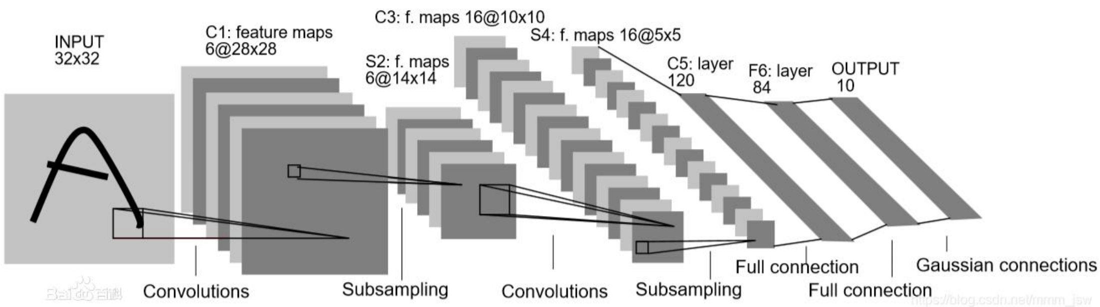
计算机视觉实践-练习 2

122106222784 贺梦瑶

1、实验目标

实现 LeNet-5 在 MNIST 数据集上的训练和测试

2、模型介绍



LeNet-5 具有一个输入层，两个卷积层，两个池化层，3 个全连接层（其中最后一个全连接层为输出层），一共由 7 层组成，分别是 C1、C3、C5 卷积层，S2、S4 降采样层（降采样层又称池化层），F6 为一个全连接层，输出是一个高斯连接层，该层使用 softmax 函数对输出图像进行分类。为了对应模型输入结构，将 MNIST 中的 28* 28 的图像扩展为 32* 32 像素大小。下面对每一层进行详细介绍。C1 卷积层由 6 个大小为 5* 5 的不同类型的卷积核组成，卷积核的步长为 1，没有零填充，卷积后得到 6 个 28* 28 像素大小的特征图；S2 为最大池化层，池化区域大小为 2* 2，步长为 2，经过 S2 池化后得到 6 个 14* 14 像素大小的特征图；C3 卷积层由 16 个大小为 5* 5 的不同卷积核组成，卷积核的步长为 1，没有零填充，卷积后得到 16 个 10* 10 像素大小的特征图；S4 最大池化层，池化区域大小为 2* 2，步长为 2，经过 S2 池化后得到 16 个 5* 5 像素大小的特征图；C5 卷积层由 120 个大小为 5* 5 的不同卷积核组成，卷积核的步长为 1，没有零填充，卷积后得到 120 个 1* 1 像素大小的特征图；将 120 个 1* 1 像素大小的特征图拼接起来作为 F6 的输入，F6 为一个由 84 个神经元组成的全连接隐藏层，激活函数使用 sigmoid 函数；最后一层输出层是一个由 10 个神经元组成的 softmax 高斯连接层，可以用来做分类任务。

3、实现说明

```
# 定义各个层的功能
class Model(Module):
    def __init__(self):
        super(Model, self).__init__()
        # 池化层
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)
        # 池化层
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)
        # 全连接层
```

```python
        self.fc1 = nn.Linear(256, 120)
        self.relu3 = nn.ReLU()
        # 全连接层
        self.fc2 = nn.Linear(120, 84)
        self.relu4 = nn.ReLU()
        # 全连接层
        self.fc3 = nn.Linear(84, 10)
        self.relu5 = nn.ReLU()

    def forward(self, x):
        # 池化层
        y = self.conv1(x)
        y = self.relu1(y)
        y = self.pool1(y)
        # 池化层
        y = self.conv2(y)
        y = self.relu2(y)
        y = self.pool2(y)
        y = y.view(y.shape[0], -1)
        # 全连接层
        y = self.fc1(y)
        y = self.relu3(y)
        # 全连接层
        y = self.fc2(y)
        y = self.relu4(y)
        # 全连接层
        y = self.fc3(y)
        y = self.relu5(y)
        return y
```

```python
# 参数更新，模型训练
        for idx, (train_x, train_label) in enumerate(train_loader):
            train_x = train_x.to(device)
            train_label = train_label.to(device)
            sgd.zero_grad()
            predict_y = model(train_x.float())
            loss = loss_fn(predict_y, train_label.long())
            loss.backward()
```

```
        sgd.step()
```

```
# 测试
        for idx, (test_x, test_label) in enumerate(test_loader):
            test_x = test_x.to(device)
            test_label = test_label.to(device)
            predict_y = model(test_x.float()).detach()
            predict_y =torch.argmax(predict_y, dim=-1)
            current_correct_num = predict_y == test_label
            all_correct_num += np.sum(current_correct_num.to('cpu').numpy(), axis=-1)
            all_sample_num += current_correct_num.shape[0]
        acc = all_correct_num / all_sample_num
```

4、结果展示

```
accuracy: 0.873
accuracy: 0.937
accuracy: 0.952
accuracy: 0.958
accuracy: 0.968
accuracy: 0.971
accuracy: 0.974
accuracy: 0.975
accuracy: 0.976
accuracy: 0.977
accuracy: 0.977
accuracy: 0.978
accuracy: 0.978
accuracy: 0.980
accuracy: 0.981
accuracy: 0.982
```

```
accuracy: 0.979
accuracy: 0.981
accuracy: 0.983
accuracy: 0.980
accuracy: 0.982
accuracy: 0.981
accuracy: 0.982
accuracy: 0.980
accuracy: 0.986
accuracy: 0.981
accuracy: 0.982
accuracy: 0.983
accuracy: 0.983
accuracy: 0.984
accuracy: 0.982
```

```
accuracy: 0.987
accuracy: 0.983
accuracy: 0.986
accuracy: 0.983
accuracy: 0.980
accuracy: 0.982
accuracy: 0.985
accuracy: 0.988
accuracy: 0.980
accuracy: 0.983
accuracy: 0.986
accuracy: 0.984
accuracy: 0.986
accuracy: 0.988
accuracy: 0.987
accuracy: 0.988
```

```
accuracy: 0.776
accuracy: 0.936
accuracy: 0.962
accuracy: 0.967
accuracy: 0.973
accuracy: 0.975
accuracy: 0.977
accuracy: 0.979
accuracy: 0.979
accuracy: 0.981
accuracy: 0.980
accuracy: 0.980
accuracy: 0.981
accuracy: 0.981
accuracy: 0.981
```

5、运行说明

运行 train.py 文件