中山大学计算机学院

人工智能

本科生实验报告

课程名称：Artificial Intelligence

学号	22336259	姓名	谢宇桐

一、 实验题目

深度学习——中药图片分类任务

1. 算法原理

本次实验要求我们利用 pytorch 框架搭建神经网络实现中药图片分类

卷积神经网络

卷积神经网络（Convolutional Neural Networks, CNN）是一类包含卷积计算且具有深度结构的前馈神经网络，是深度学习的代表算法之一。卷积神经网络具有表征学习能力，能够按其阶层结构对输入信息进行平移不变分类。

整个过程需要在如下几层进行运算：

1.输入层：输入图像等信息
2.卷积层：用来提取图像的底层特征
3.池化层：防止过拟合，将数据维度减小
4.全连接层：汇总卷积层和池化层得到的图像的底层特征和信息
5.输出层：根据全连接层的信息得到概率最大的结果

下面我们介绍一下这五层：

1.输入层

这一层的主要工作就是输入图像等信息，对于输入图像，首先要将其转换为对应的二维矩阵，这个二位矩阵就是由图像每一个像素的像素值大小组成的。

2.卷积层

想要提取图片其中特征，卷积操作会为存在特征的区域确定一个高值，否则确定一个低值。这个过程需要通过计算其与卷积核（Convolution Kernel）的乘积值来确定。通过整个卷积过程又得到一个新的二维矩阵，此二维矩阵也被称为特征图。通过特征图，其可以提取到图片特征。有几个卷积核就有多少个特征图。如给定输入图像 I 和卷积核 K，则卷积操作定义为：

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

3.池化层

池化层又称为下采样，当我们进行卷积操作后，再将得到的特征图进行特征提取，将其中最具有代表性的特征提取出来，可以起到减小过拟合和降低维度的作用。池化操作通常包括最大池化和平均池化，其中最大池化定义为：

$$P(i,j) = \max_{(m,n)\in W} I(m,n)$$

4.全连接层

全连接层位于网络的末端，其将前一层的所有输出连接到每个神经元上。这一层的目的是将学习到的抽象高级特征映射到最终的输出类别上，如在分类任务中映射到具体的类标签。

在建模过程中我们与一些重要环节：

1.数据预处理：对于输入数据进行预处理，如图像数据进行归一化、中心化等操作，可以提高模型的训练效果和泛化能力

2.激活函数选择：选择合适的激活函数，可以提高模型的非线性表达能力和训练速度。在本次实验中我们主要用 ReLU。

3.前向传播：在 CNN 中，数据通过网络层进行前向传播，从输入层到输出层。每一层的输出作为下一层的输入，最终结果通过 softmax 层输出，这一层将连续值转换为概率分布。

在训练过程中，CNN 通过使用优化器更新模型参数最小化损失函数来调整其参数，常用的损失函数是交叉熵损失函数，适用于多分类问题。以及通过反向传播计算损失函数相对于模型参数的梯度。最后将每个结果的损失和准确率输出，并将其可视化。

2. 关键代码展示（可选）

```
# 数据预处理
def trans():
    return transforms.Compose([
        transforms.ToTensor(),  # 将图像转换为Tensor
        transforms.Resize((256, 256)),  # 将图像大小调整为256x256
        transforms.CenterCrop(224),  # 中心裁剪到224x224
        transforms.Normalize( mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225])  # 标准化处理
    ])
```

```python
# 数据加载
def load_data(data_dir):
    # 获取数据预处理方法
    transform = trans()
    # 加载训练集
    train_d = datasets.ImageFolder(root=f'{data_dir}/train', transform=transform)
    # 加载测试集
    test_d = get_dataset(img_dir=f'{data_dir}/test', transform=transform)

    # 创建训练数据加载器
    train_l = DataLoader(train_d, batch_size=32, shuffle=True, num_workers=4)
    # 创建测试数据加载器
    test_l = DataLoader(test_d, batch_size=32, shuffle=False, num_workers=4)

    return train_l, test_l
```

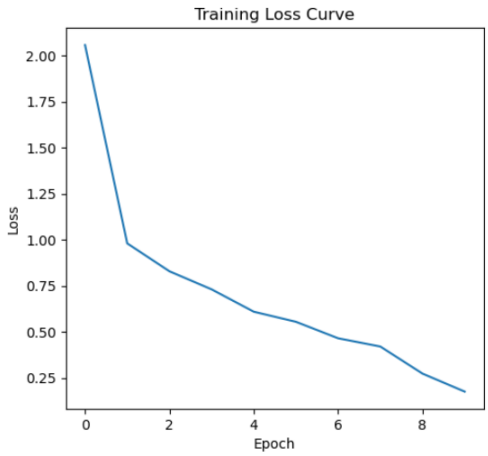
```python
# 神经网络定义
class CNN(nn.Module):
    def __init__(self, num_classes=5):  # 假设有5个类别
        super(CNN, self).__init__()
        # 第一层卷积层，输入通道3（RGB图像），输出通道32，卷积核大小3x3，填充1
        self.con1 = nn.Conv2d( in_channels: 3,  out_channels: 32,  kernel_size: 3, padding=1)
        # 第二层卷积层，输入通道32，输出通道64，卷积核大小3x3，填充1
        self.con2 = nn.Conv2d( in_channels: 32,  out_channels: 64,  kernel_size: 3, padding=1)
        # 第三层卷积层，输入通道64，输出通道128，卷积核大小3x3，填充1
        self.con3 = nn.Conv2d( in_channels: 64,  out_channels: 128,  kernel_size: 3, padding=1)
        # 最大池化层，池化窗口大小2x2
        self.pool = nn.MaxPool2d( kernel_size: 2,  stride: 2)
        # 全连接层，输入大小128*28*28，输出512
        self.connect1 = nn.Linear(128 * 28 * 28,  out_features: 512)  # 输入层维度根据实际情况调整
        # 全连接层，输入512，输出类别数
        self.connect2 = nn.Linear( in_features: 512, num_classes)

    def forward(self, x):
        # 前向传播：卷积层1 -> ReLU -> 池化层
        x = self.pool(F.relu(self.con1(x)))
        # 前向传播：卷积层2 -> ReLU -> 池化层
        x = self.pool(F.relu(self.con2(x)))
        # 前向传播：卷积层3 -> ReLU -> 池化层
        x = self.pool(F.relu(self.con3(x)))
        # 展平特征图以输入全连接层
        x = x.view(-1, 128 * 28 * 28)  # Flatten layer
        # 前向传播：全连接层1 -> ReLU
        x = F.relu(self.connect1(x))
        # 前向传播：全连接层2（输出层）
        x = self.connect2(x)
        return x
```

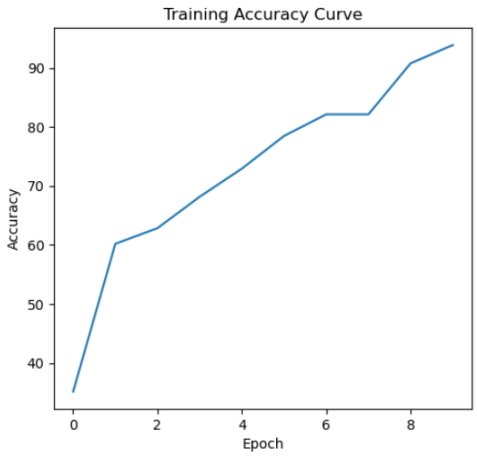
```python
# 训练模型
def ModelTrain(model, train_l, criterion, optim, num_epochs=10):
    model.train()
    losses = []  # 记录每个epoch的损失
    accuracies = []  # 记录每个epoch的准确率
    for e in range(num_epochs):
        running_loss = 0.0
        correct = total = 0
        for i, l in train_l:
            # 将数据加载到设备（GPU或CPU）
            i, l = i.to(device), l.to(device)
            optim.zero_grad()  # 梯度清零
            outputs = model(i)  # 前向传播
            loss = criterion(outputs, l)  # 计算损失
            loss.backward()  # 反向传播
            optim.step()  # 更新权重
            running_loss += loss.item()

            _, predi = torch.max(outputs, 1)  # 获取预测标签
            total += l.size(0)
            correct += (predi == l).sum().item()  # 统计正确预测的数量

        loss_e = running_loss / len(train_l)  # 计算每个epoch的平均损失
        accuracy_e = 100 * correct / total  # 计算每个epoch的准确率
        losses.append(loss_e)
        accuracies.append(accuracy_e)
        print(f'Epoch {e + 1}/{num_epochs}, Loss: {loss_e:.4f}, Accuracy: {accuracy_e:.2f}%')

    return losses, accuracies
```

```python
# 评估模型
def evaluate_model(model, test_l):
    model.eval()
    total = correct = 0
    with torch.no_grad():  # 评估时不需要计算梯度
        for imgs, labels in test_l:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = model(imgs)
            _, predi = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predi == labels).sum().item()
    accuracy = 100 * correct / total  # 计算测试集的准确率
    print(f'Accuracy on the test set: {accuracy:.2f}%')
    return accuracy
```

二、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）






```
D:\Anaconda\python.exe D:\homework\AI\第13周实验：深度学习\第13周实验：深度学习\zy.py
Epoch 1/10, Loss: 1.9263, Accuracy: 40.91%
Epoch 2/10, Loss: 0.8782, Accuracy: 60.98%
Epoch 3/10, Loss: 0.6820, Accuracy: 67.52%
Epoch 4/10, Loss: 0.6137, Accuracy: 71.84%
Epoch 5/10, Loss: 0.5323, Accuracy: 77.38%
Epoch 6/10, Loss: 0.4882, Accuracy: 80.71%
Epoch 7/10, Loss: 0.4129, Accuracy: 83.37%
Epoch 8/10, Loss: 0.3594, Accuracy: 85.59%
Epoch 9/10, Loss: 0.2703, Accuracy: 90.69%
Epoch 10/10, Loss: 0.1983, Accuracy: 92.90%
Accuracy on the test set: 90.00%
```

```
D:\Anaconda\python.exe D:\homework\AI\第13周实验：深度学习\第13周实验：深度学习\zy.py
Epoch 1/10, Loss: 2.0730, Accuracy: 34.59%
Epoch 2/10, Loss: 0.9211, Accuracy: 61.97%
Epoch 3/10, Loss: 0.6960, Accuracy: 69.07%
Epoch 4/10, Loss: 0.6187, Accuracy: 73.84%
Epoch 5/10, Loss: 0.5571, Accuracy: 78.05%
Epoch 6/10, Loss: 0.3988, Accuracy: 83.59%
Epoch 7/10, Loss: 0.2996, Accuracy: 88.36%
Epoch 8/10, Loss: 0.2284, Accuracy: 92.35%
Epoch 9/10, Loss: 0.1674, Accuracy: 94.57%
Epoch 10/10, Loss: 0.0988, Accuracy: 96.34%
Accuracy on the test set: 70.00%
```

```
D:\Anaconda\python.exe D:\homework\AI\第13周实验：深度学习\第13周实验：深度学习\zy.py
Epoch 1/20, Loss: 1.5467, Accuracy: 44.01%
Epoch 2/20, Loss: 0.7368, Accuracy: 67.41%
Epoch 3/20, Loss: 0.6744, Accuracy: 68.18%
Epoch 4/20, Loss: 0.5742, Accuracy: 76.27%
Epoch 5/20, Loss: 0.4492, Accuracy: 81.37%
Epoch 6/20, Loss: 0.3406, Accuracy: 87.03%
Epoch 7/20, Loss: 0.2698, Accuracy: 89.58%
Epoch 8/20, Loss: 0.1558, Accuracy: 95.57%
Epoch 9/20, Loss: 0.1584, Accuracy: 95.45%
Epoch 10/20, Loss: 0.0596, Accuracy: 98.00%
Epoch 11/20, Loss: 0.0583, Accuracy: 98.45%
Epoch 12/20, Loss: 0.0225, Accuracy: 99.33%
Epoch 13/20, Loss: 0.0095, Accuracy: 99.89%
Epoch 14/20, Loss: 0.0032, Accuracy: 100.00%
Epoch 15/20, Loss: 0.0010, Accuracy: 100.00%
Epoch 16/20, Loss: 0.0007, Accuracy: 100.00%
Epoch 17/20, Loss: 0.0005, Accuracy: 100.00%
Epoch 18/20, Loss: 0.0005, Accuracy: 100.00%
Epoch 19/20, Loss: 0.0004, Accuracy: 100.00%
Epoch 20/20, Loss: 0.0003, Accuracy: 100.00%
Accuracy on the test set: 90.00%

进程已结束，退出代码为 0
```

```
D:\Anaconda\python.exe D:\homework\AI\第13周实验：深度学习\第13周实验：深度学习\zy.py
Epoch 1/20, Loss: 1.9409, Accuracy: 44.12%
Epoch 2/20, Loss: 0.7923, Accuracy: 66.74%
Epoch 3/20, Loss: 0.6015, Accuracy: 73.73%
Epoch 4/20, Loss: 0.5129, Accuracy: 78.94%
Epoch 5/20, Loss: 0.4387, Accuracy: 83.15%
Epoch 6/20, Loss: 0.4086, Accuracy: 84.26%
Epoch 7/20, Loss: 0.3060, Accuracy: 89.14%
Epoch 8/20, Loss: 0.2253, Accuracy: 92.46%
Epoch 9/20, Loss: 0.0922, Accuracy: 97.34%
Epoch 10/20, Loss: 0.0474, Accuracy: 99.00%
Epoch 11/20, Loss: 0.0955, Accuracy: 96.78%
Epoch 12/20, Loss: 0.0304, Accuracy: 99.33%
Epoch 13/20, Loss: 0.0064, Accuracy: 100.00%
Epoch 14/20, Loss: 0.0024, Accuracy: 100.00%
Epoch 15/20, Loss: 0.0012, Accuracy: 100.00%
Epoch 16/20, Loss: 0.0008, Accuracy: 100.00%
Epoch 17/20, Loss: 0.0006, Accuracy: 100.00%
Epoch 18/20, Loss: 0.0005, Accuracy: 100.00%
Epoch 19/20, Loss: 0.0004, Accuracy: 100.00%
Epoch 20/20, Loss: 0.0004, Accuracy: 100.00%
Accuracy on the test set: 100.00%

进程已结束，退出代码为 0
```

2.评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

由于测试时间问题，我们以十轮和二十轮进行测试，且因为测试的随机性，普遍准确率可以达到 90%以上。由上图可得越多次训练，其准确率越高。模型经过多次训练，逐渐学会了如何通过 CNN 从训练数据中提取特征和模式，因此训练集上的准确率逐步提高，并最终达到很高的水平。

三、 参考资料

卷积神经网络 Convolutional Neural Network | CNN
卷积神经网络（CNN）详细介绍及其原理详解