```
# -*- coding: utf-8 -*-
import os
import random
import shutil
import numpy as np
import tensorflow as tf
from scipy.io import loadmat
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
import json

# 为了在测试阶段执行图表绘制，可以将训练过程中的 history 变量保存到一个文件中。在训练阶段结束时将其保存为一个 JSON 文件，然后在测试阶段加载这个文件来绘制图表。
history_path = 'F:/文件夹名/flower_recognition_history.json'
# 定义模型保存路径
model_path = 'F:/文件夹名/flower_recognition_model.h5'
# 载入标签数据
labels = loadmat('F:/文件夹名/imagelabels.mat')['labels'][0]
# 这行代码使用 scipy.io 模块中的 loadmat 函数读取一个名为 imagelabels.mat 的 MATLAB 文件。文件中应包含一个名为 labels 的变量。loadmat 函数将.mat 文件中的数据加载为一个 Python 字典，其中键是变量名，值是相应的数据。['labels'][0]从这个字典中提取名为 labels 的数据，并选择第 0 个元素，得到一个一维的 NumPy 数组。
unique_labels = np.unique(labels)
# 这行代码使用 numpy 模块中的 unique 函数找出 labels 数组中所有唯一的标签值，并将它们保存在一个新的 NumPy 数组 unique_labels 中。
class_names = {i: f'class_{label}' for i, label in
enumerate(unique_labels)}
# 这行代码使用字典推导式创建一个字典 class_names。字典的键是唯一标签值在 unique_labels 数组中的索引（从 0 开始），字典的值是一个字符串，其格式为"class_"加上对应的唯一标签值。这个字典将用于将标签值映射到类名，方便后续使用和可视化。

# 划分数据集为训练集、验证集、测试集
train_ratio, val_ratio = 0.7, 0.2
# 这行代码定义了训练集和验证集的比例。train_ratio 是训练集的比例，设置为 0.7，表示训练集占原始数据的 70%。val_ratio 是验证集的比例，设置为
```

```
0.2，表示验证集占原始数据的 20%。剩余的 10%将用作测试集。
train_labels, test_labels = train_test_split(labels, test_size=1-
train_ratio, random_state=42)
# 这行代码使用 train_test_split 函数将原始标签数据 labels 划分为训练集和测试集。test_size 参数设置为 1-train_ratio（即 0.3），表示测试集占原始数据的 30%。然后，我们将训练集和验证集的划分留给下一行代码处理。random_state=42 用于设置随机数生成器的种子，以确保每次运行代码时，数据划分的方式相同。
train_labels, val_labels = train_test_split(train_labels,
test_size=val_ratio/(train_ratio+val_ratio), random_state=42)
# 这行代码再次使用 train_test_split 函数，这次是将上一行代码得到的 train_labels 进一步划分为实际的训练集和验证集。test_size 参数设置为 val_ratio/(train_ratio+val_ratio)（即 0.2/0.9≈0.222），这意味着验证集将占据之前划分的训练集的 22.2%。由于之前训练集占原始数据的 70%，因此验证集实际上占据原始数据的 20%（0.7 * 0.222 ≈ 0.2），符合我们最初的设定。random_state=42 同样用于设置随机数生成器的种子。
# 转换为独热编码
encoder = OneHotEncoder(sparse=False)
# 这行代码创建了一个 OneHotEncoder 对象，参数 sparse=False 表示生成的独热编码不是稀疏矩阵格式，而是普通的 NumPy 数组。
train_labels = encoder.fit_transform(train_labels.reshape(-1, 1))
# 这行代码首先使用 reshape(-1, 1)将 train_labels 转换为列向量，然后调用 fit_transform 方法对训练集标签进行独热编码。fit_transform 方法首先根据训练集标签学习编码方式（即确定类别数量和每个类别的位置），然后将这些标签转换为独热编码格式。
val_labels = encoder.transform(val_labels.reshape(-1, 1))
test_labels = encoder.transform(test_labels.reshape(-1, 1))
# 这两行代码分别对验证集和测试集的标签进行独热编码。使用 reshape(-1, 1)将标签转换为列向量，然后调用 transform 方法应用在第二行中学到的编码方式，将这些标签转换为独热编码格式。



##########################################################################
########################################
# 创建数据生成器：下面的代码使用 tensorflow.keras.preprocessing.image.ImageDataGenerator 创建了用于处理图像数据的生成器。生成器可以对图像进行实时数据增强（data augmentation），包括缩放、旋转、平移等操作，这有助于提高模型的泛化能力。
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
```

```
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
# train_datagen = ImageDataGenerator(...)
# 这个生成器用于训练数据集。参数定义了对训练图像执行的各种数据增强操作：
# rescale=1./255：对图像像素值进行缩放，将其范围从[0, 255]缩放到[0, 1]。这是一个常见的预处理操作，有助于模型训练。
# rotation_range=40：在[-40, 40]度范围内随机旋转图像。
# width_shift_range=0.2：在宽度上随机平移图像，最大平移距离为图像宽度的 20%。
# height_shift_range=0.2：在高度上随机平移图像，最大平移距离为图像高度的 20%。
# shear_range=0.2：以逆时针方向随机应用剪切变换（shear transformation），剪切角度在[-20, 20]度范围内。
# zoom_range=0.2：在[1-0.2, 1+0.2]范围内随机缩放图像。
# horizontal_flip=True：随机在水平方向翻转图像。
# fill_mode='nearest'：当对图像执行平移、旋转等操作时，可能会产生新的像素。这个参数指定使用最近邻插值（nearest-neighbor interpolation）填充这些新像素。

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
# val_datagen = ImageDataGenerator(rescale=1./255)
# 这个生成器用于验证数据集。唯一的预处理操作是 rescale=1./255，将像素值缩放到[0, 1]范围。注意，验证集和测试集不应该应用数据增强操作，因为我们希望它们代表实际应用场景。
# test_datagen = ImageDataGenerator(rescale=1./255)
# 这个生成器用于测试数据集。与 val_datagen 一样，仅对图像像素值进行缩放。



# 为每个数据集创建一个生成器：根据给定的标签和数据生成器，从磁盘加载图像并为每个数据集创建一个生成器。这是一个自定义生成器函数，可以在训练、验证和测试过程中用于提供批量图像数据。
def create_generator(labels, datagen):
    while True:
        indices = np.random.choice(np.arange(len(labels)), 32, replace=False)
        images = [tf.keras.preprocessing.image.load_img(f'F:/文件夹名/jpg/image_{i+1:05d}.jpg', target_size=(128, 128)) for i in indices]
        images = np.array([tf.keras.preprocessing.image.img_to_array(img) for img in
```

```
images])
        yield datagen.flow(images, labels[indices], batch_size=32,
shuffle=False).next()
# 以下是 create_generator 函数的逐行解释：
# def create_generator(labels, datagen):
# 定义一个名为 create_generator 的函数，该函数接受标签数组和一个
ImageDataGenerator 实例作为输入参数。
# while True:
# 使用一个无限循环，以便生成器可以持续产生批量数据。
# indices = np.random.choice(np.arange(len(labels)), 32,
replace=False)
# 从标签数组中随机选择 32 个不重复的索引。
# images = [tf.keras.preprocessing.image.load_img(f'F:/文件夹名
/jpg/image_{i+1:05d}.jpg', target_size=(128, 128)) for i in indices]
# 对于每个选定的索引，从磁盘加载相应的图像并将其大小调整为 128x128 像
素。
# images = np.array([tf.keras.preprocessing.image.img_to_array(img)
for img in images])
# 将加载的图像从 PIL（Python Imaging Library）图像对象转换为 NumPy 数
组。
# yield datagen.flow(images, labels[indices], batch_size=32,
shuffle=False).next()
# 使用输入参数 datagen（一个 ImageDataGenerator 实例）为这批图像和标签
应用预处理和数据增强操作（如果有的话），并在每次迭代时生成一个新的批
次。

train_generator = create_generator(train_labels, train_datagen)
val_generator = create_generator(val_labels, val_datagen)
test_generator = create_generator(test_labels, test_datagen)
# 这里为训练、验证和测试数据集分别创建了一个生成器实例。这些生成器将
在模型训练、验证和测试过程中使用，以提供批量的图像数据。

def plot_history(history):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history['accuracy'], label='train accuracy')
    plt.plot(history['val_accuracy'], label='val accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(history['loss'], label='train loss')
    plt.plot(history['val_loss'], label='val loss')
    plt.xlabel('Epoch')
```

```
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
# plot_history 函数的作用是可视化模型训练过程中的准确率（accuracy）和损失（loss）变化。这有助于了解模型在训练和验证过程中的性能。以下是逐行解释：

# def plot_history(history):
# 定义一个名为 plot_history 的函数，该函数接受一个包含训练历史记录的字典作为输入。
#
# plt.figure(figsize=(12, 4))
# 创建一个新的 Matplotlib 图形，设置图形大小为 12x4 英寸。
#
# plt.subplot(1, 2, 1)
# 创建一个 1 行 2 列的子图阵列，激活第一个子图。这个子图将用于显示训练和验证准确率变化。
#
# plt.plot(history['accuracy'], label='train accuracy')
# 绘制训练准确率随着 epoch 数的变化曲线。
#
# plt.plot(history['val_accuracy'], label='val accuracy')
# 绘制验证准确率随着 epoch 数的变化曲线。
#
# plt.xlabel('Epoch')
# 设置 x 轴标签为'Epoch'。
#
# plt.ylabel('Accuracy')
# 设置 y 轴标签为'Accuracy'。
#
# plt.legend()
# 在图上添加一个图例，显示各个曲线的标签。
#
# plt.subplot(1, 2, 2)
# 激活第二个子图。这个子图将用于显示训练和验证损失变化。
#
# plt.plot(history['loss'], label='train loss')
# 绘制训练损失随着 epoch 数的变化曲线。
#
# plt.plot(history['val_loss'], label='val loss')
# 绘制验证损失随着 epoch 数的变化曲线。
#
# plt.xlabel('Epoch')
# 设置 x 轴标签为'Epoch'。
```

```
#
# plt.ylabel('Loss')
# 设置 y 轴标签为'Loss'。
#
# plt.legend()
# 在图上添加一个图例，显示各个曲线的标签。
#
# plt.show()
# 显示生成的图形。在这个图形中，左侧子图展示了训练和验证准确率的变化，右侧子图展示了训练和验证损失的变化。通过观察这些曲线，我们可以了解模型在训练过程中的性能表现。

if os.path.exists(model_path):
    # 加载已有模型
    model = load_model(model_path)
else:
    # 构建模型
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(102, activation='softmax'))
    # 编译模型
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # 训练模型
    history = model.fit(train_generator, epochs=100, steps_per_epoch=100, validation_data=val_generator, validation_steps=50)
    # 保存模型
    model.save('F:/文件夹名/flower_recognition_model.h5')
    with open(history_path, 'w') as f:
        json.dump(history.history, f)
    plot_history(history)
# 这段代码的作用是将训练过程中的准确率和损失值保存到一个文件中，并调
```

```
用 plot_history 函数绘制这些值的变化曲线。以下是逐行解释：
#
# with open(history_path, 'w') as f:
# 使用 with 语句打开一个文件，文件路径由 history_path 指定，模式为写入（'w'）。with 语句在执行完成后会自动关闭文件。
#
# json.dump(history.history, f)
# 将 history.history 字典中的内容（即训练过程中的准确率和损失值）以 JSON 格式写入到打开的文件中。
#
# plot_history(history)
# 调用之前定义的 plot_history 函数，传入 history 对象作为参数。这个函数将绘制训练过程中准确率和损失值的变化曲线。
#
# 通过这段代码，你可以将训练过程中的性能指标（准确率和损失值）保存到文件中，以便于后续分析和比较。同时，plot_history 函数的调用使你能够直观地观察这些指标在训练过程中的变化，以便对模型进行评估和优化。



# 测试模型
test_loss, test_acc = model.evaluate(test_generator, steps=100)
print(f"Test accuracy: {test_acc:.4f}, Test loss: {test_loss:.4f}")
# 使用 model.evaluate()方法在测试集上评估模型的性能。这里的 test_generator 是一个生成器，用于提供测试样本。steps 参数表示要评估的批次数。将损失值和准确率分别存储在 test_loss 和 test_acc 中，并输出结果。
def display_test_results(test_generator, model, class_names, num_images=5):
    # 随机选择测试样本
    batch = np.random.choice(test_generator.samples, num_images)
    for i, idx in enumerate(batch):
        x, y = test_generator[idx]
        x = np.expand_dims(x, axis=0)
        preds = model.predict(x)
        pred_class = np.argmax(preds, axis=1)
        pred_prob = np.max(preds, axis=1)
        true_class = np.argmax(y)

        # 显示图像和预测结果
        plt.subplot(1, num_images, i + 1)
        plt.imshow(x[0].astype(np.uint8))
        plt.title(f"True: {class_names[true_class]}\nPredicted: {class_names[pred_class[0]]} ({pred_prob[0]*100:.2f}%)")
```

```
        plt.axis('off')
    plt.show()
# 定义一个名为 display_test_results 的函数，用于显示测试结果。函数接受以下参数：
#
# test_generator：测试数据生成器
# model：模型对象
# class_names：类别名称的字典
# num_images：要显示的图像数量，默认为 5
# 在函数内部，首先随机选择 num_images 个测试样本。然后，对于每个选定的样本，执行以下操作：
#
# 使用模型进行预测
# 获取预测类别和概率
# 获取真实类别
# 显示图像及其真实和预测类别
# 通过这个函数，您可以直观地查看模型在测试样本上的预测效果。

# 参数可视化
# 加载训练历史记录
with open(history_path, 'r') as f:
    loaded_history = json.load(f)

# 如果有训练历史记录，则显示图像
if loaded_history:
    plot_history(loaded_history)

def display_test_results(test_generator, model, class_names, num_images=5):
    indices = np.random.choice(np.arange(len(test_labels)), num_images, replace=False)
    images = [tf.keras.preprocessing.image.load_img(f'F:/文件夹名/jpg/image_{i+1:05d}.jpg', target_size=(128, 128)) for i in indices]
    images_array = np.array([tf.keras.preprocessing.image.img_to_array(img) for img in images])
    images_array = test_datagen.flow(images_array, batch_size=num_images, shuffle=False).next()

    preds = model.predict(images_array)
    pred_classes = np.argmax(preds, axis=1)
    pred_probs = np.max(preds, axis=1)
    true_classes = np.argmax(test_labels[indices], axis=1)
```

```
    for i, (img, pred_class, pred_prob, true_class) in
enumerate(zip(images, pred_classes, pred_probs, true_classes)):
        plt.subplot(1, num_images, i + 1)
        plt.imshow(img)
        plt.title(f"True: {class_names[true_class]}\nPredicted:
{class_names[pred_class]} ({pred_prob*100:.2f}%)")
        plt.axis('off')
    plt.show()

# def display_test_results(test_generator, model, class_names,
num_images=5):
# 定义一个名为 display_test_results 的函数，输入参数为测试生成器、模
型、类别名称和要展示的图像数量。默认展示 5 张图像。
#
# indices = np.random.choice(np.arange(len(test_labels)), num_images,
replace=False)
# 从测试标签中随机选择 num_images 个不重复的索引。
#
# images = [tf.keras.preprocessing.image.load_img(f'F:/文件夹名
/jpg/image_{i+1:05d}.jpg', target_size=(128, 128)) for i in indices]
# 使用这些随机选择的索引，从指定文件夹加载对应的图像，将其大小调整为
128x128。
#
# images_array =
np.array([tf.keras.preprocessing.image.img_to_array(img) for img in
images])
# 将加载的图像转换为 Numpy 数组。
#
# images_array = test_datagen.flow(images_array,
batch_size=num_images, shuffle=False).next()
# 对图像应用测试数据生成器的规范化，即将图像数据缩放到 0-1 之间。
#
# preds = model.predict(images_array)
# 使用模型对这些图像进行预测，得到预测结果。
#
# pred_classes = np.argmax(preds, axis=1)
# 从预测结果中找到概率最高的类别索引。
#
# pred_probs = np.max(preds, axis=1)
# 获取每张图像预测概率最高的值。
#
# true_classes = np.argmax(test_labels[indices], axis=1)
# 根据随机选择的索引，获取相应的真实类别。
#
```

```
# for i, (img, pred_class, pred_prob, true_class) in
enumerate(zip(images, pred_classes, pred_probs, true_classes)):
# 遍历每张图像及其相关数据（预测类别、预测概率和真实类别）。
#
# plt.subplot(1, num_images, i + 1)
# 创建一个子图，每行显示 num_images 张图像。
#
# plt.imshow(img)
# 显示原始图像。
#
# plt.title(f"True: {class_names[true_class]}\nPredicted:
{class_names[pred_class]} ({pred_prob*100:.2f}%)")
# 在图像上方添加标题，显示真实类别、预测类别和预测概率。
#
# plt.axis('off')
# 关闭坐标轴。
#
# plt.show()
# 显示整个图像组合。
#
# 举例来说，如果 num_images=5，那么函数将随机选择 5 个不重复的索引，并
加载相应的图像。然后，使用模型进行预测并显示每张图像的真实类别、预测
类别和预测概率。最后，这 5 张图像将按顺序排列在一行中展示。
# 显示测试样本及其预测结果
display_test_results(test_generator, model, class_names)
```