# 计算机视觉实践报告（二）

## 1.实验目的

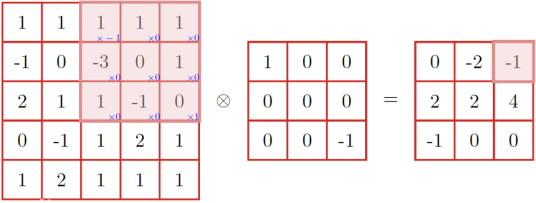
* 熟悉卷积神经网络的基本结构，包括卷积层，池化层，激活函数及最后的全连接层等。
* 学习经典手写数字识别网络LeNet-5。
* 在MNIST数据集上完成训练喝测试并进行分析

### 2.实验原理

主要包括卷积层，池化层，全连接层，激活函数，损失函数和优化器6个部分。

#### 2.1卷积层

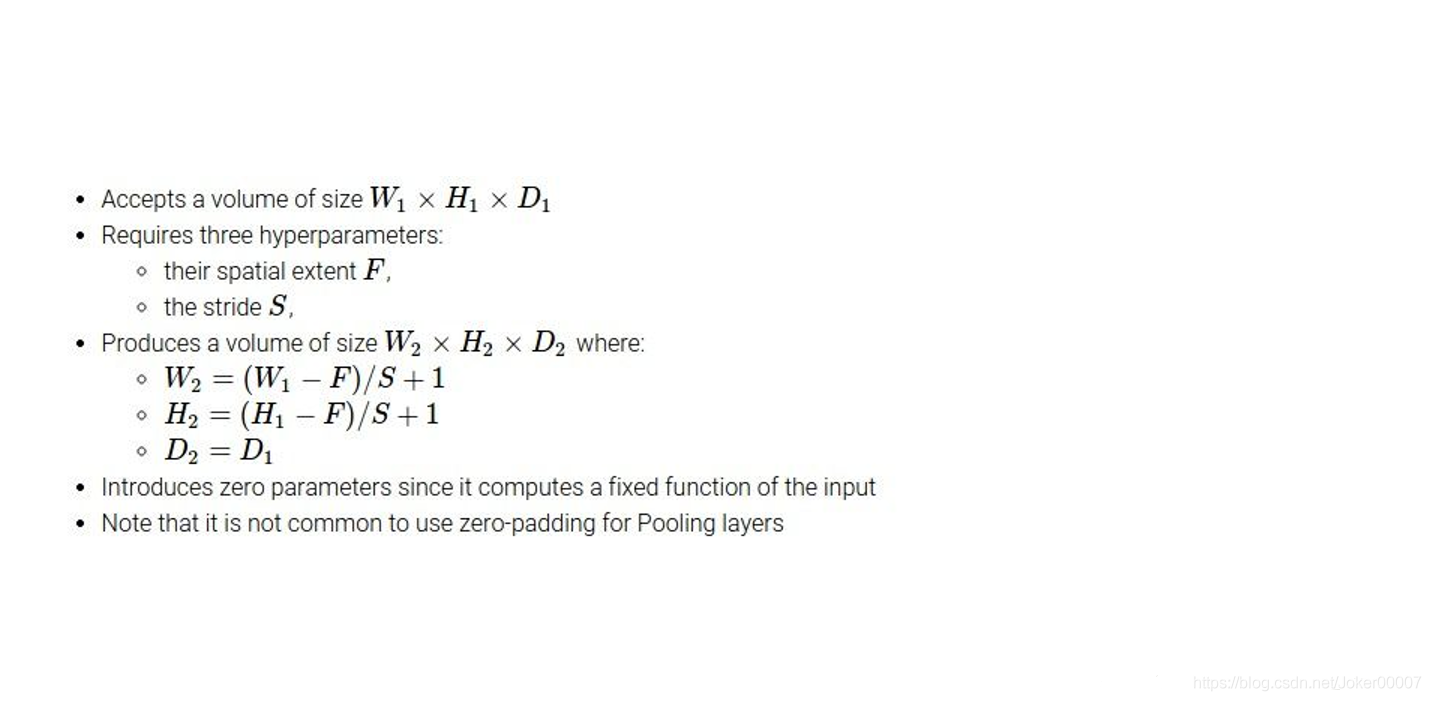
卷积核作为窗口在特征图上滑动进行卷积操作。卷积具有平移不变性，能保留输入图像的空间特点，同时相比于全连接层大大减少了参数量，将一个卷积核的参数作用与整个图像。卷积计算示意图如下：



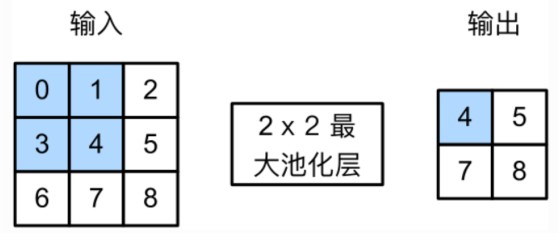
#### 2.2池化层

池化层是使用某一位置的相邻输出的总体统计特征来代替网络在该位置的输出。常见的池化方式有最大池化（MaxPooling），平均池化（AveragePooling）等。

当输入图像尺寸为𝑊1×𝐻1×𝐷1，池化核尺寸为𝐹×𝐹，池化核移动步长为S时，输出图的尺寸计算公式如下（W,H,D分别是图像宽度高度和通道数）



示意图如下：

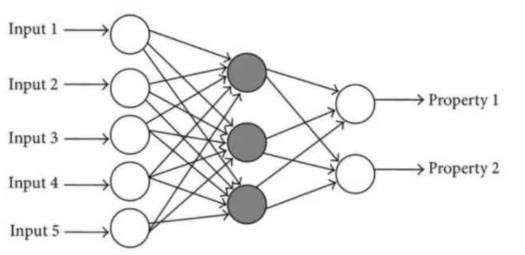


在卷积神经网络中，池化层的作用有：

1. 能够使网络自动学得不变性，包括旋转不变性和平移不变性。
2. 能够减小输入规模即参数量，提高统计效率，减少参数的存储需求。

#### 2.3全连接层

全连接层相当于矩阵乘法，对输入进行线性变换，从而把有用的信息进行整合。全连接层中间可以添加激活函数（在2.4介绍）进行非线性映射进而模拟非线性变换。示意图如下



卷积神经网络中全连接层的作用主要是作为分类器，将学到的特征映射到对应的样本类别。全连接层在实际操作过程中可以使用1×1的卷积核实现。使用nn.linear时，由于输入输出神经元之间的数量关系确定，所以输入图片必须采用指定的大小，实际应用中会有一定的限制且参数量较大。

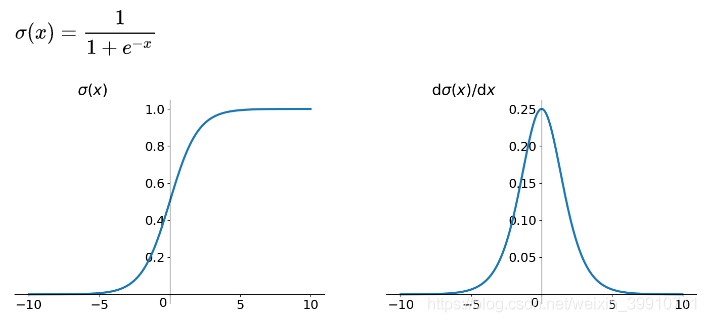
#### 2.4激活函数

激活函数（ActivationFunction）是一种添加到人工神经网络中的函数，旨在帮助网络学习数据中的复杂模式。类似于人类大脑中基于神经元的模型，激活函数最终决定了要发射给下一个神经元的内容。在人工神经网络中，一个节点的激活函数定义了该节点在给定的输入或输入集合下的输出。标准的计算机芯片电路可以看作是根据输入得到开（1）或关（0）输出的数字电路激活函数。因此，激活函数是确定神经网络输出的数学方程式。

介绍最常用的三种激活函数：Sigmoid，tanh，ReLU。

* Sigmod

Sigmoid是常用的非线性的激活函数，它的数学形式如下



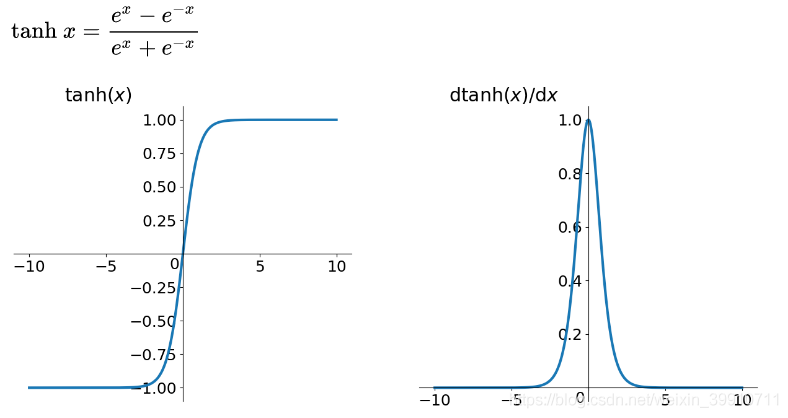
上图左边为原函数，右边为导数的图像。

特点：它能够把输入的连续实值变换为0和1之间的输出，特别的，如果是非常大的负数，那么输出就是0；如果是非常大的正数，输出就是1.

Sigmoid函数是深度学习领域开始时使用频率最高的activationfunction。它是便于求导的平滑函数。

* tanh

tanh激活函数的图像也是S形，表达式如下：

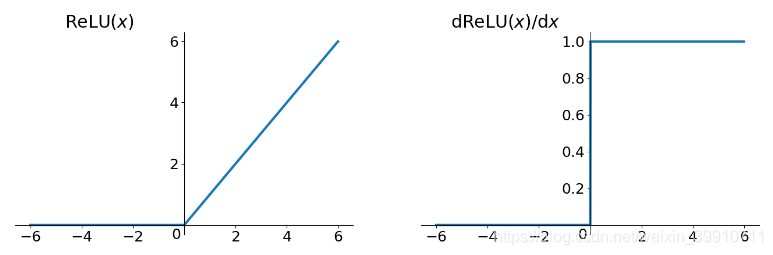


tanh读作HyperbolicTangent，如上图所示，它解决了zero-centered的输出问题，然而，gradientvanishing的问题和幂运算的问题仍然存在。

tanh是一个双曲正切函数。tanh函数和sigmoid函数的曲线相对相似。但是它比sigmoid函数更有一些优势。

* ReLU

近年来，ReLU函数变得越来越受欢迎。全称是RectifiedLinearUnit，中文名字：修正线性单元。ReLU是Krizhevsky、Hinton等人在2012年《ImageNetClassificationwithDeepConvolutionalNeuralNetworks》论文中提出的一种线性且不饱和的激活函数。它的数学表达式：



ReLU函数其实就是一个取最大值函数，注意这并不是全区间可导的，但是我们可以取sub-gradient，如上图所示。ReLU虽然简单，但却是近几年的重要成果，有以下几大优点：

* 解决了gradientvanishing问题(在正区间)
* Sigmoid和Tanh激活函数均需要计算指数，复杂度高，而ReLU只需要一个阈值即可得到激活值。ReLU函数中只存在线性关系，因此它的计算速度比sigmoid和tanh更快。计算速度非常快，只需要判断输入是否大于0。
* 收敛速度远快于sigmoid和tanh
* ReLU的非饱和性可以有效地解决梯度消失的问题，提供相对宽的激活
* 边界。
* ReLU的单侧抑制提供了网络的稀疏表达能力。

#### 2.5损失函数

简单的理解就是每一个样本经过模型后会得到一个预测值，然后得到的预测值和真实值的差值就成为损失（当然损失值越小证明模型越是成功），我们知道有许多不同种类的损失函数，这些函数本质上就是计算预测值和真实值的差距的一类型函数，然后经过库（如pytorch，tensorflow等）的封装形成了有具体名字的函数。

#### 2.6优化器

常用的优化器有SGD,BGD,Adam，Momentum，RMSprop等。在本实验中主要使用了Adam和

Momentum优化器，原理如下：

* Adam优化器

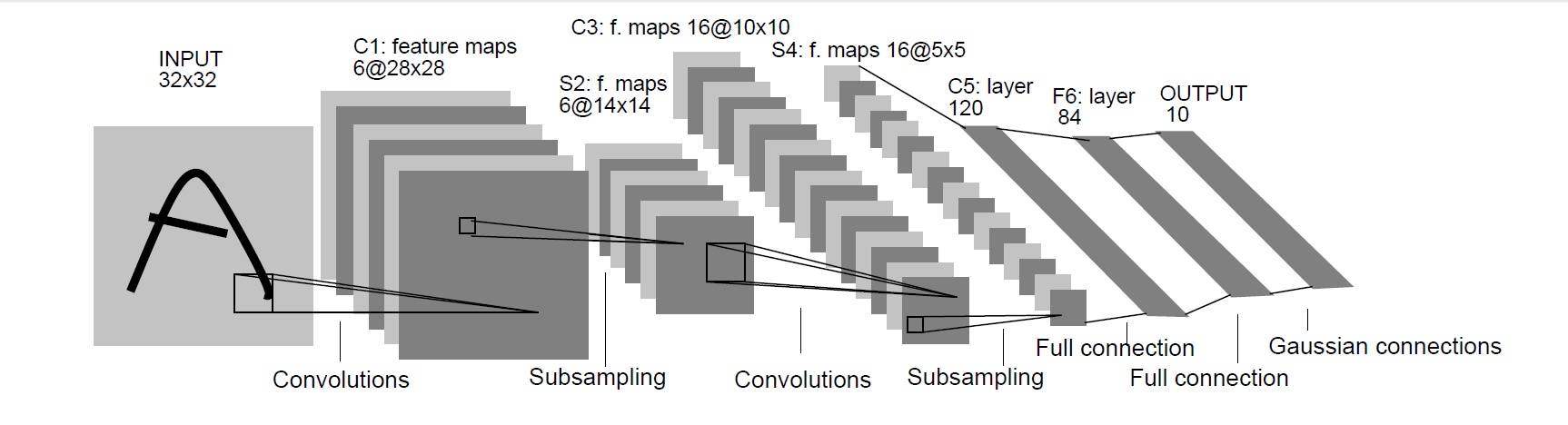
Adam吸收了Adagrad（自适应学习率的梯度下降算法）和动量梯度下降算法的优点，既能适应稀疏梯度，又能缓解梯度震荡的问题。

* Momentum优化器

Momentum梯度下降算法在与原有梯度下降算法的基础上，引入了动量的概念，使网络参数更新时的方向会受到前一次梯度方向的影响，换句话说，每次梯度更新都会带有前几次梯度方向的惯性，使梯度的变化更加平滑，这一点上类似一阶马尔科夫假设；Momentum梯度下降算法能够在一定程度上减小权重优化过程中的震荡问题。引入动量的具体方式是：通过计算梯度的指数加权平均数来积累之前的动量，进而替代真正的梯度。

## 3网络架构

LeNet-5共有7层（不包括输入层），每层包含不同数量的可训练参数和连接数，整体网络架构如下图所示：



#### 3.1C1卷积层

输入图像尺寸：1×32×32

卷积核大小：5×5

卷积核种类：6

输出图像尺寸：6×28×28

可训练参数：(5×5+1)×6=156（共6个卷积核，每个卷积核包含5×5=25个一般参数加一个可训练的偏置参数）

连接数：(5×5+1)×6×28×28=122304（6个FeatureMap分别与其对应的输入图像中的5×5像素加一个偏置之间存在连接）

说明：输入图像为单通道的灰度图，大小为32×32，经过6个卷积核卷积后生成6张FeatureMap，分别包含了经过C1层提取出来的不同特征，输出图像大小为28×28。

#### 3.2S2池化层

输入图像尺寸：6×28×28

采样核大小：2×2

采样步长：2

池化方式：最大池化

激活函数：sigmoid

输出图像尺寸：6×14×14

输出图像尺寸：28×28

连接数：(2×2+1)×14×14×6=5880（6个输出图像分别与其对应的输入图像中的2×2像素加一个偏置之间存在连接）

说明：输入图像为6张FeatureMap，大小为28×28，经过步长为2，采样核为2×2的最大池化后生成6张下采样图，输出图像大小为14×14。根据图像局部相关性的原理可知，对图像进行下采样，可以在减少数据处理量的同时保留有用信息。

#### 3.3C3卷积层

输入图像尺寸：6×14×14

卷积核大小：5×5

卷积核种类：16

输出图像尺寸：16×10×10

可训练参数：6×(3×5×5+1)+6×(4×5×5+1)+3×(4×5×5+1)+1×(6×5×5+1)=1516

连接数：10×10×1516=151600（6个FeatureMap分别与其对应的输入图像中的5×5像素加一个偏置之间存在连接）

说明：C3的前6个FeatureMap分别与S2层的第012,第123,…,第501个FeatureMap相连接，后面6个FeatureMap分别与S2层的第0123,第1234,…,第5012个FeatureMap相连接，后面3个FeatureMap分别与S2层部分不相连的4个FeatureMap相连接，最后一个FeatureMap与S2层的所有FeatureMap相连。所以共有6×(3×5×5+1)+6×(4×5×5+1)+3×(4×5×5+1)+1×(6×5×5+1)=1516个参数。而图像大小为10×10，所以共有151600个连接。

#### 3.4 S4池化层

输入图像尺寸：16×10×01

采样核大小：2×2

采样步长：2

池化方式：最大池化

激活函数：sigmoid

输出图像尺寸：16×5×5

连接数：16×(2×2+1)×5×5=2000（16个输出图像分别与其对应的输入图像中的2×2像素加一个偏置之间存在连接）

说明：S4池化层的作用方式和结果与S2层类似，不再赘述。

#### 3.5 C5卷积层

输入图像尺寸：16×5×5

卷积核大小：5×5

卷积核种类：120

输出图像尺寸：120×1×1

可训练参数：120×(16×5×5+1)=48120

连接数：120×(16×5×5+1)=48120（120个FeatureMap分别与其对应的输入图像中的5×5像素加一个偏置之间存在连接）

说明：输入图像是16张大小为5×5的图，而卷积核大小为5×5，所以卷积后形成的图的大小为1×1。得到的120个卷积结果图每个都与上一层的16个图相连，所以共有120×(16×5×5+1)=48120个参数，同样有48120个连接。

#### 3.6 FC6全连接层

输入：120×1×1

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

激活函数：tanh

输出：84×1×1

可训练参数：84×(120+1)=10164

说明：FC6层有84个节点，对应于一个7x12的比特图，-1表示白色，1表示黑色，每个符号的比特图的黑白色对应于一个编码

#### 3.7 FC7全连接层

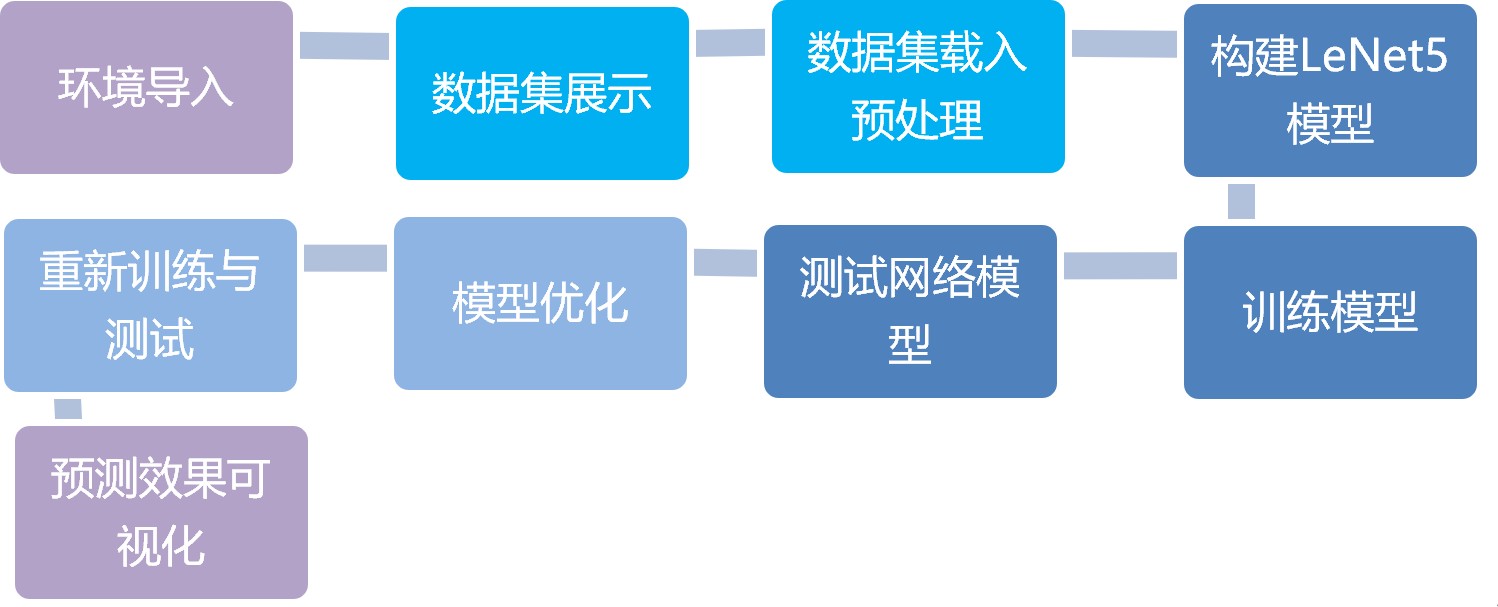
输入：84×1×1

输出：10×1

连接数：84×10=840

说明：Output层也是全连接层，共有10个节点，分别代表数字0到9，且如果节点i的值为0，则网络识别的结果是数字i。采用的是径向基函数（RBF）的网络连接方式。假设x是上一层的输入，y是RBF的输出。

## 4.实验步骤



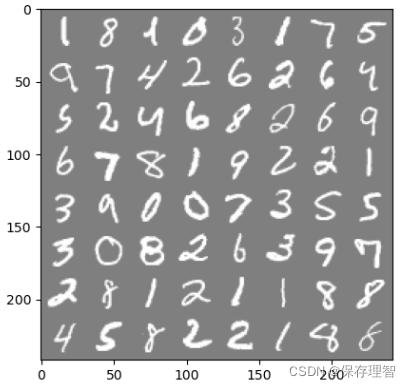
## 5.数据集

本次实验采用MNIST数据集，MNIST数据集是NIST（NationalInstituteofStandardsandTechnology，美国国家标准与技术研究所）数据集的一个子集，MNIST数据集可在http://yann.lecun.com/exdb/mnist/获取。

训练集train一共包含了 60000 张图像和标签，而测试集一共包含了 10000 张图像和标签。

idx3表示3维，ubyte表示是以字节的形式进行存储的，t10k表示10000张测试图片（test10000）。

每张图片是一个28\*28像素点的0 ~ 9的灰质手写数字图片，黑底白字，图像像素值为0 ~ 255，越大该点越白。



## 6.程序代码

#### 6.1 网络代码

class LeNet5(nn.Module):  
 def \_\_init\_\_(self, in\_dim, n\_class):  
 super(LeNet5, self).\_\_init\_\_()  
 self.conv = nn.Sequential(  
 nn.Conv2d(in\_dim, 6, 5, stride=1, padding=2),  
 nn.ReLU(True),  
 nn.MaxPool2d(2, 2),  
  
 nn.Conv2d(6, 16, 5, stride=1, padding=0),  
 nn.ReLU(True),  
 nn.MaxPool2d(2, 2))  
  
 self.fc = nn.Sequential(  
 nn.Linear(400, 120),  
 nn.Linear(120, 84),  
 nn.Linear(84, n\_class))  
  
 def forward(self, x):  
 out = self.conv(x)  
 out = out.view(out.size(0), -1)  
 out = self.fc(out)  
 return out

#### 6.2训练代码

def train(train\_loader,model,criterion,optimizer,epoch):  
 train\_loss = 0.0  
 train\_acc = 0.0  
 train\_len = 0.0  
 total = 0.0  
 model.train()  
 for i,data in enumerate(train\_loader):  
  
 img, label = data  
 img = img.to(device)  
 label = label.to(device)  
 output = model(img)  
  
 loss = criterion(output,label)  
 optimizer.zero\_grad()  
 loss.backward()  
  
 optimizer.step()  
  
 train\_loss += loss.item()  
 train\_len += output.shape[0]  
  
 #ACC  
 pred = torch.argmax(output,1)  
 num\_correct = pred.eq(label.data.view\_as(pred)).sum()  
 train\_acc += num\_correct.item()  
 total += label.size(0)

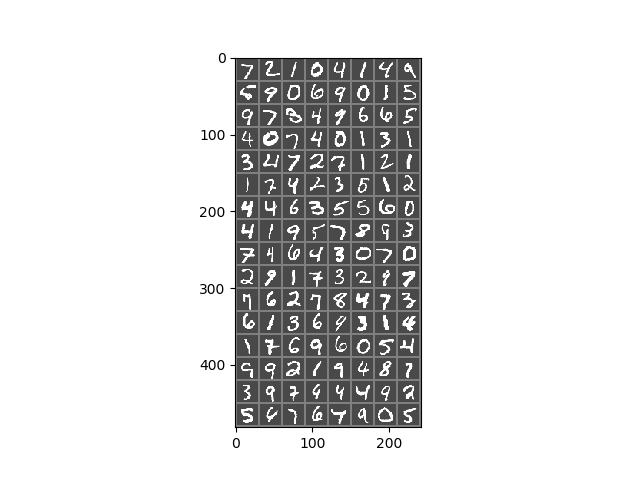
## 7.实验结果

在训练了40轮epoch后的结果：

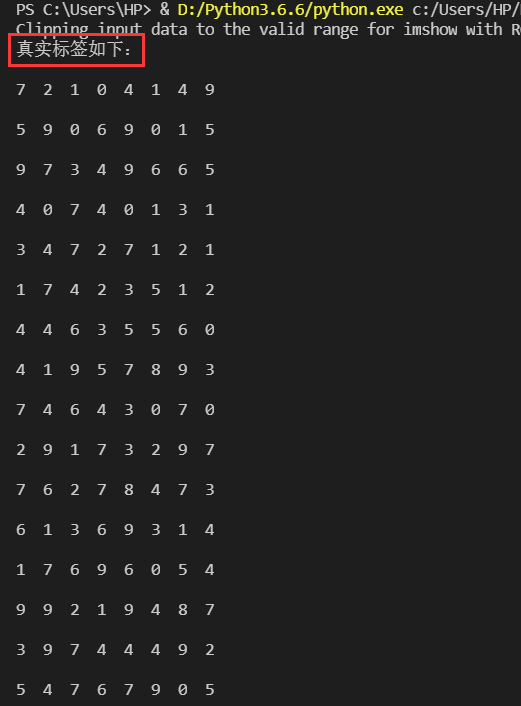


在训练完LeNet-5模型后，我保存了模型，然后利用另一个Python程序利用训练好的模型对于MNIST测试集中的前128张具体图像进行了手写数字识别的检测，实验结果如下：

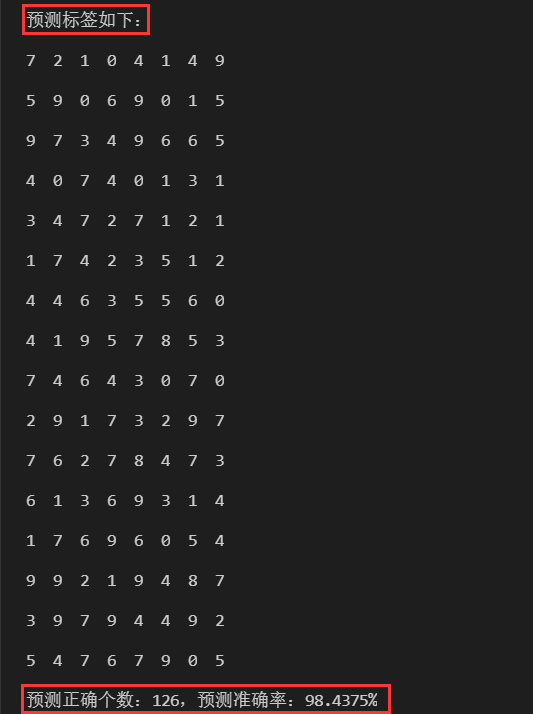
训练集中前128张图像的集合可视化图：



这批图像的真实标签：



预测之后的结果：



## 8.实验总结

#### 8.1. 学习率对分类准确率的影响

为了能够使得梯度下降法有较好的性能，我们需要把学习率的值设定在合适的范围内。学习率过小，会极大降低收敛速度，增加训练时间；学习率过大，可能导致参数在最优解两侧来回振荡，导致网络难以收敛甚至无法收敛。

分析实验过程可知，对于LeNet网络而言，若学习率设置得过小（如0.001），会导致网络的loss下降非常慢，容易陷入过拟合的局面，具体体现在第1轮epoch的测试集准确率非常低；在此基础上稍微增大学习率至0.005，就可以使准确率在一开始就获得很大的提升；设置学习率为0.01，从头至尾的准确率都已经能达到90%以上；设置学习率为0.05，可以看到准确率有了进一步提升；设置学习率为0.1，基础准确率进一步提高，但对于后面的每一个epoch而言，准确率并不会得到太大的提升了。

#### 8.2. 训练批次大小对分类准确率的影响

当batch size较小时，收敛速度似乎较快。当batch size较大时，好处是跑完一次epoch所需的训练批次数减少，并且在一定范围内batch size提高后，其确定的下降方向越准，引起训练震荡越小，但是大的batch size会导致收敛速度变慢。

显然，batch size不能盲目增大。比如，可以感受到当batch size取到256时我的电脑运算速度比取64时慢了一些。此外，可以发现在大的batch size下测试集分类准确率稍有下降。这是应该是因为epoch不够多，在同样的epoch数下参数更新变少了，因此需要更多的迭代次数才能达到和小batch一样的准确率。