中山大學
SUN YAT-SEN UNIVERSITY

计算机视觉

题 目：	Plant Pathology-2021 分类任务
姓 名：	李帅帅
学 号：	21312912
指导教师：	梁小丹
专业班级：	21 级智科一班

2023 年 12 月

本科生院制

Plant Pathology-2021 with VIT

摘要

苹果是世界上最重要的温带水果作物之一。叶面（叶）病对苹果园的整体生产力和质量构成重大威胁。目前苹果园的疾病诊断过程是基于人类的人工侦察，这既费时又昂贵。

本文首先针对数据集，对其分布、各标签含义等进行了初步解读，并针对具有复杂病的叶片 label 设计了一个**多标签分类的 one-hot 编码**方式。接着对此分类问题的难度进行了一定分析和解读。

接着，本文对比了几个常见的深度学习分类任务模型：EfficientNet，resnet50，Vit 模型，通过深入理解各架构的机制，最终选择了 ViT 模型作为本工作的网络架构。

由于 Vit 模型适用于在大规模数据集上预训练的前提条件，本文首先利用**迁移学习**的思想将在 ImageNet 数据集上预训练的 **ViT-Base** 模型作为后续工作开展的基础。

并且为了测试不同模型参数的效果，本文对比了 Adam，SGD，**AdamW** 三个优化器的性能，最终选择了 AdamW 作为模型优化器。接着对比了 BCEWithLogitsLoss 和 ArcFaceloss 两个损失函数的性能，经过对比发现，ArcFaceloss 可能与此任务不适配，最终选择了 **BCEWithLogitsLoss** 作为损失函数。

为了缩短模型训练时间，本文采用**降低图片分辨率**的方式，并且采用 **AutoAugment 策略**在训练集图片上进行数据增强。为了在一定程度上降低模型过拟合的效果，采用**余弦退火学习率调度器**，

接着利用利用 pytorch_grad_cam 库完成特征图可视化步骤，对误差进行分析，同时在一定程度上评估网络性能。

最终使模型性能有所提高，在测试集上达到了 85%左右的正确率。

关键词： ViT； 苹果叶片疾病诊断； AdamW；迁移学习；one-hot 编码；BCEWithLogitsLoss；AutoAugment 策略；余弦退火学习率调度；热力图

一、 问题重述

苹果是世界上最重要的温带水果作物之一。叶面（叶）病对苹果园的整体生产力和质量构成重大威胁。目前苹果园的疾病诊断过程是基于人类的人工侦察，这既费时又昂贵。

尽管基于计算机视觉的模型已经显示出植物病害识别的前景，但仍有一些局限性需要解决。不同苹果品种或栽培新品种之间单一疾病的视觉症状存在巨大差异，是基于计算机视觉的疾病识别的主要挑战。这些变化源于自然和图像捕捉环境的差异，例如叶色和叶片形态、受感染组织的年龄、不均匀的图像背景以及成像过程中不同的光照等。

Plant Pathology-2021 是 Kaggle 举办的 FGVC8(CVPR2021-workshop)的数据集，任务是对叶片的病害进行分类。

具体有以下任务：

- 参照 Lenet2、VGG16、ResNet50、VIT 或其他模型做适当改进，设计一个神经网络
- 使用构建的模型对 Plant Pathology-2021 数据集进行训练，对于给定的 test 测试集中的图片，准确地预测出其类别

二、 问题分析

首先观察数据集，其已经被分为训练集(train)、验证集(val)、测试集(test)三类，并且训练集有 3000 条数据，验证集和测试集分别有 600 条数据。初步观察图片对应的标签得知，有些图片对应的 label 为多标签问题，故此分类问题为多分类问题。

同时对图片进一步观察，发现叶片病害共分为 5 类（healthy 除外），存在类内差异大，类间差异小的情况，属于细粒度分类问题，进一步加大了分类难度。

- **难点一：**叶片病害分类的类内差异大，类间差异小，属于细粒度分类问题，对该问题有见解和解决方案会加分。
- **难点二：**该数据集有以下 12 个类别，不同的类别之间有重复，例如[scab]存在于多个类别中，这会影响模型的训练精度。

三、 数据统计分析

为了更好的观测数据集 label 的分布，先对 label 合并前和合并后分级别进行统计，如下表：

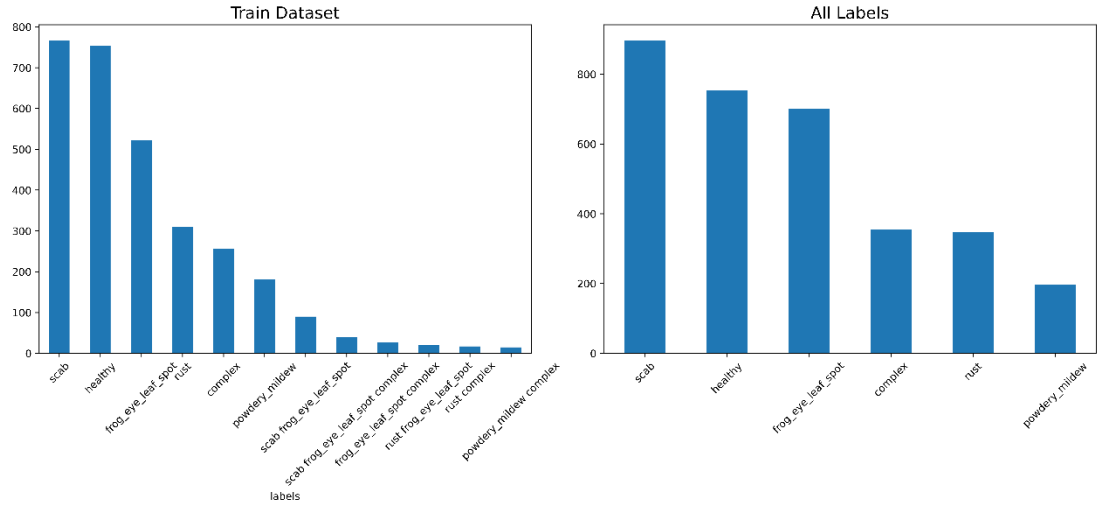
表 1 修改前类别统计

id	label_old	num
1	scab	767
2	healthy	753
3	frog_eye_leaf_spot	523
4	rust	309
5	complex	256
6	powdery_mildew	182
7	scab frog_eye_leaf_spot	90
8	scab frog_eye_leaf_spot complex	40

9	frog_eye_leaf_spot complex	27
10	rust frog_eye_leaf_spot	21
11	rust complex	17
12	powdery_mildew complex	15

表 2 修改后类别统计

id	label_new	num
1	complex	355
2	frog_eye_leaf_spot	701
3	healthy	753
4	powdery_mildew	197
5	rust	347
6	scab	897



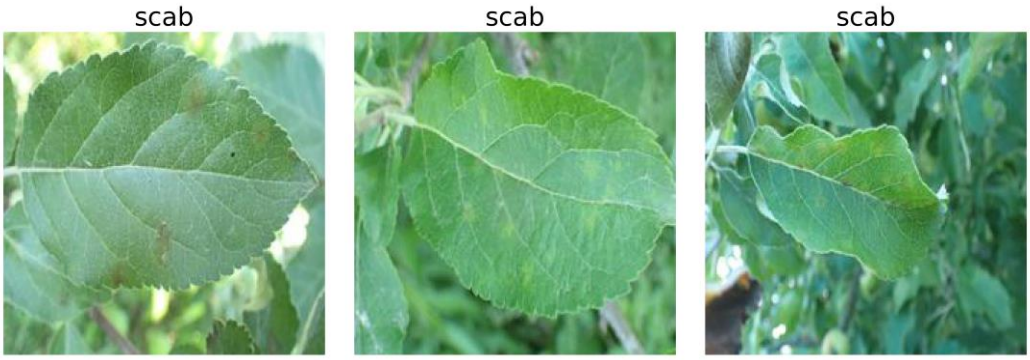
初步统计发现，对于改变后的类别，最大类别数为 897，最小类别数为 197，存在类别不均衡的问题，故在后续训练过程中需要注意，以避免类别数小的类别的正确率过低的现象。同时对各病害进行初步了解：

rust 标签指的是苹果叶部锈病，也称之为赤星病、苹桧锈病等。苹果锈病属于真菌感染而引起的病，苹果锈病的形成与 2 月到 3 月的气候息息相关，尤其是受湿度的影响最大，当天气持续降雨两天，并且雨量、空气湿度达到一定标准时，就会造成病菌冬孢子角的萌发和小孢子的侵染。该病主要发生在苹果的叶片、嫩枝、幼果和果柄这些部位，该病最终可造成苹果树叶提早脱落等。苹果树叶被感染后，在开始在 5 月下旬可见叶片正面出现病斑，最初颜色为桔红色，形为直径约 1～2mm 的小圆点，不加以防治的话，病斑会越长越大，且中间部分颜色逐渐变深直到变为黑色小点，此时病斑最外圈的颜色相对淡一些，并在病斑中间部分能观察到许多小点。在 6 月下旬，苹果锈病的病斑会变大到直径为 1cm 的圆圈，此时的苹果叶片叶肉变肥肥厚，且叶面变得凹凸不平。

powdery_mildew 标签是指苹果叶部白粉病，该病由白叉丝单囊壳引起的一种苹果病害，在我国白粉病是一种常见的苹果病害，在我国的所有苹果产区都会有苹果树患上白粉病，且每年发病程度都不同，在不同种植地区苹果树的患病程度也有差异，但是总的来说发病程度为中等程度。白粉病主要发病于苹果叶片，苹果叶片被染病后，该叶片上会产生一些病斑，颜色成白色，看上去就像铺上了一层白粉，因此该病命名为白粉病，不加以防治，患病叶片病症会越来越严重。

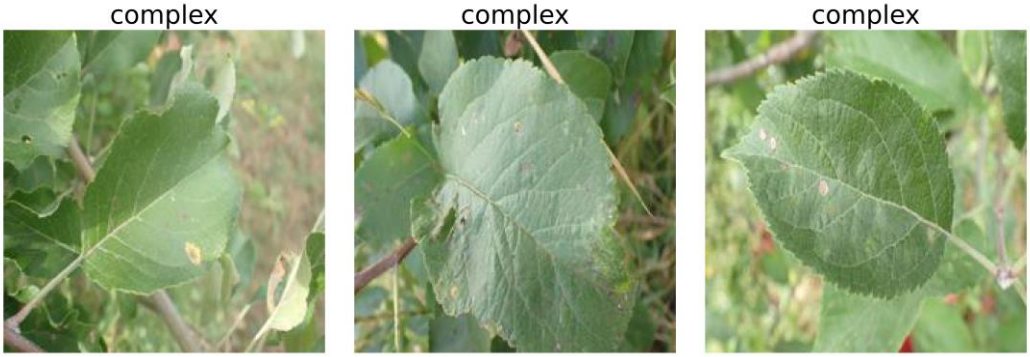
frog_eye_leaf_spot 指的是苹果叶部灰斑病（也叫青蛙眼病），主要病发苹果叶片，在秋季的危害最为严重。叶片在感染灰斑病的初期，病斑形状成圆形，颜色为黄褐色，随着患病程度的加深，病斑的颜色变为灰色。当天气条件达到高温且多雨时，患病叶片的病斑会快速病变，生成密集的病斑连成一片，并且在病斑的中心位置会出现一些小黑点，最终患病叶片不会脱落，但会患病严重的叶片会变得焦枯。

scab 苹果黑星病的症状包括叶片上的小圆斑点，随后逐渐扩大为深褐色至黑色不规则病斑。受感染的叶片可能弯曲、扭曲，并在潮湿条件下产生橙褐色孢子床。严重感染时，叶片可能变黄并凋落。



healthy 则是正常的苹果叶片。

complex 类别则是情况较为复杂的叶片。

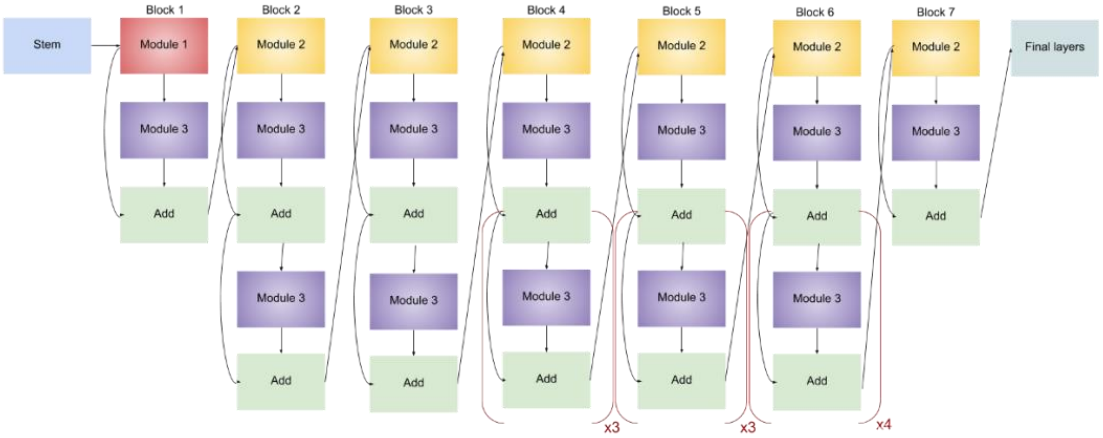


综上，可以发现，部分病害存在一定的相似性，例如 frog_eye_leaf_spot 和 rust 类，其病情均会出现黄褐色斑点；并且即使是同一种病，在不同时期，不同的严重度都会影响判别，例如 frog_eye_leaf_spot，部分后期已经出现黑色，早期则偏黄。上述问题都会影响后续病情判别，属于细粒度分类问题，在后续工作中需要着重关注。

四、 初步探索

在任务初期，本文决定先尝试套几个模型试试效果，首先为了降低难度，刚开始采用了一种较为简单的编码方式，即“如果某个数据同时对应多个标签，则将其视为第一类”，这种编码方式严格来讲是错误的，因为它忽视了多种病并存的情况，但为了降低难度，并且是“初步探索阶段”，故先用此种方式试试水。

本文首先选择了 EfficientNet，因其被誉为“可能是迄今为止最好的 CNN 网络”，其网络架构如下：



EfficientNet-B3 的网络框架，总体看，分成了 3 个 Stage：

EfficientNet-B3 网络框架
Stage1：一个卷积核大小为 3x3，步距为 2 的普通卷积层（包含 BN 和激活函数 Swish）
Stage2：重复堆叠 MBConv 结构
Stage3：一个普通的 1x1 的卷积层(包含 BN 和激活函数 Swish) + 一个平均池化层 + 一个全连接层组成

其中，MBConv 结构主要包括：

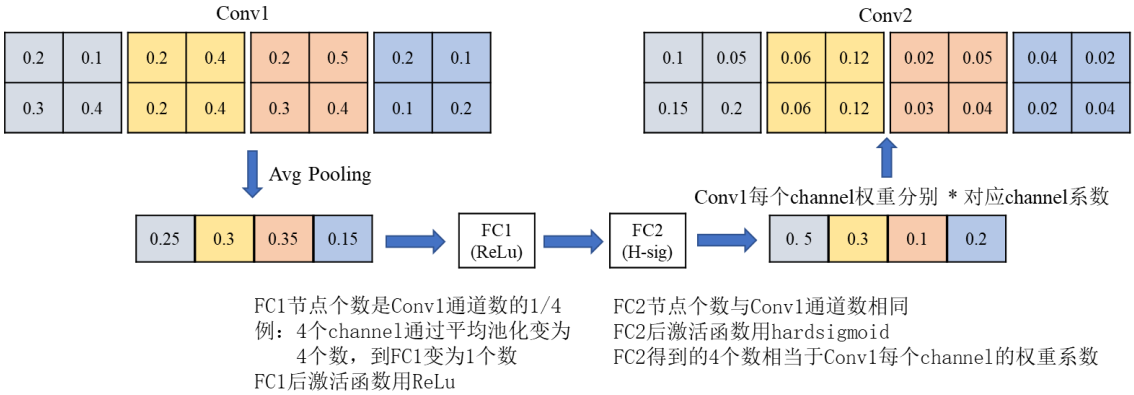
一个 1x1 的普通卷积(升维作用，包含 BN 和 Swish 激活)

一个 kxk 的 Depthwise Conv 卷积(包含 BN 和 Swish 激活)，kxk 有 3x3 和 5x5 两种情况

一个 SE 模块：由一个全局平均池化，两个全连接层组成。

一个 1x1 的普通卷积(降维作用，包含 BN 和线性激活，线性激活 y=x)

一个 Droupout 层



由于其同时兼顾了网络深度(depth)、网络宽度(width)和输入图像分辨率(resolution)大小，故在大多数数据集上都表现良好。

Model	**width coefficient**	**depth coefficient**	**drop connect rate**	**dropout rate**
EfficientNet-B3	1.2	1.4	0.2	0.3

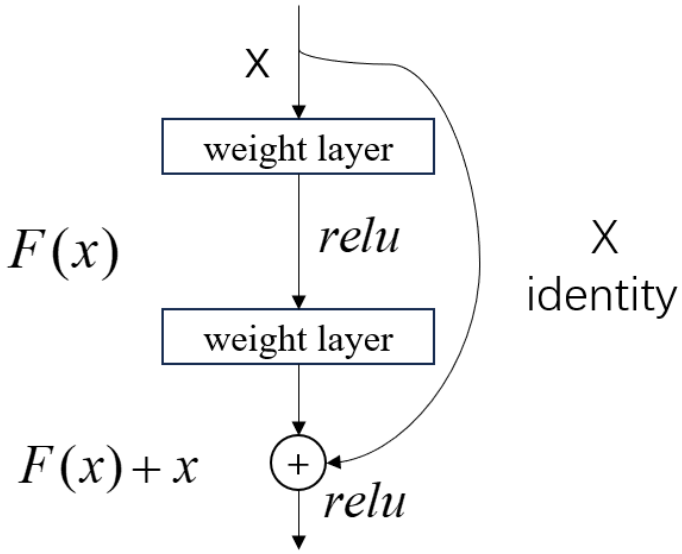
在得到其与"imagenet"上预训练的数据集后，进行了初步训练，最终训练集得到了 95%的正确率，**测试集**正确率能够达到 **80%左右**。

考虑到训练集可能存在过拟合的情况，进一步在 EfficientNet-B3 结构后面加入一个正则化层以加快网络训练，同时接入一个全连接层、一个 Dropout 层和一个全连接层，来避免过拟合，与此同时增加训练 epoch，最终的效果为：训练集正确率为 93%，**测试集正确率为 85%左右**，有所提升。

考虑到测试集正确率表现较优，这里决定加大难度，即采用任务要求中推荐的 one-hot 多标签分类问题，例如原本其标签为[rust scab]，则在对应位置均加上标签，其编码结果假设为[1, 1, 0, 0, 0, 0]，倘若对每一类结果都预测正确即为正确结果。

重新训练并进行测试，最终测试集正确率仅为 40%，可见效果很差。

后继续用 resnet50 测试，Res Net 网络的关键就在于其结构中的残差单元，如下图所示，在残差网络单元中包含了跨层连接，图中的曲线可以将输入直接跨层传递，进行了同等映射，之后与经过卷积操作的结果相加。



Resnet50 网络中包含了 49 个卷积层、一个全连接层，初步训练，最终在训练集达到 99%的正确率，而验证集和测试集均只有 30%左右的正确率，可见其严重过拟合，在此数据集上表现不佳。

进一步的，本文还测试了 Vision Transformer（VIT）模型的效果，但只有 25%左右的正确率，以下是对初步探索的归纳总结：

model	**acc**
EfficientNet-B3 改	40%
Resnet50	30%
VIT	25%

但对其进一步了解发现，ViT 原论文中最核心的结论是，当拥有足够多的数据进行预训练的时候，ViT 的表现就会超过 CNN，突破 transformer 缺少归纳偏置的限制，可以在下游任务中获得较好的迁移效果。故本文决定展开对 VIT 的进一步研究探讨。

五、 VIT 模型构建

5.1 迁移学习

有了上述探究，很容易得出这样的结论：迁移学习对于 VIT 模型来说很重要。迁移学习（Transfer Learning）是一种通过利用一个任务上学到的知识来改善在另一个相关任务上的学习的机器学习方法。迁移学习可以分为几种不同的方法，主要包括以下几类：

5.1.1 特征提取（Feature Extraction）：

在这种方法中，使用预训练模型的底层（通常是卷积神经网络的前几层）作为特征提取器。将这些层的权重冻结，然后添加一个新的全连接层或其他适当的层，用于适应特定的任务。这种方法适用于原始任务和目标任务之间有相似底层特征的情况，比如图像分类任务。

5.1.2 微调（Fine-tuning）：

在微调中，使用预训练模型的权重初始化新模型，并在目标任务上进行训练。与特征提取不同，微调允许对整个模型进行调整，包括底层和顶层。这在目标任务与原始任务相似但仍存在一些差异时是有用的。

5.1.3 领域自适应（Domain Adaptation）：

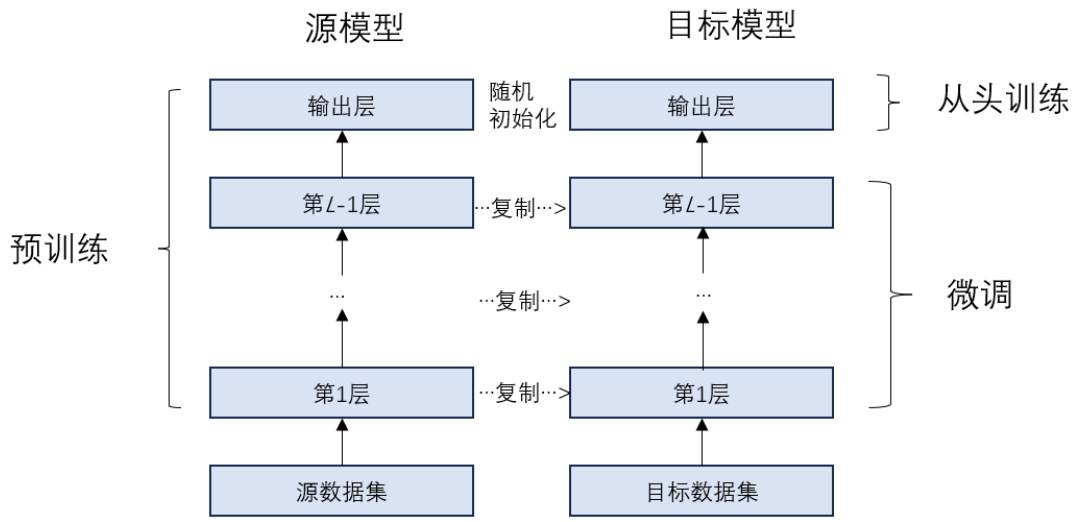
这种方法适用于源领域（训练数据）和目标领域（测试数据）之间有一些分布差异的情况。通过使用源领域的知识来适应目标领域的数据分布，以提高模型在目标领域上的性能。

5.1.4 多任务学习（Multi-task Learning）：

多任务学习通过同时训练模型来执行多个相关任务，从而提高模型的泛化性能。在这种情况下，模型在多个任务上学到的知识可以有助于目标任务的学习。

5.1.5 学习对抗特征（Adversarial Feature Learning）：

通过引入对抗网络，将源领域和目标领域之间的特征进行对抗学习，以减小领域之间的分布差异。



本文主要采用微调（Fine-tuning）的方式，即从别的数据集中预训练，再获取各层模型参数，在此数据集上进行微调，从而达到迁移的效果。

ImageNet 是一个非常有名的大型视觉数据集，它的建立旨在促进视觉识别研究。训练 ImageNet 数据集需要消耗大量的计算资源。ImageNet 为超过 1400 万张图像进行了注释，而且给至少 100 万张图像提供了边框。本文利用网上开源的 VIT-Base 模型作为预训练模型，其中其预训练的数据集为 ImageNet。

在之前的某 few-shot 比赛中，曾用过一次预训练的方式，即在其他数据集上训练，再迁移到比赛数据集，但由于数据的 num_classes 不同，故在最后的全连接层存在一定的差异，需采用“权重迁移”的方法，即将前半部分网络的权重迁移过来，舍弃最后几层的权重，与此方法有“异曲同工”之妙。

5.2 one-hot 编码

由于该数据集有 12 个类别，不同的类别之间有重复，例如[scab]存在于多个类别中，这会影响模型的训练精度。为了使 label 更加具有代表性，在前面数据预处理阶段已经将 label 中的复合标签拆分成 6 类，故这里采用 one-hot 编码，将复合标签转化为 1 维的列向量，其中每个列向量都有 6 个值，对应位置为 1 则为有对应疾病，0 则为无对应疾病，相关编码方式为：

```python
def encode_label(labels, class_list):   # 对标签进行编码(one-hot)
    target = torch.zeros(len(class_list))
    for label in labels:
        idx = class_list.index(label)  # 对应位置置1
        target[idx] = 1
    return target

def decode_label(encoded_label, class_list):
    decoded = [class_list[i] for i, val in enumerate(encoded_label) if val == 1]
    return decoded
```

image	label	encode
9742c28e983ef46b.jpg	scab frog_eye_leaf_spot	[0., 1., 0., 0., 0., 1.]
d4c367739964b80e.jpg	scab frog_eye_leaf_spot	[0., 1., 0., 0., 0., 1.]
af8743fa46a09c1d.jpg	healthy	[0., 0., 1., 0., 0., 0.]

5.3 模型参数

首先利用 vitbase 模型，并设置如下参数：

参数	取值
ema	true
optimizer	Adam
max_epochs	30
loss	BCEWithLogitsLoss
initial_lr	0.00001
criterion_params	'smooth_eps': 0.20

最终运行结果为：测试集正确率为 95.3%，验证集正确率为 83.17%，测试集正确率为 78.5%。

可以发现，正确率在低于 80%，有一定的上升空间，同时通过对模型进行 summary 发现，其参数过多，具体参数数量见下表：

```
Total params: 85,651,206
Trainable params: 85,651,206
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 325.49
Params size (MB): 326.73
Estimated Total Size (MB): 652.80
----------------------------------------------------------------
```

后续将会采取一系列措施对正确率进行改进，并采取适当策略缩短训练时间。

六、 ViT-Base 模型改进

6.1 减小图片分辨率

首先针对图片，初步观察可以发现其分辨率为 4000*2672，当使用全连接神经网络处理大尺寸图像时，有三个非常明显的缺点：（1）将图像展开为向量会丢失空间信息；（2）参数过多效率低下，训练困难；（3）大量的参数也很快会导致网络过拟合。

故将图片 resize 为 224*224 大小，以提高训练速度，一定程度上避免过拟合。

6.2 优化器选择

选择合适的优化器对于提高神经网络的性能至关重要。 不同的优化器可能导致模型在训练过程中以不同的速度收敛。一些优化器，如 Adam，通常具有较快的收敛速度，特别是在初期阶段。这有助于在有限的时间内更快地得到一个较好的模型。一些优化器对学习率的选择相对不敏感，因此对于不同任务和数据集，它们可能更具鲁棒性。这在面对不同问题时可以提供更一致的性能。

优化器的选择可能对模型的泛化性能产生影响。某些优化器可能导致模型更容易过拟合，而另一些可能对泛化性能更有利。

常见的优化器有以下几种：

6.2.1 随机梯度下降（SGD）：

是最基本的优化算法之一，它使用每个样本的梯度来更新模型参数。

$$\begin{aligned} g_t &= \nabla_{\theta_{t-1}} f(\theta_{t-1}) \\ \Delta\theta_t &= -\eta * g_t \end{aligned} \tag{1}$$

6.2.2 Adam 算法（自适应矩估计）

Adam(Adaptive Moment Estimation)是另外一种给每个参数计算不同更新速率的方法，其本质上是带有动量项的 RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。

$$\begin{aligned} m_t &= \mu * m_{t-1} + (1-\mu) * g_t \\ n_t &= \upsilon * n_{t-1} + (1-\upsilon) * g_t^{\,2} \\ \hat{m}_t &= \frac{m_t}{1-\mu^t} \\ \hat{n}_t &= \frac{n_t}{1-\upsilon^t} \\ \Delta\theta &= -\frac{\hat{m}_t}{\sqrt{\hat{n}_t + \varepsilon}} * \eta \end{aligned} \tag{2}$$

6.2.3 AdamW

AdamW 是对 Adam 优化器的一种改进，目的是解决 Adam 在权重衰减（Weight Decay）方面的问题。Weight Decay 是一种正则化方法，它通过在更新过程中对模型参数的梯度添加一个额外的项来降低模型的复杂度。

分别尝试上述优化器，最终的结果对比为：

优化器	acc
Adam	78.5%
AdamW	80.83%
SGD	74.67

6.3 损失函数选择

6.3.1 ArcFace Loss

ArcFace Loss（或称为 ArcFace 损失）是一种用于人脸识别任务的损失函数。它在深度学习中的人脸识别应用中表现出色，并通常用于提高模型对人脸图像的特征表征学习。

ArcFace Loss 的设计旨在增加类别之间的角度间隔，从而提高对不同类别之间的区分度。

ArcFace Loss 引入了一个额外的角度间隔（angular margin），通过改变角度

间隔的余弦值，使得类别之间的特征表示更有区分度。以下是 ArcFace Loss 的数学表达式：

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{e^{s\cos(\theta_{y_i}+m)}}{e^{s\cos(\theta_{y_i}+m)}+\sum_{j\neq y_i}e^{s\cos(\theta_{y_i})}}\right) \tag{3}$$

其中：

N 是批次中样本的数量。

s 是尺度因子（scale factor），用于控制类别之间的角度间隔。

m 是角度间隔的余弦值（通常称为 margin）。

θ_i 是 i 对应类别的特征向量与权重向量的夹角。

关键的思想是，对于每个样本，ArcFace Loss 通过引入额外的余弦值调整，强制类别之间的角度分布更加开放，从而提高特征的区分性。ArcFace Loss 的应用通常需要使用带有大量参数的深度卷积神经网络（如 ResNet）进行端到端的训练。

```python
class ArcFaceLoss(nn.Module):
    def __init__(self, embedding_size, num_classes, margin=0.5, scale=30.0):
        super(ArcFaceLoss, self).__init__()
        self.embedding_size = embedding_size
        self.num_classes = num_classes
        self.margin = margin
        self.scale = scale
        self.weight = nn.Parameter(torch.FloatTensor(num_classes, embedding_size))
        nn.init.xavier_uniform_(self.weight)

    def forward(self, x, labels):
        # 归一化输入特征向量
        x = F.normalize(x, p=2, dim=1)
        # 归一化权重向量
        w = F.normalize(self.weight, p=2, dim=0)
        print()
        # 计算余弦相似度
        cosine = F.linear(x, w)
        # 计算角度
        theta = torch.acos(torch.clamp(cosine, -1.0 + 1e-7, 1.0 - 1e-7))
        # 计算目标角度的余弦值
        target_logits = torch.cos(theta + self.margin)
        # 使用已经编码的标签进行计算
        output_logits = target_logits.gather(1, labels.view(-1, 1))
        # 缩放 logits
        output_logits *= self.scale
        # 计算交叉熵损失
        loss = F.cross_entropy(output_logits, labels)
        return loss
```

6.3.2 BCEWithLogitsLoss

BCEWithLogitsLoss 是二分类任务中常用的损失函数，特别适用于神经网络的输出是未经过激活函数（例如 sigmoid）的 logits 的情况。该损失函数结合了二元交叉熵损失（Binary Cross Entropy Loss）和 sigmoid 激活函数，可以方便地处理二分类问题。

损失函数的计算公式为：

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\sigma(\hat{y}_i)) + (1-y_i)\log(1-\sigma(\hat{y}_i)) \quad (4)$$

其中：

N 是样本数量。

y 是实际标签。

$\hat{y}$ 是模型的输出 logits。

$\sigma(x)$ 是 sigmoid 激活函数。

利用 AdamW 优化器，对比不同的损失函数，结果如下表：

loss	acc
BCEWithLogitsLoss	80.83%
ArcFace Loss	30.12%

初步分析，得出下列结论：

ArcFace Loss 主要设计用于人脸识别任务，特别是对于在类别间具有良好区分性的特征表示。在其他类型的问题上，尤其是在苹果叶子病害检测等农业问题中，ArcFace Loss 可能不是最适用的选择，原因如下：

任务不同： ArcFace Loss 是为人脸识别任务设计的，其目标是提高对不同人脸的区分度。在苹果叶子病害检测任务中，问题的性质和数据的分布可能与人脸识别有很大的不同，因此使用专门设计用于目标检测或图像分割的损失函数可能更合适。

数据分布差异： 人脸图像和植物图像的数据分布可能截然不同。在苹果叶子病害检测中，可能需要更多地关注局部特征、纹理和形状等方面，而不仅仅是全局特征，这与 ArcFace Loss 的设计可能不太匹配。

故尽管此问题属于细粒度分类问题，但在 Vit 模型上效果不佳。

同时对于处理分布不均衡的数据，采用反转类计数并按最大权重对其进行归一化，以获得与其频率成反比的类权重。使用 pos_weight 参数定义 BCEWithLogitsLoss 损失函数以传递类权重。BCEWithLogitsLoss 损失函数的 pos_weight 参数表示损失计算中正示例的权重。在二元分类问题中，pos_weight 可用于解决类不平衡问题，方法是给予正面示例比负面示例更多的权重。在多标签分类问题中，该 pos_weight 可用于解决类不平衡问题，方法是为频率较低的类赋予比频率较高的类更多的权重。

最终得出各类的权重如下表：

complex	frog_eye_leaf_spot	healthy	powdery_mildew	rust	scab
0.5549	0.2810	0.2616	1.0000	0.5677	0.2196

对应代码如下：

```
# # Compute inverse class frequency
class_weights = torch.reciprocal(torch.tensor(class_counts.values).float()) # invert the counts and convert them to floats
class_weights /= torch.max(class_weights) # normalize the weights by the maximum weight

# 于类别分布为正类和负类示例分配不同的权重
criterion = nn.BCEWithLogitsLoss(pos_weight=class_weights)
```

最终得到的正确率为：训练集正确率为 99.9%，测试集正确率为 81.5%。可见效果有所提升。

七、 数据增强

数据增强也叫数据扩增，意思是在不实质性的增加数据的情况下，让有限的数据产生等价于更多数据的价值。

数据增强可以分为，有监督的数据增强和无监督的数据增强方法。其中有监督的数据增强又可以分为单样本数据增强和多样本数据增强方法，无监督的数据增强分为生成新的数据和学习增强策略两个方向。

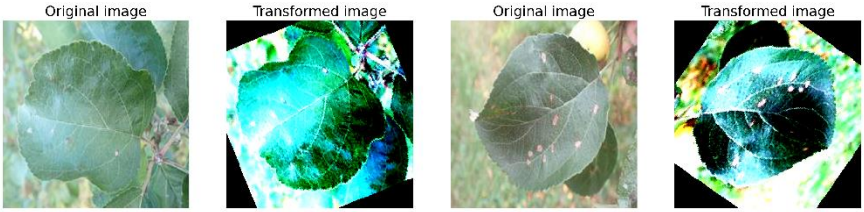
7.1 有监督的数据增强

有监督数据增强，即采用预设的数据变换规则，在已有数据的基础上进行数据的扩增，包含单样本数据增强和多样本数据增强，其中单样本又包括几何操作类，颜色变换类。

7.1.1 单样本数据增强

所谓单样本数据增强，即增强一个样本的时候，全部围绕着该样本本身进行操作，包括几何变换类，颜色变换类等。

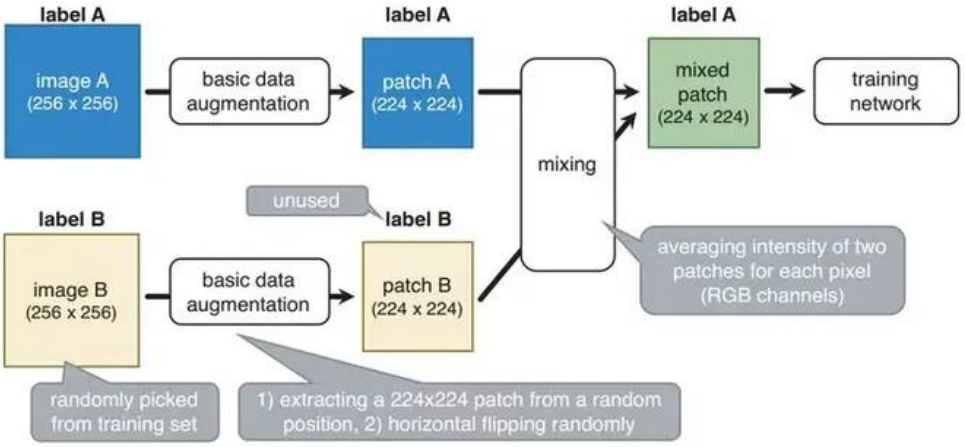
几何变换类即对图像进行几何变换，包括翻转，旋转，裁剪，变形，缩放等各类操作，下面展示其中的若干个操作。几何变换类操作没有改变图像本身的内容，它可能是选择了图像的一部分或者对像素进行了重分布。如果要改变图像本身的内容，就属于颜色变换类的数据增强了，常见的包括噪声、模糊、颜色变换、擦除、填充等等。



7.1.2 多样本数据增强

不同于单样本数据增强，多样本数据增强方法利用多个样本来产生新的样本。其中较为典型的是 SamplePairing 和 mixup。

SamplePairing 方法的原理非常简单，从训练集中随机抽取两张图片分别经过基础数据增强操作(如随机翻转等)处理后经像素以取平均值的形式叠加合成一个新的样本，标签为原样本标签中的一种。这两张图片甚至不限制为同一类别，这种方法对于医学图像比较有效。经 SamplePairing 处理后可使训练集的规模从 N 扩增到 N×N。实验结果表明，因 SamplePairing 数据增强操作可能引入不同标签的训练样本，导致在各数据集上使用 SamplePairing 训练的误差明显增加，而在验证集上误差则有较大幅度降低。



尽管 SamplePairing 思路简单，性能上提升效果可观，符合奥卡姆剃刀原理，但遗憾的是可解释性不强。

mixup 是 Facebook 人工智能研究院和 MIT 在“Beyond Empirical Risk Minimization”中提出的基于邻域风险最小化原则的数据增强方法，它使用线性插值得到新样本数据。

令(xn,yn)是插值生成的新数据，(xi,yi)和(xj,yj)是训练集随机选取的两个数据，则数据生成方式如下：

$$(x_n, y_n) = \lambda(x_i, y_i) + (1-\lambda)(x_j, y_j) \tag{5}$$

λ 的取指范围介于 0 到 1。提出 mixup 方法的作者们做了丰富的实验，实验结果表明可以改进深度学习模型在 ImageNet 数据集、CIFAR 数据集、语音数据集和表格数据集中的泛化误差，降低模型对已损坏标签的记忆，增强模型对对抗样本的鲁棒性和训练生成对抗网络的稳定性。

7.2 无监督的数据增强

无监督的数据增强方法包括两类：

(1) 通过模型学习数据的分布，随机生成与训练数据集分布一致的图片，代表方法 **GAN**。

(2) 通过模型，学习出适合当前任务的数据增强方法，代表方法 AutoAugment。

关于 **GAN(generative adversarial networks)**，它包含两个网络，一个是生成网络，一个是对抗网络，基本原理如下：

(1) G 是一个生成图片的网络，它接收随机的噪声 z，通过噪声生成图片，记做 G(z) 。

(2) D 是一个判别网络，判别一张图片是不是“真实的”，即是真实的图片，还是由 G 生成的图片。

AutoAugment 是 Google 提出的自动选择最优数据增强方案的研究，这是无监督数据增强的重要研究方向。它的基本思路是使用增强学习从数据本身寻找最佳图像变换策略，对于不同的任务学习不同的增强方法，流程如下：

(1) 准备 16 个常用的数据增强操作。

(2) 从 16 个中选择 5 个操作，随机产生使用该操作的概率和相应的幅度，

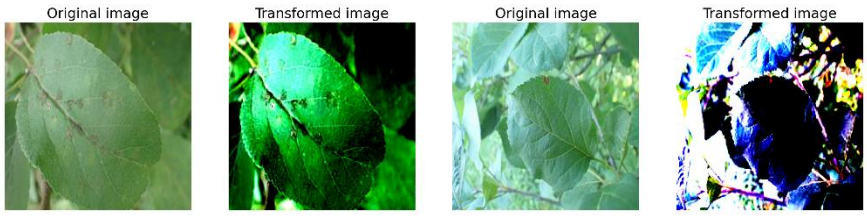
将其称为一个 sub-policy，一共产生 5 个 sub-polices。

(3) 对训练过程中每一个 batch 的图片，随机采用 5 个 sub-polices 操作中的一种。

(4) 通过模型在验证集上的泛化能力来反馈，使用的优化方法是增强学习方法。

(5) 经过 80~100 个 epoch 后网络开始学习到有效的 sub-policies。

(6) 之后串接这 5 个 sub-policies，然后再进行最后的训练。

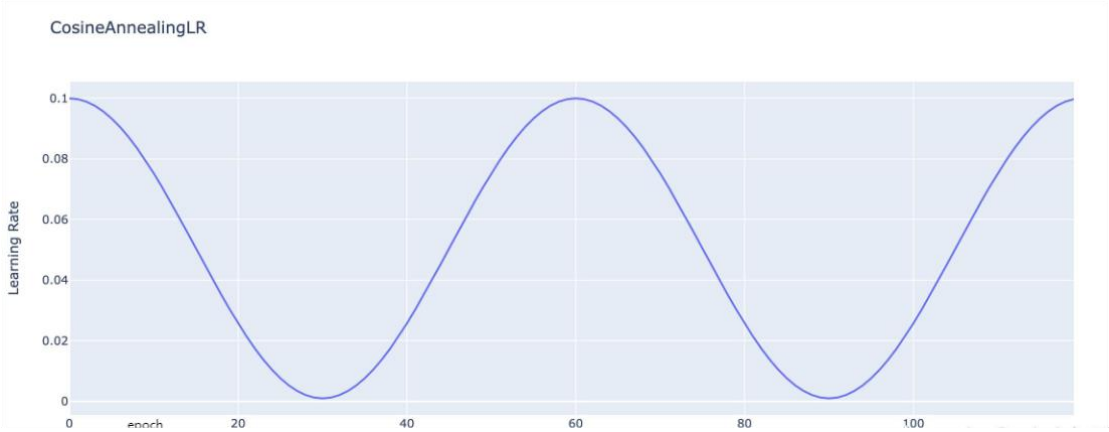


对比发现，AutoAugment 机制在此处比较合适，故后续采取此策略。

八、 余弦退火学习率调度

在深度学习训练中，学习率是一个重要的超参数，决定了模型参数在每次迭代中的更新幅度。余弦退火学习率调度器是一种调整学习率的策略，其特点是学习率在训练过程中呈余弦形状的变化，有助于在训练的后期更加精细地调整模型参数。

如果我们使得网络训练的 loss 最小，那么一直使用较高学习率是不合适的，因为它会使得权重的梯度一直来回震荡，很难使训练的损失值达到全局最低谷。所以学习率还是需要下降，可以通过余弦函数来降低学习率。余弦函数中随着 x 的增加余弦值首先缓慢下降，然后加速下降，再次缓慢下降。这种下降模式能和学习率配合，以一种十分有效的计算方式来产生很好的效果。



九、 最终呈现效果

下图为最终搭建的 ViT 模型网络整体架构：

```
ViTBase(
  (patch_embedding): PatchEmbed(
    (proj): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
    (norm): Identity()
  )
  (dropout): Dropout(p=0, inplace=False)
  (transformer): Transformer(
    (blocks): ModuleList(
      (0-11): 12 x TransformerBlock(
        (layers): ModuleList()
        (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (attn): Attention(
          (attend): Softmax(dim=-1)
          (to_qkv): Linear(in_features=768, out_features=2304, bias=True)
          (proj): Linear(in_features=768, out_features=768, bias=True)
        )
        (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): FeedForward(
          (fc1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU(approximate='none')
          (dropout): Dropout(p=0, inplace=False)
          (fc2): Linear(in_features=3072, out_features=768, bias=True)
        )
      )
    )
  )
  (pre_head_norm): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
  (head): Linear(in_features=768, out_features=6, bias=True)
)
```

模型架构解读：（具体架构见附件中“网络架构.txt”）

Patch Embedding：

PatchEmbed 模块将输入的图像分割成固定大小的图块（patches），然后通过一个卷积操作 (Conv2d) 将每个图块映射为一个高维的向量。这个映射使用了一个 kernel 大小为 (16, 16)、步幅为 (16, 16) 的卷积操作，输出维度为 768。

Dropout：

一个 Dropout 操作，用于在模型训练过程中进行随机丢弃，以减少过拟合。

Transformer：

Transformer 模块包含多个 TransformerBlock，此架构中有 12 个这样的块。

每个 TransformerBlock 包含以下子模块：

Layer Normalization (norm1, norm2)： 用于对输入进行归一化。

Attention (attn)： 自注意力机制，通过学习注意力权重来捕捉输入序列中不同位置之间的关系。

FeedForward (mlp)： 一个前馈神经网络，用于对注意力输出进行非线性映射。

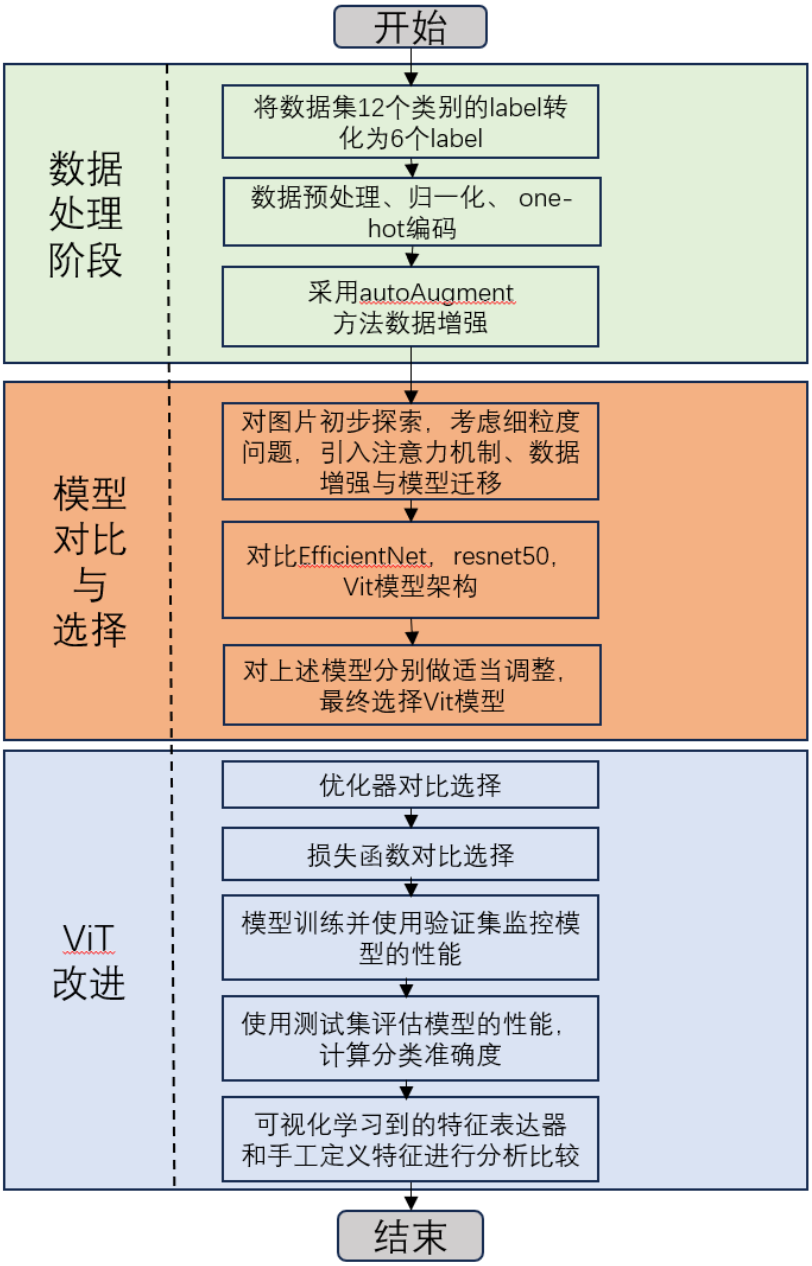
Pre-Head Layer Normalization：

另一个 Layer Normalization 操作，对 Transformer 的输出进行再次归一化。

Head：

最终的分类头部，是一个全连接层 (Linear)，将 Transformer 的输出映射到最终的输出类别。其中输出类别的数量为 6。

以下图示为本文工作主要流程：



通过上述模型，最终模型的分类效果为：**训练集正确率为 95.5%，验证集正确率为 85%，测试集正确率为 83.5%。**

具体参数见附件.pth 文件。

十、 误差分析之特征表达其可视化

深度学习误差分析是一种通过详细研究模型在测试数据上的错误来改进模型性能的方法。它有助于深入了解模型在实际应用中的表现，从而指导进一步的改进和优化。以下是进行深度学习误差分析的一些建议和步骤：

混淆矩阵分析：

构建混淆矩阵，以查看模型在每个类别上的性能。这可以帮助识别模型容易混淆的类别。

错误样本分析：

对模型在测试数据上的错误样本进行详细分析。这包括检查模型错误分类的图像，了解为什么模型在这些样本上出现错误。

可视化特征图：

可视化模型的中间层特征图，以理解模型在图像的不同部分上的关注程度。这有助于理解模型的决策过程。

误差分析是迭代深度学习中十分重要的一个环节，当训练的模型完成后，如何对模型的性能进行改进才是提分的关键点，本文主要通过热力图的方式将模型对标签关键的识别部位提取出来，这样就能很清晰的知道模型主要看到了哪些部位才将图片识别为对应的类型，当我们把识别错误的图片拿出来分析后，就可以知道数据增强的改进点和网络训练的改进点。

步骤如下：

获取模型输出： 使用已训练好的模型，通过前向传播获取模型对输入图像的输出。这通常是模型最后一层的激活值或 logits。

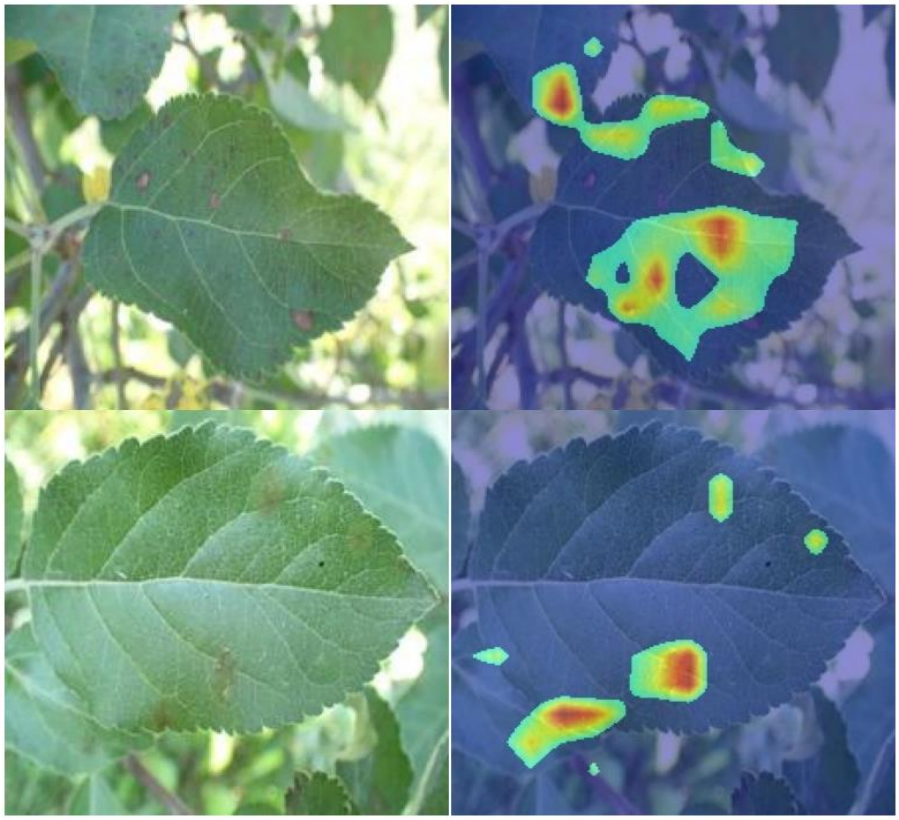
选择特定层的输出： 选择希望可视化的模型层的输出。

计算类别激活图（CAM）： 对于分类任务，可以使用类别激活图 (Class Activation Map, CAM) 或其他类似的方法来生成热力图。CAM 是通过权重的线性组合将卷积层的特征图与模型最后一层的权重关联起来的方法。

热力图后处理： 对生成的热力图进行适当的后处理，例如归一化或插值，以获得更好的可视化效果。

可视化： 使用图像处理工具或库，将生成的热力图叠加在原始图像上，以显示模型对标签关键的识别部位。

利用 pytorch_grad_cam 库完成特征图可视化步骤，并将图片保存，最终抽样效果展示为：



可以发现，网络学习到的特征与实际的病害位置之间存在一定的偏差，甚至有地方将特征识别到叶子旁边的背景，这也是网络正确率不是非常高的一大原因。

但在总体上，网络标记的特征分布于病害位置附近，说明网络的效果话不错。

十一、 心得体会

通过此次实验，让我深刻地认识到多种深度学习网络架构，如 EfficientNet，resnet50，Vit 模型等架构，并且对于模型的微调与改进部分，通过尝试不同的优化器和损失函数，最终达到了较为不错的效果。

数据集分析和标签设计： 通过对数据集的分布和各标签含义的解读，本文为复杂病的叶片设计了一个多标签分类的 one-hot 编码方式。这种设计考虑到了问题的复杂性，为模型提供了更准确的目标。

模型选择和迁移学习： 通过比较 EfficientNet，resnet50 和 Vit 模型，本文深入理解各架构的机制，并最终选择了 ViT 模型。在模型选择上，考虑到 ViT 适用于大规模数据集上预训练，作者采用了迁移学习的思想，使用在 ImageNet 数据集上预训练的 ViT-Base 模型作为基础。

优化器和损失函数选择： 本文通过对比 Adam，SGD，AdamW 三个优化器的性能，最终选择了 AdamW 作为模型优化器。在损失函数选择上，通过对比 BCEWithLogitsLoss 和 ArcFaceloss，本文认为 ArcFaceloss 可能不适配此任务，选择了 BCEWithLogitsLoss 作为损失函数。

数据增强和模型训练策略： 为了缩短模型训练时间，本文采用了降低图片分辨率的方式，并且利用 AutoAugment 策略在训练集图片上进行数据增强。为了降低过拟合的效果，采用了余弦退火学习率调度器。

模型性能分析和可视化： 通过利用 pytorch_grad_cam 库完成特征图可视化步骤，本文对误差进行了分析，同时在一定程度上评估了网络性能。这种分析和可视化有助于理解模型的决策过程和提高模型的可解释性。

总结成果和展望： 最终，本文通过所采用的方法和策略，使得模型性能有所提高，在测试集上达到了 85%左右的正确率。

十二、 附录

附件 1

介绍：全部代码

```
import os
import math
import random
from typing import Dict, List,Tuple
import requests
from pathlib import Path

import numpy as np
import matplotlib.pyplot as plt
import glob
from pathlib import Path, PurePath
```

```
import pathlib
import pandas as pd

import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn

import torchvision
from torchvision import datasets
from torchvision import transforms

from PIL import Image

from sklearn.model_selection import train_test_split

from imutils import paths

import textwrap
from tqdm import tqdm
import super_gradients
from super_gradients.common.object_names import Models
from super_gradients.training import Trainer
from super_gradients.training import training_hyperparams
from super_gradients.training.metrics.classification_metrics import Acc
uracy, Top5
from super_gradients.training.utils.early_stopping import EarlyStop
from super_gradients.training import models
from super_gradients.training.utils.callbacks import Phase
from super_gradients.common.registry import register_metric, register_m
odel,register_loss
class config:
    #  指定数据集的路径
    train_data_dir = Path('C:/Users/shuai/Desktop/plant_dataset/train/i
mages')
    val_data_dir = Path('C:/Users/shuai/Desktop/plant_dataset/val/image
s')
    test_data_dir = Path('C:/Users/shuai/Desktop/plant_dataset/test/ima
ges')
    root_dir = Path('./data')
    train_dir = root_dir.joinpath('train')
    test_dir = root_dir.joinpath('test')
    val_dir = root_dir.joinpath('val')

    # set the input height and width
    input_height = 224
    input_width = 224

    # 图像预处理使用
mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]来正则化
    # 减去数据对应维度的统计平均值，可以消除公共部分，以凸显个体之前的差异和特
征。
```

```
    IMAGENET_MEAN = [0.485, 0.456, 0.406]
    IMAGENET_STD = [0.229, 0.224, 0.225]

    image_type = '.jpg'
    batch_size = 32
    # will use the vision transformer
    model_name = 'vit_base'

    DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
    training_params = 'training_hyperparams/default_train_params'
    labels = ['complex', 'frog_eye_leaf_spot', 'healthy', 'powdery_mild
ew', 'rust', 'scab']
    num_classes = len(labels)
    checkpoints_dir = 'checkpoints'
!mkdir data
# Set the desired output dimensions
output_size = (224, 224)

# Get a list of all image file paths in the input directory
train_image_paths = list(paths.list_images(config.train_data_dir))
val_image_paths = list(paths.list_images(config.val_data_dir))
test_image_paths = list(paths.list_images(config.test_data_dir))

# Create a progress bar object
train_progress_bar = tqdm(total=len(train_image_paths), desc='Resizing
train images')

# Loop over all image file paths
for image_path in train_image_paths:
    # Load the image with PIL
    image_path=Path(image_path)
    image = Image.open(image_path)

    # Resize the image
    resized_image = image.resize(output_size)

    # Get the output file path
    output_path = config.root_dir / image_path.name

    # Save the resized image to disk
    resized_image.save(output_path)
    # Update the progress bar
    train_progress_bar.update(1)

# Close the progress bar
train_progress_bar.close()

val_progress_bar = tqdm(total=len(val_image_paths), desc='Resizing val
images')

# Loop over all image file paths
```

```
for image_path in val_image_paths:
    # Load the image with PIL
    image_path=Path(image_path)
    image = Image.open(image_path)

    # Resize the image
    resized_image = image.resize(output_size)

    # Get the output file path
    output_path = config.root_dir / image_path.name

    # Save the resized image to disk
    resized_image.save(output_path)
    # Update the progress bar
    val_progress_bar.update(1)

# Close the progress bar
val_progress_bar.close()

test_progress_bar = tqdm(total=len(test_image_paths), desc='Resizing te
st images')

# Loop over all image file paths
for image_path in test_image_paths:
    # Load the image with PIL
    image_path=Path(image_path)
    image = Image.open(image_path)

    # Resize the image
    resized_image = image.resize(output_size)

    # Get the output file path
    output_path = config.root_dir / image_path.name

    # Save the resized image to disk
    resized_image.save(output_path)
    # Update the progress bar
    test_progress_bar.update(1)

# Close the progress bar
test_progress_bar.close()
def get_split_df(csv_dir):
    '''
    This function take csv file and split it into train, valid, and tes
t
    '''
    train_df = pd.read_csv(csv_dir)
    train_df['images'] =  train_df['images'].apply(lambda x: './data/'
+ x)

    valid_df = pd.read_csv('C:/Users/shuai/Desktop/plant_dataset/val/va
```

```
l_label.csv')
    valid_df['images'] =  valid_df['images'].apply(lambda x: './data/'
+ x)

    test_df = pd.read_csv('C:/Users/shuai/Desktop/plant_dataset/test/te
st_label.csv')
    test_df['images'] =  test_df['images'].apply(lambda x: './data/' +
x)

    return train_df, valid_df, test_df
train_df, valid_df, test_df = get_split_df('C:/Users/shuai/Desktop/plan
t_dataset/train/train_label.csv')
train_df
import matplotlib.pyplot as plt
import pandas as pd

# 假设 train_df 是你的数据框，all_labels 是所有标签的数据框或序列
# 创建画布和子图
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15,7), dpi=500)
# 绘制第一个柱状图
train_df['labels'].value_counts().plot(kind='bar', ax=axes[0])
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45)  # 设
置 x 轴标签的旋转角度
# axes[0].set_xticks([])
axes[0].set_title('Train Dataset' , fontsize=16)

# 绘制第二个柱状图
all_labels.value_counts().plot(kind='bar', ax=axes[1])
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45)  # 设
置 x 轴标签的旋转角度
axes[1].set_title('All Labels', fontsize=16)
# 调整布局
plt.tight_layout()
# 显示图形
plt.savefig('output_plot.png', dpi=500)
plt.show()
train_df[train_df['labels']=='complex']
fig, axs = plt.subplots(1, 3, figsize=(15, 5),tight_layout=True, dpi=50
0)
image1 = Image.open(Path('./data/91f0af891fa147ac.jpg'))
axs[0].imshow(image1)
axs[0].set_title("complex", fontsize=25)
axs[0].axis('off')

image1 = Image.open(Path('./data/decb693131273323.jpg'))
axs[1].imshow(image1)
axs[1].set_title("complex", fontsize=25)
axs[1].axis('off')

image1 = Image.open(Path('./data/e511d31b8ce339c5.jpg'))
axs[2].imshow(image1)
```

```
axs[2].set_title("complex", fontsize=25)
axs[2].axis('off')
fig.get_figure().savefig('complex')
fig, axs = plt.subplots(1, 3, figsize=(15, 5),tight_layout=True, dpi=50
0)
image1 = Image.open(Path('./data/c1fc2e657b4a05b4.jpg'))
axs[0].imshow(image1)
axs[0].set_title("powdery_mildew", fontsize=25)
axs[0].axis('off')

image1 = Image.open(Path('./data/c88f30fb361df141.jpg'))
axs[1].imshow(image1)
axs[1].set_title("powdery_mildew", fontsize=25)
axs[1].axis('off')

image1 = Image.open(Path('./data/ddf4b843459ea10d.jpg'))
axs[2].imshow(image1)
axs[2].set_title("powdery_mildew", fontsize=25)
axs[2].axis('off')
fig.get_figure().savefig('powdery_mildew.png')
fig, axs = plt.subplots(1, 3, figsize=(15, 5),tight_layout=True, dpi=50
0)
image1 = Image.open(Path('./data/de27e42a4465a67c.jpg'))
axs[0].imshow(image1)
axs[0].set_title("rust", fontsize=25)
axs[0].axis('off')

image1 = Image.open(Path('./data/9bda896a2c98eba1.jpg'))
axs[1].imshow(image1)
axs[1].set_title("rust", fontsize=25)
axs[1].axis('off')

image1 = Image.open(Path('./data/ad32dea155d5aa05.jpg'))
axs[2].imshow(image1)
axs[2].set_title("rust", fontsize=25)
axs[2].axis('off')
fig.get_figure().savefig('rust.png')
def examine_images(df, num_images=20):
    image_paths = df['images'].sample(n=num_images, random_state=42)
    labels = df['labels'].loc[image_paths.index]

    num_rows = int(math.ceil(num_images/5))
    num_cols = 5
    fig, axs = plt.subplots(num_rows, num_cols, figsize=(30, 30),tight_
layout=True)
    axs = axs.ravel()

    for i, image_path in enumerate(image_paths):
        image = Image.open(Path(image_path))
        label = labels.iloc[i]
        axs[i].imshow(image)
```

```
        axs[i].set_title(f"Label: {label}", fontsize=25)
        axs[i].axis('off')
    plt.show()
examine_images(train_df, num_images=20)
# 初始化数据增强函数
resize = transforms.Resize(size=(config.input_height,config.input_width
))  # 将输入图片转化成 224×224 的输入特征图
make_tensor = transforms.ToTensor()  # 将图片转化成张量类型
normalize = transforms.Normalize(mean=config.IMAGENET_MEAN, std=config.
IMAGENET_STD)  # 逐 channel 的对图像进行标准化（均值变为 0，标准差变为 1），可
以加快模型的收敛
center_cropper = transforms.CenterCrop((config.input_height,config.inpu
t_width))  # 从图片中心开始沿两边裁剪，裁剪后的图片大小为（size*size）
random_horizontal_flip = transforms.RandomHorizontalFlip(p=0.75)  # 图片
以 0.75 概率水平翻转
random_vertical_flip = transforms.RandomVerticalFlip(p=0.75)  # 图片以
0.75 概率上下翻转
random_rotation = transforms.RandomRotation(degrees=90)  # 对图片旋转随机
90°
random_crop = transforms.RandomCrop(size=(200,200))  # 对图像随机裁剪出尺
寸为 size 的图片
augmix = transforms.AugMix(severity = 3, mixture_width=3, alpha=0.2)  #
 混合增强
auto_augment = transforms.AutoAugment()
random_augment = transforms.RandAugment()

# 对训练集做数据增强
train_transforms = transforms.Compose([

#   auto_augment,


# #   random_augment,
#     center_cropper,
#     random_horizontal_flip,
#     random_vertical_flip ,
#     random_rotation,
#     augmix,
#     auto_augment,
    make_tensor,
    normalize
])

val_transforms = transforms.Compose([make_tensor, normalize])
img1 = Image.open(train_df['images'].sample(n=1).iloc[0])
img_array1 = apply_transform(img1, train_transforms)
img2 = Image.open(train_df['images'].sample(n=1).iloc[0])
img_array2 = apply_transform(img2, train_transforms)
def visualize_transform(image1: np.ndarray, image2: np.ndarray, origina
l_image1: np.ndarray=None, original_image2: np.ndarray = None) -> None:
    fontsize = 18
```

```
    # Create a plot with 1 row and 2 columns.
    f, ax = plt.subplots(1, 4, figsize=(20, 5), dpi=200)

    # Show the original image in the first column.
    ax[0].imshow(original_image1)
    ax[0].set_title('Original image', fontsize=fontsize)
    ax[0].axis('off')

    # Show the transformed image in the second column.
    ax[1].imshow(image1)
    ax[1].set_title('Transformed image', fontsize=fontsize)
    ax[1].axis('off')

    ax[2].imshow(original_image2)
    ax[2].set_title('Original image', fontsize=fontsize)
    ax[2].axis('off')

    # Show the transformed image in the second column.
    ax[3].imshow(image2)
    ax[3].set_title('Transformed image', fontsize=fontsize)
    ax[3].axis('off')

# 转换为 uint8 格式的 numpy 数组，以便能够在 imshow 中显示
img_array1_display = (img_array1 * 255).astype(np.uint8)
img_array2_display = (img_array2 * 255).astype(np.uint8)

# 调用 visualize_transform 函数
visualize_transform(img_array1_display, img_array2_display, np.array(im
g1), np.array(img2))
# 保存图像
plt.savefig('transformed_images.png')
plt.show()
def encode_label(labels, class_list):   # 对标签进行编码(one-hot)
    target = torch.zeros(len(class_list))
    for label in labels:
        idx = class_list.index(label)  # 对应位置置1
        target[idx] = 1
    return target

def decode_label(encoded_label, class_list):
    decoded = [class_list[i] for i, val in enumerate(encoded_label) if
val == 1]
    return decoded
class PlantDataset(Dataset):  # 定义了PyTorch 数据集（Dataset）的类，名为
PlantDataset。这个类用于加载和处理植物图像数据集。
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform

    def __len__(self):  # 返回数据集的样本数量
        return len(self.dataframe)
```

```
    def __getitem__(self, idx):  # 从DataFrame中获取图像路径和标签，加载图像，对标签进行编码
        image_path = self.dataframe['images'].iloc[idx]
        image = Image.open(image_path)
        labels = self.dataframe.iloc[idx]['labels'].split(' ')
        encoded_labels = encode_label(labels, config.labels)
        if self.transform:
            image = self.transform(image)
        return image, encoded_labels
train_dataset = PlantDataset(train_df, transform = train_transforms)
val_dataset = PlantDataset(valid_df, transform = val_transforms)
test_dataset = PlantDataset(test_df, transform = val_transforms)

train_loader = DataLoader(train_dataset, batch_size=config.batch_size, shuffle=True)  # 加载数据集并对数据进行打乱
val_loader = DataLoader(val_dataset, batch_size=config.batch_size)
test_loader = DataLoader(test_dataset, batch_size=config.batch_size)
#get the class counts
class_counts = all_labels.value_counts()[config.labels]
class_counts
# # Compute inverse class frequency
class_weights = torch.reciprocal(torch.tensor(class_counts.values).float()) # invert the counts and convert them to floats
class_weights /= torch.max(class_weights) # normalize the weights by the maximum weight

# 于类别分布为正类和负类示例分配不同的权重
criterion = nn.BCEWithLogitsLoss(pos_weight=class_weights)
from torchmetrics import Metric
import torch
from super_gradients.common.registry import register_metric

@register_metric('my_accuracy')
class MyAccuracy(Metric):
    # Constructor method that takes in a number of classes as an argument
    def __init__(self, num_classes=config.num_classes):
        # Calls the constructor of the parent class Metric
        super().__init__()
        # Adds two states to the instance of the class: "correct" and "total"
        self.add_state("correct", default=torch.tensor(0), dist_reduce_fx="sum")
        self.add_state("total", default=torch.tensor(0), dist_reduce_fx="sum")

    # A method that takes in predictions and target values and updates the state of the class
    def update(self, preds: torch.Tensor, target: torch.Tensor):
        # Applies a threshold of 0.5 to the predictions and converts th
```

```
em to integers
        preds = (torch.sigmoid(preds) > 0.50).int()
        self.correct += torch.sum((preds == target).all(dim=1))
        self.total += target.shape[0]

    # A method that calculates and returns the accuracy
    def compute(self):
        # Calculates the accuracy as the ratio of the number of correct
 predictions to the total number of predictions
        return self.correct.float() / self.total
training_params =  training_hyperparams.get(config.training_params)
training_params['lr_schedule_function']='CosineAnnealingLR'
# To reduce clutter in the notebook I've turned the verbosity off, you
can turn it on to see the full output
training_params["train_metrics_list"] = ['my_accuracy']
training_params["valid_metrics_list"] = ['my_accuracy']
training_params["metric_to_watch"] = "my_accuracy"

# Set the silent mode to True to reduce clutter in the notebook, you ca
n turn it on to see the full output
training_params["silent_mode"] = False
# training_params["optimizer"] = 'AdamW'
training_params["optimizer"] = 'Adam'
training_params['average_best_models'] = True
training_params['ema'] = True
training_params["criterion_params"] = {'smooth_eps': 0.20}
training_params["max_epochs"] = 30
training_params["initial_lr"] = 0.00001
training_params["loss"] = criterion
model = models.get(config.model_name, num_classes=config.num_classes, p
retrained_weights='imagenet')
full_model_trainer = Trainer(experiment_name='test1_Experiment', ckpt_r
oot_dir=config.checkpoints_dir)

full_model_trainer.train(model=model,
              training_params=training_params,
              train_loader=train_loader,
              valid_loader=val_loader)
full_model_trainer.test(model=best_full_model,
            test_loader=test_loader,
            test_metrics_list=['my_accuracy'])
best_full_model
import sys
from contextlib import contextmanager

@contextmanager
def stdout_redirected(to=None):
    """
    上下文管理器，用于临时将 stdout 重定向到文件或控制台。
    使用方法：`with stdout_redirected(to='output.txt'):`
    """
```

```
    if to is None:
        yield
    else:
        sys.stdout.flush()
        original_stdout = sys.stdout
        with open(to, 'w') as file:
            sys.stdout = file
            try:
                yield file
            finally:
                sys.stdout = original_stdout

# 假设你已经创建了PyTorch模型，并将其命名为`model`
# 下面演示如何使用`torch.summary`函数并将输出保存到文件
from torchsummary import summary

output_file = "model_summary.txt"

with stdout_redirected(to=output_file):
    summary(model, input_size=[(3, 224, 224)], batch_size=-
1, device="cuda")
print(f"模型摘要已保存到{output_file}文件中。")
import cv2
import numpy as np
import torch

from pytorch_grad_cam import GradCAM, \
                             ScoreCAM, \
                             GradCAMPlusPlus, \
                             AblationCAM, \
                             XGradCAM, \
                             EigenCAM, \
                             EigenGradCAM, \
                             LayerCAM, \
                             FullGrad

from pytorch_grad_cam import GuidedBackpropReLUModel
from pytorch_grad_cam.utils.image import show_cam_on_image, preprocess_
image

model.eval()

# 判断是否使用 GPU 加速
use_cuda = torch.cuda.is_available()
if use_cuda:
    model = model.cuda()
def reshape_transform(tensor, height=14, width=14):
    # 去掉cls token
    result = tensor[:, 1:, :].reshape(tensor.size(0),
    height, width, tensor.size(2))
```

```
    # 将通道维度放到第一个位置
    result = result.transpose(2, 3).transpose(1, 2)
    return result
cam = GradCAM(model=model,
              target_layers=[model.transformer.blocks[-1].norm1],
              # 这里的target_layer要看模型情况,
              # 比如还有可能是:target_layers = [model.blocks[-1].ffn.norm]
              use_cuda=use_cuda,
              reshape_transform=reshape_transform)
# 读取输入图像
image_path = "./data/fab3f2b1c0d2a982.jpg"
rgb_img = cv2.imread(image_path, 1)[:, :, ::-1]
img = np.array(rgb_img)
rgb_img = np.float32(img)/255  #归一化
# 预处理图像
input_tensor = preprocess_image(rgb_img,
mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])

# 看情况将图像转换为批量形式
# input_tensor = input_tensor.unsqueeze(0)
if use_cuda:
    input_tensor = input_tensor.cuda()
# 计算 grad-cam
target_category = None # 可以指定一个类别,或者使用 None 表示最高概率的类别
grayscale_cam = cam(input_tensor=input_tensor, targets=target_category)
grayscale_cam = grayscale_cam[0, :]

# 将 grad-cam 的输出叠加到原始图像上
visualization = show_cam_on_image(rgb_img, grayscale_cam)

# 保存可视化结果
cv2.cvtColor(visualization, cv2.COLOR_RGB2BGR, visualization)
cv2.imwrite('cam.jpg', visualization)
from typing import Tuple
import requests
import torchvision
import random
import textwrap

def pred_and_plot_image(image_path: str,
                        subplot: Tuple[int, int, int],  # subplot tuple
 for `subplot()` function
                        ground_truth:str = None,
                        model: torch.nn.Module = best_full_model,
                        image_size: Tuple[int, int] = (config.INPUT_HEI
GHT, config.INPUT_WIDTH),
                        transform: torchvision.transforms = None,
                        device: torch.device=config.DEVICE):

    if isinstance(image_path, pathlib.WindowsPath):
```

```
        img = Image.open(image_path)
    else:
        img = Image.open(requests.get(image_path, stream=True).raw)

    # create transformation for image (if one doesn't exist)
    if transform is None:
        transform = transforms.Compose([
            transforms.Resize(image_size),
            transforms.ToTensor(),
            transforms.Normalize(mean=config.IMAGENET_MEAN,
                                 std=config.IMAGENET_STD),
        ])
    transformed_image = transform(img)

    # make sure the model is on the target device
    model.to(device)
    # turn on model evaluation mode and inference mode
    model.eval()
    with torch.inference_mode():
        # add an extra dimension to image (model requires samples in [batch_size, color_channels, height, width])
        transformed_image = transformed_image.unsqueeze(dim=0)
        # make a prediction on image with an extra dimension and send it to the target device
        target_image_pred = model(transformed_image.to(device))
        # apply sigmoid to predictions, return 1 at each index where greater than threshold
    preds = (torch.sigmoid(target_image_pred) > 0.50).int()
    # from tensor to list
    preds = torch.round(preds).squeeze().tolist()
    # convert float to ints
    preds = [int(i) for i in preds]
    predicted_labels = decode_label(preds, config.labels)

    # plot image with predicted label
    plt.subplot(*subplot)
    plt.imshow(img)
    if isinstance(image_path, pathlib.WindowsPath):
        # actual label
        title = f"Ground Truth: {ground_truth} | Pred: {' '.join(predicted_labels)}"
    else:
        title = f"Pred: {' '.join(predicted_labels)}"
    plt.title("\n".join(textwrap.wrap(title, width=20)))  # wrap text using textwrap.wrap() function
    plt.axis(False)


def plot_random_test_images(model, test_images):
    num_images_to_plot = 30
```

```
    # extract image paths and labels from the dataframe
    test_image_paths = test_images['images'].tolist()
    test_image_labels = test_images['labels'].tolist()

    # sample k image paths and labels
    random_indices = random.sample(range(len(test_image_paths)), k=num_
images_to_plot)
    test_image_path_sample = [pathlib.WindowsPath(test_image_paths[i])
for i in random_indices]
    test_image_label_sample = [test_image_labels[i] for i in random_ind
ices]

    # set up subplots
    num_rows = int(np.ceil(num_images_to_plot / 5))
    fig, ax = plt.subplots(num_rows, 5, figsize=(15, num_rows * 3))
    ax = ax.flatten()

    # Make predictions on and plot the images
    for i, image_path in enumerate(test_image_path_sample):
        label = test_image_label_sample[i]
        pred_and_plot_image(model=model,
                            image_path=image_path,
                            ground_truth=label,
                            subplot=(num_rows, 5, i+1),
                            image_size=(config.input_height, config.inp
ut_width))

    # adjust spacing between subplots
    plt.subplots_adjust(wspace=1)
    plt.show()
plot_random_test_images(best_full_model, test_df)
```