基于 MNIST 数据集的生成对抗网络性能优化：架构调整的研究

1. 研究背景和介绍

1.1 关于 GAN 的简单介绍

生成对抗网络（GAN）是一种深度学习架构，由 Ian Goodfellow 等人于 2014 年提出。GAN 由两个主要部分组成：生成器（Generator）和判别器（Discriminator）。这两部分在训练过程中相互对抗，以提高模型的性能。

• 生成器（Generator）：它负责生成看起来与真实数据相似的假数据样本。生成器接收随机噪声或潜在空间的向量作为输入，并将其转换为与训练数据相似的输出数据。

• 判别器（Discriminator）：它负责区分生成器生成的假数据和来自真实数据集的真实数据。判别器尝试将生成器生成的数据与真实数据区分开来。

GAN 的基本原理是通过生成器和判别器之间的对抗来促使两者不断提升。生成器和判别器在训练过程中相互竞争，这种对抗性训练会导致生成器生成更逼真的数据样本，同时判别器更难区分真假样本。

训练过程如下：

1）生成器接收随机噪声并生成假数据，这些假数据会被传递给判别器。
2）判别器接收真实数据集中的真实样本和来自生成器的假样本，尝试对它们进行分类，区分真实数据和假数据。
3）生成器根据判别器的反馈来优化生成的样本，让生成器生成的假数据更接近于真实数据，以尽可能地欺骗判别器。
4）判别器同时优化自身，以更准确地区分真假数据。

这个过程是一个动态平衡，生成器试图生成逼真的数据以骗过判别器，而判别器则试图更好地区分真实数据和生成器生成的假数据。最终，希望生成器能够生成非常逼真的数据，以至于判别器无法区分生成的数据是否真实，即生成器生成的数据与真实数据难以区分。

1.2 实验目的

本次实验主要是对生成对抗网络中的生成器和判别器进行修改，使生成器生成的数据更难与真实数据区分开来。同时进行多次调参训练，以期改进整个模型在 MNIST 数据集上的表现。

2. 数据集

数据来源：描述使用的数据集，包括数据集的来源、规模和特点。

本次实验所采用的数据集是 MNIST 数据集。

MNIST 数据集是一个广泛使用的手写数字数据集，用于训练各种图像处理系统。它由来自美国国家标准与技术研究所（NIST）的手写数字样本组成。MNIST 数据集包含来自 250 个不同人手写的数字样本，涵盖了 0 到 9 的数字。

数据集规模：

- 训练集包含 60,000 个样本
- 测试集包含 10,000 个样本
- 每个样本都是一个 28x28 像素的灰度图像，表示单个手写数字（0 到 9 之间的数字）。这些图像已经进行了预处理和标准化，每个像素值都在 0 到 255 之间，并已缩放到范围[0, 1]内。

MNIST 数据集的特点：

- 用于简单数字识别任务的标准基准数据集。
- 每个图像都是灰度图，较小且易于处理，非常适合用于训练和测试图像分类模型。
- 数据集的图像质量较高，样本之间的差异性较小，使得模型能够集中于数字识别本身。

3. 模型架构

生成器（Generator）：

模型架构

```
#全连接层，将 100 维的输入向量转换成一个形状为 7x7x256 的张量
self.layer1 = keras.layers.Dense(7*7*256, use_bias=False)
#批量归一化层，用于加速模型训练并提高稳定性
self.layer2 = keras.layers.BatchNormalization()
#LeakyReLU 激活层，修正线性单元，用于引入非线性
self.layer3 = keras.layers.LeakyReLU()
#重塑层，将输出的向量形状重塑为 7x7x256 的张量
self.layer4 = keras.layers.Reshape((7, 7, 256))
#转置卷积层，输出 128 个特征图，使用 5x5 的卷积核进行卷积操作，步幅为 1，padding 为'same'，不使用偏置
self.layer5 = keras.layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False)
self.layer6 = keras.layers.BatchNormalization()
self.layer7 = keras.layers.LeakyReLU()
#转置卷积层，输出 64 个特征图，使用 5x5 的卷积核进行卷积操作，步幅为 2，padding 为'same'，不使用偏置。
self.layer8 = keras.layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False)
self.layer9 = keras.layers.BatchNormalization()
self.layer10 = keras.layers.LeakyReLU()
#转置卷积层，输出 1 个特征图，使用 5x5 的卷积核进行卷积操作，步幅为 2，padding 为'same'，不使用偏置，最后一层使用 tanh 激活函数
self.layer11 = keras.layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same',
```

use_bias=False, activation='tanh')

原始模型通过多个转置卷积层逐渐将输入向量映射为最终的图像输出。全连接层负责将输入的随机向量映射到一个高维空间，接着经过批量归一化和 LeakyReLU 激活层进行特征提取和非线性处理，最后通过转置卷积层逐渐将特征图还原为生成的图像。

修改模型

• 添加一个的卷积层、一个批量归一化层和一个激活层。（添加在第二个转置卷积层后面）

```
self.layer8 = keras.layers.Conv2DTranspose(128, (5, 5), strides=(2, 2), padding='same', use_bias=False)
self.layer9 = keras.layers.BatchNormalization()
self.layer10 = keras.layers.LeakyReLU()
```

增加的卷积层，输出 128 个特征图，使用 5x5 的卷积核进行卷积操作，步幅为 2，padding 为'same'，不使用偏置。批量归一化层和 LeakyReLU 激活层和原始模型里的一样，没有做改变。

增加的卷积层有助于捕获更多数据特征，提高生成图像的质量和多样性。批量归一化层和 LeakyReLU 激活层的增加有助于加速训练过程并稳定模型训练。

判别器（Discriminator）：

原始模型架构

```
#卷积层,输出 64 个特征图，使用大小为 5x5 的卷积核进行卷积操作。步幅为 2，padding 为'same'
self.layer1 = keras.layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same')
#LeakyReLU 激活层，引入非线性，允许一些负值通过以减少神经元的丢失。
self.layer2 = keras.layers.LeakyReLU()
#Dropout 层。设置为 0.3 的丢弃率，用于在训练过程中随机丢弃输入神经元，以减少过拟合。
self.layer3 = keras.layers.Dropout(0.3)
#卷积层,输出 128 个特征图，使用大小为 5x5 的卷积核进行卷积操作。步幅为 2，padding 为'same'
self.layer4 = keras.layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same')
self.layer5 = keras.layers.LeakyReLU()
self.layer6 = keras.layers.Dropout(0.3)
#展平层，用于将多维数据压扁成一维数据，为后续的全连接层做准备
self.layer7 = keras.layers.Flatten()
#全连接层，输出节点数为 1，用于进行二分类任务，输出一个值来判断输入数据是真实图像还是生成图像。
self.layer8 = keras.layers.Dense(1)
```

原始模型主要由两个卷积层（带有 LeakyReLU 激活函数和 Dropout 层）和一个全连接层组成。两个卷积层都使用了'同卷积'的方式来保持输入输出大小相同，并使用步长为(2, 2)来减小特征图的大小。LeakyReLU 激活函数用于引入非线性，而 Dropout 层有助于减少过拟合。最后的全连接层输出一个值，用于进行二分类任务，例如对输入的图像进行真假判断。

修改模型

self.layer4 = keras.layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same', kernel_regularizer=regularizers.l2(0.01))

在原始模型的第二个卷积层，增加了正则化项，在这一层的权重矩阵中的每个系数都将受到 L2 正则化的约束。这个约束将会增加在损失函数中，由 0.01 乘以权重的平方和组成。有助于防止模型过度拟合训练数据，使其更倾向于学习简单的模式，并且对大权重值进行惩罚。正则化有助于改善模型的泛化能力。

self.layer7 = keras.layers.Conv2D(256, (5, 5), strides=(2, 2), padding = 'same', kernel_regularizer = regularizers.l2(0.01))
self.layer8 = keras.layers.LeakyReLU()
self.layer9 = keras.layers.Dropout(0.3)

在展开层前添加了一个卷积层，输出 64 个特征图，使用大小为 5x5 的卷积核进行卷积操作。步幅为 2，padding 为'same'，当然这一卷积层也加了正则化，扩增了模型的复杂度，有助于捕获更多数据特征，提高判别器的性能。LeakyReLU 激活层的增加有助于避免神经元饱和问题，进而改善梯度流和模型收敛性。

完整的代码可以见附件 GAN3.ipynb

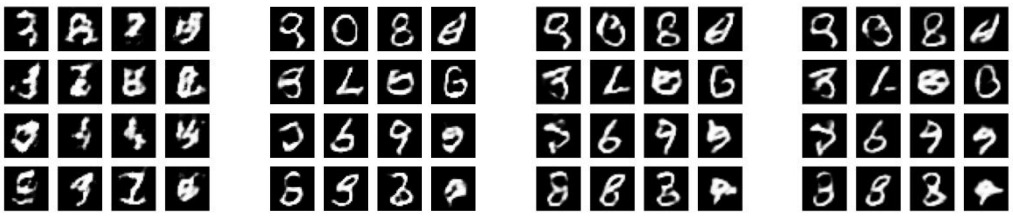
4. 实验设置

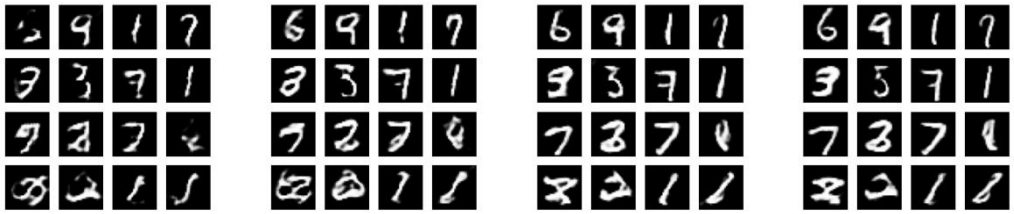
超参数设置：
学习率：1e-4
批量大小：256

训练过程：
优化器：Adam
训练迭代次数：50

5. 实验结果

经过多次的参数调整和模型的层的参数调整，得到了以下较为不错的结果，下面是原始模型和经过修改之后的模型两者在不同的 epoch 下生成的图像的对比（第一行是原始模型生成的图像，第二行是改进模型后生成的图像，分别对应 epoch=20，30，40，49）





Ps：详细的各个 epoch 的图像可以见附件 GAN 文件夹以及 GAN1 文件夹

6. 分析和讨论

模型表现

在观察生成对抗网络（GANs）的表现时，生成器和判别器的性能是重要的衡量标准。经过修改后的生成器显示出更强的图像生成能力，生成的图像在视觉上更接近真实图像。通过增加卷积层、批量归一化和激活函数，生成器更能够捕获数据特征，提高了生成图像的质量和多样性。

修改的影响

对模型进行的调整和修改对生成对抗网络的训练稳定性和图像生成的质量产生了深远的影响。增加卷积层和正则化项使模型更加复杂，但有助于防止过拟合并改善模型泛化能力。批量归一化和 LeakyReLU 激活函数的引入提升了模型的训练效率和稳定性，有助于加速收敛并提高模型的鲁棒性。这些修改使模型更具有表达能力，提高了模型在图像生成任务上的性能。

除此之外，我还观察到，似乎生成器比较擅长生成数字 6 和 9，在数字 2 和 3 和 8 的生成上则是不太优越。可能和手写数字的形状有密切关系。

7. 结论

当调整生成对抗网络（GANs）模型的架构时，我观察到在 MNIST 数据集上的性能改进。在对生成器和判别器进行多次修改的过程中，我发现增加卷积层、批量归一化层和激活层对模型的性能有着显著的积极影响。这些修改对于提高图像生成的质量和 GAN 的训练稳定性起到了关键作用。

通过引入更多的卷积层和正则化项，我们有效地增加了模型的复杂度并且减少了过拟合的风险。此外，批量归一化层和 LeakyReLU 激活函数的使用有助于加速训练过程并改善了模型的收敛性。这些调整不仅提高了模型的图像生成能力，还使其在训练过程中更加稳健。

然而，尽管我观察到了性能的提升，但在图像质量方面仍存在一些限制。在改进模型的同时，我也遇到了一些挑战，如调整超参数、模型层数的增加等，这些挑战对于提高模型性能产生了一定的影响。最主要的是本地跑不了大数据大模型，狠狠被限制了。

未来可以考虑在更大规模的数据集上进行实验，尝试不同的架构调整和超参数设置，以期望进一步提升生成对抗网络在图像生成任务上的表现。此外，还可以探索其他正则化技术、优化算法的改进以及不同的激活函数对模型性能的影响。这些工作将有助于进一步拓展生成对抗网络的应用领域，并提高其在图像生成任务中的效果和稳定性。