人工智能实验报告

实验7　深度学习

陳日康　信息与计算科学　**22336049**

一、实验概述

利用 pytorch 框架搭建神经网络实现中药图片分类，具体见给出的数据集和测试集。要求搭建合适的网络框架，利用训练集完成网络训练，统计网络模型的训练准确率和测试准确率，画出模型的训练过程的 loss 曲线、准确率曲线。

二、算法原理

卷积神经网络 (Convolutional Neural Networks, CNN) 算法原理主要包括以下几个方面：

1. 卷积层：卷积层是 CNN 的核心层，它通过卷积操作提取图像的特征，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。通过使用多个滤波器，卷积层可以提取出不同的特征。
2. 池化层：又称为下采样，用于减小卷积层输出的特征图的大小，将其中最具有代表性的特征提取出来，保留重要的特征。常用的池化操作包括最大池化和平均池化，它们分别取池化窗口内的最大值和平均值作为输出值。
3. 激活函数：用于引入非线性因素，增加模型的表达能力。常用的激活函数包括 ReLU，Sigmoid 和 Tanh 等，而在多分类问题中常用到 softmax。
4. 全连接层：通常位于网络的末端，用于整合卷积层和池化层提取的特征，将所有的特征都展开并进行运算得到最终的识别概率。

三、流程图

```
开始
|
|---> 导入所需库
|       |
|       |---> 导入 PyTorch、torchvision、matplotlib 等库
|
|---> 定义模型
|       |
|       |---> 定义 NewCustomModel 类
|               |
|               |---> 定义卷积层
|               |
|               |---> 定义全连接层
|
|---> 数据预处理
|       |
|       |---> 定义数据预处理变换
|               |
```

```
|             |---> 调整图像大小、转换为张量和标准化
|
|---> 加载数据集
|       |
|       |---> 使用 ImageFolder 加载训练数据
|       |
|       |---> 使用 ImageFolder 加载测试数据
|       |
|       |---> 使用 DataLoader 创建数据加载器
|
|---> 初始化模型、损失函数和优化器
|       |
|       |---> 实例化 NewCustomModel
|       |
|       |---> 定义交叉熵损失函数
|       |
|       |---> 使用 Adam 优化器
|
|---> 训练和评估过程
|       |
|       |---> 进行 25 个 epoch 的训练和评估
|               |
|               |---> 训练阶段
|               |       |
|               |       |---> 模型设置为训练模式
|               |       |
|               |       |---> 遍历训练数据，进行前向传播、计算损失、反向传播和参数更新
|               |
|               |---> 评估阶段
|               |       |
|               |       |---> 模型设置为评估模式
|               |       |
|               |       |---> 遍历测试数据，进行前向传播和准确率计算
|
|---> 记录和打印结果
|       |
|       |---> 记录每个 epoch 的训练损失和测试准确率
|       |
|       |---> 打印训练损失和测试准确率
|
|---> 绘制结果
|       |
|       |---> 绘制训练损失和测试准确率曲线
|
结束
```

四、关键代码展示

`NewCustomModel(nn.Module)` 是自定义模型类，继承自 `nn.Module`，包含卷积部分和全连接部分。

`conv_layers` 卷积层序列，通过一系列卷积、激活和池化层提取特征。

`fc_layers` 全连接层序列，通过一系列全连接、激活和 `Dropout` 层进行分类。

`forward(self, x)` 为前向传播函数，接收输入 `x`，通过卷积部分和全连接部分进行计算，并返回输出。

```
class NewCustomModel(nn.Module):
    def __init__(self):
        super(NewCustomModel, self).__init__()
        # 定义卷积部分
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),                      # 激活函数ReLU
            nn.MaxPool2d(kernel_size=2, stride=2),      # 最大池化层，池化窗口2x2，步幅2
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        # 定义全连接部分
        self.fc_layers = nn.Sequential(
            nn.Linear(256 * 14 * 14, 1024),   # 全连接层，将256*14*14的特征映射到1024个节点
            nn.ReLU(inplace=True),    # 激活函数ReLU
            nn.Dropout(0.5),          # Dropout层，丢弃50%的节点防止过拟合
            nn.Linear(1024, 512),   # 全连接层，将1024个节点映射到512个节点
            nn.ReLU(inplace=True),    # 激活函数ReLU
            nn.Dropout(0.5),          # Dropout层
            nn.Linear(512, 5)       # 全连接层，将512个节点映射到最终的5类
        )

    def forward(self, x):
        x = self.conv_layers(x)       # 通过卷积层
        x = x.view(x.size(0), -1)     # 展平操作
        x = self.fc_layers(x)         # 通过全连接层
        return x
```

数据预处理和加载数据集：

```
data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),    # 调整图像大小为224x224
    transforms.ToTensor(),            # 转换为Tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
# 标准化
])

train_data = DataLoader(
    ImageFolder(root='D:/python object/pythonProject/test/cnn_train',
transform=data_transforms),
    batch_size=64, shuffle=True       # 训练集，批量大小64，打乱顺序
```

```
)
test_data = DataLoader(
    ImageFolder(root='D:/python object/pythonProject/test/cnn_test',
transform=data_transforms),
    batch_size=64, shuffle=False      # 训练集，批量大小64，不打乱顺序
)
```

初始化模型、损失函数和优化器：

```
net = NewCustomModel()                                  # 实例化模型
loss_function = nn.CrossEntropyLoss()                   # 交叉熵损失函数
optimizer = optim.Adam(net.parameters(), lr=0.001)      # Adam优化器，学习率0.001
```

训练和评估过程：

```
train_loss_values = []         # 记录每个epoch的训练损失
test_accuracy_values = []      # 记录每个epoch的测试准确率

for epoch in range(25):
    # 训练阶段
    net.train()
    total_loss = 0.0                                    # 初始化总损失
    for images, labels in train_data:                   # 遍历训练集
        optimizer.zero_grad()                           # 清空梯度
        outputs = net(images)                           # 前向传播
        loss = loss_function(outputs, labels)           # 计算损失
        loss.backward()                                 # 反向传播
        optimizer.step()                                # 更新参数
        total_loss += loss.item()                       # 累计损失
    avg_loss = total_loss / len(train_data)             # 计算平均损失
    train_loss_values.append(avg_loss)                  # 记录平均损失
    print(f"Epoch {epoch + 1}, Training Loss: {avg_loss:.4f}")      # 打印训练损失

    net.eval()                                          # 进入评估模式
    correct_predictions = 0                             # 初始化正确预测数
    total_samples = 0                                   # 初始化总样本数
    with torch.no_grad():                               # 不计算梯度
        for images, labels in test_data:                # 遍历测试集
            outputs = net(images)                       # 前向传播
            _, preds = torch.max(outputs, 1)            # 获取预测结果
            total_samples += labels.size(0)             # 累计总样本数
            correct_predictions += (preds == labels).sum().item()       # 累计正确
预测数
    accuracy = 100 * correct_predictions / total_samples                # 计算准确
率
    test_accuracy_values.append(accuracy)                               # 记录准确
率
    print(f"Test Accuracy: {accuracy:.2f}%")

print("Training Complete.")        # 训练完成
```

五、实验结果及分析

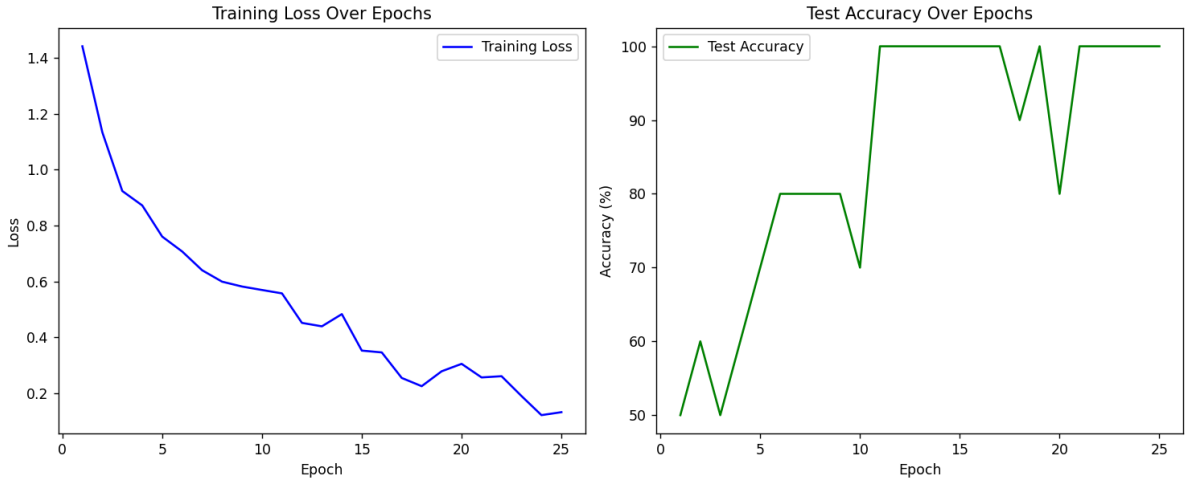
5.1 实验结果展示

```
Epoch 1, Training Loss: 1.4421
Test Accuracy: 50.00%
Epoch 2, Training Loss: 1.1340
Test Accuracy: 60.00%
Epoch 3, Training Loss: 0.9238
Test Accuracy: 50.00%
Epoch 4, Training Loss: 0.8721
Test Accuracy: 60.00%
Epoch 5, Training Loss: 0.7605
Test Accuracy: 70.00%
Epoch 6, Training Loss: 0.7072
Test Accuracy: 80.00%
Epoch 7, Training Loss: 0.6400
Test Accuracy: 80.00%
Epoch 8, Training Loss: 0.5991
Test Accuracy: 80.00%
Epoch 9, Training Loss: 0.5817
Test Accuracy: 80.00%
Epoch 10, Training Loss: 0.5694
Test Accuracy: 70.00%
Epoch 11, Training Loss: 0.5572
Test Accuracy: 100.00%
Epoch 12, Training Loss: 0.4515
Test Accuracy: 100.00%
Epoch 13, Training Loss: 0.4392
Test Accuracy: 100.00%
Epoch 14, Training Loss: 0.4829
Test Accuracy: 100.00%
Epoch 15, Training Loss: 0.3523
Test Accuracy: 100.00%
Epoch 16, Training Loss: 0.3458
Test Accuracy: 100.00%
Epoch 17, Training Loss: 0.2544
Test Accuracy: 100.00%
Epoch 18, Training Loss: 0.2249
Test Accuracy: 90.00%
Epoch 19, Training Loss: 0.2782
Test Accuracy: 100.00%
Epoch 20, Training Loss: 0.3049
Test Accuracy: 80.00%
Epoch 21, Training Loss: 0.2562
Test Accuracy: 100.00%
Epoch 22, Training Loss: 0.2606
Test Accuracy: 100.00%
Epoch 23, Training Loss: 0.1896
Test Accuracy: 100.00%
Epoch 24, Training Loss: 0.1212
Test Accuracy: 100.00%
Epoch 25, Training Loss: 0.1318
```

```
Test Accuracy: 100.00%

Training Complete.
```

Loss 曲线与准确率曲线：



5.2 评测指标展示及分析

根据训练结果。`loss` 值从初始的 1.4421 逐渐降低，最终稳定在一个较低的水平，这表明模型在训练过程中逐渐学习到了数据的特征和模式，成功地拟合了训练数据。在测试集上，模型表现出了较高的准确率，从第 11 个 Epoch 开始，测试准确率就达到了 100%。这意味着模型能够在未见过的数据上进行准确的分类预测，说明模型在这个分类任务上具有很强的泛化能力。但测试准确率在一些 Epochs 上出现了波动，这可能是由于训练数据集和测试数据集之间的差异，或者模型在某些时候过于拟合训练数据而导致的。

六、参考资料

1. https://blog.csdn.net/weixin_43312117/article/details/118852828?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217162698111680021508 4930%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=171626981116800215084930&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-5-118852828-null-null.142^v100^pc_search_result_base8&utm_term=pytorch%E5%AE%89%E8%A3%85%E6%95%99%E7%A8%8Bgpu&spm=1018.2226.3001.4187
2. https://blog.csdn.net/weixin_73044854/article/details/137427320
3. https://blog.csdn.net/IronmanJay/article/details/128689946?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217168877681680022657 9388%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=171688776816800226579388&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_click~default-1-128689946-null-null.142^v100^pc_search_result_base8&utm_term=cnn%E5%8D%B7%E7%A7%AF&spm=1018.2226.3001.4187