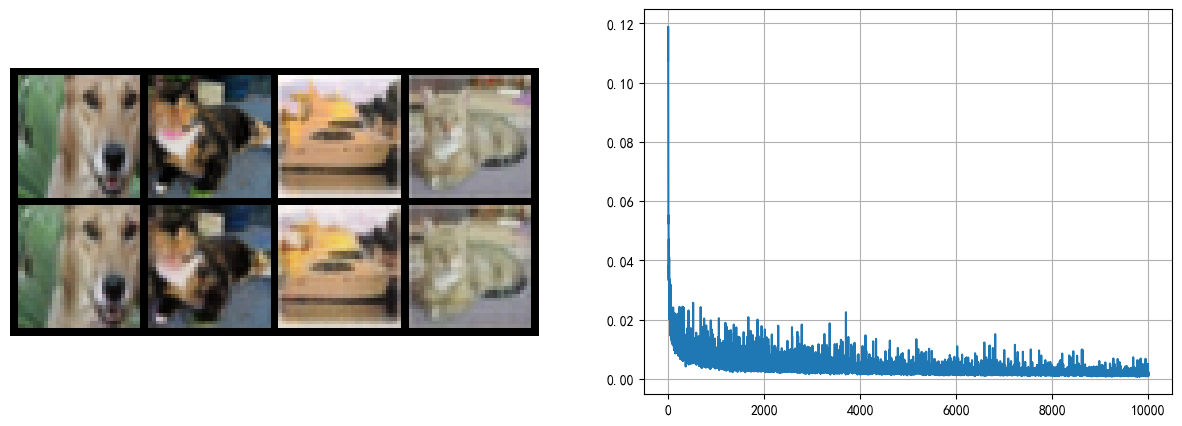


图 14 卷积层AutoEncoder在CIFAR10生成结果及loss曲线

将测试集的所有样本都输入到AutoEncoder中，对其生成的结果作t-SNE,并进行可视化，如图 15所示。a)为线性层AutoEncoder的结果，b)为卷积层AutoEncoder的结果，对比两个子图可以发现使用卷积层AutoEncoder相同类别样本间的距离更近，不同类别样本间的距离更远。这说明卷积层AutoEncoder对数据集特征提取的效果更好，与上文的结论吻合，及卷积层AutoEncoder的效果要好于线性层。

|  |  |
| --- | --- |
| a)线性层AutoEncoder | b)卷积层AutoEncoder |

图 15 AutoEncoder的t-SNE结果

图 16为AutoEncoder生成的10个类别手写数字。

a)图为线性层AutoEncoder的结果，b)图为卷积层AutoEncoder的结果，虽然卷积层生成的图片在部分内容上不太完整，但可以直观的看出，卷积层生成的图片清晰图要高于线性层。这一结果与上文结果十分吻合。

|  |  |
| --- | --- |
| a)线性层AutoEncodere | b)卷积层AutoEncoder |

图 16 AutoEncoder生成的手写数字

图 17为VAE生成结果及loss曲线，使用了Adam优化器，学习率为1e-3，一共训练了20000次。最终的loss为0.2195。可以看到VAE生成的图片质量不如AutoEncoder,最终的loss也比AutoEncoder要大。这是因为VAE的Encoder将图片编码为k个均值和k个方差。VAE假设数据服从正态分布，因而将k个正态分布输入到VAE的Decoder中，正态分布的均值和方差对应Encoder的编码结果。相比与直接作反运算的AutoEncoder，VAE的训练难度更大，因而其结果也比AutoEncoder要差。

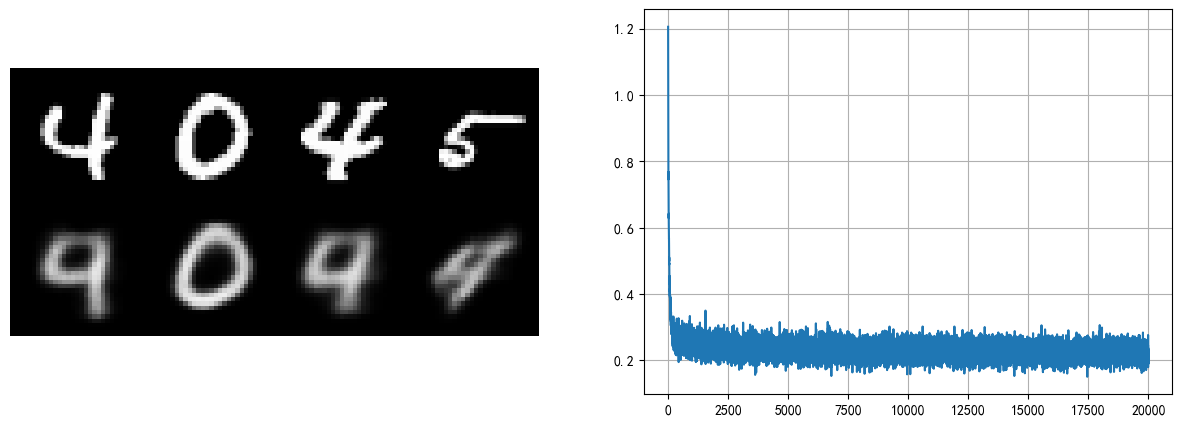


图 17 VAE生成结果及loss曲线

既然Decoder的输入通过采样得来，同时我们已经促使我们的输入所处的分布为‘K’个标准正态分布的混合。因此，我们直接从标准正态分布中随机采样 ‘K’个值，就能当作编码 输入解码器了。图 18为VAE生成结果，此处取K=10。可以看到生成的10个样本基本可以辨识出具体数字，训练出来的VAE效果较优。



图 18 VAE生成结果

图 19为GAN生成结果及loss曲线。正中间的图为判别器（discriminator）的loss变化曲线，右图为生成器（generator）的loss变化曲线。两个loss都有较大幅度的振荡，这正是生成器和判别器“左右互博”的结果。最理想的结果为生成器的结果让判别器无法辨别，即生成器生成的图片，判别器有0.5的概率预测为真实的图片，有0.5的概率预测为假的图片。生成器生成了图片“1714”，较清晰，较真实，效果较好，较难辨别真伪。不过，随着训练次数的增加，生成器生成的种类明显减少，且更频繁生成构成较为简单的数字（如1,7等），这大概是训练的时候loss陷入了局部极小点，生成器发现生成1,7这样简单的数字能更好地逃过判别器的辨识。

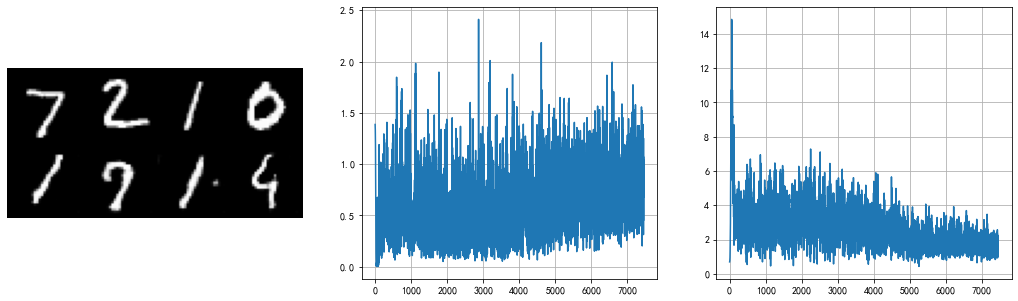


图 19 GAN生成结果及loss曲线

# 九、总结及心得体会：

通过本次实验，我对自编码器（AutoEncoder），变分自编码器（VAE），生成对抗网络（GAN）有了深入的了解。①AutoEncode是一种从无标签训练数据中学习低维特征表示的无监督方法。选择低维特征是因为AutoEncoder希望学习到能够捕捉数据中有意义的变化因子。AutoEncoder由编码器（Encoder）和解码器（Decoder）组成。Encoder将任意输入数据通过非线性映射到低维特征向量，而Decoder通过另一个非线性映射尽可能还原输入数据。AutoEncoder编码的隐变量能够很好的反映原始数据分布，它能够对原始数据进行压缩，并很好地还原。通过附加额外约束，它能够生成“假”数据。②VAE也是一种常见的生成模型，它能够学习一个函数或者模型，使得输出数据的分布尽可能的逼近原始数据分布，其基本思路是：把一堆真实样本通过编码器网络变换成一个理想的数据分布，然后这个数据分布再传递给一个解码器网络，得到一堆生成样本，生成样本与真实样本足够接近的话，就训练出了一个VAE模型。相比于AutoEncoder，VAE不再把分布映射到固定值或变量上，而是映射到一个分布上。③生成对抗网络（GAN）是一类在无监督学习中使用的神经网络，其有助于解决按文本生成图像、提高图片分辨率、药物匹配、检索特定模式的图片等任务。GAN中存在着两个网络，一个是判别网路D,一个是生成网路G。生成网络输出的是“看起来很逼真”的虚假数据，判别网络实现一个从复杂结构数据到简单判别结果的映射。训练过程中二者相互博弈，一起进步，最后G可以生成和真实图片难以区别的虚假图片，由于生成网络G并没有原图片数据，它是从一些随机噪声中生成判别器D看起来真实的图片，因而可以用GAN来合成一些不存在的图片，或者生成一些伪样本来以假乱真。

通过本次实验，我对python基本语法更加熟悉，对pytorch框架更加熟悉。在实验一的基础上，我能够进一步较熟练地使用pytorch搭建Encoder,Decoder,VAE,GAN等模型。在实验中有时会出现不能求梯度这样的报错，这是因为使用了中间变量，而中间变量初始化时，默认不需要梯度，此时只需要.requires\_grad\_(True)操作就可以正常进行反向传播算法了。

# 十、对本实验过程及方法、手段的改进建议及展望：

无

**报告评分：**

**指导教师签字：**