计算机视觉应用与实践实验报告

目录

一、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 LeNet5 神经网络
- 自行设计程序，在 MNIST 数据集上进行训练和测试

二、实验原理

本节首先介绍卷积神经网络的基本结构，然后再详细陈述 LeNet-5 模型结构及其中的参数构成。

2.1 卷积神经网络（CNN）

主要介绍卷积神经网络基本结构，主要包括卷积层，池化层，激活函数和全连接层、损失函数、优化器等。

2.1.1 卷积层

卷积层是卷积神经网络网络的核心组成部分，卷积层得名于卷积运算，虽然卷积层中用到的是互相关运算，但是由于卷积核是可学习的，所以互相关和卷积运算在卷积核是可学习种情况下没有本质区别。卷积层是通过卷积核对原数据进行特征提取，提取更多抽象的高级的局部特征，所以经过卷积得到的图也叫特征图。其中卷积核是卷积层的特征检测器，是一个 m×n 的矩阵，通常有 1×1、3×3、5×5 等尺寸，将卷积核与图像的像素矩阵进行矩阵点积运算并滑动就可以得到一个新的矩阵，卷积核的参数不是固定的，而是需要通过神经网络反向传播进行学习的。在卷积的计算当中，假设将 5×5 的图像像素矩阵与 3×3 的卷积核进行点积，边界不补 0，步长为 1 时，卷积计算如图 2.1 所示，通过卷积计算之后会输出一个 4×4 的特征矩阵，比如输出的特征图的第一个元素 4 =1×1＋1×0＋1×1＋0×0＋1×1＋1×0＋0×1＋0×0+1×1。在实际卷积中不只有一个，而是多个卷积核，每个卷积核的参数不同，卷积核所关注的特征不同，用多个卷积核对同一区域进行加权求和可以提取不同的特征，所以卷积层卷积核的个数也决定了输出的通道数。

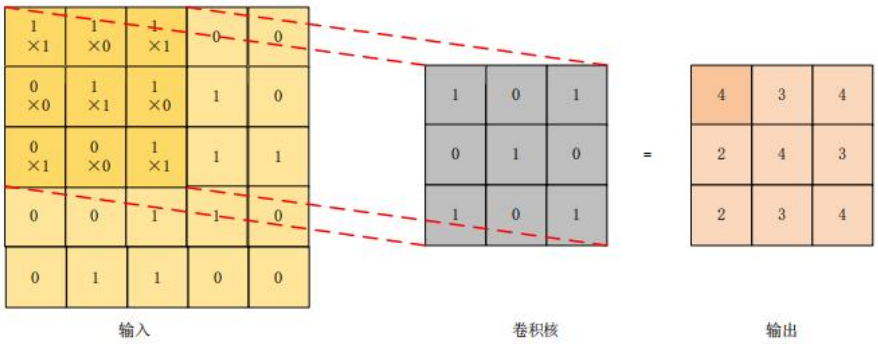


图 2.1 卷积计算

2.1.2 池化层

池化层主要是进行下采样，有特征降维，去除冗余信息，减少计算量以及防止过拟合得作用，主要有最大池化和平均池化两种方式下采样。最大池化和平均池化分别是取池化窗口的最大值和平均值。当输入图像尺寸为$W_1 \times H_1 \times D_1$，池化核尺寸为$F \times F$，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)

$$W_2 = \frac{W_1 - F}{S} + 1$$

$$H_2 = \frac{H_1 - F}{S} + 1$$

$$D_2 = D_1$$

具体的最大池化和平均池化操作如图 2.2 所示，池化窗口为均 2×2，步长均为 2，均无填充，经过池化操作后尺寸 4×4 的特征图变成 2×2 的特征图。

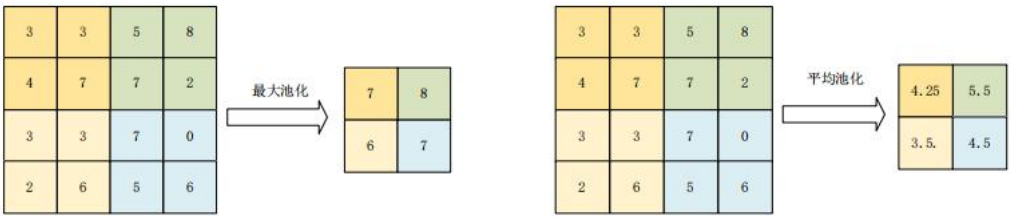


图 2.2 池化层操作示意图

2.1.3 激活函数

激活层在神经网络中具有非常重要的作用，在实际应用中，输入 x 与输出 y 的映射关系往往是非线性的，为了实现网络的非线性，增强网络的非线性能力，那就需要引入激活函数，激活函数引入的非线性因素使得神经网络可以模拟任何非线性函数，从而解决很多实际生活中的线性问题。常见的激活函数有 Sigmoid、Tanh、ReLu、Leaky ReLu 等。

（1）Sigmoid 激活函数

Sigmoid 函数的数学表达式如下：

$$f(x) = \frac{1}{1 + e^{-x}}$$

从上式和图 2.3 可以看出 Sigmoid 是一个单调的且连续的函数，可以将输入映射到（0,1）之间。它可以用来表示概率，常用在分类问题中，也可以对输入进行归一化。Sigmoid 激活函数在输入比较大或者比较小的情况下，接近饱和状态，一阶导数接近于 0，容易出现梯度消失现象，从而导致训练效率不高，模型收敛停滞不前。且不是零均值、需要指数运算，计算效率较高。

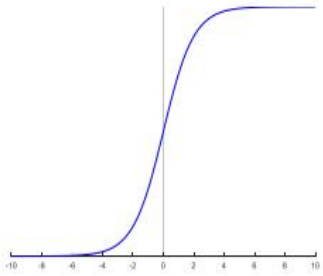


图 2.3 Sigmoid 激活函数

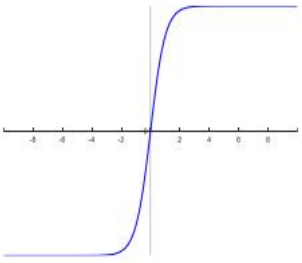


图 2.4 Tanh 激活函数

（2）Tanh 激活函数

Tanh 函数的数学表达式如下：

$$f(x) = \frac{sinhx}{coshx} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Tanh 函数相对 Sigmoid 而言，解决了非零均值的问题，从上式和图 2.4 可以看出它将输入映射到（-1,1）之间，其在一定程度上解决了输出样本分布非零中心的问题，但是它和 Sigmoid 函数一样需要进行指数运算，导致计算效率低，

且也存在着过饱和的问题，容易出现梯度消失。

（3）ReLu 激活函数

Relu 函数的数学表达式如下：

$$f(x) = \max(0, x)$$

从上式和图 2.5 可以看出，Relu 函数是一种分段的线性函数，大于零的输入则输出不变，小于等于 0 的输入则输出为 0。因为 Relu 激活函数只需要进行 max 计算，计算很简单，计算速度大大提升。而且由于 Relu 的非负区间的导数是一个常数，因此不存在像 sigmoid 和 tanh 激活函数一样的梯度消失问题，可以使得模型的收敛速度保持在一个较为稳定的状态。但是它的输出不是零均值，在训练过程中很可能出现大量神经元失效。

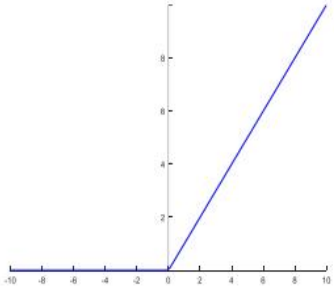


图 2.5 ReLu 激活函数

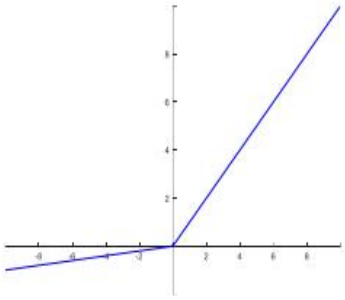


图 2.6 Leaky ReLu 激活函数

（4）Leaky ReLu 激活函数

Leaky Relu 函数的数学表达式如下：

$$f(x) = \begin{cases} x & x > 0 \\ \alpha x & x <= 0 \end{cases}$$

从上式和图 2.6 可以看出，Leaky ReLu 函数通过把一个非常小的线性分量给予负输入来调整负值得零梯度问题，通常α取 0.1。Leaky ReLu 激活函数解决了 ReLu 激活函数在输入为负时，反向传播时梯度完全为 0 的问题。

2.1.4 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进行非线性映射进而模拟非线性变换。示意图如下：

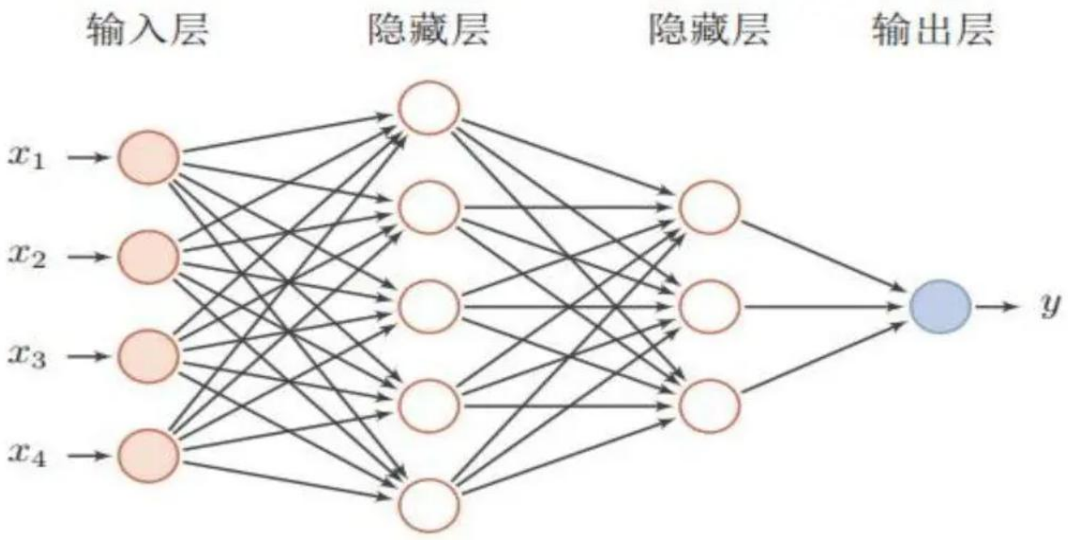


图 2.7 全连接层

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。

2.1.5 损失函数

本实验中解决的是一个多分类问题，选择交叉熵（CrossEntropy）作为损失函数，公式如下：

$$L=\frac{1}{N}\sum_{i}L_i=-\frac{1}{N}\sum_{i}\sum_{c=1}^{M}y_{ic}log(p_{ic})$$

其中 M 表示类别的数量，y_{ic}是符号函数，如果分类正确取 1，错误取 0，p_{ic}是观测样本属于对应类别的预测概率。

在网络训练过程中，对损失函数进行梯度下降进行优化，训练网络中的权值。

2.1.6 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 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。首先对梯度，梯度的平方使用指数加权平均

$$m_t=\beta_1 m_{t-1}+(1-\beta_1)g_t$$
$$v_t=\beta_2 v_{t-1}+(1-\beta_2)g_t^2$$

随后进行偏差修正

$$\widehat{m}_t=\frac{m_t}{1-\beta_2^t}$$
$$\hat{v}_t=\frac{v_t}{1-\beta_2^t}$$

权重更新公式如下

$$\theta_t=\theta_{t-1}-\frac{\eta}{\sqrt{\hat{v}_{t-1}+\epsilon}}\widehat{m}_{t-1}$$

2.2 LeNet-5 网络

LeNet-5 是一种经典的卷积神经网络模型，用于手写数字识别任务，是卷积神经网络的开创性工作之一。该模型由 Yann LeCun 等人在 1998 年提出，并在 MNIST 数据集上取得了非常好的分类效果，为深度学习领域的发展奠定了基础。

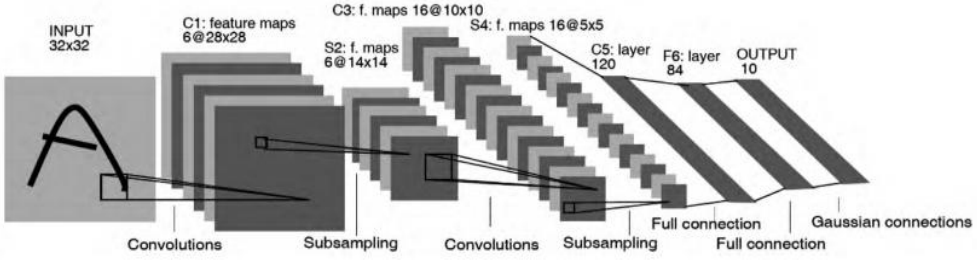


图 2.8 LeNet-5 网络模型

从图 2.8 可以看出，LeNet-5 共有 7 层，不包含输入，输入层图像的尺寸统一归一化为 32*32。每层都包含可训练参数；每个层有多个特征图，每个特征图通过一种卷积滤波器提取输入的一种特征，然后每个特征图有多个神经元。

2.2.1 C1 层-卷积层

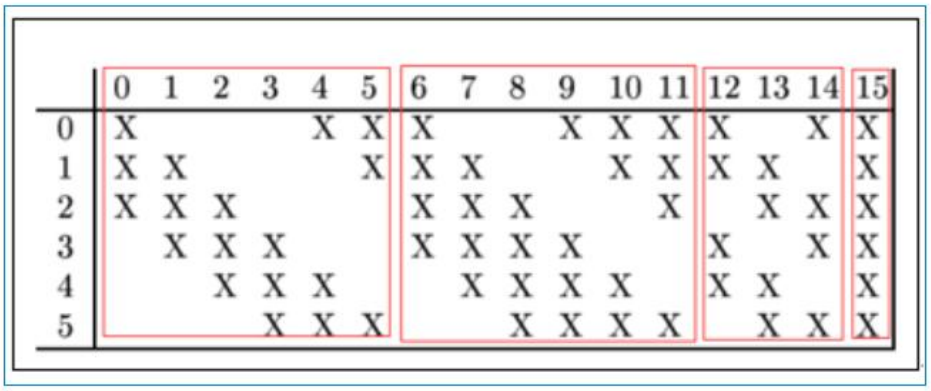
使用 6 个大小为 5*5 的卷积核对输入图像进行第一次卷积运算，得到 6 个大小为 28*28 的特征图。其中，卷积核的参数总共有 6*(5*5+1)=156 个，+1 表示每个核都有一个 bias。对于卷积层 C1，C1 内每个像素都与输入图像中的 5*5 个像素和 1 个 bias 有链接，所有一共有 156*28*28=122304 个链接。但只需要学习 156 个参数，主要是通过权值共享实习的。

2.2.2 S2 层-池化层（下采样层）

第一次卷积之后紧接着就是池化运算，使用 2*2 核进行池化，于是得到了 6 个 14*14 的特征图。S2 中 pooling 层是对 C1 中的 2*2 区域内的像素求和乘以一个权值系数再加上一个偏置，然后将这个结果再做一次映射。同时有 5*14*14*6=5880 个连接。

2.2.3 C3 层-卷积层

第一次池化之后是第二次卷积，第二次卷积的输出是 C3，16 个 10*10 的特征图，卷积核大小是 5*5。16 个特征图是通过对 S2 进行特殊组合计算得到的。具体体现为



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

图 2.9 16 个特征图的形成

C3 的前 6 个特征图（对应图 2.9 第一个红框的 6 列）与 S2 层相连的 3 个特征图相连接（图 2.9 第一个红框），后面 6 个特征图与 S2 层相连的 4 个特征图相连接（图 2.9 第二个红框），后面 3 个特征图与 S2 层部分不相连的 4 个特征图相连接，最后一个与 S2 层的所有特征图相连卷积核大小仍为 5*5，所以总共有 1516 个参数。而图像大小为 10*10，所以共有 151600 个连接。

2.2.4 S4 层-池化层（下采样层）

S4 是 pooling 层，窗口大小仍然是 2*2，共计 16 个特征图，C3 层的 16 个 10*10 的图分别进行以 2*2 为单位的池化得到 16 个 5*5 的特征图。有 5*5*5*16=2000 个连接。连接的方式与 S2 层类似。

2.2.5 C5 层-卷积层

C5 层是一个卷积层。由于 S4 层的 16 个图的大小为 5*5，与卷积核的大小相同，所以卷积后形成的图的大小为 1*1。这里形成 120 个卷积结果。每个都与上一层的 16 个图相连。所以共有(5*5*16+1)*120 = 48120 个参数，同样有 48120 个连接。

2.2.6 F6 层-全连接层

6 层是全连接层。F6 层有 84 个节点，对应于一个 7*12 的比特图，-1 表示白色，1 表示黑色，这样每个符号的比特图的黑白色就对应于一个编码。该层的训练参数和连接数是(120 + 1)*84=10164。

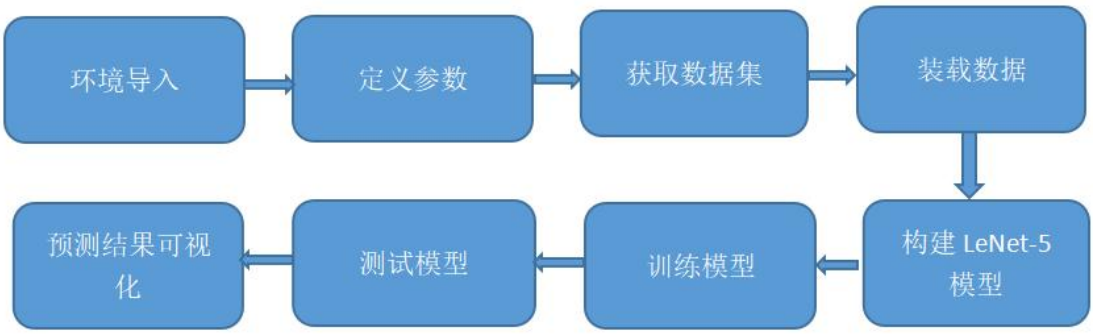
2.2.7 Output 层-全连接层

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，则 RBF 输出的计算方式是：

$$y_i = \sum_j (x_j - w_{ij})^2$$

上式w_{ij}的值由i的比特图编码确定，i从 0 到 9，j取值从 0 到 7*12-1。RBF 输出的值越接近于 0，则越接近于i，即越接近于i的 ASCII 编码图，表示当前网络输入的识别结果是字符i。该层有 84x10=840 个参数和连接。

三、实验步骤



四、数据集

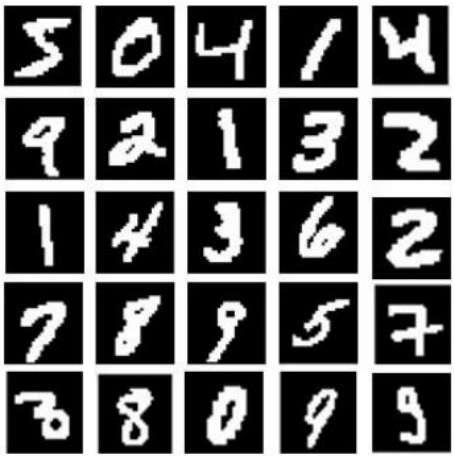


图 4.1 MNIST 数据集示例

本实验使用 MNIST 手写数据集。MNIST 数据集是深度学习里面一个非常经典的数据集，该数字集为自动识别邮编而生成，该数据集包括了 60000 个训练样本图像(共分成 10 类，每类大约有 6000 张数字图像)，也包括了 10000 张测试样本图像(共分成 10 类，每类大约有 1000 张数字图像)，如图 4.1 所示，每个样本都是 28 * 28 像素的灰度手写数字图片，即每个图像都是 28 * 28 的矩阵，矩阵中每个元素的值都在 0~255 之间。MNIST 数据集中的标签是介于 0~9 的数字，MNIST 中的标签都是用独热码(One-Hot Encoding)的形式表示这些类标，一个独热码表示的向量除了某一位数字是 1 以外，其余维度的数组都是 0。

五、程序代码

本节主要介绍 LeNet-5 在 MNIST 数据集上训练和测试的代码实现。

5.1 导入环境并定义参数

如图 5.1，导入环境后需要预先定义参数。本次实验代码中定义一个 batch_size 大小为 64，学习率为 0.001。并且用 ToTensor 实例将图像数据从 PIL 类型变换成 32 位浮点数格式，再除以 255 将像素数值做归一化处理。

```
import torch
import torchvision
import torch.nn as nn
from torch.utils import data
from torchvision import transforms
from torch.autograd import Variable
from tensorboardX import SummaryWriter

'''定义参数'''
batch_size = 64
lr = 0.001
num_classes = 10
writer = SummaryWriter(log_dir='scalar')
# 通过ToTensor实例将图像数据从PIL类型变换成32位浮点数格式
# 并除以255使得所有像素的数值均在0到1之间
# 图片转换为tensor
trans = transforms.ToTensor()
```

图 5.1 导入环境并定义参数

5.2 获取并装载数据

如图 5.2，获取训练集和测试集，下载到新建的 data 文件夹并将数据转换为 tensor。接着，用 dataloader 加载数据集。其中，训练集打乱而测试集不用打乱。

```
'''获取数据集'''
# 训练集
mnist_train = torchvision.datasets.MNIST(
    root="./data", train=True, transform=trans, download=True)
# 测试集
mnist_test = torchvision.datasets.MNIST(
    root="./data", train=False, transform=trans, download=True)

'''装载数据'''
train_loader = data.DataLoader(mnist_train, batch_size, shuffle=True)
test_loader = data.DataLoader(mnist_test, batch_size, shuffle=False)
```

图 5.2 获取并装载数据

5.3 构建 LeNet-5 模型

如下图 5.3 所示，按照实验原理 LeNet-5 原理构建模型。

```
class LeNet(nn.Module):
    def __init__(self, num_class=10):
        super().__init__()# 继承nn.model类
        '''第一层卷积，卷积核大小为5*5，步距为1，输入通道为1，
        输出通道为6(feature map),padding = 2（32*32->28*28）'''
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=2)
        '''第一层池化层，卷积核为2*2，步距为2'''
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        '''第二层卷积，卷积核大小为5*5，步距为1，输入通道为6，输出通道为16'''
        self.conv2=nn.Conv2d(6, 16, kernel_size=5, stride=1)
        '''第二层池化层，卷积核为2*2，步距为2'''
        self.pool2=nn.MaxPool2d(kernel_size=2, stride=2)
        '''第一层全连接层，维度由16*5*5=>120'''
        self.linear1 = nn.Linear(16 * 5 * 5, 120)
        '''第二层全连接层，维度由120=>84'''
        self.linear2 = nn.Linear(120, 84)
        '''第三层全连接层，维度由84=>10'''
        self.linear3 = nn.Linear(84, num_classes)
    def forward(self,x):
        out = torch.tanh(self.conv1(x))
        out = self.pool1(out)
        out = torch.tanh(self.conv2(out))
        out = self.pool2(out)
        out = out.reshape(-1, 16*5*5)
        out = torch.sigmoid(self.linear1(out))
        out = torch.sigmoid(self.linear2(out))
        out = self.linear3(out)
        return out
```

图 5.3 构建 LeNet-5 网络模型

5.4 训练模型

如下图 5.4 所示训练模型，损失函数用交叉熵损失函数，优化器用 Adam，epoch 设为 20。每隔 100 组数据输出一下 loss，

```python
def do_train():
    model = LeNet(num_classes)
    model = model.cuda()
    '''设置损失函数'''
    criterion = nn.CrossEntropyLoss()
    '''设置优化器'''
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    '''开始训练'''
    total_step = len(train_loader)
    total_loss = 0
    for epoch in range(20):
        for i, (images, labels) in enumerate(train_loader):
            images = images.cuda()
            labels = labels.cuda()
            images = Variable(images)
            labels = Variable(labels)
            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)
            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_loss += loss
            if (i + 1) % 100 == 0:
                print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                      .format(epoch + 1, 20, i + 1, total_step, loss.item()))
        writer.add_scalar("Train_Loss: ", total_loss/len(train_loader), epoch)
        writer.flush()
    # torch.save(model, './model/lenet5.pth')
```

图 5.4 训练模型

5.5 测试模型

如下图 5.5 测试模型效果，同样的，损失函数用交叉熵损失函数，优化器用 Adam。得到测试结果，计算准确率。

```python
def do_test():
    model = torch.load('./model/lenet5.pth')
    model.eval()
    model = model.cuda()

    '''设置损失函数'''
    criterion = nn.CrossEntropyLoss()
    '''设置优化器'''
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    correct = 0
    with torch.no_grad():
        for i, (images, labels) in enumerate(test_loader):
            images = images.cuda()
            labels = labels.cuda()
            images = Variable(images)
            labels = Variable(labels)
            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)
            # 获取预测结果的下标（即预测的数字）
            _, preds = torch.max(outputs, dim=1)
            # 累计预测正确个数
            correct += torch.sum(preds == labels)

    print('accuracy rate: ', correct/len(test_loader.dataset))
```

图 5.5 测试模型

六、实验结果

如下图 6.1 所示，在训练了 20 个 epoch 后，模型的损失几乎达到了 100%。

```
Epoch [19/20], Step [300/938], Loss: 0.0002
Epoch [19/20], Step [600/938], Loss: 0.0041
Epoch [19/20], Step [900/938], Loss: 0.0015
Epoch [20/20], Step [300/938], Loss: 0.0000
Epoch [20/20], Step [600/938], Loss: 0.0010
Epoch [20/20], Step [900/938], Loss: 0.0006
```

图 6.1 训练结果

如下图 6.2 所示，LeNet5 网络在测试集上的数据达到了 98.86%，设备用的是 GPU 进行测试。

```
accuracy rate:  tensor(0.9886, device='cuda:0')
```

图 6.2 测试结果

七、实验分析与总结

通过这次实验，我了解了卷积神经网络的基本概念和原理，熟悉了 LeNet-5 模型的结构和参数设置。实验结果表明，LeNet5 是一种非常有效的手写数字识别模型，可以在 MNIST 数据集上取得非常好的分类效果。通过对模型进行训练和测试，可以得到准确率高达 98%以上，这说明 LeNet5 在手写数字识别领域的性能非常优秀，可以用于实际应用。

数据预处理对于 LeNet5 在 MNIST 数据集上的表现具有重要影响。对于输入图像进行归一化和中心化等数据预处理，可以提高模型的鲁棒性和准确率。

此外，LeNet5 采用的卷积神经网络结构和训练方法为后来的深度学习模型提供了很好的启示和基础。LeNet5 的卷积神经网络结构被广泛应用于各种视觉任务中，如图像分类、目标检测等。同时，LeNet5 的训练方法也为后来的深度学习模型提供了很好的基础。

LeNet5 的超参数调整也对模型的性能有很大影响。可以通过调整学习率、批大小、卷积核大小等超参数来优化模型的性能。不同的超参数设置可能会对模型的性能产生不同的影响，因此需要进行一定的实验来寻找最优超参数设置。

虽然 LeNet5 在手写数字识别领域表现优秀，但在实际应用中也存在一些缺陷和局限性。例如，对于复杂的图像场景可能无法进行有效的识别和分类。因此，在实际应用中需要根据具体任务的需求选择合适的模型。同时，LeNet5 的卷积神经网络结构和训练方法为后来的深度学习模型提供了很好的启示和基础，有助于我们更好地理解和应用深度学习。