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	23336179	姓名	马福泉

一、 实验题目

深度学习

中药图片分类任务

利用pytorch框架搭建神经网络实现中药图片分类，其中中药图片数据分为训练集 `train` 和测试集 `test` ，训练集仅用于网络训练阶段，测试集仅用于模型的性能测试阶段。训练集和测试集均包含五种不同类型的中药图片： `baihe` 、 `dangshen` 、 `gouqi` 、 `huaihua` 、 `jinyinhua` 。请合理设计神经网络架构，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。

提示

1. 最后提交的代码只需包含性能最好的实现方法和参数设置．只需提交一个代码文件，请不要提交其他文件．
2. 本次作业可以使用 `pytorch` 库、 `numpy` 库、 `matplotlib` 库以及python标准库．
3. 数据集可以在Github上下载。
4. 模型的训练性能以及测试性能均作为本次作业的评分标准。
5. 测试集不可用于模型训练。
6. 不能使用开源的预训练模型进行训练。

二、 实验内容

（一）算法原理

1. 数据预处理

数据预处理流水线（Data_transform）

(1) 目标：将输入图像统一处理为适合模型输入的格式。

(2) 操作：

- 调整图像尺寸：使用 transforms.Resize((224,224)) 将所有图像调整为 224×224 像素，确保输入图像尺寸一致。
- 转换为张量：通过 transforms.ToTensor() 将图像从 PIL 格式转换为 PyTorch 张量，并将像素值归一化到 [0,1] 区间。
- 标准化：使用 transforms.Normalize 对图像进行标准化处理，均值和标准差分别设为 [0.485, 0.456, 0.406] 和 [0.229, 0.224, 0.225]。（这些参数是基于 ImageNet 数据集的统计值，有助于加速模型收敛。）

2. 数据加载与管理

自定义数据集类（MyDataset）

(1) 目标：从索引文件（如 train.txt 和 test.txt）中读取图像路径和标签，并加载图像数据。

(2) 实现：

- 读取索引文件：从指定的 .txt 文件中逐行读取图像路径和标签（格式为 image_path:label）。
  比如：train/baihe/b (108).jpg:0
  表示：路径是 train/baihe/b (108).jpg，编号是 0（表示第 0 类药材百合）
- 存储图像路径和标签：将图像路径和标签分别存储在 self.imgs 和 self.labels 列表中。
- __getitem__ 方法：根据索引加载指定图像，应用预处理操作（Data_transform），并返回图像及其标签。
- __len__ 方法：返回数据集中的样本总数。

3. 模型定义

卷积神经网络（CNN）：定义一个用于图像分类的 CNN 模型。

(1) 卷积层：

- 第一层卷积：输入通道数为 3（彩色图像），输出通道数为 64，卷积核大小为 3×3，步长为 1，填充为 1。之后接批量归一化（BatchNorm2d）和 ReLU 激活函数，最后通过 2×2 的最大池化层。
- 第二层卷积：输入通道数为 64，输出通道数为 128，卷积核大小为 3×3，步长为 1，填充为 1。结构与第一层类似，但池化层大小为 2×2。
- 第三层卷积：输入通道数为 128，输出通道数为 256，卷积核大小为 3×3，步长为 1，填充为 1。池化层大小为 4×4。

(2) 全连接层：输入特征维度为 256×14×14（由卷积层输出维度决定），经过三个全连接层，每层之间使用 Dropout（概率为 0.5）防止过拟合，最终输出类别数为 5。

(3) 前向传播：定义数据在模型中的流动路径，依次通过卷积层、池化层、全连接层，最终输出分类结果。

4. 模型训练

训练函数（TrainNetwork）：对模型进行多轮（epoch）训练，并在每个 epoch 结束时评估模型在测试集上的性能。

(1) 初始化指标记录：用于存储训练和测试过程中的损失值和准确率。

(2) 多轮迭代训练：

- 每个 epoch 开始时，模型切换到训练模式（model.train()）。
- 遍历训练数据加载器（train_loader），对每个批次的数据进行以下操作：
  将图像和标签移动到指定设备（CPU 或 GPU）。
  输入图像通过模型得到预测输出。
  计算损失值（使用交叉熵损失函数）。
  清空梯度，反向传播计算梯度，更新模型参数。
- 计算该 epoch 的平均训练损失。
- 更新学习率（使用学习率调度器）。

(3) 评估模型性能：

每个 epoch 结束时，模型切换到评估模式（model.eval()）。

在训练集和测试集上分别计算准确率和平均损失。

输出训练指标：打印每个 epoch 的训练和测试损失、准确率。
返回指标记录：返回包含训练和测试过程中的各项指标的字典。

5. 结果可视化

可视化函数（save_visualization）:将训练过程中的损失值和准确率可视化，并保存为图像文件。

(1) 创建保存目录：在指定路径下创建保存结果的目录。
(2) 绘制曲线：绘制批次训练损失曲线。
绘制每个 epoch 的训练和测试损失曲线。
绘制训练和测试准确率曲线。
(3) 创建表格：展示每个 epoch 的训练和测试损失、准确率。
(4) 保存图像：将可视化结果保存为 PNG 文件。

6. 准确率计算

准确率计算函数（Accurary）：计算模型在训练集和测试集上的准确率。

(1) 模型切换到评估模式：禁用批量归一化和 Dropout。
(2) 遍历数据加载器：分别在训练集和测试集上计算模型的预测准确率。
(3) 输出准确率：打印训练集和测试集的准确率。

7. 主程序

主程序逻辑（if __name__ == '__main__':）：整合上述模块，完成模型训练、评估和结果可视化。

(1) 设置路径：定义训练和测试索引文件路径以及图像数据目录。
(2) 定义训练参数：设置批量大小、学习率、训练轮数等。
(3) 加载数据集：创建训练和测试数据集实例，并通过 DataLoader 构建数据加载器。
(4) 初始化模型和训练工具：

- 初始化 CNN 模型。
- 定义损失函数（交叉熵）。
- 配置优化器（Adam）。
- 设置学习率调度器。

(5) 训练模型：调用 TrainNetwork 函数进行模型训练。
(6) 保存可视化结果：调用 save_visualization 函数生成训练过程的可视化图像。
(7) 计算准确率：调用 Accurary 函数计算模型在训练集和测试集上的准确率。

（二）关键代码展示

1. 数据预处理

```python
# 图像预处理流水线：对输入图像进行变换操作
Data_transform=transforms.Compose([
    # 将所有图像调整为统一尺寸224x224像素
    transforms.Resize((224,224)),
    # 将图像转换为PyTorch张量格式，并归一化像素值到[0,1]区间
    transforms.ToTensor(),
    # 标准化处理：使用ImageNet数据集的均值和标准差进行归一化
    transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225])
])
```

2. 自定义数据集类

```python
# 图像数据集类：用于管理图像数据的加载和预处理
class MyDataset(Dataset):
    def __init__(self, txt_path, data_dir=""):
        imgs=[]
        labels=[]
        # 从索引文件读取图像数据信息
        Read=open(txt_path,'r')
        for i in Read:
            i=i.rstrip()
            img=i.split(':')
            img_path = os.path.join(data_dir, img[0]) if data_dir else img[0]
            imgs.append(img_path)
            labels.append(int(img[1]))
        self.imgs=imgs
        self.labels=labels

    # 返回数据集中样本总数
    def __len__(self):
        return len(self.imgs)

    # 根据索引获取数据样本及其对应标签
    def __getitem__(self, index):
        # 使用PIL库加载指定索引的图像文件
        img=Image.open(self.imgs[index])
        # 应用预定义的数据转换操作（缩放、转换张量、归一化）
        img=Data_transform(img)
        # 获取对应的分类标签
        label=self.labels[index]
        return img,label
```

3. CNN 模型定义

```python
class CNN(nn.Module):
    def __init__(self):
        # 初始化父类属性
        super(CNN,self).__init__()
        # 定义卷积层
        # 第一层卷积
        self.conv1=nn.Sequential(
            # 卷积操作：提取特征
            nn.Conv2d(
                in_channels=3, # 输入通道数（灰度图为1，彩色图为3）
                out_channels=64, # 卷积核数量，决定输出通道数
                kernel_size=3, # 卷积核大小3x3
                stride=1, # 步长为1
                padding=1, # 填充以保持输出尺寸与输入相同
            ),
            # 批量归一化：加速训练并稳定模型
            nn.BatchNorm2d(num_features=64),
            # 激活函数：ReLU
            nn.ReLU(inplace=True),
            # 最大池化：降维并保留重要特征
            nn.MaxPool2d(
                kernel_size=2, # 池化核大小2x2
                stride=2, # 步长为2
            ),
        )
```

```
        # 第二层卷积
        self.conv2=nn.Sequential(
            nn.Conv2d(64,128,3,1,1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2,2),
        )

        # 第三层卷积
        self.conv3=nn.Sequential(
            nn.Conv2d(128,256,3,1,1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(4,4),
        )

        # 全连接层（输出层）
        self.output=nn.Sequential(
            # Dropout：以0.5的概率随机丢弃神经元，防止过拟合
            nn.Dropout(0.5),
            nn.Linear(256*14*14,256),
            nn.BatchNorm1d(256),
            nn.ReLU(inplace=True),

            nn.Dropout(0.5),
            nn.Linear(256,256),
            nn.BatchNorm1d(256),
            nn.ReLU(inplace=True),

            nn.Dropout(0.5),
            # 最终输出类别数为5
            nn.Linear(256,5),
        )

    # 前向传播：定义数据流经网络的方式
    def forward(self,x):
        # 卷积层提取特征
        x=self
```

4. 模型训练函数

输入图像 (3x224x224)
↓
卷积层 1 (Conv2d: 3 -> 64, 3x3, stride=1, padding=1)
↓
批量归一化 (BatchNorm2d: 64)
↓
ReLU 激活函数
↓
最大池化 (MaxPool2d: 2x2, stride=2)
↓
卷积层 2 (Conv2d: 64 -> 128, 3x3, stride=1, padding=1)
↓
批量归一化 (BatchNorm2d: 128)
↓

ReLU 激活函数
↓
最大池化 (MaxPool2d: 2x2, stride=2)
↓
卷积层 3 (Conv2d: 128 -> 256, 3x3, stride=1, padding=1)
↓
批量归一化 (BatchNorm2d: 256)
↓
ReLU 激活函数
↓
最大池化 (MaxPool2d: 4x4, stride=4)
↓
展平 (Flatten)
↓
全连接层 1 (Linear: 256x14x14 -> 256)
↓
批量归一化 (BatchNorm1d: 256)
↓
ReLU 激活函数
↓
Dropout (0.5)
↓
全连接层 2 (Linear: 256 -> 256)
↓
批量归一化 (BatchNorm1d: 256)
↓
ReLU 激活函数
↓
Dropout (0.5)
↓
全连接层 3 (Linear: 256 -> 5)
↓
输出 (5 个类别)

（三）创新点&优化

1. 模块化设计

在代码中，CNN 模型被清晰地分解为多个模块，每个模块负责特定的功能：

- 卷积层 (conv1, conv2, conv3)：这些模块通过堆叠多个卷积层来提取图像的特征。每个卷积层后接批量归一化和 ReLU 激活函数，形成一个完整的特征提取模块。
- 池化层：在每个卷积模块后，使用最大池化层来降低特征图的维度，减少参数数量和计算量，同时增加模型对小位移的鲁棒性。
- 全连接层 (output)：在卷积和池化层之后，使用全连接层将提取的特征映射到输出空间，进行最终的分类。

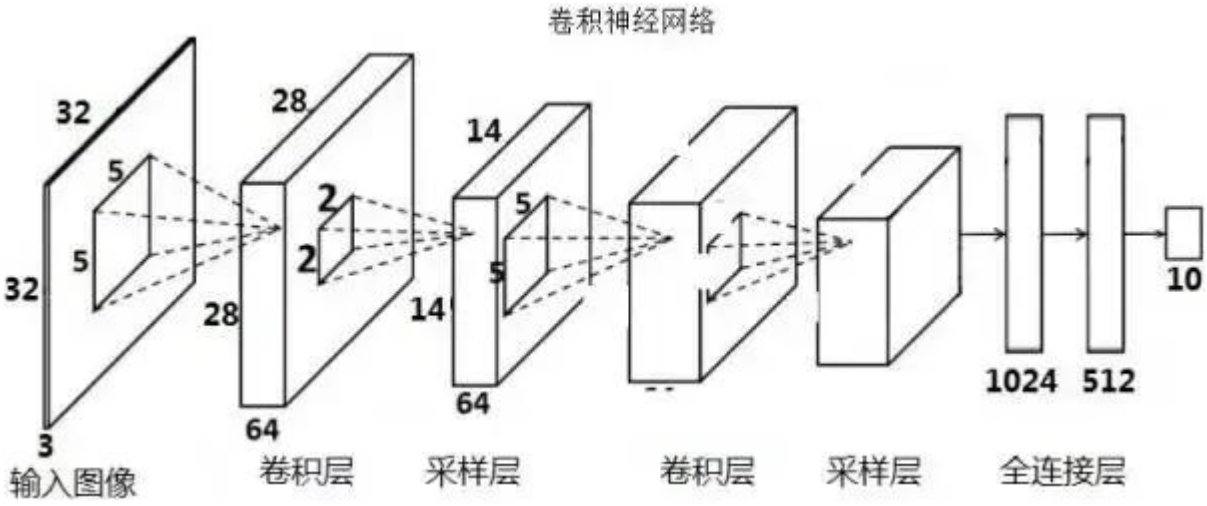
这种模块化设计使得模型结构清晰，易于理解和修改。每个模块可以独立开发和测试，有助于快速迭代和优化。

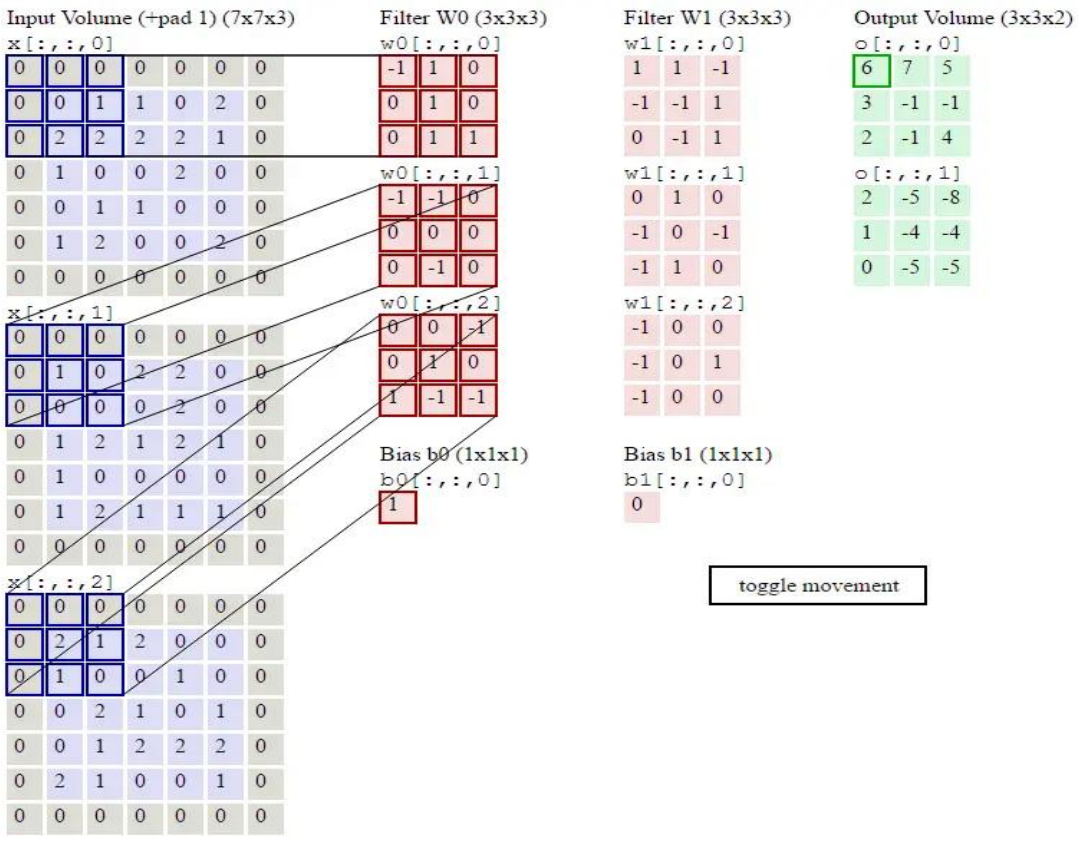
2. 多层卷积结构

代码中的 CNN 模型通过堆叠三个卷积层来学习图像的多层次特征：

- 第一层卷积（conv1）：提取基本的边缘和纹理特征。
- 第二层卷积（conv2）：在第一层的基础上，提取更复杂的形状特征。
- 第三层卷积（conv3）：进一步提取高级的特征，如对象的部分。

这种层次化的特征学习是深度学习在图像识别中取得成功的关键因素。通过逐层提取更复杂的特征，模型能够更好地理解和分类图像。





3. 归一化和标准化

在数据预处理阶段，代码中对图像进行归一化和标准化处理：

- 归一化：将图像转换为 PyTorch 张量，并归一化像素值到 [0,1] 区间。
- 标准化：使用 ImageNet 数据集的均值和标准差对图像进行标准化处理。
- 这些预处理步骤有助于模型更快地收敛，并提高模型的泛化能力。通过标准化处理，可以加速模型的训练过程，并提高模型在新数据上的表现。

4. 池化层的使用

- 在每个卷积模块后，代码中使用最大池化层来降低特征图的维度：

第一层池化：在 conv1 后，使用 2x2 的最大池化层。
第二层池化：在 conv2 后，使用 2x2 的最大池化层。
第三层池化：在 conv3 后，使用 4x4 的最大池化层。

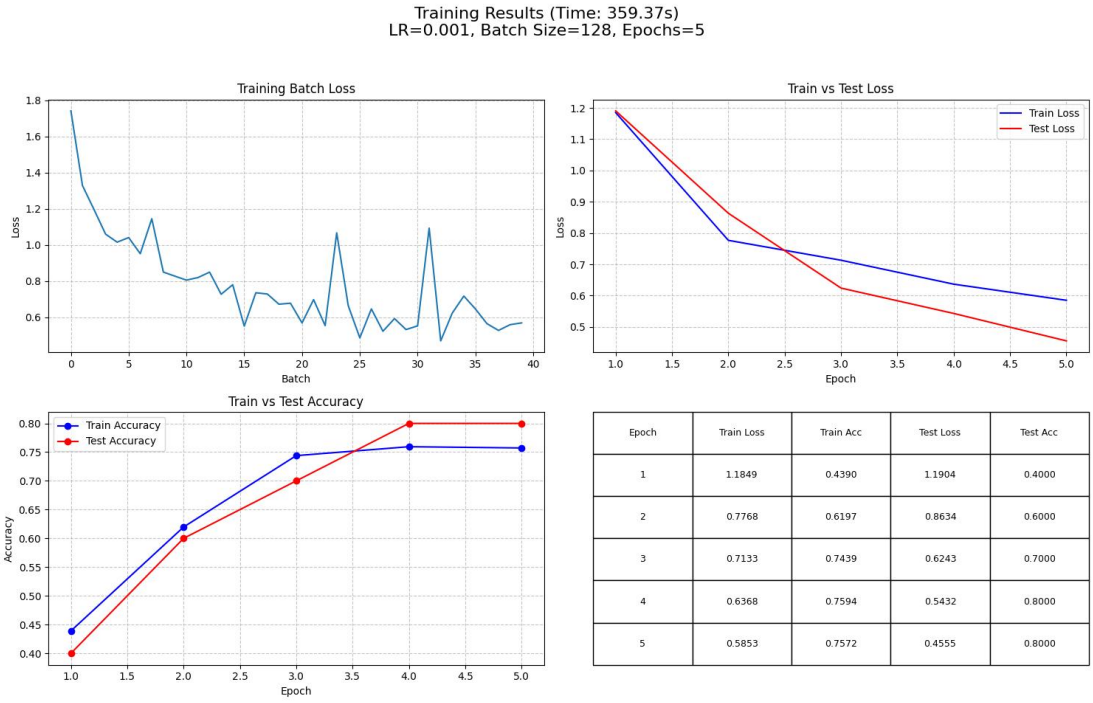
- 这些池化层有助于减少参数数量和计算量，同时也能够提供一定程度的平移不变性。这使得模型在面对图像中的小的位移或变形时更加鲁棒。

5. 动态学习率调整

在代码中，使用学习率调度器（StepLR）来动态调整学习率：
学习率调度器：在每个 epoch 结束后，根据预设的步长和衰减率调整学习率。
这种动态学习率调整策略可以在训练初期使用较大的学习率快速收敛，在训练后期使用较小的学习率进行精细调整，从而提高模型性能。

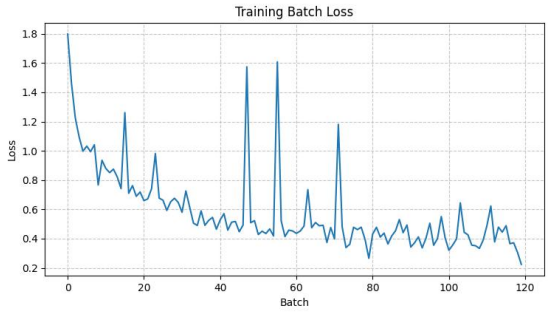
三、 实验结果及分析

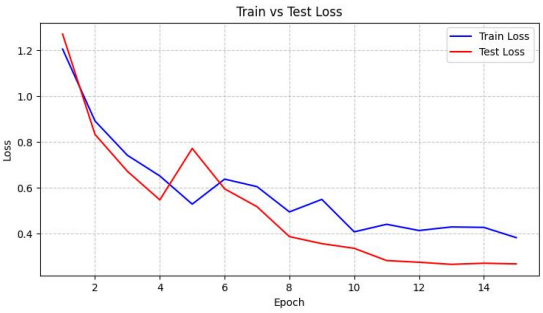
1. 实验结果展示示例

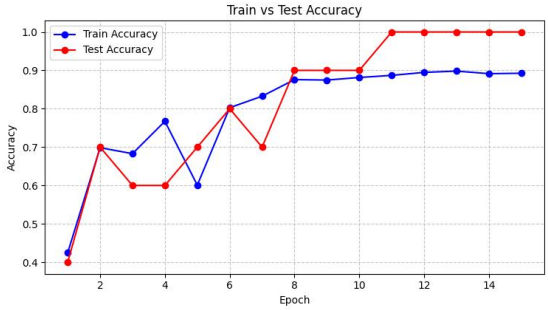


Epoch	Train Loss	Train Acc	Test Loss	Test Acc
1	1.1849	0.4390	1.1904	0.4000
2	0.7768	0.6197	0.8634	0.6000
3	0.7133	0.7439	0.6243	0.7000
4	0.6368	0.7594	0.5432	0.8000
5	0.5853	0.7572	0.4555	0.8000

Training Results (Time: 1071.67s)
LR=0.001, Batch Size=128, Epochs=15

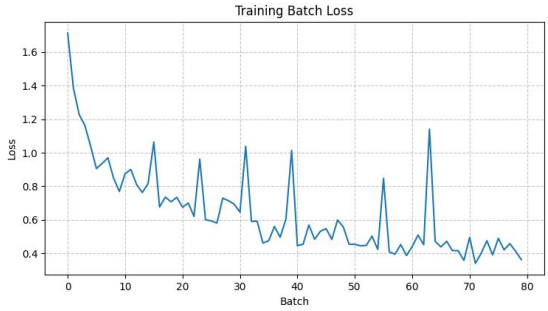


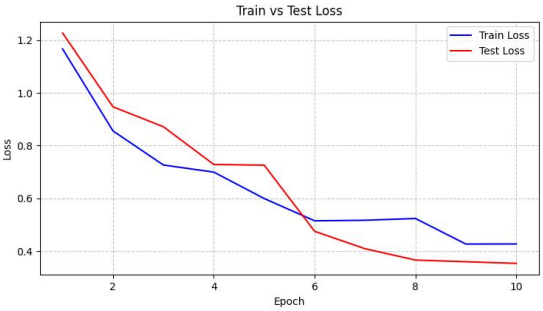


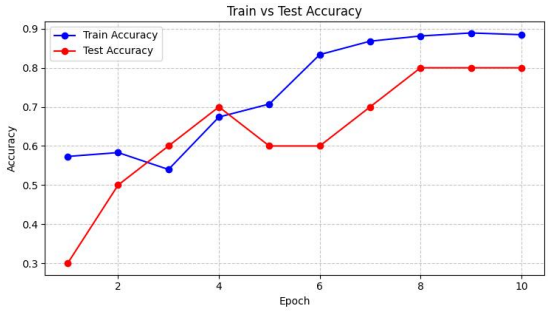


Epoch	Train Loss	Train Acc	Test Loss	Test Acc
1	1.2057	0.4246	1.2714	0.4000
2	0.8918	0.6984	0.8331	0.7000
3	0.7418	0.6829	0.6719	0.6000
4	0.6517	0.7672	0.5471	0.6000
5	0.5284	0.6009	0.7717	0.7000
6	0.6377	0.8027	0.5954	0.8000
7	0.6049	0.8326	0.5173	0.7000
8	0.4946	0.8758	0.3866	0.9000
9	0.5492	0.8747	0.3559	0.9000
10	0.4074	0.8814	0.3355	0.9000
11	0.4405	0.8869	0.2820	1.0000
12	0.4133	0.8947	0.2746	1.0000
13	0.4289	0.8980	0.2654	1.0000
14	0.4269	0.8914	0.2704	1.0000
15	0.3822	0.8925	0.2675	1.0000

Training Results (Time: 726.26s)
LR=0.001, Batch Size=128, Epochs=10







Epoch	Train Loss	Train Acc	Test Loss	Test Acc
1	1.1670	0.5732	1.2272	0.3000
2	0.8558	0.5831	0.9475	0.5000
3	0.7266	0.5399	0.8717	0.6000
4	0.6996	0.6741	0.7286	0.7000
5	0.5994	0.7073	0.7260	0.6000
6	0.5149	0.8337	0.4750	0.6000
7	0.5170	0.8681	0.4089	0.7000
8	0.5237	0.8814	0.3660	0.8000
9	0.4267	0.8891	0.3596	0.8000
10	0.4270	0.8847	0.3534	0.8000

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）

(1) 第一组结果（Epochs=5）

- 训练批次损失：损失值在初期迅速下降，但波动较大，表明模型在早期训练中不稳定。
- 训练与测试损失：训练损失持续下降，而测试损失在初期下降后趋于平稳，两者之间的差距不大，表明模型没有明显的过拟合。
- 训练与测试准确率：训练准确率快速上升并趋于稳定，测试准确率也有所提

升，但略低于训练准确率，显示出一定的泛化能力。

- 性能表格：训练损失从 1.1849 降至 0.5853，测试损失从 1.1904 降至 0.4555，训练准确率从 43.90%提升至 75.72%，测试准确率从 40.00%提升至 80.00%。

(2) 第二组结果（Epochs=10）

- 训练批次损失：损失值整体呈下降趋势，但波动依然存在，尤其是在训练的后期。
- 训练与测试损失：训练损失持续下降，测试损失在初期下降后出现波动，但总体趋势是下降的，显示出模型在更多周期下的学习效果。
- 训练与测试准确率：训练准确率持续提升，测试准确率也有显著提升，两者之间的差距较小，表明模型具有良好的泛化能力。
- 性能表格：训练损失从 1.1670 降至 0.4270，测试损失从 1.2272 降至 0.3354，训练准确率从 57.32%提升至 88.47%，测试准确率从 30.00%提升至 80.00%。

(3) 第三组结果（Epochs=15）

- 训练批次损失：损失值整体呈下降趋势，波动较前两组结果有所减少，显示出模型在更多训练周期下的稳定性提升。
- 训练与测试损失：训练损失持续下降，测试损失在初期下降后趋于平稳，两者之间的差距较小，表明模型没有明显的过拟合。
- 训练与测试准确率：训练准确率持续提升并趋于稳定，测试准确率也有显著提升，两者之间的差距较小，显示出模型的泛化能力。
- 性能表格：训练损失从 1.2057 降至 0.3822，测试损失从 1.2714 降至 0.2675，训练准确率从 42.46%提升至 89.25%，测试准确率从 40.00%提升至 100.00%，且可以较为稳定维持在 100%。

(4) 综合考虑，选择 10 个训练周期（Epochs）。

理由可以从以下几个方面进行阐述：

- 平衡性能与过拟合：在 10 个周期时，模型已经能够取得较好的性能，同时避免了过多的训练周期可能导致的过拟合风险。从结果中可以看到，10 个周期的训练准确率和测试准确率都相对较高，且两者之间的差距较小，表明模型具有良好的泛化能力。
- 计算资源与时间效率：相比于 15 个周期，10 个周期的训练时间更短，对计算资源的需求也相对较低。这使得在有限的计算资源下，可以更快地完成模型训练和评估，提高研究和开发的效率。
- 避免冗余训练：从训练批次损失图中可以看出，随着训练周期的增加，损失值的下降速度逐渐减缓，表明模型的学习速度在减慢。在 10 个周期时，模型已经能够较好地拟合训练数据，继续增加周期可能带来的性能提升有限，而更多的训练周期则可能导致模型对训练数据的过度拟合。
- 交叉验证的便利性：在进行交叉验证时，10 个周期可以提供一个合理的训练周期范围，使得模型在不同的数据划分下都能得到充分的训练，同时又不会因为周期过长而导致过拟合。
- 模型性能的稳定性：从结果中可以看出，10 个周期的训练结果在测试集上的性能较为稳定，测试准确率和测试损失的变化趋势表明模型在这一周期下已经达到了较好的性能水平。

四、 参考资料

1. 部分代码改进建议，debug 与注释添加等除了参考实验指导书，还参考了大语言模型

2. 参考文章，博客部分如下：

PyTorch 入门实战（四）——利用 Torch.nn 构建卷积神经网络_import torch.nn as nn-CSDN 博客
Pytorch 入门（二）神经网络的搭建_神经网络搭建-CSDN 博客
Pytorch 实现中药材(中草药)分类识别(含训练代码和数据集) 中草药数据集-CSDN 博客
CNN 入门讲解：卷积层是如何提取特征的？ - 知乎
深度学习-CNN 提取图像特征_cnn 特征提取-CSDN 博客