# 信息与软件工程学院上机实验报告

# （第 3 次）

## 一、实验名称

利用卷积神经网络进行树叶图片分类

## 二、实验目的及要求

1. 掌握卷积神经网络的结构、原理
2. 掌握池化的原理和作用

## 三、实验环境

1. Python 3
2. Anaconda
3. PyTorch

## 四、实验内容

1. 实现树叶分类算法
2. 提交 Kaggle 评价自己的算法结果
3. 提交学习通

## 五、实验设计内容要求

### 1. 实验步骤

本实验的目标是利用卷积神经网络完成树叶图片分类。数据集中包含 18353 张训练图片、8800 张测试图片，共 176 个类别。由于这是一个标准的图像多分类任务，因此实验流程按照“数据读取 → 图像预处理 → 模型训练 → 验证评估 → 预测生成提交文件”的思路展开。

1.1 设计思想

本实验首先读取 `train.csv`、`test.csv` 和 `sample_submission.csv`，并检查图片文件是否完整，确保每条记录对应的图片都能够被正确读取。随后，将训练集中的字符串标签映射为整数编号，以便于分类模型训练。为了评估模型效果，程序将原始训练集按照 8 : 2 的比例划分为训练子集和验证子集，并优先使用分层抽样，以尽量保持类别分布稳定。

在模型方面，搭建了一个较为基础的卷积神经网络。模型主体由 4 个卷积块组成，每个卷积块都包含卷积层、批归一化、ReLU 激活函数和最大池化层；在卷积特征提取之后，再通过自适应平均池化和全连接层完成最终的 176 类分类。这样设计的好处是结构更直观，更能对应卷积和池化的内容。训练过程中，程序会在每个 epoch 后计算验证集准确率，并保存表现最好的模型参数，从而保证最终预测时使用的是泛化效果最好的 checkpoint。

1.2 实现流程

程序的整体实现流程如图所示。

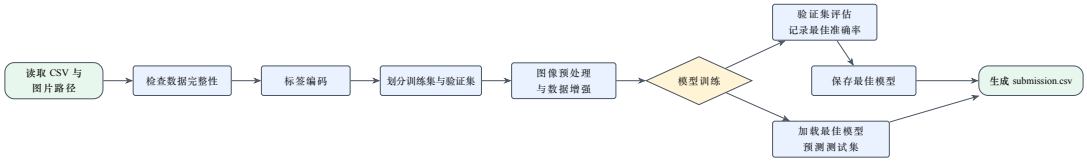



图 1: 实验三程序流程图

1.3 主要实现过程

本实验的主要实现步骤如下。按照“数据准备 → 模型训练 → 结果预测”的顺序，把图像分类任务的完整流程跑通。模型部分采用自己实现的卷积神经网络，重点放在理解卷积层、池化层和分类层之间的连接关系。

1. 在程序开头集中定义数据目录、输出目录、超参数和模型保存路径，便于统一修改。
2. 读取 `train.csv`、`test.csv` 和 `sample_submission.csv`，作为训练、预测和提交文件生成的基础数据。
3. 使用 `build_label_maps()` 将类别标签映射为整数索引，同时保留从索

引到类别名的反向映射，便于最后恢复预测标签。

4. 使用 `train_test_split()` 按 $8:2$ 的比例划分训练子集和验证子集，并优先使用分层抽样保持类别分布稳定。

5. 对所有图片统一缩放到 $224 \times 224$；训练阶段额外加入随机水平翻转和轻量颜色扰动；所有阶段都使用 ImageNet 的均值和标准差进行标准化。

6. 自定义 `LeavesDataset` 类，并配合 `DataLoader` 统一组织训练、验证和测试数据。

7. 使用自定义 `LeafCNN` 作为分类模型，其主要结构为：

$$\begin{aligned}\text{ConvBlock}(3,32) &\to \text{ConvBlock}(32,64) \to \text{ConvBlock}(64,128)\\ &\to \text{ConvBlock}(128,256) \to \text{AdaptiveAvgPool}\\ &\to \text{Linear}(256,128) \to \text{Linear}(128,176)\end{aligned}$$

其中每个 `ConvBlock` 都包含卷积、批归一化、ReLU 和最大池化，用于逐步提取图像特征。

8. 对于分类任务，损失函数选择交叉熵损失：

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{p}_i)$$

其中 $C$ 为类别数，$y_i$ 为真实标签的 one-hot 表示，$\hat{p}_i$ 为模型预测概率。

9. 优化器选择 `AdamW`，本实验使用的主要超参数如下表所示。

表 1: 主要训练超参数

| 参数项 | 取值 |
|---|---|
| 学习率 | `1e-3` |
| 权重衰减 | `1e-4` |
| Batch Size | `32` |
| 最大 Epoch | `12` |
| Patience | `5` |

10. 在每个 epoch 结束后分别计算训练集与验证集的损失和准确率；如果验证集准确率提升，则立即保存最佳模型参数文件 `best_leaf_cnn.pt`。

11. 训练结束后，通过 `predict` 命令读取最佳模型，对测试集 8800 张图片逐批预测，并输出 `submission.csv`。

## 2. **调试过程及实验结果**

### 2.1 调试重点

在调试过程中，主要遇到了以下几个问题。

第一，最开始下载的数据并不完整，`train.csv` 和 `test.csv` 中引用的部分图片文件不存在，导致训练无法直接开始。之后重新从 Kaggle 比赛页面下载完整数据并解压，数据检查通过后才进入正式训练。

第二，在本地调试阶段，完整训练一轮的速度较慢，不利于反复试验和生成结果，因此在流程跑通之后，将同一套代码迁移到 Kaggle 免费 GPU 环境中继续训练和提交，这样既能保留本地方便调试的优点，也能缩短正式训练时间。

第三，在自己搭建卷积神经网络时，需要保证卷积层输出的特征维度能够正确接入后面的全连接层。为避免手工计算特征图大小时出错，程序在卷积部分后加入了自适应平均池化层，将输出统一压缩为固定大小，再送入分类层。

第四，在训练后期，训练集准确率持续上升，但验证集准确率在不同 epoch 间开始波动，说明模型逐渐接近过拟合。为避免最终使用最后一轮而不是最佳一轮的模型，程序在每轮验证后都会保存验证集准确率最高的 checkpoint。

第五，在预测阶段，需要确保使用的是最佳模型参数而不是最后一轮参数。本实验通过加载 `best_leaf_cnn.pt` 来保证提交文件对应的是验证集表现最好的模型。

### 2.2 实验结果

下面先给出一次完整训练过程中的本地验证结果，再给出 Kaggle 竞赛页面上的提交结果。

本次实验的主要结果如下表所示。

表 2: 本地完整训练的主要结果

| 指标 | 数值 |
| --- | --- |
| 训练样本数 | 14682 |
| 验证样本数 | 3671 |
| 类别数 | 176 |
| 最佳 epoch | 12 |
| 最佳验证准确率 | 0.4887 |
| 最终训练准确率 | 0.3713 |

不同 epoch 的关键结果如下表所示。

表 3: 各 epoch 关键训练结果

| Epoch | Train Loss | Train Acc | Valid Loss | Valid Acc |
| --- | --- | --- | --- | --- |
| 1 | 4.6982 | 0.0379 | 4.0930 | 0.0806 |
| 2 | 3.9638 | 0.0807 | 3.5128 | 0.1278 |
| 4 | 3.2818 | 0.1593 | 3.0837 | 0.1948 |
| 6 | 2.8867 | 0.2303 | 2.4212 | 0.3231 |
| 8 | 2.6125 | 0.2790 | 2.3934 | 0.3043 |
| 10 | 2.3637 | 0.3318 | 2.1663 | 0.3887 |
| 12 | 2.1659 | 0.3713 | 1.7781 | 0.4887 |

从结果可以看到，模型在训练过程中逐步收敛，训练损失持续下降，训练准确率也由 0.0379 提升到了 0.3713；验证集准确率整体呈上升趋势，在第 12 个 epoch 达到 0.4887。说明当前自定义卷积神经网络已经能够学习到一部分有效的图像特征，但整体分类效果还有进一步优化空间。

本实验生成的训练过程曲线如下图所示。

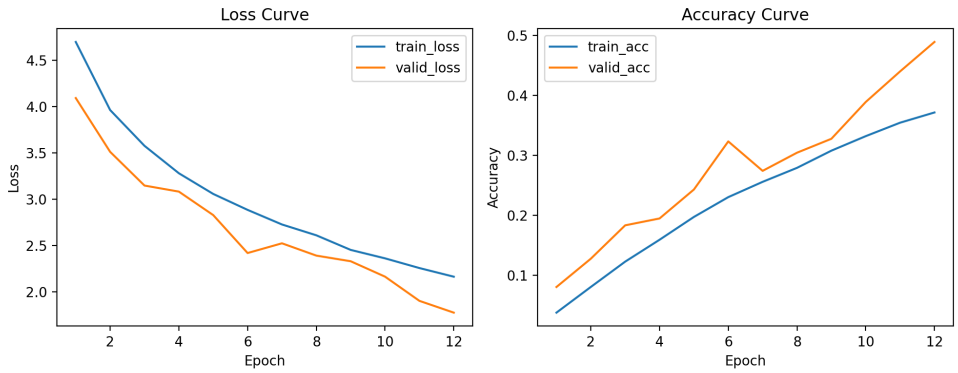



图 2: 训练损失与准确率变化曲线

从图中可以看出，训练损失整体下降，验证损失也在波动中逐步减小；训练准确率与验证准确率都在持续提升，这与表格中的数值变化基本一致。

程序已经成功生成 `submission.csv`，并完成 Kaggle 竞赛页面提交。在截图中，自定义 `LeafCNN` 的提交结果为 Public Score 0.46795、Private Score 0.45659；采用 `EfficientNet-B0` 骨干网络并接自定义分类头后的提交结果为 Public Score 0.83954、Private Score 0.83340。

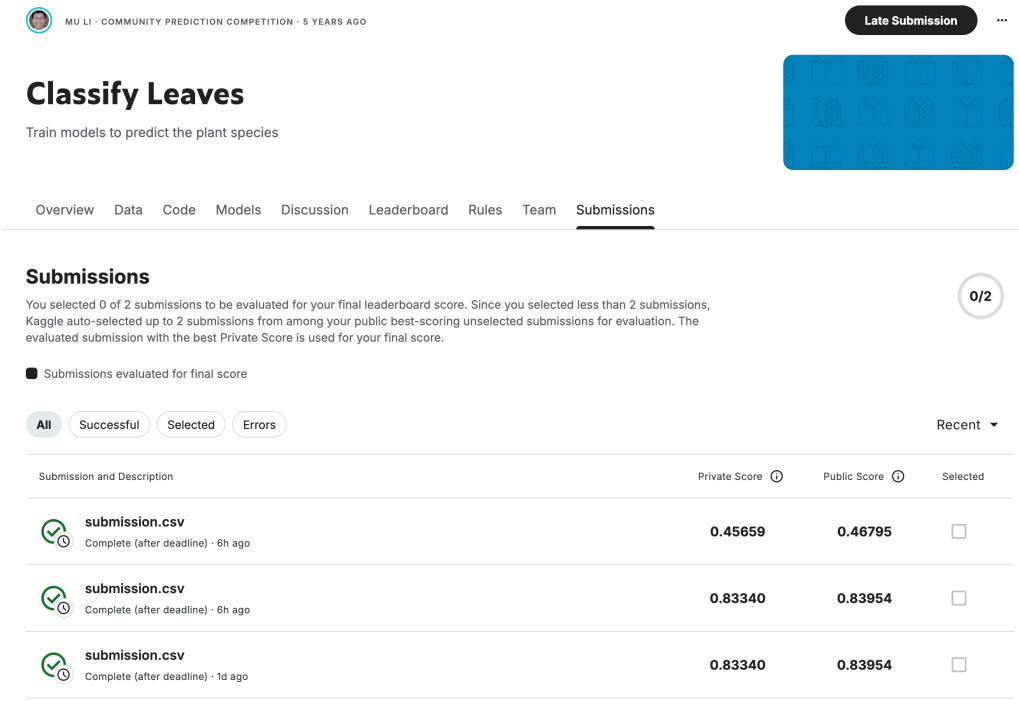



图 3: Kaggle 提交结果截图

## 2.3 结果分析

从本次实验结果来看，自定义卷积神经网络已经能够完成 176 类树叶分类任务，说明数据读取、图像预处理、模型训练、验证评估和测试预测各环节均能够正常工作。

从本地训练结果来看，模型在 12 轮训练内保持了稳定收敛，但验证准确率仍然不算很高，说明当前网络结构和训练策略还有进一步改进空间；而从 Kaggle 提交结果来看，完整流程已经成功跑通，并且模型在竞赛测试集上取得了较为稳定的成绩，这也说明当前实现已经具备完成实验任务的能力。

3. **总结**

通过本次实验，我对卷积神经网络在图像分类任务中的完整实现流程有了更加系统的认识。与前两个实验相比，本次实验的数据类型从表格和模拟数据变成了真实图像，因此我进一步理解了图像预处理、数据增强、自定义卷积网络和分类评价指标在实际任务中的作用。

在这次实验过程中，我也进一步体会到，图像分类实验不仅要会调用现成工具，更要真正理解卷积层、池化层、批归一化和全连接层是如何组合成一个完整分类模型的。通过自己搭建 `LeafCNN`，我对卷积神经网络从特征提取到最终分类的整个过程有了更具体的认识，也更加理解了各个模块在实际任务中的作用。

本次实验让我比较完整地掌握了图像分类任务从数据检查、标签编码、图像预处理、数据增强、模型训练、验证评估到测试集预测和 Kaggle 提交的整套流程，也进一步理解了交叉熵损失、准确率指标和最佳 checkpoint 保存机制在分类任务中的实际作用。相比只看课件中的概念，这次亲自把代码跑通之后，我对为什么要做标准化、为什么训练和验证要分开、为什么最终预测必须加载验证集表现最好的模型，都有了更加直观的认识。

如果后续继续改进，本实验还可以在保持代码清晰、便于理解的前提下，逐步加入学习率调度策略来提升后期训练稳定性，并进一步调整自定义卷积网络的通道数、卷积块数量和 dropout 比例，比较不同结构对分类效果的影响；同时也可以进一步使用 K 折交叉验证来提高结果稳定性，并观察本地验证结果与 Kaggle 线上成绩之间的对应关系，从而对完整实验流程建立更扎实的认识。

## 4. 附录

由于篇幅限制，这里仅保留能够体现完整流程的核心代码摘录，完整实现见源文件 nn03.py。

### 4.1 核心代码一：数据预处理与数据集封装

```
def build_transform(image_size: int, *, is_train: bool) -> transforms.Compose:
    if is_train:
        return transforms.Compose(
            [
                transforms.Resize((image_size, image_size)),
                transforms.RandomHorizontalFlip(p=0.5),
                transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
                transforms.ToTensor(),
                transforms.Normalize(mean=IMAGENET_MEAN, std=IMAGENET_STD),
            ]
        )

    return transforms.Compose(
        [
            transforms.Resize((image_size, image_size)),
            transforms.ToTensor(),
            transforms.Normalize(mean=IMAGENET_MEAN, std=IMAGENET_STD),
        ]
    )


class LeavesDataset(Dataset):
    def __init__(self, dataframe, *, image_root, transform, class_to_idx):
        self.dataframe = dataframe.reset_index(drop=True)
        self.image_root = image_root
        self.transform = transform
        self.class_to_idx = class_to_idx

    def __getitem__(self, index: int):
        row = self.dataframe.iloc[index]
        image = Image.open(self.image_root / row["image"]).convert("RGB")
        image_tensor = self.transform(image)
```

```
        if self.class_to_idx is None:
            return image_tensor

        label_index = self.class_to_idx[row["label"]]
        return image_tensor, label_index
```

4.2 核心代码二：模型构建与训练主循环

```
class ConvBlock(nn.Module):
    def __init__(self, in_channels: int, out_channels: int) -> None:
        super().__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.block(x)


class LeafCNN(nn.Module):
    def __init__(self, num_classes: int) -> None:
        super().__init__()
        self.features = nn.Sequential(
            ConvBlock(3, 32),
            ConvBlock(32, 64),
            ConvBlock(64, 128),
            ConvBlock(128, 256),
        )
        self.pool = nn.AdaptiveAvgPool2d((1, 1))
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256, 128),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.3),
            nn.Linear(128, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
```

```
        x = self.pool(x)
        x = self.classifier(x)
        return x


def train() -> None:
    device = get_device()
    train_df = load_csv(TRAIN_CSV, require_label=True)
    class_names, class_to_idx = build_label_maps(train_df)
    train_subset, valid_subset = split_train_valid(train_df)

    train_dataset = LeavesDataset(
        train_subset,
        transform=build_transform(is_train=True),
        class_to_idx=class_to_idx,
    )
    valid_dataset = LeavesDataset(
        valid_subset,
        transform=build_transform(is_train=False),
        class_to_idx=class_to_idx,
    )

    model = LeafCNN(num_classes=len(class_names)).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.AdamW(
        model.parameters(),
        lr=LEARNING_RATE,
        weight_decay=WEIGHT_DECAY,
    )

    for epoch in range(1, EPOCHS + 1):
        train_metrics = train_one_epoch(model, train_loader,
    criterion, optimizer, device, ...)
        valid_metrics = evaluate(model, valid_loader, criterion,
    device, ...)

        if epoch == 1 or valid_metrics["accuracy"] >
    best_valid_accuracy:
            best_valid_accuracy = valid_metrics["accuracy"]
            save_checkpoint(model, class_names,
    best_valid_accuracy)
```

### 4.3 核心代码三：测试集预测与提交文件生成

```
@torch.no_grad()
```

```python
def predict_indices(model: nn.Module, loader: DataLoader, device: torch.device) -> list[int]:
    model.eval()
    predictions = []

    for images in loader:
        images = images.to(device)
        logits = model(images)
        predictions.extend(logits.argmax(dim=1).cpu().tolist())

    return predictions


def predict() -> None:
    checkpoint = torch.load(CHECKPOINT_PATH, map_location="cpu")
    class_names = checkpoint["class_names"]

    model = LeafCNN(num_classes=len(class_names)).to(get_device())
    model.load_state_dict(checkpoint["model_state_dict"])

    test_df = load_csv(TEST_CSV)
    test_dataset = LeavesDataset(
        test_df,
        transform=build_transform(is_train=False),
        class_to_idx=None,
    )
    test_loader = build_loader(test_dataset, shuffle=False)

    prediction_indices = predict_indices(model, test_loader, get_device())
    submission = pd.read_csv(SAMPLE_SUBMISSION_CSV)
    submission["label"] = [class_names[index] for index in prediction_indices]
    submission.to_csv(SUBMISSION_PATH, index=False)
```