**《人工智能概论》实验内容与要求**

**实验1 基于A\*算法的8数码问题求解**

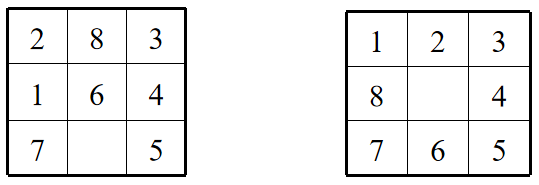
问题描述：在3\*3的棋盘中有8个数码（数字块）和一个空格，只有与空格相邻的数码能移动到空格位置。从初始状态以最小的步长移动到目标状态。

实验要求：参考A\*算法核心代码，以8数码问题为例实现A\*算法的求解程序。

实验内容：

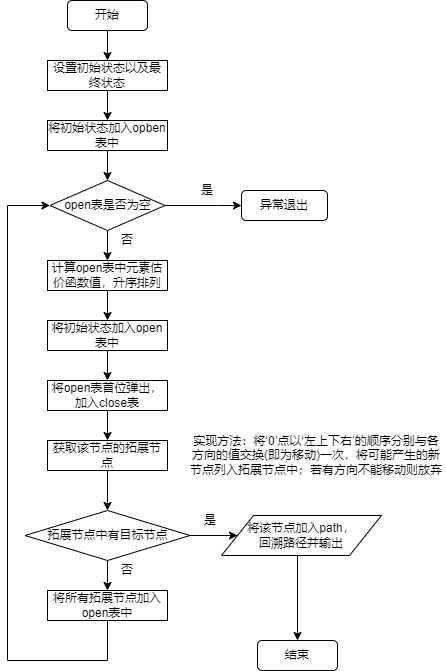
1. 基于参考代码，运行程序，并画出程序的总体流程图。自动统计并输出到达目标的总步数，自动记录并输出每次移动后open表与closed表的变化。

要求初始状态与目标状态分别为：



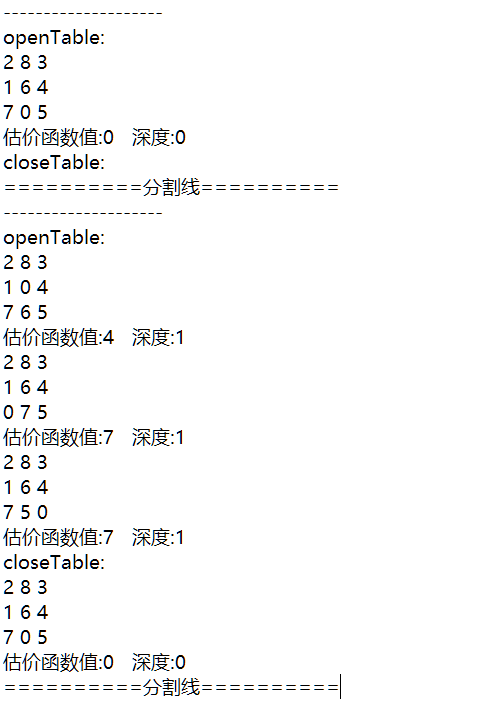
**【撰写内容】**

1. 经测试发现参考代码有错误，故首先应该修改参考代码。将深度优先搜索改为广度优先搜索，同时改为以文件形式进行输出，方便后续处理。
2. 根据1、要求，首先给出程序的总体流程图如下：

****

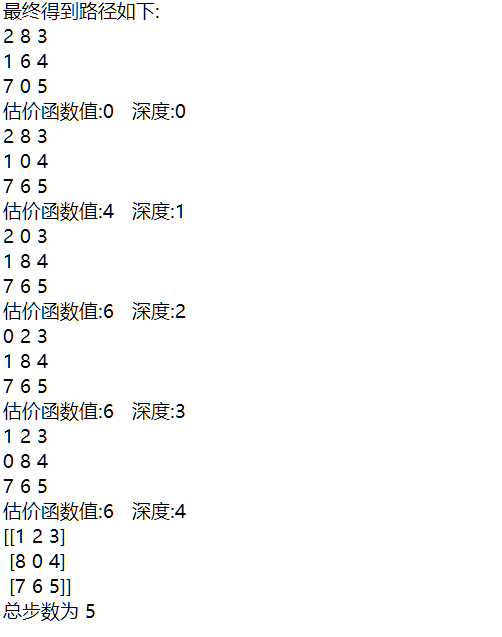
**图1 程序流程图**

1. 根据1、要求给出程序运行情况如下（结果为**文件输出**）：



**图2 程序运行结果**

用分割线区分不同步，每一步都会更新open表和close表，并在每个节点计算估价函数值与搜索深度并展示。



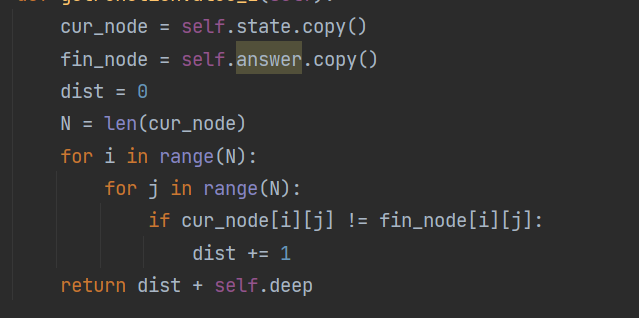
**图3 得到的结果path**

运行结束后在最后输出得到的path，并给出步数。

1. 设计两种新的启发式函数，并运行程序。对包含示例在内的3种启发式函数的特点与结果进行对比分析。

**【撰写内容】**

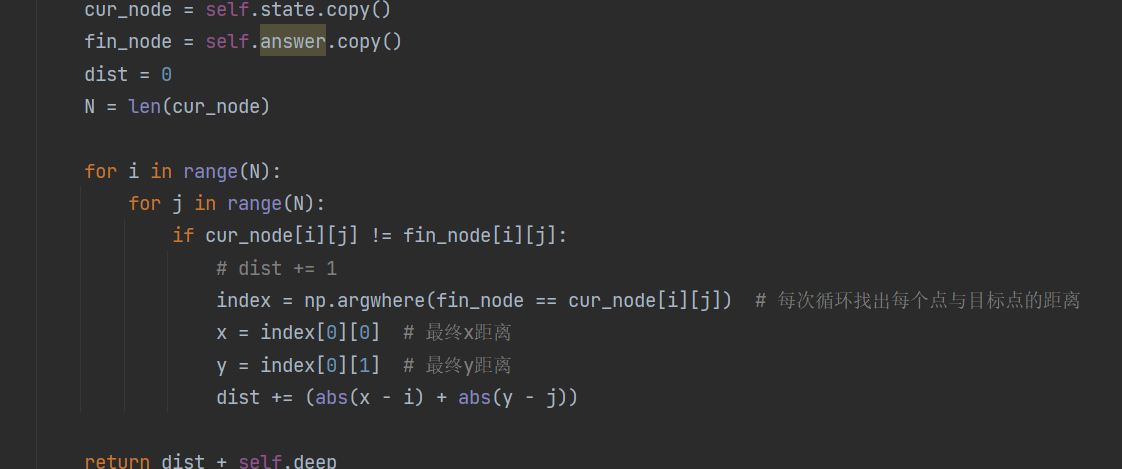
启发式函数1：取一棋局与目标棋局相比，其位置不符的数码数目



**图4 启发式函数1**

该函数产生的open表和close表相比于另外两种函数要明显多一些，即代表拓展节点更多，效率较低。

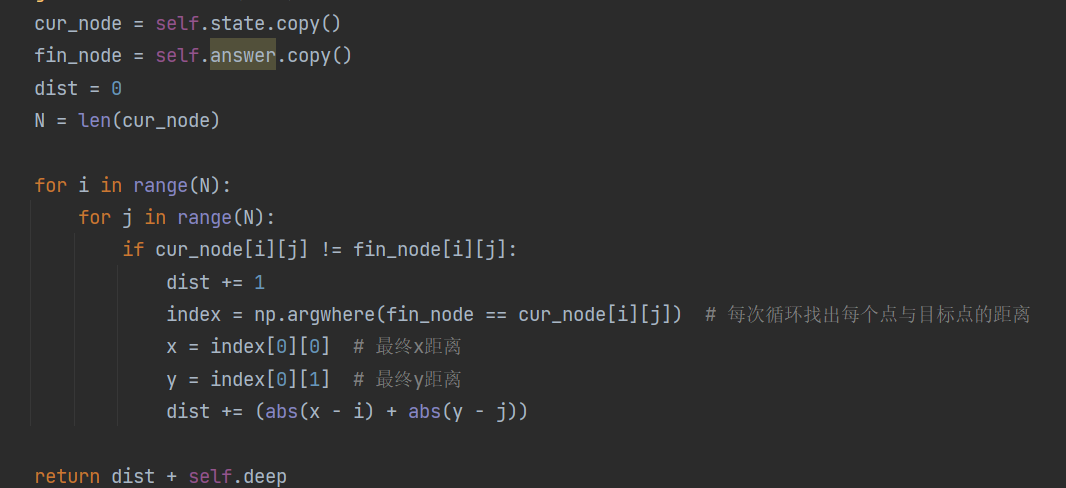
启发式函数2（示例所给出）：各数码移到目标位置所需移动的距离的总和



**图5 启发式函数2**

该函数在3种函数中拓展的节点最少，效率最高。

启发式函数3：将启发式函数1和2相加



**图6 启发式函数3**

结果与函数2相近，可能是因为函数1和2的值中1占权重小，影响不大。

对比总结：

在两个A\*启发策略的h1和h2中，如果对搜索空间中的任一状态n都有h1(n) ≤ h2(n)，就称策略h2比h1具有更多的信息性。如果某一搜索策略的h(n)越大，则A\*算法搜索的信息性越多，所搜索的状态越少。但更多的信息性需要更多的计算时间，可能抵消减少搜索空间所带来的益处。

**实验2 基于遗传算法的TSP问题求解**

问题描述：假设有一个旅行商人要拜访N个城市，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。

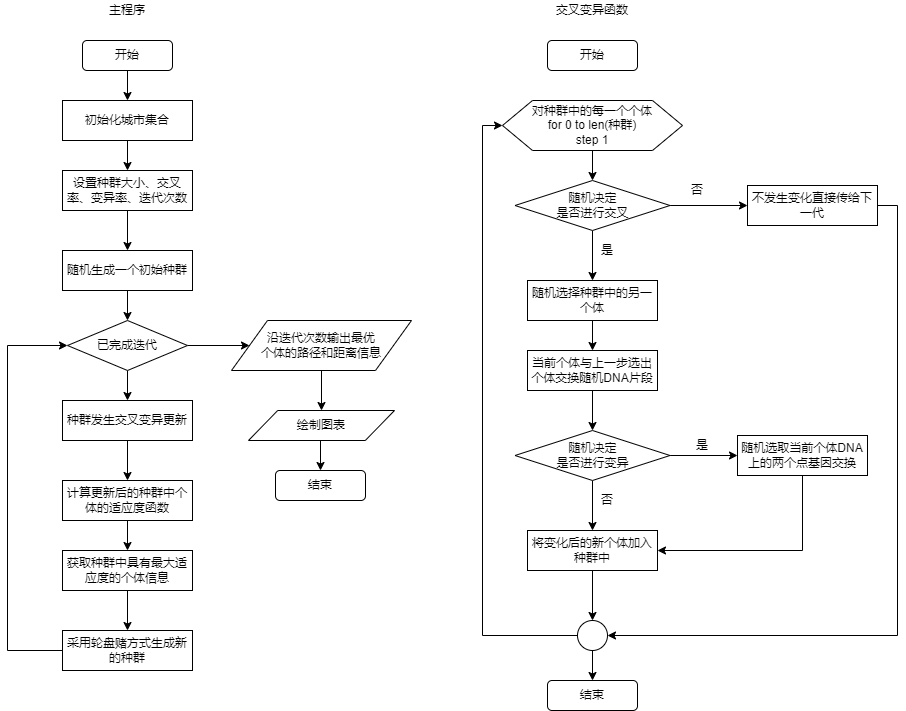
实验要求：参考遗传算法核心代码，以TSP问题为例实现遗传算法的求解程序。

实验内容：

1. 基于参考代码，运行程序，并画出程序的总体流程图。给出适应度函数的定义。城市数量设定为10个。

**【撰写内容】**

1. 根据1、要求，给出程序的总体流程图如下：

****

**图1 程序流程图**

1. 适应度函数的定义：

进化论中的适应度,是表示某一个体对环境的适应能力,也表示该个体繁殖后代的能力。遗传算法的适应度函数也叫评价函数,是用来判断群体中的个体的优劣程度的指标,它是根据所求问题的目标函数来进行评估的。

本实验中个体的适应度函数是个体所代表的城市按照一定访问顺序需要走的总距离的倒数。本实验中总距离越小越好，而适应度函数一般要求是越大越好，因此设计函数取的是总距离的倒数。最终的适应度函数也减去了种群中最小的适应度值，这可以使得最小的适应度函数的值为0，简化了计算。

1. 基于某一个固定数量（10个）与位置的城市集合，修改种群数量、交叉概率、变异概率、迭代次数等参数，对比分析算法的特性与结果。

**【撰写内容】**

设定算法初始参数：

POP\_SIZE = 200 种群大小

CROSS\_RATE = 0.6 交叉率

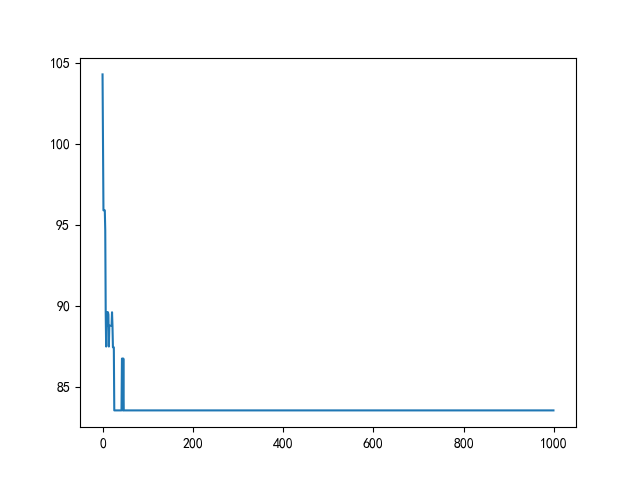
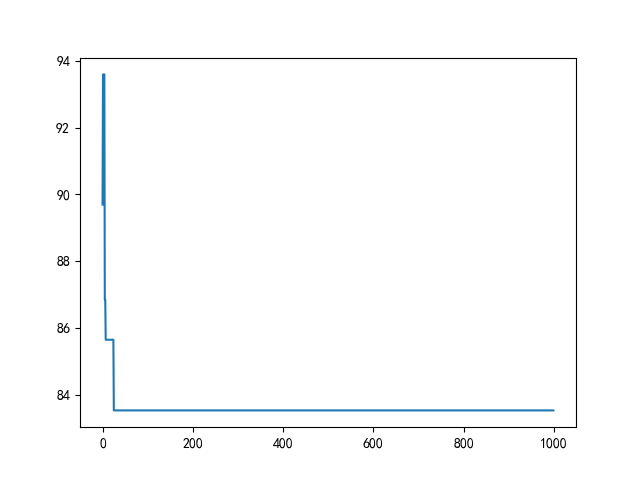
MUTA\_RATE = 0.2 变异率

Iterations = 1000 迭代次数

City\_Map=[[106.54,29.59],[91.11,29.97],[106.27,38.47],[111.65,40.82],[125.35,43.88],[114.48,38.03],[101.74,36.56],[113.6,34.76],[118.78,32.04],[117.27,31.86]] 10个城市的坐标

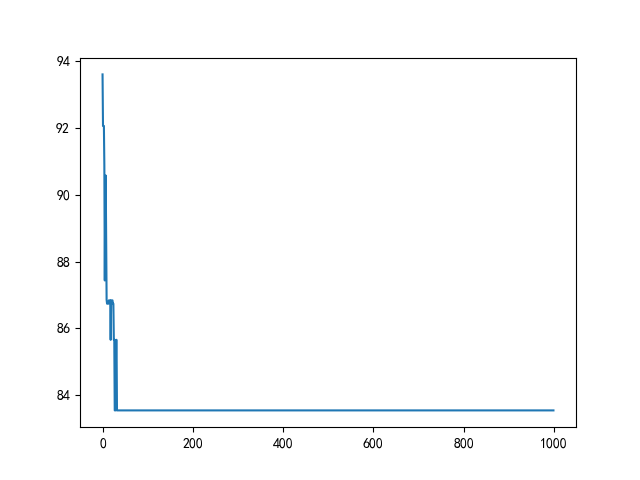
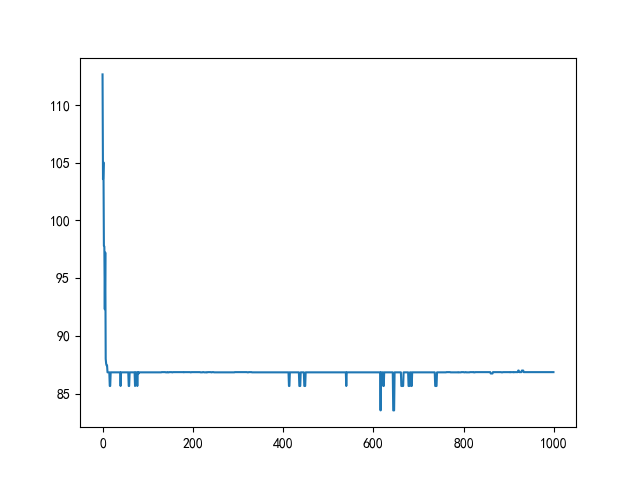
下面给出调整各参数后算法运行结果：

**注：横坐标为迭代次数，纵坐标为最佳距离**



**图2 初始状态算法运行结果**

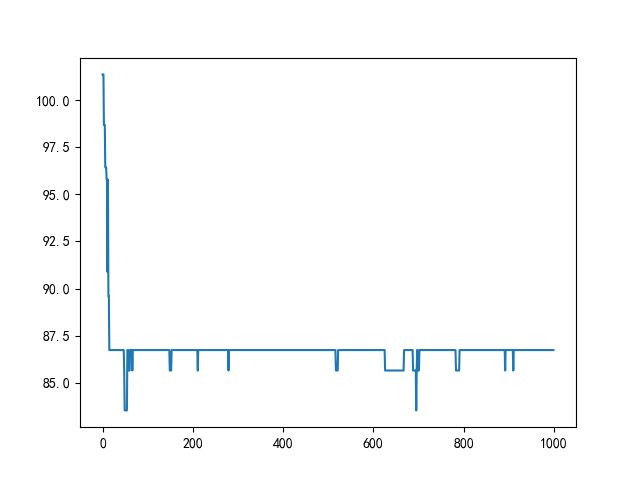
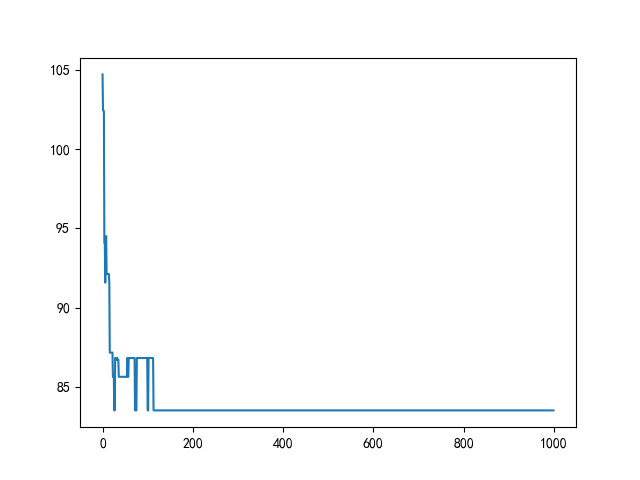
**（1）修改种群大小：**



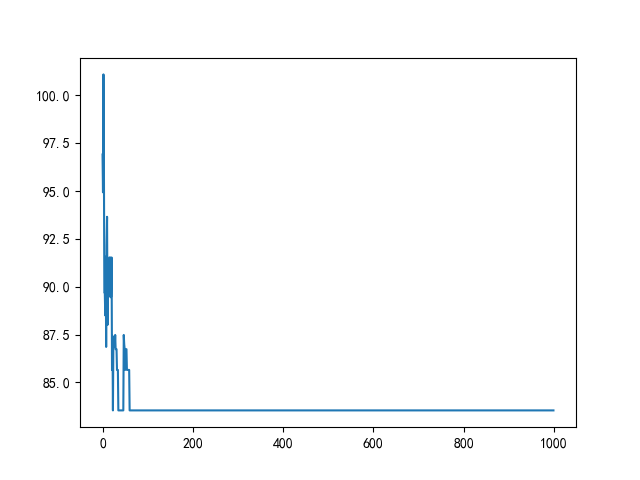
**图3 种群大小20** **图4 种群大小1000**

由结果分析，种群小时，运算和收敛速度高，但会降低群体多样性，可能会引起早熟现象，最终得到的结果可能并非最优，类比遗传学中的基因池，种群个体太少，基因型不足，导致不能很好地适应环境。在本次实验中反映为可选路径少，难以得出最优路径。种群大时，会明显导致算法运行时间代价增大，收敛速度变慢，但搜索质量提高。

**（2）修改交叉率：**



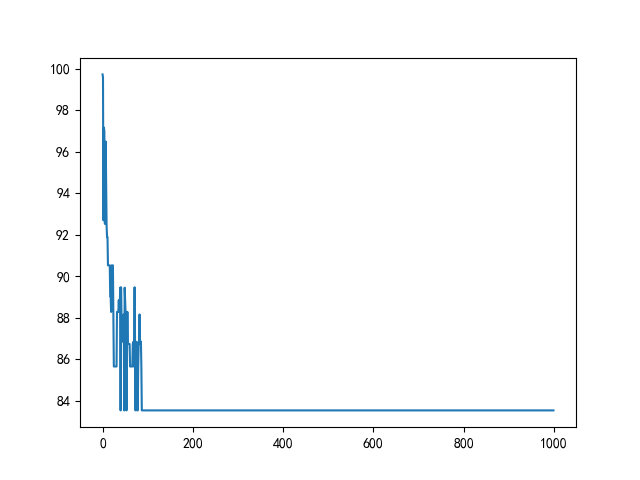
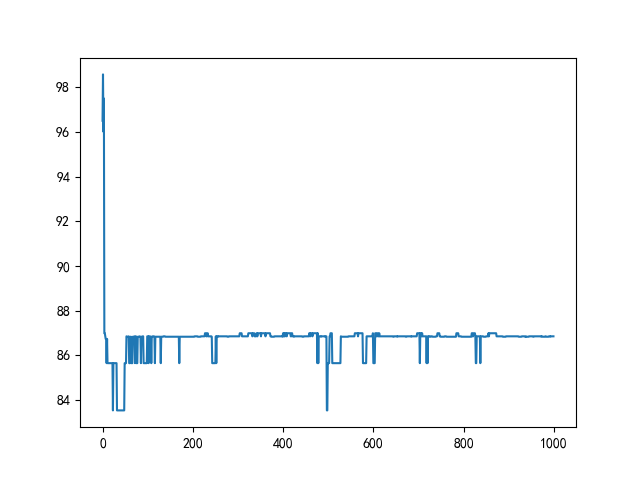
**图5 交叉率0.3 更优结果** **图6 交叉率0.3 更差结果**



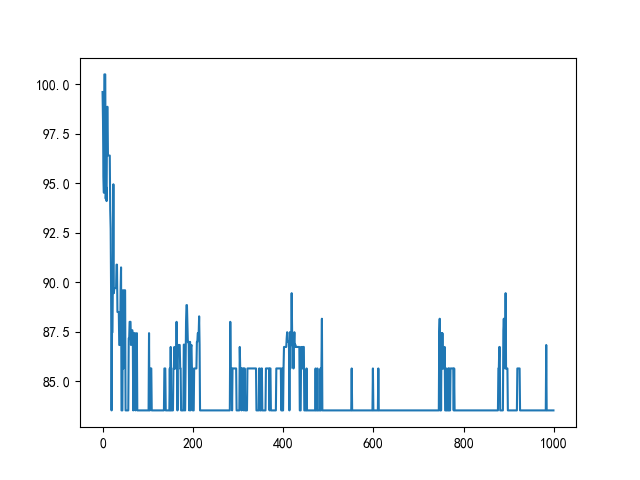
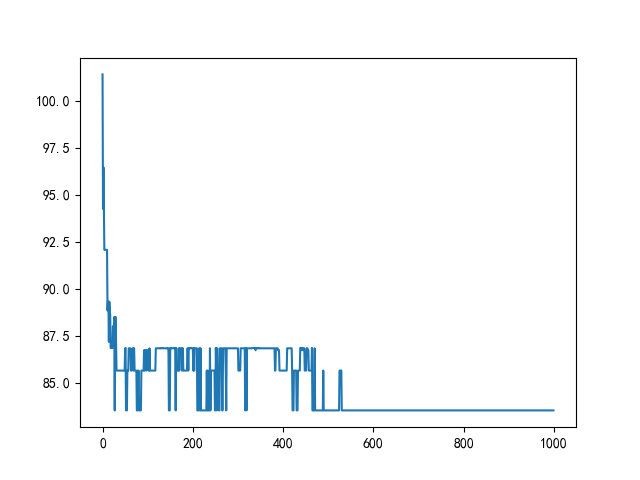
**图7 交叉率0.9**

由结果结合资料分析，如果交叉概率大，可能会导致个体基因信息过于混合，各代充分交叉，但群体中的优良模式遭到破坏的可能性增大，以致产生较大的代沟，从而使搜索走向随机化；如果交叉概率小，可能会导致个体基因信息过于单一，产生的代沟越小，就会使得更多的个体直接复制到下一代，遗传搜索可能陷入停滞状态

**（3）修改变异率：**

****

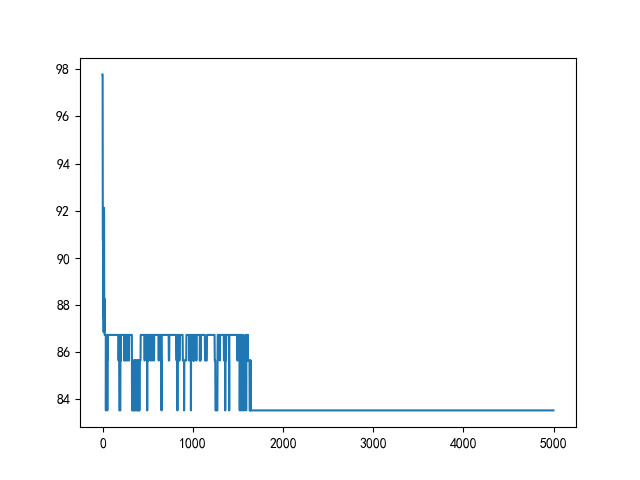
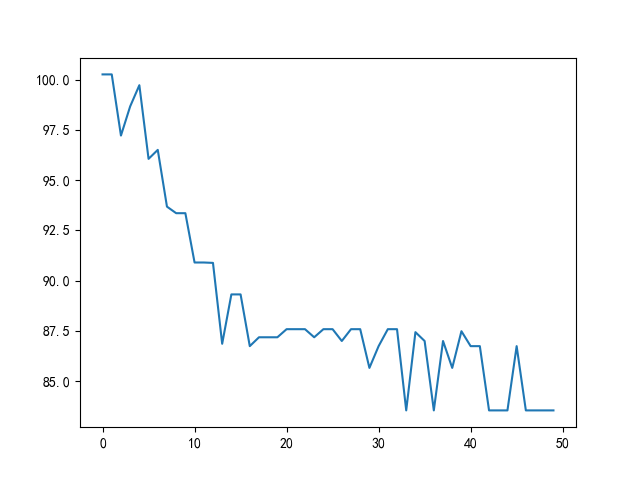
**图8 变异率0.1** **图9 变异率0.4**



**图10 变异率0.6** **图11变异率0.8**

由图分析，若变异率太低,一些有用的基因就难以进入选择，导致无法得到最优解。若太高,即随机的变化太多,那就可能失去从继承下来的好特性，在变异率超过0.5时会退化为随机搜索，失去优势。

**（4）修改迭代次数：**



**图12 迭代次数50** **图13 迭代次数5000**

由图分析，迭代次数少，算法运行速度高，但算法可能没来得及收敛；迭代次数高，算法运行速度慢，但结果收敛效果好。

**实验3 基于神经网络的图像分类**

问题描述：基于特定图像数据集，研究人工神经网络分类器的设计与实现方法。

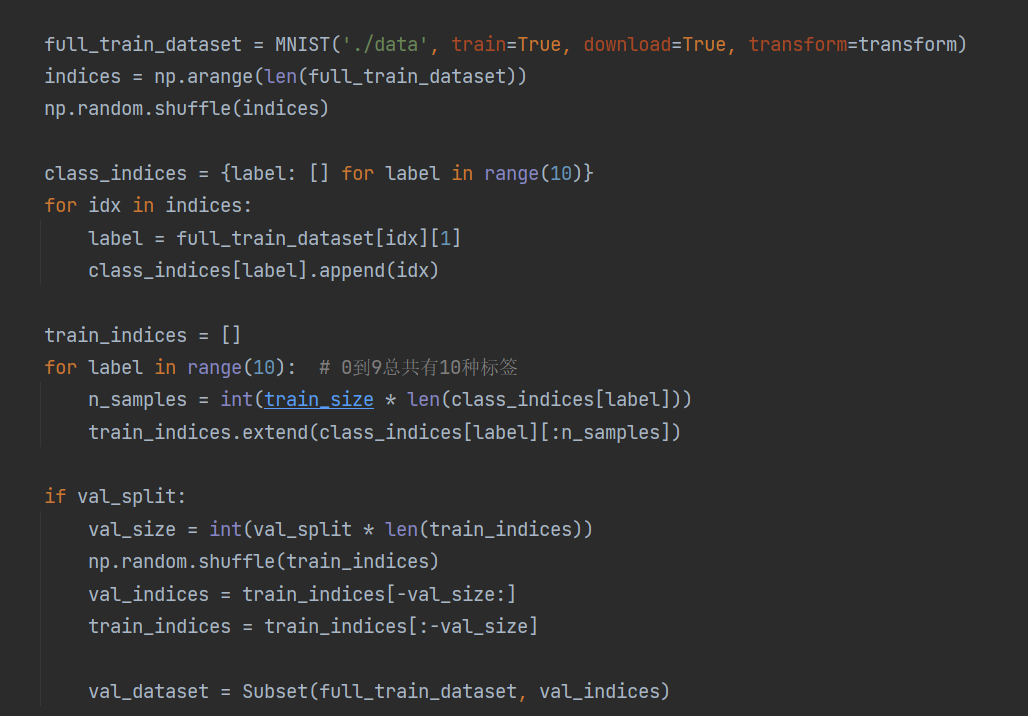
实验要求：基于MNIST手写数字图像数据集，完成神经网络模型的网络设计及参数选择，并在训练集上进行训练，在测试集上进行性能评估。了解Pytorch或TensorFlow等深度学习框架，基于上述算法库完成实验内容。

实验内容：

1. 训练集分组：从原始训练集中分别抽取10%，30%，50%，70%，100%的样本构成新的训练集，每个类别样本数量相同。可根据需要选择是否从新的训练集中抽取一定比例样本作为验证集。

**【撰写内容】**

根据要求构建data\_loader.py包含load\_mnist\_data函数读取数据并分组。使用Pytorch框架以读取MNIST数据并进行初步划分。这里我选择划分了训练集和验证集。



**图1 划分训练集和验证集**

给出一些关键内容解释：

**batch\_size (int):** 每个批次的大小，即每次迭代中返回的数据样本数。

**shuffle (bool):** 是否在每个 epoch 之前打乱数据。如果设置为 True，则在每个 epoch 之前重新排列数据集以获得更好的训练效果。

**root：**下载数据的目录；

**train**决定是否下载的是训练集；

**download**为true时会主动下载数据集到指定目录中，如果已存在则不会下载

**transform()**是接收PIL图片并返回转换后版本图片的转换函数，

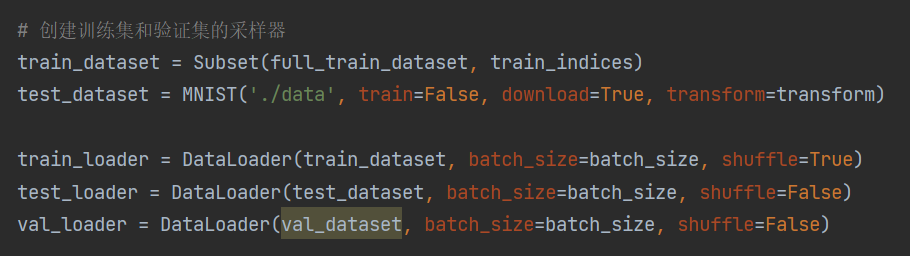
**transform()函数**是一个 PyTorch 转换操作，它将图像转换为张量并对其进行标准化，其中均值为 0.1307，标准差为 0.3081。

**torchvision.transforms**是pytorch中的图像预处理包。一般用Compose把多个步骤整合到一起

**transforms.ToTensor()**函数的作用是将原始的PILImage格式或者numpy.array格式的数据格式化为可被pytorch快速处理的张量类型。

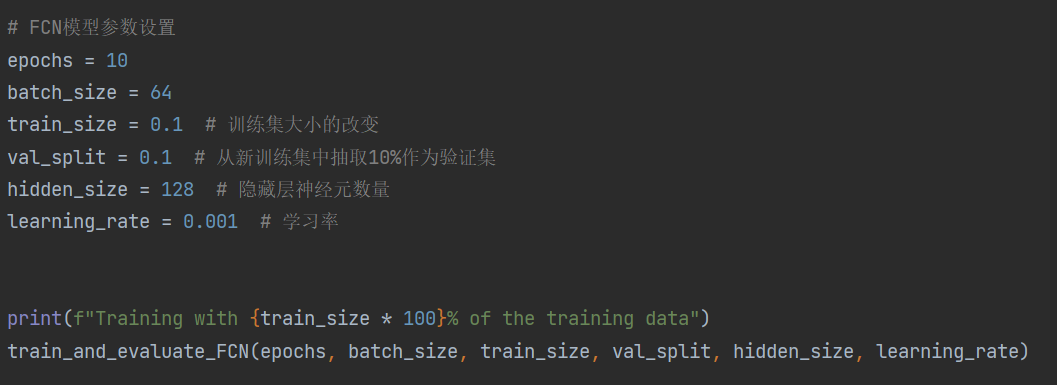
**transforms.Normalize()**用均值和标准差归一化张量图像

**归一化：**把数据变成[0,1]或者[-1,1]之间的小数。主要是为了数据处理方便提出来的，把数据映射到0～1范围之内处理，更加便捷快速。



**图2 创建训练集和验证集的采样器**

由于数据集通常很大，直接使用整个数据集进行训练和验证是不切实际的。因此，我们需要采用采样器对数据集进行抽样，以便在保留数据分布的同时，将数据集分成小批次进行处理。



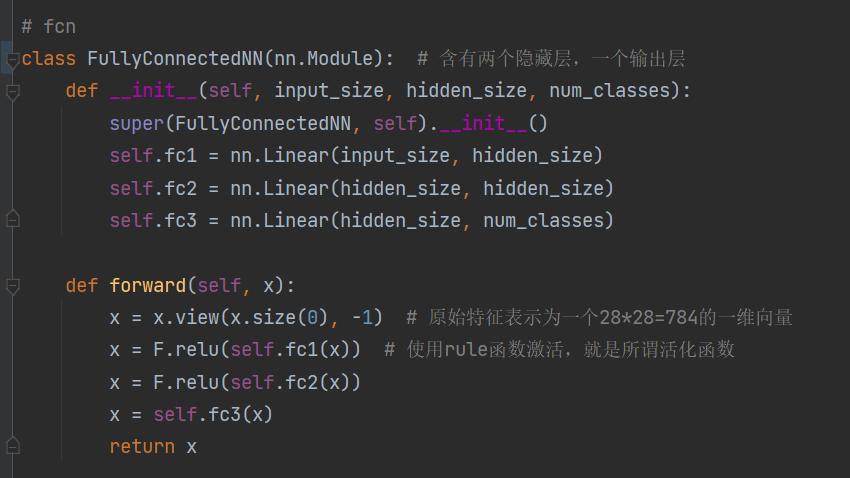
**图3 在后续参数设置中调整train\_size和val\_split**

本次实验中，我构建load\_mnist\_data函数返回train\_loader, val\_loader, test\_loader。在后续的train\_and\_evaluate\_FCN函数中会调用load\_mnist\_data函数，可以在此调整训练集大小和验证集大小等参数。

1. 建立全连接神经网络模型，隐含层数量不超过2层。给出网络结构的所有参数设置情况，包括网络层数、每层神经元数量、活化函数的选择、学习率的设定、损失函数的定义等。对网络的输入进行描述，即原始特征或其他特征的表示方式。

**【撰写内容】**

根据题目要求**建立全连接神经网络模型（Full Connect Neural Network）**。



**图 4 全连接神经网络模型**

其中fc1和fc2为隐含层，fc3为输出层。输入层的大小由input\_size参数指定，它是在模型初始化时传递给fullyconnectednn类的。可以看到，在\_\_init\_\_方法中，nn.linear()函数使用input\_size来定义第一层全连接层的输入大小，即表示输入层的大小。因此，虽然没有显式地提到“输入层”，但它仍然存在于代码中。

**网络结构参数设置**如下：

1. 网络层数：3层
2. 每层神经元数量：fc1为784，fc2为128，fc3为10。

784是因为mnist数据集中的每个图像都由28x28个像素组成，每个像素代表了一个特征。因此，将28x28个像素展平后得到的特征向量具有784个特征。第一层网络的784个神经元分别对应于784个输入特征，每个神经元接收一个输入特征。

128是经过不断调整所得。第二层网络的神经元数量由超参数 hidden\_size 决定，需要调参根据实际情况来确定。如果该值过小，模型可能会欠拟合；如果该值过大，则会增加模型的复杂度，容易出现过拟合的问题。

10是因为每个神经元对应一个数字类别。这些神经元的输出表示了神经网络预测输入图像属于每个类别的概率。本次实验中它需要将输入的手写数字图像映射到0到9的10个不同类别中的一个，所以总共有10个神经元。

1. 活化函数的选择：relu。因为它有以下优点：

非线性：由于输出值和输入值不再成线性关系，因此可以更好地拟合非线性数据。

计算快速：relu的计算速度很快，比sigmoid和tanh等传统的激活函数要快得多，使得神经网络的训练速度更快。

1. 学习率的设定：初始设为0.01。后续可以逐步提高学习率，直到模型出现过拟合或者性能不再提升为止。
2. 损失函数的定义：交叉熵损失函数。因为它有以下优点：

交叉熵损失函数可以有效地衡量预测结果的正确性，特别是对于多分类问题，它的效果非常好。在mnist数据集中，每张图片只有一个正确的数字标签，所以可以将问题看作一个多分类问题，使用交叉熵损失函数可以有效地衡量模型的准确性。

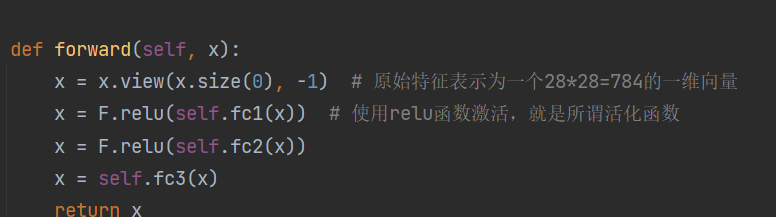
有很好的数学性质。使得优化器可以很好地进行优化。在训练全连接神经网络时，需要使用优化算法来不断调整参数，使得模型的预测结果更加准确。而交叉熵损失函数具有凸性和光滑性等良好的数学性质，这意味着优化器可以很好地寻找到最优解，并且训练过程更加稳定和高效。

比较适合梯度下降优化算法。交叉熵损失函数对于每个样本的偏导数可以用一种非常简单的方式计算，而且具有良好的凸性，这使得梯度下降算法可以很好地在学习过程中更新权重和偏置。

1. 优化器：Adam算法。它是一种基于梯度的优化算法，通过自适应地调整学习率来加速收敛。在大多数情况下，相较于传统的梯度下降法，adam算法收敛更快，并且具有更高的泛化能力。

**对网络的输入进行描述，即原始特征或其他特征的表示方式：**

mnist数据集中每张图像大小为28x28像素。网络的输入应该是将每张28x28像素的图像展开成一个784维的向量。因此，原始特征表示方式就是将每个像素点的值（0-255）作为一个特征，总共有784个特征，构成一个大小为(1, 784)的输入向量。



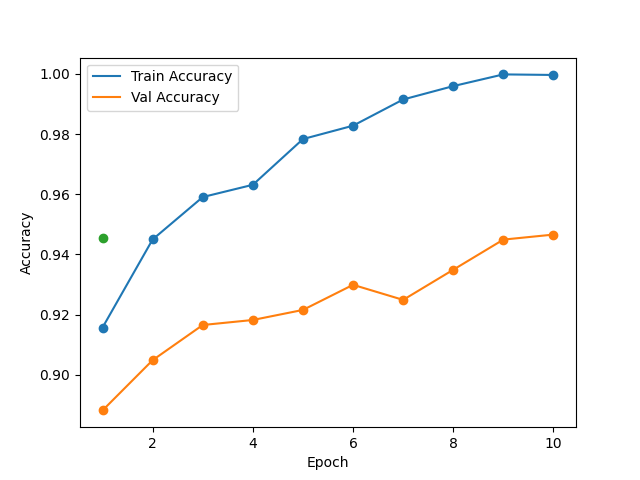
**图5 原始特征处理**

x.view(x.size(0), -1) 的作用是将输入x转化为一个二维张量，其中第一维的大小是 x.size(0)（即batch size）第二维的大小是自动计算得到的，使得转化后的张量总元素个数与原始张量相同。

1. 基于每个训练集，训练模型，记录训练集的准确率，以图表等方式进行呈现。进一步，在完整的测试集上进行性能评测，结果以图表等方式进行呈现。对实验结果进行一定的描述与分析。

**【撰写内容】**

使用的训练模型为**全连接神经网络模型**。



**图6 默认参数结果**

参数：

epochs = 10

batch\_size = 64

train\_size = 0.1 # 训练集大小

val\_split = 0.1 # 从新训练集中抽取10%作为验证集

hidden\_size1 = 256 # 隐藏层1神经元数量

hidden\_size2 = 128 # 隐藏层2神经元数量

learning\_rate = 0.001 # 学习率

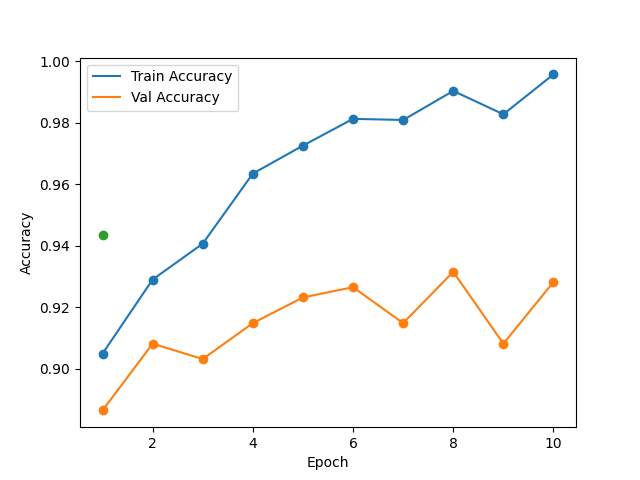
损失函数：交叉熵

优化器：Adam

结果描述与分析：

随着训练完毕的epochs数量增多，训练集训练的准确率不断提高。验证集整体也呈现上升趋势，偶有波动。绿色的点为将训练完毕的模型应用于测试集的结果，准确率相对来说处于正常范围。我认为本次结果并没有发生过拟合，因为梯度是基于局部样本计算的，具有一定的随机性。特别是在接近极限精度的情况下，轻微的震荡是有可能的。

测试集上的准确率低于最终训练集的准确率，可能是因为在处理训练集数据时神经网络的参数已经被相应地优化，从而使得神经网络能够更好地拟合训练数据。但是，当处理测试集数据时，神经网络可能会遇到它没有见过的图像样本，这些样本可能与训练集中的样本有所不同，导致准确率有所下降

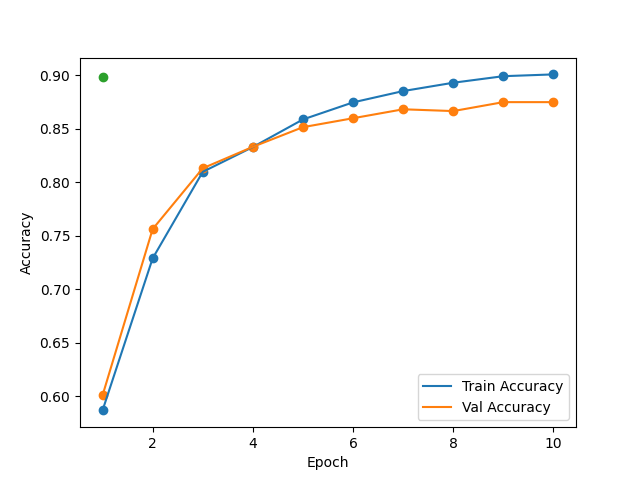


**图7 修改隐含层神经元数量参数结果**

对比图6，将hidden\_size1 值改为128，使得第一层隐含层神经元数量减少为原来的一半。

结果描述与分析：

对比图6，图六的结果震荡更显著。可能因为隐含层的神经元数量的减少会导致网络的容量（capacity）减小。神经网络的空间变得更小，存在的有效权重参数的数量也可能减少。如果数据中存在非线性关系或者噪声，那么模型可能无法捕捉到这些信息，导致模型更加敏感，使得震荡更显著。



**图8 修改优化器参数结果**

对比图6，将优化器改为SGD，动量设置为0.9

结果描述与分析：

对比图6，训练集和验证集的准确率都有明显降低，但是测试集结果与最终训练集的准确率几乎相同。准确率降低可能因为adam算法是可以自适应调整学习率的优化算法而可以在训练过程中动态地调整学习率。

1. 实验总结。基于整个实验过程与结果，对实验进行总结。

**【撰写内容】**

**实验步骤**如下：

数据准备：下载MNIST手写数字图像数据集，并将其划分为训练集、验证集和测试集。

模型构建：设计一个基于FCNN的神经网络模型，包括一个输入层、两个隐含层和一个输出层，并确定每个层的神经元数量。

损失函数和优化器选择：选用交叉熵损失函数作为神经网络模型的优化目标，并使用Adam作为优化器，进行模型训练。

模型训练：在训练集上进行模型训练，设置batch\_size为64，进行10个epoch。在训练过程中，采用验证集对模型进行调参，避免过拟合问题。

模型评估：在测试集上对训练好的模型进行性能评估，比较实际预测值和实际的手写数字图像类别。

**实验结果**总结：

本次实验的最终结果表明，采用上述步骤所构建的基于FCNN的神经网络模型在MNIST手写数字图像数据集上取得了不错的性能表现。通过实验的准确率计算可以发现，在测试集上，精确度大约为94.5%左右。并且通过增加隐含层层数使得模型复杂度提高可以使得精确度再次提升，表明了FCNN模型在MNIST手写数字图像数据集上的出色性能。

另外，在实验过程中通过实践发现，选择交叉熵损失函数和Adam优化器进行训练的组合是有效的，可以提高模型训练的稳定性和效果。此外，采用验证集可以更好地避免过拟合问题。

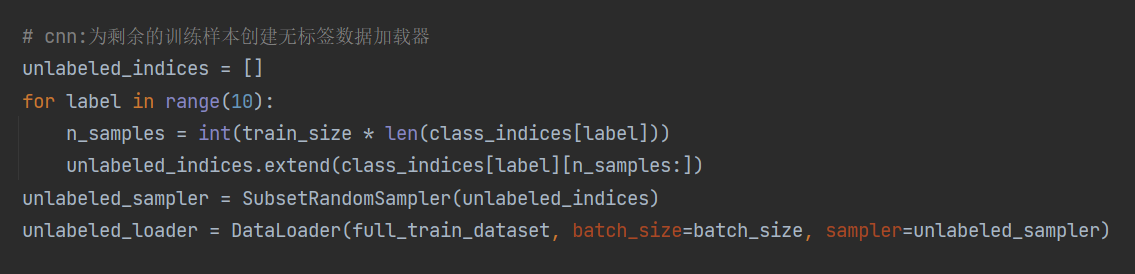
1. 小样本分类创新实验。以10%抽样的训练集数据为训练样本，剩余的训练样本可根据需要作为无标签数据进行利用。请查阅有关课外文献，探索新的特征表示或分类模型，要求是深度神经网络类型的学习方法。包括有监督、半监督或无监督等模式均可，代表性的方法如深度卷积网络、生成式对抗网络、强化学习、Transformer、自监督学习、去噪/变分自编码器等。在完整的测试集上进行性能的评估。本部分的撰写内容自行安排，将样本的选择、模型的构建、参数的设定、训练过程及结果进行清晰的描述与展示即可。要注重实验结果的对比，结合实验结果的分析得出模型的优点与缺点，以及可能改进的方向。

**【撰写内容】**

以10%抽样训练，训练方法选用**卷积神经网络。**

实验步骤如下：

**数据准备**：下载MNIST手写数字图像数据集，并将其划分为训练集、验证集和测试集。与FCNN的准备相同。区别在于将剩余的训练样本作为无标签数据，以供选用。



**图9 创建无标签数据集**

**模型构建**：

设计一个基于CNN的神经网络模型，含有两个卷积层、两个最大池化层和两个全连接层。激活函数选用relu，理由同前文所述。

其中设置了卷积核大小为3\*3，且padding设置为1，使得每次卷积计算后的结果依然可以保持原来的大小，使得输入的图像的边缘信息在后续卷积计算中依然存在，保持信息量的稳定性。卷积层的卷积核通道大小会直接影响模型的性能和运算速度。通道越大，卷积层的参数数量就越多，模型所需的计算资源也越大，同时容易导致过拟合；而通道设置过小可能导致特征提取能力受限、容易出现欠拟合，难以适应复杂任务。

每次卷积之后都要经过最大池化层，目的是主要的作用在于对卷积层提取的特征进行降维，并保留其最显著的特征。它的好处主要有三个：

降低计算量。在卷积神经网络中，池化层通过对相邻区域内的特征值进行统计来减少特征图的大小，从而减少了CNN中需要处理的参数数量，降低了整体的计算复杂度。

改善网络鲁棒性。在进行卷积操作时，特征图的每一个像素都被卷积核扫描过，这就导致了一定程度的过拟合。最大池化层可以有效地解决这个问题，它能够通过取最大值来选择图像中最具代表性的特征点，同时抑制不重要的特征对模型产生的影响。

提高模型泛化性能。最大池化层可以在一定程度上缩小特征图，使模型更加关注本质特征，从而减少误差造成的影响。

全连接层是网络的最后一层，它的主要作用是将卷积和池化操作提取的特征映射转换成固定长度的向量，然后将这个向量输入到下一个全连接层或最终的输出层用于分类或回归。

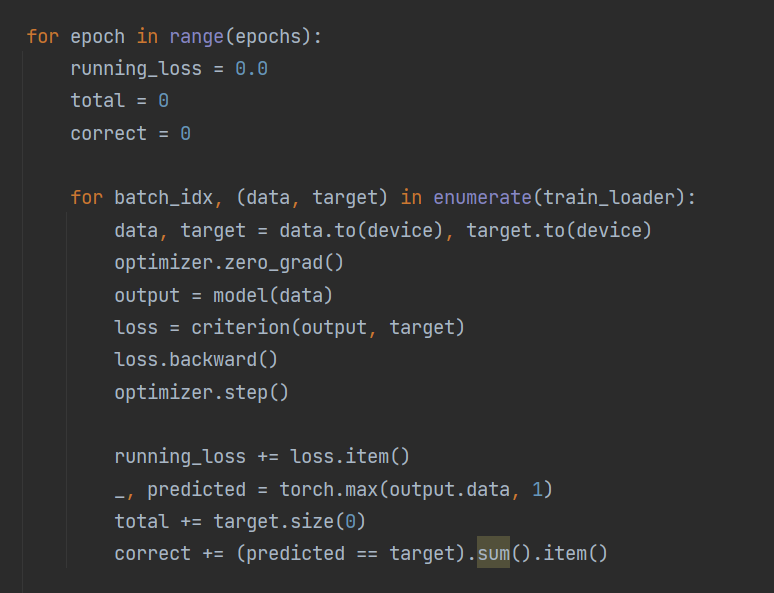
经过两次卷积和两次最大池化后，最后一层的特征图大小为64x7x7。具体来说，输入的图片大小为28x28，经过第一次卷积后得到的特征图大小为28x28x32（32是卷积核个数），经过第一次最大池化后，特征图大小减半变为14x14x32。同理，经过第二次卷积后得到的特征图大小为14x14x64，经过第二次最大池化后，特征图大小再次减半变为7x7x64。因此，将最后的特征图展开后，得到的向量大小为64x7x7，作为第一个全连接层的输入维度，输出维度设置为128，同时也是输出层的输入维度。最后一个全连接层的输出节点数应该等于数据集中类别的数量，也就是10（图10中的num\_class）。



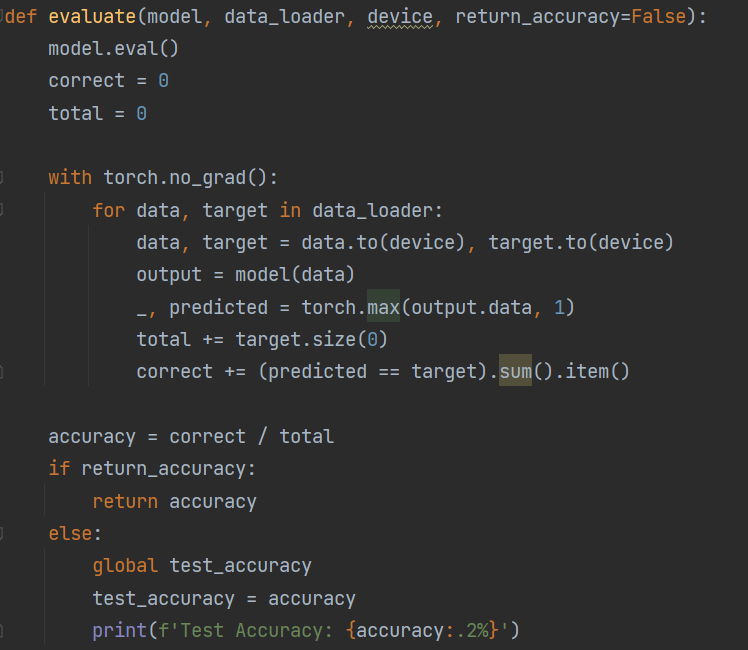
**图10 CNN模型**

**损失函数和优化器选择**：选用交叉熵损失函数作为神经网络模型的优化目标，并使用Adam作为优化器，进行模型训练。

**模型训练**：在训练集上进行模型训练，设置batch\_size为64，进行10个epoch，学习率设置为0.001。在训练过程中，采用验证集对模型进行调参，避免过拟合问题。验证集训练过程与训练集训练过程类似，不给出。

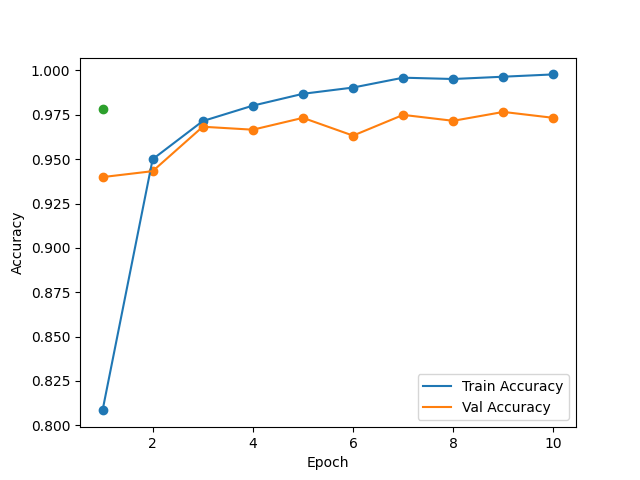


**图11 训练过程**

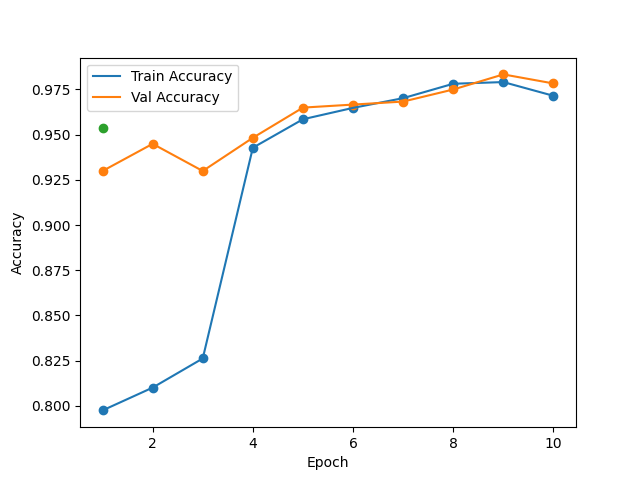


**图12 评估过程**

**模型评估**：在测试集上对训练好的模型进行性能评估，比较实际预测值和实际的手写数字图像类别。



**图13 不使用无标签数据集**



**图14 使用无标签数据集**

参数如下：

epochs = 10

batch\_size = 64

train\_size = 0.1 # 训练集大小的改变

val\_split = 0.1 # 从新训练集中抽取10%作为验证集

self.conv1 = nn.Conv2d(1, 32, kernel\_size=3, padding=1)

self.conv2 = nn.Conv2d(32, 64, kernel\_size=3, padding=1)

self.pool = nn.MaxPool2d(2, 2)

self.fc1 = nn.Linear(64 \* 7 \* 7, 128)

self.fc2 = nn.Linear(128, 10)

learning\_rate = 0.001 # 学习率

**结果分析：**

模型的验证集结果一直维持在高准确率，有小幅度震动，在正常范围内。但是对于训练集结果，我们发现它的起点相比之下低了很多，但是随着训练完毕的epochs增加，显著提高到和验证集差不多准确率的水平并超越。多次尝试后我发现这种情况仍然存在。通过查阅资料给出解释：

1. 训练集准确率低开高走：在卷积神经网络中，权重和偏差会随机初始化。这是因为在训练CNN模型时，需要通过优化算法对权重和偏差进行更新，从而使模型逐渐优化。在初始化过程中，一般会采用高斯分布或者均匀分布来生成随机数，作为权重和偏差的初始值。这样可以保证权重和偏差的值不同，有利于模型的学习和收敛。在训练过程中，随着epoch次数的增加，CNN会根据当前数据对权重和偏差进行更新，从而逐渐优化模型，提高准确率

2. 验证集准确率一直比较高： CNN在卷积和池化操作中通过滑动窗口的方式对图像进行特征提取和压缩，保证了对特征的位置不变性，并且减少了同一位置上的特征提取次数。因此，CNN在训练过程中会学习到更多的特征，能够更好地适应不同的数据集。这种特性使得CNN模型早期在验证集上表现比训练集更好。

3. 效果好于FCNN：即使初期在训练集上表现不如FCNN，在后期时CNN通过学习更多的特征和减少过拟合的方式，CNN的验证集正确率一直保持高水平。也有可能是模型复杂度的关系，因为前面构造的FCNN的模型结构比较简单，而我设置的CNN相比之下层数更多，更复杂，能更好地捕捉数据中的非线性关系和复杂模式，从而有更高的准确率。另外，如果模型很复杂，它通常会比较灵活，可以适应更广泛的输入数据。这意味着它可能具有更好的泛化能力，并且可能可以通过减少偏差（bias）和方差（variance）之间的平衡来减少误差率。

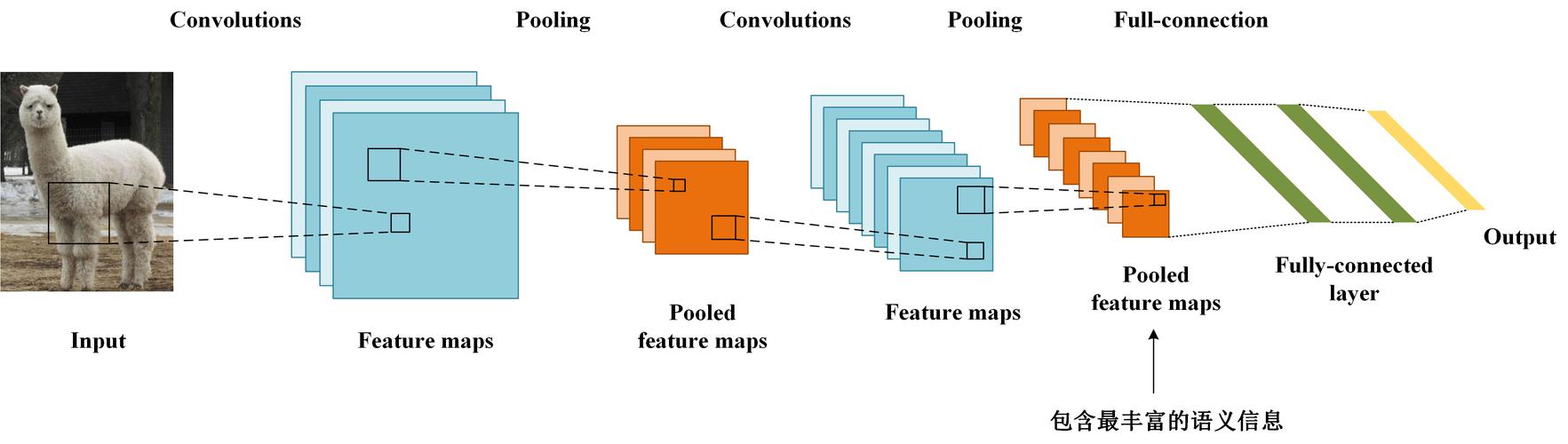
4. 使用无标签数据集与否对比：结果相差不大，在可接受误差范围内。但是训练集的准确率提升速度更慢了。使用没有标签的数据进行半监督学习的结果比不使用要差是可能的。这可能是由于未标注数据中包含许多“噪音”或与训练集不一致的数据，这可能会干扰训练过程并导致性能下降，训练速度变慢。一般来说，使用未标注数据的优点是可以提供更多的数据用于训练，并且可以帮助模型更好地捕捉数据中的复杂关系和特征，从而提高模型的泛化能力和准确率。

给出基于本次实验的CNN工作原理总结：

CNN是一种深度学习模型，其工作原理基于图像卷积和池化机制。卷积操作通过搭载一组可学习的滤波器在输入图像上滑动实现，从而提取出不同尺度、方向和形状的特征。这些特征被送入池化层，通过采样和降维进一步减小计算复杂度，并增强了模型的泛化能力。

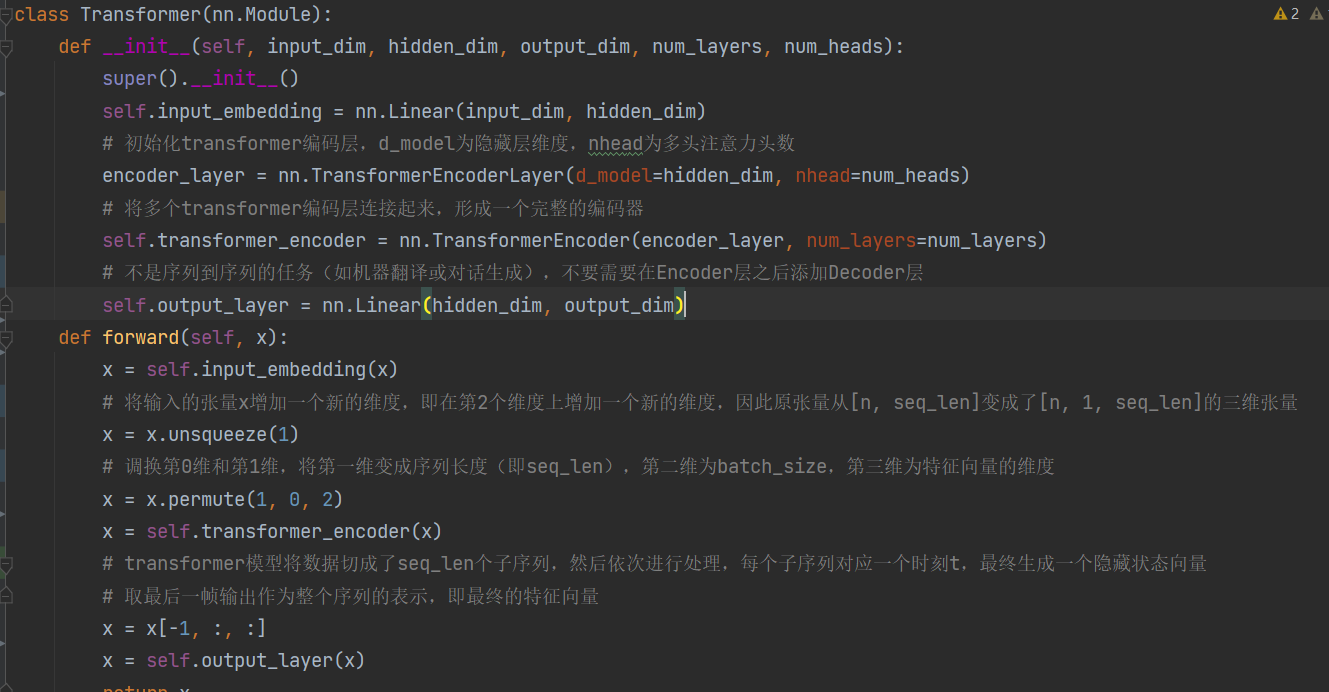
随后，卷积与池化交替进行多次，进一步提取高层次、抽象的特征，最终将特征图展开并输入全连接层中进行分类或回归任务。全连接层将所有神经元连接起来，并产生相应的输出，从而实现对输入图像的分类或回归。

同时，训练CNN的过程利用反向传播算法，计算模型损失函数关于参数的梯度并利用优化算法迭代更新权重和偏置，从而提高模型的准确性和精度。

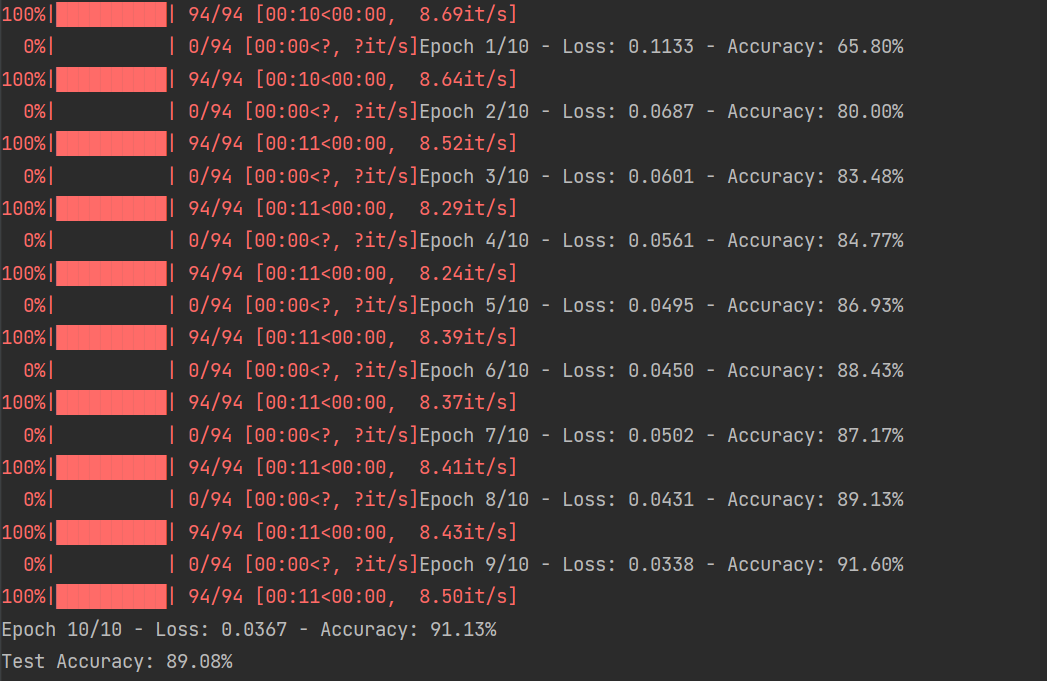
总体而言，CNN具有自动特征提取、参数共享和平移不变性等特点，可以处理高维度、大规模和复杂结构的数据。

**图15 CNN在本次实验中的工作流程**

除**CNN**以外，我还尝试过使用**Transformer**，但起到的效果并不是很好，相关信息如下：



**图16 transformer模型**



**图17 transformer训练结果**

分析效果不好的原因可能是：

1. Transformer模型适用于序列到序列的任务，例如自然语言翻译、对话生成等，而MNIST是一个图像分类任务，不太适合使用Transformer模型。如果使用了Transformer模型，需要对图像进行展开成一个序列作为模型的输入，这可能会丢失图像中的一些空间信息，影响模型的分类准确率。

2. MNIST数据集中的数字都是黑白的，只有一个通道，而Transformer模型通常使用多头注意力机制，需要多个通道的输入才能更好地捕捉特征。

3. 可能需要提高模型复杂度，增加模型的隐藏层数、头数和隐藏维度等参数，使其更好地匹配MNIST数据集的任务特点。

4. 可能需要调整学习率或优化器等超参数，或者采用其他的数据增强方法，例如旋转、缩放、平移、仿射变换等，可以帮助提高模型的泛化能力和准确率。