# 手写数字识别报告

# 一、选题背景和意义

手写数字识别技术能够自动化许多文档和数据输入过程，减少人工错误和提高效率。随着技术的发展，这一领域的应用越来越广泛，研究和改进手写数字识别算法的重要性也日益增加。在此基础上，探索新的数据集和改进模型的性能变得尤为重要。

# 二、问题建模

手写体数字识别是一个典型的计算机视觉问题，主要涉及到图像中手写数字的检测和识别。这个问题可以通过监督学习方法来解决，其中模型被训练后用以从图像中识别出数字。这一任务不仅在技术领域内具有广泛的应用，如自动表单处理、银行支票处理等，还对机器学习和图像处理的教育研究具有重要意义。

## 数据集

我们使用的主要数据集是MNIST，一个包含70,000个手写数字图像的标准数据集。每个图像是一个28x28像素的灰度图像，表示数字0到9。同时，为丰富数据集，我们采取了反色处理，共计140,000个数据集

## 问题转换

课程要求包括将MNIST数据集中的白底黑字图像转换为黑底白字图像，并将这些转换后的图像与原图混合，形成一个新的数据集。这种变换是为了增加模型处理不同背景和颜色方案的能力，从而使模型更加健壮。

# 方法描述

## 使用 PaddlePaddle 实现手写数字识别

PaddlePaddle 是一个由百度开发的深度学习平台，它提供了丰富的API和工具来支持深度学习应用的开发。在手写数字识别任务中，我们可以利用 PaddlePaddle 来构建、训练和测试模型。

## 具体流程

### 1.数据预处理：

我们加载原始的MNIST数据集。为了实现数据的增强，满足课程要求，我们将图像的白底黑字转换为黑底白字，通常通过0减去原像素值实现。将原始和转换后的数据集混合，增加数据多样性。

### 2.构建模型：

使用卷积神经网络（CNN）进行模型构建。CNN 在图像识别任务中表现出色，因其能够捕捉图像中的局部特征。模型架构通常包含多个卷积层、激活层（如ReLU）、池化层以及最后的全连接层，这里我们考虑了vgg网络和lenet网络，对模型的激活函数，池化层，优化器，学习率，卷积层的安排，epoch以及batch-size做出了修改，最终目的是追求更高精度的预测结果。

3.模型训练：

使用交叉熵损失函数进行模型的训练。

### 4.模型评估和调优：

在测试集上评估模型性能，以损失率和准确度作为最终的评判标准，根据结果反馈调整模型参数或结构。

### 5.可视化分析：

我们以控制变量作为原则，将我们得到的数据进行可视化分析，并对可视化的结果做出详细的阐释和讨论。

# 对实验结果的分析

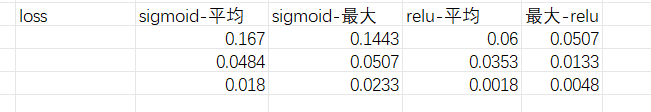
对于不同参数的调整，我们统一采取控制变量的方法，对模型参数进行修改，通过表格和图形相结合的方式进行分析。

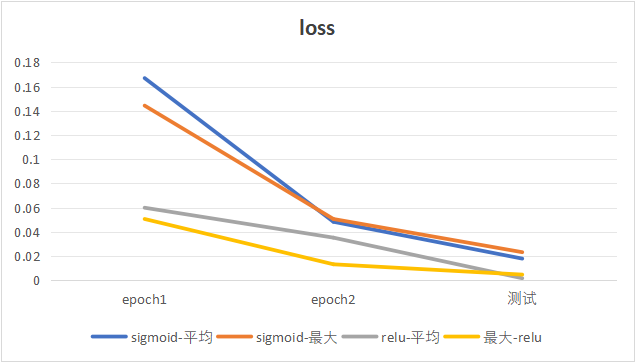
## 探究激活函数和池化层

常用的激活函数有sigmoid和relu，池化层有最大池化层和平均池化层。在Lenet网络中采用的是sigmoid激活函数和平均池化层，这里我们对这两个参数进行了修改，选取了epoch为2，batch-size为64，学习率为0.001等统一参数下进行测试与分析

### loss函数的对比

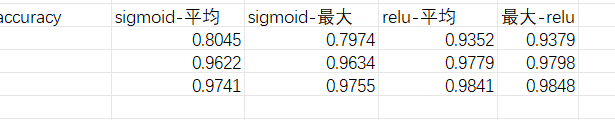
下面是结果图

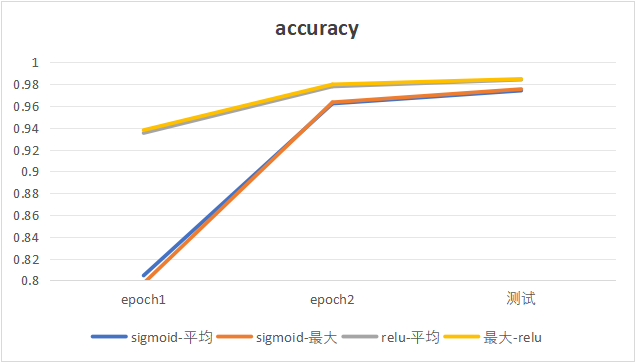




### 准确率的对比

下面是准确率的结果图





分析图表我们可以得到：最大池化层和relu函数分别优于平均池化层和relu函数，同时relu函数对精确度提升和loss的减少是要显著优于池化层的选择最大池化层。

这是因为

1池化层

主要用于卷积神经网络（CNN）中，其主要目的是减少数据的空间尺寸，从而减少参数数量和计算复杂性，同时还能提高特征检测的鲁棒性。常见的池化操作包括最大池化（Max Pooling）和平均池化（Average Pooling）：

最大池化：选择池化窗口中的最大值作为输出，这有助于突出重要的特征，通常能提高模型的表现，这有助于图像特性的提取，有利于识别不同图像之间的特征区别。

平均池化：计算池化窗口中所有值的平均，这有助于平滑特征表达。

2.激活函数

激活函数在神经网络中用于引入非线性，使得网络能够学习和表示复杂的数据模式。常用的激活函数包括：

Sigmoid函数：它将输入值压缩到0和1之间，通常用于输出层的二分类问题。然而，在隐藏层中使用Sigmoid可能导致梯度消失问题，这是因为其导数在输入值较大或较小时接近于零。

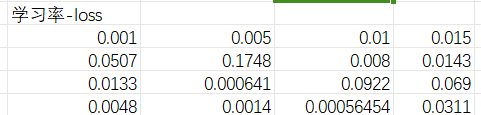
ReLU函数：线性整流函数（Rectified Linear Unit, ReLU）输出输入值的正部（如果输入为正则输出该值，否则输出0）。由于其计算简单且在正区间内梯度恒定，ReLU通常有助于加速神经网络的收敛，并减少梯度消失问题。

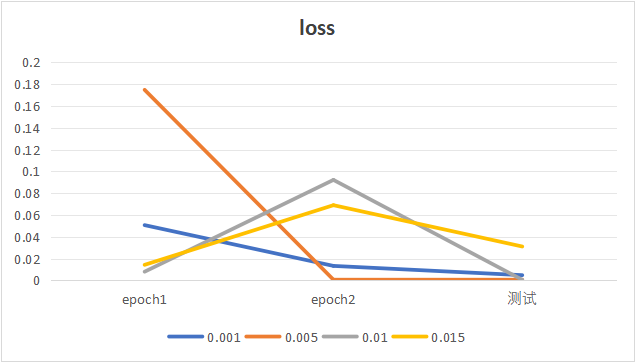
网络中大量使用Sigmoid，可能会导致梯度消失，从而使训练过程中的loss下降缓慢。相反，ReLU通常能提供更快的收敛速度。

## 探究学习率参数：

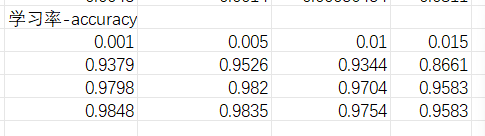
查阅相关资料，我们得知常见的初始学习率范围通常在0.001到0.1之间。我们基于此设置学习率参数，并分析

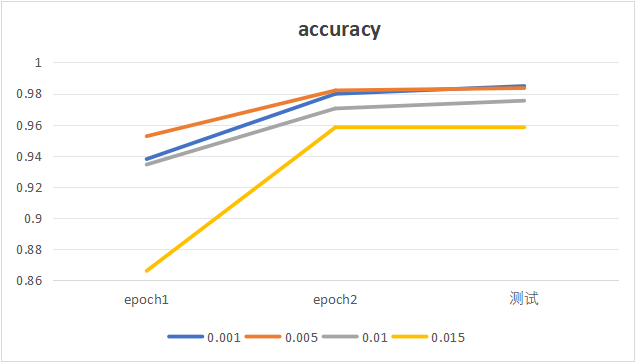
### 损失函数对比





### 准确率对比





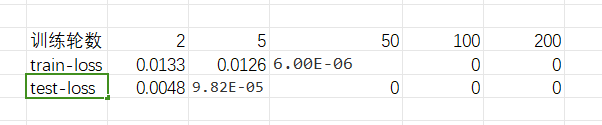
分析结果我们可以发现学习率为0.001和0.005时，训练结果的loss和accuracy都显著的优于学习率为其他的情况，而当学习率高达0.015时，整个训练和测试结果都明显的变差了。查阅相关资料，这或许是因为此时0.015，学习率设置得过大，模型权重和偏置更新的步长会很大，这可能导致模型在训练过程中跳过最优解，导致模型无法收敛，从而降低精准度。

我们还了解到如果学习率设置得过小，模型权重更新的步长会很小，虽然模型可能会收敛到最优解，但收敛速度会非常慢，且容易陷入局部最优解，同样会降低精准度。

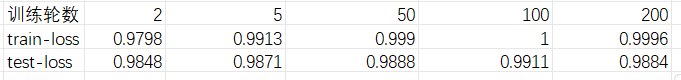
## epoch

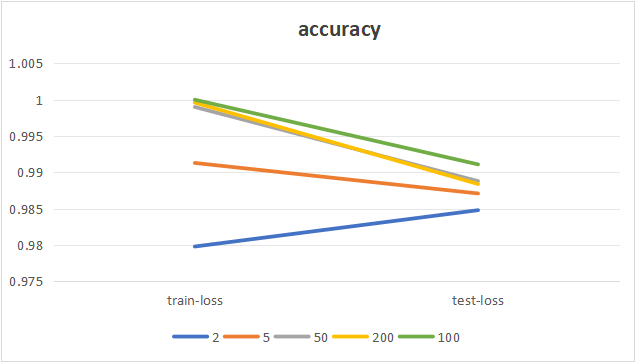
查阅资料可以得到，手写数字识别的数据集规模是中等数据集，epoch的合理范围为10到100，，基于此我们修改参数，进行训练

### loss



### 精确度：



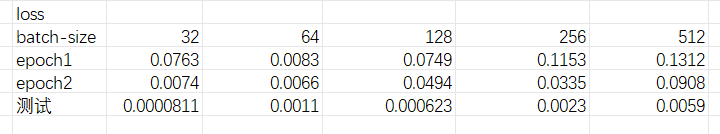


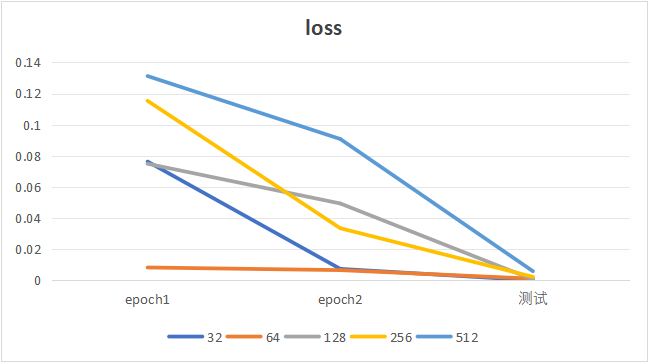
我们发现，随着训练轮数的增加，loss逐渐降低，当训练轮数为50的时候，损失函数几乎已经接近于0了，我们结果的最高精确度出现在训练轮数为100的时候，达到了0.9911而当训练轮数为200时，精确地反而降低了。对于训练集，随着轮数的增加，准确度呈现了上升的趋势，这也是合理的。分析原因，如果epoch数过少比如只有2时，模型可能没有足够的时间学习数据，导致欠拟合。欠拟合的模型在训练数据和未见过的新数据上都表现不佳，这也使得精确度降低。相反，如果epoch数过多比如200轮时，模型可能会学习到数据中的噪声和非代表性特征，导致过拟合。过拟合的模型在训练数据上表现良好，但在新数据上表现较差，也就是最终的测试集精确度反而降低了。

## batch\_size

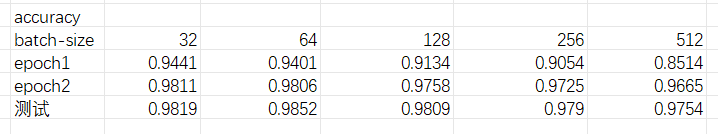
常见的batch\_size设置为32、64或128。这些值是经验性的选择，可以提供良好的性能和训练速度的平衡，基于此我们设置了参数

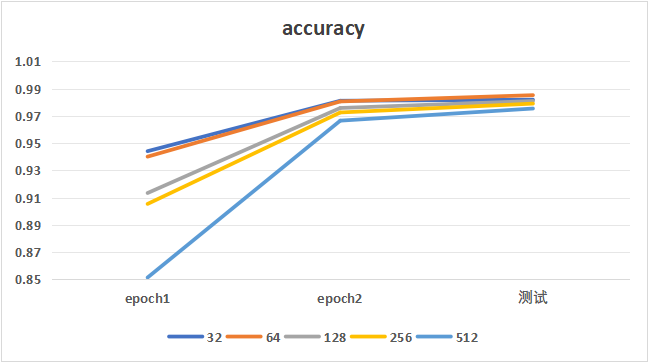
### loss





### accuracy





分析结果我们发现，随着batch\_size的逐渐提升，整体的损失函数和精确度都下降了，而最优的精确度和Loss出现在了batch-size为64的情况下。这是因较大的batch-size可能会使大批量可能无法充分捕捉到数据中的多样性，对于精确度有下降。而较小的batch-size通常具有更好的泛化能力，因为它们能够在每次更新时暴露给模型更多的数据多样性。然而理论上，较大的batch-size使模型在训练集上的性能更好，loss更低，然后结果中却并不是这样的表现，这或许于随机性有一定关系

## 网络的修改

修改：

第三个卷积层 (conv3):

输入通道数: 16（来自上一个卷积层的输出）

输出通道数: 32

卷积核大小: 3x3

步长: 1

填充: 1

这层之后跟着一个最大池化层，池化窗口为2x2，步长为2。这样的设置有助于进一步减小特征图的空间尺寸，增加特征的抽象程度。

第四个卷积层 (conv4):

输入通道数: 32（来自上一个卷积层的输出）

输出通道数: 64

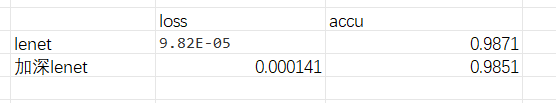
卷积核大小: 3x3

步长: 1

填充: 1

与传统的LeNet-5不同，此层后面没有池化层。保留较大的特征图尺寸，可以为随后的全连接层提供更丰富的特征。

因此之后的第一个全连接层也做出了修改，输入的是256

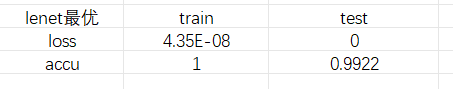


最终的结果如上图所示

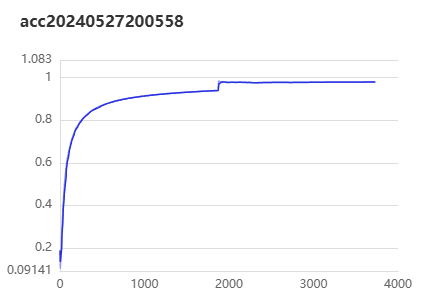
## 精确度最高：

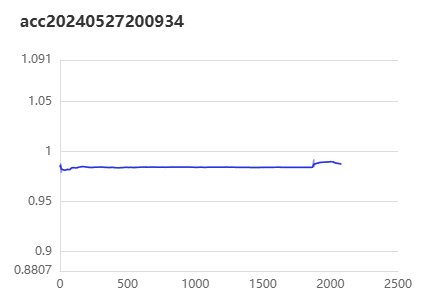
### Lenet

我们将batch设置为100，epoch设置为128，学习率为0.001，采用Adam优化器，采用最大池化层和relu激活函数，经过训练后得到的最高预测精度为0.9922，loss和acc的结果详见下表。



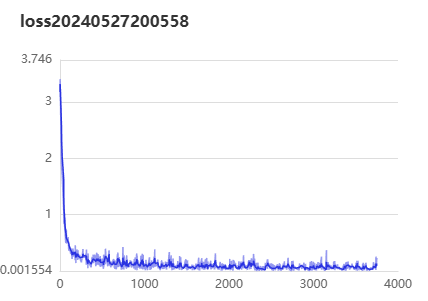
训练集的accu变化如下所示，结合训练的中间结果，我们可以得到epoch为43精确度发生了飞跃，之后的epoch的精确度并没有显著变化，这也可能是因为已经要达到50了

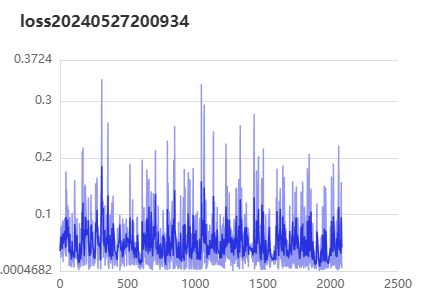




上图是测试集预测的精确度

训练集和测试集的损失函数变化如下图所示，随着训练轮数的增加，损失函数迅速下降。测试集的损失函数也达到了很小量级。





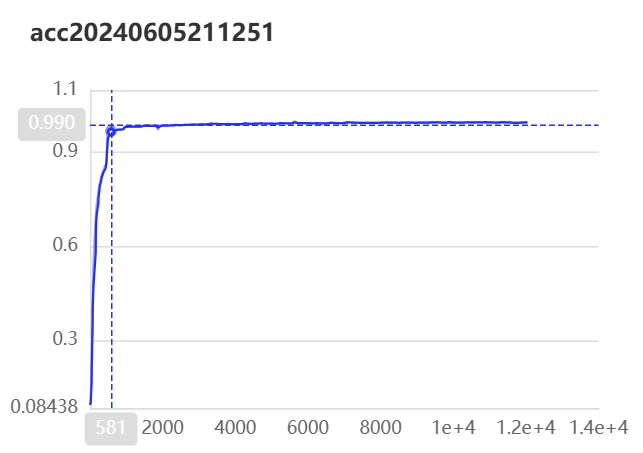
### VGG

VGG：

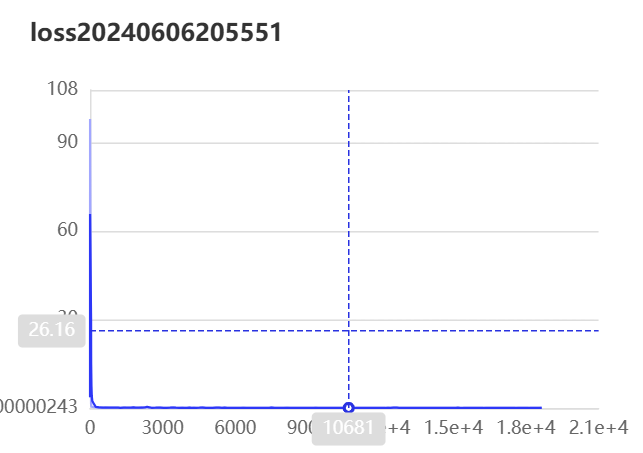
batch设为256,epoch设为35,学习率为0.001。得到了0.9838的预测精度。

|  |  |  |
| --- | --- | --- |
| VGG最优 | train | test |
| loss | 0.0066 | 5.0889e-4 |
| accu | 0.9927 | 0.9838 |

可以看到，训练精度上升很快，并且在第3个epoch接近上限。后续缓慢上升。



损失率也很快下降。



可以看出，当参数设置合理时。VGG能用较少的训练轮数实现较高的准确度，处于效率和准确率的综合考虑下，训练轮数设置为20轮左右较好。

### 

# 分工以及分配比

余旺：代码书写，调参，模型修改，文档书写 23%

吕伯华：代码书写，调参 20 %

常高林：调参， 文档书写 19 %

刘治岐：调参，数据处理，ppt 19%

万守正：调参，数据处理，文档书写 19%