

基于 Inception net 及 VIT 解决 Plant Pathology 2021 分类任务

刘付伟嘉^{*}, 梁小丹老师[†]

中山大学 智能科学与技术 21312193

【摘要】本次实验的目的是参照已有深度学习网络模型,在其基础上改进设计一个神经网络,并使用该模型对Plant Pathology-2021数据集进行微调训练,对于给定的 test 测试集中的图片,准确地预测出其类别,故我在kaggle上查阅了诸多资料,对已有的各种模型都有大致的了解,最终对 Inception net v3^[1]以及 VIT^[2]模型进行相应的构建和修改,在数据集上进行微调后,测试的准确率约为 80% 上下。

【关键词】深度学习, Inception net, VIT, Plant Pathology-2021

1 引言

这篇文章介绍了使用 Inception net v3 以及 VIT 模型,并进行网络架构的改进,完成多标签分类任务,并且对实验结果进行可视化和两个模型的性能进行对比分析。其关键思想在于迁移学习^[3]和多标签分类任务^[4]的性质。

2 介绍

2.1 现实背景

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

尽管基于计算机视觉的模型在植物疾病识别方面显示出了前景,但仍有一些局限性需要解决。不同苹果品种或栽培中新品种的单一疾病视觉症状的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获环境的差异,例如,叶片颜色和叶片形态、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。

2.2 数据样本分析

1. scab(黑斑病) 是一种影响多种植物的真菌性病害,特别是水果和蔬菜。例如,苹果、葡萄、马铃薯等植物都可能受到黑斑病的影响。这种病害在植物的叶片上形成黑褐色的斑点,如图1所示。



图 1 scab

2. healthy (健康叶片) 如图2所示



图 2 healthy

3. frog_eye_leaf_spot 是一种真菌性病害. 这个名

实验时间: 2023 年 12 月 9 日

报告时间: 2023 年 12 月 9 日

[†]指导教师

*学号: 21312193

*E-mail: liufwj5@mail.sysu.edu.cn

字来源于叶片上出现的病斑，其外观类似青蛙眼睛的模样。病斑呈圆形或椭圆形，中间是深色的区域，周围有较亮的环形，中央通常是紫色、红褐色或黑色，而周围环形区域则可能是黄色或橙色，斑块通常分布在叶片的表面，特别是叶片的基部和中央区域。如图3所示



图 3 frog_eye_leaf_spot

4. rust (锈病)。该病害主要由锈菌引起，通常会在叶片表面形成带有橙色、红色或棕色小点的斑点。这些小点会逐渐扩大，形成锈色的斑块，给人一种锈蚀的印象。如图4所示



图 4 rust

5. complex，具有太多疾病而无法直观分类的非健康叶片将具有该类别。如图5所示

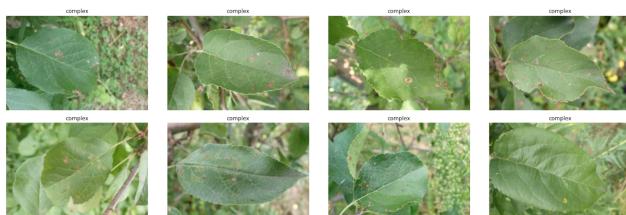


图 5 complex

6. powdery_mildew (白粉病) 是一种真菌性病害。该病害通常由白粉病真菌引起，其特征是在植物叶片表面形成白色的粉末状物质。受感染的叶片可能会出现变形和翘曲的情况。如图6所示



图 6 powdery_mildew

2.3 任务特点

1. 迁移学习

迁移学习是指把从一个任务中学到的知识应用于另一个相关的任务。其核心思想是通过利用一个任务上学到的知识来改善在一个相关任务上的学习性能。

2. 多标签分类

多标签分类是一种机器学习任务，其中每个样本可以分配给多个标签。与传统的单标签分类任务不同，多标签分类考虑一个样本可能属于多个类别或具有多个属性。且标签之间可能存在相互关系，一个标签的存在可能与其他标签的存在有关。具体到本次任务中，不健康的叶片可能同时拥有多种疾病，不同疾病之间还有可能存在依赖关系等。

前期个人尝试按数据集中的标签种类，划分成 12 类，使用单标签分类模型去完成该任务，实验效果不好，原因可能是不能将疾病按独立割裂的方式来区分，当模型判断该叶片既有 A 类病又有 B 类病时，而标签却是 C 类 (A 类 +B 类) 的话，模型可能就无法真正理解到图像的特征，使得准确率下降。此外，由于前期对题目理解错误，我还错将多类的标签归为其中的第一类标签，从而得到六个标签，在数据分布极不合理的情况下进行了单标签分类，得到的效果也很差。

3. One-hot 编码

One-hot 编码是一种常用的将分类变量表示为二进制向量的方法。在深度学习任务中，经常需要处理分类数据，而计算机更喜欢处理数字。因此，需要将分类变量转换为计算机易于处理的形式。其编码的步骤如下：

- (a) 确定类别数量: 确定分类变量中不同类别的数量。
- (b) 为每个类别分配一个整数标识: 给每个类别分配一个唯一的整数标识, 通常从 0 开始递增。
- (c) 创建二进制向量: 对每个样本, 创建一个与类别数相同长度的二进制向量, 对应整数标识的位置设置为 1, 其余设置为 0。

这样, 每个类别都用一个唯一的二进制编码表示。

3 实验过程

3.1 前期调研

通过在Github和paperswithcode上查阅各种模型和资料, 以及在各大视频网站上学习相关模型, 我对近年来的一些图像分类模型的结构框架都有了一定的理解, 例如 Lenet^[5], Resnet^[6], VGG^[7], VIT^[8]等, 对其结构设计及目的也有了初步掌握。

3.2 模型选择

在当前任务中, 我观察到数据集中的叶片位置、大小和方向等信息呈现较大的方差, 分布不均匀。若简单地采用朴素的卷积神经网络并进行深度堆叠, 可能导致模型无法有效地学习全画幅中的特征, 因为该模型可能会在处理不同位置和尺寸的叶片时面临困难。

因此我认为, 在同一层使用不同大小的卷积核是必要的, 以便对图像中具有不同位置和大小分布的叶片进行有效的特征提取。这样的操作将有助于使模型更集中地学习训练集的特征, 并提高其学习效率。

于是我先搜索使用相关结构的网络模型, 使得其更适合本次任务。正如梁老师在课上所说, 没有好与坏的模型, 只有适合与不适合的模型。最终, 我选择 Inception net 模型作为我本次搭建模型的基本框架。并且在此基础上, 我还尝试使用 VIT 模型来完成本次任务, 并对比两者的性能差异。

3.3 Inception netv3

3.3.1 提出初衷

Inception 系列模型提出的初衷主要为了解决 CNN 分类模型的两个问题:

1. 如何增加神经网络深度且同时提升分类性能。
与简单的 VGG 网络不同, 我们希望通过增加深度来持续改善性能, 而不陷入性能饱和的瓶颈。VGG 等经典结构在一定深度后容易面临性能瓶颈, 这可能是由于过度参数化、训练数据不足导致的过拟合以及计算昂贵的大型卷积运算的累积效应。
2. 如何在努力提高或维持分类网络准确率的同时, 有效地减少计算开销和内存开销。

在这两个问题中, Inception net 尤其关注后者, 毕竟在移动互联网大行天下的今天, 如何将复杂的 CNN 模型部署在计算与存储资源均有限的移动端, 并使之有效地运行有着更大的实际价值。

3.3.2 结构特点

1. 并行应用不同大小卷积核

图像中包含的有用信息可能呈现极大的尺寸变化。由于信息位于图像中位置的广泛变化, 选择适当的卷积核大小变得异常复杂。较大的卷积核有助于捕获全局信息, 而较小的卷积核则更适合获取局部信息。Inception 网络在神经网络的层与层之间引入了卷积操作的扩展, 这是它与其他神经网络模型最显著的区别。通过采用多尺寸的卷积核, Inception 网络能够实现对不同尺度感受野的融合, 最终通过拼接的方式使不同尺度的特征得到综合。如图7所示, Inception 网络以并行方式应用不同大小的卷积核进行特征提取, 从而使其对图像信息的综合提取更为全面和准确。

2. 大 kernel 分解为多个小 kernel 的累加

如图8所示, 将一个 5x5 的 conv 分解为了两个累加在一块的 3x3 conv。如此可以有效地只使用约 $(3 \times 3 + 3 \times 3) / (5 \times 5) = 72\%$ 的计算开销。此外还有将 1*3 和 3*1 叠加替换 3*3 卷积, 都有效地减少了计算量。

3.3.3 实验环境和设备

实验设备如图9所示:

3.3.4 读取数据

调用 ImageDataGenerator 函数, 对数据进行预处理, 并对数据集的标签进行 one-hot 编码表示, 用于多分类任务;

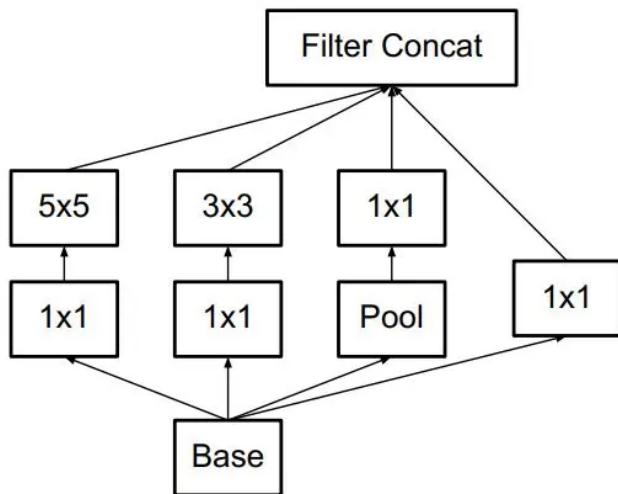


图 7 Inception net 采用不同大小卷积核

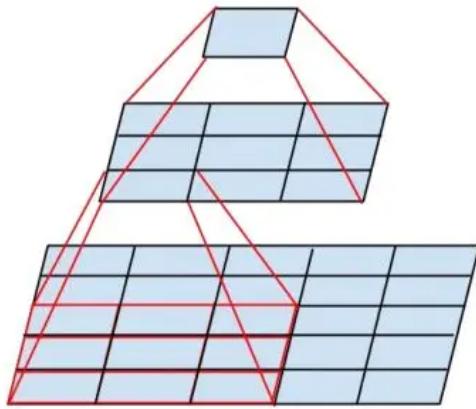


图 8 大 kernel 转换为小 kernel 叠加

```

1 # 设置图像大小
2 img_size=(128,128)
3 # 创建图像数据生成器，用于数据增强
4 datagen = ImageDataGenerator(rescale=1./255,
5 zoom_range=0.2,
6 horizontal_flip=True)
7
8 # 创建训练数据生成器
9 train_generator=datagen.flow_from_dataframe(
10 dataframe=df_train,
11 directory=train_path, # 图像文件的目录路径
12 x_col="images", # DataFrame中包含文件名的列
13 y_col="labels", # 标签列
14 batch_size=64, # 批处理大小
15 seed=42, # 随机种子，确保可重复性
16 shuffle=True, # 每个epoch后打乱数据
17 class_mode="categorical", # one-hot编码
18 target_size=img_size) # 图像目标大小
  
```

下面是关键代码的具体分析：

镜像 TensorFlow 2.9.0 Python 3.8(ubuntu20.04) Cuda 11.2 [更换](#)
 GPU RTX 3080 Ti(12GB)* 1 [升降配置](#)
 CPU 12 vCPU Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz

图 9 实验设备

1. 数据增强

为了提升模型的鲁棒性，并防止模型对训练数据的过度拟合，训练数据需要增强并扩充。因此，我通过查阅相关网页和资料，发现在TensorFlow Keras 中就集成有用于图像数据增强的实用函数 `ImageDataGenerator`。

在参数设置上，`rescale=1./255` 用于归一化图像像素；`zoom_range=0.2` 用于随机缩放图像范围；`horizontal_flip=True` 用于随机水平翻转图像；通过对图像的增强处理，增加了训练样本的多样性，模拟了不同视角和尺度的图像。

2. one-hot 编码

`flow_from_dataframe` 函数用于生成训练数据，其中”`class_mode`” 参数用于设置标签格式，这里设置为 `categorical`，在这种情况下，标签会被转换为 one-hot 编码的形式。每个标签将被表示为一个向量，只有与样本对应的类别索引的位置为 1，其余位置为 0。

3.3.5 总体结构

通过调用 tensorflow 中的 `keras.applications` 库中已有的模型，我在模型的基础上再继续添加了一些层，代码如下：

```

1 # 使用 InceptionV3 模型加载预训练权重，指定输入
2 # 图像的形状为 (128, 128, 3)
3 model_pretrained = InceptionV3(weights='
4     imagenet',
5 include_top=False,
6 input_shape=(128,128,3))
7 # 创建序贯模型
8 model=keras.models.Sequential()
9 # 添加预训练的 InceptionV3 模型作为第一层
10 model.add(model_pretrained)
11 # 将模型展平为一维数组
12 model.add(keras.layers.Flatten())
13 # 添加全连接层，激活函数为 relu
14 model.add(keras.layers.Dense(300, activation="relu"))
15 # 添加 Dropout 层，防止过拟合，丢弃率为 20%
16 model.add(keras.layers.Dropout(0.2))
  
```

```

15 # 添加全连接层
16 model.add(keras.layers.Dense(100,activation='
17     relu'))
18 # 再次添加 Dropout 层
19 model.add(keras.layers.Dropout(0.2))
20 # 添加输出层, 包含 6 个神经元, 使用 sigmoid 激
21    活函数
22 model.add(keras.layers.Dense(6,activation="
23     sigmoid"))

```

该模型的设计采用了迁移学习的思想，使用了预训练的 InceptionV3 模型来提取图像的特征，然后在其基础上构建了一个新的全连接神经网络（FCNN）层来执行特定的任务。下面是对每个部分的分析设计：

1. 使用预训练的 InceptionV3 模型：

InceptionV3 经过在大规模图像数据集（ImageNet）上的训练，学到了对图像特征的高度抽象的表示。通过加载预训练权重，模型能够捕捉图像中的复杂模式和层次化特征。将其作为第一层，可以有效地利用这些学到的特征，来为后续的分类做出巨大贡献。

2. 全连接层的设计：

模型在 InceptionV3 之后添加了全连接层，用于将从卷积层提取的高级特征映射到具体的任务空间。这里使用了两个全连接层，一个包含 300 个神经元，另一个包含 100 个神经元。我在这些层的激活函数中选择了 relu 函数，这是因为 relu 是一个常用于隐藏层的激活函数，具有非线性特性，有助于模型学习复杂的非线性映射。

3. Dropout 层的引入：

为了防止过拟合，我在模型中引入了 Dropout 层。Dropout 是一种正则化技术，通过在训练过程中随机丢弃一部分神经元，减少了神经元之间的依赖关系，有助于提高模型的泛化能力。这里设置了两个 Dropout 层，每个的丢弃率为 20%。

4. 输出层的设计：

因为本实验任务的分类数量为 6 类，因此设计输出层是一个具有 6 个神经元的全连接层，使用 sigmoid 激活函数。每个神经元对应

一个类别，输出该类别的概率。选择 sigmoid 激活函数是因为其适用于多标签分类问题，可以输出每个类别的独立概率。

总体来说，这个模型的设计充分利用了预训练的 InceptionV3 模型的优势，通过迁移学习在具体任务上进行微调。全连接层引入了模型对任务特定特征的学习能力，而 Dropout 层有助于提高模型的鲁棒性。这种结合预训练模型和自定义顶层的方法通常在样本量较小的任务中表现较好。

进行模型设计后，调用 summary 函数，进行模型结构打印，以及可训练参数展示，如图10所示，可以看到相比于 VIT 模型的可训练参数如图18中所示，为 87459846 个，Inception net 的可训练参数为 24256958 个，可以算是较为轻量化的一个设计了。

```

# 打印模型的摘要信息
model.summary()

Model: "sequential"
-----  

Layer (type)           Output Shape        Param #
-----  

inception_v3 (Functional)    (None, 2, 2, 2048)      21802784  

flatten (Flatten)         (None, 8192)          0  

dense (Dense)            (None, 300)           2457900  

dropout (Dropout)         (None, 300)           0  

dense_1 (Dense)           (None, 100)           30100  

dropout_1 (Dropout)        (None, 100)           0  

dense_2 (Dense)           (None, 6)             606  

-----  

Total params: 24,291,390
Trainable params: 24,256,958
Non-trainable params: 34,432

```

图 10 模型设计结构

3.3.6 参数设置及编译

在训练前，对模型进行参数设置，如下代码所示：

```

1 # 训练轮数和学习率
2 epoch=100
3 learning_rate=0.01
4 # 计算学习率衰减的速率和动量参数
5 decay_rate=learning_rate/epoch
6 momentum=0.8
7 # 使用SGD优化器
8 sgd=SGD(learning_rate=learning_rate,momentum=
9     momentum,decay=decay_rate)
10 # 创建回调函数
11 callback = keras.callbacks.EarlyStopping(
12     monitor='val_loss', patience=15)

```

```

11 # 编译模型
12 model.compile(loss='binary_crossentropy',
13 optimizer='sgd',
14 metrics=[tf.keras.metrics.Precision()]) # 监测
    模型性能的评估指标

```

下面是对各类关键设置的分析：

1. SGD 优化器

随机梯度下降 (Stochastic Gradient Descent, SGD) 是一种迭代优化算法，用于最小化损失函数并更新模型参数，使其逐步趋向最优解。

其基本原理是通过计算损失函数对模型参数的梯度，并沿着梯度的负方向更新参数，以降低损失。与传统的梯度下降不同，SGD 每次迭代使用随机选择的小批量样本来估计梯度，从而提高计算效率并降低计算开销。

SGD 的更新规则如下：

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t; x^{(i)}, y^{(i)}) \quad (1)$$

其中， θ_t 是第 t 次迭代后的模型参数向量。 η 是学习率，控制每次更新的步长。 $J(\theta_t; x^{(i)}, y^{(i)})$ 是损失函数，衡量模型在样本 i 上的预测与实际标签的差异， $\nabla J(\theta_t; x^{(i)}, y^{(i)})$ 是损失函数关于模型参数的梯度

SGD 的优势在于：

- (a) 计算效率：通过使用小批量样本估计梯度，SGD 降低了计算复杂度，使得大规模数据集的训练更为可行。
- (b) 在线学习：SGD 适用于在线学习，模型可以在不断产生的数据上进行实时更新。
- (c) 适应性：SGD 对于非凸、大规模问题的优化效果较好。

此外，在 SGD 函数中，我还使用了动量参数，通过使用动量，SGD 在更新模型参数时考虑了之前的梯度信息，有助于平稳优化过程，减少参数更新的方差，从而提高训练的稳定性。

2. binary_crossentropy

binary_crossentropy 是深度学习中常用的二分类问题损失函数之一。它通常用于衡量

模型对二分类任务中每个样本的预测与实际标签之间的差异。这个损失函数基于交叉熵 (cross entropy) 的概念，其更新公式如下所示，其中， N 是样本数量， $y_{\text{true}}^{(i)}$ 是第 i 个样本的实际标签 (0 或 1)， $y_{\text{pred}}^{(i)}$ 是模型对第 i 个样本的二分类预测概率。

$$J(y_{\text{true}}, y_{\text{pred}}) = -\frac{1}{N} \sum_{i=1}^N \left[y_{\text{true}}^{(i)} \cdot \log(y_{\text{pred}}^{(i)}) + (1 - y_{\text{true}}^{(i)}) \cdot \log(1 - y_{\text{pred}}^{(i)}) \right] \quad (2)$$

实际上，binary_crossentropy 损失函数也是可以用于多分类问题的。在这种情况下，对每个类别都进行一次二元分类，使用 binary_crossentropy 计算损失，并在所有类别上进行求和与平均。

3. callback.EarlyStopping

其主要作用是在训练过程中对模型进行及早停止，以防止过拟合。当训练过程中监测的指标在一定轮数内不再改善时，EarlyStopping 会停止训练，从而避免模型在训练数据上过度拟合，提高模型的泛化能力。

在我们的代码中，monitor 用于指定监测的性能指标，这里设定为验证集上的损失或准确度。patience 指定允许指定在性能不再改善时等待的轮数。如果在设定的轮数内指标没有改善，则训练将提前停止。

3.3.7 训练

```

1 # 训练
2 model_history=model.fit(train_generator,epochs=
    epoch,validation_data=val_generator,shuffle
    =True, callbacks=[callback])

```

3.3.8 训练结果

以图像的形式，展示 epoch 轮数对模型训练过程中的损失及准确率变化，如图11所示。

最终，在测试集上进行测试，得到准确率为 80.33%，如图 12 所示。

3.3.9 特征图可视化

为了更直观地观察模型内部的卷积过程，我分别输出了从 Inception net 的第一层和第 10 层卷积后的图像，原图为图13，卷积后特征图如图14和

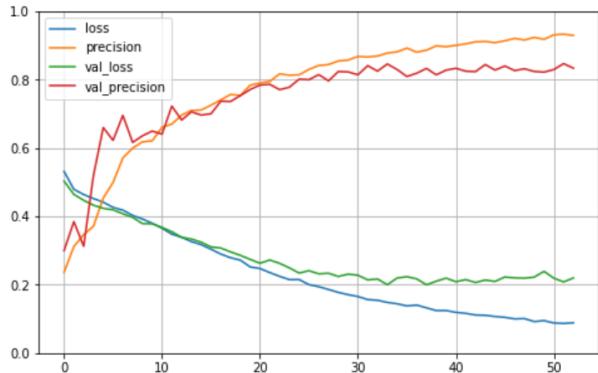


图 11 训练，验证损失及准确率变化图像

```
[14]: # 在测试集上评估模型
evaluation = model.evaluate(test_generator, steps=test_generator.samples // 1)
print("Test Accuracy: {:.2f}%".format(evaluation[1] * 100))
600/600 [=====] - 34s 56ms/step - loss: 0.2554 - precision: 0.8033
Test Accuracy: 80.33%
```

图 12 测试准确率

图15所示。可以看到，随着卷积的不断累加，其图像特征也越来越高层和抽象。



图 13 输入原图

3.3.10 混淆矩阵

为了更细致地分析模型的性能，我调用函数输出了模型的混淆矩阵，如图16所示，可以看到，对于不同类别，模型都基本上能实现对于该类的正确判断。但对于 complex 类来说，其被误判为其他类的概率相对较大。这也能够合理解释，因为 complex 类本就是人类都无法分别是哪些疾病的综合类，这样的误判对于当前这个规模不大的模型来说也是可以被原谅的。

3.3.11 数据流图

TensorBoard是TensorFlow提供的一个可视化工具，用于分析、监测和调试机器学习模型的训练

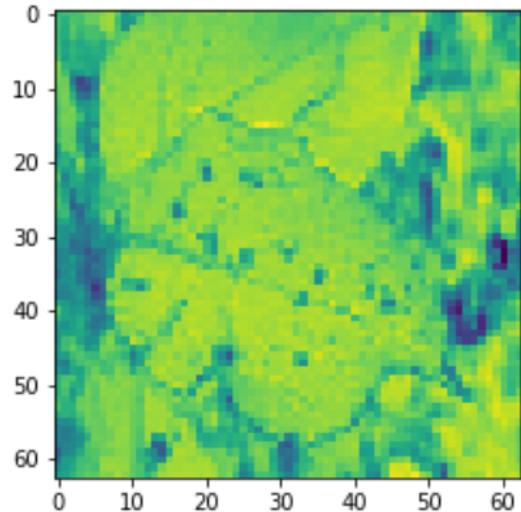


图 14 Inception net 卷积第一层

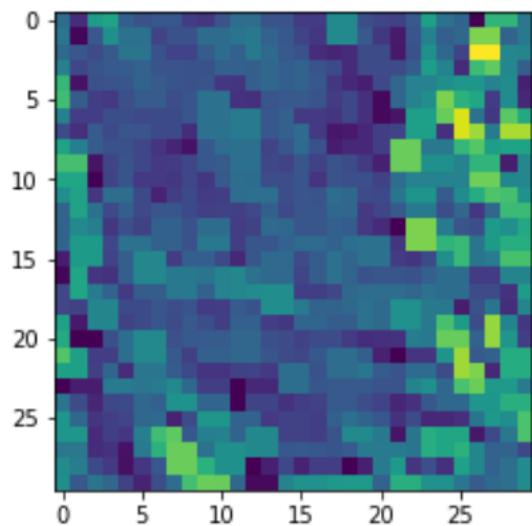


图 15 Inception net 卷积第十层

过程。它提供了许多有用的功能，包括数据流图的可视化、训练和评估指标的跟踪、嵌入式数据的查看等。

为进一步观察模型内部的数据流，在学习了如何使用 tensorboard 之后，我在代码的训练部分的回调函数设置中加入了 tensorboard 参数，在本地的网站上打开，观察模型的数据流图，如图17所示。

数据流图展示了模型的结构，图中的节点表示操作（例如，矩阵相乘、权重更新、激活函数等），边表示数据流动的路径，展示了模型中各个操作之间的依赖关系。数据流图中的边上还显示了 Tensor 的形状。其中，我注意到在 SGD 与 gradient_type

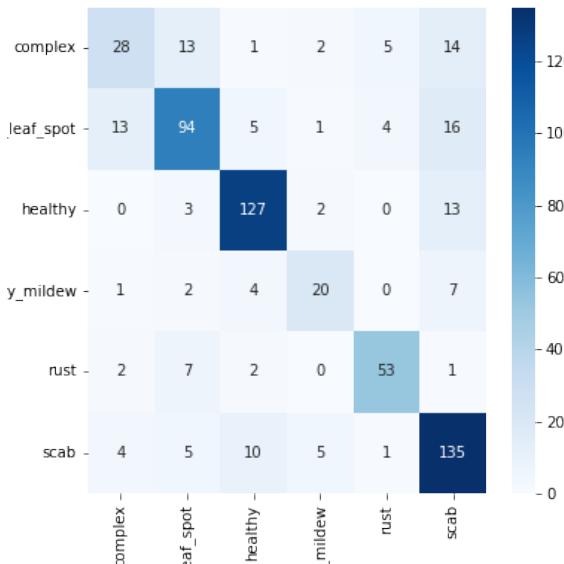


图 16 Inception 混淆矩阵

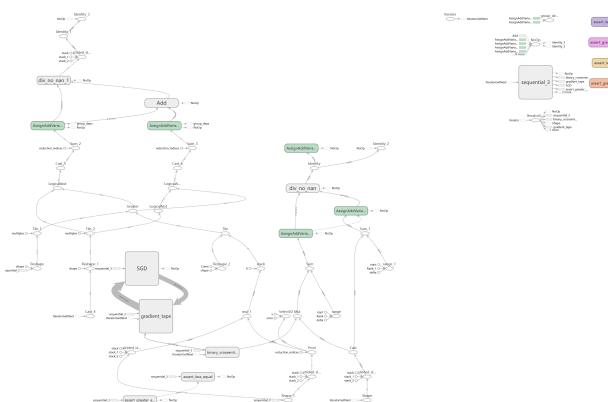


图 17 Inception net 数据流图

之间的线条尤为粗壮，这是因为在模型更新过程中，对于梯度的更新和计算是非常复杂且庞大的。

4 VIT

在尝试修改 Inception netv3 之后，我仍想使用其他模型来完成该任务，由于近年来 Transformer^[9]模型十分盛行，于是我又使用 VIT 模型来尝试解决该任务，流程以及框架与 Inception net 类似，模型结构如图18所示，而训练轮数设置为 200，具体代码结构在代码文件中可以查看。实验过程中的训练，验证损失及准确率如图19所示，最终的测试集上的效果达到 82.24%，如图20所示，比 Inception net 高大概 2 个百分点。同时我也输出了 VIT 模型的混淆矩阵，如图21所示。

[9]: model.summary()		
Model: "ViT-B-32-224"		
Layer (type)	Output Shape	Param #
patch_embedding (PatchEmbed)	(None, 49, 768)	2360064
add_cls_token (AddCLSToken)	(None, 50, 768)	768
position_embedding (AddPositionalEmbedding)	(None, 50, 768)	38400
transformer_block_0 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_1 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_2 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_3 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_4 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_5 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_6 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_7 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_8 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_9 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_10 (TransformerEncoder)	(None, 50, 768)	7087872
transformer_block_11 (TransformerEncoder)	(None, 50, 768)	7087872
layer_norm (LayerNormalization)	(None, 50, 768)	1536
extract_token (Lambda)	(None, 768)	0
mlp_head (Dense)	(None, 6)	4614

Total params: 87,459,846
Trainable params: 87,459,846
Non-trainable params: 0

图 18 VIT 模型结构

5 模型对比分析

通过对比 Inception net 和 VIT 模型的训练验证损失及准确率变化图像，如图11和19所示，可以发现，Inception net 在训练过程中，准确率和损失变化较 VIT 要更加平滑，VIT 的曲线则相对更加振荡，变化幅度较大。以下是个人分析的可能原因：

1. 模型框架不同

Inception 和 VIT 是两种不同的模型架构，它们的网络结构、层次和参数数量都不同。这些差异可能导致它们在训练过程中对梯度的响应不同，从而影响曲线的平滑度。VIT 是一种基于注意力机制的架构，其复杂度高于 Inception Net。而较复杂的模型的参数数量也更大，可能对训练数据更敏感，导致在训练过

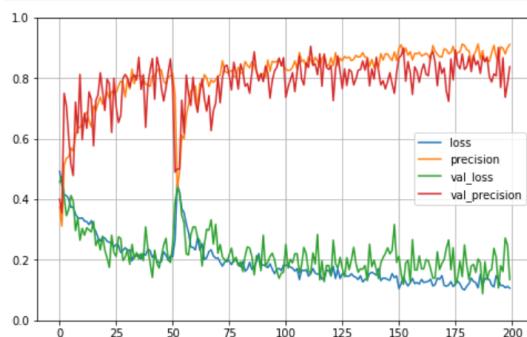


图 19 VIT 训练，验证损失及准确率

```
[13]: # 在测试集上评估模型
evaluation = model.evaluate(test_generator, steps=test_generator.samples // 1)
print('Test Accuracy: {:.2f}%'.format(evaluation[1] * 100))
600/600 [=====] - 52s 86ms/step - loss: 0.1947 - precision: 0.8224
Test Accuracy: 82.24%
```

图 20 VIT 测试准确率

程中更多的波动。

2. 超参数设置不同

两个模型在学习率、批处理大小、优化器等训练超参数的选择也会影响其曲线的平滑度。不同的超参数设置可能导致模型在训练过程中收敛速度不一，以及梯度更新幅度不同。

此外，我还发现，当训练轮数达到一定数值后，两个模型的验证集准确率都在 80% 上下波动，即使训练损失不断下降，然而测试损失和测试准确率都不会有好的提升。分析可能原因如下：

1. 过拟合：

模型过度拟合训练数据，导致在训练集上表现良好但在验证集上泛化性能差。模型可能记住了训练数据的噪声或者特定的样本，而不能很好地适应新的数据。在发现了该现象之后，我通过改变 callback 参数中的 patience 来提前结束该情况。

2. 数据不平衡：

如果训练集和验证集之间存在很大的分布差异，模型可能更容易过拟合训练数据。对此，我特意查看训练和验证集中的 label 标签并计算分布，但是发现两者的数据分布几乎一致，所以排除了这种可能性。

3. 数据质量问题：

数据中可能存在部分噪声或错误，导致模型在学习和测试时受到不准确的信号引导。

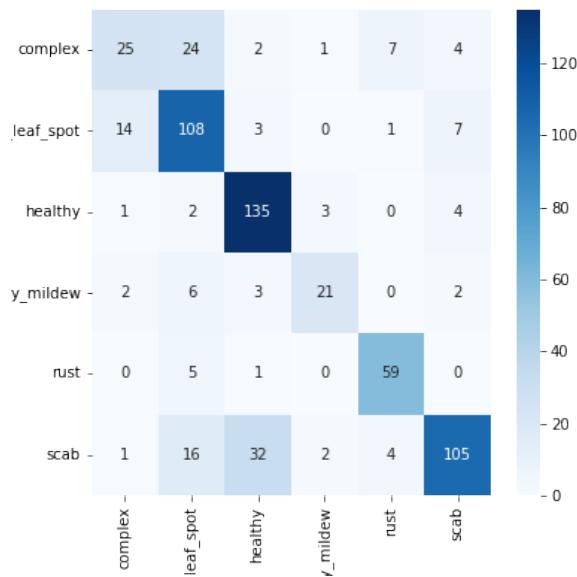


图 21 VIT 混淆矩阵

6 模型改进

在完成任务后，我与其他同学的模型结构与效果进行了比较，发现了一些有助于提高完成任务效果的方法，我也进行了相关的学习，例如：

1. 自蒸馏和软标签

自蒸馏 (Self-Knowledge Distillation) 是一种深度学习中的模型训练方法，它通过使用模型自身生成的软标签 (soft labels) 来指导训练过程，以提高模型的泛化性能、减小模型大小，并促使模型学会更加鲁棒和泛化的特征表示。其核心是：

(a) 软标签：

自蒸馏的核心是使用引导模型对输入数据进行预测，并将其输出作为软标签。这些软标签通常是类别的概率分布，而不是传统的硬标签。软标签包含了更多关于模型对输入数据的不确定性和复杂关系的信息。

(b) 迁移模型训练：

接着，一个迁移模型通过使用引导模型生成的软标签，以及原始的硬标签，进行联合训练。迁移模型的目标是使其输出尽可能地与引导模型的软标签一致，同时保持对硬标签的准确性。

自蒸馏的效果如下：

(a) 知识传递：

通过知识的传递，即引导模型对数据的理解和模型生成的软标签的信息传递给迁移模型。这有助于迁移模型学到更鲁棒、泛化的特征表示，提高模型对未见过的叶片病害样本的泛化能力。

(b) 鲁棒性提升：

自蒸馏不仅有助于提高泛化性能，还可以增强模型对输入数据的鲁棒性。叶片病害的外观可能受到光照、角度、拍摄条件等因素的影响。自蒸馏有助于使模型更具鲁棒性，更好地处理这些变化，从而提高对各种环境条件下的叶片图像的分类准确性。

总的来说，自蒸馏是一种强大的自监督学习方法，我将在未来的相关实验中尝试使用该方法来提升和改善模型。

2. TTA 策略

TTA（Test Time Augmentation）是一种在模型测试阶段采用数据增强技术的策略。与传统不同的是，TTA 将数据增强应用到模型推理的过程，模拟在不同视角、尺度或旋转条件下观察同一样本（叶片），以进一步提高模型在测试集上的性能。

其实现步骤如下：

- (a) 对测试样本进行多次增强：**对测试集中样本应用多种数据增强方式，生成多个不同样式的叶片图片。
- (b) 进行多次预测：**使用训练好的模型对每个增强后版本进行预测，得到多次预测结果。
- (c) 结果集成：**将多次预测结果进行集成，如投票机制。

总的来说，TTA 的优势在于通过模拟不同的输入条件，提高模型对于输入数据的适应性，从而改善了模型在测试集上的性能。

7 感受与体会

通过本次任务实践，我对各种分类模型都有了更为深入的学习和理解，同时对于不同的模型的结构以及其性能优势也有了大致判断。

在完成过程中，我也增强了自身的检索能力和整体完成项目能力。在服务器上部署项目，配置各项参数和熟悉使用服务器，了解各大框架下的不同，阅读大量已有论文，向作者请求源文件，在 Github 上复现项目等实践过程，对我来说，都是很好的一次锻炼。

最后，我也十分感谢在完成过程中，各位同学以及老师和助教对我的指导和帮助。通过同学之间对比不同模型的性能差异，讨论和交流已有成果和想法思路，我本次的任务收获与体验都得到了最大化。

参考文献

- [1] SZEGEDY C, VANHOUCKE V, IOFFE S, et al. Rethinking the inception architecture for computer vision[A]. 2016.
- [2] DOSOVITSKIY A, BEYER L, KOLESNIKOV A, et al. An image is worth 16x16 words: Transformers for image recognition at scale [A]. 2020.
- [3] 庄福振, 罗平, 何清, 等. 迁移学习研究进展[J]. 软件学报, 2014, 26(1): 26-39.
- [4] [李冬梅孟湘皓张小平宋潮赵玉凤]. 多标签分类综述[J]. 计算机科学与探索, 2023, 17(11): 2529-2542.
- [5] LECUN Y, BOTTOU L, BENGIO Y, et al. Gradient-based learning applied to document recognition[J/OL]. Proceedings of the IEEE, 1998, 86(11): 2278-2324. DOI: 10.1109/5.726791.
- [6] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]/Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.
- [7] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[C]/International Conference on Learning Representations. 2015.
- [8] DOSOVITSKIY A, BEYER L, KOLESNIKOV A, et al. An image is worth 16x16 words: Transformers for image recognition at scale [A]. 2020.
- [9] VASWANI A, SHAZER N, PARMAR N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.