

中山大學



《计算机视觉小组作业》

基于多神经网络与多标签ArcFaceLoss的Plant Pathology-2021解决方案与结果可视化

授课教师 : 陈俊周
小组成员 : 周宇平 王忠灿 张佩伦 李梓牧 吴振喆
日期 : 2024.12.9

基于多神经网络与多标签ArcFaceLoss的Plant Pathology-2021解决方案与结果可视化

- 1 概况
- 2 数据集介绍
 - 2.1 数据集概况
 - 2.2 类别说明
- 3 数据处理
 - 3.1 One-Hot编码操作
 - 3.2 数据增强策略
 - 3.3 不平衡类别补偿策略
- 4 评估指标说明
 - 4.1 1. Precision (精确率, P)
 - 4.2 2. Recall (召回率, R)
 - 4.3 3. F1 Score (F1 分数, F1)
 - 4.4 4. Average Precision (平均精度, AP)
 - 4.5 宏平均 (macro-all) 和微平均 (micro-all) 说明
 - 4.5.1 1. 宏平均 (Macro Average)
 - 4.5.2 2. 微平均 (Micro Average)
- 5 Efficient与EVA-CLIP的改进与可视化
 - 5.1 EfficientNet介绍
 - 5.2 EfficientNet改进
 - 5.3 EfficientNet的可视化实现
 - 5.4 EfficientNet模型可视化分析
 - 5.5 EfficientNet模型评估
 - 5.6 EVA-CLIP模型介绍
 - 5.7 EVA-CLIP模型改进
 - 5.8 EVA-CLIP的可视化实现
 - 5.9 EVA-CLIP模型评估
- 6 SwinTransformer的改进与可视化
 - 6.1 SwinTransssformer介绍
 - 6.2 ArcFaceLoss介绍与MultilLabelArcFaceLoss实现
 - 6.3 PSAattention介绍与PSAattention实现
 - 6.4 ArcPSASwinTransformer实现
 - 6.5 ArcPSASwinTransformer可视化实现
 - 6.6 ArcPSASwinTransformer模型评估
- 7 训练过程概述
 - 7.1 数据预处理与增强
 - 7.2 数据加载
 - 7.3 损失函数与优化器
 - 7.4 混合精度训练
 - 7.5 早停机制
 - 7.6 模型保存与加载
 - 7.7 验证机制
 - 7.8 训练完成
- 8 项目结构说明
 - 8.1 目录
 - 8.2 代码结构

9 ArcPSASwinTransFormer训练历史

10 训练输出

1 概况

本项目主要使用了三个基准模型，分别是

- EfficientNet
- Eva-CLIP
- SwinTransformer

因为本问题重点在于模型对细粒度的敏感性，我在前两个模型上进行了改进和可视化探索，对于可视化结果进行分析，参考模型架构，经过思考与调研后，最终决定把重心放在SwinTransformer模型上，基于SwinTransformer模型进行了诸多改进，得到了不错的效果。

ArcFaceLoss是很有效的面对细粒度问题的损失函数，但是只能用于单标签分类问题上，我修改了SwinTransformer的架构，修改了ArcFaceLoss，增加归一化的操作，修改了Margin的逻辑，将修改后的损失函数命名为MultiLabelArcFaceLoss。最终实现了在多标签分类问题上使用MultiLabelArcFaceLoss作为损失函数进行训练与测评（这一部分网上没有任何相关资料）。并且我引入了PSA注意力的模块，将PSA注意力模块注入到模型中，我将基于SwinTransformer模型修改得到的模型命名为ArcPSASwinTransformer模型。

在模型测评方面，我首先测评了基础的EfficientNet与EVA-CLIP模型，使用BCELoss作为损失函数，然后对EfficientNet和EVA-CLIP处理图像的过程进行了可视化，在分析后，我发现一些模型的问题，在EVA-CLIP模型中加入了STN模块来增强模型对细粒度的敏感性。而后我使用了比较先进的SwinTransformer模型，并且基于这个模型修改得到了ArcPSASwinTransforme模型，能够匹配MultiLabelArcFaceLoss的处理逻辑。我分别测试了多标签的SwinTransformer+FotalLoss，多标签的ArcPSASwinTransformer+MultiLabelArcFaceLoss，对最后的结果进行了可视化处理。

在数据处理方面，我对原始数据集进行了类别合并，并且实现了One-Hot编码，应用了数据增强的策略，抑制了类间不平衡现象，提高了数据集的质量。

在训练过程方面，我应用了梯度裁剪，早停，保存模型检查点与实时监测与最终可视化训练与评估参数，在这个多标签分类问题中，我以F1分数来作为模型效果评估的主要参数，根据F1分数来决定哪一批次的模型效果最好。通过这些操作，提升了训练的速度与精度，有效防止模型陷入局部最优解或者过拟合，

在评估过程方面，我应用了验证集的数据来为每一个类别根据F1分数来寻找最优阈值，并且引入了TTA的增强测试策略，在最后按照类别和指标的索引可视化了验证结果，在这个过程严密防止了数据泄露，同时应用验证集数据来进行超参数调整。

在测试方面，我应用最后得到的最优阈值，为每个类别和总体测试数据评估对应的P,R,F1,AP,,AUC以及Accuracy，最后应用折线图，热力图等进行结果可视化，能够全方位直观地展示模型的效果。完整的结果见代码中 `checkpoints` 部分，我也贴出训练历史在附录中。

在结果可视化方面，我基于GradCam对EfficientNet的Feature进行了可视化，能够直观感受模型在训练后关注图像的哪些部分，帮助我做后续的操作。同时，我参考timm库中的可视化代码，修改而后在我自己的项目中对Eva-CLIP以及SwinTransformer和ArcPSASwinTransformer进行了每一层features在训练图像上的可视化，能够直观感受模型在训练后注意力在图像哪些部分，方便后续分析。

本代码已开源在Github，链接https://github.com/Chenlun17/ArcFace_PSA_SwinTransformer

2 数据集介绍

2.1 数据集概况

本项目使用的Plant Pathology-2021包含12个类别，每个类别对应不同的叶片病害类型。然而，不同类别之间存在标签重复（例如，`scab`在多个类别中出现），这增加了模型训练的复杂性。因此，我对数据集进行了类别合并，将原有的12个类别转换为6个类别，并将其视为一个多标签分类问题。

2.2 类别说明

经过合并后的6个类别如下：

- scab
- healthy
- frog_eye_leaf_spot
- rust
- complex
- powdery_mildew

每个类别通过*One-Hot*编码进行标签表示，以适应多标签分类任务。

3 数据处理

3.1 One-Hot编码操作

One-Hot编码是一种将分类数据转换为二进制向量的技术。在多标签分类任务中，每个样本可以同时属于多个类别。**One-Hot**编码能够有效地表示这种多标签关系，使得模型能够处理每个类别的独立预测。

在本项目中，由于原始数据集包含12个类别，且存在标签重复（如scab在多个类别中出现），我将类别合并为6个类别，并将其转换为多标签分类问题。具体操作如下：

1. **读取原始标签文件**：使用 `pandas` 读取包含图像名称及其对应标签的 CSV 文件。
 2. **One-Hot编码**：使用 `MultiLabelBinarizer` 将标签列表转换为 One-Hot 编码向量。
 3. **合并数据**：将图像名称与编码后的标签向量合并，生成最终的标签文件
- 具体的代码实现可以参考项目中 `label.py` 文件，处理后生成三个csv文件，分别对应训练集，验证集与测试集存放于data文件夹中。

以下为One-Hot编码后的csv文件示例：

```
images,scab,healthy,frog_eye_leaf_spot,rust,complex,powdery_mildew
80bcfd9f60f06307.jpg,0,0,0,0,0,1
e062c0b63cd36f1b.jpg,0,0,0,0,0,1
df3b8800e2f892fa.jpg,0,0,0,0,1,0
cff02d717850c0b7.jpg,1,0,0,0,0,0
ac4fd4118a9e2df8.jpg,1,0,0,0,0,0
fd37c332c54d120d.jpg,1,0,0,0,0,0
84f81ac3c5ed6566.jpg,0,1,0,0,0,0
db9961a5d34242cf.jpg,0,0,1,0,0,0
cbe3ad8ca8898bb4.jpg,1,0,0,0,0,0
.....
```

可见标签列表已经被转换为 One-Hot 编码向量。

3.2 数据增强策略

在深度学习中，数据增强（Data Augmentation）是一种通过对训练数据进行随机变换来生成更多样本的方法。这不仅能够增加数据的多样性，减少过拟合，还能帮助模型更好地泛化到未见过的数据。在本项目中，由于叶片病害分类属于细粒度分类，类内差异大、类间差异小，数据增强显得尤为重要。

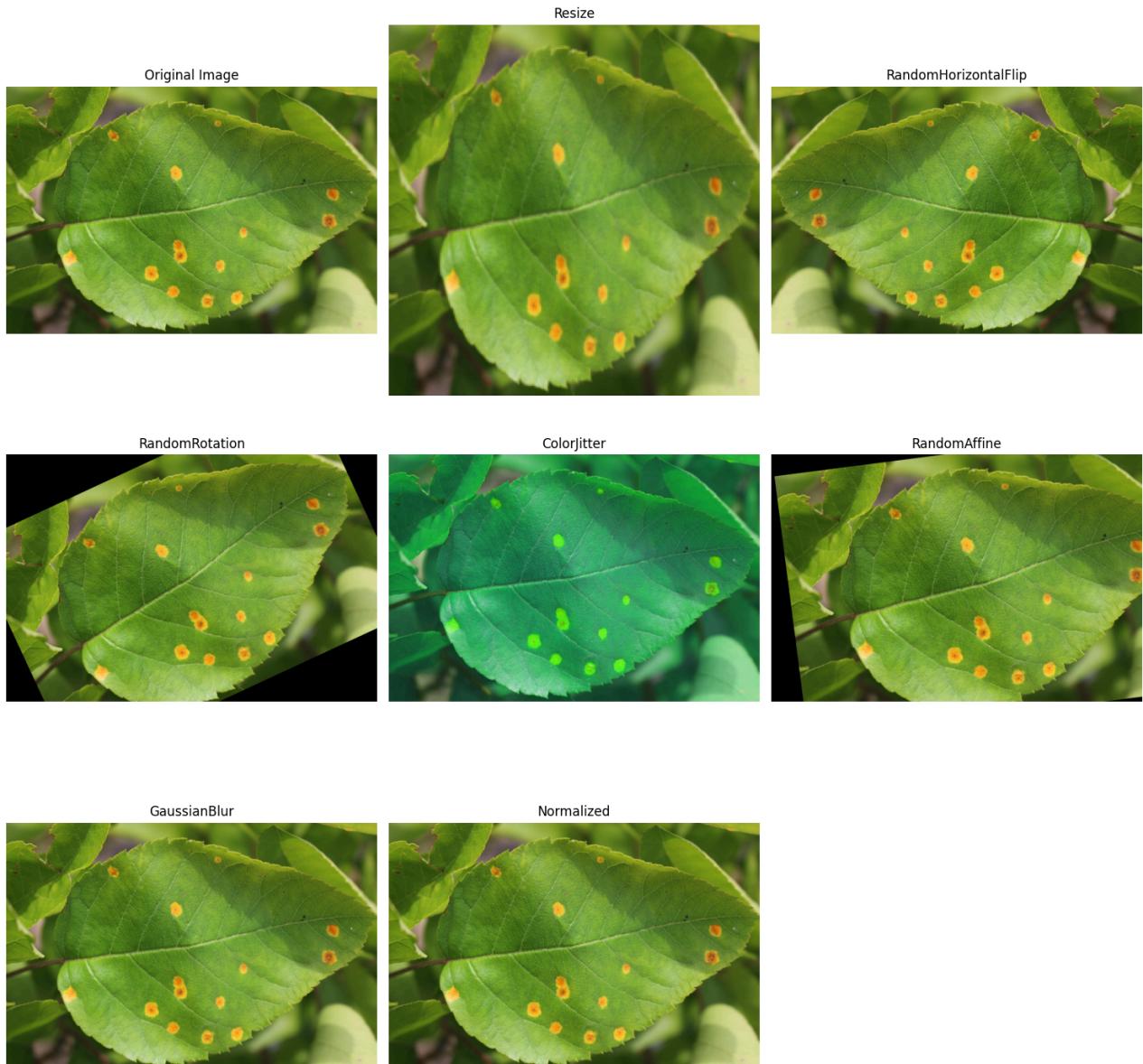
在本项目的 `train.py` 中，我对训练数据应用了多种数据增强技术，具体如下：

```
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(30),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    transforms.RandomAffine(degrees=10, translate=(0.1, 0.1), scale=(0.9, 1.1)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])
```

具体方法介绍

1. Resize：将所有图像调整为统一的尺寸（224x224），确保模型输入的一致性。
2. RandomHorizontalFlip：随机水平翻转图像。此方法可以模拟植物叶片在不同方向上的自然生长情况，增加模型对方向变化的鲁棒性。
3. RandomRotation：随机旋转图像最多30度。旋转变换能够使模型更好地识别不同角度下的叶片病害特征，提升模型的泛化能力。
4. ColorJitter：模拟不同光照环境下的数据。此方法可以使模型更好地识别不同光照下的叶片病害特征，提升模型的泛化能力。
5. RandomAffine：可以进行更广泛的空间变换，如平移、旋转和缩放。这样可以增加对视角、位置和尺度的鲁棒性。
6. ToTensor：将PIL图像或NumPy ndarray 转换为形状为 (C, H, W) 的张量，并且将像素值归一化到 [0, 1] 之间。
7. Normalize：因为选用的模型在ImageNet上进行预训练，使用 ImageNet 的均值和标准差对图像进行归一化处理，有助于加快训练收敛速度，提高模型性能。

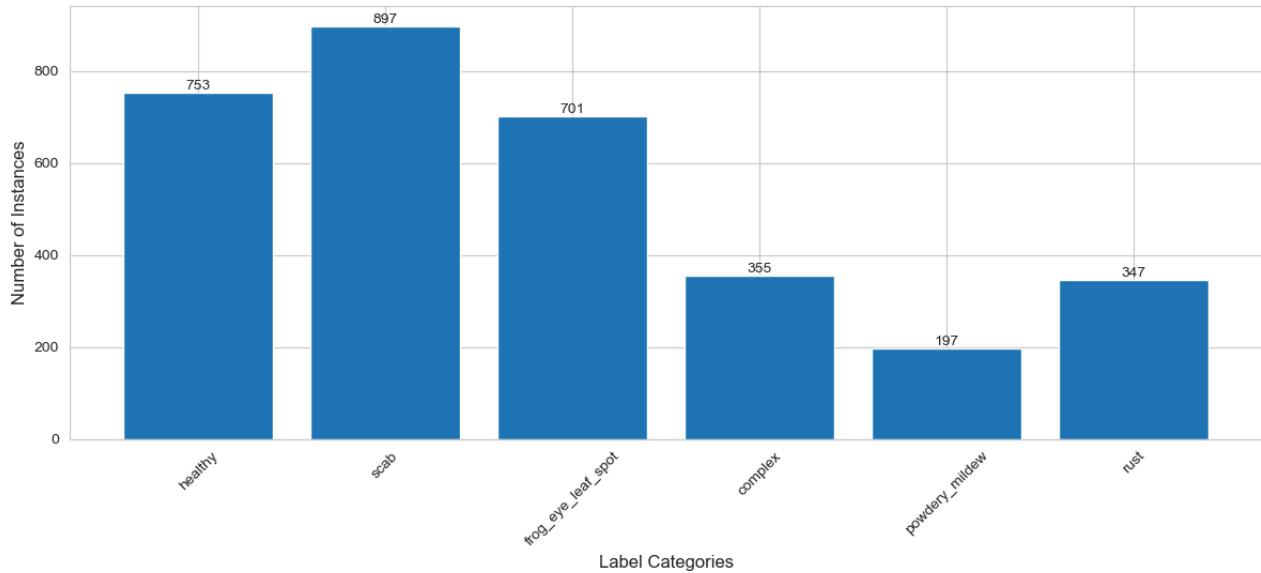
这些数据增强的操作显著提升了模型的泛化能力，并且强化了对细粒度特征的学习。效果示意如下：



3.3 不平衡类别补偿策略

我首先分析了各个类别数据的数量，得到如下结果：

Distribution of Labels in Dataset



可以发现在训练集中类别不均衡现象较严重，而且在初步试验与对苹果叶片病调研时发现 [Complex](#) 类很复杂，难以区分，于是我在训练前将 [complex](#) 类的数据复制了五份来进行训练，目的是加强模型对于该类别的学习强度。

4 评估指标说明

在此先说明我的评估指标

4.1 1. Precision (精确率, P)

定义:

精确率表示模型预测为正类的样本中，实际为正类的比例。换句话说，它衡量的是模型预测的准确性。

公式:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **TP (True Positive) :** 真正例，模型正确预测为正类的样本数。
- **FP (False Positive) :** 假正例，模型错误预测为正类的样本数。

意义:

高精确率意味着模型在预测为正类时，错误的概率较低，即模型的假正例较少。

4.2 2. Recall (召回率, R)

定义:

召回率表示实际为正类的样本中，模型正确预测为正类的比例。它衡量的是模型的覆盖能力。

公式:

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **FN (False Negative) :** 假负例，模型错误预测为负类的样本数。

意义:

高召回率意味着模型能够识别出大部分的正类样本，假负例较少。

4.3 3. F1 Score (F1 分数, F1)

定义:

F1 分数是精确率和召回率的调和平均值，综合考虑了两者的平衡。它是一个权衡精确率和召回率的指标。

公式:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

意义:

F1 分数在精确率和召回率之间寻找平衡，特别适用于类别不平衡的情况。

4.4 4. Average Precision (平均精度, AP)

定义:

平均精度是基于精确率-召回率曲线 (Precision-Recall Curve) 下的面积。它衡量的是模型在不同阈值下的整体性能。

公式:

$$AP = \int_0^1 P(r) dr$$

意义:

AP 提供了模型在所有可能的阈值下的性能概述，较高的 AP 表明模型在整体上表现良好。

4.5 宏平均 (macro-all) 和微平均 (micro-all) 说明

在多类别或多标签分类任务中，宏平均和微平均是两种常用的聚合方法，用于计算整体的评估指标。

4.5.1 1. 宏平均 (Macro Average)

定义:

宏平均是对每个类别的评估指标（如 Precision、Recall、F1、AP）分别计算后，再取这些指标的简单平均值。每个类别在计算时被赋予相同的权重。

计算方法:

- 对于每个类别，计算 Precision、Recall、F1 和 AP。
- 将所有类别的 Precision、Recall、F1 和 AP 分别取平均。

公式:

$$\text{Macro Average} = \frac{1}{N} \sum_{i=1}^N \text{Metric}_i$$

其中， N 是类别的总数， Metric_i 是第 i 个类别的某一指标。

意义:

宏平均强调各类别的独立表现，适用于类别数量较少且每个类别同等重要的情况。然而，对于类别不平衡的数据集，宏平均可能会被少数类别的表现所主导。

4.5.2 2. 微平均 (Micro Average)

定义：

微平均是将所有类别的 True Positives (TP) 、 False Positives (FP) 和 False Negatives (FN) 累加后，再计算总体的评估指标。各类别在计算时的权重与其样本数成正比。

计算方法：

- 将所有类别的 TP、FP 和 FN 累加。
- 使用累加后的 TP、FP 和 FN 计算 Precision、Recall 和 F1。

公式：

$$\text{Micro Average Precision} = \frac{\sum_{i=1}^N TP_i}{\sum_{i=1}^N (TP_i + FP_i)}$$

$$\text{Micro Average Recall} = \frac{\sum_{i=1}^N TP_i}{\sum_{i=1}^N (TP_i + FN_i)}$$

$$\text{Micro Average F1} = \frac{2 \times \text{Micro Precision} \times \text{Micro Recall}}{\text{Micro Precision} + \text{Micro Recall}}$$

意义：

微平均考虑了各类别的样本数量，对于类别不平衡的数据集，微平均更能反映整体性能。然而，它可能会掩盖少数类别的表现。

5 Efficient与EVA-CLIP的改进与可视化

这个模块我引入了两个模型，分别是 [EfficientNet](#)与[EVA-CLIP](#)。

5.1 EfficientNet介绍

EfficientNet源自Google Brain的论文EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. 从标题也可以看出，这篇论文最主要的创新点是Model Scaling. 论文提出了compound scaling，混合缩放，把网络缩放的三种方式：深度、宽度、分辨率，组合起来按照一定规则缩放，从而提高网络的效果。EfficientNet在网络变大时效果提升明显，把精度上限进一步提升。EfficientNet-B7在ImageNet上获得了先进的 84.4%的top-1精度 和 97.1%的top-5精度，比在EfficientNet之前最好的卷积网络（GPipe, Top-1: 84.3%, Top-5: 97.0%）大小缩小8.4倍、速度提升6.1倍。

EfficientNet的主要创新点并不是结构，不像ResNet、SENet发明了shortcut或attention机制，EfficientNet的base结构是利用结构搜索搜出来的，然后使用compound scaling规则放缩，得到一系列表现优异的网络：B0~B7. 下面得图是ImageNet的Top-1 Accuracy随参数量变化关系图，可以看到EfficientNet饱和值高，并且到达速度快。

增加网络参数可以获得更好的精度（有足够的数据，不过拟合的条件下），例如ResNet可以加深从ResNet-18到ResNet-200，GPipe将baseline模型放大四倍在ImageNet数据集上获得了84.3%的top-1精度。增加网络参数的方式有三种：深度、宽度和分辨率。

深度是指网络的层数，宽度指网络中卷积的channel数（例如wide resnet中通过增加channel数获得精度收益），分辨率是指通过网络输入大小（例如从112x112到224x224）

直观上来讲，这三种缩放方式并不独立。对于分辨率高的图像，应该用更深的网络，因为需要更大的感受野，同时也应该增加网络宽度来获得更细粒度的特征。

之前增加网络参数都是单独放大这三种方式中的一种，并没有同时调整，也没有调整方式的研究。EfficientNet使用了compound scaling方法，统一缩放网络深度、宽度和分辨率。

如下图所示，(a)为baseline网络，(b)、(c)、(d)为单独通过增加width, depth以及resolution使得网络变大的方式，(e)为compound scaling的方式。

EfficientNet中的base网络是和MNAS采用类似的方法（唯一区别在于目标从硬件延时改为了FLOPS），使用了compound scaling后，效果非常显著，在不同参数量和计算量都取得了多倍的提升。

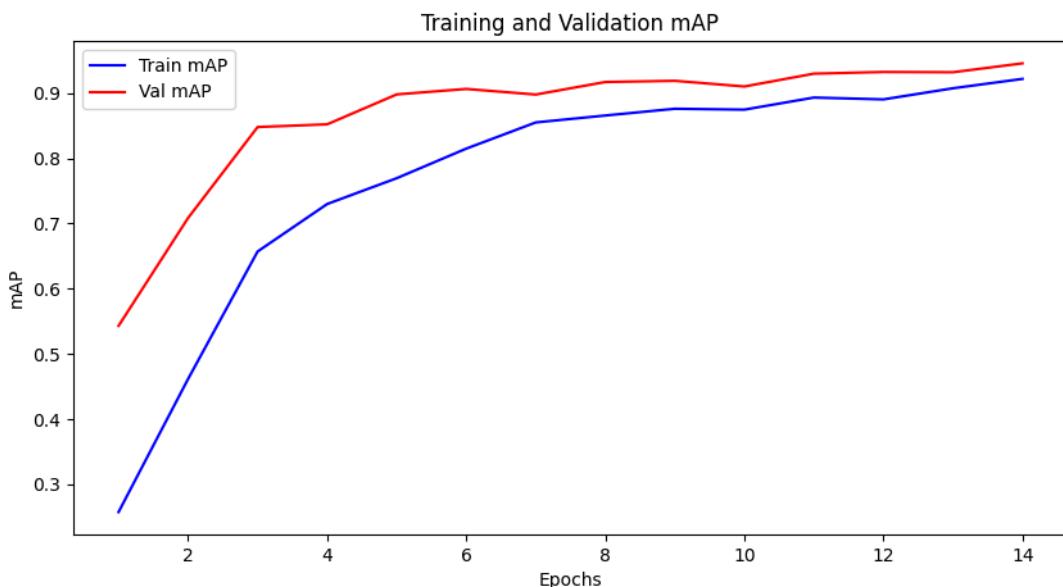
5.2 EfficientNet改进

我对于EfficientNet的改进主要在模型推理效率方面，参考了[How we made EfficientNet more efficient](#)的方法，我在原始的模型上引入了分组卷积，EfficientNet将深度卷积用于所有空间卷积操作。它们以其FLOP和参数量中的效率而闻名，因此已经成功地应用于许多最先进的CNN中。然而，在实践中，EfficientNet在加速方面遇到了若干挑战。

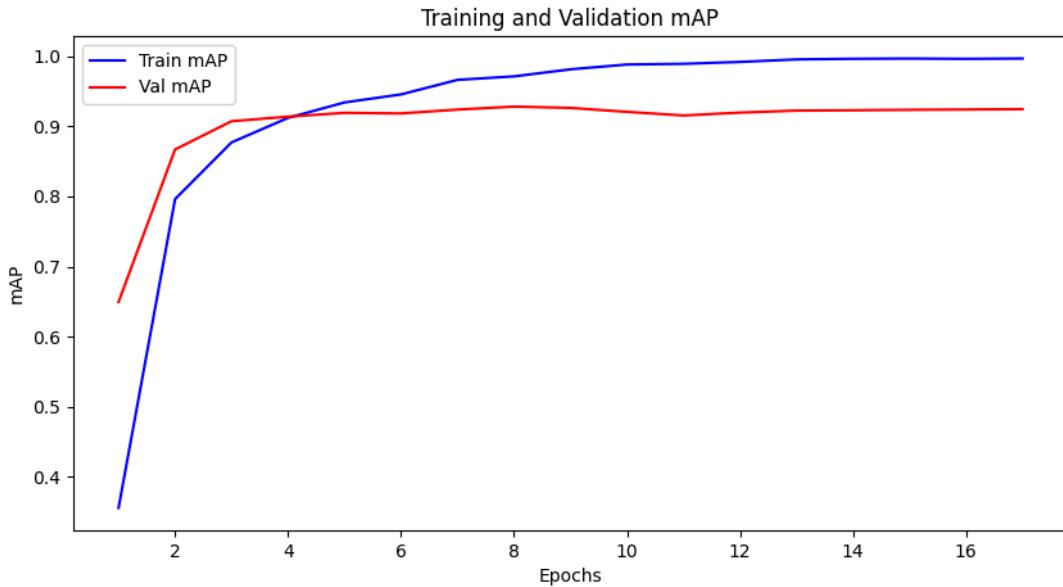
例如，由于每个空间核都需要独立考虑，因此，通常由向量乘积型硬件加速的点积操作的长度是有限的。这意味着硬件不能总是被充分利用，导致“浪费”了周期。

为了解决问题，我对MBConv块做了一个简单但重要的修改。在模型的config文件中进行修改，我将卷积的组大小从1增加到16；这将带来更好的IPU硬件利用率。然后，为了补偿FLOPs和参数的增加，并解决内存问题，我将扩展因子降低到4。这将带来一个更高效的内存和计算紧凑的EfficientNet。

这是修改前的mAP曲线：



这是修改后的mAP曲线：



可以看到虽然这些改变主要是由于吞吐量的提高，但我也发现，在所有模型大小上，它们使我能够实现比普通的组大小为1的基线模型更高的验证精度。这一修改导致了实际效率的显著提高。

5.3 EfficientNet的可视化实现

我应用了 Grad-CAM (Gradient-weighted Class Activation Mapping) 对 EfficientNet 进行可视化，EfficientNet 是一种基于卷积神经网络的架构，具有明确的卷积层结构，而Grad-CAM主要用于卷积层的特征图，所以我考虑应用 Grad-CAM 来对可视化进行实现。

GradCam的基本原理：

1. 目标层选择：

- 选择靠近输出层的卷积层作为目标层。
- 这些层包含高层次语义信息，同时保留空间位置信息。

2. 前向传播：

- 输入图像通过模型进行前向传播，得到预测结果（例如分类概率或得分）。

3. 梯度计算：

- 对于目标类别 (c)，计算其得分 (y^c) 相对于目标卷积层特征图 (A^k) 的梯度：

$$\frac{\partial y^c}{\partial A^k}$$

- 梯度反映了特征图中每个通道对目标类别的重要性。

4. 权重计算：

- 对每个通道的梯度进行全局平均池化，得到每个通道的权重 α_k^c ：

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{i,j}^k}$$

- 其中 (Z) 是特征图的像素数量。

5. 生成热力图：

- 使用权重 α_k^c 对目标卷积层的特征图进行加权求和：

$$L^c = \text{ReLU} \left(\sum_k \alpha_k^c A^k \right)$$

- ReLU 激活函数去除负值，仅保留对目标类别有积极贡献的区域。

6. 上采样与叠加：

- 将生成的热力图 (L^c) 上采样到与原始图像相同的尺寸。
- 与原始图像叠加，生成可视化结果。

代码实现

在加载模型后，我首先对图像进行预处理：

```
inputs = [Compose([Resize((224,224)), ToTensor(), image_net_preprocessing])(x).unsqueeze(0) for x in images]
inputs = [i.to(device) for i in inputs]
```

然后将Grad—CAM实例化：

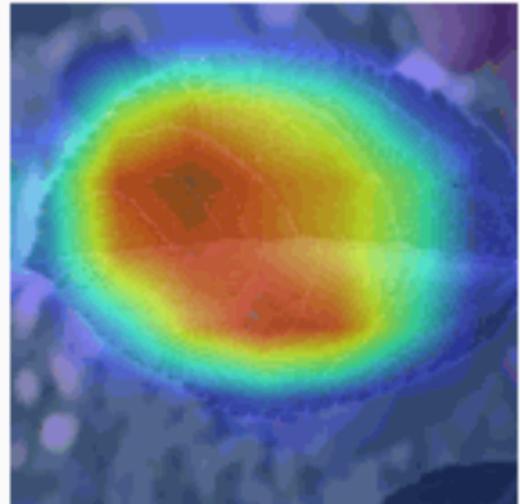
```
vis = GradCam(model, device)
target_layer = model._conv_head
cam_result = vis(inp, target_layer, postprocessing=image_net_postprocessing)[0]
cam_img = tensor2img(cam_result)
```

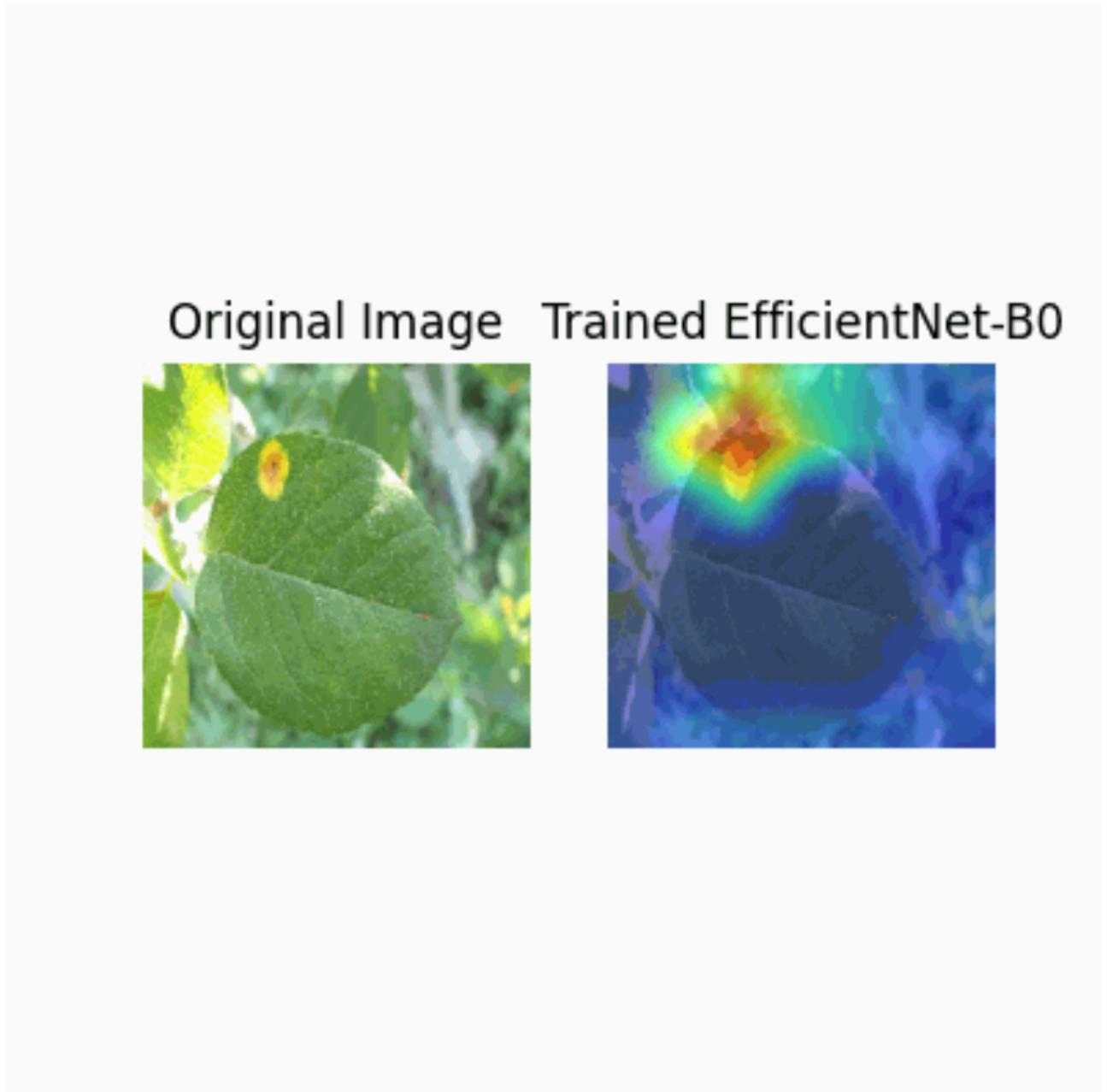
这样可以对每张预处理后的图像应用 Grad-CAM，生成热力图并转换为图像格式。

具体的代码实现在 [/src/Efficient_visualize.py](#) 中，最后可以得到的结果存放在 [outputs/example.gif](#) 中。

得到结果之二如下：

Original Image Trained EfficientNet-B0





5.4 EfficientNet模型可视化分析

可以看到优化后的EfficientNet模型对于特征的提取效果不错，即使在光照或图像噪声的干扰下仍然能准确的关注图像中的特征。

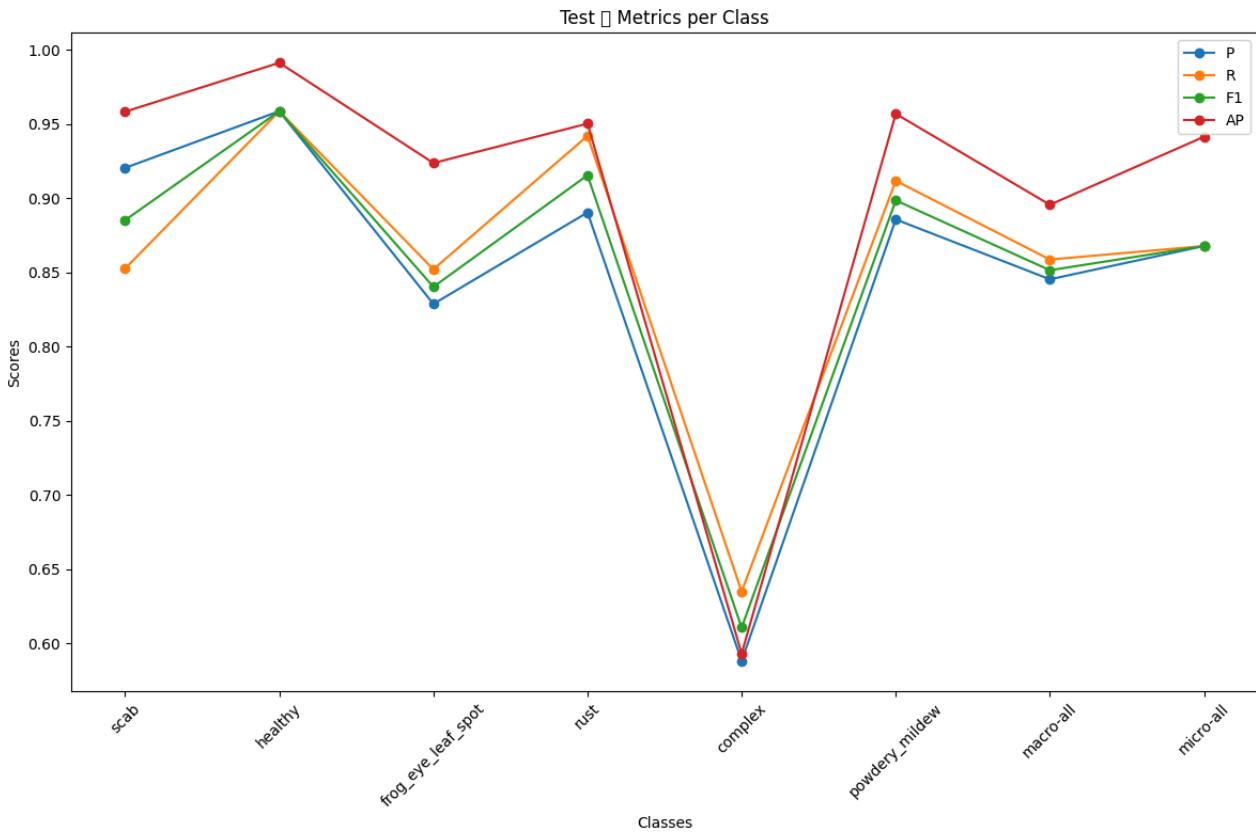
但是EfficientNet模型的可视化暴露出一个缺陷，最终热力图集中的区域过大，因为这是一个细粒度分类问题，通过一些在植物叶片病领域的调研，我发现植物叶片病种类之间差距很小，如果模型对于一个叶片的单一特征注意力过重，范围过大，有可能会忽视掉与关注区域比较相邻的部分，从而导致学习效果不好，或者最后分类时无法正确分类。

当图中没有明显的特征时，模型关注区域无法收敛到叶脉，而是专注于一个叶片上，可能会导致学习效果不好，最后分类时无法正确分类。

5.5 EfficientNet模型评估

最后我测试了改进后EfficientNet模型，先在验证集上找到了最佳的阈值，然后在测试集上进行了测试

EfficientNet模型在Test数据集上的指标：



Class	Threshold	P	R	F1	AP
scab	0.5500	0.9205	0.8526	0.8852	0.9583
healthy	0.5000	0.9586	0.9586	0.9586	0.9913
frog_eye_leaf_spot	0.5000	0.8288	0.8521	0.8403	0.9237
rust	0.5000	0.8904	0.9420	0.9155	0.9503
complex	0.4000	0.5882	0.6349	0.6107	0.5932
powdery_mildew	0.5500	0.8857	0.9118	0.8986	0.9569
macro-all	N/A	0.8454	0.8587	0.8515	0.8956
micro-all	N/A	0.8678	0.8678	0.8678	0.9414

可以看到各指标都处于一个“还不错”的水平，但是对于complex这种复杂类，学习效果比较差，在优化后我的模型在各个类的F1值上都有所提升，但是在complex类上提升很不明显。效果局限于模型的性能，我决定使用网络架构更大的模型。

5.6 EVA-CLIP模型介绍

EVA-CLIP-18B沿用了EVA系列weak-to-strong的视觉模型scale up策略，实现了视觉模型规模的渐进式扩增。该策略遵循“以小教大，以弱引强”的规模扩增思想。

具体而言，团队首先使用一个较小的EVA-CLIP-5B模型作为教师，以掩码图像建模为训练目标，蒸馏出一个较大的EVA-18B纯视觉模型。随后，EVA-18B作为EVA-CLIP-18B视觉-语言对比训练中视觉编码器的初始化，帮助EVA-CLIP模型进一步scale up。

研究结果表明，使用较弱的EVA-CLIP模型引导更大的EVA模型完成视觉训练，再使用得到的EVA模型作为更大EVA-CLIP训练的初始化，这种渐进式地用弱模型引导大模型的scale up方式，可以有效解决大型视觉模型训练中的不稳定问题，并加速模型训练收敛。

5.7 EVA-CLIP模型改进

为什么? 在原始的EVA-CLIP中已经有了注意力机制,对于在现实维度的细粒度分类问题,这个模型没有针对现实中的几何意义的增强,导致在一些比较复杂或者数据量较小的情况下,模型对于图像细节学习可能不到位。于是我思考引入模块来加强EVA-CLIP模型。

怎么做? 我在EVA-CLIP中引入了STN(空间变换网络)模块,自适应空间变换STN能够自动学习一个空间变换矩阵,对输入图像进行仿射变换(如旋转、缩放、平移等)。这种变换是可微的,可以通过反向传播来优化。这意味着模型可以自动调整要关注的区域的视角和尺度,使其更适合后续的特征提取。

意义是? 加入STN模块后,EVA-CLIP模型不仅能够决定“看什么”(传统注意力的功能),还能决定“如何看”(通过空间变换)。这对于捕捉细粒度特征特别重要。在空间变换后,图像中的关键区域会被调整到更适合特征提取的状态。这样一来,EVA-CLIP的视觉编码器就能够更有效地提取判别性特征。比如,在这个任务中,STN可以自动将叶片调整到一个标准朝向,便于后续的特征比较。

可行性? 端到端的学习STN是完全可微的,可以与EVA-CLIP模型无缝集成,进行端到端的训练。这意味着空间变换的参数会根据分类任务的目标来优化,形成一个协同优化的整体。

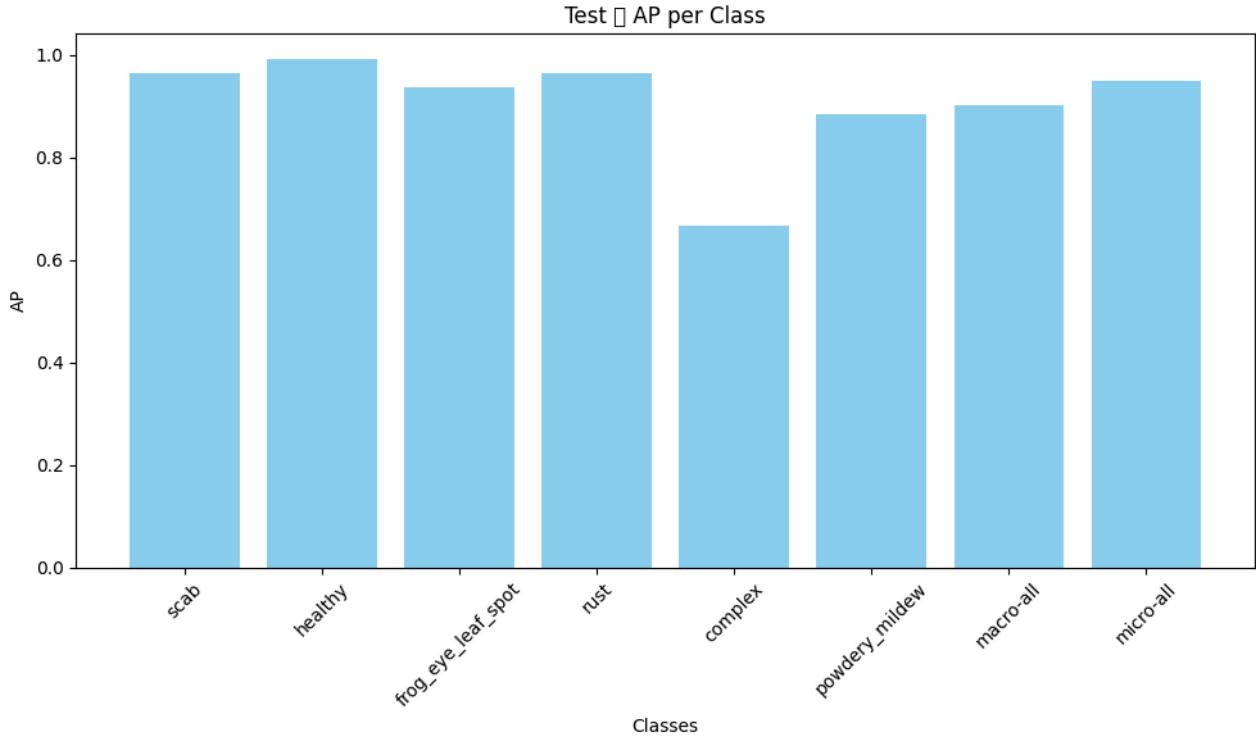
我在 `/src/enhanced_modules.py` 代码中具体有以下实现:

```
class STN(nn.Module):
    def __init__(self):
        super(STN, self).__init__()
        self.localization = nn.Sequential(
            nn.Conv2d(3, 8, kernel_size=7),
            nn.MaxPool2d(2, stride=2),
            nn.ReLU(True),
            nn.Conv2d(8, 10, kernel_size=5),
            nn.MaxPool2d(2, stride=2),
            nn.ReLU(True)
        )
        # 这里的输入维度调整为 10 * 52 * 52
        self.fc_loc = nn.Sequential(
            nn.Linear(10 * 52 * 52, 32), # 27040 是 10 * 52 * 52 的计算结果
            nn.ReLU(True),
            nn.Linear(32, 3 * 2)
        )
        self.fc_loc[2].weight.data.zero_()
        self.fc_loc[2].bias.data.copy_(torch.tensor([1, 0, 0, 0, 1, 0], dtype=torch.float))

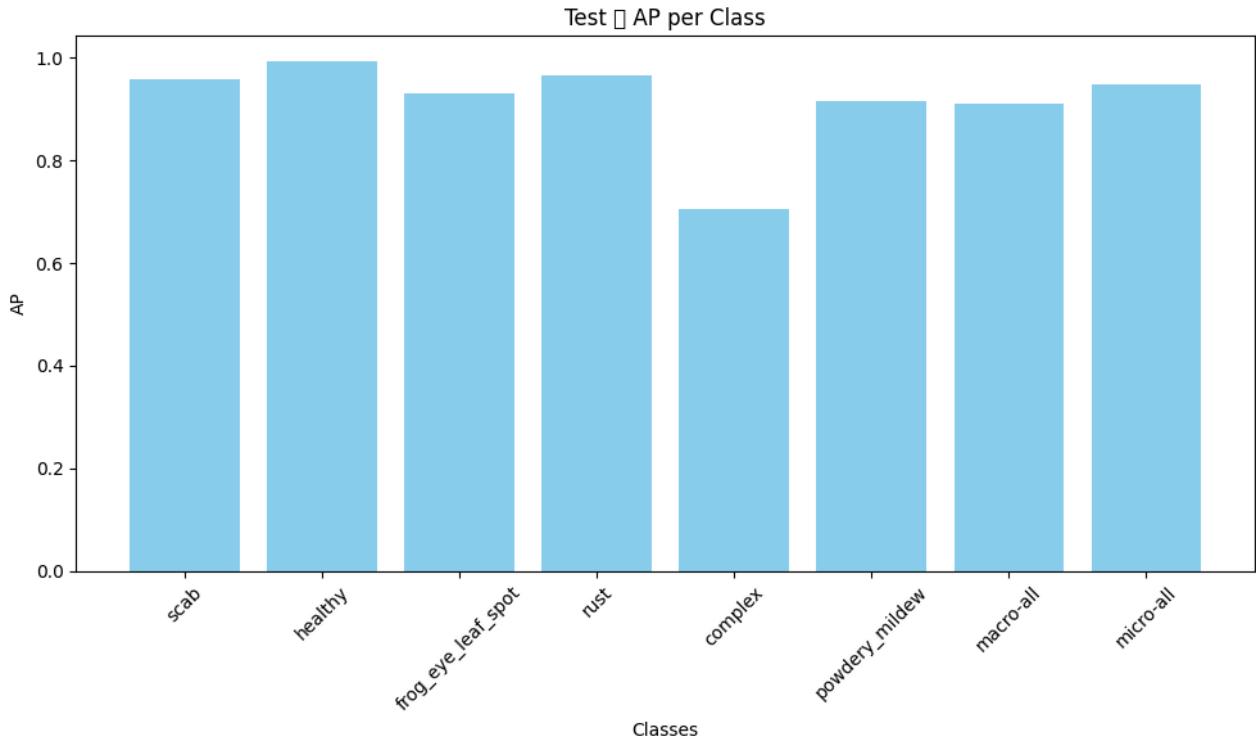
    def forward(self, x):
        xs = self.localization(x)
        x_size_lst = list(xs.size())
        total_size = x_size_lst[0] * x_size_lst[1] * x_size_lst[2] * x_size_lst[3]
        xs = xs.view(x_size_lst[0], total_size // x_size_lst[0]) # 展开成二维张量
        theta = self.fc_loc(xs)
        theta = theta.view(-1, 2, 3)
        grid = F.affine_grid(theta, x.size())
        x = F.grid_sample(x, grid)
        return x
```

在集成了STN模块后,我对模型进行重新训练和测试,得到了如下结果:

未加入STN模块的模型:



加入STN模块的模型：



可以看到在complex这个特征复杂的类中，加入了STN模块的模型提升显著，说明新增的STN模块通过提供可学习的空间变换能力，帮助模型更好地处理细粒度分类中的关键挑战，如视角变化和局部特征提取。

5.8 EVA-CLIP的可视化实现

EVA-CLIP模型依赖于注意力机制来捕捉图像中的全局依赖关系。注意力机制通过计算不同图像区域之间的关联性，决定模型在处理当前任务时应关注哪些部分。于是我引入了[AttentionExtract](#)，从模型中提取注意力图。具体来说，它利用torch.fx的符号跟踪功能，捕捉模型在前向传播过程中各层的注意力权重。这些权重反映了模型在处理图像时对不同区域的关注程度。提取了多层注意力图后，我参考了Ross Wightman的[timmAttentionViz](#)的实现方法，

使用 `rollout` 方法用于整合多层的注意力图，生成一个综合的注意力图。这种方法通过累积各层的注意力权重，生成一个最终的注意力分布，能够更好地捕捉模型对全局特征的关注。

具体的代码实现见 `src/attentionvis.py` 中

我对每一层都进行了可视化，分别保存为png图像，存放于 `/outputs/` 中，这些png图像最终拼接为一个gif图像，存放于 `/gifs` 文件夹中

可视化的结果如下：

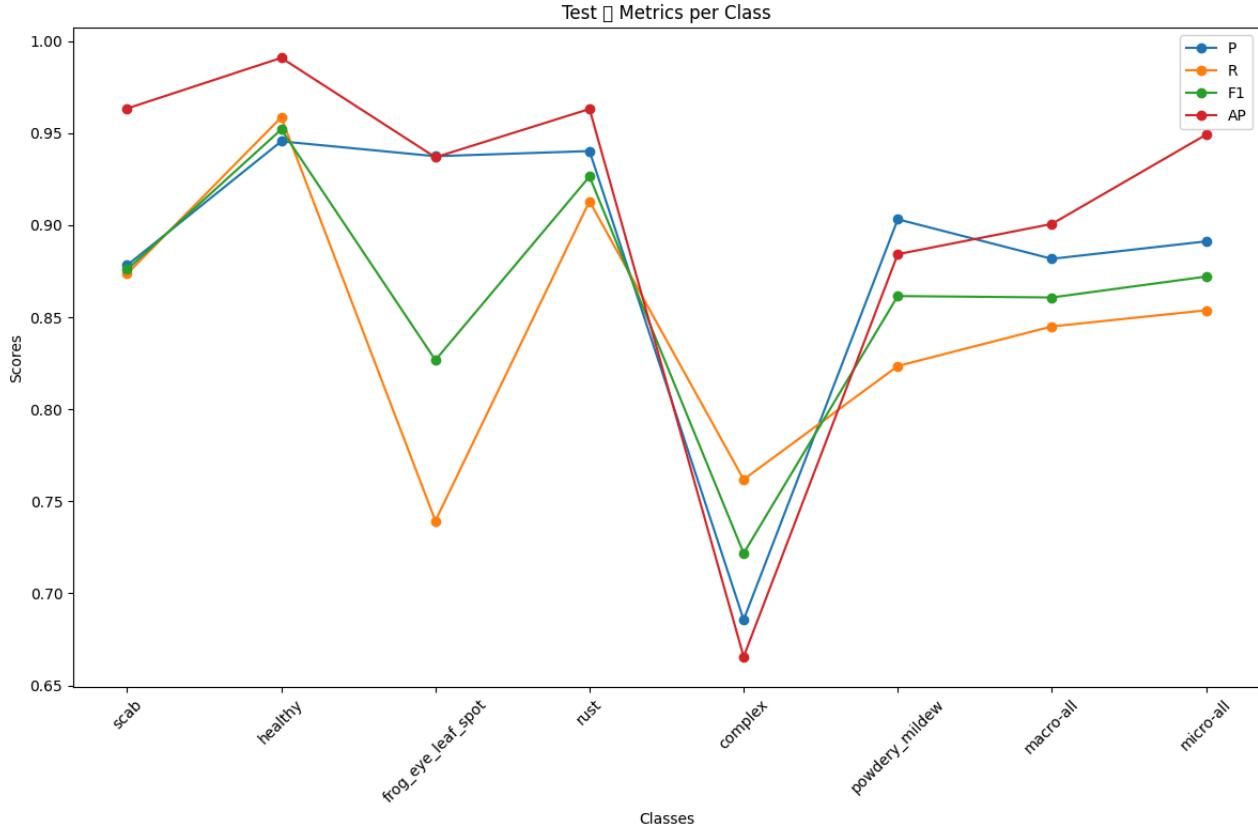


可以看到虽然效果是好了不少，但是仍然会受到环境光照的影响，比如第二张图的左上角，模型收到了叶片阴影的干扰，所以最后rollout图中有一部分注意力关注在阴影处，这也是可以改进的地方，思考如何最大程度的去除环境的噪声，在样本中没有明显关注点的情况下。

5.9 EVA-CLIP模型评估

最后我测试了改进后的EVA-CLIP模型，先在验证集上找到了最佳的阈值，然后在测试集上进行了测试

加入STN模块的EVA-CLIP模型在Test数据集上的指标：



Class	Threshold	P	R	F1	AP
scab	0.5000	0.8783	0.8737	0.8760	0.9634
healthy	0.6000	0.9456	0.9586	0.9521	0.9909
frog_eye_leaf_spot	0.6500	0.9375	0.7394	0.8268	0.9369
rust	0.5000	0.9403	0.9130	0.9265	0.9632
complex	0.3500	0.6857	0.7619	0.7218	0.6656
powdery_mildew	0.7500	0.9032	0.8235	0.8615	0.8843
macro-all	N/A	0.8818	0.8450	0.8608	0.9007
micro-all	N/A	0.8912	0.8538	0.8721	0.9492

可以看到在complex这一类上的F1分数有很多进步，一部分是因为模型庞大的结构，一部分是因为我加入的STN模块对模型捕捉细节空间变换有很好的效果，这一点在上一部分AP分数对比中很显著，但是EVA-CLIP模型庞大的结构要进行进一步的优化有些困难，我最终选择在更小一点的模型上进行核心实验。

6 SwinTransformer的改进与可视化

6.1 SwinTransssformer介绍

SwinTransformer是一个很强大的模型，对比以往的ViT，在结构上有两大核心改进：

1. 层次化特征图 (Hierarchical feature maps)：通过类似 CNN 那样的分层下采样 ($4\times$ 、 $8\times$ 、 $16\times$ 等)，获得多尺度特征，这对于下游任务（如检测、分割）非常必要，也便于与传统检测头（FPN 等）对接。
2. Window Multi-Head Self-Attention (W-MSA) 及 Shifted Windows (SW-MSA)：将较大分辨率的特征图分割成小窗口内自注意力 (W-MSA)，显著降低了计算量；再通过偏移窗口 (SW-MSA) 来实现跨窗口的信息

交互，保证局部与全局的兼顾。

这种分层+局部窗口的注意力方式在浅层（高分辨率）时能够降低计算量；同时，每个 Stage 的下采样又让后续层能够观察更大的空间范围，兼具局部和全局信息。

对于细粒度多标签分类而言，很多目标（或子类别特征）往往只在局部区域有所区别（如病斑细微花纹等），Swin 的窗口注意力也能更好地学习到这些差异化特征。

Swin 的一个特色是“图像先分块，然后在每个 Stage 做 Patch Merging”。Patch Merging 会将相邻 2×2 的特征拼合，再线性映射到通道数减半或增多，从而完成下采样、增强特征表达。这种分层方法让最终的 Stage 4 具有 $16 \times$ 的下采样倍数（对于 Swin-T 来说），非常适合后续分类或检测头。

以下是SwinTransformer的模型结构

6.2 ArcFaceLoss介绍与MultiLabelArcFaceLoss实现

ArcFace Loss是在人脸识别领域提出的一种角度型损失函数。它的核心思想是在特征空间中最大化类间距离，同时最小化类内距离。具体来说，ArcFace通过在原有的余弦相似度的基础上添加了一个角度间隔(angular margin)，使得不同类别之间的判别边界更加明确。

ArcFaceLoss的表达式为：

$$L = -\frac{1}{N} \sum_{i=1}^N \log \frac{e^{scos(\theta_{y_i}, i+m)}}{\exp scos(\theta_{y_i}, i+m) + \sum_{j=1, j \neq y_i}^n e^{scos(\theta_{j,i})}}$$

其中， θ 是特征向量与权重向量之间的角度， m 是添加的角度间隔， s 是特征的尺度因子。

ArcFace在细粒度问题有显著的优势：

ArcFace通过角度间隔（Margin），增强了特征的判别性。在细粒度分类中，不同类别之间的视觉差异往往很小，比如该任务中不同病种的叶子花纹。ArcFace通过在角度空间中增加额外的间隔，能够更好地区分这些相似但不同的类别。

ArcFace的归一化特性使得模型训练更加稳定。在细粒度分类中，由于类别之间差异微小，模型训练容易出现梯度爆炸或消失的问题。ArcFace通过对特征和权重进行L2归一化，很好地缓解了这个问题。

ArcFace的几何意义明确。它直接在超球面上优化角度分布，这种基于角度的度量方式比欧氏距离更适合处理细粒度特征的区分。

但是ArcFaceLoss只能用于单标签的分类问题，但是具有的这些特性很契合该任务，我先做了以下的分析：

可行性?：原有的ArcFace逻辑是在特征向量与分类权重之间注入角度Margin，从而让类间边界更大、类内更紧凑。抓住这个区分度的概念，可以对每一个正样本的标签都加margin，这样能让每个标签在向量空间里保持较高的区分度。

意义是?：当有多个相似的子类别时，ArcFace的角度惩罚能让特征朝向各自的类别中心，更易分开，能让每个标签在向量空间里保持较高的区分度。

问题有?：一个样本可能会激活多个类，那么特征在“多个类的中心”之间都要尽可能靠近，这就带来了一定的冲突，需要在margin选择上做平衡。

在思考了这些之后，可以着手于ArcFaceLoss的多标签版本实现，我把重新实现后的损失函数命名为 MultiLabelArcFaceLoss，具体实现如下：

```
class MultiLabelArcFaceLoss(nn.Module):
```

```
    """
```

多标签版本的 ArcFace Loss:

- 与单标签 ArcFace 不同之处在于: labels 为 multi-hot (batch_size x num_classes), 可能在同一样本中对多个类别同时为 positive。
- 对所有正类位置添加 margin, 即 $\phi = \cos(\theta) * \cos_m - \sin(\theta) * \sin_m$;
- 对负类保持原 $\cos(\theta)$ 。
- 使用 BCEWithLogitsLoss 作为多标签损失。

可选: easy_margin, 当 $\cos(\theta) < 0$ 时是否直接用 $\cos(\theta)$ 取代 ϕ 。

也保留了阈值 self.th 与修正项 self.mm 处理，以符合原生 arcface 对 large angle 的限制。

建议默认设置为False，实验表明False效果更好，估计是因为类别类别重叠没那么多，出现的特征重叠较少

```
"""
def __init__(self,
             in_features,
             num_classes,
             scale=30.0, #调优
             margin=0.5, #调优
             easy_margin=False):
    super(MultiLabelArcFaceLoss, self).__init__()
    self.in_features = in_features
    self.num_classes = num_classes
    self.s = scale
    self.m = margin
    self.easy_margin = easy_margin

    # 权重矩阵，形状 [num_classes, in_features]
    self.weight = nn.Parameter(torch.FloatTensor(num_classes, in_features))
    nn.init.xavier_uniform_(self.weight)

    # 预计算常量
    self.cos_m = math.cos(margin)
    self.sin_m = math.sin(margin)
    self.th = math.cos(math.pi - margin)
    self.mm = math.sin(math.pi - margin) * margin

def forward(self, features, labels):
    """
    Args:
        features: [B, in_features], 未必归一化；这里再行 L2 归一化
        labels: [B, num_classes], multi-hot (0/1)，表示每个类别是否为正类
    Returns:
        loss: (scalar) BCE 多标签损失
        logits: [B, num_classes], 做 sigmoid 前的 raw logits
    """
    # (1) L2 归一化
    features_norm = F.normalize(features, p=2, dim=1)      # [B, in_features]
    weight_norm = F.normalize(self.weight, p=2, dim=1)      # [num_classes, in_features]

    # (2) cos(theta) = features_norm • weight_norm
    #      形状: [B, num_classes]
    cosine = F.linear(features_norm, weight_norm)

    # (3) sin(theta) = sqrt(1 - cos^2)
    sine = torch.sqrt((1.0 - cosine**2).clamp(min=1e-9))

    # (4) phi = cos(theta + m) = cos(theta)*cos(m) - sin(theta)*sin(m)
    phi = cosine * self.cos_m - sine * self.sin_m

    # (5) 如果启用 easy_margin:
    #      当 cos(theta) < 0 时，直接用原 cos(theta)，否则用 phi
    #      (这在大角度时可能更稳定)
    if self.easy_margin:
        phi = torch.where(cosine > 0, phi, cosine)
    else:
        # 当 cos(theta) <= self.th (== cos(pi - m)) 时，用 (cosine - self.mm) 替换 phi
```

```

# 用于避免 cos(theta + m) 在 theta 较大时反而变正
phi = torch.where(cosine > self.th, phi, cosine - self.mm)

# (6) 用 labels 进行逐元素选择:
#      labels=1 的位置 => phi,  labels=0 => cos(theta)
#      这样就对所有正类加了 margin, 负类维持原 cos(theta)
outputs = torch.where(labels.bool(), phi, cosine)

# (7) 乘以 scale
outputs = outputs * self.s

# (8) 计算多标签 BCE Loss
loss = F.binary_cross_entropy_with_logits(outputs, labels)

return loss, outputs

```

6.3 PSAattention介绍与PSAattention实现

PSA (Polarized Self-Attention)机制是一种创新的注意力计算方法，其核心思想是通过两个相互补充的注意力分支来全面增强特征表示。第一个是空间注意力分支，关注特征图中不同空间位置的重要性，能够突出显示对分类任务重要的区域，而第二个是通道注意力分支，关注特征图中不同通道的重要性，增强对特定类型特征的响应，通过通道维度的权重分配来优化特征组合。

在这个任务中，我将PSA模块集成到了模型中，主要的思路是：

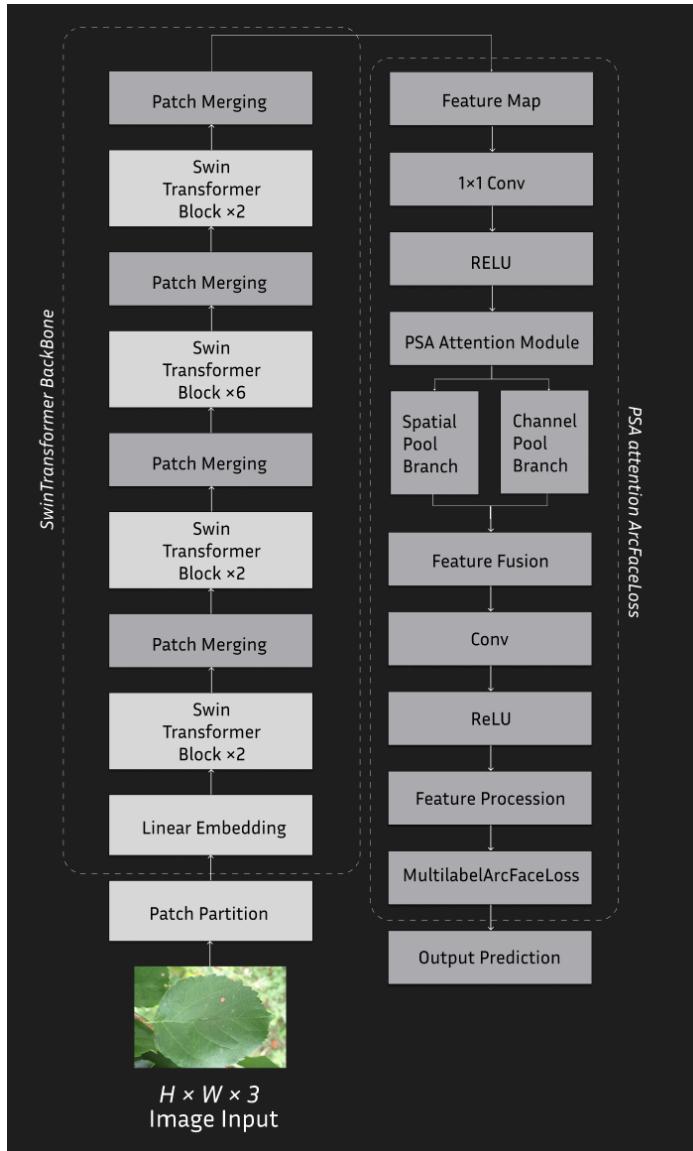
1. 进行特征提取

使用backbone网络提取基础的特征，然后在特征图中应用PSA模块进行注意力增强

2. 在特征处理阶段

通过PSA增强的特征进行进一步的处理，使用全局平均池化获取全局特征表示，通过全连接层降维得到最终的特征向量

模型架构如下：



具体实现如下：

```
class PSANFeatureModule(nn.Module):
    """整合PSA注意力机制的特征处理模块"""
    def __init__(self, in_channels, feature_dim=512):
        super(PSANFeatureModule, self).__init__()

        # First conv block
        self.conv1 = nn.Conv2d(in_channels, in_channels, kernel_size=1)
        self.bn1 = nn.BatchNorm2d(in_channels)
        self.relu1 = nn.PReLU()

        # PSA attention module
        self.psa = PSA_s(in_channels, in_channels)

        # Second conv block
        self.conv2 = nn.Conv2d(in_channels, in_channels, kernel_size=1)
        self.bn2 = nn.BatchNorm2d(in_channels)
        self.relu2 = nn.PReLU()

        # Feature reduction path
        self.feature_path = nn.Sequential(
            nn.AdaptiveAvgPool2d((1, 1)), # Global Average Pooling
            nn.Conv2d(in_channels, feature_dim, kernel_size=1),
            nn.BatchNorm2d(feature_dim),
            nn.PReLU(),
            nn.Linear(feature_dim, 1)
        )
```

```

        nn.Flatten(),
        nn.Linear(in_channels, 768),
        nn.BatchNorm1d(768),
        nn.PReLU(),
        nn.Linear(768, feature_dim),
        nn.BatchNorm1d(feature_dim),
        nn.PReLU(),
    )

    self._init_weights()

def _init_weights(self):
    """初始化模块的权重"""
    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
            if m.bias is not None:
                nn.init.zeros_(m.bias)
        elif isinstance(m, (nn.BatchNorm1d, nn.BatchNorm2d)):
            nn.init.ones_(m.weight)
            nn.init.zeros_(m.bias)
        elif isinstance(m, nn.Linear):
            nn.init.normal_(m.weight, 0, 0.01)
            if m.bias is not None:
                nn.init.zeros_(m.bias)

def forward(self, x):
    # First conv block
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu1(x)

    # PSA attention
    x = self.psa(x)

    # Second conv block
    x = self.conv2(x)
    x = self.bn2(x)
    x = self.relu2(x)

    # Feature reduction
    x = self.feature_path(x)

    return x

```

PSA的具体实现在目录的 `enhanced_modules.py` 中。

6.4 ArcPSASwinTransformer实现

基于我设计的MultiLabelArcFaceLoss，我改进了SwinTransformer的架构，主要思路是首先移除默认的Swin的分类头，在backbone后接一段特征处理函数，得到的特征向量 `[B,feature_dim]` 与可训练权重进行角度计算，再做多标签的BCE训练。具体实现与思路如下：

```

class PSANArcFaceModel(nn.Module):
    def __init__(self,

```

```

base_model,
num_classes,
feature_dim=512,
scale=30.0,
margin=0.5
):
    super(PSANArcFaceModel, self).__init__()

    # 提取backbone
    self.backbone = base_model.model
    self.in_features = self.backbone.num_features

    # 移除原始分类头
    if hasattr(self.backbone, 'head'):
        self.backbone.head = nn.Identity()

    # 替换为PSAN特征模块
    self.feature_module = PSANFeatureModule(1024, feature_dim)
    # print("PSA已注入!")
    self.feature_dim = feature_dim

    # 多标签ArcFace Loss
    self.arcface_loss = MultiLabelArcFaceLoss(
        in_features=feature_dim,
        num_classes=num_classes,
        scale=scale,
        margin=margin
    )

def forward(self, x, labels=None):
    """
    前向传播
    Args:
        x: 输入图像
        labels: 如果是训练阶段，提供标签(多标签矩阵)；推理阶段为None
    Returns:
        训练阶段: (loss, logits)
        推理阶段: features
    """
    # 1. 提取backbone特征
    features = self.backbone.forward_features(x)

    # 2. 调整维度顺序 [B, H, W, C] -> [B, C, H, W]
    features = features.permute(0, 3, 1, 2)

    # 3. 通过PSAN特征处理模块
    features = self.feature_module(features)

    # 4. 根据阶段返回不同结果
    if labels is None:
        return features
    else:
        loss, logits = self.arcface_loss(features, labels)
        return loss, logits

```

这样修改后，多标签训练时，网络只用特征来和损失函数的权重进行角度计算，模型的技术路线是：
图像->特征->loss，而具体的分类判断在MultiLabelArcFaceLoss内部实现。

6.5 ArcPSASwinTransformer可视化实现

在当前网上已有很多应用GradCam对SwinTransformer进行可视化的应用，对于SwinTransformer的可视化并不是一个难事，但是我的ArcPSASwinTransformer模型因为架构的改动，以及最后推理阶段只返回features而不返回logits，并不能直接应用这些方法，于是在实现可视化前，我先进行了以下的操作：

1. 对GradCam架构进行适应性的改进：

我设计了一个 `ArcFaceWrapper` 类，来包装模型：

```
class ArcFaceWrapper(nn.Module):  
    """  
    包装 ArcFace 模型以适应 GradCAM  
    """  
  
    def __init__(self, model):  
        super().__init__()  
        self.model = model  
        self.weight = model.arcface_loss.weight  
  
    def forward(self, x):  
        # 1. 获取 backbone 特征  
        features = self.model.backbone.forward_features(x)  
  
        # 2. 调整维度顺序  
        features = features.permute(0, 3, 1, 2)  
  
        # 3. 通过特征处理模块  
        features = self.model.feature_module(features)  
  
        # 4. 计算分类分数  
        features_norm = F.normalize(features, p=2, dim=1)  
        weight_norm = F.normalize(self.weight, p=2, dim=1)  
        return F.linear(features_norm, weight_norm)
```

这个包装器的创新之处在于它保持了 ArcFace 的核心计算流程，同时使其适配 GradCAM 的梯度计算要求。它在前向传播过程中复现了 ArcFace 的余弦相似度计算机制。

同时，这是多标签的场景，我引入了 `ClassifierOutputTarget`，来实现可视化多标签场景下模型同时关注图像中多个相关特征区域，具体实现如下：

```
# 5. 确定目标类别  
num_classes = model.arcface_loss.num_classes  
target_classes = target_classes or range(num_classes)  
  
# 6. 生成每个类别的 GradCAM  
for class_idx in target_classes:  
    # 计算 GradCAM  
    targets = [ClassifierOutputTarget(class_idx)]  
    grayscale_cam = cam(input_tensor=input_tensor, targets=targets)  
    grayscale_cam = grayscale_cam[0, :]
```

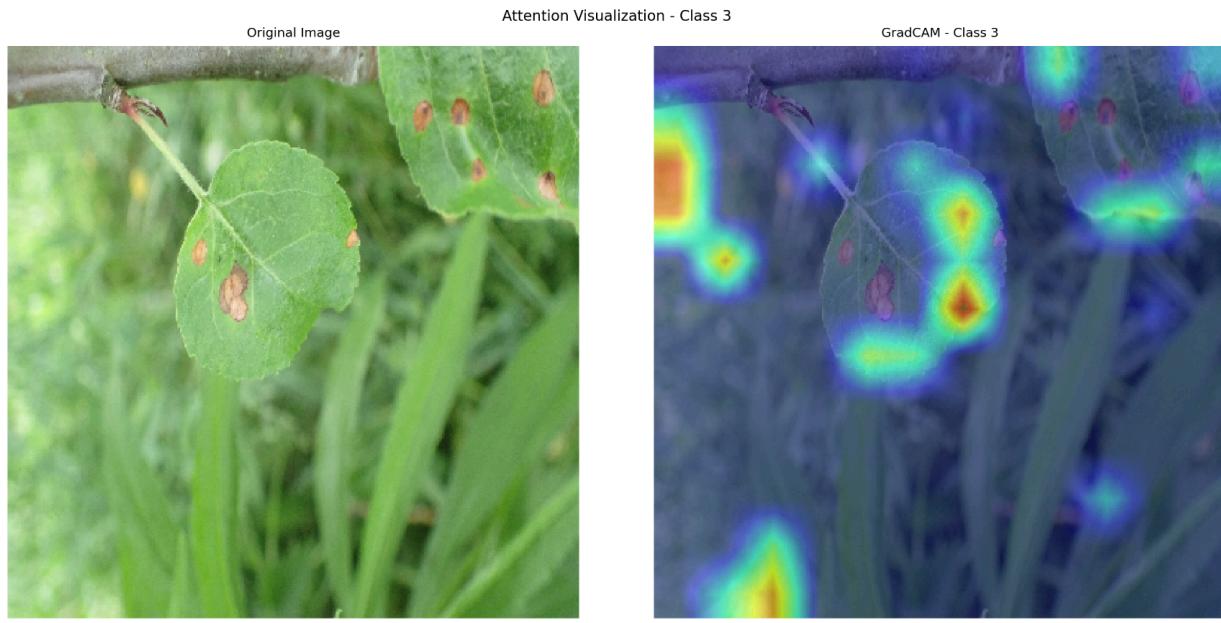
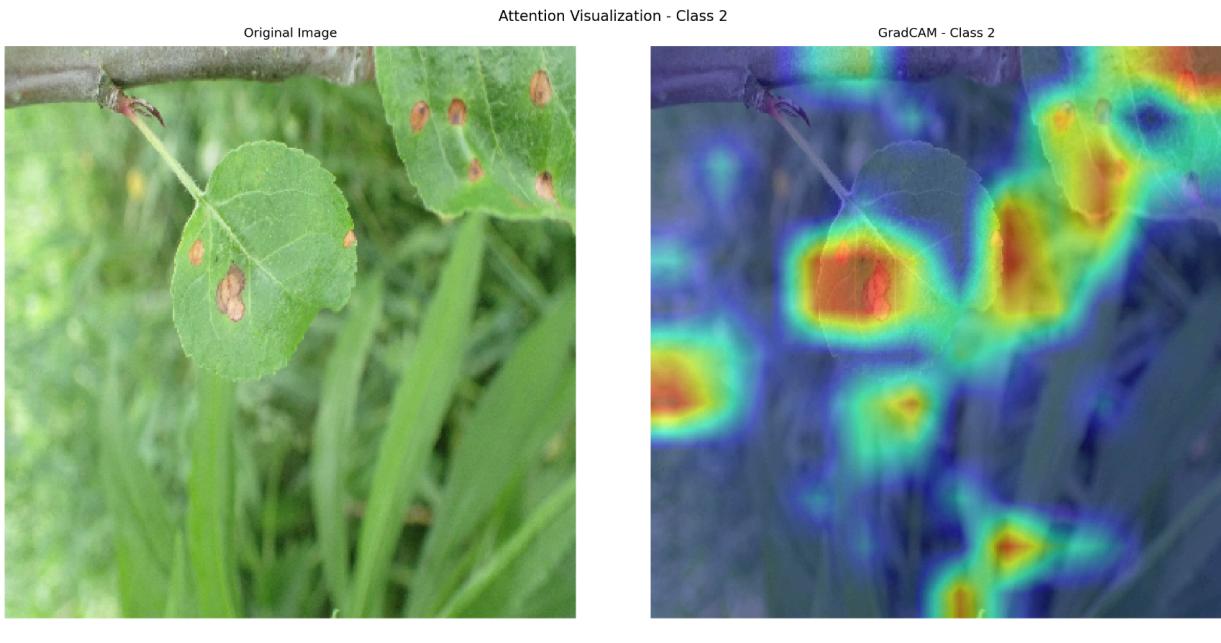
对每个类别单独进行注意力图的生成，保证了每个类别的特征关注点不会互相干扰，并且每个类别的注意力图是独立计算的，避免了多个类别的梯度相互叠加可能带来的混淆。

更重要的是，在这个可视化实现中，能更直观的关注到类间距这个概念，ArcFaceLoss中类别判断是基于特征向量与类别权重向量之间的角度关系，而通过为每个类别单独计算注意力图，保证了这种角度度量的语义在可视化中得到保留

最后我将这些图像拼接成一个gif图像，存放在[/outputs/gifs](#)中。

具体实现在[/src/arcviz.py](#)中。

可视化的结果如下：



可以看到对同一张图像但是不同类别进行的可视化，在一张图像中，我选取了[frog_eye_leaf_spot](#)类与[rust](#)类作为例子，能直观地看到在[healthy](#)类中，模型注意主要是平时[frog_eye_leaf_spot](#)类叶片中会出现斑点的叶片边缘以及该图上能明确标识叶片疾病的区域，通过关注这两个区域，模型能发现这个叶片在[frog_eye_leaf_spot](#)类中一般会出现斑点的区域并没有斑点而在不应该有斑点的区域存在斑点，从而能够轻易地分辨这个叶片并不属于[frog_eye_leaf_spot](#)这一类。而对于[rust](#)类来说，模型则关注叶片边缘是否存在连续的小斑点来判断该叶片是否属于该类，[rust](#)一类主要的特征是叶子上有一些圆形病斑，中心呈现棕色或褐色，周围有黄色晕圈，所以模型识别这一类为[rust](#)类。

同时分析这个可视化结果，也能看到类间距在结果上的体现，不同类别的可视化分析覆盖的区域很不一样，各自关注的重点在空间上分散，这样可以让模型在完成细粒度问题时更好的捕捉细节上的差异，提升模型的性能。

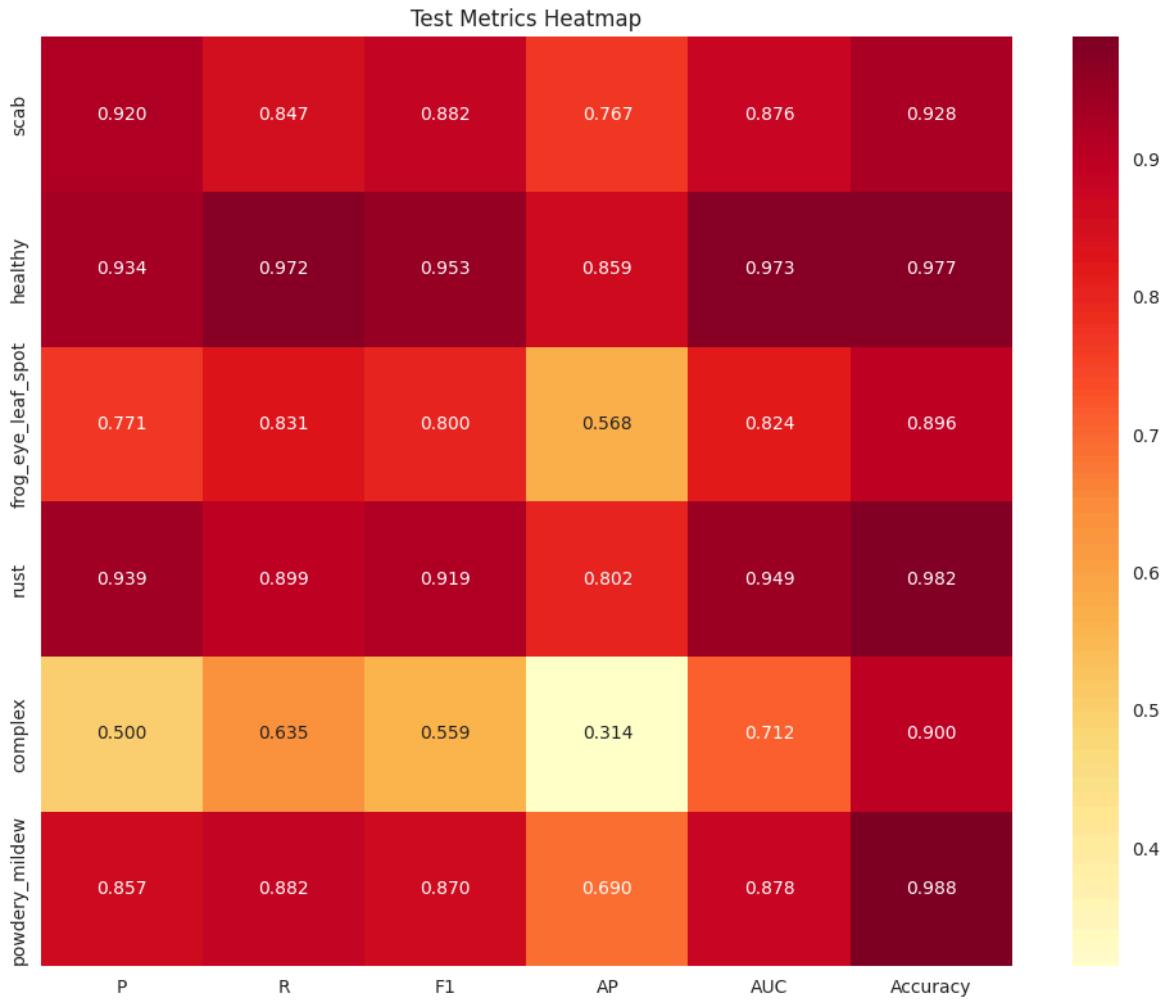
6.6 ArcPSASwinTransformer模型评估

训练有关的部分在附录，可以直接[点此翻阅](#)前往查看可视化结果和参数设置。

我对比了没有引入PSA模块的模型与引入了PSA模块的模型，分别生成了热力图来进行直观的评估。

这是没有引入PSA模块的模型在测试集上的评估结果：

热力图：

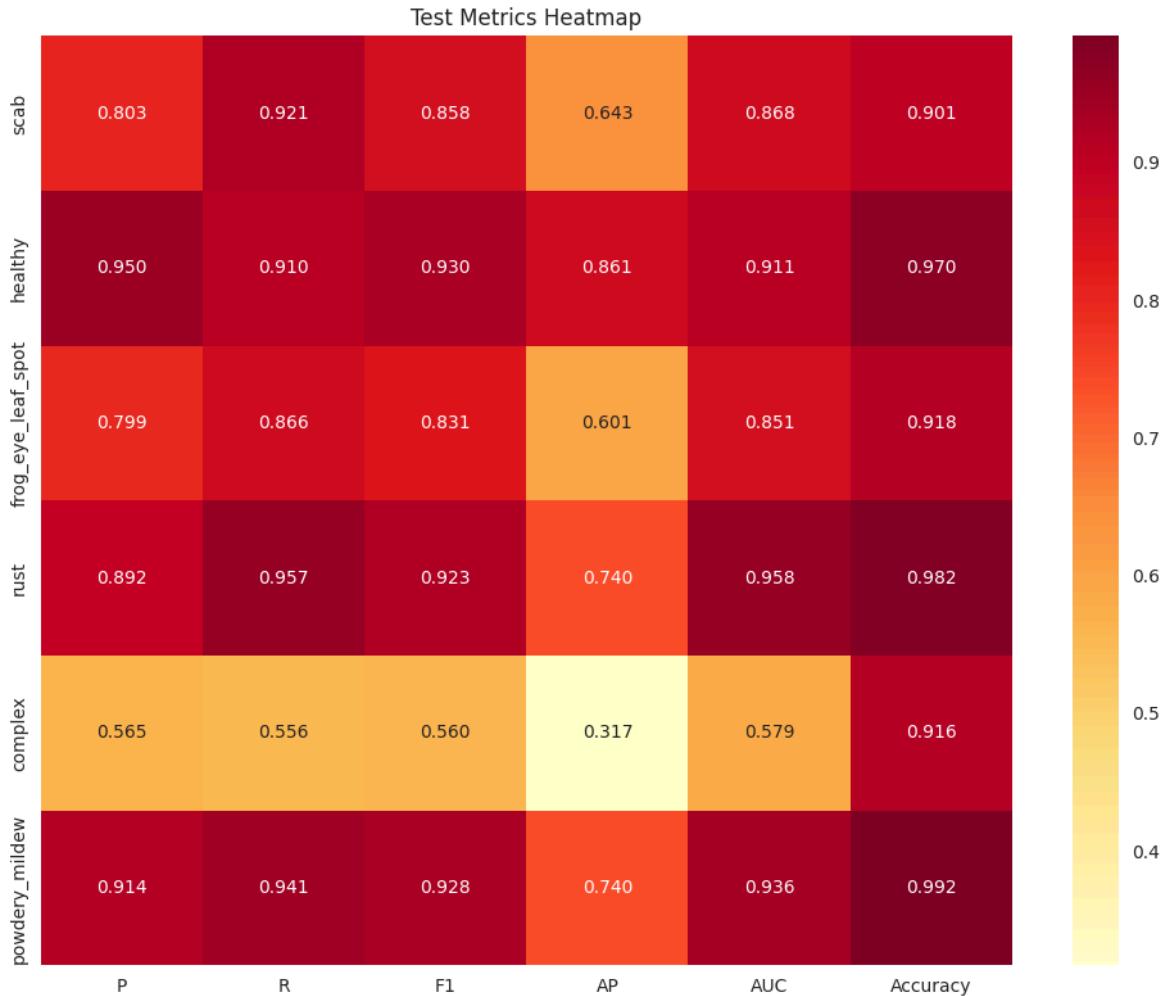


表格：

Class	P	R	F1	AP	AUC	Accuracy
scab	0.9200	0.8474	0.8822	0.7668	0.8755	0.9276
healthy	0.9338	0.9724	0.9527	0.8595	0.9732	0.9770
frog_eye_leaf_spot	0.7712	0.8310	0.8000	0.5682	0.8243	0.8964
rust	0.9394	0.8986	0.9185	0.8021	0.9493	0.9819
complex	0.5000	0.6349	0.5594	0.3144	0.7118	0.8997
powdery_mildew	0.8571	0.8824	0.8696	0.6897	0.8777	0.9885
macro-avg	0.8203	0.8444	0.8304	0.6668	0.8686	0.9452
micro-avg	0.8364	0.8585	0.8473	0.6453	0.8744	0.9452

可以发现整体准确率很好，各类别的AUC在0.87左右，证明引入的类间距的概念赋予模型不错的分类能力，但是各个类别的平衡性不够好，对于复杂的空间结构还是无法做到精确分类。

这是引入了PSA模块的ArcPSASwinTransformer模型在测试集上的评估结果：



表格：

Class	P	R	F1	AP	AUC	Accuracy
scab	0.8028	0.9211	0.8578	0.6427	0.8677	0.9013
healthy	0.9496	0.9103	0.9296	0.8608	0.9112	0.9704
frog_eye_leaf_spot	0.7987	0.8662	0.8311	0.6014	0.8510	0.9178
rust	0.8919	0.9565	0.9231	0.7400	0.9580	0.9819
complex	0.5645	0.5556	0.5600	0.3170	0.5787	0.9161
powdery_mildew	0.9143	0.9412	0.9275	0.7403	0.9362	0.9918
macro-avg	0.8203	0.8585	0.8382	0.6503	0.8505	0.9465
micro-avg	0.8255	0.8756	0.8498	0.6387	0.8674	0.9465

可以发现加入PSA模块后，各个类别的性能差距减小，说明模型的平衡性提升了。其中在complex类上的精确率从0.500提升到了0.5645，提升最为显著。而且所有类别的Accuracy都在0.90以上，AUC值分布更加集中，在不同类别上的表现更加稳定，F1分数也有一定的提升，对于complex类的提升效果非常明显。

但是总体的F1分数还有很多提升的空间，目前最高只能到达0.85左右，后续如果优化需要仔细调整

MultiLabelArcFaceLoss的scale与margin进行调参，在这个实验中我只是简单进行了参数的设置，没有进一步调参，在调参后模型效果应该会进一步增强。

7 训练过程概述

7.1 数据预处理与增强

这里的逻辑在前文已经介绍过，可以[点此翻阅](#)。

7.2 数据加载

数据通过自定义的 `PlantPathologyDataset` 类进行加载，并使用 `DataLoader` 进行批量处理。`DataLoader` 提供了高效的数据读取和批处理机制，支持多线程加载，加快训练过程。

7.3 损失函数与优化器

本项目在前两个模型上采用了 `BCEWithLogitsLoss` 作为损失函数，在SwinTransformer为基准模型的任务上采用了 `FocalLoss` 和自行设计的 `MultiLabelArcFaceLoss`，适用于多标签二分类任务。优化器选择了 `Adam`，因其在处理大规模数据和参数时表现出色，能够快速收敛。

- **优化器**: `Adam` 优化器凭借自适应学习率调整能力，能够在复杂的损失曲面上表现优异。
- **学习率调度器**: 使用 `CosineAnnealingLR`: 学习率随训练周期呈余弦退火，初期较大，后期逐渐减小，有助于模型在收敛时更细致地调整参数。

7.4 混合精度训练

为了加快训练速度并减少显存占用，项目引入了 **混合精度训练**。通过 `torch.cuda.amp` 中的 `autocast` 和 `GradScaler`，实现了半精度浮点数计算，提升了训练效率，同时保持了模型的精度。

7.5 早停机制

为了防止模型在训练集上过拟合，引入了 **早停机制**。该机制通过监控验证集的 F1 分数，当连续多个 epoch 验证分未见提升时，提前终止训练。

- **监控指标**: 主要监控验证集的 F1 分数，以衡量模型的整体性能。
- **模型保存**: 每当验证 F1 分数提升时，保存当前最优模型的检查点。
- **提前终止**: 当连续 `patience` 个 epoch 内验证 F1 分数未提升时，提前停止训练，避免过拟合。

7.6 模型保存与加载

训练过程中，最佳模型的检查点被保存，以便后续的评估和部署。通过 `utils.py` 中的 `save_checkpoint` 和 `load_checkpoint` 函数，实现了模型的保存与加载。

- **保存内容**: 包括当前 epoch、模型参数、优化器状态以及验证损失。
- **加载模型**: 在评估和时，使用 `load_checkpoint` 函数加载最佳模型，确保评估的准确性和一致性。

7.7 验证机制

同时，在 `evaluate.py` 中，也设置了为每一类别寻找最佳的 `threshold`，多标签分类任务中，因为不同类别具有不同的特性和难易程度，因此一个统一的阈值可能无法为所有类别提供最佳性能。并且我保存了最后得到的最佳结果

7.8 训练完成

训练结束后，脚本输出最佳验证 F1 分数及其对应的 epoch，标志着训练过程的完成。

在此贴出一个训练循环来辅助说明：

```
def train_epoch(model, loader, criterion, optimizer, device, scaler, accumulation_steps=2):
    ...
```

训练一个epoch，使用混合精度训练和梯度累积来提高效率。

```
"""
model.train()
running_loss = 0.0

# 初始化每个类别的准确率统计
running_class_accuracies = np.zeros(6)
num_batches = 0

all_targets = []
all_outputs = []

optimizer.zero_grad()
progress_bar = tqdm(loader, desc="Training")

for idx, (images, labels) in enumerate(progress_bar):
    images = images.to(device)
    labels = labels.to(device)

    # 混合精度训练
    with autocast():
        # model 输出的是特征向量，而不是 logits
        features = model(images)
        # criterion: MultiLabelArcFaceLoss(features, labels) => (loss, logits)
        loss, logits = criterion(features, labels)
        # 为了梯度累积，对 loss 做平均
        loss = loss / accumulation_steps

    # 反向传播
    scaler.scale(loss).backward()

    # 梯度累积判断
    if (idx + 1) % accumulation_steps == 0:
        scaler.unscale_(optimizer)
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

        scaler.step(optimizer)
        scaler.update()
        optimizer.zero_grad()

    # ---- 准确率计算：对 logits 做 sigmoid ----
    # 只有经过 sigmoid 才能阈值分割多标签
    probs = torch.sigmoid(logits)

    # 计算每个类别的准确率
    batch_class_accuracies, batch_overall_accuracy = calculate_batch_accuracy(
        probs, labels
    )
    running_class_accuracies += batch_class_accuracies
    num_batches += 1

    # 累积loss（要用加回 *images.size(0) 还原之前除的 accumulation_steps）
    running_loss += loss.item() * accumulation_steps * images.size(0)

    all_targets.append(labels.detach().cpu().numpy())
    all_outputs.append(probs.detach().cpu().numpy())

```

```

# 更新进度条信息
progress_info = {
    'loss': f'{loss.item():.4f}',
    'avg_acc': f'{batch_overall_accuracy:.4f}'
}
for i, acc in enumerate(batch_class_accuracies):
    progress_info[f'c{i}_acc'] = f'{acc:.2f}'
progress_bar.set_postfix(progress_info)

# 计算 epoch 级别指标
epoch_loss = running_loss / len(loader.dataset)
all_targets = np.vstack(all_targets)
all_outputs = np.vstack(all_outputs)

# 计算f1、mAP等
f1_score, _, map_score = calculate_metrics(all_outputs, all_targets)

# 计算每个类别平均准确率
class_accuracies = running_class_accuracies / num_batches
overall_accuracy = np.mean(class_accuracies)

return {
    'loss': epoch_loss,
    'accuracy': overall_accuracy,
    'class_accuracies': class_accuracies.tolist(),
    'f1': f1_score,
    'map': map_score
}

```

8 项目结构说明

8.1 目录

`data`

文件夹中需要自己导入图片数据，但已经有处理好的 `processed_train_labels.csv` 等标签文件，可以直接使用

`checkpoints`

- 存放了训练好的模型权重，需要时可以自行导入
- 存放了test与validation的指标可视化结果，包括热力图，折线图，柱状图与表格

`outputs`

- 主要存放可视化的结果，按照图片ID为索引创建每个图片的注意力可视化结果
- 存放了EfficientNet的可视化gif图片
- `gifs`

存放了注意力可视化拼接后的gif图片

8.2 代码结构

项目代码位于 `src/` 目录下，主要包括以下脚本：

`dataset.py`

负责数据集的加载和预处理。定义了 `PlantPathologyDataset` 类，用于读取图像及其对应的标签，并应用必要的图像变换。

`model.py`

定义了模型结构。通过 `get_model` 函数加载EfficientNet 模型，加入STN模块的EVA-CLIP模型与 ArcPSASwinTransformer模型。

`train.py`

包含训练和验证的循环逻辑。利用 GPU 进行加速训练，使用混合精度训练技术提高效率，并实现早停机制防止过拟合。

`evaluate.py`

用于在验证集上评估训练好的模型性能。计算损失、F1 分数和准确率等指标，并且在验证集上动态更新模型的 `threshold`，在得到最优的 `threshold` 后，在测试集上进行测试，并且生成对应的曲线

`lossfunction.py`

定义了 `FotolLoss` 与 `MultiLabelArcFace` 损失函数，在训练评估测试过程可以直接调用

`visualize.py`

应用GradCam，将EfficientNet的运行过程可视化，生成训练过程的热力图

`attentionvis.py`

引入注意力可视化机制，将EVA-CLIP的运行过程可视化，生成训练过程的热力图

`arcvis.py`

`enhanced_modules`

- 定义STN模块，导入到EVA-CLIP模型中进行优化
- 定义了PSA模块，导入到SwinTransformer模型中进行优化

`utils.py`

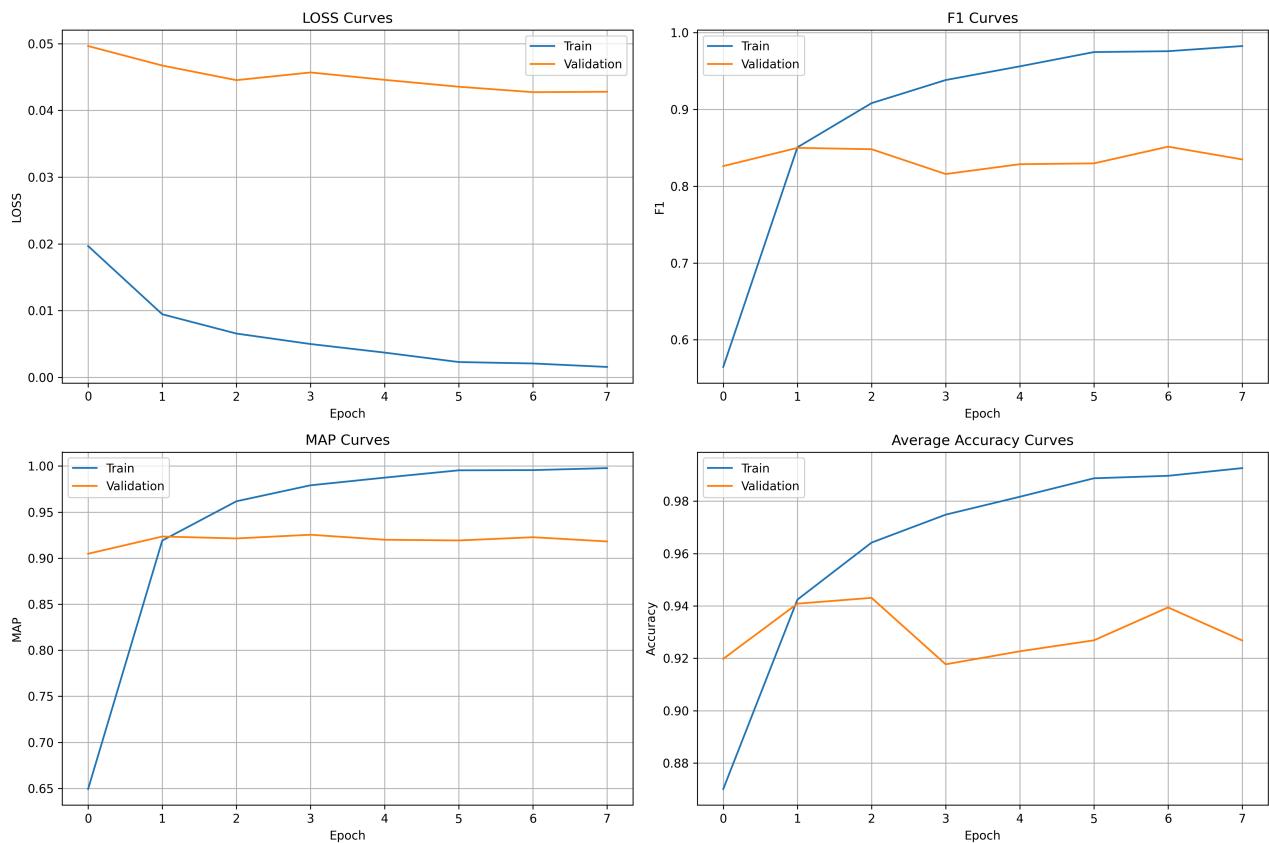
提供辅助功能，包括：

- `calculate_metrics`: 计算 F1 分数和准确率。
- `save_checkpoint`: 保存模型检查点。
- `load_checkpoint`: 加载模型检查点。
- `tensor2img`: 将张量转换为图像格式。
- 其他可视化辅助函数。

附录

1 ArcPSASwinTransFormer训练历史

Epoch	Train Loss	Train Acc	Val Loss	Val Acc	Train F1	Val F1	Train mAP	Val mAP
1	0.0197	0.8701	0.0496	0.9198	0.5646	0.8262	0.6493	0.9048
2	0.0094	0.9424	0.0467	0.9409	0.8507	0.8500	0.9191	0.9235
3	0.0066	0.9642	0.0445	0.9431	0.9082	0.8482	0.9618	0.9214
4	0.0050	0.9748	0.0457	0.9177	0.9383	0.8158	0.9791	0.9254
5	0.0037	0.9817	0.0446	0.9227	0.9561	0.8287	0.9874	0.9200
6	0.0023	0.9887	0.0435	0.9268	0.9747	0.8297	0.9954	0.9192
7	0.0021	0.9897	0.0427	0.9394	0.9759	0.8515	0.9956	0.9227
8	0.0016	0.9926	0.0428	0.9268	0.9826	0.8349	0.9978	0.9182



Training Configuration and Results

=====
Model: ArcPSASwinTransFormer
Image Size: 384x384
Batch Size: 16
Learning Rate: 0.0001
Number of Epochs: 8
Best Val F1: 0.8515 (Epoch 7)

2 训练输出

此处选取某一次EVA-CLIP的输出，目的是展示输出的格式与训练代码规范性

```
Original complex samples: 355
Augmented complex samples: 2130
Original complex samples: 63
Augmented complex samples: 378
已加载本地模型权重: /home/visllm/.cache/huggingface/EVA-CLIP
Epoch 1/30
Train Loss: 0.4665 | Train F1: 0.2551 | Train mAP: 0.3618 | Train Acc: 0.1967
Val Loss: 0.2867 | Val F1: 0.3884 | Val mAP: 0.6193 | Val Acc: 0.4950
保存最佳模型于 epoch 1
Epoch 2/30
Train Loss: 0.2245 | Train F1: 0.6544 | Train mAP: 0.8040 | Train Acc: 0.6400
Val Loss: 0.1632 | Val F1: 0.7307 | Val mAP: 0.8878 | Val Acc: 0.7333
保存最佳模型于 epoch 2
Epoch 3/30
Train Loss: 0.1542 | Train F1: 0.8008 | Train mAP: 0.8825 | Train Acc: 0.7623
Val Loss: 0.1411 | Val F1: 0.7694 | Val mAP: 0.9032 | Val Acc: 0.7850
保存最佳模型于 epoch 3
Epoch 4/30
Train Loss: 0.1278 | Train F1: 0.8495 | Train mAP: 0.9175 | Train Acc: 0.8047
Val Loss: 0.1170 | Val F1: 0.8616 | Val mAP: 0.9208 | Val Acc: 0.8200
保存最佳模型于 epoch 4
Epoch 5/30
Train Loss: 0.1072 | Train F1: 0.8751 | Train mAP: 0.9326 | Train Acc: 0.8343
Val Loss: 0.1140 | Val F1: 0.8686 | Val mAP: 0.9238 | Val Acc: 0.8267
保存最佳模型于 epoch 5
Epoch 6/30
Train Loss: 0.0913 | Train F1: 0.8965 | Train mAP: 0.9494 | Train Acc: 0.8560
Val Loss: 0.1175 | Val F1: 0.8742 | Val mAP: 0.9231 | Val Acc: 0.8200
保存最佳模型于 epoch 6
Epoch 7/30
Train Loss: 0.0809 | Train F1: 0.9057 | Train mAP: 0.9628 | Train Acc: 0.8697
Val Loss: 0.1131 | Val F1: 0.8794 | Val mAP: 0.9334 | Val Acc: 0.8317
保存最佳模型于 epoch 7
Epoch 8/30
Train Loss: 0.0715 | Train F1: 0.9212 | Train mAP: 0.9718 | Train Acc: 0.8853
Val Loss: 0.1199 | Val F1: 0.8713 | Val mAP: 0.9242 | Val Acc: 0.8367
Epoch 9/30
Train Loss: 0.0612 | Train F1: 0.9332 | Train mAP: 0.9795 | Train Acc: 0.8963
Val Loss: 0.1253 | Val F1: 0.8661 | Val mAP: 0.9238 | Val Acc: 0.8300
Epoch 10/30
Train Loss: 0.0527 | Train F1: 0.9421 | Train mAP: 0.9823 | Train Acc: 0.9147
Val Loss: 0.1196 | Val F1: 0.8771 | Val mAP: 0.9294 | Val Acc: 0.8383
Epoch 11/30
Train Loss: 0.0442 | Train F1: 0.9519 | Train mAP: 0.9875 | Train Acc: 0.9300
Val Loss: 0.1312 | Val F1: 0.8660 | Val mAP: 0.9226 | Val Acc: 0.8333
Epoch 12/30
Train Loss: 0.0404 | Train F1: 0.9584 | Train mAP: 0.9907 | Train Acc: 0.9340
Val Loss: 0.1333 | Val F1: 0.8655 | Val mAP: 0.9224 | Val Acc: 0.8317
Epoch 13/30
Epoch 13: reducing learning rate of group 0 to 1.0000e-05.
Train Loss: 0.0362 | Train F1: 0.9598 | Train mAP: 0.9918 | Train Acc: 0.9403
Val Loss: 0.1271 | Val F1: 0.8708 | Val mAP: 0.9253 | Val Acc: 0.8350
```

Epoch 14/30
Train Loss: 0.0312 | Train F1: 0.9720 | Train mAP: 0.9960 | Train Acc: 0.9510
Val Loss: 0.1300 | Val F1: 0.8705 | Val mAP: 0.9255 | Val Acc: 0.8333
Epoch 15/30
Train Loss: 0.0300 | Train F1: 0.9736 | Train mAP: 0.9956 | Train Acc: 0.9557
Val Loss: 0.1288 | Val F1: 0.8692 | Val mAP: 0.9270 | Val Acc: 0.8383
Epoch 16/30
Train Loss: 0.0286 | Train F1: 0.9749 | Train mAP: 0.9959 | Train Acc: 0.9567
Val Loss: 0.1309 | Val F1: 0.8720 | Val mAP: 0.9253 | Val Acc: 0.8433
Epoch 17/30
Train Loss: 0.0293 | Train F1: 0.9712 | Train mAP: 0.9961 | Train Acc: 0.9520
Val Loss: 0.1293 | Val F1: 0.8707 | Val mAP: 0.9260 | Val Acc: 0.8400
早停
训练完成。最佳验证 F1 分数: 0.8794 在 epoch 7

损失曲线已保存
可视化曲线已保存