计算机视觉实践实验报告

目录

一. 实验目的

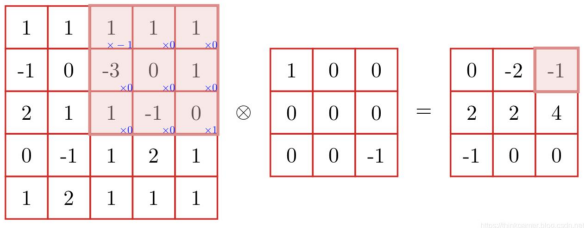
- 学习 LeNet-5 ，熟悉其网络框架。
- 在 MNIST 数据集上完成数据集的训练和测试。

二. 实验原理

LeNet-5的基本结构包括7层网络结构（不含输入层），其中包括2个卷积层、2个降采样层（池化层）、2个全连接层和输出层。以下将从卷积层、池化层、全连接层、输出层、损失函数以及优化器6个部分进行介绍。

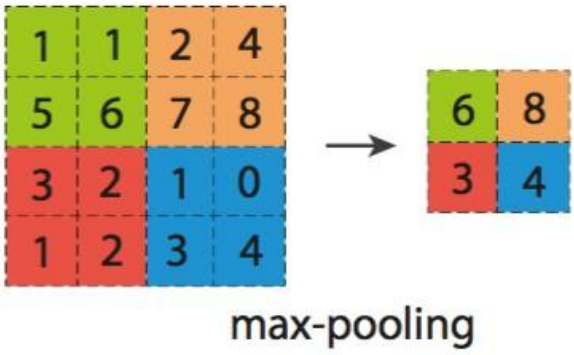
2.1 卷积层

卷积操作的实现可以使用滑动窗口的方式，即在输入图像上滑动一个卷积核，将卷积核和输入图像对应位置的像素值相乘并求和，得到输出图像中对应位置的像素值。对于多通道卷积，卷积核的通道数需要与输入数据的通道数相同，每个卷积核的每个通道都会与输入数据的相应通道进行卷积操作。，以下是卷积计算示意图：



2.2 池化层

池化层用于降低特征图的空间分辨率，并增强模型对输入图像的平移不变性和鲁棒性。常用的池化方式包括最大池化和平均池化。最大池化的操作是在一个滑动窗口中取最大值作为输出，平均池化的操作是在一个滑动窗口中取平均值作为输出。以最大池化为例，示意图如下：



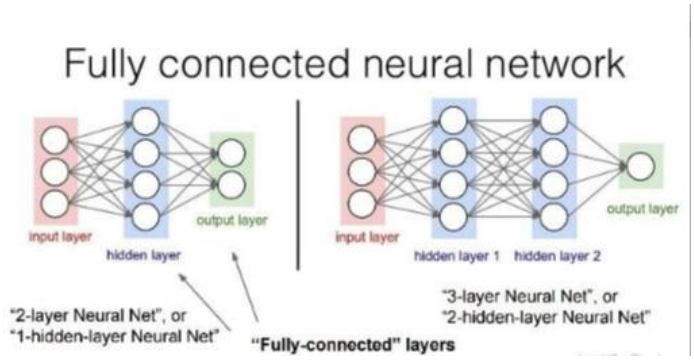
在卷积神经网络中， 池化层的作用有：

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，全连接层通常用于将卷积层和池化层提取的特征进行分类或回归。它的输入是一维向量，其输出的维度与任务的分类数或回归值的维度相同。示意图如下



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 输出层

输出层由10个神经元组成，每个神经元对应0-9中的一个数字，并输出最终的分类结果。在训练过程中，使用交叉熵损失函数计算输出层的误差，并通过反向传播算法更新卷积核和全连接层的权重参数。

然而，在实际应用中，通常会对LeNet-5进行一些改进，例如增加网络深度、增加卷积核数量、添加正则化等方法，以进一步提高模型的准确性和泛化能力。

2.5 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：

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

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。

2.6 优化器

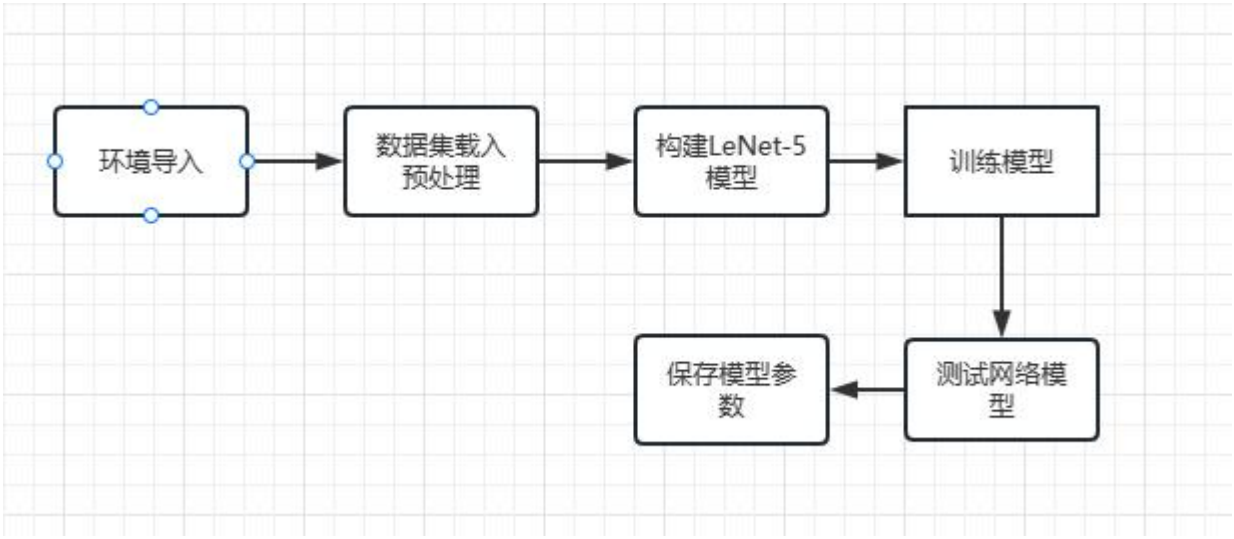
常用的优化器有SGD,BGD,Adam，Momentum，RMSprop等。在本实验中主要使用了Momentum优化器，原理如下：

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。公式如下：

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_{t-1})$$

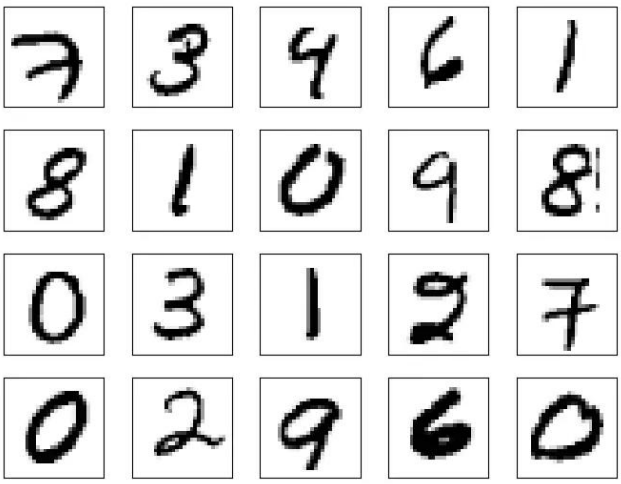
$$\theta_t = \theta_{t-1} - v_t$$

三. 实验步骤



四. 数据集

本实验用 MNIST 数据集，共包含 10 类、共 70000 张图片。其中，60000 张是训练集，10000 张是测试集。每张图片大小为 28*28。部分展示如下：



五. 程序代码

- 初始化各参数，学习率、动量、打印周期、训练轮数、训练批量大小、测试批量大小。

```
lr = 0.001  # 学习率
momentum = 0.5
log_interval = 10  # 跑多少次batch进行一次日志记录
epochs = 20
batch_size = 64
test_batch_size = 1000
```

- 数据集载入。加载训练数据集以及测试数据集，对数据依据数据集官方给出的均值和标准差系数进行归一化处理，加快模型的收敛。

```
train_loader = torch.utils.data.DataLoader(  # 加载训练数据
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))  # 数据集给出的均值和标准差系数
                   ])),
    batch_size=batch_size, shuffle=True)

test_loader = torch.utils.data.DataLoader(  # 加载训练数据
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))  # 数据集给出的均值和标准差系数
    ])),
    batch_size=test_batch_size, shuffle=True)
```

- 构建Lenet-5网络模型，包括7层网络结构（不含输入层），其中包括2个卷积层、2个降采样层（池化层）、2个全连接层和输出层。

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Sequential(  # input_size=(1*28*28)
            nn.Conv2d(1, 6, 5, 1, 2),  # padding=2保证输入输出尺寸相同
            nn.ReLU(),  # input_size=(6*28*28)
            nn.MaxPool2d(kernel_size=2, stride=2),  # output_size=(6*14*14)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(6, 16, 5),
            nn.ReLU(),  # input_size=(16*10*10)
            nn.MaxPool2d(2, 2)  # output_size=(16*5*5)
        )
        self.fc1 = nn.Sequential(
            nn.Linear(16 * 5 * 5, 120),
            nn.ReLU()
        )
        self.fc2 = nn.Sequential(
            nn.Linear(120, 84),
            nn.ReLU()
        )
        self.fc3 = nn.Linear(84, 10)

    # 定义前向传播过程，输入为x
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # nn.Linear()的输入输出都是维度为一的值，所以要把多维度的tensor展平成一维
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return x  # F.softmax(x, dim=1)
```

- 训练模型，打印相关信息。

```
def train(epoch):  # 定义每个epoch的训练细节
    model.train()  # 设置为trainning模式
    for batch_idx, (data, target) in enumerate(train_loader):
        data = data.to(device)
        target = target.to(device)
        data, target = Variable(data), Variable(target)  # 把数据转换成Variable
        optimizer.zero_grad()  # 优化器梯度初始化为零
        output = model(data)  # 把数据输入网络并得到输出，即进行前向传播
        loss = F.cross_entropy(output, target)  # 交叉熵损失函数
        loss.backward()  # 反向传播梯度
        optimizer.step()  # 结束一次前传+反传之后，更新参数
        if batch_idx % log_interval == 0:  # 准备打印相关信息
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                       100. * batch_idx / len(train_loader), loss.item()))
```

- 测试模型，打印相关信息并保存测试的准确率以及损失方便后续的可视化以及结果分析。

```
def test():
    model.eval()  # 设置为test模式
    test_loss = 0  # 初始化测试损失值为0
    correct = 0  # 初始化预测正确的数据个数为0
    for data, target in test_loader:
        data = data.to(device)
        target = target.to(device)
        data, target = Variable(data), Variable(target)  # 计算前要把变量变成Variable形式，因为这样子才有梯度

        output = model(data)
        test_loss += F.cross_entropy(output, target, size_average=False).item()  # sum up batch loss 把所有loss值进行累加
        pred = output.data.max(1, keepdim=True)[1]  # get the index of the max log-probability
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()  # 对预测正确的数据个数进行累加

    test_loss /= len(test_loader.dataset)  # 因为把所有loss值进行过累加，所以最后要除以总得数据长度才得平均loss
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
    Loss_list.append(test_loss)
    Accuracy_list.append(100. * correct / len(test_loader.dataset))
```

- 保存模型，将模型测试的结果可视化。

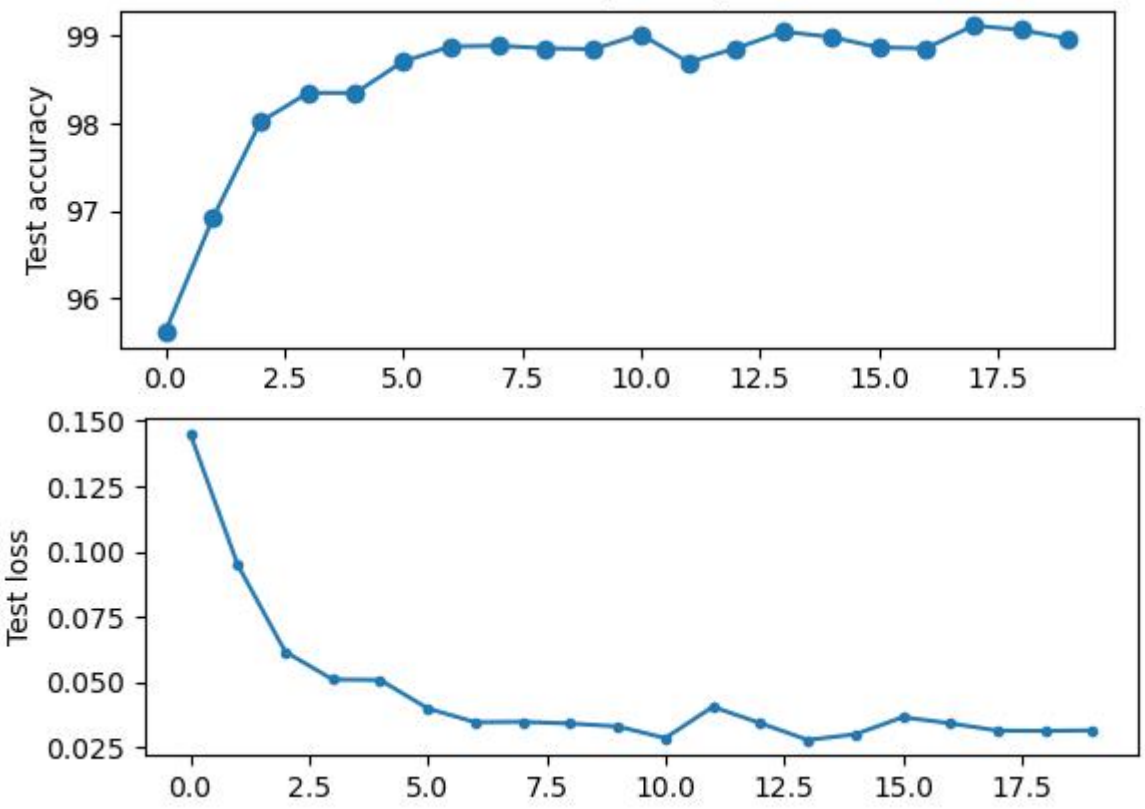
```
torch.save(model, 'model.pth')  # 保存模型
# 测试损失和准确率可视化
x1 = range(0, epochs)
x2 = range(0, epochs)
y1 = Accuracy_list
y2 = Loss_list
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'o-')
plt.title('Test accuracy vs. epoches')
plt.ylabel('Test accuracy')
plt.subplot(2, 1, 2)
plt.plot(x2, y2, '.-')
plt.xlabel('Test loss vs. epoches')
plt.ylabel('Test loss')
plt.show()
```

六. 实验结果

- 在Lenet-5网络上训练MNIST数据集，训练20个epoch，同时进行测试集的测试（学习率使用0.01）。

```
Train Epoch: 18 [0/60000 (0%)]  Loss: 0.018739
Train Epoch: 18 [32000/60000 (53%)] Loss: 0.006851

Test set: Average loss: 0.0332, Accuracy: 9890/10000 (99%)

Train Epoch: 19 [0/60000 (0%)]  Loss: 0.011292
Train Epoch: 19 [32000/60000 (53%)] Loss: 0.006610

Test set: Average loss: 0.0427, Accuracy: 9869/10000 (99%)

Train Epoch: 20 [0/60000 (0%)]  Loss: 0.052704
Train Epoch: 20 [32000/60000 (53%)] Loss: 0.015091

Test set: Average loss: 0.0386, Accuracy: 9873/10000 (99%)


Process finished with exit code 0
```

- 20个epoch内达到的最好的测试集准确率为99.12%。以下是损失以及准确率的变化情况。



七. 实验分析与总结

- 由实验结果可见，Lenet-5在手写数字识别应用上有着很好的应用，在MNIST数据集上达到很好的实验结果。
- 模型训练时，在到达一个临界点之后，训练集的误差下降，测试集的误差上升了，这个时候就进入了过拟合区域（模型复杂度高于实际问题，模型在训练集上表现很好，但在测试集上却表现很差），此时需要采取方法防止过拟合。可以采用正则化和Dropout来防止过拟合。
- 实验采用0.01的初始学习率，实验过程中也尝试了0.001的初始学习率，20个epoch的测试集准确率在98%左右，模型收敛相比0.01的初始学习率来说更慢。