《人工智能导论》

**深度学习算法LeNet的学习与复现报告**

作业类型：小组作业

指导教师：冀俊忠

小组成员：王宇飞 21071029

马雪 19071007

高立扬 21071003

李博文 21071015

郭铸仁 21071009

报告人签字：

2023年11月14日

### 一、问题描述

数字已有数千年的历史，在世界上使用很广，然而，在当今社会里，如何快速高效地将数字输入计算机，已成为影响人机接口效率的一个重要瓶颈，也关系到计算机能否真正在我国得到普及应用!在20世纪80年代末，美国的邮政局和银行希望有一种人工智能算法，能够识别各式各样的手写数字，应用到他们的业务当中。手写识别是常见的图像识别任务。计算机通过手写体图片来识别出图片中的字，与印刷字体不同的是，不同人的手写体风格迥异，大小不一， 造成了计算机对手写识别任务的一些困难。

手写数字识别是光学字符识别技术(Optical Character Recognition，简称OCR)的一个分支，它研究的对象是: 如何利用电子计算机自动辨认人手写在纸张上的阿拉伯数字数字手写体识别由于其有限的类别（0~9共10个数字）成为了相对简单的手写识别任务。DBRHD和MNIST是常用的两个数字手写识别数据集。手写数字识别作为机器视觉入门项目，无论是基于传统的OpenCV方法还是基于目前火热的深度学习、神经网络的方法都有这不错的训练效果。

其实手写数字识别相关算法的研究，早在20世纪50-60年代，也就是机器学习时期就开始了，比如用K-邻近算法、朴素贝叶斯、线性回归等算法，然而对于一张又一张的图片，这些算法的准确率一般在80-90%左右，虽然一眼看上去很高，但是一旦投入到了实际生产应用中，10%+的错误率显然很高很高。在20实际80年代末，正值深度学习相关方法陆续被提出的时期，长久以来，深度学习一直不温不火，杨立昆等人研究出了LeNet框架，通过卷积池化和全连接的深度学习网络模型，输入MNIST手写数字数据集进行训练后，实现了该领域当时的SOTA模型(State-of-the-art Model)，也就是在当时的科学技术水平和实验条件下，所能达到的顶尖模型。MNIST是一个包含数字0~9的手写体图片数据集，图片已归一化为以手写数字为中心的28\*28规格的图片。MNIST由训练集与测试集两个部分组成，各部分规模如下：训练集60,000个手写体图片及对应标签 ；测试集10,000个手写体图片及对应标签

杨立昆是深度学习“三巨头”之一，2018年图灵奖得主，是当前人工智能得以繁荣发展的重要贡献人。正是因为无数杨立昆这样的科学家，肯在科学界推崇机器学习抨击深度学习的年代，坐得住冷板凳，坚持得住真理和本心，潜心研究深度学习算法，如今深度学习才会如此生机勃勃，蒸蒸日上。而LeNet作为元老级深度学习算法，他不仅扮演着人工智能界奠基石的重要角色，更是指引万千科学家的明灯。因此，我们小组选择阅读杨立昆等人的论文"Gradient-based learning applied to document recognition"[1]，也就是上文提到的LeNet，通过阅读学习和复现，领略人工智能领域的无穷巧妙。

### 二、算法思路

#### 2.1 LeNet结构

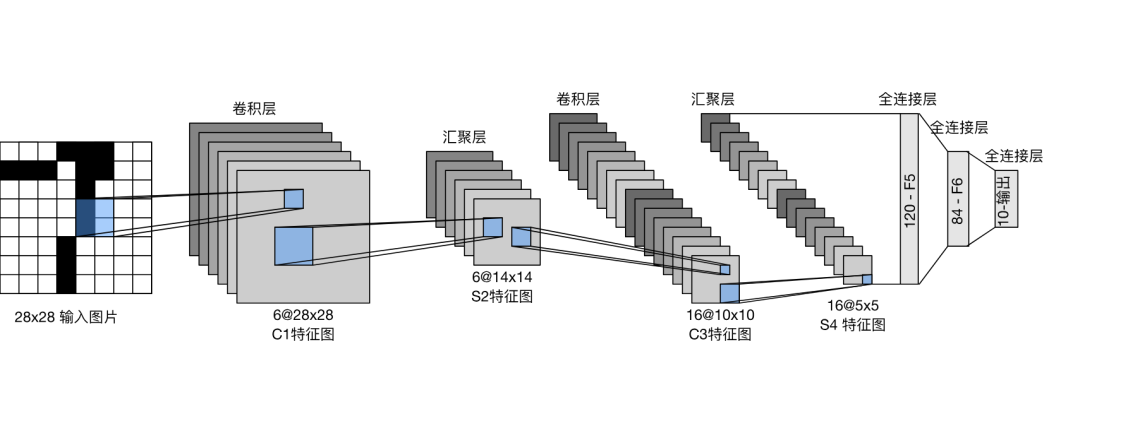


图1 LeNet架构图

如图1所示，LeNet由一系列的卷积层和池化层构成，最后连接全连接层进行分类输出预测结果。

基本结构：

1. 输入层（Input Layer）：接收输入图像数据。
2. 卷积层（Convolutional Layer）：使用卷积核对输入图像进行卷积操作，提取图像特征。同时通过激活函数引入非线性。
3. 池化层（Pooling Layer）：对卷积层的输出进行下采样，减小数据的空间维度，减少计算量，并保留重要的特征。
4. 全连接层（Fully Connected Layer）：将池化层的输出展平，并连接到一个或多个全连接层，用于图像分类。
5. 输出层（Output Layer）：进行最终的分类操作，输出预测结果。

其中INPUT即输入的图像，输入层的尺寸为32x32，当图像不是这个尺寸的时候，LeNet需要对图像进行处理，将图像的尺寸通过Transforms.Resize方式转换成32x32的尺寸大小，注意LeNet的输入应为1x32x32的灰度图像，因此该架构不适合RGB图像，即多通道图像。我们这里使用的数据集是MNIST数据集，其中图像均为灰度图像，大小均为28x28，因此需要通过Transforms.Resize方式将图像大小转换成32x32。OUTPUT输出层的10维对应得到0 ~ 9的数字识别结果。

#### 2.2 LeNet特点

如图2所示，传统的手写数字识别应用的是固定的特征提取器和可训练的分类器，神经网络之前的图像识别基本都是这样的架构，其中的特征提取器包含了很多的先验知识，因此对任务具有较强的针对性，而这也是大部分工作的重点。而分类器具有通用性和可训练性，但识别的准确性在很大程度依赖于特征的提取。因而面对数据广、易获取的场景，如手写数字，使得设计人员更偏向于依赖真实数据，而非手动提取特征来构建识别系统。

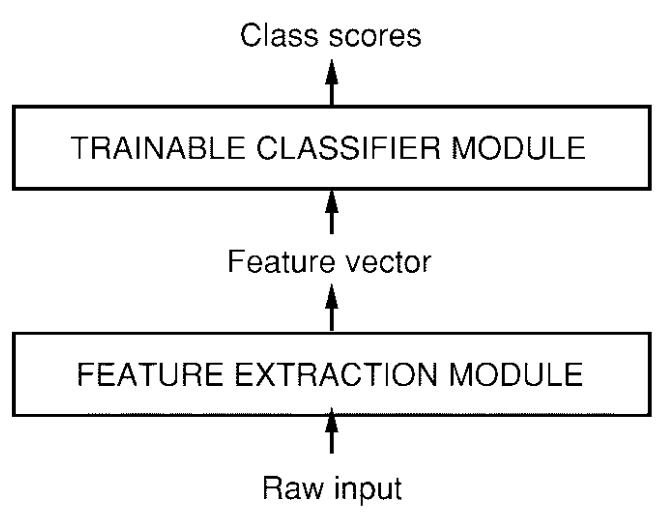


图2 传统模块

相比传统的图像识别，LeNet采用卷积进行自动提取特征，比传统的特征提取器效率更高，可解释性更强，然后采用2×2的平均池化，通过空间下采样将维数减少4倍，其中的突出贡献是使用了非线性变换，为目前的深层卷积网络奠定了基础。

这里不得不提到Hinton。早在80年代，Hinton等人便提出了反向传播(BP)算法，使得模型具备了非线性映射的能力，解决了历史遗留下来的异或问题。但当时很多神经网络的研究者都是物理学家或者是心理学家，神经网络普遍不能被工程师和计算机科学家所接受。并且，因为受限于当时的数据规模，模型训练很容易出现“过拟合”，加上机器计算能力的不足和神经网络算法的“神秘性”，那些本就寥寥无几的研究者也逐渐对它失去了兴趣，纷纷转行钻研支持向量机(SVM)的机器学习算法。虽然如此，杨立昆在前人的基础，将非线性变换应用到LeNet-5，在手写数字识别等任务上表现优异。而后，他们和Hinton一起获得了2019年图灵奖。

##### 2.3 LeNet优势

###### 2.3.1 卷积

卷积层是卷积神经网络的核心部分，它的主要功能是对输入数据进行特征提取。这一操作是通过使用多个卷积核完成的，每个卷积核由多个元素组成，每个元素都对应一个权重系数和一个偏差量。这些卷积核在工作时，会有规律地扫过输入特征，在被称为“感受野”的区域内部对输入特征做矩阵元素乘法求和并叠加偏差量。LeNet每个卷积层使用5x5的卷积核。

具体来说，卷积层的工作原理如下：首先，在某个位置覆盖滤波器；然后，将滤波器中的值与图像中的对应像素的值相乘；接着，把上述乘积加起来，得到的和就是输出图像中目标像素的值；（上述过程也称为“互相关运算”）最后，对图像的所有位置重复此操作。如下图3所示（图片来源于“花书”《动手学深度学习》）

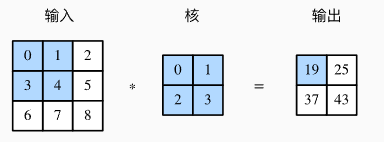


图3 卷积

我们可以通过设计不同的卷积核来达到不同目的的特征提取。比如拉普拉斯算子可以对图像进行频域层面的处理，从而实现对图像的特征区域进行提取，实现图像增强。高斯滤波核可以使得图片变模糊，过滤出更强的特征。还有很多其他功能的卷积核，比如边缘检测卷积核，颜色检测卷积核等。其设计目的都是为了提取图像不同层面的特征。通过用户输入的数据，这些卷积层可以进行学习，自训练出更适合于应用问题的参数。

同时，卷积层还可以设计stride来进行像素级别的跨越卷积，设计padding为原图进行加边，防止卷积出来的图片与原图长宽不符（如上图），设计dilation来扩大感受野，使得单次卷积能获取更多特征。通过设计输入通道和输出通道，可以控制每次卷积时的计算开销，即控制参数量的多少。

###### 2.3.2 池化

同卷积层一样，池化层每次对输入数据的一个固定形状窗口（又称池化窗口）中的元素计算输出。不同于卷积层里计算输入和核的互相关性，池化层直接计算池化窗口内元素的最大值或者平均值。该运算也分别叫做最大池化或平均池化。在二维最大池化中，池化窗口从输入数组的最左上方开始，按从左往右、从上往下的顺序，依次在输入数组上滑动。当池化窗口滑动到某一位置时，窗口中的输入子数组的最大值或平均值，即输出数组中相应位置的元素。

实施池化操作的主要目的包括：降低信息冗余，提升模型的尺度不变性、旋转不变性和平移不变性，以及防止过拟合。通过这些操作，可以更有效地表示图像的高层次特征。在LeNet中，应用到了平均池化，卷积核为2x2。

###### 2.3.3 全连接

全连接层也称为密集连接层（Dense Layer）或者全连接层（Fully Connected Layer）。全连接层可以将输入特征与每个神经元之间的连接权重进行矩阵乘法和偏置加法操作，从而得到输出结果。

在LeNet的卷积和池化层之后，特征图的维度逐渐缩小，将第二个池化层特征图展开成一维向量，这个操作也被称为拉平（Flattening）。它将保留之前提取到的所有特征作为全连接层的输入，在全连接层中，每个神经元都与上一层的所有神经元相连，每个输入特征都与每个神经元之间都存在一定的连接权重。在训练过程中，神经网络通过反向传播算法来优化每个神经元的权重和偏置，从而使得输出结果能够更好地拟合训练数据。捕捉到输入特征之间的复杂关系，从而更好的完成任务，更准确的识别手写字体。

全连接层多用于分类任务和回归任务等，在LeNet中，全连接层会将之前学习到的特征进行整合，将这些局部特征映射成全局特征，形成对于输入数据的综合理解。在第一个全连接层中，将第二个池化层的16个下采样后的特征图展开成一维向量，并传递到一个包含120个神经元的全连接层上，用于进行分类操作。这一层的输出尺寸为120；再传入到第二个包含84个神经元的全连接层，用于进一步处理第一个全连接层的输出，这一层的输出的尺寸为84；最后传入输出层，即包含10个神经元的全连接层，进行最后的处理，这是个神经元对应了手写数字识别任务中的十个数字类别

###### 2.3.4 激活函数

在全连接层的每个神经元后面通常会使用激活函数，而LeNet所使用的激活函数是Sigmoid，在当时Sigmoid是常用的激活函数之一。它将输入映射到一个范围在 0 到 1 之间的连续输出，其表达式为：

在 LeNet 中，Sigmoid 主要用于引入非线性变换，使得网络可以学习输入数据的非线性关系。激活函数的引入是为了增加神经网络的表达能力，使其能够更好地适应复杂的数据分布和任务。

后来，Sigmoid 激活函数逐渐被 Rectified Linear Unit（ReLU）激活函数取代。ReLU 函数的表达式为：

(x)=max(0,x)

这是由于ReLU相比Sigmoid计算效率更快，ReLU仅是对输入取了一个阈值，而不涉及复杂的指数运算，更容易在大规模数据上训练深度神经网络。此外还可以避免梯度消失的问题，ReLU的正区间上为1，不会像Sigmoid一样，出现当输入很大或很小的情况下，梯度会接近于零。此外，ReLU 的输出在正区间上是稀疏的（即大部分神经元输出为零），这有助于提高网络的泛化能力，减轻过拟合问题。在使用 ReLU 的情况下，神经网络通常能够更好地学习复杂的特征和模式。

### 程序框图

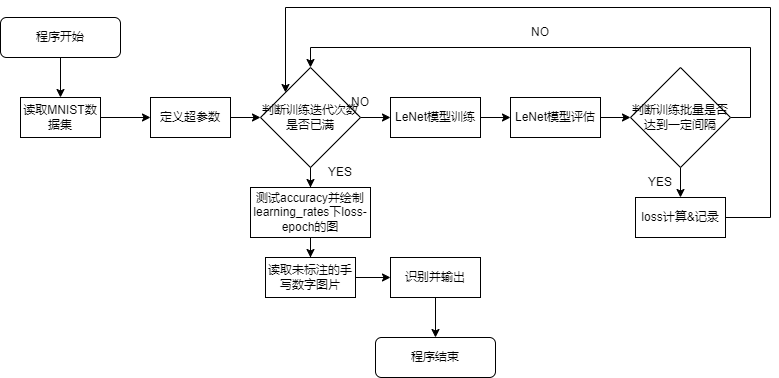


图4 程序框图

如上图4所示，该图为我们复现代码的程序框图，主要分为了读取训练集、训练模型、评估模型、模型应用四个大部分。

### 主程序伪代码及主要函数说明

#### 4.1 主程序伪代码

LeNet中用到的深度学习卷积运算，设输入图像大小为inH \* inW，卷积核c1CNum个，输出feature map大小为c1H \* c1W, 偏置个数、feature map个数与卷积核个数相等。输入图像inmap：inH \* inW，卷积核：c1conv: c1CNum \* 5 \* 5, 输出图像：c1map: c1CNum \* c1H \* c1W, 偏置c1bias：c1CNum

|  |
| --- |
| 卷积层 |
| functioin Conv:  for ith convolution kernel:  for hth row in feature map i:  for coth col in feature map i row h:  令curc1为c1map + i \* c1H \* c1W + h \* c1W + co  令指针curc1指向位置的值为0  for cr in ranges 5:  for cc in ranges 5:  令curc1指向位置的值自加 (inmap[(h+cr)\*inW + co +cc]乘以c1conv[i\*5\*5 + cr \* 5 + cc])  endfor cc  endfor cr  令curc1指向位置的值自加 (c1bias[i])  将curc1指向的值输入到激活函数，输出赋给它自己  endfor co  endfor h  endfor i  endfunction Conv |

LeNet中用到的深度学习池化运算，输入图像c1map: c1CNum \* c1H \* c1W， 输出图像：s2map: s2Num \* s2H \* s2W，其中s2Num = c1CNum，池化权值: s2pooling: s2Num，偏置：s2bias: s2num

|  |
| --- |
| 池化层 |
| function Pooling:  for ith s2 feature map:  for hth row in feature map i:  for coth col in feature map i row h:  令curs2为s2map + i \* s2H \* s2W + h \* s2W + co  令curs2指向的值 为以下四个值的均值乘以s2pooling[i] 加上s2bias[i]：  c1map[i, h \* 2 \* c1W, co \* 2], c1map[i, h \* 2 \* c1W, co \* 2+1], c1map[i, (h \* 2 +1) \* c1W, co \* 2], c1map[i, (h \* 2+1) \* c1W, co \* 2 + 1]  将curs2指向的值输入到激活函数，输出赋给它自己  endfo co  endfor h  endfor i  endfunction Pooling |

LeNet中用到的深度学习全连接运算：输入图像：c5map: c5CNum \* c5H \* c5W，输出图像：outmap: outNum, 全连接权值：outfullconn: c5CNum \* outNum，偏置: outbias: outNum:

|  |
| --- |
| 全连接层 |
| function FullConnect:  for i in ranges outNum:  令curout为outmap + i  令curout指向的值为0  for ch in ranges c5CNum:  curout指向的值自加（c5map[ch] 乘以outfullconn[ch, i]）  endfor ch  令curout指向的值自加outbias[i]然后输入到激活函数，将返回值赋值给curout指向的值  endfor i  endfunction FullConnect |

#### 4.2 主要函数说明

深度学习算法非常繁琐，书写算法层的代码不算复杂，但是藏在深度学习模型算法背后的，其实是深度学习模型数据集的读取和输入、训练和验证模式的调整、训练代码逻辑和测试代码的编写，这些代码因模型应用背景不同，难度也大不相同。深度学习模型的构建往往都是一个工程中的一小环节，通常科学家都在数据的处理和预测代码中大费心思，这些代码没有在论文中提到，是因为它们不需要创新。此外，各个环节代码的衔接更是困难的。综上，在复现时我们发现，如果没有一些封装好的工具函数，非常难完成代码的复现。

因此在进行复现时，本着知识理解和学习应用的原则，我们用到了深度学习框架PyTorch，它集成了深度学习中的非常多的算法，在构建深度学习模型时，可以直接调用框架内封装好的卷积层、池化层、全连接层这样的算法层，并且还内置了多种优化和训练加速的算法。下面是复现代码中的主要函数说明。

1. dataLoader函数（无参）

本函数作用是读取MNIST数据集。用到了深度学习框架中的torchvision.datasets.MNIST，它的作用是下载MNIST并读取，还调用了torch.utils.data.DataLoader，它使得MNIST读取时可以规定batch大小，将图像数组转换为Tensor等，十分方便高效。本函数会返回读取到的MNIST。

1. Net函数（有参）

本函数作用是搭建LeNet网络并返回网络模型，需要输入一个整数参数num\_classes（数据集有几类图片）。函数继承自torch.nn.Module，它是一个抽象深度学习模型类，一些必要的类内方法需要手动实现，这其中包括了网络的前向推理，也就是常说的网络模型。反馈网络已被torch内置的算法封装好，因此不用书写，大大简化了代码书写和衔接的复杂度。本函数会返回构建好的LeNet模型。

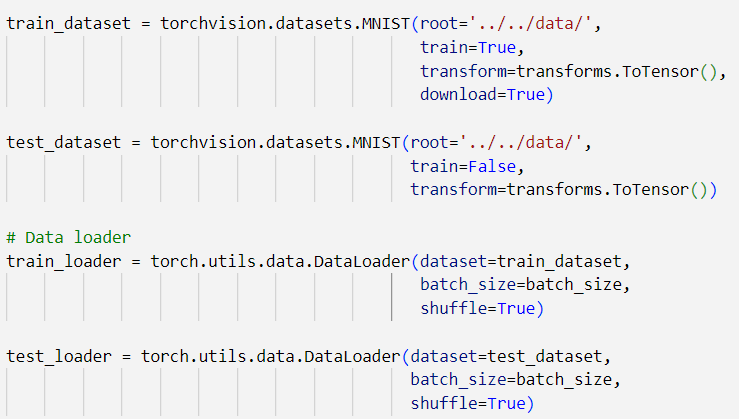
1. Train函数（有参）

本函数作用是读取MNIST并训练LeNet深度学习网络模型，需要输入若干参数：batch\_size（训练批量）、learning\_rates（学习率）、epochs（迭代次数）。函数用到了torch框架封装的CrossEntropyLoss（交叉熵损失评估函数）、Adam（优化器），还用到了matlab的插件matplotlib来对实验结果进行绘图。本函数会返回模型最终的准确率。

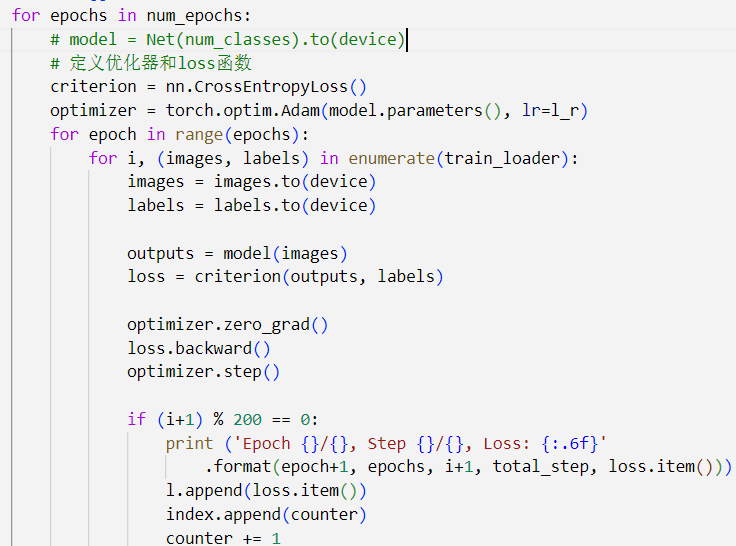
1. Predict函数（有参）

本函数的作用是对输入的手写数字图片进行识别，并输出它的预测结果，需要输入图片。

#### **4.3 重点代码节选**



代码1.数据读取 代码2.网络模型构建



代码3.模型训练 代码4.模型评估

识别代码见下文

### 实验结果

为获取最好的准确率，我们分别使用学习率分别取0.1, 0.05, 0.01, 0.005, 0.001, 0.0001, 0.00005对LeNet采用Adam优化器，以梯度下降法进行独立训练，每一个学习率的训练迭代次数取值为5, 10及15。得到预测准确率结果如表1所示：

表1.实验结果

|  |  |  |  |
| --- | --- | --- | --- |
| lr Epoch | 5 | 10 | 15 |
| 0.1 | 11.35% | 10.32% | 9.74% |
| 0.05 | 10.28% | 10.09% | 9.74% |
| 0.01 | 96.98% | 97.87% | 97.25% |
| 0.005 | 98.34% | 98.52% | 98.36% |
| 0.001 | 98.72% | 98.7% | 98.88% |
| 0.0005 | 98.56% | 98.46% | 98.89% |
| 0.00001 | 84.47% | 89.94% | 92.61% |

由表1可知，在迭代次数为15，学习率为0.0005时，LeNet预测准确率最高。下面我们对不同取值导致的loss值变化过程进行了绘图统计结果如下：

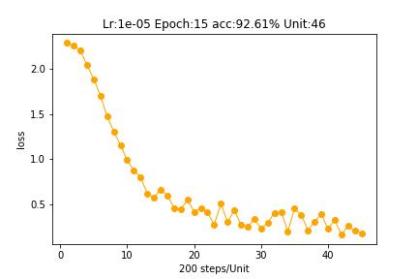
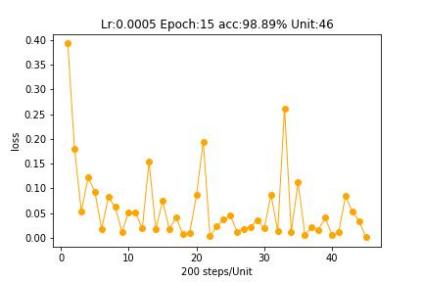
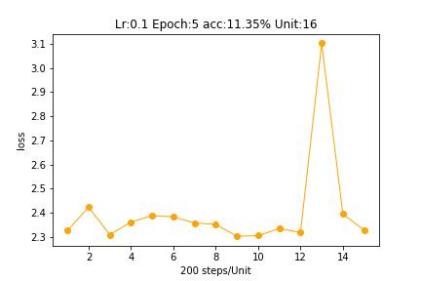


图5-7

如图5所示，过高的学习率会导致loss很难下降，而且会出现loss突变的情况，而且预测准确率很低

如图6所示，0.0005的学习率虽然可以训练出本次实验中最高的预测准确率，但是从loss图线可以看出来，loss整体呈下降趋势，但是频繁出现突变，说明训练不稳定，而由于随机数种子没有固定，说明预测准确率具有随机性，还需要继续降低学习率，获得使loss稳定下降的学习率

如图7所示，0.00001的学习率呈现的loss图线十分理想，整体呈稳定下降趋势，略有微小跳变（可以接受），而预测准确率很高。

综合loss变化图，本次任务中，我们选择下面这组参数作为最终识别参数。因随机数种子未固定，所以准确率有所区别（差了一个百分点）。

下图为设置迭代次数为15，学习率为0.0005，进行LeNet模型训练的过程，最终准确率为97.83%，训练过程如下图8

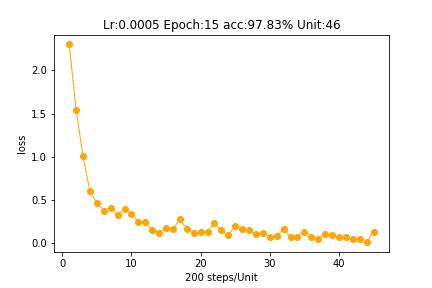
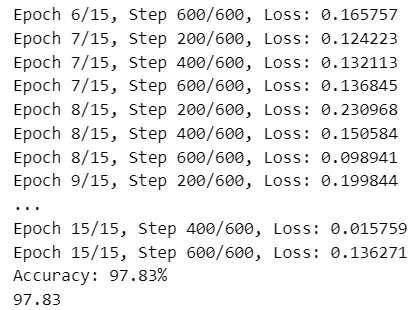
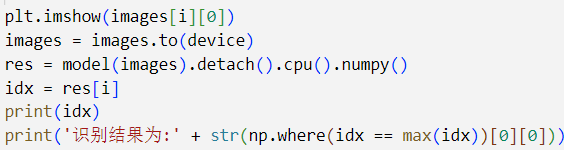


图8-9 训练过程

其中，loss值变化折线图如上图9，由图可知0.00005的学习率呈现的loss图线十分理想，整体呈稳定下降趋势，略有微小跳变（可以接受），最终预测准确率较高。



代码5.识别代码

下图为利用训练后LeNet进行实例测试的结果，数组内的值依次为LeNet对该图片中图像识别为0-10的概率，被判定为最大概率的数字即为识别结果。

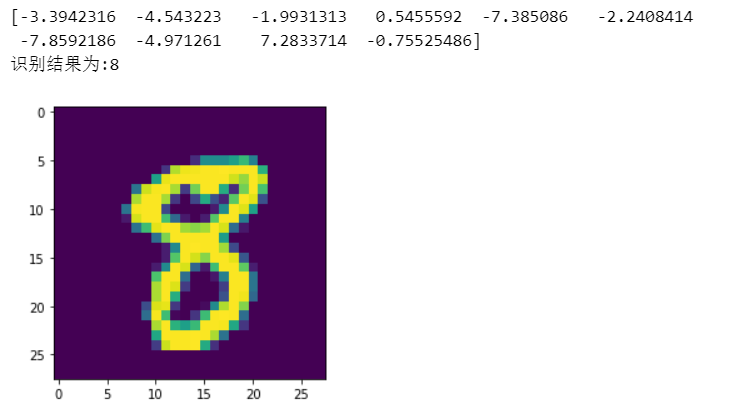
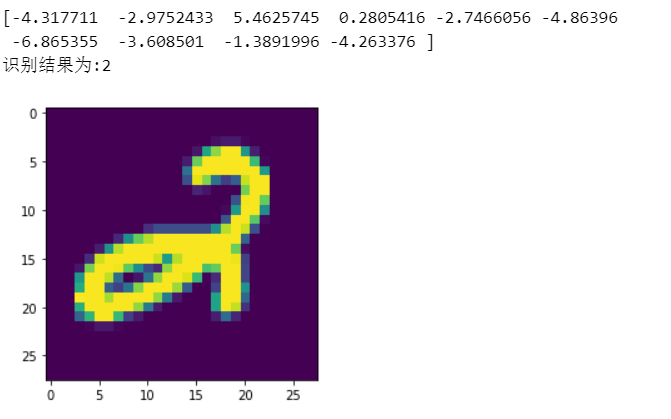
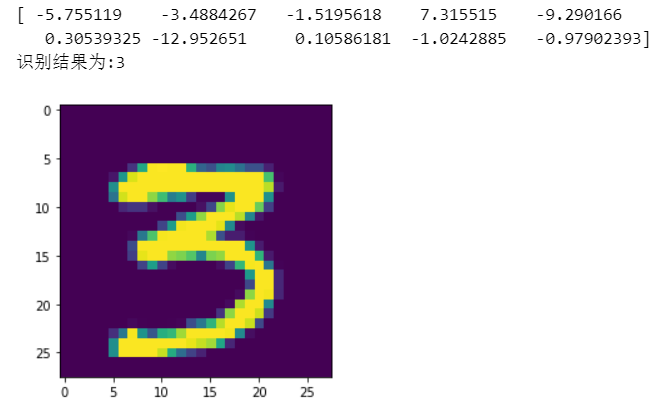
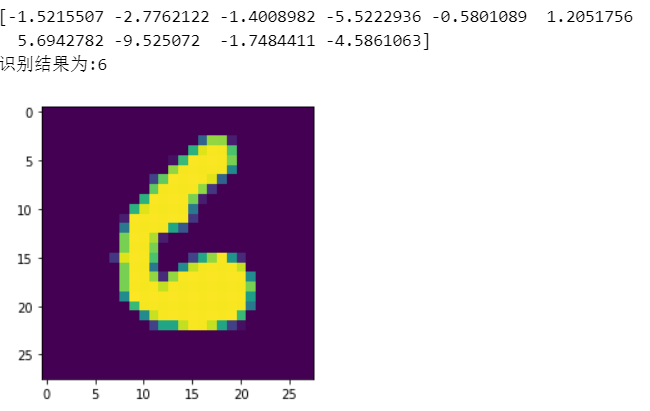
 

图10-13 预测结果

经过多次测试，发现识别结果均准确，可见模型的训练效果还是很好的。

### 收获与不足

在完成这次大作业的过程中，我们小组收获了丰富的经验和知识。通过深入阅读和学习杨立昆等人的论文"Gradient-based learning applied to document recognition"，我们对LeNet模型有了更深入的理解，领略了人工智能领域的无穷巧妙。在实践中，我们探索了LeNet模型的训练过程，并且尝试了多种调参方法，比如使用Adam优化器和不同的学习率进行模型训练。此外，我们还从中学到了如何处理手写数字识别数据集以及如何选择训练参数，比如迭代次数和学习率。通过对学习率的实验观察，我们发现过高或过低的学习率都会导致训练不稳定和性能下降的情况。这种深入探索让我对算法训练过程中参数的重要性有了更清晰的认识，也让我们意识到优化模型性能需要细致的调整和大量的实验验证。

但是我们也发现了一些不足之处。尽管loss呈稳定下降趋势，但是我们在预测准确率上并没有取得理想的结果。我们发现在训练过程中loss图线存在微小跳变，虽然可以接受，但仍希望能够进一步优化。此外，我们也发现模型在测试集上的表现可能不够理想，显示出模型的泛化能力有待提高。并且我们注意到预测准确率存在一定的随机性，这可能是由于随机数种子未固定造成的。这种情况可能会影响实验结果的可重复性和可信度，为了得到更加稳定的实验结果，我们需要进一步控制随机性。另外，由于训练时间较长，我们只能训练到50个epochs，这限制了我们对模型性能的全面评估。我们希望能够寻找更高效的训练方法，以便能够更好地探索模型的性能和参数调整。我们还在实验中意识到了超参数调整的重要性。除了学习率外，其他超参数（如正则化系数、批量大小等）的选择也会对模型性能产生重要影响。

在未来的工作中，我们将继续努力改进模型性能，可能需要深入分析模型在预测错误的样本上的表现，并尝试其他改进方法。同时，我们也将继续学习深度学习领域的最新技术和算法，以不断提升我们的能力和见识。

这次的小组作业使我们受益匪浅，我们的团队在这次合作中培养了更加严谨的学习求知的态度。通过对实验结果的详细记录和分析，我们能够更好地理解模型行为背后的原理，并且能够从实验中总结出有益的经验教训。这种严谨的学习求知的态度将成为我们未来工作中的宝贵财富，帮助我们更好地应对各种挑战并取得更大的成就。

论文文献：[1]Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.