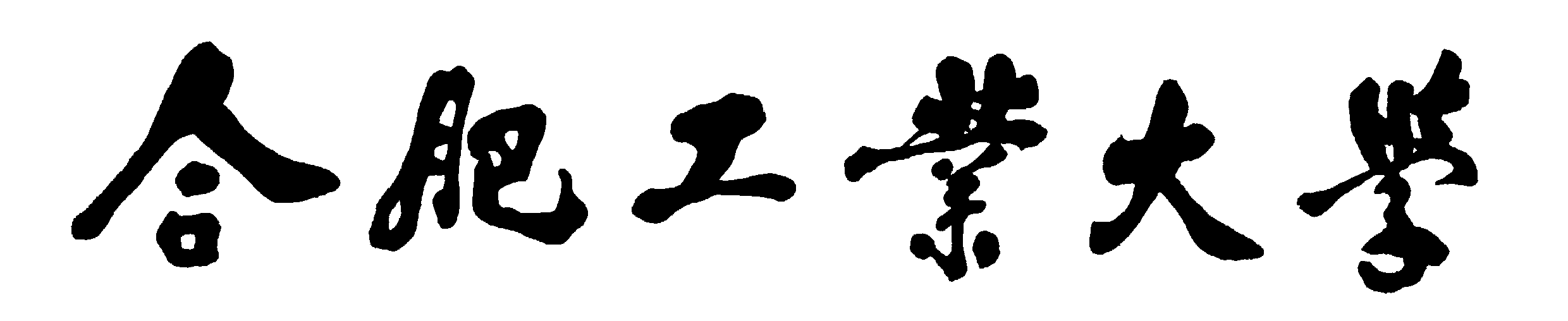
****

计算机与信息学院

人工智能原理设计报告

|  |  |
| --- | --- |
| 专 业 班 级 | 计算机科学与技术20-3班 |
| 学生姓名及学号 | 管斌2020216359 |
| 任 课 教 师 | 张赞 |
| 2022 ~2023学年第 一 学期 宣城校区 | |

说 明

实验报告是关于实验教学内容、过程及效果的记录和总结，因此，应注意以下事项和要求：

1．实验报告要求：**格式规范，语言表达清楚，数据和程序真实。并能够理论联系实际，认真分析实验中出现的问题与现象，总结经验。**

2．每位同学应独立完成实验报告的撰写，严禁抄袭或拷贝，否则，一经查实，按作弊论取，并取消理论课考试资格。

3. 可根据实际需要调整每个单元格的篇幅。

4. 请按照要求填写实验报告。算法源代码请放置在附录中。

实验序号\_\_4\_\_ 作业名称： 基于 MindSpore 框架的 MNIST 手写体识别

实验时间∶2022年11月13日

|  |
| --- |
| **实验内容** |
| **一、题目内容和要求：**  **1.1题目内容**  利用Mindspore 搭建Lenet-5卷积神经网络框架，实现对0-9的数字手写体的识别。  **1.2实验要求**  本实验拟实现一个简单的图片分类的功能，整体流程如下：  1、 处理需要的数据集，这里使用了 MNIST 数据集。  2、 定义一个网络，这里我们使用 LeNet 网络。  3、 定义损失函数和优化器。  4、 加载数据集并进行训练，训练完成后，查看结果及保存模型文件。  5、 加载保存的模型，进行推理。  6、 验证模型，加载测试数据集和训练后的模型，验证结果精度。 |
| **二、问题背景和相关工作介绍**  **2.1 实验平台介绍**  本实验所有内容在windows-64操作系统，python 3.8，MIndspore版本号为1.9，整个实验过程在jupyter notebook 上完成。  **2.2 Mindspore框架**  MindSpore是华为公司推出的一个全场景深度学习框架，旨在实现易开发、高效执行、全场景覆盖三大目标，其中易开发表现为API友好、调试难度低，高效执行包括计算效率、数据预处理效率和分布式训练效率，全场景则指框架同时支持云、边缘以及端侧场景。  MindSpore总体架构如下图所示，分为前端表示层（Mind Expression，ME）、计算图引擎（Graph Engine，GE）和后端运行时三个部分。  **MindSpore Extend（扩展层）：**MindSpore的扩展包，期待更多开发者来一起贡献和构建；  **MindExpress（表达层）：**基于Python的前端表达，未来计划陆续提供C/C++、Java等不同的前端；MindSpore也在考虑自研编程语言前端-仓颉，目前还处于预研阶段；同时，内部也在做与Julia等第三方前端的对接工作，引入更多的第三方生态。  **MindCompiler（编译优化层）：**图层的核心编译器，主要基于端云统一的MindIR实现三大功能，包括硬件无关的优化（类型推导/自动微分/表达式化简等）、硬件相关优化（自动并行、内存优化、图算融合、流水线执行等）、部署推理相关的优化（量化/剪枝等）；其中，MindAKG是MindSpore的自动算子生成编译器，目前还在持续完善中。  **MindRE（全场景运行时）**：这里含云侧、端侧以及更小的IoT。    MindSpore函数式可微分编程架构可以让用户聚焦模型算法数学原生表达。资深的深度学习开发者都体会过手动求解的过程，不仅求导过程复杂，结果还很容易出错。所以现有深度学习框架，都有自动微分的特性，帮助开发者利用自动微分技术实现自动求导，解决这个复杂、关键的过程。深度学习框架的自动微分技术根据实现原理的不同，分为以Google的TensorFlow为代表的图方法，以Facebook的PyTorch为代表的运算符重载，以及以MindSpore为代表的源码转换方法（Source to Source，S2S自动微分技术）  IMG_256  **2.3 卷积神经网络**  卷积神经网络（Convolutional Neural Networks）是一种深度学习模型或类似于人工神经网络的多层感知器，常用来**分析视觉图像**。卷积神经网络的创始人是着名的计算机科学家 Yann LeCun，目前在 Facebook 工作，他是第一个通过卷积神经网络在 MNIST 数据集上解决手写数字问题的人。  一个卷积神经网络主要由以下 5 层组成：  输入层（Input layer）  卷积层（CONV layer）  ReLU 激励层（ReLU layer）  池化层（Pooling layer）  全连接层（FC layer）  **2.3.1 输入层**  该层要做的处理主要是对原始图像数据进行**预处理**，其中包括去均值、归一化、白化。  **2.3.2 卷积层**  卷积层，卷积神经网络中每层卷积层由若干卷积单元组成，每个卷积单元的参数都是通过 反向传播算法 最佳化得到的。. 卷积运算的目的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级，更多层的网路能从低级特征中迭代提取更复杂的特征，卷积层的主要作用就是提取图片的特征。  对于图像输入输出的个数，输入图像的个数和每一个fileter中卷积核的个数是相同的图中卷积核Fileter的个数与输出通道数Output是对应的，每一个输出由多通道卷积累加并加上偏执Bias得到，一个卷积核只有一个偏执值，我们可以根据这个来确定每一个卷积层过后输出图像的个数。  IMG_256  **2.3.3 激励层**  激励层需要用到激活函数，该层提供网络的非线性建模能力，是用来加入非线性因素的，提高神经网络对模型的表达能力，解决线性模型所不能解决的问题。如果不使用激活函数，这种情况下每一层输出都是上一层输入的线性函数。无论神经网络有多少层，输出都是输入的线性函数，这样就和只有一个隐藏层的效果是一样的。这种情况相当于多层感知机(MLP)。CNN 常用的一些激活函数有：   1. **ReLU函数**   它的特点是收敛快，求梯度简单，但较脆弱，图像如下：     1. **Sigmoid函数**   是使用范围最广的一类激活函数，具有指数函数形状，它在物理意义上最为接近生物神经元。此外，(0, 1) 的输出还可以被表示作概率，或用于输入的归一化，代表性的如Sigmoid交叉熵损失函数。    但是Sigmoid 的软饱和性，使得深度神经网络在二三十年里一直难以有效的训练，是阻碍神经网络发展的重要原因。   1. **Tanh函数**   Tanh也是一种非常常见的激活函数。与Sigmoid相比，它的输出均值是0，使得其收敛速度要比Sigmoid快，减少迭代次数。然而，从途中可以看出，Tanh一样具有软饱和性，从而造成梯度消失。    **2.3.4池化层**  池化层的本质是一个**下采样**，夹在连续的卷积层中间， 用于压缩数据和参数的量，**减小过拟合**。有多种不同形式的[非线性](https://so.csdn.net/so/search?q=%E9%9D%9E%E7%BA%BF%E6%80%A7&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/Chen_Swan/article/details/_blank)池化函数，例如均值池化和最大值池化，如果输入是图像的话，那么池化层的最主要作用就是压缩图像，池化层的具体作用有：  （1）**特征不变性**，也就是我们在图像处理中经常提到的特征的尺度不变性，池化操作就是图像的 resize，平时一张狗的图像被缩小了一倍我们还能认出这是一张狗的照片，这说明这张图像中仍保留着狗最重要的特征，我们一看就能判断图像中画的是一只狗，图像压缩时去掉的信息只是一些无关紧要的信息，而留下的信息则是具有尺度不变性的特征，是最能表达图像的特征。  （2）**特征降维**，我们知道一幅图像含有的信息是很大的，特征也很多，但是有些信息对于我们做图像任务时没有太多用途或者有重复，我们可以把这类冗余信息去除，把最重要的特征抽取出来，这也是池化操作的一大作用。  （3）在一定程度上**防止过拟合**，更方便优化。  （4）**实现非线性。**  （5）**扩大感受野。**  **2.3.5 全连接层**  全连接层在整个卷积神经网络中起到的是**“分类器”**的作用，将学到的“分布式特征表示”**映射到样本标记空间**的作用，即将前面经过多次卷积后高度抽象化的特征进行整合，然后可以进行归一化，对各种分类情况都输出一个概率，之后的分类器(Classifier)可以根据全连接得到的概率进行分类。  **2.4 Lenet-5网络**    如图所示，Lenet-5神经网络模型包括两个卷积层，两个池化层和三个全连接层这样一个七层模型组成，其中未加入dropout，最后一层改用softmax函数实现。  其每一层的结构顺序为：  **输入层-->卷积层-->池化层-->卷积层-->池化层-->全连接层-->全连接层-->全连接层-->输出**   1. **INPUT层**   输入图像的尺寸统一归一化为 32x32。   1. **C1层 (卷积层)**   输入图片：32x32  卷积核大小：5x5  卷积核种类：6  输出6个特征图(feature map)，每个特征图的大小：28x28   1. **S2层-(Pooling层)**   输入大小：28x28  采样区域：2x2  采样方式：4个输入相加，乘以一个可训练参数，再加上一个可训练偏置。结果通过sigmoid函数  采样种类：6  输出6个特征图(feature map)，每个特征图的大小：14x14 (28/2)   1. **C3层 (卷积层)**   输入大小：S2中所有6个或者几个特征图(feature map)组合  卷积核大小：5x5  卷积核种类：16  输出16个特征图(feature map)，每个特征图的大小：10x10   1. **S4层(Pooling层)**   输入大小：10x10  采样区域：2x2  采样方式：4个输入相加，乘以一个可训练参数，再加上一个可训练偏置。结果通过sigmoid  采样种类：16  输出16个特征图(feature map)，每个特征图的大小：5x5 (10/2)   1. **F5层 (全连接层)**   输入大小：S4层的全部16个单元特征图(与S4全相连)  卷积核大小：5x5  卷积核种类：120  输出120个特征图(feature map)，每个特征图的大小：1x1  可训练参数/连接：120\*(16\*5\*5+1)=48120  参数详细说明： F5层是一个卷积层。由于S4层的16个图的大小为5x5，与卷积核的大小相同，所以卷积后形成的图的大小为1x1。  这里形成120个卷积结果。每个都与上一层的16个图相连。所以共120\*(16\*5\*5+1)=48120个参数，同样有48120个连接。   1. **F6层 (全连接层)**   F6层与F5进行全连接  输入大小：C5层输出的 120维向量(120x1)  计算方式：计算输入向量和权重向量之间的点积，再加上一个偏置，结果通过sigmoid函数输出。  输出：84维向量(84x1)   1. **Output层 (全连接层)**   输入：84维向量(84x1)  输出：10维向量(10x1)，代表所抽取的特征属于哪个类别 |
| **三、解题思路**  **3.1 前期准备**  **3.1.1 mindspore安装**  安装Mindspore API，进入官方下载界面**https://www.mindspore.cn/install/**，根据自己的操作系统和硬件平台选择所要安装的版本，可以利用pip安装方法如下：  pip install <https://ms-release.obs.cn-north-4.myhuaweicloud.com/1.9.0/MindSpore/cpu/x86_64/>  mindspore-1.9.0-cp38-cp38-win\_amd64.whl --trusted-host ms-Release  .obs.cn-north-4.myhuaweicloud.com -i https://pypi.tuna.tsinghua.edu.cn/simple  安装Mindvision API，Mindvision是华为自主研发的一个基于MindSpore的计算机视觉开源工具箱，运用该工具像可以调用其中各种神经网络模型，对模型的构建更加简单，就像搭积木一样，其中就包括Lenet5，Alexnet模型等，本实验在后期的讨论部分尝试使用多种模型实现数字手写体的识别，对不同的模型进行一个对比，选择出针对此问题的最佳模型，以及对应的模型参数，安装Mindvision的方法如下：  pip install mindvision  **3.1.2 数据集的下载和处理**  我们使用的数据集是Mnist数据集，数据集是由 10 类 28×28 的灰度图片组成，训练数据集包含 60000 张图片，测试数据集包含 10000 张图片。我们需要先从http://yann.lecun.com/exdb/mnist/ 上面下载数据集，放在同级目录下新创建的MNIST\_DATA下，将下载的训练集和测试集分开，分别放在train和test两个文件夹当中，里面对应的数据一个是10个种类的图片，另一个是其所有图片对应的标签，供后续训练过程中使用。    **3.2 Lenet\_5模型搭建**  我们需要计算卷积以后的图片尺寸的大小，计算公式如下：    其中n为输入图片的边长，k为卷积核的边长，p为所需填充数，s为卷积的步长，该层中卷积的步长为s=1，且不需要填充，即p=0，所以由32×32做卷积变成28×28的图像，需要的卷积核为5×5。  首先，我们定义第一层**卷积层**，设置输出通道为6，卷积核大小为5×5，并且不对输出图像进行填充，这样就能够得到一组经过卷积核处理过后的28×28×6的图像   1. *# 第一层为卷积层* 2. self.conv1 = nn.Conv2d(num\_channel,6,5,pad\_mode = "valid")   第二层为池化层，这里选择的是池化尺寸大小为2×2的最大池化，并且同样不对其进行填充，最后经过下采样后的到的输出结果为14×14×6的一组图像。   1. *# 第二层为池化层,这里我们选用最大值池化，可以突出特征* 2. self.pool = nn.MaxPool2d(kernel\_size=(2,2),stride=2)   第三层为卷积层，使用5×5的卷积核，并且不对其进行填充，输入图像数为6，输出图像通道数为16，最后得到一组10×10×16的图像。   1. *# 第三层为卷积层* 2. self.conv2 = nn.Conv2d(6,16,5,pad\_mode="valid")   第四层为池化层，池化方式与第二层的池化方式相同，利用2×2的最大池化，得到的一组图像为5×5×16。  第五层为全连接层，要求输出的个数为120个，所以设置输出通道为120，权重的初始化方式为mindspore.common.initializer的Normal，即初始化一个正态数组，并从均匀分布中获取数值N(0, sigma)来填充输入张量。   1. *# 第五层为全连接层* 2. self.fc1 = nn.Dense(16\*5\*5,120,weight\_init=Normal(0.02))   第六层仍为全连接层，输入120×1的一个张量，输出的是84×1的张量。   1. *# 第六层仍为全连接层,输入为120，输出为84* 2. self.fc2 = nn.Dense(120,84,weight\_init=Normal(0.02))   第七层为输出层，通过全连接层来实现，输入的是84×1的张量，输出的是10×1的张量。   1. *# 第七层为输出层，输出十个概率值* 2. self.fc3 = nn.Dense(84,num\_class,weight\_init=Normal(0.02))   我们使用的激活函数为Relu激活函数。   1. *# 定义激活函数* 2. self.relu = nn.ReLU()   我们搭建好LeNet5的框架以后，下面开始构建整个LeNet5模型，具体的构建过程如下：   1. *# 神经网络搭建的过程* 2. def construct(self,gra): 3. gra = self.relu(self.conv1(gra)) 4. gra = self.pool(gra) 5. gra =self.relu(self.conv2(gra)) 6. gra = self.pool(gra) 7. gra = self.flatten(gra) 8. gra = self.relu(self.fc1(gra)) 9. gra = self.relu(self.fc2(gra)) 10. gra = self.fc3(gra) 11. return gra   根据刚才一步步搭建起来的框架，对输入的张量进行一步一步的处理，在每一层网络过后，都需要使用relu激活函数对每一层的输出结果进行一个激活处理，这样做的目的是考虑到线性模型表达能力的局限性，通过激活函数这样一个处理可以引入非线性因素，从而提升了神经网络模型的表达能力。  接下来，我们通过mindspore来定义callback回调函数，MindSpore自定义在train的过程中实时验证的回调函数，继承callback类自定义StepLossAccInfo，可以用来设置每隔几个epoch进行验证，实时输出model指定的metrics评价指标。然后通过实例化对象，将这个回调过程放入model.train()中的callback()中，简单来说就是可以实时的输出训练集训练的效果，我们利用回调函数来记录损失值和训练精度。   1. class StepLoss\_and\_Acc(Callback): 2. def \_\_init\_\_(self,model,eval\_dataset,step\_loss,step\_eval): 3. self.model = model 4. self.eval\_dataset =eval\_dataset 5. self.step\_loss =step\_loss 6. self.step\_eval=step\_eval 7. def step\_end(self, run\_context): 8. cb\_params = run\_context.original\_args() 9. cur\_epoch = cb\_params.cur\_epoch\_num 10. cur\_step = (cur\_epoch-1)\*1875 + cb\_params.cur\_step\_num 11. self.step\_loss["loss\_value"].append( 12. str(cb\_params.net\_outputs)) 13. self.step\_loss["step"].append(str(cur\_step)) 14. if cur\_step % 125 == 0: 15. acc = self.model.eval(self.eval\_dataset, 16. dataset\_sink\_mode=False) 17. self.step\_eval["step"].append(cur\_step) 18. self.step\_eval["acc"].append(acc["Accuracy"])   至此，我们所有的LeNet5神经网络模型的构建就已经全部完成了，接下来开始模型的训练部分。  设置训练轮次为1，设置momentum为0.9，momentum 动量是依据物理学的势能与动能之间能量转换原理提出来的。当 momentum 动量越大时，其转换为势能的能量也就越大，就越有可能摆脱局部凹域的束缚，进入全局凹域。momentum 动量主要用在权重更新的时候。设置学习率lr为0.01，损失函数我们使用softmax交叉熵损失函数，使用稀疏标签的格式，并且一均值作为其损失类型，以准确率来作为评价标准。   1. network = LeNet\_5() 2. loss\_fun = SoftmaxCrossEntropyWithLogits(sparse=True, 3. reduction="mean") 4. mom\_fun = nn.Momentum(network.trainable\_params(), 5. learning\_rate = 0.01,momentum = 0.9) 6. model\_2 = Model(network=network,loss\_fn=loss\_fun,optimizer=mom\_fun, 7. metrics={'Accuracy':Accuracy()})   然后，对数据集进行加载，对其进行初始化处理，最后开始训练模型，调用model.train这个函数的接口对其进行训练，即可得到最终的模型。   1. traindata = create\_dataset("./MNIST\_Data/train/") 2. testdata = create\_dataset("./MNIST\_Data/test/") 3. config\_ck = CheckpointConfig(save\_checkpoint\_steps=375, 4. keep\_checkpoint\_max=16) 5. ckpoint\_cb = ModelCheckpoint(prefix="LeN\_four", 6. directory="./lenet4\_11-20/",config=config\_ck) 7. step\_loss\_acc = StepLoss\_and\_Acc(model\_2,testdata,steps\_loss 8. ,steps\_eval) 9. model\_2.train(epoch=1,train\_dataset=traindata,callbacks=[ckpoint\_cb,LossMonitor(125),step\_loss\_acc],dataset\_sink\_mode=False)   **3.3 模型测试**  我们使用load\_checkpoint和load\_param\_into\_net将训练好的模型加载出来，利用测试集来对其进行一个测试，测试数据为从测试集当中选择一个batch的数据来进行预测，最后将预测值和准确值进行一个对比，对结果进行一个分析。   1. mnist = create\_dataset\_Alexnet("./MNIST\_Data/test/") 2. ds\_test = mnist.create\_dict\_iterator() 3. data = next(ds\_test) 4. imgs = data['image'].asnumpy() 5. labels = data['label'].asnumpy() 6. plt.figure() 7. for i in range(1,33): 8. plt.subplot(4,8,i) 9. plt.imshow(imgs[i-1][0],interpolation='None',cmap = 'gray') 10. plt.axis(False) 11. output = model\_2.predict(Tensor(data["image"])) 12. predicted = np.argmax(output.asnumpy(),axis=1) 13. print("预测值为：",predicted) 14. print("准确值为：",data['label'].asnumpy()) |
| 1. **算法流程图**   **wps** |
| **五、实验设置**  **5.1 实验工具**  Mindspore 1.8、win10系统、模型训练在CPU R7-4800H上完成，实验平台为jupyter notebook、编程语言 python3.8  **5.2 算法参数设置**  **（1）图像处理**  图片初始化为32×32的图像，并对像素值进行标准化、归一化处理，增加模型的鲁棒性，处理前后数据集的对比：  data  由于本次模型训练的数据集由Mnist提供，本身训练集就比较多，多达六万张，故没有对其进行旋转、翻转等过多的数据增强操作。   1. **全连接层权重初始化**   三个全连接层模型的权重初始化使用N(0,0.02)的均匀分布来对数组进行初始化，全连接层权重初始化很重要，权重过小有可能导致最终的模型不收敛，权重过大有可能导致梯度爆炸。  比如，当我设置为N(0,0.2)的均匀分布来对全连接层的权重进行初始化时，训练得到的loss\_acc图像为：  acc_loss_0.2  由图像可知，很明显已经产生了一定的梯度爆炸，从准确率上面来看，虽然整体上是收敛的，但是局部区域的收敛性并不强，有一定的波动性，模型的鲁棒性较差。  又比如，当我设置为N(0,0.002)的均匀分布来对全连接层的权重进行初始化时，训练得到的loss\_acc图像为：  acc_loss0.002  从图像上来看，模型的损失值维持在2.30左右附近，这是一个较高的损失值了，模型的精度值只是维持在0.114这个非常低的值附近，很明显的可以看出图像并不收敛，这就是因为权重设置的值太小而导致的，由此可知，全连接层权值的设置是至关重要的，它影响了模型的训练效果。   1. **训练速率**   我们定义训练轮数为1，并且将六万张图片分为1875个batch，每个batch有32张图片，定义学习速率lr=0.01，学习速率的快慢影响了模型的收敛速率，定义momentum为0.9，momentum 动量是依据物理学的势能与动能之间能量转换原理提出来的。当 momentum 动量越大时，其转换为势能的能量也就越大，就越有可能摆脱局部凹域的束缚，进入全局凹域。momentum 动量主要用在权重更新的时候。简单的来说，就是可以加快模型的训练速率。使用softmax交叉熵函数来作为损失函数，利用训练结果的均值来计算损失值。     1. **硬件层面**   使用cpu来对模型进行训练。  **5.3 使用Lenet4模型解决数字手写体识别问题**  Lenet4 模型和Lenet5模型的区别不大，只是比Lenet5少了一层全连接层。因此，我们构建的过程中，只需要把这个全连接层去掉就可以了，Lenet4的模型框架图如下：  IMG_256  我们根据其框架图来构建Lenet4模型，将含有84个通道的全连接层删去，再构建一个输入为120个通道，输出为10个通道的全连接层作为输出层即可，具体代码如下：   1. class LeNet\_4(nn.Cell): 2. def \_\_init\_\_(self,num\_class = 10,num\_channel=1): 3. super(LeNet\_4,self).\_\_init\_\_() 4. *# 第一层为卷积层* 5. self.conv1 = nn.Conv2d(num\_channel,6,5,pad\_mode = "valid") 6. *# 第二层为池化层,这里我们选用最大值池化，可以突出特征* 7. self.pool = nn.MaxPool2d(kernel\_size=(2,2),stride=2) 8. *# 第三层为卷积层* 9. self.conv2 = nn.Conv2d(6,16,5,pad\_mode="valid") 10. *# 第五层为全连接层* 11. self.fc1 = nn.Dense(16\*5\*5,120,weight\_init=Normal(0.02)) 12. *# 第六层为输出层，输出十个概率值* 13. self.fc3 = nn.Dense(120,num\_class,weight\_init=Normal(0.02)) 14. *# 定义激活函数* 15. self.relu = nn.ReLU() 16. self.flatten = nn.Flatten() 17. *# 神经网络搭建的过程* 18. def construct(self,gra): 19. gra = self.relu(self.conv1(gra)) 20. gra = self.pool(gra) 21. gra =self.relu(self.conv2(gra)) 22. gra = self.pool(gra) 23. gra = self.flatten(gra) 24. gra = self.relu(self.fc1(gra)) 25. gra = self.fc3(gra) 26. return gra   利用该模型来对Mnist数据集进行一个训练，然后对该训练结果和Lenet5训练结果进行一个对比，分析其差异性。  **5.4 试用Alexnet模型解决数字手写体识别问题**  AlexNet网络结构相对与Lenet5模型来说复杂了些，使用了8层卷积[神经网络](https://so.csdn.net/so/search?q=%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/guzhao9901/article/details/_blank)，前5层是卷积层，剩下的3层是全连接层，而且图像的尺寸大小需要resize到227×227的大小，具体如下所示：    值得注意的一点：原图输入224 × 224，实际上进行了随机裁剪，实际大小为227 × 227。  **（1）卷积层C1**  C1的基本结构为：卷积–>ReLU–>池化  卷积：输入227 × 227 × 3，96个11×11×3的卷积核，不扩充边缘padding = 0，步长stride = 4，因此其FeatureMap大小为(227-11+0×2+4)/4 = 55，即55×55×96;  激活函数：ReLU；  池化：池化核大小3 × 3，不扩充边缘padding = 0，步长stride = 2，因此其FeatureMap输出大小为(55-3+0×2+2)/2=27, 即C1输出为27×27×96（此处未将输出分到两个GPU中，若按照论文将分成两组，每组为27×27×48）；  **（2）卷积层C2**  C2的基本结构为：卷积–>ReLU–>池化  卷积：输入27×27×96，256个5×5×96的卷积核，扩充边缘padding = 2， 步长stride = 1，因此其FeatureMap大小为(27-5+2×2+1)/1 = 27，即27×27×256;  激活函数：ReLU；  池化：池化核大小3 × 3，不扩充边缘padding = 0，步长stride = 2，因此其FeatureMap输出大小为(27-3+0+2)/2=13, 即C2输出为13×13×256（此处未将输出分到两个GPU中，若按照论文将分成两组，每组为13×13×128）；  **（3）卷积层C3**  C3的基本结构为：卷积–>ReLU。  注意：此层没有进行MaxPooling操作。  卷积：输入13×13×256，384个3×3×256的卷积核， 扩充边缘padding = 1，步长stride = 1，因此其FeatureMap大小为(13-3+1×2+1)/1 = 13，即13×13×384;  激活函数：ReLU，即C3输出为13×13×384（此处未将输出分到两个GPU中，若按照论文将分成两组，每组为13×13×192）；  **（4）卷积层C4**  C4的基本结构为：卷积–>ReLU。  注意：此层也没有进行MaxPooling操作。  卷积：输入13×13×384，384个3×3×384的卷积核， 扩充边缘padding = 1，步长stride = 1，因此其FeatureMap大小为(13-3+1×2+1)/1 = 13，即13×13×384;  激活函数：ReLU，即C4输出为13×13×384（此处未将输出分到两个GPU中，若按照论文将分成两组，每组为13×13×192）；  **（5）卷积层C5**  C5的基本结构为：卷积–>ReLU–>池化  卷积：输入13×13×384，256个3×3×384的卷积核，扩充边缘padding = 1，步长stride = 1，因此其FeatureMap大小为(13-3+1×2+1)/1 = 13，即13×13×256;  激活函数：ReLU；  池化：池化核大小3 × 3， 扩充边缘padding = 0，步长stride = 2，因此其FeatureMap输出大小为(13-3+0×2+2)/2=6, 即C5输出为6×6×256（此处未将输出分到两个GPU中，若按照论文将分成两组，每组为6×6×128）；  **（6）全连接层FC6**  FC6的基本结构为：全连接–>ReLU–>Dropout  全连接：此层的全连接实际上是通过卷积进行的，输入6×6×256，4096个6×6×256的卷积核，扩充边缘padding = 0, 步长stride = 1, 因此其FeatureMap大小为(6-6+0×2+1)/1 = 1，即1×1×4096;  激活函数：ReLU；  Dropout：全连接层中去掉了一些神经节点，达到防止过拟合，FC6输出为1×1×4096；  **（7）全连接层FC7**  FC7的基本结构为：全连接–>ReLU–>Dropout  全连接：此层的全连接，输入1×1×4096;  激活函数：ReLU；  Dropout：全连接层中去掉了一些神经节点，达到防止过拟合，FC7输出为1×1×4096；  **（8）全连接层FC8**  FC8的基本结构为：全连接–>softmax  全连接：此层的全连接，输入1×1×4096;  softmax：softmax为1000，FC8输出为1×1×1000；  其代码编写如下：   1. *#AlexNet* 2. class AlexNet(nn.Cell): 3. def \_\_init\_\_(self): 4. super(AlexNet,self).\_\_init\_\_() 5. self.conv1=nn.Conv2d(1,96,11,stride=4,pad\_mode="valid") 6. self.conv2=nn.Conv2d(96,256,5,pad\_mode="same") 7. self.conv3=nn.Conv2d(256,384,3,pad\_mode="same") 8. self.conv4=nn.Conv2d(384,384,3,pad\_mode="same") 9. self.conv5=nn.Conv2d(384,256,3,pad\_mode="same") 10. self.maxpool\_2d=nn.MaxPool2d(kernel\_size=3,stride=2) 11. self.flatten=nn.Flatten() 12. self.fc1=nn.Dense(6\*6\*256,4096) 13. self.fc2=nn.Dense(4096,4096) 14. self.fc3=nn.Dense(4096,10) 15. self.relu=nn.ReLU() 17. def construct(self,gra): 18. gra=self.relu(self.conv1(gra)) 19. gra=self.maxpool\_2d(gra) 20. gra=self.relu(self.conv2(gra)) 21. gra=self.maxpool\_2d(gra) 22. gra=self.relu(self.conv3(gra)) 23. gra=self.relu(self.conv4(gra)) 24. gra=self.relu(self.conv5(gra)) 25. gra=self.maxpool\_2d(gra) 26. gra=self.flatten(gra) 27. gra=self.relu(self.fc1(gra)) 28. gra=self.relu(self.fc2(gra)) 29. gra=self.fc3(gra) 30. return gra   与原始的[LeNet](https://blog.csdn.net/guzhao9901/article/details/118518440" \t "https://blog.csdn.net/guzhao9901/article/details/_blank)相比，AlexNet网络结构更深，LeNet为7层，AlexNet为8层。在随后的神经网络发展过程中，AlexNet逐渐让研究人员认识到网络深度对性能的巨大影响。 |
| **六、实验结果**  **6.1 LeNet5**  训练过程以及中间结果如下：    训练过程中每间隔125个batch就打印一次损失值，根据打印数据我们可以看出，随着模型训练次数的增加，模型的损失值也越来越小，损失精度值逐渐收敛于零，我们再看一下训练过程的模型精度值的变化情况：    同样可以看到随着训练次数的增加，模型精度也越来越高，逐步趋近于100%，达到饱和状态，我们根据当前所获得的数据绘制该模型训练过程中的损失值变化情况和精度变化情况如下图：  **acc_loss**  由图中可见，损失值和精度值呈现明显的反比例关系，说明模型训练相对比较成功，在训练的前800轮左右，模型的损失值维持在相对较高的位置，在800轮以后，模型开始快速收敛，损失值极具减小，模型精度也呈现陡崖式增高。  **6.2 Lenet4**  从打印结果上来看，利用该模型框架进行手写体识别进行训练，其损失值也是收敛的，平且相较Lenet5收敛速度快很多。    模型准确率也可以达到97%左右：    最后我们根据数据画出损失值和准确率的图像如下：  acc_loss_lenet4  **6.3 Alexnet**  训练过程打印的损失值结果如下：    准确率可以高达98%，而且相对于Lenet4模型来说，收敛更快。    将准确率和损失值在同一图像中画出，图像如下：  acc_loss_Alexnet  由图中可以看出，Alexnet模型的收敛性更好，收敛速度更快，得到的模型同样可以用于手写体识别。  **6.4 模型测试**  首先将已经训练完成的模型加载出来，利用test集当中一个batch（32张）的图像，来对其进行一个预测，测试的数据集如下：  test  利用模型对这一组图片进行一个预测，将预测值和精确值全部打印出来，进行结果对比：    根据结果可以看出，预测出来的结果是完全正确的。  在利用第二组数据来对其进行测试，测试结果如下：  IMG_256  从这次的测试结果中可以看出，在32张图片中，预测值有两个值和准确值不同，说明在预测过程中，即使准确率可以高达97%，但预测结果仍然存在一定的误差，这是没办法避免的。 |
| **七、实验结果分析**  **7.1 训练结果分析**  下面三个图分别是Lenet5和Lenet4、Lenet5和Alexnet的对比图，从对比图中可以发现，Alexnet模型的收敛速度是最快的，但同时也是最不稳定的，前期训练时损失的波动较大，而且训练时间是最长的，lenet5模型是里面收敛最慢的一个训练集将近过半是才开始梯度下降到逐渐收敛，三者之间的训练模型的准确率都差不多，都维持在百97%左右，达到了一般模型训练准确率的精度要求，综合来看，Lenet4在本次实验中表现是最为出色的一个模型。  compare  下面是三个模型的准确率的对比图，在图中很容易就可以看出，Lenet4和Alexnet模型的收敛速度都是比较快的，这两个模型的准确率变化情况都差不多，而lenet5相较于这两种收敛速度较慢，但他们最终的准确率都相差不多。  acc_line  **7.2 模型的选择**  在训练过程中，lenet4和lenet5模型训练过程都非常的快，只用了不到一分钟的时间，就可以将模型的训练精度高达97%左右，这样一个精确度是非常可观的。但对于Alexnet模型来说，由于Alexnet模型的深度为8层，而且所处理的图像分辨率也很大，再加上模型的数据集数量十分庞大，导致其训练过程所耗费的时间较长，仅仅只是一轮的模型训练过程就用了一个小时的时间，所以，选择模型需要从多个角度来对其进行考虑，最对某个问题选择最适合的模型来对其进行训练，才能够得到最好的结果。  **7.3 深度学习的输入数据集要做均值和标准差的处理的原因**  可以达到防止某一维或某几维对数据影响过大，同时有抗异常值的能力，比较稳定，适合嘈杂的数据场景。首先可以做归一化，可以防止某一维或某几维对数据影响过大，rgb三通道里，比如r通道里，某个像素值a在0-10之间变动，而另一个像素值b在10-255之间变动，那b像素造成的影响会掩盖像素a的影像，归一化就缓解了这种现象。归一化后，模型也会容易受到最大值和最小值的影响，因此如果数据集中存在一些异常点，结果将发生很大的改变，因此这种方法鲁棒性（稳定性）比较差，只适合数据量比较精确，比较小的情况。从而需要进行标准化。  **X= (x-μ)/σ**  常用的方法是z-score标准化，经过处理后的数据均值为0，标准差为1，满足标准正太分布，其中μ是样本的均值，σ是样本的标准差，它们可以通过现有的样本进行估计，在已有的样本足够多的情况下比较稳定，适合嘈杂的数据场景。 |
| 1. **总结**   **8.1 过程总结**  本次设计报告我完成了针对数字手写体模型的几种典型的卷积神经网络模型的训练，其中包括Lenet5、Lenet4和Alexnet三个模型的训练和测试，来完成对手写体识别这个功能的测试，三个模型各自都有各自的特点，使用mnist中的数据集来对其进行训练，mnist数据集中封装了七万张数字手写体的图片，尺寸大小都为28\*28，其中六万张用来做训练集，剩下的一万张用来做测试集，训练数据相对较多，可以有效的避免在训练过程中由于数据集不足而欠拟合的情况，但是反而需要担心的是，需要避免数据训练过程中的过拟合，而导致模型精度下降，然后使用Mindspore API 对三个不同的模型来进行逐步的搭建，Mindspore虽然和pytorch和tensorflow有很多相似之处，但是刚上手学习是还是有些困难这里要感谢华为Mindspore的官方教程，下面是链接：  **https://www.bilibili.com/video/BV1CS4y1z72r/?spm\_id\_from=333.337.search-card.all.click&vd\_source=a6e37c8b1ee9ae140546b5c201fca8ae**  让我能够快速的入门Mindspore API，顺利的完成本次的设计实验。  Lenet5模型是一个七层的网络模型，他的大致流程为：  **卷积层-->池化层-->卷积层-->池化层-->全连接层-->全连接层-->全连接层**  所以只要按照这个流程一层一层的搭建模型就可以将lenet5的模型框架搭建好，但是其中要注意的是，在每一个卷积层过后都需要对其使用激活函数进行一个激活，这里我们使用relu函数对其进行一个激活，增加神经网络模型的非线性表达能力，其次，就是全连接层权重的初始化，权重过大可能会导致梯度爆炸，而权重过小有可能会导致模型不收敛，所以我们在构建全连接层的时候需要选择合适的权重，但其实大多时候权重值都被初始化为零了，就比如训练Alexnet模型的过程中全连接层的权重就全部被初始化为了0，卷积层的步数，卷积核的大小，选择池化的方式，最大池化或者均值池化，选用什么样的激活函数等等这些因素都是可以考虑的地方，构建模型过程中超参数的选择是我们尤其需要注意的地方。  搭建Lenet4模型的时候相对来说就比较简单了，有了Lenet5模型的基础，我们只需要在这个基础上更改一下就可以，Lenet4和Lenet5唯一的差别就是Lenet4少了一个全连接层，只需要把倒数第二层的全连接层去掉，然后更改一下输入和输出通道个数即可完成网络模型的搭建。  最复杂的就是Alexnet模型的搭建，首先在数据集的处理上就和前两个模型有一些差别，Alexnet模型的输入图片的分辨率大小为227\*227，所以需要将28\*28的图像resize成227\*227的图像，同样需要对其做标准化处理操作，模型为八层的一个网络模型，在构建过程中明显可以感觉到这个模型的计算量很大，果然在训练过程中训练的时间是最长的。  其实实验当中我还有很多模型的参数可以调整，但是由于模型精度本身就已经很高了，所以有些参数就没有再改，因为想把模型的精度从97%再上升一个高度已经是很难的了，更改模型的参数有可能反而会使模型的精度降低，反而造成了不好的影响。  **8.2 实验的不足之处**  在本次实验当中，我认为有些地方做的还并不是特别好，最主要的一点就是模型的交互性太差了，没有实现GUI界面的设计，通过GUI实现用户通过鼠标手写一个0-9的数字，格式转换成指定格式的图片，然后放入神经网络模型当中进行一个预测，最后将预测结果返还给用户端，这是我接下来仍需要实现的部分，但是由于我对这方面知识有一定的欠缺，所以目前还没有能够实现这个模块，在后面的学习中，我会努力学习这方面的知识，完善本次的设计实验，然后在利用一些计算机网络的知识，将手写体识别部署到服务器上，真正实现模型的投入使用。 |
| * **感想、体会、建议∶**   本次实验过程我学到了很多，特别是对卷积神经网络这一部分，让我从头到尾的对图像处理领域的卷积神经网络模型有了一个深刻的认识，实验过程中，只要有一个环节的内容和知识，不了解，就没有办法搭建出一个成功的神经网络模型，之前有人说过卷积神经网络的搭建实际上就是一个**搭积木**的过程，这回我确实是真正体会到了这个过程，你需要根据**模型的框架图**一步一步的搭建**卷积层、池化层和全连接层**，理解了这个过程，甚至可以搭建自己的神经网络，你可以根据具体的问题来对神经网络模型的结构和参数进行一个修改，然后不断的尝试，最后得到一个针对该问题的最合适的模型。  我是从今年2月份入手**[机器学习](https://so.csdn.net/so/search?q=%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_41401843/article/details/_blank)**，当时是因为学校开设了一门机器学习的课程，那时只是初步的了解，后面就没有继续深入学习，知道这学期开设了人工智能这门课程，我又重新拾起了这些知识，进入**卷积神经网络**阶段，并由此对计算机视觉起了较大的兴趣。卷积网络入门其实没有想象的那么难，但是越往高处走越来越需要我们去细心琢磨。在卷积神经网络中，有必要去深入了解一些经典的卷积神经网络。由浅入深，例如**LeNet、AlexNet、VGG16、Inception网络、ResNet网络**等等。在了解这些网络的时候应该多考虑别人为什么要这么做，这样做又什么好处。例如VGG16中有大量的3\*3卷积核，加入较多的3\*3卷积核在感受野不变的情况下能够缩小模型参数量。在学习卷积神经网络的过程中，应该较多的做好一些**实践**，这样不仅仅能够让你更加深入的去理解网络的构成流程而且能让你产生更大的兴趣去学习去钻研。在学习卷积网络的过程中，应该适当的去修改一些**经典网络**，看看在修改过后有什么的不同，在从这些不同的地方吸取一些教训，然后在以后碰到类似的问题的时候能够有一些经验。  通过这次学习，我也加深了对**深度学习**的理解，现在深度学习在[机器学习](https://so.csdn.net/so/search?q=%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/shirley67269/article/details/_blank)领域是一个很热的概念，不过经过各种媒体的转载播报，这个概念也逐渐变得有些神话的感觉：例如，人们可能认为，深度学习是一种能够模拟出人脑的神经结构的机器学习方式，从而能够让计算机具有人一样的智慧；而这样一种技术在将来无疑是前景无限的。深度学习是机器学习领域中对**模式（声音、图像等等）**进行建模的一种方法，它也是一种基于统计的概率模型。在对各种模式进行建模之后，便可以对各种模式进行识别了，例如待建模的模式是声音的话，那么这种识别便可以理解为语音识别。  深度学习中的“**深度**”一词，人们从感性上可能会认为，深度学习相对于传统的机器学习算法，能够做更多的事情，是一种更为“高深”的算法。而事实可能并非我们想象的那样，因为从算法输入输出的角度考虑，深度学习算法与传统的有监督机器学习算法的输入输出都是**类似的**，无论是最简单的LogisticRegression，还是到后来的SVM、boosting等算法，它们能够做的事情都是类似的。深度学习的学名又叫**深层[神经网络](https://so.csdn.net/so/search?q=%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/shirley67269/article/details/_blank)**，是从很久以前的**人工神经网络模型**发展而来。  如今随着**硬件**的高速发展，深度学习也席卷而来，这是十几年前的硬件条件想都不敢想的事情，如果以后研究生有机会的话，希望我可以深入学习这方面的知识。 |
| **上机报告成绩、评语∶**  指导教师签名：  年 月 日 |