南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

“计算机视觉与应用实践”

课程作业

姓 名：余 涛 学 号：122106222833

专 业：计算机科学与工程

项目名称：实现 LeNet-5 在 MNIST 数据集上的训练和测试

时间：2023 年 4 月 14 日

1、实验目标

（1）、实现 LeNet-5 在 MNIST 数据集上的训练和测试

（2）、对 LeNet-5 模型进行分析。

（3）、在训练和测试的基础上完成实验报告。

（4）、提交 LeNet-5 代码。

2、具体要求

（1）、实现 LeNet-5 在 MNIST 数据集上的训练和测试；

（2）、理解 LeNet-5 的每个卷积层和池化层，搞清楚每层的输入与输出尺度等；

3、总体思路

（1）、LeNet-5 是一种用于手写体字符识别的非常高效的卷积神经网络；

（2）、LeNet5 这个网络虽然很小，但是它包含了深度学习的基本模块：卷积层，池化层，全链接层。是其他深度学习模型的基础；

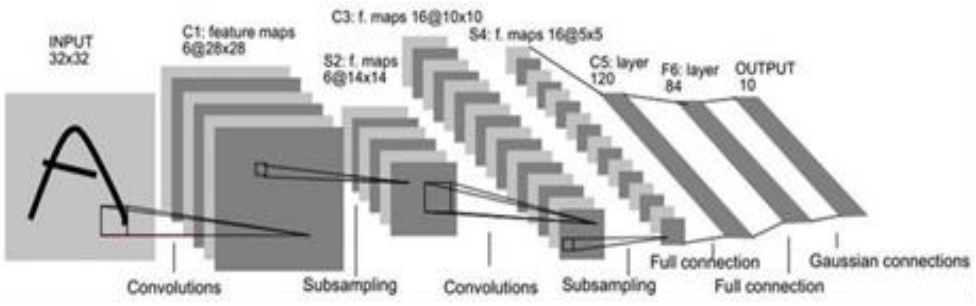
（3）、LeNet-5 共有 7 层，不包含输入，每层都包含可训练参数；每个层有多个 Feature Map，每个 FeatureMap 通过一种卷积滤波器提取输入的一种特征，然后每个 FeatureMap 有多个神经元.

4、项目实现

一、原理与实现

1、Lenet-5

LeNet5 这个网络虽然很小，但是它包含了深度学习的基本模块：卷积层，池化层，全链接层。是其他深度学习模型的基础。LeNet-5 共有 7 层，不包含输入，每层都包含可训练参数；每个层有多个 Feature Map，每个 FeatureMap 通过一种卷积滤波器提取输入的一种特征，然后每个 FeatureMap 有多个神经元。



2、各层参数详解

（1）、INPUT 层-输入层:

首先是数据 INPUT 层，输入图像的尺寸统一归一化为 32*32。

注意：本层不算 LeNet-5 的网络结构，传统上，不将输入层视为网络层次结构之一。

（2）、C1 层-卷积层:

输入图片：32*32；

卷积核大小：5*5；

卷积核种类：6；

输出 featuremap 大小：28*28 （32-5+1）=28；

神经元数量：28*28*6；

可训练参数：（5*5+1）* 6（每个滤波器 5*5=25 个 unit 参数和一个 bias 参数，一共 6 个滤波器）；

连接数：（5*5+1）*6*28*28=122304；

详细说明：对输入图像进行第一次卷积运算（使用 6 个大小为 5*5 的卷积核），得到 6 个 C1 特征图（6 个大小为 28*28 的 feature maps, 32-5+1=28）。我们再来看看需要多少个参数，卷积核的大小为 5*5，总共就有 6*（5*5+1）=156 个参数，其中+1 是表示一个核有一个 bias。对于卷积层 C1，C1 内的每个像素都与输入图像中的 5*5 个像素和 1 个 bias 有连接，所以总共有 156*28*28=122304 个连接（connection）。有 122304 个连接，但是我们只需要学习 156 个参数，主要是通过权值共享实现的。

（3）、S2 层-池化层（下采样层）

输入：28*28

采样区域：2*2

采样方式：4 个输入相加，乘以一个可训练参数，再加上一个可训练偏置。结果通过 sigmoid

采样种类：6

输出 featureMap 大小：14*14（28/2）

神经元数量：14*14*6

连接数：（2*2+1）*6*14*14

S2 中每个特征图的大小是 C1 中特征图大小的 1/4。

详细说明：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 S2，6 个 14*14 的 特征图（28/2=14）。S2 这个 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 5x14x14x6=5880 个连接。

（4）、C3 层-卷积层

输入：S2 中所有 6 个或者几个特征 map 组合

卷积核大小：5*5

卷积核种类：16

输出 featureMap 大小：10*10 (14-5+1)=10

C3 中的每个特征 map 是连接到 S2 中的所有 6 个或者几个特征 map 的，表示本层的特征 map 是上一层提取到的特征 map 的不同组合

存在的一个方式是：C3 的前 6 个特征图以 S2 中 3 个相邻的特征图子集为输入。接下来 6 个特征图以 S2 中 4 个相邻特征图子集为输入。然后的 3 个以不相邻的 4 个特征图子集为输入。最后一个将 S2 中所有特征图为输入。

则：可训练参数：

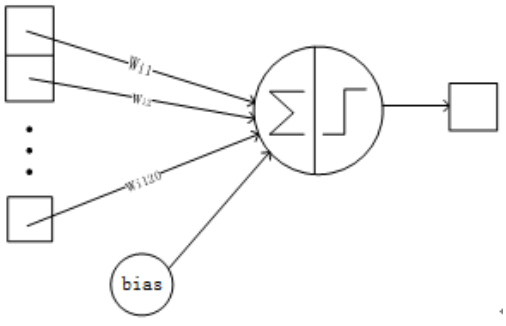
6*(3*5*5+1)+6*(4*5*5+1)+3*(4*5*5+1)+1*(6*5*5+1)=1516

连接数：10*10*1516=151600

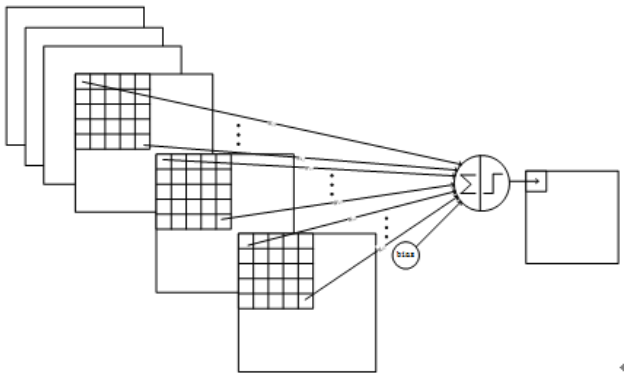
详细说明：第一次池化之后是第二次卷积，第二次卷积的输出是 C3，16 个 10x10 的特征图，卷积核大小是 5*5. 我们知道 S2 有 6 个 14*14 的特征图，怎么从 6 个特征图得到 16 个特征图了？ 这里是通过对 S2 的特征图特殊组合计算得到的 16 个特征图。具体如下：

	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

C3 的前 6 个 feature map（对应上图第一个红框的 6 列）与 S2 层相连的 3 个 feature map 相连接（上图第一个红框），后面 6 个 feature map 与 S2 层相连的 4 个 feature map 相连接（上图第二个红框），后面 3 个 feature map 与 S2 层部分不相连的 4 个 feature map 相连接，最后一个与 S2 层的所有 feature map 相连。卷积核大小依然为 5*5，所以总共有 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。



C3 与 S2 中前 3 个图相连的卷积结构如下图所示：



上图对应的参数为 3*5*5+1，一共进行 6 次卷积得到 6 个特征图，所以有 6*（3*5*5+1）参数。 为什么采用上述这样的组合了？论文中说有两个原因：1）减少参数，2）这种不对称的组合连接的方式有利于提取多种组合特征。

（5）、S4 层-池化层（下采样层）

输入：10*10

采样区域：2*2

采样方式：4 个输入相加，乘以一个可训练参数，再加上一个可训练偏置。结果通过 sigmoid

采样种类：16

输出 featureMap 大小：5*5（10/2）

神经元数量：5*5*16=400

连接数：16*（2*2+1）*5*5=2000

S4 中每个特征图的大小是 C3 中特征图大小的 1/4

详细说明：S4 是 pooling 层，窗口大小仍然是 2*2，共计 16 个 feature map，C3 层的 16 个 10x10 的图分别进行以 2x2 为单位的池化得到 16 个 5x5 的特征图。有 5x5x5x16=2000 个连接。连接的方式与 S2 层类似。

（6）、C5 层-卷积层

输入：S4 层的全部 16 个单元特征 map（与 s4 全相连）

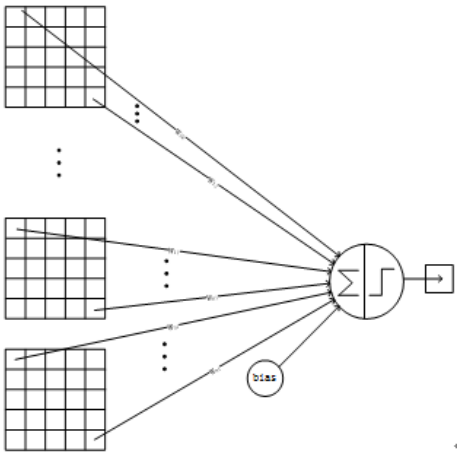
卷积核大小：5*5

卷积核种类：120

输出 featureMap 大小：1*1（5-5+1）

可训练参数/连接：120*（16*5*5+1）=48120

详细说明：C5 层是一个卷积层。由于 S4 层的 16 个图的大小为 5x5，与卷积核的大小相同，所以卷积后形成的图的大小为 1x1。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(5x5x16+1)x120 = 48120 个参数，同样有 48120 个连接。C5 层的网络结构如下：



（7）、F6 层-全连接层

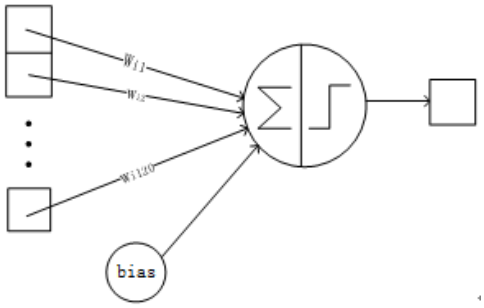
输入：c5 120 维向量

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，结果通过 sigmoid 函数输出。

可训练参数:84*(120+1)=10164

详细说明：6 层是全连接层。F6 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，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)x84=10164。ASCII 编码图如下：

F6 层的连接方式如下：

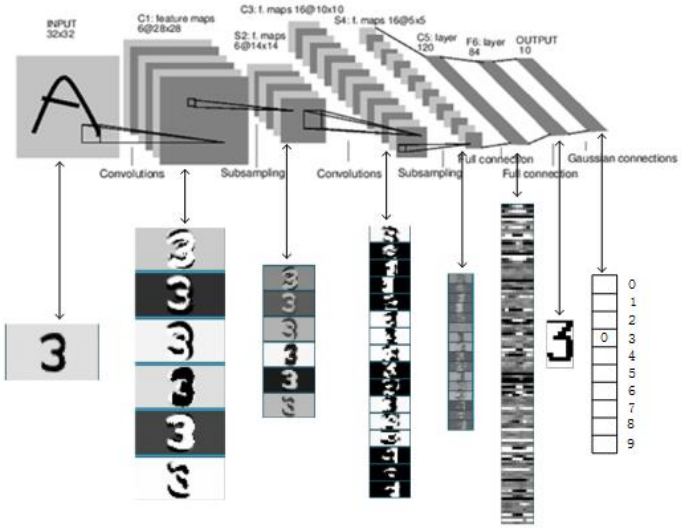


（8）、Output 层-全连接层

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。



上图是 LeNet-5 识别数字 3 的过程。

二、具体步骤及过程

1. 下载并加载数据，并做出一定的预先处理：

由于 MNIST 数据集图片尺寸是 28x28 单通道的，而 LeNet-5 网络输入 Input 图片尺寸是 32x32，因此使用 transforms.Resize 将输入图片尺寸调整为 32x32。

标准化（Normalization）是神经网络对数据的一种经常性操作。标准化处理指的是：样本减去它的均值，再除以它的标准差，最终样本将呈现均值为 0 方差为 1 的数据分布。

```
pipline_train = transforms.Compose([
    #随机旋转图片
    transforms.RandomHorizontalFlip(),
    #将图片尺寸 resize 到 32x32
    transforms.Resize((32,32)),
    #将图片转化为 Tensor 格式
    transforms.ToTensor(),
    #正则化(当模型出现过拟合的情况时，用来降低模型的复杂度)
    transforms.Normalize((0.1307,),(0.3081,))
])
pipline_test = transforms.Compose([
    #将图片尺寸 resize 到 32x32
    transforms.Resize((32,32)),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,),(0.3081,))
])
#下载数据集
train_set = datasets.MNIST(root="./data", train=True, download=True, transform=pipline_train)
test_set = datasets.MNIST(root="./data", train=False, download=True, transform=pipline_test)
#加载数据集
trainloader = torch.utils.data.DataLoader(train_set, batch_size=64, shuffle=True)
testloader = torch.utils.data.DataLoader(test_set, batch_size=32, shuffle=False)
```

2. 搭建 LeNet-5 神经网络结构，并定义前向传播的过程

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.relu = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.maxpool2 = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16*5*5, 120)
```

```
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.maxpool1(x)
        x = self.conv2(x)
        x = self.maxpool2(x)
        x = x.view(-1, 16*5*5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        output = F.log_softmax(x, dim=1)
        return output
```

3. 将定义好的网络结构搭载到 GPU/CPU，并定义优化器，定义训练过程

```
def train_runner(model, device, trainloader, optimizer, epoch):
    #训练模型，启用 BatchNormalization 和 Dropout，将 BatchNormalization 和 Dropout 置为True
    model.train()
    total = 0
    correct =0.0


    #enumerate 迭代已加载的数据集,同时获取数据和数据下标
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        #把模型部署到 device 上
        inputs, labels = inputs.to(device), labels.to(device)
        #初始化梯度
        optimizer.zero_grad()
        #保存训练结果
        outputs = model(inputs)
        #计算损失和
        #多分类情况通常使用 cross_entropy(交叉熵损失函数)，而对于二分类问题，通常使用 sigmod
        loss = F.cross_entropy(outputs, labels)
        #获取最大概率的预测结果
        #dim=1 表示返回每一行的最大值对应的列下标
        predict = outputs.argmax(dim=1)
        total += labels.size(0)
        correct += (predict == labels).sum().item()
        #反向传播
        loss.backward()
```

```
            #更新参数
            optimizer.step()
            if i % 1000 == 0:
                #loss.item()表示当前 loss 的数值
                print("Train Epoch{} \t Loss: {:.6f}, accuracy: {:.6f}%".format(epoch, loss.item(),
100*(correct/total)))
                Loss.append(loss.item())
                Accuracy.append(correct/total)
    return loss.item(), correct/total
```

4. 定义测试过程，并运行代码

```
def test_runner(model, device, testloader):
    #模型验证，必须要写，否则只要有输入数据，即使不训练，它也会改变权值
    #因为调用 eval()将不启用 BatchNormalization 和 Dropout, BatchNormalization 和
Dropout 置为 False
    model.eval()
    #统计模型正确率，设置初始值
    correct = 0.0
    test_loss = 0.0
    total = 0
    #torch.no_grad 将不会计算梯度，也不会进行反向传播
    with torch.no_grad():
        for data, label in testloader:
            data, label = data.to(device), label.to(device)
            output = model(data)
            test_loss += F.cross_entropy(output, label).item()
            predict = output.argmax(dim=1)
            #计算正确数量
            total += label.size(0)
            correct += (predict == label).sum().item()
        #计算损失值
        print("test_avarage_loss: {:.6f}, accuracy: {:.6f}%".format(test_loss/total,
100*(correct/total)))

# 调用
epoch = 5
Loss = []
Accuracy = []
for epoch in range(1, epoch+1):

print("start_time",time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time())))
    loss, acc = train_runner(model, device, trainloader, optimizer, epoch)
    Loss.append(loss)
    Accuracy.append(acc)
```

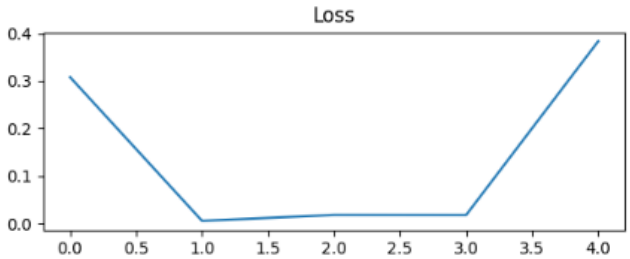
```
        test_runner(model, device, testloader)
        print("end_time:
",time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time())),'\n')


print('Finished Training')
plt.subplot(2,1,1)
plt.plot(Loss)
plt.title('Loss')
plt.show()
plt.subplot(2,1,2)
plt.plot(Accuracy)
plt.title('Accuracy')
plt.show()
```

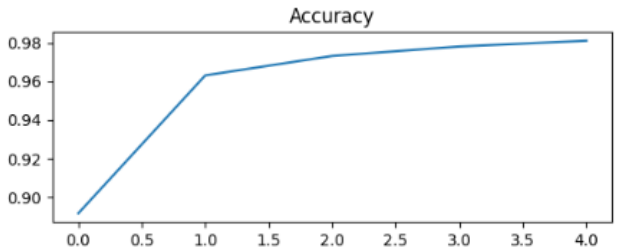
三、结果展示

（1）、训练过程（epoch）

```
start_time 2023-04-17 21:12:55
Train Epoch1     Loss: 2.302818, accuracy: 14.062500%
test_avarage_loss: 0.004113, accuracy: 95.820000%
end_time:  2023-04-17 21:13:21

start_time 2023-04-17 21:13:21
Train Epoch2     Loss: 0.188924, accuracy: 92.187500%
test_avarage_loss: 0.002700, accuracy: 97.210000%
end_time:  2023-04-17 21:13:42

start_time 2023-04-17 21:13:42
Train Epoch3     Loss: 0.075157, accuracy: 98.437500%
test_avarage_loss: 0.002713, accuracy: 97.240000%
end_time:  2023-04-17 21:14:03

start_time 2023-04-17 21:14:03
Train Epoch4     Loss: 0.035005, accuracy: 98.437500%
test_avarage_loss: 0.002350, accuracy: 97.790000%
end_time:  2023-04-17 21:14:24

start_time 2023-04-17 21:14:24
Train Epoch5     Loss: 0.099896, accuracy: 98.437500%
test_avarage_loss: 0.001895, accuracy: 98.200000%
end_time:  2023-04-17 21:14:45

Finished Training
```

（2）、经历 5 次 epoch 的 loss 和 accuracy 曲线如下:





三、代码运行说明

（1）、代码使用 python 编写，运行时请使用 pycharm 等 python 编译器。

（2）、lenet_5 总共包含三个文件:data,network,testtt,train,以及 dataset2d.py,exp42.py。

（3）、使用时直接运行 exp42.py 文件,此外 data 包含 MNIST 数据集,network 内定义 lenet_5 网络结构，testt 和 train 分别定义测试和训练函数。

。