一、代码分析

（1）model.py：ResNet 骨干网络定义模块

①核心功能：实现 ResNet（18、34、50、101、152）结构，用于图像特征提取，自定义 BasicBlock 和 Bottleneck 两种基本模块（对应浅层与深层网络）。

②关键代码详解：

✓ **BasicBlock 模块（适用于 ResNet18/34）：**

```
class BasicBlock(nn.Module):
    expansion = 1
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        ...
```

expansion=1 表示输出通道数不变，包含两个 3×3 卷积 + BN + ReLU，如果输入输出通道数不匹配，用 downsample 做捷径匹配。

✓ **Bottleneck 模块（适用于 ResNet50/101/152）：**

```
class Bottleneck(nn.Module):
    expansion = 4
```

比 BasicBlock 更深，用 1x1 降维 → 3x3 卷积 → 1x1 升维，这种结构能减少计算量，适用于深层网络。

✓ **ResNet 主体结构：**

```
self.layer1 = self._make_layer(block, 64, num_layer[0])
```

按照论文设置每层残差模块数量（如 ResNet50 为 [3, 4, 6, 3]），初始卷积为 7x7 → 最大池化 → 四层残差模块 → 平均池化。

✓ **forward() 函数：**

```
x = self.avgpool(x)
x = x.view(x.size(0), -1)
return x
```

返回池化后的特征，不经过 fc 层（为后续自定义分类器保留灵活性）。

✓ **多个模型构造函数：**

```
def resnet50(pretrained=False, **kwargs):
    model = ResNet(Bottleneck, [3, 4, 6, 3], **kwargs)
```

模拟 torchvision 接口，支持预训练与不预训练。

（2）classifier.py：分类器模型封装

①核心功能：构建完整的图像分类模型 = ResNet 主干 + 自定义 MLP 分类头

②关键代码详解：

✓ **模型定义：**

```
self.backbone = getattr(model, args.backbone)(pretrained=False)
```

根据 args.backbone 字符串动态加载模型（如 resnet50）。

✓ **分类器结构：**

```
self.classifier = nn.Sequential()
self.classifier.add_module('fc1', nn.Linear(2048, self.hidden_dim))
...
self.classifier.add_module('fc2', nn.Linear(self.hidden_dim, self.class_num))
```

使用两层全连接网络：2048 → hidden_dim → class_num，中间使用了 BatchNorm、ReLU、Dropout，提高泛化能力。

✓ **前向传播：**

```
feature_map = self.backbone(x)
output = self.classifier(feature_map)
```

将图像送入主干网络提取特征，再由 MLP 分类器输出预测类别。

（3）dataloader.py：数据加载与划分

①核心功能：加载数据集并按类别进行划分（60% 训练、20% 验证、20% 测试），对不同子集应用不同的 Transform，封装为 DataLoader 返回

②关键代码详解：

✓ **加载数据：**

```
dataset = datasets.ImageFolder(root=os.path.join(root_path, dir), transform=base_transform)
```

使用 ImageFolder 按文件夹加载图像数据，初始只 Resize 到 256，用于后续裁剪。

✓ **数据划分：**

```
for label, indices in class_indices.items():
    np.random.shuffle(indices)
    ...
```

每个类别按比例划分，确保类别分布均衡，未设置随机种子，不利于实验复现。

✓ **应用 Transform：**

```
train_transform = transforms.Compose([...])
val_test_transform = transforms.Compose([...])
```

训练集使用随机裁剪 + 翻转增强，验证与测试集中心裁剪、标准归一化。

✓ **封装 ApplyTransform：**

自定义 Subset 类封装不同 Transform 应用于同一数据集，兼容 Tensor 与 PIL Image 数据类型。

（①） main.py：训练主程序

①核心功能：完整定义训练流程、验证流程、测试流程，启用学习率调度器、保存最优模型

②关键代码详解：

✓ **参数设定：**

```
parser.add_argument('--backbone', type=str, default='resnet50')
```

使用 argparse 定义命令行参数，便于控制训练细节。

✓ **数据准备：**

```
train_loader, val_loader, test_loader = dataloader.data_load(...)
```

✓ **模型初始化与优化器：**

```
model = Model4Classifier(args).to(DEVICE)
optimizer = torch.optim.Adam([...])
```

参数分别设置主干网络与分类器的学习率，使用 Adam 优化器，默认不含 momentum。

✓ **训练循环：**

```
for epoch in range(args.epochs):
    model.train()
```

```
3.      ...
```

标准训练流程：前向传播 → 损失计算 → 反向传播 → 更新参数，每轮后进行验证、记录准确率，保存最优模型。

✓ **学习率调度：**

```
1.  scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(...)
```

采用余弦退火调度策略。

✓ **模型保存：**

```
1.  torch.save(model.state_dict(), 'best_model.pth')
```

只保存模型参数，没有保存 epoch/optimizer，不能断点续训。

✓ **测试阶段：**

加载最优模型，在测试集评估最终性能，输出准确率与损失。

二、问题

✓ **任务目标：** 使用深度神经网络在数据集“Real World”上完成图像分类任务。

✓ **当前方法：** 基于 ResNet 提取特征 + MLP 分类器，在 PyTorch 框架下训练，划分训练/验证/测试集，采用交叉熵损失和 Adam 优化器。

（1）模型结构问题

①当前做法

使用 ResNet50 主干特征提取器（model.py），接一个两层的 MLP 作为分类器（classifier.py）

②问题分析

分类器结构浅，学习能力弱：当前 Linear(2048 → hidden_dim → class_num)，仅一层隐藏，模型表达能力受限，尤其面对图像间高类间相似性时。

③改进建议

使用更深的 MLP（如 2048→1024→hidden_dim→class_num）或引入注意力机制（如 SE Block 或 Transformer Head）

（2）数据处理问题

①当前做法

训练集增强包含 RandomResizedCrop 和 HorizontalFlip，验证/测试集只做中心裁剪和标准化，每类按 6:2:2 比例划分为 train/val/test

②问题分析

数据增强手段单一，泛化能力差：仅使用基础的裁剪与翻转，无法有效模拟图像在真实场景中的变化（如光照、色差、遮挡等）

划分随机但无种子，结果不可复现：使用 np.random.shuffle() 但无 np.random.seed()，每次运行分布不同，无法比较模型效果

③改进建议

增加数据增强方式，如：transforms.ColorJitter、RandomErasing、CutMix、MixUp，在 np.random.shuffle() 之前加上 np.random.seed(args.seed)

统计每类样本数，计算加权 CrossEntropyLoss(weight=...)

（3）训练策略与调参问题

①当前做法

优化器为 Adam，学习率固定（支持 CosineAnnealingLR），训练 100 个 epoch，每轮验证保存最优模型

②问题分析

未使用 EarlyStopping：若模型在第 30 epoch 后性能已不再提升，其后 70 轮训练是无效甚至有害的，容易过拟合。

未记录训练过程曲线：不记录 loss/acc，调参缺少直观反馈

未冻结主干网络，训练不稳定：初始阶段直接训练全部网络，梯度剧烈波动，尤其在小数据集场景下不利于学习

③改进建议

引入 EarlyStopping 机制（基于验证集准确率），用 TensorBoard 或 Matplotlib 可视化训练曲线，初始阶段冻结 backbone，仅训练分类器，5~10 轮后再解冻 backbone 微调

（4）模型保存与复现问题

①当前做法

每轮保存验证精度最好的模型（.pth 文件）

②问题分析

未保存 optimizer、epoch、loss 状态：无法中断恢复训练，或做后续 fine-tune

训练不可复现（随机性高）

③改进建议

使用标准 checkpoint 格式保存更多信息：

```
torch.save({
    'epoch': epoch,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    ...
}, 'checkpoint.pth')
```

（5）模型评估问题

①当前做法

在 test_loader 上评估最终准确率

②问题分析

仅报告 Accuracy，忽视精度/召回/F1 等重要指标：在类别不平衡的任务中，准确率可能误导模型好坏

缺乏混淆矩阵、错误样本分析等可解释性手段

③改进建议

引入 sklearn.metrics，计算：

```
classification_report(y_true, y_pred, target_names=class_names)
confusion_matrix(y_true, y_pred)
```

绘制热力图帮助识别哪些类别易混淆

三、改进思路

（1）模型结构问题

问题 1：分类器结构过浅

目标：增强模型的非线性建模能力，更好地划分复杂类别边界。
改进思路：增加隐藏层

```
self.classifier = nn.Sequential(
    nn.Linear(2048, 1024),
    nn.BatchNorm1d(1024),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(1024, self.hidden_dim),
    nn.GELU(),
    nn.Linear(self.hidden_dim, self.class_num)
)
```

尝试更先进的激活函数如 GELU, Swish。

（2）数据处理问题

问题 2：增强手段单一

目标：增加数据的多样性，提高模型鲁棒性和泛化能力。
改进思路：增加训练数据增强策略：

```
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(0.4, 0.4, 0.4, 0.2),
    transforms.RandomGrayscale(p=0.1),
    transforms.RandomErasing(p=0.3),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])
```

问题 3：未设置随机种子

目标：保证训练结果可复现、划分一致。
改进思路：在任意涉及随机的部分添加种子控制：

```
import random
np.random.seed(args.seed)
torch.manual_seed(args.seed)
```

```
random.seed(args.seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

（3）训练策略问题

问题 4：无 EarlyStopping

目标：避免训练过程过拟合，提高资源利用率。

改进思路：创建 EarlyStopping 类并监控验证准确率：

```
if val_acc > best_val_acc:
    best_val_acc = val_acc
    counter = 0
else:
    counter += 1
if counter >= patience:
    print("Early stopping triggered")
break
```

问题 5：训练过程中未冻结 backbone

目标：稳定模型训练，减少干扰主干已学到的知识。

改进思路：第一阶段仅训练分类器，后续 N 轮后再解冻：

```
for param in model.backbone.parameters():
param.requires_grad = False
if epoch == 10:
    for param in model.backbone.parameters():
        param.requires_grad = True
```

问题 6：无训练可视化与调参辅助

目标：通过训练/验证曲线判断模型状态

改进思路：接入 TensorBoard：

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
writer.add_scalar("Loss/train", train_loss, epoch)
writer.add_scalar("Acc/val", val_acc, epoch)
```

（4）模型保存与测试评估问题

问题 7：模型保存信息不足（仅 state_dict）

目标：支持断点训练、后续微调。

改进思路：保存完整 checkpoint：

```
torch.save({
    'epoch': epoch,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'val_acc': val_acc
}, 'checkpoint.pth')
```

问题 8：测试指标单一，仅 Accuracy

目标：更全面衡量模型性能，发现易混类别。

改进思路：使用 sklearn，针对每类输出 Precision / Recall / F1-score，可用于优化类别权重

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_true, y_pred))
sns.heatmap(confusion_matrix(y_true, y_pred), annot=True)
```

问题 & 详细改进一览表

问题编号	问题描述	改进目标	技术方案
1	分类器过浅	增强非线性表达能力	增加多层 FC，替换激活函数
2	增强手段少	提高鲁棒性	增加 ColorJitter、Erasing、Mixup 等
3	随机划分未设种子	保证复现性	设置 numpy、torch、random 等随机种子
4	无 EarlyStopping	节省训练资源	验证集准确率无提升时提前停止

5	没冻结主干网络	提高初始训练稳定性	先冻结 backbone，后期再解冻
6	训练过程不可视	便于调参与诊断	使用 TensorBoard 实时绘制 loss/acc 曲线
7	模型保存信息不全	支持断点训练	保存 optimizer、epoch、val_acc 等完整 checkpoint
8	测试评估指标单一	提高可解释性	添加分类报告、混淆矩阵