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	23336266	姓名	熊彦钧

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

1. 算法原理

（1）卷积神经网络（CNN）的基本原理

CNN 的核心思想是通过局部感受野和权值共享来提取图像的层次化特征。

局部感受野：每个卷积核仅关注输入图像的局部区域（如 3×3 或 5×5 的窗口），而不是全连接层的全局输入，这使得 CNN 能够高效捕捉局部特征（如边缘、纹理）。

权值共享：同一个卷积核在整张图像上滑动计算，大幅减少参数量，同时增强平移不变性（即无论目标在图像中的位置如何变化，CNN 都能识别）。

本实验的 CNN 架构采用 5 层卷积块，每层包含：

①Conv2d（卷积层）：提取局部特征，如 nn.Conv2d(3, 32, kernel_size=3, padding=1) 表示输入通道 3（RGB），输出通道 32，3×3 卷积核，padding=1 保持空间尺寸不变。

②BatchNorm2d（批归一化）：加速训练并稳定梯度，防止梯度消失/爆炸。

③ReLU（激活函数）：引入非线性，增强模型表达能力。

④MaxPool2d（最大池化）：降采样（如 2×2 池化使特征图尺寸减半），减少计算量并增强特征不变性。

（2）特征提取与分类

CNN 的深层结构能够自动学习从低级到高级的特征：①浅层（前几层）：检测边缘、颜色、纹理等低级特征。②中层：检测局部形状（如花瓣、叶片轮廓）。③深层（后几层）：组合低级特征，形成高级语义（如整朵花的形状）。

本实验在卷积层后采用 AdaptiveAvgPool2d（全局平均池化），将特征图压缩为 1×1，再送入全连接层分类。相比传统 CNN 的 Flatten + Dense 结构，全局池化能减少参数，防止过拟合。

（3）训练优化策略

①损失函数：使用 CrossEntropyLoss（交叉熵损失），适用于多分类任务。

②优化器：采用 Adam（自适应学习率优化器），相比 SGD 能自动调整学习率，加速收敛。

③学习率调度：StepLR 每 step_size 个 epoch 衰减学习率（gamma=0.1），避免后期震荡。

④正则化：Dropout（随机丢弃神经元，如 Dropout(0.3)）防止过拟合；BatchNorm 稳定训练，减少对初始化的敏感性。

（4）数据增强与预处理

①Resize(224, 224)：统一输入尺寸，适应 CNN 的固定输入要求。

②Normalize：使用 ImageNet 的均值和标准差（mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]）标准化数据，加速收敛。

2. 关键代码展示

（1）数据预处理和加载部分代码如下：

```python
# ================= 数据加载 =================
class SmartDataset(Dataset):
    """智能数据集类：支持文件夹和文件名两种组织方式"""
    def __init__(self, root_dir, transform=None, mode='train'):
        """
        Args:
            root_dir: 数据根目录
            transform: 数据预处理
            mode: 'train'（按文件夹分类）或 'test'（按文件名分类）
        """
        self.root_dir = root_dir
        self.transform = transform
        self.mode = mode
        self.samples = self._load_samples()

    def _load_samples(self):
        samples = []
        if self.mode == 'train':
            # 训练模式：按文件夹分类
            for class_idx, class_name in enumerate(Config.class_names):
                class_dir = os.path.join(self.root_dir, class_name)
                if os.path.isdir(class_dir):
                    for img_name in os.listdir(class_dir):
                        if img_name.lower().endswith(('.jpg', '.png', '.jpeg')):
                            samples.append((
                                os.path.join(class_dir, img_name),
                                class_idx
                            ))
        else:
            # 测试模式：按文件名分类
            for img_name in os.listdir(self.root_dir):
                if img_name.lower().endswith(('.jpg', '.png', '.jpeg')):
                    img_path = os.path.join(self.root_dir, img_name)
                    for class_idx, class_name in enumerate(Config.class_names):
                        if class_name.lower() in img_name.lower():
                            samples.append((img_path, class_idx))
                            break
        return samples

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        img_path, label = self.samples[idx]
        img = Image.open(img_path).convert('RGB')
        if self.transform:
            img = self.transform(img)
        return img, label
```

代码说明：

（1）支持两种数据组织方式（文件夹分类 / 文件名分类），增强代码灵活性。

（2）使用 PIL.Image 加载图片并统一转换为 RGB 格式，避免通道数不一致问题

（2）模型架构代码如下：

```python
# ================= 模型架构 =================
class EnhancedCNN(nn.Module):
    """增强版CNN：结合深度特征提取与高效分类器"""
    def __init__(self, num_classes):
        super().__init__()
        # 特征提取器（5层卷积）
        self.features = nn.Sequential(
            # 卷积块1 (224x224 -> 112x112)
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # 卷积块2 (112x112 -> 56x56)
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # 卷积块3 (56x56 -> 28x28)
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # 卷积块4 (28x28 -> 14x14)
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # 卷积块5 (14x14 -> 7x7)
            nn.Conv2d(256, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
```

```python
        # 自适应池化 (7x7 -> 1x1)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))

        # 分类器（带Dropout）
        self.classifier = nn.Sequential(
            nn.Dropout(0.3),  # 适度正则化
            nn.Linear(512, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(0.3),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

说明：

（1）深度卷积结构：5 层卷积逐步提取低级→高级特征，适用于细粒度分类。

（2）全局平均池化：减少参数，防止过拟合（相比传统 Flatten + Dense）。

（3）Dropout：在训练时随机关闭神经元，提升泛化能力。

（3）训练逻辑部分代码如下：

```python
    def train_epoch(self, epoch):
        """执行一个epoch的训练，并记录批次级别的损失"""
        self.model.train()
        running_loss = 0.0
        correct = 0
        total = 0
        epoch_batch_losses = []  # 记录当前epoch的每个batch的loss

        for batch_idx, (inputs, labels) in enumerate(self.train_loader):
            inputs = inputs.to(self.config.device)
            labels = labels.to(self.config.device)

            # 前向传播
            self.optimizer.zero_grad()
            outputs = self.model(inputs)
            loss = self.criterion(outputs, labels)

            # 反向传播
            loss.backward()
            self.optimizer.step()

            # 记录当前batch的loss
            batch_loss = loss.item()
            epoch_batch_losses.append(batch_loss)
```

```python
            # 为整体批次损失列表添加当前batch loss和epoch信息
            self.batch_losses.append({
                'epoch': epoch + 1,
                'batch': batch_idx + 1,
                'loss': batch_loss
            })

            # 统计指标
            running_loss += batch_loss
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

        # 计算epoch指标
        epoch_loss = running_loss / len(self.train_loader)
        epoch_acc = correct / total
        return epoch_loss, epoch_acc, epoch_batch_losses
```

说明：

（1）zero_grad()：防止梯度累加，确保每次迭代独立计算。

（2）loss.backward() + optimizer.step()：核心训练步骤，计算梯度并更新参数。

（3）Batch 统计：累计损失和准确率，用于监控训练过程。

（4）测试评估部分代码如下：

```python
def evaluate(self):
    """评估模型性能"""
    self.model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for inputs, labels in self.test_loader:
            inputs = inputs.to(self.config.device)
            labels = labels.to(self.config.device)

            outputs = self.model(inputs)
            loss = self.criterion(outputs, labels)

            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

    test_loss = running_loss / len(self.test_loader)
    test_acc = correct / total
    return test_loss, test_acc
```

说明：

（1）torch.no_grad()：关闭自动求导，节省内存并加速计算。

（2）测试模式：不更新模型参数，仅计算损失和准确率。

以上便是数据加载→模型构建→训练优化→评估分析的全流程，至于图片生成的部分，因为不涉及算法，我们就在此省略了，完整代码请见附件。

3. 创新点&优化

（1）深度 CNN 架构优化

①5 层卷积+全局池化：相比传统 CNN（如 LeNet-5 的 2 层卷积），更深的网络能提取更复杂的特征，适用于细粒度分类（如不同中药的细微差异）。

②批归一化（BatchNorm）：加速训练，减少对超参数（如学习率）的依赖。

③Dropout 正则化：防止过拟合，提升泛化能力。

（2）动态学习率调整（StepLR）

初始学习率 lr=0.001，每 step_size=5 个 epoch 衰减 gamma=0.1，避免后期震荡。

（3）智能数据加载（SmartDataset）

支持按文件夹分类（train）和按文件名分类（test），增强代码通用性。

（4）全面的可视化分析

训练曲线（Loss & Accuracy）监控模型收敛情况；混淆矩阵分析各类别的分类性能，找出易混淆的中药类别；批次级损失跟踪观察训练稳定性，辅助调参。

（5）计算效率优化

DataLoader 多线程加载（num_workers=2）加速数据读取；GPU 加速减少 CPU→GPU 数

据传输时间。

三、 实验结果及分析

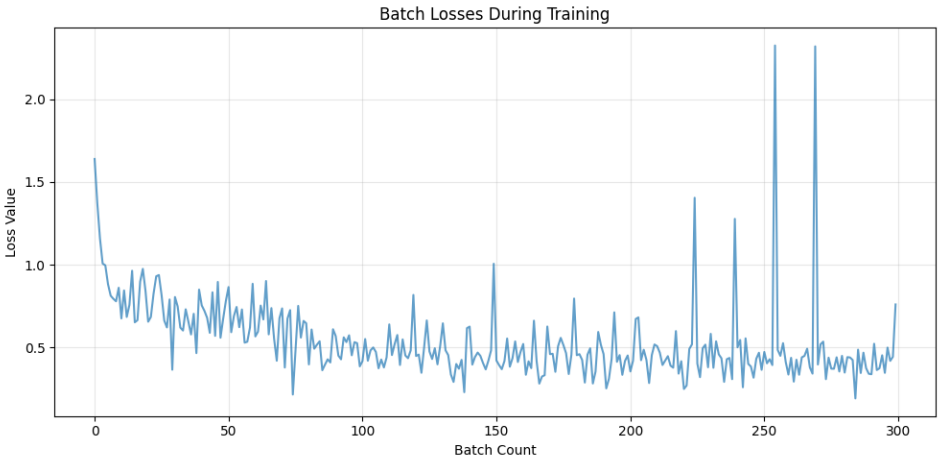
1. 实验结果展示示例

由于实验具有随机性，这里只展示其中一次运行结果。

程序运行时终端部分的输出如下，这里我们可以看到每一个训练批次的训练损失、训练准确率、测试损失、测试准确率以及运行时间。

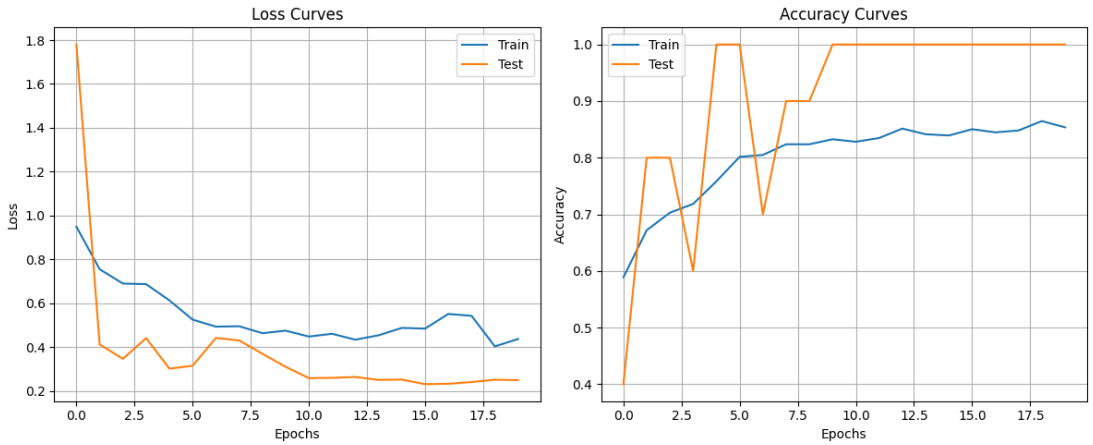
```
Train Loss: 0.5252 | Train Acc: 0.8016 | Test Loss: 0.3146 | Test Acc: 1.0000 | 时间: 15.2s

Epoch 7/20:
Train Loss: 0.4931 | Train Acc: 0.8049 | Test Loss: 0.4415 | Test Acc: 0.7000 | 时间: 14.9s

Epoch 8/20:
Train Loss: 0.4950 | Train Acc: 0.8237 | Test Loss: 0.4303 | Test Acc: 0.9000 | 时间: 14.7s

Epoch 9/20:
Train Loss: 0.4632 | Train Acc: 0.8237 | Test Loss: 0.3695 | Test Acc: 0.9000 | 时间: 14.6s

Epoch 10/20:
Train Loss: 0.4748 | Train Acc: 0.8326 | Test Loss: 0.3100 | Test Acc: 1.0000 | 时间: 14.8s

Epoch 11/20:
Train Loss: 0.4480 | Train Acc: 0.8282 | Test Loss: 0.2585 | Test Acc: 1.0000 | 时间: 14.9s

Epoch 12/20:
Train Loss: 0.4606 | Train Acc: 0.8348 | Test Loss: 0.2595 | Test Acc: 1.0000 | 时间: 15.0s

Epoch 13/20:
Train Loss: 0.4336 | Train Acc: 0.8514 | Test Loss: 0.2636 | Test Acc: 1.0000 | 时间: 15.1s

Epoch 14/20:
Train Loss: 0.4537 | Train Acc: 0.8415 | Test Loss: 0.2501 | Test Acc: 1.0000 | 时间: 15.0s

Epoch 15/20:
Train Loss: 0.4874 | Train Acc: 0.8392 | Test Loss: 0.2518 | Test Acc: 1.0000 | 时间: 15.0s

Epoch 16/20:
Train Loss: 0.4842 | Train Acc: 0.8503 | Test Loss: 0.2308 | Test Acc: 1.0000 | 时间: 15.5s

Epoch 17/20:
Train Loss: 0.5510 | Train Acc: 0.8448 | Test Loss: 0.2326 | Test Acc: 1.0000 | 时间: 15.4s

Epoch 18/20:
Train Loss: 0.5425 | Train Acc: 0.8481 | Test Loss: 0.2405 | Test Acc: 1.0000 | 时间: 15.0s

Epoch 19/20:
Train Loss: 0.4031 | Train Acc: 0.8647 | Test Loss: 0.2511 | Test Acc: 1.0000 | 时间: 15.0s

Epoch 20/20:
Train Loss: 0.4368 | Train Acc: 0.8537 | Test Loss: 0.2490 | Test Acc: 1.0000 | 时间: 14.8s

训练完成，总用时 298.72 秒
最佳测试准确率: 1.0000 (Epoch 5)
正在生成可视化结果...
可视化结果已保存到: E:/大学课件/人工智能作业/7 深度学习/mine\results
```

下面是一个线形图，记录了每一个 batch 的损失（程序一共运行了 20 个批次，每个批次有 15 个 batch）：



我们可以看到，除了偶尔会出现尖峰，每个 batch 的损失逐渐减小并趋于稳定。

下面是损失曲线和准确率曲线图：



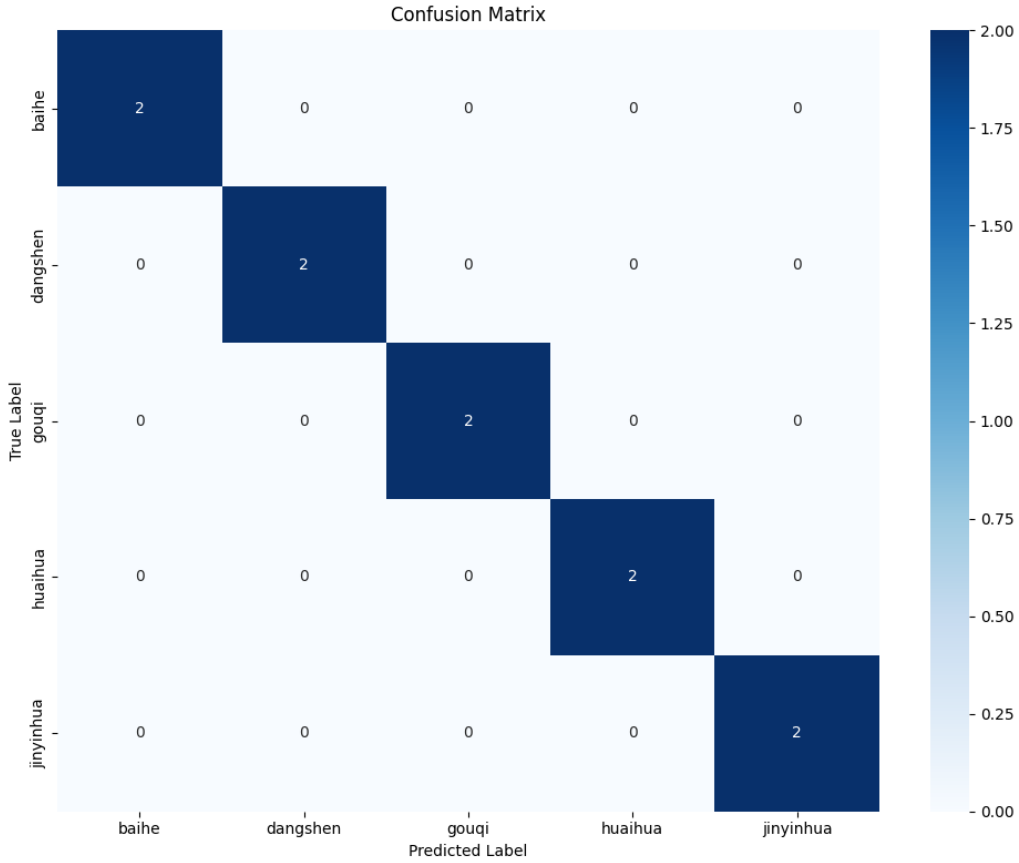
我们可以看到，每个批次的损失整体呈下降趋势，最后稳定在 0.4-0.5 之间；而训练准确率逐渐上升并稳定在 0.8-0.9 之间；而测试的准确率最后基本可以稳定在 1。（经过大量运行我们发现，大部分的情况下准确率是可以稳定在 1 的，有少部分的情况会在 1 和 0.9 之间反复跳跃）；

下面是训练过程的数据表，记录了每个批次的各个数据，其中绿色高亮的数据为第一次测试完全正确的批次。

Training Metrics Details

epoch	train_loss	train_acc	test_loss	test_acc	lr	time_sec
1	0.9490	0.5887	1.7803	0.4000	0.001000	15.5
2	0.7552	0.6718	0.4119	0.8000	0.001000	14.2
3	0.6897	0.7029	0.3463	0.8000	0.001000	14.3
4	0.6870	0.7184	0.4410	0.6000	0.001000	14.7
5	0.6131	0.7583	0.3016	1.0000	0.001000	14.9
6	0.5252	0.8016	0.3146	1.0000	0.000100	15.2
7	0.4931	0.8049	0.4415	0.7000	0.000100	14.9
8	0.4950	0.8237	0.4303	0.9000	0.000100	14.7
9	0.4632	0.8237	0.3695	0.9000	0.000100	14.6
10	0.4748	0.8326	0.3100	1.0000	0.000100	14.8
11	0.4480	0.8282	0.2585	1.0000	0.000010	14.9
12	0.4606	0.8348	0.2595	1.0000	0.000010	15.0
13	0.4336	0.8514	0.2636	1.0000	0.000010	15.1
14	0.4537	0.8415	0.2501	1.0000	0.000010	15.0
15	0.4874	0.8392	0.2518	1.0000	0.000010	15.0
16	0.4842	0.8503	0.2308	1.0000	0.000001	15.5
17	0.5510	0.8448	0.2326	1.0000	0.000001	15.4
18	0.5425	0.8481	0.2405	1.0000	0.000001	15.0
19	0.4031	0.8647	0.2511	1.0000	0.000001	15.0
20	0.4368	0.8537	0.2490	1.0000	0.000001	14.8

下面是绿色高亮批次的混淆矩阵图。我们可以看到元素都在对角线上，说明所有样本均被正确识别，并且展现了很高的相关性（因为样本数量少，而且正确率为 100%，因此混淆矩阵所有的数据都集中在对角线上）。



2. 评测指标展示及分析

经过大量测试，我们的训练准确率稳定在 85%上下，测试准确率基本能稳定在 1，说明我们的模型具有比较强的学习能力，能够在学习之后作出正确的判断。

我们的模型每个批次的平均运行时间为 15 秒左右，这是一个比较快的速度。主要是因为我们每个批次的 batch 仅为 15，这是一个比较小的数值，并且我们通过大量运行程序，发现绝大部分情况下测试准确率能稳定在 1，因此 20 个批次、15 个 batch 已经足够了，不需要再增加。

四、 参考资料