**21大数据b班**

**421470153**

**吴文豪**

**基于深度学习的图像识别系统——水果分类项目**

**一、项目背景与目的**

**1. 背景**

图像识别是人工智能领域的重要技术之一，其核心是通过计算机视觉技术，模拟人类视觉系统，分析和理解图像内容。在实际应用中，图像识别被广泛应用于自动驾驶、人脸识别、医疗影像分析等领域。

水果分类是图像识别的一项具体任务，主要目标是识别不同种类的水果。这在智能农业、零售自动化（如无人超市）等场景中有重要应用。例如，自动分拣系统可以通过识别水果类别实现智能化物流，提高效率。

**2.目的**

1、培养对图像处理和深度学习技术的基本理解。

2、提升使用深度学习模型解决实际问题的能力。

3、掌握数据预处理、模型构建、评估及优化的技能。

4、通过实验探索如何设计和改进深度学习模型，以提高模型性能。

**二、数据预处理**

**1.数据集描述：**

实验使用了一组包含多种水果类别的图片数据集。该数据集包括 苹果（apple）、香蕉（banana）、葡萄（grape）、橙子（orange） 和 梨（pear） 五种水果的图片。主要特点如下：

图片特性： 数据集由彩色图片组成，但分辨率不统一。

类别分布： 每个类别的图片数量接近均匀，有助于模型分类任务的平衡性。

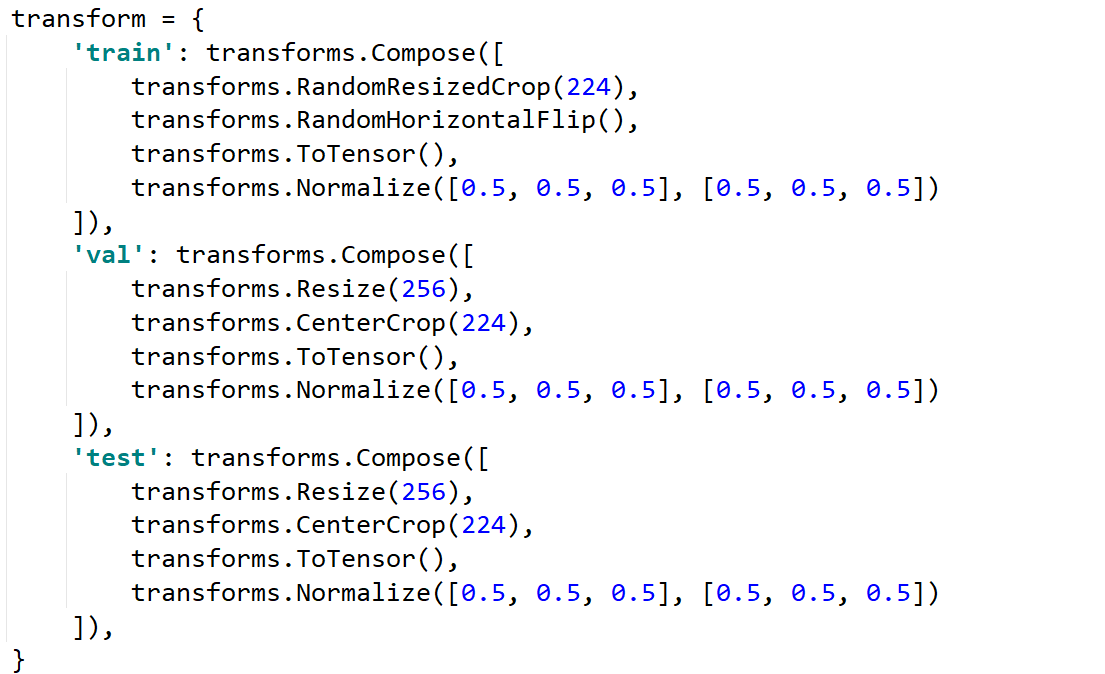
背景复杂度： 图片背景的复杂度适中，为模型学习特征和增强鲁棒性提供了良好的基础。

**2.预处理步骤：**

1、图像尺寸调整：为适应模型输入要求，将所有图像调整为统一大小（224×224 像素）。

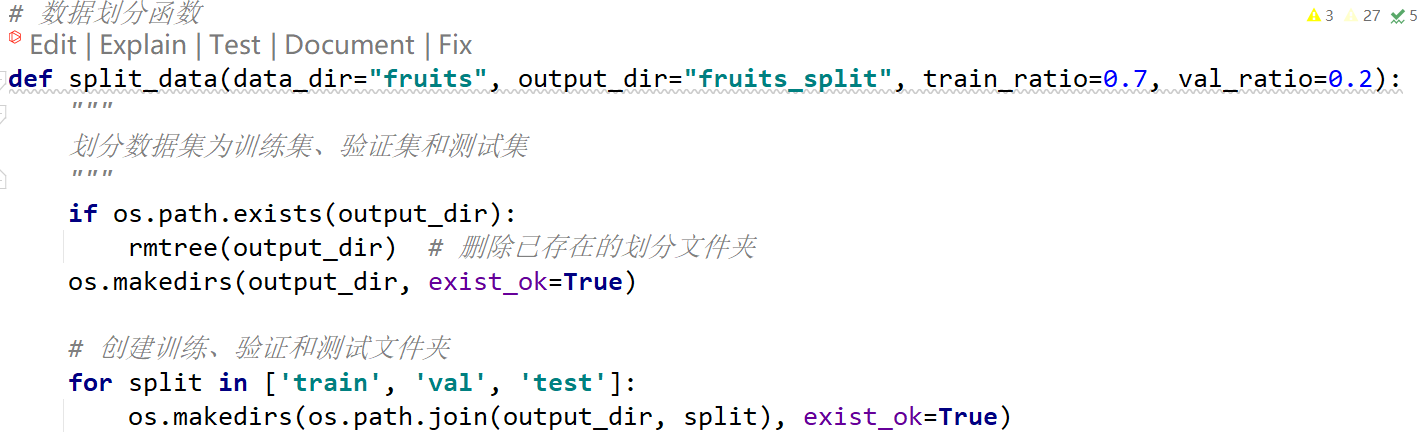
2、归一化处理：将像素值缩放至 [0, 1] 的范围，同时以均值和标准差对其进行标准化处理，使模型更快收敛。

3、数据增强：使用旋转、水平翻转、随机裁剪等方法生成更多样化的数据，提高模型对不同图像变换的适应能力。



**3.数据集划分：**

按比例将数据集划分为训练集、验证集和测试集，比例分别为 70%、20%、10%。确保每个类别在各子集中均匀分布。



**三、模型构建**

**1、模型选择**

实验选择卷积神经网络（CNN）作为基础模型。CNN 是处理图像数据的最常用深度学习模型，其卷积操作能够提取图像的空间特征，同时池化操作减少了特征维度并提高了计算效率。

**2、模型架构：**

模型的具体设计如下：

（1）卷积层（Conv Layer）：

第1层：输入通道数为3（RGB图像），输出通道数为16，使用3×3卷积核，激活函数为ReLU。

第2层：输出通道数为32，卷积核大小为3×3，激活函数为ReLU。

第3层：输出通道数为64，卷积核大小为3×3，激活函数为ReLU。

（2）池化层（Max Pooling）：

在每个卷积层之后添加最大池化层，池化核大小为2×2，减少特征图大小。

（3）全连接层（Fully Connected Layer）：

第1层：输入特征维度为64×28×28（卷积输出展开），输出节点数为128。

第2层：输入节点数为128，输出节点数等于类别数（5）。

（4）激活函数：

卷积层和全连接层使用 ReLU 激活函数，提高非线性表达能力。

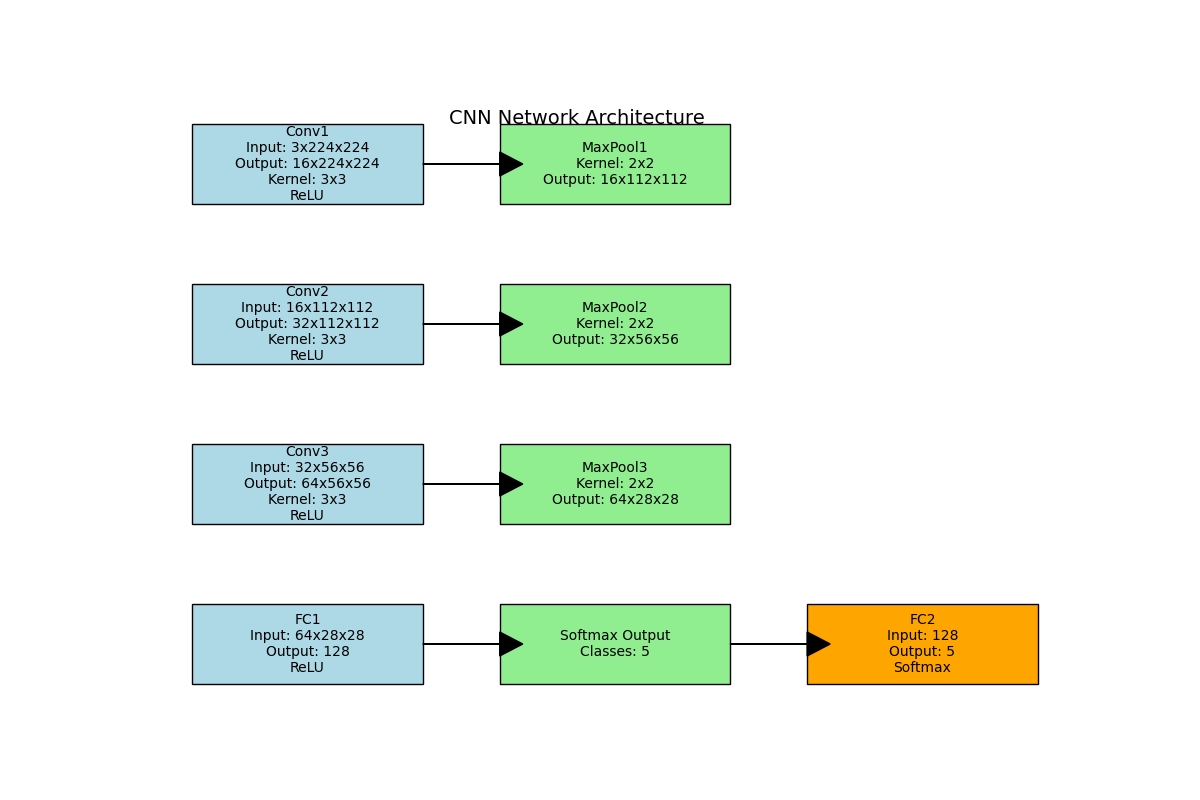
输出层使用 Softmax 激活函数，生成每类的概率分布。

（5）损失函数：

使用交叉熵损失函数（CrossEntropyLoss），适用于多分类问题。

（6）优化器：

使用 Adam 优化器，初始学习率设为 0.001，兼顾收敛速度和稳定性。



**四、模型评估**

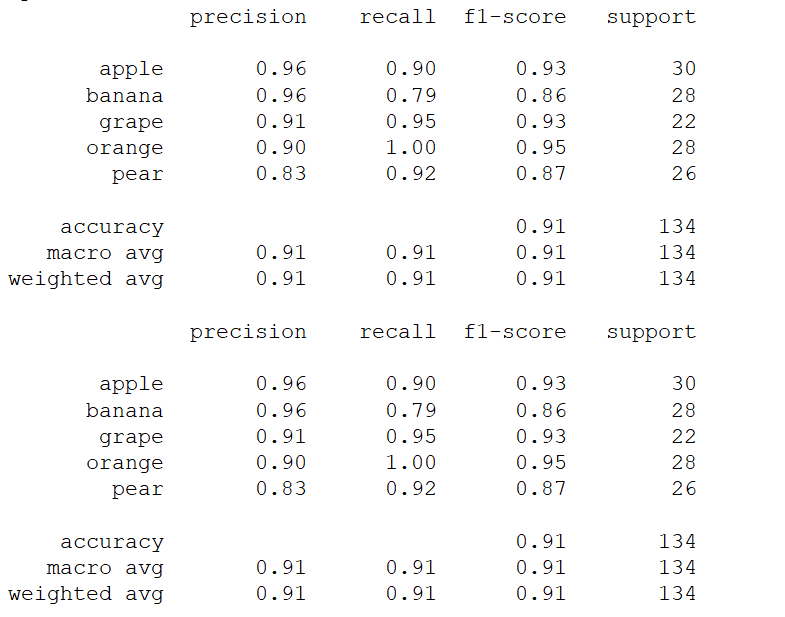
**1.评估指标：**

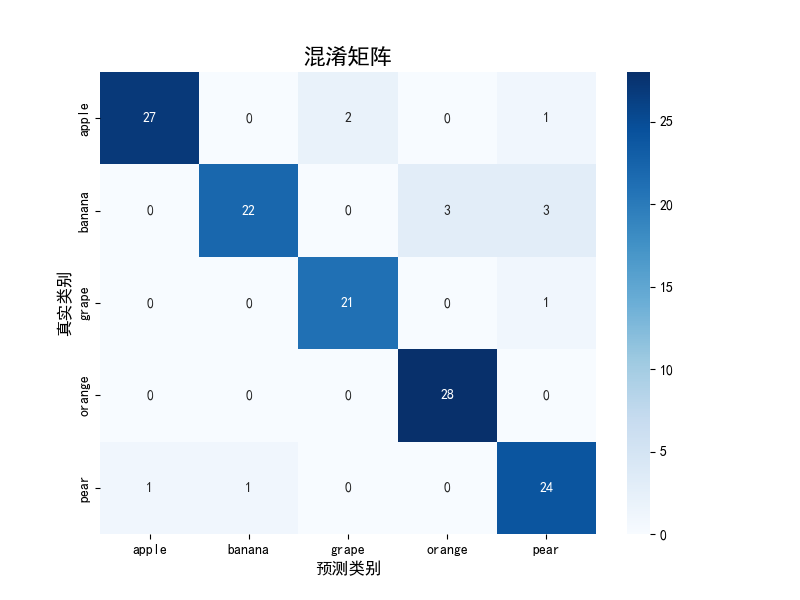
1、准确率（Accuracy）：分类正确样本占总样本的比例，用于衡量整体分类性能。

2、召回率（Recall）：针对每一类别，表示被正确分类的样本占实际样本的比例。

3、F1 分数：综合考虑精准率和召回率，适用于不平衡数据集。

4、混淆矩阵（Confusion Matrix）：展示分类的具体分布情况，分析模型对每一类别的识别能力。





**2、评估方法：**

使用验证集在训练过程中评估模型性能，记录验证损失，防止过拟合。

在测试集上进行最终评估，使用混淆矩阵和分类报告展示模型结果。

**五、结果分析与优化**

**1.结果分析：**

（1）性能评估：

模型在测试集上的分类准确率达到了 91%，表明整体分类性能较好。

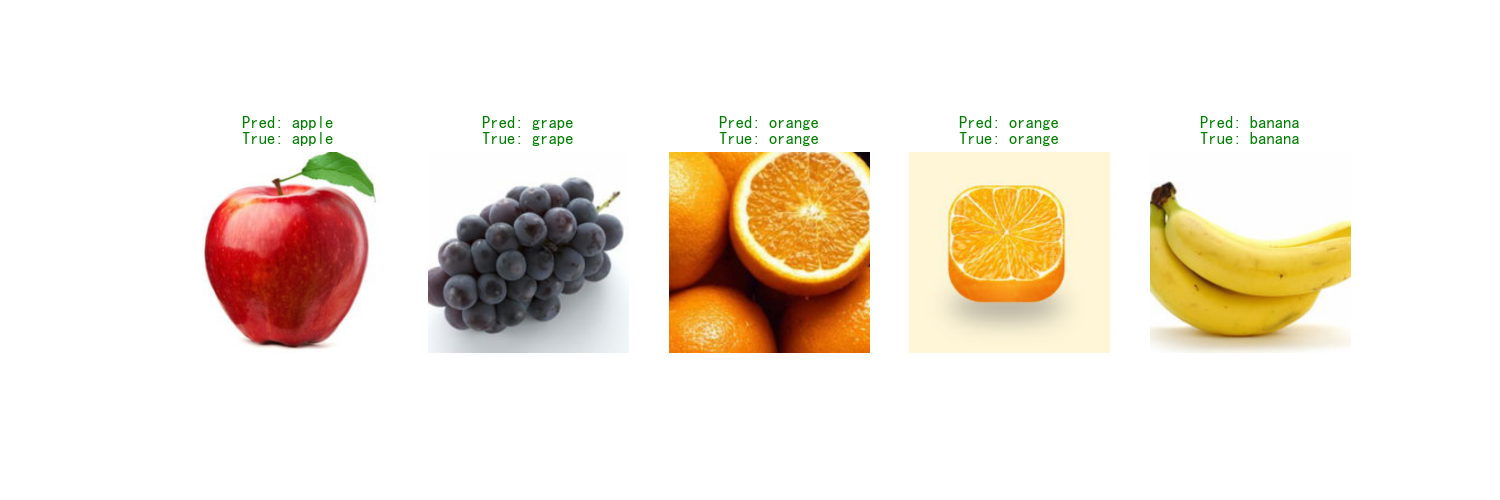
分类报告显示，所有类别的 Precision、Recall 和 F1 分数均高于 85%，其中橙子类（orange）表现最佳，F1 分数达到 95%。

混淆矩阵显示，苹果（apple）和梨（pear）之间存在一定程度的混淆，这可能是由于它们在形状或颜色上的相似性导致。

（2）训练过程：

在 20 个训练周期中，训练损失从 3.02 降至 0.45，验证损失从 1.05 降至 0.27，表明模型收敛效果良好。

验证损失在训练中后期的下降幅度逐渐减小，表明模型性能逐步达到上限，训练效果稳定。



**2.模型优化：**

（1）调整超参数：

尝试降低学习率，避免训练后期的振荡。

增加训练周期，观察是否能进一步提高性能。

（2）改进数据预处理：

增加更多的数据增强方式，如颜色抖动、随机旋转等，增强模型对复杂背景的适应能力。

（3）调整模型结构：

增加卷积层或调整卷积核大小，提取更多高级特征。

引入 Dropout 正则化技术，进一步减少过拟合风险。

（4）迁移学习：

使用预训练模型（如 ResNet 或 VGG），结合实验数据进行微调，可能进一步提升分类性能。

**六、总结**

本实验成功构建了一个基于深度学习的水果分类系统。通过数据预处理、模型设计和优化，模型在测试集上取得了 91% 的分类准确率。结果表明，卷积神经网络在水果图像分类任务上具有较强的适用性。

未来可以尝试引入更复杂的网络架构或数据增强技术，提高模型的泛化能力。此外，还可以扩展至更多类别的水果分类任务，为实际应用提供更通用的解决方案。

附录

import os

import random

from shutil import copy2, rmtree, copy

from PIL import Image

from matplotlib import pyplot as plt

from torchvision import datasets, transforms

import numpy as np

import torch

import torch.nn as nn

from torch import optim

from torch.utils.data import DataLoader, Dataset

from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

# 导入模块

import copy # 引入 copy 模块

from shutil import copy as shutil\_copy # 避免覆盖 copy 模块

# 设置 Matplotlib 支持中文显示

plt.rcParams['font.family'] = 'SimHei'

plt.rcParams['axes.unicode\_minus'] = False # 解决坐标轴负号显示问题

# 设置随机种子

def setup\_seed(seed):

random.seed(seed)

torch.manual\_seed(seed)

np.random.seed(seed)

setup\_seed(42)

# 数据划分函数

def split\_data(data\_dir="fruits", output\_dir="fruits\_split", train\_ratio=0.7, val\_ratio=0.2):

"""

划分数据集为训练集、验证集和测试集

"""

if os.path.exists(output\_dir):

rmtree(output\_dir) # 删除已存在的划分文件夹

os.makedirs(output\_dir, exist\_ok=True)

# 创建训练、验证和测试文件夹

for split in ['train', 'val', 'test']:

os.makedirs(os.path.join(output\_dir, split), exist\_ok=True)

# 遍历类别文件夹

for class\_name in os.listdir(data\_dir):

class\_path = os.path.join(data\_dir, class\_name)

if not os.path.isdir(class\_path):

continue

# 获取所有图像文件

images = [f for f in os.listdir(class\_path) if os.path.isfile(os.path.join(class\_path, f))]

random.shuffle(images)

# 计算每个子集的样本数量

train\_count = int(len(images) \* train\_ratio)

val\_count = int(len(images) \* val\_ratio)

# 划分数据

train\_images = images[:train\_count]

val\_images = images[train\_count:train\_count + val\_count]

test\_images = images[train\_count + val\_count:]

# 复制数据到对应文件夹

for img in train\_images:

os.makedirs(os.path.join(output\_dir, 'train', class\_name), exist\_ok=True)

copy2(os.path.join(class\_path, img), os.path.join(output\_dir, 'train', class\_name))

for img in val\_images:

os.makedirs(os.path.join(output\_dir, 'val', class\_name), exist\_ok=True)

copy2(os.path.join(class\_path, img), os.path.join(output\_dir, 'val', class\_name))

for img in test\_images:

os.makedirs(os.path.join(output\_dir, 'test', class\_name), exist\_ok=True)

copy2(os.path.join(class\_path, img), os.path.join(output\_dir, 'test', class\_name))

print(f"数据划分完成，结果存储在 {output\_dir}")

# 数据加载

def load\_data(data\_dir="fruits\_split", batch\_size=32):

transform = {

'train': transforms.Compose([

transforms.RandomResizedCrop(224),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])

]),

'val': transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])

]),

'test': transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])

]),

}

# 加载数据

train\_dataset = datasets.ImageFolder(os.path.join(data\_dir, 'train'), transform=transform['train'])

val\_dataset = datasets.ImageFolder(os.path.join(data\_dir, 'val'), transform=transform['val'])

test\_dataset = datasets.ImageFolder(os.path.join(data\_dir, 'test'), transform=transform['test'])

# 数据加载器

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

val\_loader = DataLoader(val\_dataset, batch\_size=batch\_size, shuffle=False)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)

return train\_loader, val\_loader, test\_loader, train\_dataset.classes

# CNN 模型定义

class CNN(nn.Module):

def \_\_init\_\_(self, num\_classes=3):

super(CNN, self).\_\_init\_\_()

self.conv1 = nn.Sequential(

nn.Conv2d(3, 16, kernel\_size=3, stride=1, padding=1),

nn.BatchNorm2d(16),

nn.ReLU(),

nn.MaxPool2d(kernel\_size=2, stride=2)

)

self.conv2 = nn.Sequential(

nn.Conv2d(16, 32, kernel\_size=3, stride=1, padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.MaxPool2d(kernel\_size=2, stride=2)

)

self.conv3 = nn.Sequential(

nn.Conv2d(32, 64, kernel\_size=3, stride=1, padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

nn.MaxPool2d(kernel\_size=2, stride=2)

)

self.fc1 = nn.Linear(64 \* 28 \* 28, 128)

self.fc2 = nn.Linear(128, num\_classes)

def forward(self, x):

x = self.conv1(x)

x = self.conv2(x)

x = self.conv3(x)

x = x.view(x.size(0), -1)

x = torch.relu(self.fc1(x))

x = self.fc2(x)

return x

# 测试模型并绘制结果

def test\_and\_visualize\_model(model, test\_loader, classes, num\_images=5):

"""

测试模型并随机选取测试集中的图片进行预测和可视化

"""

model.eval()

all\_preds = []

all\_labels = []

all\_images = []

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

\_, preds = torch.max(outputs, 1)

all\_preds.extend(preds.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

all\_images.extend(images.cpu()) # 保存图片

# 打印分类报告

print(classification\_report(all\_labels, all\_preds, target\_names=classes))

# 随机选择 num\_images 张图片进行可视化

indices = random.sample(range(len(all\_images)), num\_images)

selected\_images = [all\_images[i] for i in indices]

selected\_preds = [all\_preds[i] for i in indices]

selected\_labels = [all\_labels[i] for i in indices]

# 绘制图片和预测结果

plt.figure(figsize=(15, 5))

for i, image in enumerate(selected\_images):

image = image.permute(1, 2, 0) # 将图片从 (C, H, W) 转为 (H, W, C)

image = (image \* 0.5) + 0.5 # 去标准化

plt.subplot(1, num\_images, i + 1)

plt.imshow(image.numpy())

plt.axis('off')

pred\_label = classes[selected\_preds[i]]

true\_label = classes[selected\_labels[i]]

plt.title(f"Pred: {pred\_label}\nTrue: {true\_label}", color="green" if pred\_label == true\_label else "red")

plt.show()

# 主程序

if \_\_name\_\_ == "\_\_main\_\_":

# 设置设备

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# 数据划分

split\_data(data\_dir="fruits", output\_dir="fruits\_split")

# 数据加载

train\_loader, val\_loader, test\_loader, classes = load\_data(data\_dir="fruits\_split")

# 模型构建

model = CNN(num\_classes=len(classes)).to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

# 模型训练

def train\_model(model, train\_loader, val\_loader, criterion, optimizer, epochs=20):

best\_model = None

best\_val\_loss = float('inf')

for epoch in range(epochs):

model.train()

train\_loss = 0

for images, labels in train\_loader:

images, labels = images.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(images)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

train\_loss += loss.item()

val\_loss = 0

model.eval()

with torch.no\_grad():

for images, labels in val\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

loss = criterion(outputs, labels)

val\_loss += loss.item()

print(

f"Epoch {epoch + 1}, Train Loss: {train\_loss / len(train\_loader)}, Val Loss: {val\_loss / len(val\_loader)}")

# 保存当前最优模型

if val\_loss < best\_val\_loss:

best\_val\_loss = val\_loss

best\_model = copy.deepcopy(model)

return best\_model

best\_model = train\_model(model, train\_loader, val\_loader, criterion, optimizer)

# 保存模型

torch.save(best\_model.state\_dict(), "fruit\_model.pth")

# 测试模型

def test\_model(model, test\_loader, classes):

model.eval()

all\_preds = []

all\_labels = []

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

\_, preds = torch.max(outputs, 1)

all\_preds.extend(preds.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

# 打印分类报告

print(classification\_report(all\_labels, all\_preds, target\_names=classes))

# 绘制混淆矩阵

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)

plt.xlabel('预测类别', fontsize=12) # 中文

plt.ylabel('真实类别', fontsize=12) # 中文

plt.title('混淆矩阵', fontsize=16) # 中文

plt.show()

test\_model(best\_model, test\_loader, classes)

# 测试并绘制测试集结果

test\_and\_visualize\_model(best\_model, test\_loader, classes)