**《深度学习基础》课程报告**

**课程编码 U03L11008.01**

**姓 名**

**学 号**

**院 系 xxxx学院**

**授课教师 xxxxx**

**日 期 2024年1 月**

本报告的模板仅用于《深度学习基础课程》的大作业使用，报告的章节内容和相关排版格式可由撰写报告人适当增删，但是核心的关键信息，比如模型的架构、算法的设计、模型的调试和分析等基本环节需要写入本报告中。

目录

[一、分类问题描述 1](#_Toc124254523)

[二. 分类方案 1](#_Toc124254524)

[三、数据集处理流程 2](#_Toc124254525)

[四、深度模型结构描述 4](#_Toc124254526)

[五、分类精度展示 9](#_Toc124254527)

[六、结论 10](#_Toc124254528)

### 一、分类问题描述

轴承故障诊断数据集（CWRU数据集）由美国西储大学收集并公开，已经被广泛用于数字信号方面以及深度学习方面的研究工作。CWRU数据集是将加工过的故障轴承重新装入测试电机中，分别在0、1、2和3马力的电机负载工况工作条件下记录振动加速度信号数据。利用该实验台获取的8个正常样本，53个外圈损伤样本，23个内圈损伤样本及11个滚动体损伤样本。本次问题的数据集采用了其中10个数据样本。

CWRU数据集共包括了10个数组，每个数组中由78400个数字组成，并用mat类型文件组成。每个数组分别被放在10个文件夹中，每个文件夹的命名即为数据标签名。

任务描述：

1. 编写函数进行数据分割，把数据长度为784的100个样本。对每一类故障建立100个样本，10类总计1000个样本。
2. 根据Pytorch数据集的设计规则。设计CWRU数据集
3. 建立深度学习分类模型，至少包括4层神经网络，Epoch不大于100。实现对轴承故障分类。

### 二. 分类方案

基于课程中的问题分解思路，拟定本问题的解决方案如图 1所示，详细步骤如下：

(1) 构建数据集。通过io.loadmat导入mat数据集。因为数据较少，因此先建立文件标签再按批次提取。可以一次性将所有数据导入。导入后将数据集分为1000份，作为1000个样本数据。并将每个样本整理为28\*28的二维数组，并进行归一化。方便后续分类。而Pythorch数据集的定义方法，本文将在下一节中具体描述。

(2) 深度学习网络构建。数据样本可以认为是28\*28的灰度图片。因此采用CNN网络进行分类。本文构建网络的思路，先设计一个足够深的CNN网络（防止欠拟合）。然后通过loss曲线和acc曲线结果，对网络的参数及结构进行针对性优化。该过程将在后文进行具体阐述。

(3) 网络与超参数调节。通过loss和acc曲线的结果，分析估计数据的梯度分布情况。调整网络的超参数和网络参数到合适值，得到最优模型

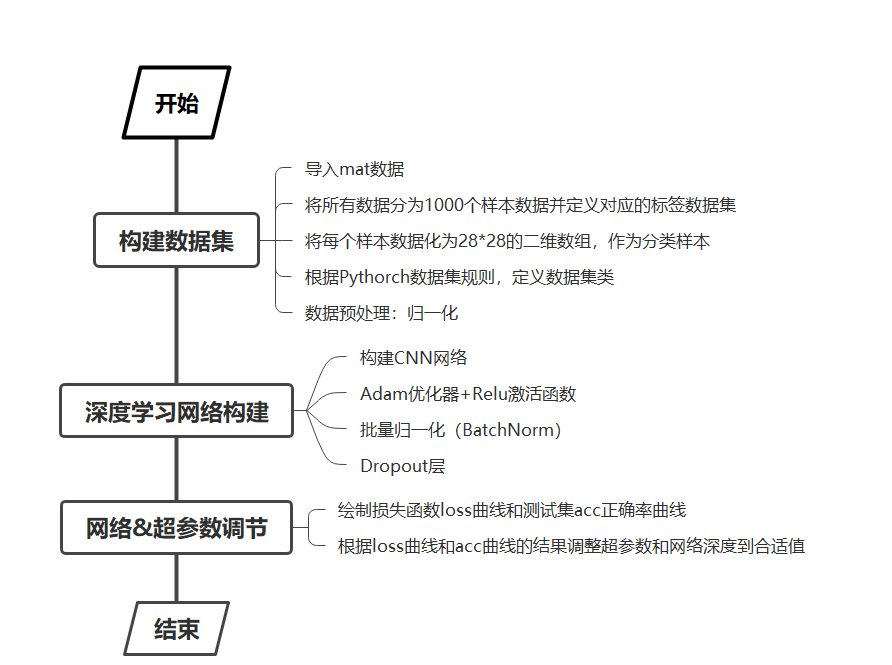


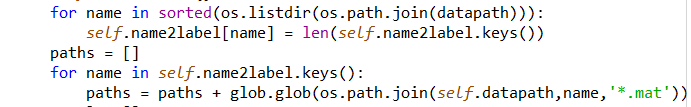
图 1 分类问题的解决方案

### 三、数据集处理流程

数据集的处理及Pytorch中数据类的构建总共分为以下几个步骤：

(1) 获取所有数据的文件路径。为了保证程序在标签改变、数据量改变的情况下也可以正常运行。首先提取文件路径下的变量路径。以此得到标签和变量数量。其核心实现代码如下

(2) 根据获取的文件路径，分离出标签数据。并导入mat文件数据。并将数据打乱。以方便后续的数据读取。

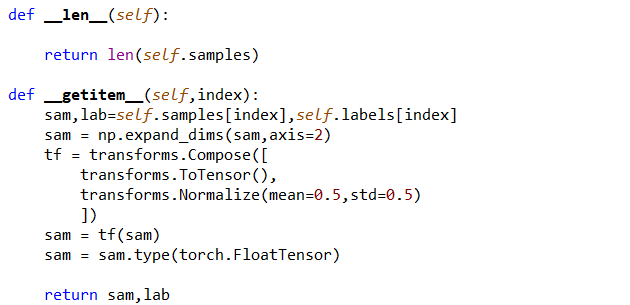
****



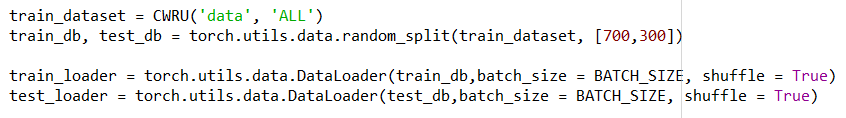
若不在\_\_init\_\_内部打乱数据，而只在data\_loader时将shuffle=True的话。会导致提取出的数据集集中。例如若按训练集和测试集7：3的比例设置数据的话。若不内部打乱，会导致训练集只有数据中的前7类标签的数据，测试集只有后3类的数据，导致两个数据集数据割裂。因此需要内部提前打乱。

(3) 输入样本变换。**在第二步中，我们以784为步长，从原数据中提取了1000个数据样本。**并导入了这些样本对应的标签数据。接下来，需要将每个样本里的784\*1的数组转化为28\*28的二维数组。由此可以把样本作为一个单通道图片来进行分类。方便导入卷积网络模型。

(4) 构建数据类。**根据Pythorch数据集规则，在类中添加\_\_len\_\_函数和\_\_getitem\_\_函数**，并将数据进行归一化处理。

****

(5) 数据读取。创建类以后，将全部信息读取出来。利用torch中的random\_split进行随机切片。训练集和测试集的数据比例为7：3。（这里如果样本的数据总量变了，需要重新设置rand\_split中的参数）

****

### 四、深度模型结构描述

#### 4.1 网络构建代码解释

图 2为网络构建的代码,其中变量par中的即为网络的参数。

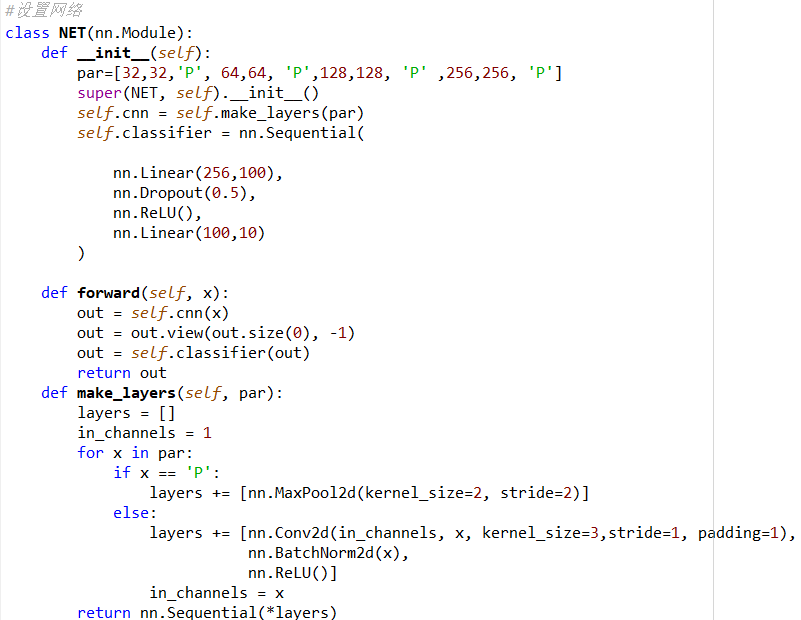


图 2 深度网络的设计代码

1. \_\_init\_\_函数的作用为构建每一层的网络，self.cnn中为卷积层部分，self.classifier为分类器（全连接层部分）。nn.Linear为全连接层，第一个参数为输入参数数量，第二个参数为输出参数数量。nn.Dropout为加入一个Dropout层，神经节点失活概率为50%。
2. \_\_forward\_\_函数为前向传播函数，其中self.cnn是返回卷积层结果值，torch.view的作用是把一个多维的二维数组的形状调整为一个维数不变的一维数组，以保证其可以顺利的写入到全连接网络里；
3. make\_layer函数的作用是按照par中的参数构建卷积层网络。下面对par中的参数作用进行说明。

* 当par中的参数为数字时。构建一个卷积层，卷积核大小为3\*3。维数即为par中的数字大小。每个卷积层后面增加一个比正则化（BN层）和ReLU激活函数。同时，修改输入通道值为par中的参数值，保证和下一层的卷积层的输入一致。
* 当par中的参数为‘P’时，加入一层最大池化层。Kernel=2\*2。

以上仅为代码解释部分，具体的设计准则和设计原因将在后文具体阐述。

#### 4.2 网络结构分析

###### 4.2.1 网络结构概述

最后的网络结构如下面的参数集所示。本文所用网络共包括了8层卷积层，2层全连接层，8层BN层，1层Dropout层。激活函数为ReLU。

NET(

(cnn): Sequential(

(0): Conv2d(1, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU()

(3): Conv2d(32, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(5): ReLU()

(6): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(7): Conv2d(32, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(9): ReLU()

(10): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(11): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(12): ReLU()

(13): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(14): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(15): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(16): ReLU()

(17): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(18): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(19): ReLU()

(20): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(21): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(22): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(23): ReLU()

(24): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(25): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(26): ReLU()

(27): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

)

(classifier): Sequential(

(0): Linear(in\_features=256, out\_features=100, bias=True)

(1): Dropout(p=0.5, inplace=False)

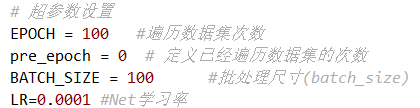
(2): ReLU()

(3): Linear(in\_features=100, out\_features=10, bias=True)

)

)

此外，优化器选择Adam算法，损失函数为交叉熵，相关超参数设如下：



###### 4.2.2 网络设计要点

* 网络深度选择:从上面的网络结构可以看出，对于一个28\*28的分类样本数据而言。本报告设计的网络结构层度较深。其主要原因包括：

1. 数据集足够的小，计算量小。可以支持参数较多、层数较深的模型。
2. 卷积层越多，能提取的特征就越多。网络潜在的泛化能力就越强，可训练空间越大。不用担心因为欠拟合导致精度过低。
3. 过拟合的情况可以通过loss曲线和acc曲线明显地看出来。（损失函数很小，但是测试机精度较低），且防止过拟合的策略很多；而欠拟合下的结果（损失函数和精度都很低）的可能原因有很多，不好分析。

当然，过深的网络也可能会出现网络退化、梯度爆炸等问题。所以网络的深度也并不能过深。这里本文参考了几种常用的深度学习模型的层数和它们处理的图像大小。最终设计了一个10层的深度网络，其中8层卷积层、2层全连接层。

* 卷积层参数设计：可以看出来我们在卷积层里多次用了相同维度的卷积层进行了连续两次卷积。使用的卷积核都是3\*3大小的卷积核。参考VGG系列网络。2个3\*3的卷积核可以用一个5\*5的卷积核代替。而之所以使用两层同纬度的卷积网络。主要目的在提高网络的非线性程度，两个3\*3网络的比一个5\*5的网络参数要多的多。由此来提高网络的性能。而对于卷积层的维度（或者说每一层卷积核的个数），维数越大能提取的特征就越多。因此在一个合理范围内，本文尽量提高了每一层的维数。
* 防止梯度爆炸/梯度消失/梯度饱和/的一些方法：加入BN层、调整学习率，Batch\_size，学习率权重衰减等参数、使用Adam优化（自适应学习率）而不是Sgd、使用ReLU作为激活函数等。
* 防止过拟合的一些方法：

1. 对最优模型（最大测试集精度数据）进行保存，在做最终测试时调用的是最优模型，而不是最后一次epoch训练得到的模型。
2. 全连接层里加入Dropout层，使得随机部分神经节点失活，防止过拟合。

#### 4.3 超参数调节方法

下面分析了一些在调试过程中的典型情况（loss曲线和acc曲线）和相应的解决方案。

1. 如图 3所示，训练过程收敛速度很快，但是精度和loss很低且震荡。

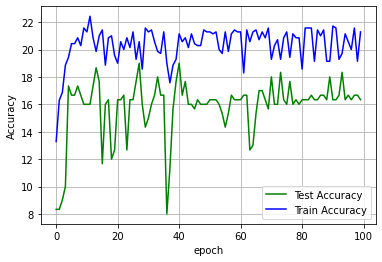
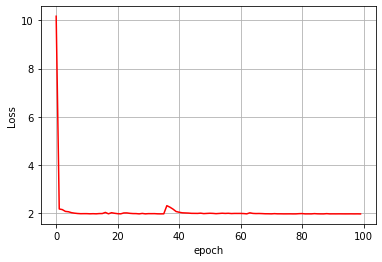
 

图 3 深度网络的设计代码

**分析原因**：学习率太大，到不了最优解，且在最优解附近来回震荡。

**解决措施**：降低学习率

1. 如图 4所示，收敛速度慢，acc和loss缓慢上升

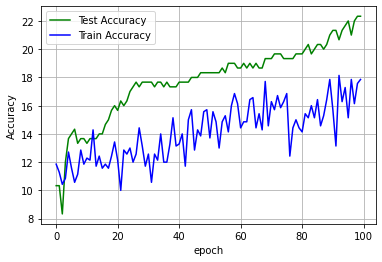
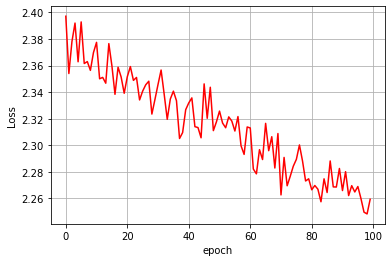
 

图 4 深度网络的设计代码

**分析原因**：学习率太小，网络收敛慢或不收敛。

**解决措施**：提高学习率

1. 如图 5所示，acc和loss来回跳动，起伏很大

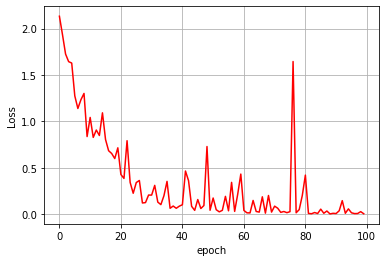
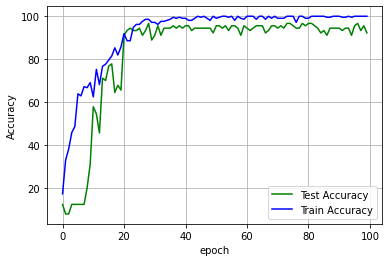


图 5 深度网络的设计代码

**分析原因：**这种情况的出现原因有很多，一种是学习率太大了，导致参数在最优解附近来回徘徊。还有一个很有意思的原因，是Batch\_size的值不能整除训练样本的数量。

比如说，当训练样本为210时，若把Batch\_size设置成100，则一个epoch内，网络会计算三次梯度，前两次都是用100个样本计算梯度。而第三次却只用了10个样本计算梯度！因此，第三次计算的梯度结果十分的不可靠，所以loss曲线才会出现大幅度的震荡。如果把Batch\_size调整为105的话，该情况会有很大改善。

**解决措施：**适当减小学习率，调整Batch\_size的值

1. 如图 6所示，acc精度高，但是loss曲线的收敛速度慢

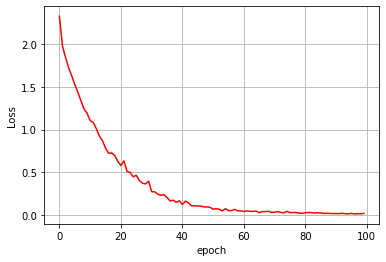
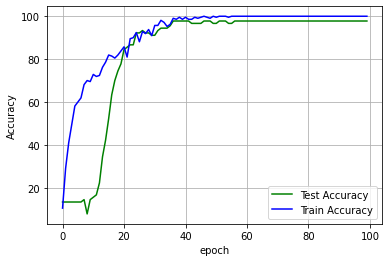


图 6 深度网络的设计代码

**分析原因：**刚开始的时候，学习率过低，收敛速率慢。但是收敛精度高。需要更多的epoch才能达到目标精度。

**解决措施：**采用学习率衰减措施，在epoch高于某一值时，降低学习率的大小。

### 五、分类精度展示

图 7展示了1000个样本数据下的loss曲线和acc曲线结果。精度达99.6%。epoch在20次以内即可收敛，深度模型实现了较高的分类精度。

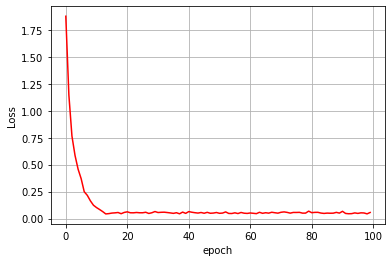
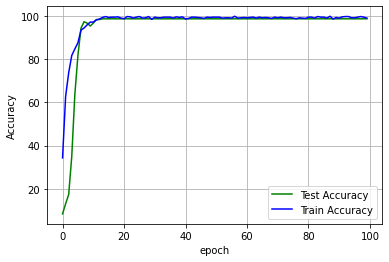


图 7 深度网络的设计代码

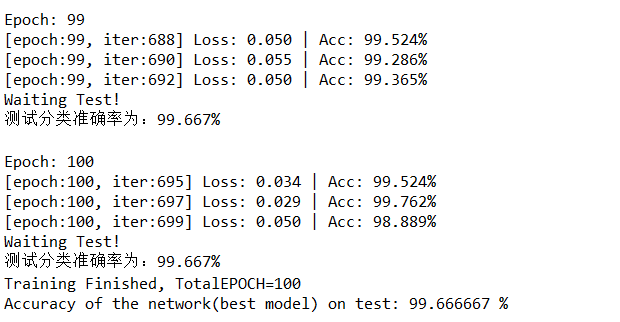


图 8展示了300个样本数据下的loss曲线和acc曲线结果。精度达98.8%。epoch在30次以内即可收敛，本报告提出的深度模型即使在少样本下，也能实现较高精度，表明提出的模型具有较优的泛化性能。

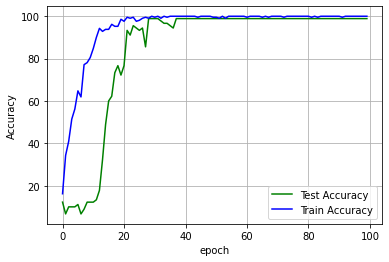
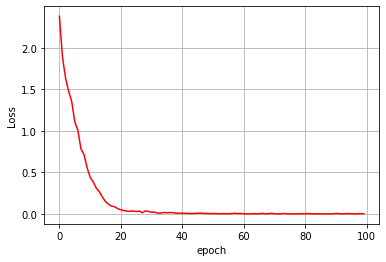
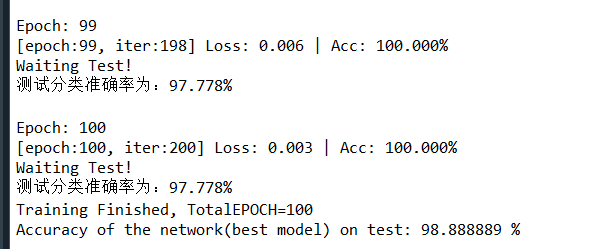
 

图 8 深度网络的设计代码



### 六、结论

本报告分析了CWRU数据集的结构和类别，在Pytorch软件框架下，开发了dataloader数据读取类。依据分类要求，设计了8层卷积分类网络，精度已到达98%。同时分析了模型训练过程中存在的训练精度和测试精度的变化规律，针对其中的不少问题，从课程中学习的理论进行了分析，提出了相应的解决方案，同时，提出的深度模型在样本充足和较少情况下均到达了较好的分类精度，表明模型具有较好的泛化性能。通过本课程的学习，收获了较多人工智能知识和Python编程技能，在后面的大创实验中，将利用本课程的知识，解决大创中的数据处理问题，实现学以致用。