1 实验原理

1.1 训练集图片预处理

首先对训练集进行数据增强和标准化处理的，目的是提升模型的泛化能力，防止过拟合，涉及的内容如下

- 随机裁剪图像的一部分: 让模型学会在图像不同局部特征下也能做出正确判断
- 以 50% 的概率水平翻转图像：增加数据多样性，适应物体在不同方向上的外观，尤其是左右对称或者非对称但常见方向变化的物体。
- 随机将图像进行旋转: 提升模型对图像轻微旋转变化的适应能力，比如拍摄角度略有不同的情况。
- 随机调整图像的亮度、对比度和饱和度: 增强模型对光照变化和色彩变化的鲁棒性，比如不同拍摄环境、不同天气条件。
- 给定的均值和标准差对每个通道（RGB）进行标准化，使得数据均值为 0，标准差为 1: 加快收敛速度，提高训练稳定性。
- 最终输入的图片转化为三维 Tensor 的维度为 (3,224,224)

而对于验证集、测试集数据，不进行处理。

1.2 CNN 网络的构建

CNN 网络的结构如下:

卷积块	卷积核	输入通道	输出通道	池化	输入维度	输出维度
1	5*5	3	64	2*2	(3,224,224)	(64,112,112)
2	3*3	64	32	2*2	(64,112,112)	(32,56,56)
3	3*3	32	16	2*2	(32,56,56)	(16,28,28)

最后由 4 层全连接层映射到每个类别

1.3 学习率调度器

综合模型的参数数量，采用动态调度学习率的方式，训练早期使用高学习率，学习率随着训练轮数慢慢降低，具体为：
初始学习率为 5e-5，若连续五轮的验证集最佳准确率不改变，则学习率减少为之前的一半，最低的学习率为 1e-6。

1.4 早停机制

若连续 50 个 epoch 在验证集的最佳准确率无改善，则停止训练

1.5 损失函数和优化器

损失函数为 softmax 后的 CrossEntropyLoss，优化器为 Adam

2 关键代码展示

2.1 划分训练集和验证集

将验证集单独划出来并从训练集中删去

```
def split_train_val(train_dir, val_dir, split_ratio=0.15, seed=42):
    random.seed(seed)
    os.makedirs(val_dir, exist_ok=True)

    classes = os.listdir(train_dir)
    for cls in classes:
        cls_train_dir = os.path.join(train_dir, cls)
        cls_val_dir = os.path.join(val_dir, cls)

        if not os.path.isdir(cls_train_dir):
            continue
        os.makedirs(cls_val_dir, exist_ok=True)

        images = [f for f in os.listdir(cls_train_dir) if f.
           endswith('.jpg')]

        # 随机选择15%的图片
        num_val = max(1, int(len(images) * split_ratio))
        val_images = random.sample(images, num_val)

        for img_name in val_images:
            src_path = os.path.join(cls_train_dir, img_name)
```

```
            dst_path = os.path.join(cls_val_dir, img_name)

            shutil.move(src_path, dst_path)

        print(f"类别 {cls}: 共{len(images)}张, 移动{num_val}张到验
            证集。")
```

2.2 数据处理

对训练集采用上述的图片处理，而验证集不进行裁减缩放等处理

```
# 训练集用的数据增强（增加模型泛化能力）
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),  # 随机裁
        剪并缩放
    transforms.RandomHorizontalFlip(p=0.5),  # 50%概率水平翻转
    transforms.RandomRotation(15),  # 随机旋转±15度
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation
        =0.2),  # 颜色抖动
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225],
    ),
])
# 验证/测试集的预处理（保持固定，不做随机变换）
val_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # 直接Resize到224
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225],
    ),
])
```

```
script_dir = os.path.dirname(os.path.abspath(__file__))
train_dir = os.path.join(script_dir, "image", "train")
val_dir = os.path.join(script_dir, "image", "val")
train_dataset=datasets.ImageFolder(
    root=train_dir,
    transform=train_transform
)
val_dataset=datasets.ImageFolder(
    root=val_dir,
    transform=val_transform
)
train_loader=DataLoader(train_dataset,batch_size=32,shuffle=True)
val_loader=DataLoader(val_dataset,batch_size=1,shuffle=False)
```

2.3 模型构建

```
    class CNN(nn.Module):
    def __init__(self, num_classes=5):  # 5 类
        super(CNN, self).__init__()

        self.conv1 = nn.Conv2d(3,64,5, padding=2)
        self.bn1 = nn.BatchNorm2d(64)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(64, 32, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(32)

        self.conv3 = nn.Conv2d(32, 16, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(16)

        self.fc1 = nn.Linear(16 * 28 * 28, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
```

```
        self.fc4 = nn.Linear(64, num_classes)
    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))

        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))

        x = x.view(x.size(0), -1)  # Flatten
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
    return x
model = CNN()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu
    ")
model = model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=5e-5)
# 添加学习率调度器
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode='max',
    factor=0.5,
    patience=5,
    verbose=True,
    min_lr=1e-6
)
```

2.4 训练和验证

保存在验证集中准确率最高的模型 (泛化性能最好)，并以此进行早停机制

```
def train(epochs):
```

```
train_history_loss=[]
val_history_loss=[]
best_acc=0
stop_count=0
for epoch in range(epochs):
    epoch_loss = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        model.train()
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        epoch_loss += loss.item()
        optimizer.step()
        if batch_idx % 10 == 0:
            print(f"Epoch {epoch} [{batch_idx}/{len(
                train_loader)}] Loss: {loss.item():.4f}")
    print(f"Epoch {epoch} Loss: {epoch_loss/len(train_loader)
        :.4f}")

    # 评估模型并更新学习率
    acc, loss = evaluate()
    scheduler.step(acc)  # 根据验证集准确率调整学习率

    if acc > best_acc:
        best_acc = acc
        torch.save(model.state_dict(), 'best_model.pth')
        stop_count = 0
    else:
        stop_count += 1
    val_history_loss.append(loss)
    train_history_loss.append(epoch_loss/len(train_loader))
```

```
        # 打印当前学习率
        current_lr = optimizer.param_groups[0]['lr']
        print(f"Current learning rate: {current_lr:.2e}")

        if stop_count >= 50:
            print(f"Early stopping at epoch {epoch}")
            break
    return train_history_loss,val_history_loss
def evaluate():
    model.eval()
    with torch.no_grad():
        correct = 0
        loss=0
        total = 0
        for data, target in val_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            _, predicted = torch.max(output.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()
            loss+=criterion(output,target).item()
    print(f"Accuracy of the model on the {total} val images: {100 *
        correct / total:.2f}%")
    return 100 * correct / total,loss/total
```

2.5 测试

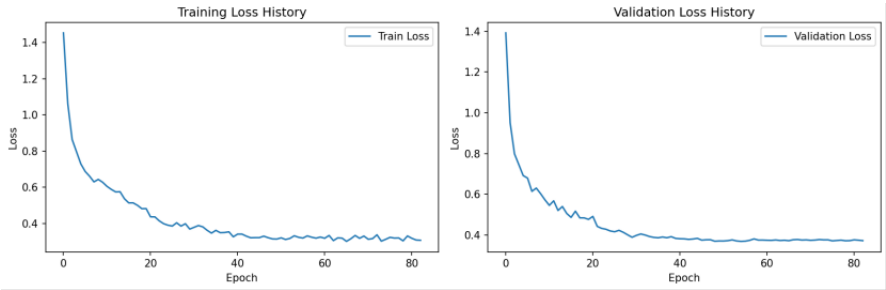
在测试集上测试, 其中 idx_to_class 是序号到类名的映射:

```
def infer(img_path,idx_to_class):
    if os.path.exists('best_model.pth'):
        model.load_state_dict(torch.load('best_model.pth'))
    model.eval()
    with torch.no_grad():
```

```
        img = Image.open(img_path)
        img = val_transform(img).unsqueeze(0)
        img = img.to(device)
        output = model(img)
        _, predicted = torch.max(output.data, 1)
        return idx_to_class[predicted.item()]
def test(idx_to_class):
    script_dir = os.path.dirname(os.path.abspath(__file__))
    test_dir = os.path.join(script_dir, "image", "test")
    for root, dirs, files in os.walk(test_dir):
        for file in files:
            file_path=os.path.join(root,file)
            predicted_class=infer(file_path,idx_to_class)
            print(f'file:{file} predict_class:{predicted_class}')
```

3 实验结果展示

在训练了 83 个 epoch 之后发生了早停，loss 图如下:



泛化性能最佳的模型在训练集和验证集的准确率如下:

```
在训练集上的准确率: 87.89%
correct:675,total:768
在验证集上的准确率: 88.06%
correct:118,total:134
```

此模型是在第 47 个 epoch 得到的

```
Current learning rate: 3.13e-06
Epoch 47 [0/24] Loss: 0.2466
Epoch 47 [10/24] Loss: 0.5006
Epoch 47 [20/24] Loss: 0.4784
Epoch 47 Loss: 0.3215
Accuracy of the model on the 134 val images: 88.06%
```

在测试集的测试结果如下:

```
file:baihe01.jpg predict_class:baihe
file:baihe02.jpg predict_class:baihe
file:dangshen01.jpg predict_class:dangshen
file:dangshen02.jpg predict_class:dangshen
file:gouqi01.jpg predict_class:gouqi
file:gouqi02.jpg predict_class:gouqi
file:huaihua01.jpg predict_class:huaihua
file:huaihua02.jpg predict_class:huaihua
file:jinyinhua01.jpg predict_class:jinyinhua
file:jinyinhua02.jpg predict_class:jinyinhua
```

可以看到，完美通过测试集。作为对比，选取了最后一个 epoch 得到的模型:

```
在训练集上的准确率: 96.74%
correct:743,total:768
在验证集上的准确率: 82.84%
correct:111,total:134
```

可以发现，此模型在训练集的准确率超过了 95%，但泛化性能较差，此模型发生了严重的过拟合。
因此，泛化性能最佳的模型在训练集的准确率并非最高。

4 创新点 & 优化

- 采用动态学习率调度策略：使用 ReduceLROnPlateau 调度器，根据验证集准确率动态调整学习率。当验证集性能停滞时，通过降低学习率来帮助模型找到更优的局部最小值，提高模型收敛质量。
- 实现早停机制：设置了 50 轮验证集性能无提升的早停阈值，有效防止过拟合，同时节省计算资源。实验结果表明在第 83 轮时触发早停，成功避免了模型继续过拟合。
- 分层设计的 CNN 架构：采用三层递减通道数 (64→32→16) 的卷积层设计，配合 BatchNorm 和 ReLU 激活函数，在降低参数量的同时保持了良好的特征提取能力。
- 全面的数据增强策略：实现了包括随机裁剪、水平翻转、旋转和颜色抖动在内的多种数据增强方法，显著提升了模型的泛化能力。实验结果显示，最终模型在测试集上取得了 100% 的准确率。
- 模型保存策略优化：不是简单地保存最后一个 epoch 的模型，而是保存验证集性能最优的模型状态。实验证明这种策略是有效的，因为最后 epoch 的模型出现了严重过拟合（训练集准确率超 95% 但泛化性能差）。