计算机视觉应用与实践（二）

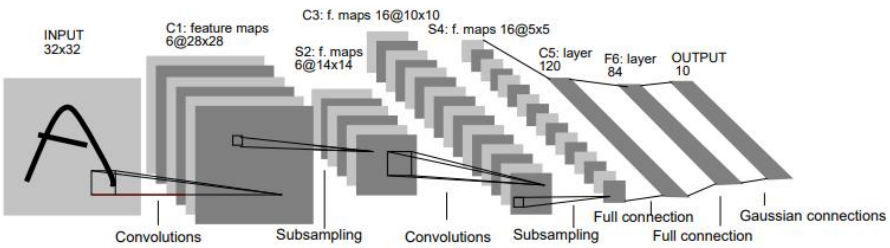
目录

一、 实验目的

1、 理解 LeNet5 的网络结构。
2、 在手写数字识别数据集上训练和测试 LeNet5，评估其性能和准确性，并尝试改进。

二、 实验原理

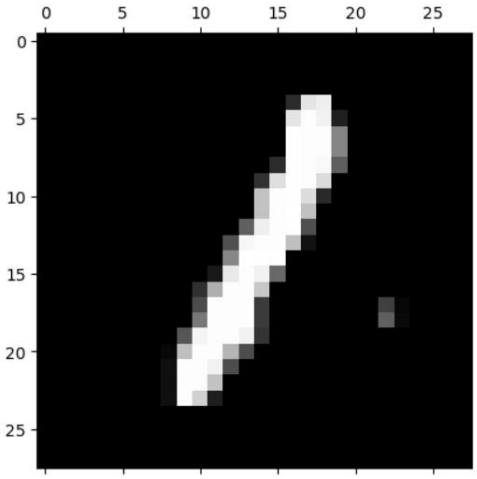
LeNet5 是 LeCun 在 1989 年提出的网络结构，是卷积神经网络（CNN）的鼻祖。它的具体网络结构如下图所示。



LeNet5 由两个卷积层，两个池化层以及三个线性层构成。具体来说，对于一个输入形状为[1,32,32]的灰度图像，输入到第一个卷积层后会变成[6,28,28]，然后经过池化层变成[6,14,14]。再然后经过第二个卷积层和池化层变成[16,5,5]。随后展平输入到三个线性层得到最终 10 个分类结果。模型最后选用的 10 个数中最大作为分类结果。

三、 数据集

本次实验使用的数据集是 MNIST 数据集。MNIST 数据集来自美国国家标准与技术研究所, National Institute of Standards and Technology (NIST)，由来自 250 个不同人手写的数字构成。其中每张图像的大小为 28*28，数据中共有 70000 张手写数字照片，其中 60000 张作为训练集，10000 张作为测试集。数据集中的图片图下所示



四、 程序代码

LeNet5 模型代码，因为 MNIST 数据集中的图片被裁剪为 28*28 而非最初的 32*32，所以在第一个卷积层中进行了 padding。

```
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5,self).__init__()
        self.conv1 = nn.Conv2d(1,6,5,padding=2)
        self.pool1 = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(6,16,5)
        self.pool2 = nn.MaxPool2d(2,2)
        self.fc1 = nn.Linear(16*5*5,120)
        self.fc2 = nn.Linear(120,84)
        self.fc3 = nn.Linear(84,10)
        self.activate = nn.Sigmoid()

    def forward(self,x):
#         print(x.shape)
        x = self.pool1(self.activate(self.conv1(x)))
        x = self.pool2(self.activate(self.conv2(x)))
#         print(x.shape)
        x = x.view(-1, 16*5*5)
```

```
        x = self.activate(self.fc1(x))
        x = self.activate(self.fc2(x))
        x = self.fc3(x)
        return x
```

修改后的 LeNet5 代码，主要是将激活函数换成 ReLU，然后在引入 BN 层和 Dropout 减少模型的过拟合。

```
class LeNet5Plus(nn.Module):
    def __init__(self):
        super(LeNet5Plus,self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1,6,5,padding=2),
            nn.BatchNorm2d(6),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(6,16,5),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )
        self.linear = nn.Sequential(
            nn.Flatten(),
            nn.Linear(16*5*5,120),
            nn.ReLU(),
            nn.Linear(120,84),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(84,10)


        )
    def forward(self,x):
        x = self.conv(x)
        x = self.linear(x)
        return x
```

训练函数

```
def train_epoch(net,train_loader,optim,loss):
    net.train()
#     print(net.fc3.weight)
    losses = 0
    L = 0
    for data in train_loader:
        X,y = data
        X,y = X.to(device),y.to(device)
```

```
#         print(X.shape,y.shape)
        optim.zero_grad()
        y_pre = net(X)
#         print(y_pre.shape,y.shape)
        l = loss(y_pre,y)
        L+=X.shape[0]
        l.backward()
        losses+=l.item()
        optim.step()
    net.eval()
    return losses/L
def train(network,epochs,train_loader,val_loader):
    LOSS = []
    ACC_TRAIN = []
    ACC_VAL = []
    acc = 0
    best_model = None
    for epoch in range(epochs):
        epoch_loss = train_epoch(network,train_loader,optimizer,loss)
        acc_val = test(network,val_loader)
        acc_train = test(network,train_loader)
        LOSS.append(epoch_loss)
        ACC_TRAIN.append(acc_train)
        ACC_VAL.append(acc_val)
        if acc_val>acc:
            acc = acc_val
            best_model = network
    plt.plot(range(epochs),LOSS,label='loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend()
    plt.show()
    plt.plot(range(epochs),ACC_TRAIN,label='train acc')
    plt.plot(range(epochs),ACC_VAL,label='val acc')
    plt.ylabel('acc')
    plt.xlabel('epoch')
    plt.legend()
    plt.show()
    return best_model
```

测试函数

```
def test(net,test_loader):
    correct = 0
    total = 0
    with torch.no_grad():
```

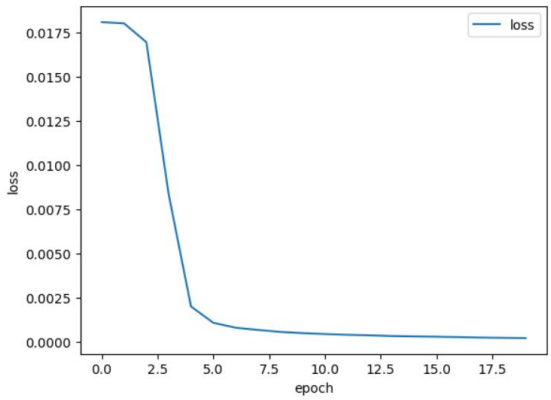
```
        for data in test_loader:
            X,y = data
            X,y = X.to(device),y.to(device)
            y_pre = net(X)
            _, predicted = torch.max(y_pre.data, dim=1)
            total += y.size(0)
            correct += (predicted == y).sum().item()
    return 100 * correct / total
```

五、 实验结果

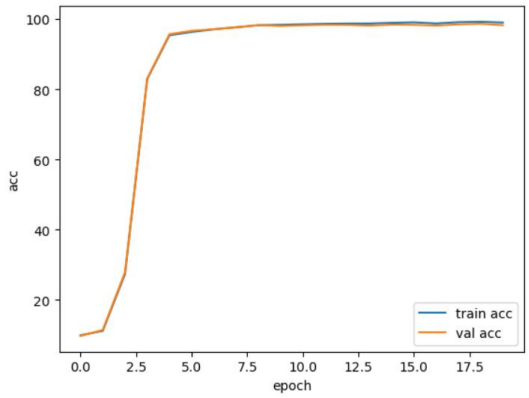
在训练时，我们将训练集的 10%作为验证集，最终选取的模型为在验证集上准确率最高的模型。

对于 LeNet5，在实验时选用的学习率为 0.9，优化器为 SGD，激活函数为 Sigmoid。

训练损失函数如下所示



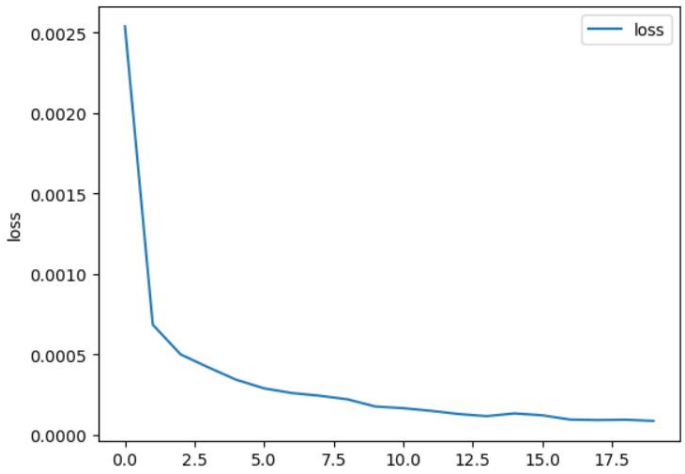
训练集和测试集的准确率如下所示



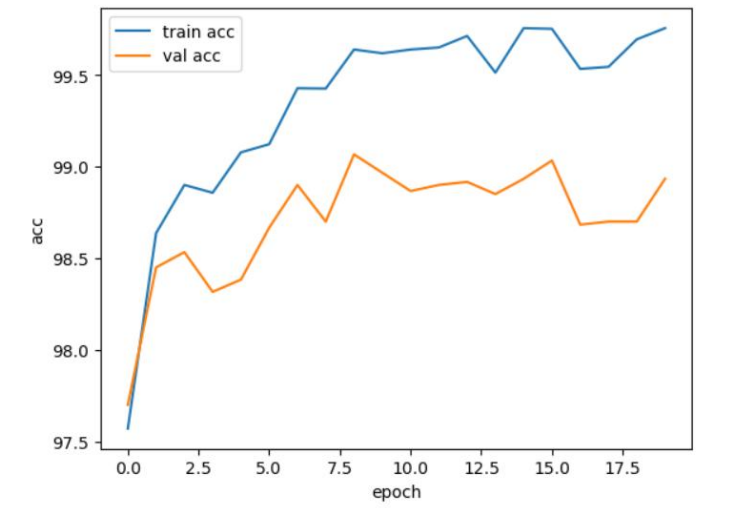
最终分类准确率为 98.62%

对于 LeNet5Plus，在实验时选用的学习率为 1e-3，优化器为 Adam，激活函数为 ReLU。

训练损失函数如下所示



训练集和测试集的准确率如下所示



最终分类准确率为 99.12%。结果显示，改进后的模型在 MNIST 上的分类精度得到提升。

六、 实验总结

通过这次实验，我学习了 LeNet5 模型的网络结构，并使用该模型完成了对 MNIST 模型的分类任务。此外，我还使用了对原有模型做了一定的改进，有效的提高了模型在 MNIST 数据集上的分类精度。