

编号 \_\_\_\_\_



重庆理工大学

# 毕业设计（论文）

题 目 基于深度学习的农作物遥感图像语义分割

二级学院 两江国际学院

专 业 计算机科学与技术

班 级 116193701

学生姓名 胡淳钧 学 号 11619370114

指导教师 苟光磊 职 称 副教授

时 间 2020 年 6 月



# 目 录

摘要 .....	I
Abstract.....	II
1 绪论 .....	1
1.1 研究背景和意义 .....	1
1.2 国内外研究现状 .....	2
1.2.1 基于阈值的图像语义分割 .....	2
1.2.2 基于区域的图像语义分割 .....	2
1.2.3 基于边缘的图像语义分割 .....	3
1.2.4 基于卷积神经网络的图像语义分割 .....	3
1.3 研究内容及其主要工作 .....	3
2 相关理论与方法 .....	4
2.1 DeepLab-v3+网络 .....	4
2.1.1 空间金字塔模块 .....	4
2.1.2 编码-解码结构 .....	5
2.1.3 ASPP(Atrous Spatial Pyramid Pooling) .....	6
2.1.4 Xception 结构以及深度可分离卷积 .....	8
2.2 Snapshot Ensemble .....	12
2.3 标签平滑 .....	14
3 DeepLab-v3+网络模型在农作物遥感语义分割中的应用 .....	16
3.1 农作物遥感图像特点 .....	16
3.2 DeepLab-v3+网络模型基本结构 .....	16
3.3 网络模型实现 .....	17
3.4 网络训练过程 .....	19
3.4.1 设定实验必要参数与超参数 .....	19
3.4.2 网络训练流程 .....	21
3.5 评价指标 .....	26
3.6 本章小结 .....	27
4 实验过程与结果分析 .....	28
4.1 准备数据集 .....	28

4.2 实验过程 .....	29
4.2.1 增强测试 .....	30
4.2.2 膨胀预测 .....	30
4.3 实验结果 .....	32
4.4 实验结果分析 .....	35
4.4.1 MIOU 评价结果 .....	35
4.4.2 本实验方法的优点 .....	36
5 总结与展望 .....	37
5.1 结论 .....	37
5.2 农作物遥感图像语义分割存在的问题 .....	38
5.2.1 农业遥感技术仍存在不足 .....	38
5.2.2 DeepLab-v3+缺陷 .....	38
5.3 农作物遥感图像语义分割未来展望 .....	38
致 谢 .....	39
参考文献 .....	40

## 摘要

随着深度学习技术的不断发展，越来越多的人将图像语义分割运用到遥感技术上，从 20 世纪 70 年代开始，由于民用资源卫星的出现，使得农业成为了遥感技术最先投入使用和获得收益的区域。

本文主要探索了 DeepLab-v3+ 深度学习网络在农作物遥感语义分割方向上的应用。选取了贵州省兴仁市特色优势和支柱产业—薏仁米为对象，以薏仁米作物识别作为语义分割的目标。在训练模型过程中，将 Snapshot Ensemble 与 DeepLab-v3+ 网络相结合，通过余弦退火方法调整网络学习率，实验模型收敛速度，并利用实验损失函数加入 Label Smoothing 方法，对训练数据设定过滤带，将图像边缘与类间交界处数据设为硬标签，进行标签平滑操作以提高模型精度；在测试模型过程中，采用了膨胀预测方法，仅仅对实验数据中心附近位置保留预测结果，周边预测不准边缘数据舍弃，提升预测精度；并对预测图像采用水平翻转、垂直翻转、水平垂直翻转等多方位预测方式进行测试增强。

通过 DeepLab-v3+、Snapshot Ensemble 和标签平滑等多种技术相结合，对图像目标识别精度达到了较好水准，最终 MIOU 评价结果为 78.66，圆满完成项目任务，并探讨了农作物遥感语义分割现今仍存在的问题，对其未来发展进行了展望。

**关键词：** 语义分割；农作物遥感；DeepLab-v3+；Snapshot Ensemble；标签平滑

## Abstract

With the continuous development of deep learning technology, more and more people apply image semantic segmentation to remote sensing technology. Since the 1970s, due to the emergence of civil resource satellites, agriculture has become the first region where remote sensing technology is put into use and benefits are obtained.

This paper mainly explores the application of deeplab-v3 + deep learning network in the semantic segmentation of crop remote sensing. This paper takes barley rice, a characteristic which is pillar industry in Xingren city, Guizhou province, as the object, and takes job's tears barley rice identification as the target of semantic segmentation. In the process of training model, will the Snapshot Ensemble combined with DeepLab-v3+ network, adjustment by using the method of cosine tempering learning-ratio and experimental model convergence speed, and the experimental loss function to join Label Smoothing method, the training data set to filter belt, the image edge with the class at the junction between the data set to for hard tags, tag smooth operation in order to improve the precision of model; In the process of testing model, used the inflation forecast method, just keep to the position of the experimental data center near the predicted results and the surrounding forecast data are not allowed to be on the verge of renunciation, enhance forecasting precision, and the predicted images using flip horizontal, vertical, horizontal, vertical flip the way such as directional prediction test.

Through the combination of deeplab-v3+, Snapshot Ensemble, tag smoothing and other technologies, the recognition accuracy of the image target has reached a good standard. Finally, the MIOU evaluation result is 78.66, successfully completing the project task, pointing out the existing problems of remote sensing semantic segmentation of crops, and putting forward its beautiful vision.

**Keywords:** Semantic segmentation, DeepLab-v3+, Snapshot Ensemble, Label Smoothing, Remote sensing of crops

# 1 結論

## 1.1 研究背景和意义

遥感图像语义分割是指根据图像的特征如颜色、形状和纹理等将图像不同含义的区域进行分割，为其标注不同的标签，对图像中的内容进行分类。

自上世纪六、七十年代开始，关于图像语义分割的研究就没有停滞过。随着计算机技术的迅速发展，图像语义分割算法发展十分迅速，并在机器学习和物体识别等领域有长足的应用。在计算机技术发展起来之前，全球学者对语义分割也同样提出了各种经典分割方法，例如基于区域、基于边缘、基于阈值等分割方法，但是由于这些方法是根据图像像素的低阶视觉信息来进行图像分割，缺少算法训练，造成分割精度较低。但是在深度学习进入图像语义分割领域之后，通过卷积神经网络进行语义分割成为了趋势，然而传统的语义分割模型存在许多问题，例如计算需要极大的储存空间、计算效率低下、提取特征不完善等问题，使得其语义分割效率不佳。

全卷积网络的提出很大程度上解决了以上罗列的问题。2017 年，加州大学伯克利分校 Shelhamer 等提出全卷积网络（FCN），使得卷积神经网络（CNN）无需全连接层即可进行密集的像素预测，卷积网络从而得到普及<sup>[1]</sup>。任意大小的图像分割结果可以可以通过 FCN 的方式生成，而且这比图像块分类法的分割速度要快许多。语义分割领域几乎所有先进方法都采用了该模型。

全卷积网络拥有两个十分重要的优点，一是对输入数据尺寸没有限制，不再需要相同尺寸的图像，省去了一部分预处理步骤；二是计算效率更高，由于省去了大量储存空间，避免重复计算。虽然其语义分割有了极大的改进，但是仍然有着一些不足，主要表现在一是分割结果较粗糙。FCN 在恢复特征图尺寸时基本使用的是简单的上采样，容易造成细节丢失；二是对大尺寸物体，容易分配错误标签，而对小尺寸物体容易忽略其存在；三是 FCN 虽然实现了逐像素分类，但是其忽略了相邻像素间的类别相关性，造成实验结果缺少空间一致性。

为此，2015 年，谷歌公司发布了 DeepLab 模型，经过卷积神经网络特征提取器、目标维度建设模型的技术、语言环境信息处理技术、模型训练框架、机器学习硬件和软件的不断提升与更新，最终在 DeepLab-v3 模型的基础上，添加了一个解码器模块以对目标边界处的结果进行精炼分割，并且该模型探索了将深度可分卷积与空间多孔金字塔池化（ASPP）和解码器模块相结合的方法，得到了语义分割速度更快速、更精确的语义分割编码器-解码器网络<sup>[1]</sup>。

目前，图像语义分割算法被广泛应用于人脸识别、指纹识别、脑肿瘤分割、医学仪器跟踪和自动驾驶等具体领域。例如，语义分割可以应用于面部分割，通常情况下，肤色、发状、以及面部器官信息和背景等特征信息是对于面部的语义分割的关键部分。对人脸的语义分割往往能获取到很多特征，比如说岁数、穿戴、表情、容貌等等；语义分割可以应用于自动驾驶，智能计算机在自动驾驶中十分关键，因为自动驾驶技术不得不在动态环境中进行识别、计算和执行。而对于驾驶员的人身安全是在该技术中最需要考虑的部分，这也就需要对该事件进行最高精度的处理。语义分割提供道路空间信息、车道标记信息和交通标志等信息。

## 1.2 国内外研究现状

当今社会下，经典的语义分割算法主要是有以下三种：基于阈值的语义分割、基于边缘的语义分割以及基于区域的语义分割。但是这些方法均基于图像的像素信息进行判断，并没有引入算法训练阶段，故语义分割的精度不够高。但随着近些年来深度学习技术的发展，将其应用到图像语义分割技术上成为了趋势。

### 1.2.1 基于阈值的图像语义分割

基于阈值的图像语义分割算法的基本思想是利用图像的灰度特征，进而得到一个或者多个灰度阈值，之后将其与每个像素的灰度阈值进行比较，最终得出分类结果，因而选取恰当的准则函数变得非常重要。阈值分类算法有多种，其中 OTSU 最为知名。

在上世纪七十年代，日本学者 OTSU 提出了一种可以对图像进行二值化操作的算法，后来命名为 OTSU 算法。后经过数十年的优化，2018 年陈宏伟等人通过烟花算法<sup>[2]</sup>，高效而准确地获得 OTSU 多阈值图像语义分割的最佳组合，该方法将多阈值的求解转变为多维度变量的求解问题，利用烟花算法进行迭代运算计算结果。OTSU 算法计算简单快速，不受图像亮度和对比度的影响，但是其对图像噪声敏感，只能针对单一目标进行分割，但是当目标的类间方差函数出现双峰或多峰的情况，即大小比例相差很大，在这种情况下，该方法的识别效果比较低下。

### 1.2.2 基于区域的图像语义分割

基于区域的语义分割是将一幅图像以像素为单位进行分割，再利用聚类算法聚合相似像素。其中分水岭算法在它们中间最具有代表性，该方法来源于拓扑算法的数学形态的语义分割方法，其中心理论是将语义分割的图像看做地图上高矮不一的地貌，而图像上不同像素点间不同的灰度值则代表不同的海拔，局部极小被形象的称为集水盆，其边界称为分水岭，这就是算法名称的由来。为了说明分水岭的概念，可以通过计算机模拟将模型浸入水中来实现。首先，对局部极小做刺穿处理，让水流入模型，随着流入水量的增加，每个

局部极小的影像也随着水的增加而加大，最后会在两个不同的集水盆的交界处形成分水岭。

### 1.2.3 基于边缘的图像语义分割

基于边缘的语义分割算法可以说就是基于灰度值的边缘检测。边缘检测需要用到边缘检测算子，例如：Sobel 算子、Canny 算子、Prewitt 算子等，但是边缘检测后得到的图像并不能作为最终结果，仍需通过边缘松弛、边缘跟踪<sup>[3]</sup>等进行图像的后期处理

### 1.2.4 基于卷积神经网络的图像语义分割

1998 年，LeNet-5 首次由 Lecun 等人提出，这是一个经典的卷积神经网络，该网络对手写数字的识别精度较为准确，后来随着深度学习的发展，卷积神经网络开始运用到图像语义分割方面。

本文主要准备采用 Deeplab-v3+网络进行研究，空间金字塔池模块或编码-解码结构用于深度神经网络的语义分割任务。前者能够通过用过滤器探测传入的特征或者以多个速率和多个感受野的池化操作来编码多尺度上下文信息，而后者可以通过逐步恢复空间信息来捕捉更清晰的物体边界<sup>[7]</sup>。而 DeepLab-v3+网络就是结合这两者的优点，并将深度可分离卷积应用于空间金字塔池和解码器模块，从而形成了一个更快、更强的编码-解码网络。

## 1.3 研究内容及其主要工作

根据已掌握的深度学习相关知识，通过无人机航拍的高空间分辨率遥感数据，开发算法模型，利用 Deeplabv3+语义分割模型为基础进行深度学习，形成模型并进行训练、自动调参辅以定时人工检测正确率辅助调参，以此来探索作物分类的精准算法，最终要求识别薏仁米、玉米、烤烟、人造建筑四大类型，提升作物识别的准确度，降低对人工实地勘察的依赖，提升农业资产盘点效率。

本文选用具有独特的地理环境、气候条件以及人文特色的贵州省兴仁市作为研究区域，聚焦当地的特色优势产业和支柱产业——薏仁米产业，以薏仁米作物识别作为语义分割的目标。

## 2 相关理论与方法

### 2.1 DeepLab-v3+网络

在 DeepLab-v3+ 中，使用了两种类型的神经网络结构，分别是空间金字塔模块和编码-解码结构做语义分割。

#### 2.1.1 空间金字塔模块

2015 年，一篇发表在 IEEE 上的由何凯明等人撰写的论文<sup>[8]</sup>提出了 SPP-Net 网络，如图 2-1、2-2 所示，该网络的出现改变了之前的神经网络输入的数据维度是固定的状态。

在空间金字塔池化模块出现之前，传统的深度卷积神经网络（CNNs）都需要一个固定的输入数据尺寸。这种人为的需要输入固定尺寸的需求导致识别不同尺寸和比例的数据时，识别精度降低。因为为了达到固定尺寸的要求，我们需要对图像进行预处理，传统的处理方法有裁剪、缩放两种。对于裁剪，该方法易造成原始图像信息的丢失，而对于缩放操作可能会使图像畸形失真。

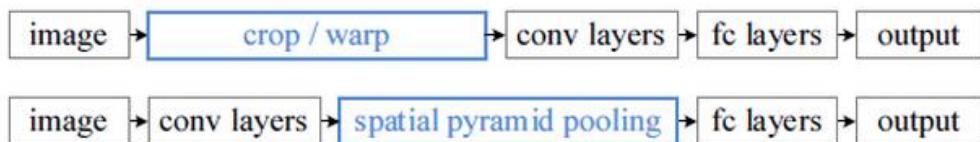


图 2-1 空间金字塔模型

（该图上面的流程为传统卷积神经网络的计算过程，下面的计算流程为空间金字塔池化的计算流程（SPP），可以看出空间金字塔模块省去了裁剪和缩放操作，也因此不需要对图像进行预处理，空间金字塔池化层被添加在卷积层和全连接层直接，保证了不同大小比例的图像语义分割的精确性）

对于多尺度训练，例如 180\*180, 224\*224 这样的输入，我们通过使用共享参数来使两个固定尺寸的网络实现不同输入尺寸的 SPP-Net，但为了降低从一个网络到另一个网络的切换的开销，在保留权重的前提下，我们不得不在每一个网络上训练一个完整的 Epoch 之后再切换到另一个网络。而且也多尺度训练的计算速度也得到了保证，多次测试表明多尺度数据集训练的模型收敛速度跟单尺度数据的收敛速度大体一致。

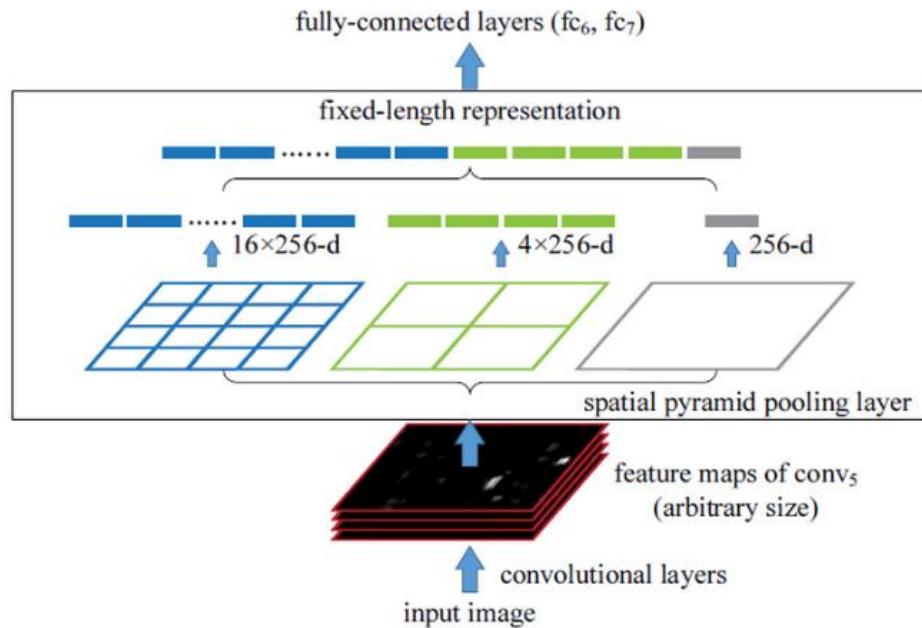


图 2-2 SPP-Net 网络中空间金字塔池化层内部构造

(该图为 SPP-Net 网络中，空间金字塔池化层的内部构造，这里我们举一个最简单的例子，假设我们的输入为 13\*13 的图像，我们需要得到 4\*4、2\*2、1\*1 的网络，我们的池化层需要分别设置大小为 4，步幅为 3；大小为 7，步幅为 6；大小为 13，步幅为 13 的三个不同大小的格子进行计算，以保证输出的一致性。Windows\_size=[a/n] 向上取整，Stride\_size=[a/n] 向下取整。)

### 2.1.2 编码-解码结构

分割任务中的编码器（Encode）和解码器（Decode）各司其职，这两者就像来玩你画我猜的游戏，比划的人把看到的东西用一种方式描述出来，而猜的人则是根据比划的信息猜出答案。具体来说，编码器的任务是在给定图像数据后，通过神经网络的迭代学习获取图像的特征；解码器则是在获得图像特征后，实现每个像素的类别标注，即分割。

最经典的编码-解码结构是 SegNet，由 Vijay Badrinarayanan 等人<sup>[9]</sup>于 2015 年提出，其是在 FCN 的语义分割任务基础上，搭建了对称的编码器-解码器结构，如图 2-3 所示，实现了端到端的像素级别图像分割。

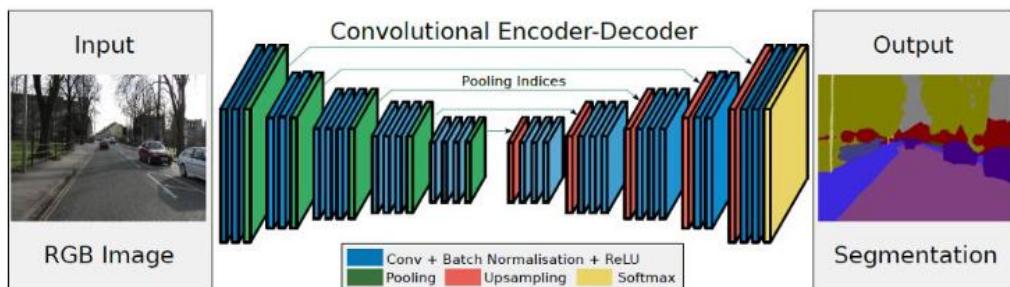


图 2-3 SegNet 编码-解码结构

(其具有编码器网络和相应的解码器网络，最后是按照最终像素进行分类的分类层)

在解码器方面，执行最大池化的索引上采样和卷积操作后，softmax 分类器会来处理每个被送来的像素，对它们进行分类。如图 2-4 所示，计算机使用的最大池化索引进行上采样后，每个像素的种类被 K 类 softmax 分类器进行分类。

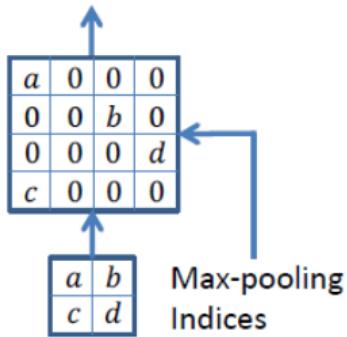


图 2-4 Decoder 解码器

### 2.1.3 ASPP(Atrous Spatial Pyramid Pooling)

ASPP 是 DeepLab 中用于语义分割的一个模块，它是出自 2016 年发表在 IEEE 上的论文<sup>[10]</sup>。通常情况下，语义分割的目标具有各不相同的尺度，这种不确定性给语义分割增加了难度。其中一种处理方法是通过 rescale 图片，然后将处理过的图片经过 DCNN (Deep Convolutional Neural Network) 处理融合，但是使用该方式会大大降低计算速度，为了解决这个问题，DeepLab 的研究团队创造了一个 ASPP 模块，这个模块不仅可以对多尺度目标进行特征提取，而且需要的计算机工作量又没有太大。

在介绍 ASPP 之前，我们不得不先对空洞卷积 (Atrous Convolution) 进行讲解，空洞卷积是为了解决基于 FCN 思想的语义分割中，在 FCN 中输出图像的大小必须和输入图像的大小一致，但是由于 FCN 中使用池化操作来增大感受野，图像的分辨率同样会降低，这导致了数据无法还原由于池化操作导致的一些细节信息的损失。为了减少损失，我们必须移除池化层，因此提出了空洞卷积的概念，如图 2-5 所示。

但是，空洞卷积存在一定的弊端，在空洞卷积的过程中并不是所有的像素点都用来计算，这对于像素级别的预测来说是致命的，如果仅仅采用大的扩张速率或许对一些大物体的分割有效果，但是对小物体就是有弊无利了。

R-CNN 空间金字塔池化对 ASPP 方法的创造有至关重要的影像，ASPP 方法可以对任意大小尺度的特征进行分类，这是通过对单一尺度的特征进行卷积操作重采样来实现的。而通过使用不同采样率的多并行 Atrous Convolutional Layers 就是实现这个功能的另外一条途径。该途径内为每一个不同的采样率的特征在其各自的分支中进行处理，最后每个不同的特征进行融合生成最后结果。提出的“*Atrous Spatial Pyramid Pooling* 池” (DeepLab-ASPP) 方法概括了我们的 DeepLab-LargeFov 变量<sup>[10]</sup>，如下图 2-6 示。

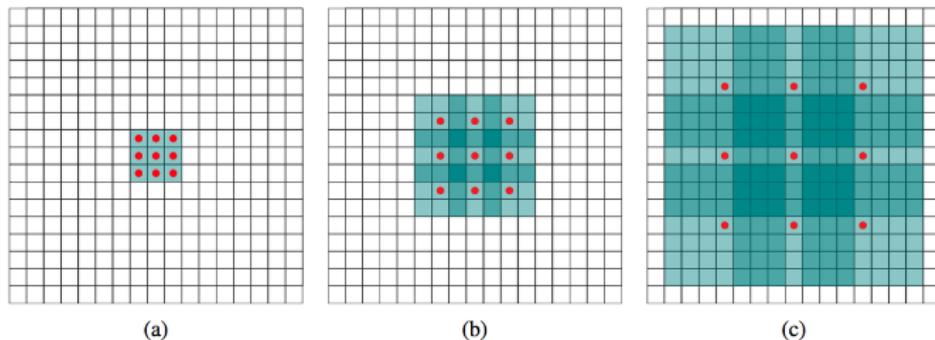


图 2-5 空洞卷积与普通卷积的区别直观图

- (a) 普通卷积, 1-dilated convolution, 卷积核的感受野为  $3 \times 3$   
 (b) 扩张卷积, 2-dilated convolution, 卷积核的感受野为  $7 \times 7$   
 (c) 扩张卷积, 4-dilated convolution, 卷积核的感受野为  $15 \times 15$

空洞卷积的计算公式为:  $[(n - 1) * (\text{Ksize} + 1) + \text{Ksize}]$  (n: 扩张速率; Ksize: 卷积核的大小)

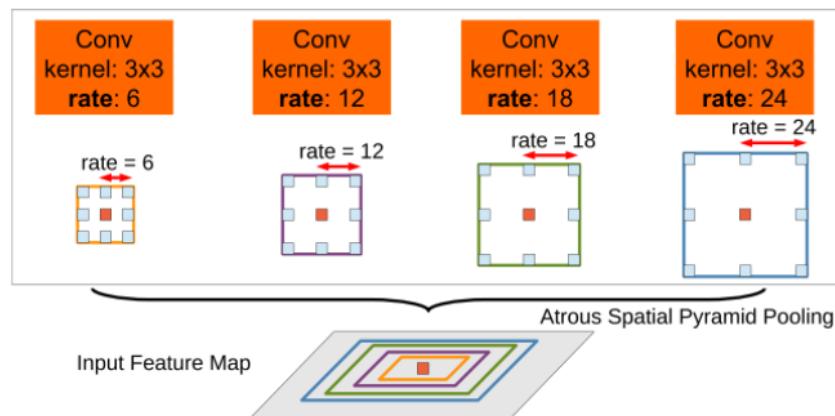


图 2-6 DeepLab 的 ASPP 结构

我们可以看出, ASPP 在顶部映射图中使用了四种不同采样率的空洞卷积, 这表明其可以以不同尺寸来进行采样, 在 DeepLab-v3 中, 在 ASPP 中添加了 BN 层, 这使得不同采样率的空洞卷积可以捕获多尺度信息, 但随着采样率的增加, 滤波器有效权重逐渐减小, 如图 2-7 所示。

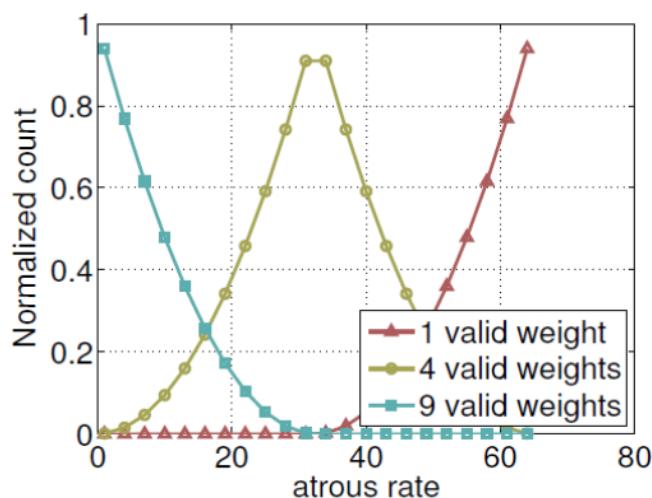


图 2-7 空洞卷积与采样率和权重的关系

当以不同的采样率让  $3 \times 3$  的卷积核应用到  $65 \times 65$  的特征映射上，当采样率接近特征映射大小时，滤波器通过大小为  $1 \times 1$  的滤波器替代捕捉整体图像的上下文信息，由空洞卷积理论可知，图像仅仅只有最中心的位置才拥有有效权重，为了能克服这个难题，在 DeepLab-v3 中使用了图片级特征，具体来说，就是在模型最后的特征映射上应用全局平均，将结果经过  $1 \times 1$  的卷积，再双线性上采样得到所需的空间维度，所有的特征通过  $1 \times 1$  卷积，如图 2-8 所示，生成最终分数<sup>[11]</sup>。

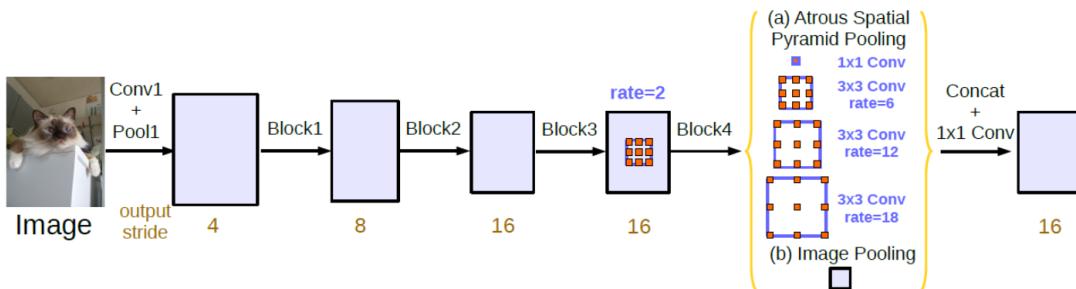


图 2-8 改进后的 ASPP 模块

(改进后 ASPP 模块包括一个  $1 \times 1$  的卷积层以及采样率为 {6, 12, 18} 的三个  $3 \times 3$  的空洞卷积，其中模块的滤波器的数量为 256，而且 ASPP 模块中包含有 BN 层；图像级特征，将特征做全局平均池化，经过卷积，再融合，如上图 (b) 部分所示 (注：input stride 为我们正常进行卷积时候设置的 stride 值，output stride 为该矩阵经过多次卷积 pooling 操作后，尺寸缩小的值，例如：input image 为  $224 \times 224$ ，经过多次卷积 pooling 操作后，feature map 为  $7 \times 7$ ，那么 output stride 为  $224/7 = 32$ )

## 2.1.4 Xception 结构以及深度可分离卷积

深度可分离卷积 (Depthwise Separable Convolutions) 最早是由 Google Brain 的一名实习生 Laurent Sifre 于 2013 年提出的，后来在其博士论文中对深度可分离卷积进行了详细的分析和实验。就像论文中提到的，卷积操作的卷积核实际上是由三维滤波器组成的，其包含通道维和空间维，空间维即特征图的宽和高，而通道相关和空间相关的联合映射就是常说的卷积操作。Inception 模块的背后存在这样的一种假设：而这两者是可以退耦合的，如果我们将其分开映射就可以达到更加精确的分类<sup>[12]</sup>。Inception 模块结构示意图如图 2-9、2-10、2-11、2-12 所示。

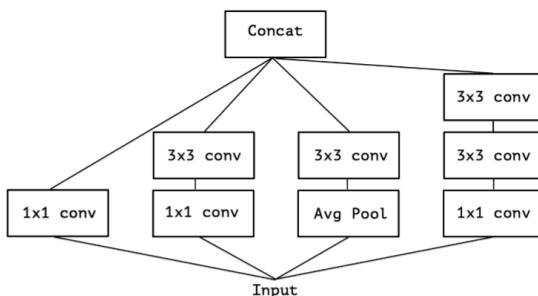


图 2-9 标准的 Inception 模块 (Inception V3)

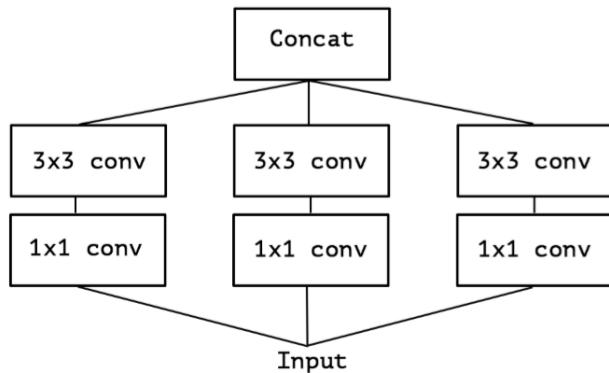


图 2-10 简化的 Inception 模块

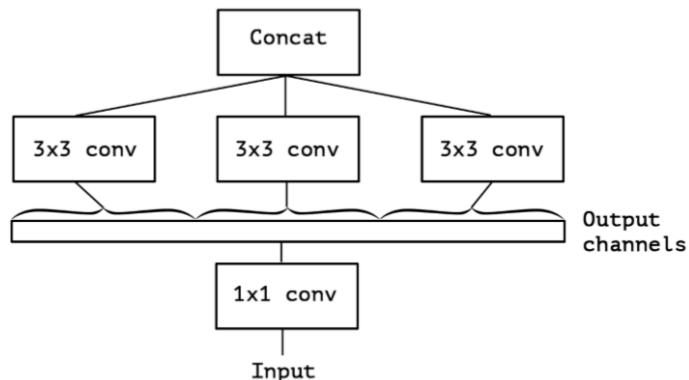


图 2-11 属于简化 Inception 模块的严格等价变形

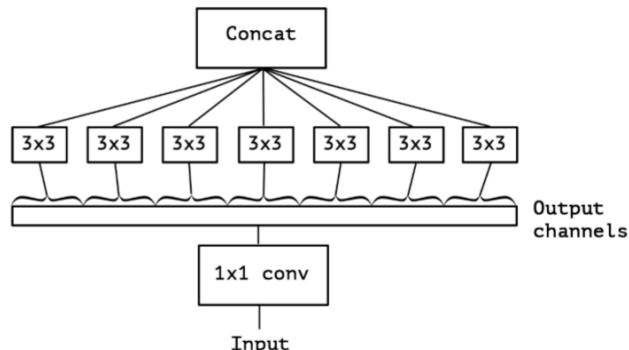


图 2-12 Inception 模块的“极端”版本，每个输出通道都伴有一个 1\*1 的空间卷积

以上的图中，图 2-9 是一个典型的 Inception 模块，首先利用通道相关性，将输入的数据通过一个 1\*1 的卷积使其映射到比原来维度更小的几个空间上，这也可以说是将时输入图片的每个通道分别计算，乘上各异的因子进行线性组合，再将这些处理过的空间通过 3\*3 的卷积，这是对输入数据的空间相关性进行映射。现在以第二个分支作为标准，首先假设输入的数据为 28\*28\*192 的图片代销，首先将该图片通过 32 个 1\*1\*192 的卷积，这步是对图片的通道相关性做线性组合，处理结束后会反馈一张 28\*28\*32 的特征图，之后将其通过 256 个 3\*3\*32 大小的卷积核，这步就完成了通道相关性与空间相关性的联合映射，最终会获取到 28\*28\*256 的特征图。而这个结果可以很容易得出，其跟对输入的数据

进行直接通过 256 个  $3 \times 3 \times 192$  的卷积核是一致的。也因此，Inception 模块做出假设 32 个  $1 \times 1 \times 192$  和 256 个  $3 \times 3 \times 32$  的卷积核对输入的数据进行退耦级联，与直接用 256 个  $3 \times 3 \times 192$  卷积核等效。而两种方式的参数量则分别为  $32 \times 1 \times 1 \times 192 + 256 \times 3 \times 3 \times 32 = 79872$  和  $256 \times 3 \times 3 \times 192 = 442368^{[12]}$ 。

简化后的 Inception 模块如图 2-10 所示，在图 2-10 的基础上，可以略做调整，首先将  $1 \times 1$  的卷积核进行合并，例如，现在有三组 32 个  $1 \times 1 \times 192$  的卷积核，可以将其进行合并成 96 个  $1 \times 1 \times 192$  的卷积，在合并后的卷积核后连接三组  $3 \times 3$  的卷积，其输入为之前输出的三分之一，如图 2-11 所示。而 Xception 理论对该情况作出了新的假设，它假设通道相关性和空间相关性是可以全部分开的，如图 2-12 所示，该图解释了 Inception 的一种极端状况，首先对输入数据通过  $1 \times 1$  的卷积，即对图片的通道相关性进行处理，在这之后，将输出通过  $3 \times 3$  的卷积核，但是这里  $3 \times 3$  卷积的个数与输出的通道数量必须一致。

图 2-12 中的 Inception 模块已经跟深度可分离卷积非常相似了，但仍然有两点区别：

- (1) 相比于 Inception，名为 Channel-wise 的空间卷积会优先被深度可分离卷积进行，之后再将结果通过  $1 \times 1$  的通道卷积；(2) 与深度可分离卷积相反，在 Inception 模块中，每一个操作的结果都会通过一个名为 ReLU 的非线性激活函数。

因此，Xception 即为深度可分离卷积与 Inception 的结合，如图 2-13 所示，其中 SeparableConv 就是深度可分离卷积，并且可以分析出除了模块的开头和结尾外，残差连接着每一个模块：

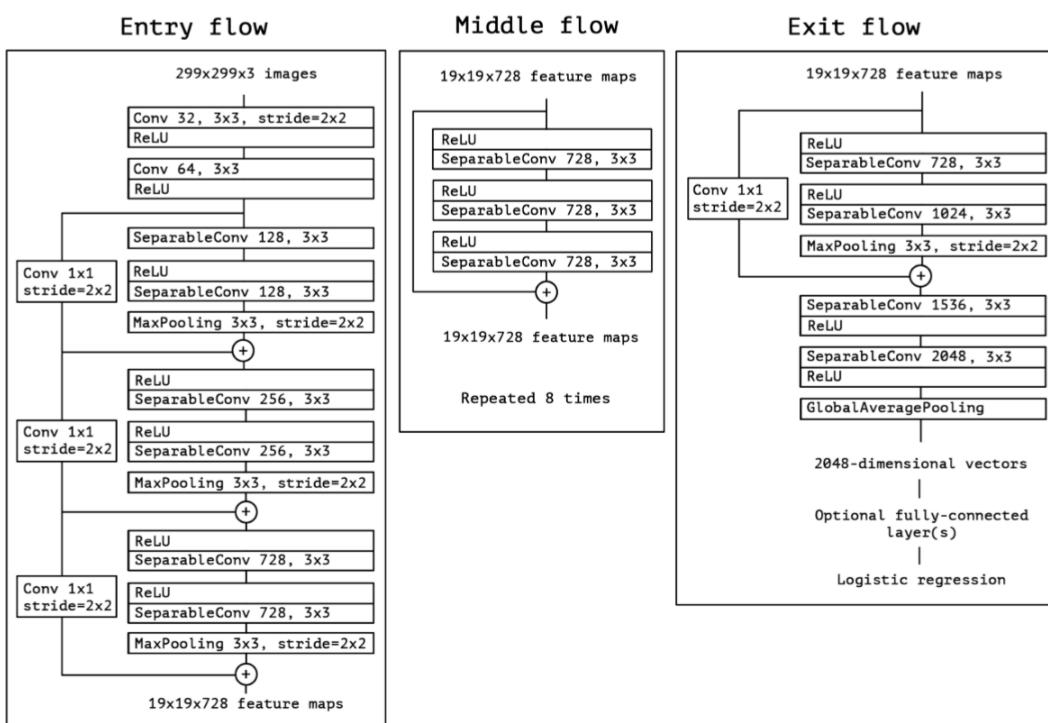


图 2-13 Xception 结构图

对于 DeepLab-v3+网络，则是在 DeepLab-v3 的基础上将多个技术结合到一起，并成功达到了新的高度。

空间金字塔模块在输入特征上应用多采样率扩张卷积、多接收野卷积或池化，探索多尺度上下文信息。编码-解码结构通过逐渐恢复空间信息来捕捉清晰的目标边界。

DeepLab-v3+结合了这两者的优点，具体来说，以 DeepLab-v3 为编码器架构，在此基础上添加了简单却有效的解码器模块用于细化分割结果。可通过扩张卷积直接控制提取编码器特征的分辨率，用于平衡精度和运行时间。此外其进一步探究了以 Xception 结构为模型主干，并探讨了深度可分离网络在 ASPP 和解码器模块上的应用，最终得到了更快更强大的编码-解码网络<sup>[7]</sup>。具体如图 2-14 所示。

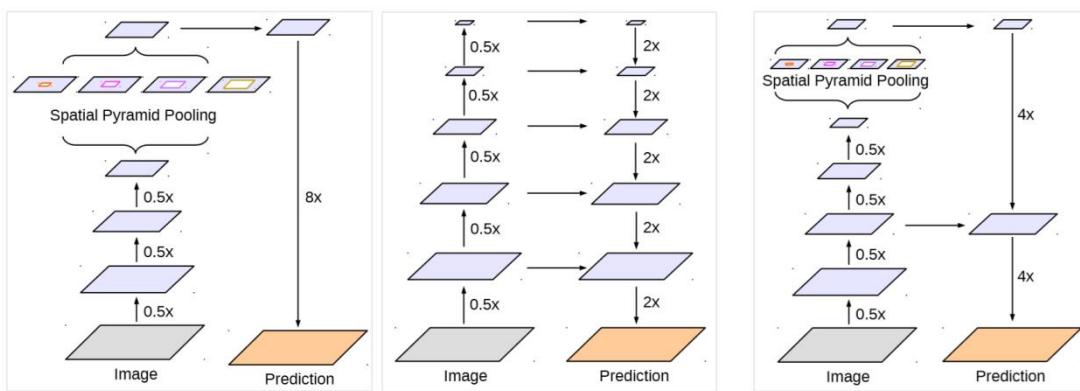


图 2-14 (a) 空间金字塔池

(b) 编码解码器

(c) 带有 Atrous 卷积的编码解码器

图 2-14 (a) DeepLab-v3 的结构，使用 ASPP 模块获取多尺度上下文信息，直接上采样得到预测结果

(b) 编码-解码结构，高层特征提供语义，解码器逐步恢复边界信息

(c) DeepLab-v3+结构，以 DeepLab-v3 作为编码器，结合一个简单的解码器

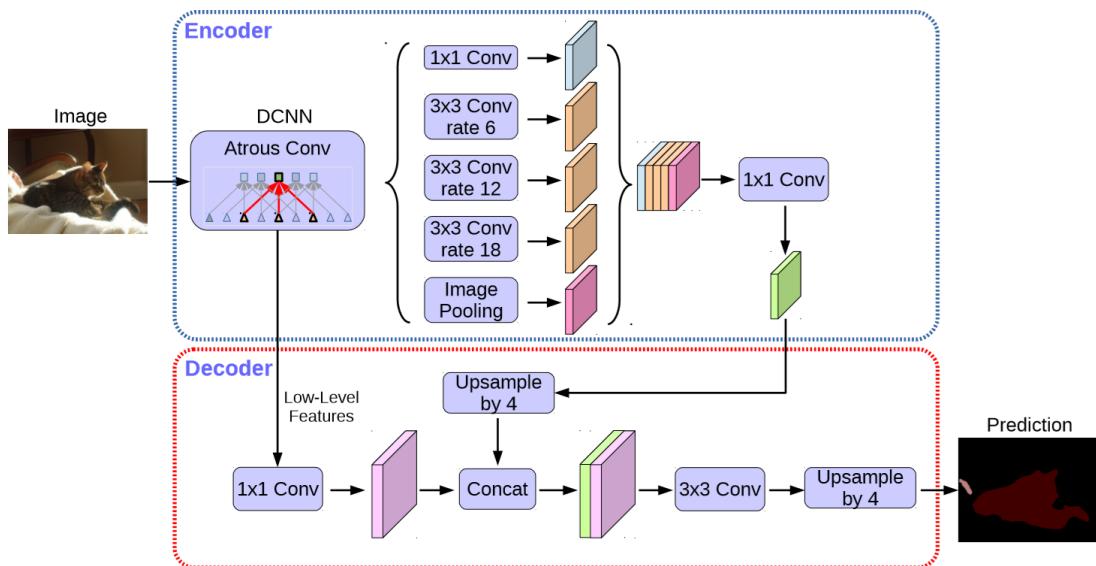


图 2-15 DeepLab-v3+网络

DeepLab-v3+网络如图 2-15 所示，首先使用 DeepLab-v3 的编码-解码结构作为其编码

器的上采样，输出的特征通道为 256，对于下采样，则使用扩张卷积，输出特征的 output\_stride = 16。

对于其解码器部分，由于网络编码器部分的上采样需经过双线性上采样 4 倍得到，其 output\_stride = 4。因此对于编码器的下采样需取得与上采样相同的分辨率并对其进行降通道，即通过 1\*1 的卷积网络，降上下采样获得的特征进行 Concat，经过 3\*3 的卷积细化特征，最终由双线性上采样 4 倍得到与原始图像相同的分辨率，取得预测结果。

深度可分离卷积可以大幅降低参数量和计算量，而扩张卷积可以在不增加计算量和参数量的基础上，有效扩展感受野。DeepLab-v3+将两者结合到一起，称为扩张分离卷积，扩张分离卷积能够显著的减少模型的计算复杂度并维持相似的表现，用其对 Xception 进行改进，应用到编码器网络替换 DeepLab-v3 的 ResNet101，如图 2-16 所示。

到此为止，对 DeepLab-v3+ 网络理论的阐述已经完成，但由于农作物遥感的数据较为复杂，存在大量随机性，造成实验预测模型的训练速度下降、精度不高，为了缓解这些情况，引入了两个新的概念，Snapshot Ensemble 和 Label Smoothing。

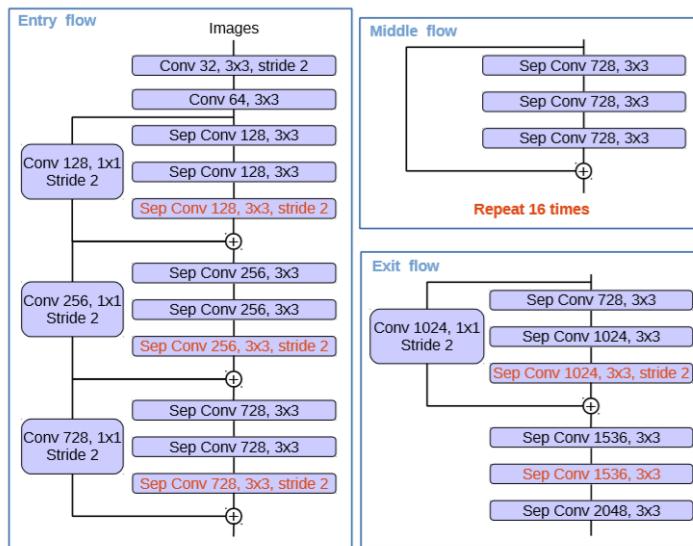


图 2-16 改进的 Xception 结构

（采用更深的 Xception 结构，不同的地方在于为了快速计算和有效使用内存而不修改 Entry flow 网络的结构；所有的最大池化操作被带下采样的深度可分离卷积替代，以便保持与上采样相同的分辨率；在每个 3\*3 的深度卷积后增加 BN 层和 ReLU 层）

## 2.2 Snapshot Ensemble

在神经网络的训练过程中，模型往往不会收敛到全局最小上，通常情况下，模型会收敛到局部极小，通常认为局部极小拥有较为平滑的地区，这些区域对模型的泛化能力都比较不错，这些位置是更加优秀的局部极小。所以模型会以大的学习率使其以较快的速度移动到局部极小的平坦区域，当到达没有更加有效的提升模型的阶段后，降低学习率，使得

模型收敛到局部极小内，这就是 SGD 方法的核心思想。

但是在实际的训练过程中，不同局部极小会拥有极其相似的误差，但是不同局部极小训练出来的不同模型在进行推理预测时产生的错误都不尽相同，这些特异点在进行 Ensemble 时会用到，通过对多模型进行投票会对预测的最终结果有一定提升，因此多模型 Ensemble 被许多科学的研究者使用。

但是在神经网络训练时，每个 Ensemble 的基模型都需要单独计算，需要耗费大量时间，不需要增加额外的训练消耗，2017 年黄高等人<sup>[13]</sup>提出了一种方法，在同一次训练的过程中，保存多个不同的模型，最终对模型进行投票，即 Ensemble，确定最终模型。

在使用 SGD 方法进行模型训练时，由于 SGD 方法可以使模型收敛和脱离局部极小，如图 2-17 所示，该函数公式见式 1。在一次训练过程中，使模型多次收敛于不同的局部极小，每次模型收敛完成，这个模型就可以作为最终模型，所以就需要将模型保存。然后使用一个较大的学习率逃离此时的局部极小，在最终收敛之前，访问多个局部极小，在每个局部极小保存 Snapshot 作为一个模型（要求：有尽量小的误差；每个模型误分类的样本尽量不要重复，保证模型的差异性），在预测推理时使用所有保存的模型进行预测，最后取平均值最为最终结果，如图 2-18 所示。

对 Snapshot Ensemble 方法的阐述已经完成，该方法主要应对模型的训练速度缓慢的问题，下一节将对 Label Smoothing 进行阐述。

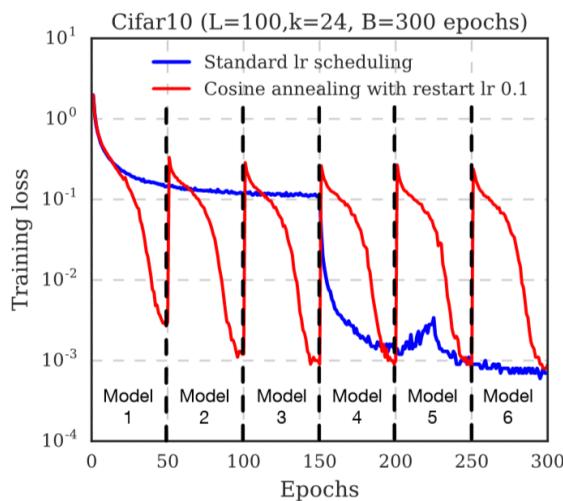


图 2-17 神经网络 SGD 方法

（通过一种余弦函数实现对学习率的控制，上图为该函数的学习过程的表现，要求：急剧提升学习率；在训练的过程中学习率快速下降。）

$$\alpha(t) = \frac{\alpha_0}{2} \left( \cos \left( \frac{\pi \text{mod}(t-1, \lceil T/M \rceil)}{\lceil T/M \rceil} \right) + 1 \right) \quad (1)$$

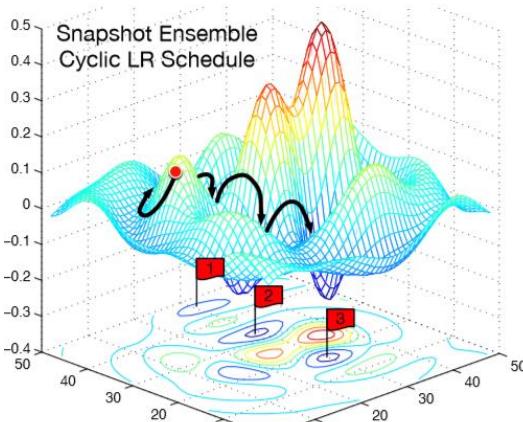


图 2-18 Snapshot Ensemble 的图像表现

### 2.3 标签平滑

对于标签平滑，Hinton 等人于 2015 年发表的论文<sup>[14]</sup>和《When does label smoothing help?》<sup>[15]</sup>的工作，其中 Hinton 提出了知识蒸馏的概念，从大模型所学习到的知识中学习有用信息来训练小模型，在保证性能差不多的情况下进行模型压缩，具体的来说，论文中指出 Teacher 模型输出的 Soft Target 训练 Student 模型，而 Student 模型的目标函数由以下两项的加权平均组成：Soft Target 和小模型的输出数据的交叉熵；Hard Target 和小模型的输出数据的交叉熵。结果显示，这样比直接用 Hard Target 训练的模型具有更强的泛化能力。

如图 2-19 所示，该图的纵坐标代表比较流行的语义分割数据集，分别为 CIFAR10、CIFAR100、ImageNet(Course)、ImageNet(fine)，在网络上进行训练的倒数第二层输出数据的可视化图，横坐标代表对数据进行不同的标签平滑处理，第一例为硬标签训练集结果，第二列为硬标签测试集结果，第三列为软标签训练集结果，第四列为软标签测试集结果，从图上可以得出，通过软标签训练的数据集类间分类更加明显，类内更加聚集。

对于标签平滑来说，在多分类任务中往往倾向于将类别用一种名为 One-hot Vector 对应的向量来表示，例如，[0, 1, 0]，即对于长度为 n 的数组，只有一个元素是 1，其余都为 0。由此可以得出，将同一类的值设定为 1，其他类的值设定为 0，通过这种方法可以对已知样本进行分类。但是这种方法会在计算损失函数时带来两个问题，一是该方法比较易造成模型的过拟合现象，无法确保模型的泛化能力；二是 One-hot Vector 方法使得不同类之间的差距加大，但是梯度是有界的，网络很难适应这种不一致的情况，使模型对预测的类别的信任程度过于巨大。

以一个最简单的标签平滑实例进行验证：

```
def label_smoothing(inputs, epsilon=0.1):
    K = inputs.get_shape().as_list()[-1]      # number of channels
    return ((1-epsilon) * inputs) + (epsilon / K)
```

**解释：**代码的第一行是取输入数据的通道数，即类别数；代码的第二行是计算部分，可以看出在计算的过程中，为了使用标签平滑，计算机将同类内的概率均匀分配了一部分给其他两类，使得不同类之间的界限不会那么明显，留给学习一点泛化的空间，从而使训练的模型更加稳定和泛化能力更强。

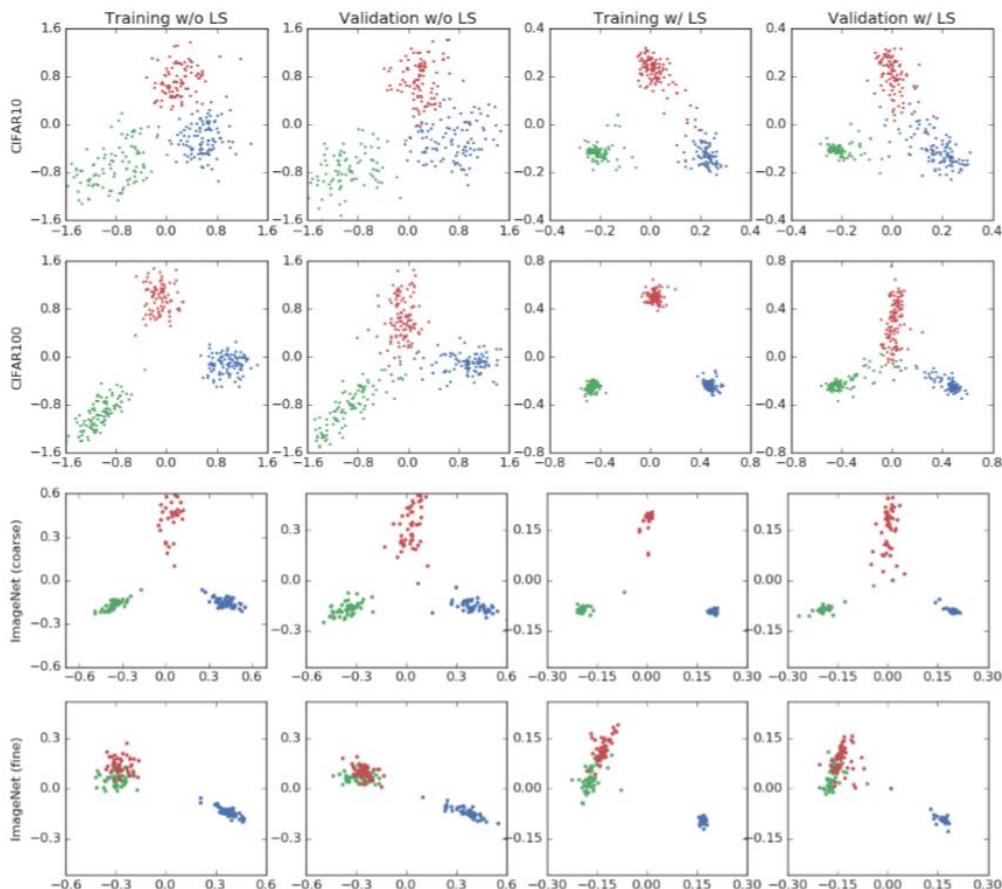


图 2-19 《When does label smoothing help?》论文标签平滑示意图

该方法主要为应对由于农作物遥感数据的随机性造成预测模型的精度不高的情况，通过设置硬标签进行标签平滑操作，以此来提高模型的预测精度，并通过 Snapshot Ensemble 方法提高模型收敛速度。最终将这两种方法与 DeepLab-v3+网络相结合，来提升语义分割的效率。

神经网络训练的过程中通过引入伪标签也可以对实验预测结果有一定的提升，伪标签技术即为对输入的数据不需要人工标记标签，而是根据首次网络训练模型给出的结果对输入数据标注近似的标签。下面给出伪标签的执行步骤：第一步：使用标签数据训练模型；第二步：将没有标签的数据通过已经训练好的模型进行训练，添加标签；第三步：将预测出的伪标签和原始标签数据重新输入网络，重新训练模型，该步中最终得到的模型即为进行对测试集进行推理预测的最终模型。

这种技术的优点是模型在更精准的决策边界下得到了改进。

### 3 DeepLab-v3+网络模型在农作物遥感语义分割中的应用

#### 3.1 农作物遥感图像特点

以农作物遥感图像作为语义分割的目标,农作物遥感图像相比如其他经典的语义分割图像,例如, PASCAL Visual Object Classes (VOC)、Imagenet、PASCAL-Person-Part 等,有类间分布不均和类内时间不统一的特点。所谓类间分布不均即为在不同类之间,由于农作物种植不像普通建筑,人类无法对农作物的种植范围有一个精确的测量,这会造成不同类之间以锯齿状相互交错,但往往类与类之间仅仅相差几个像素,对计算机来说这几个像素之间是高度相似的,加大了语义分割的难度。而类内时间不统一即为由于农作物存在着随着天气、湿度、自身生长情况等外部环境的变化而变化的情况,往往获取到的图像是不同时间点拍摄后进行拼接的高分辨率图像,这会造成类内由于拍摄时间不同而影响在相同标签下的图像特征不一致,增加了错误数据,从而加大了语义分割难度,如图 3-1 所示。

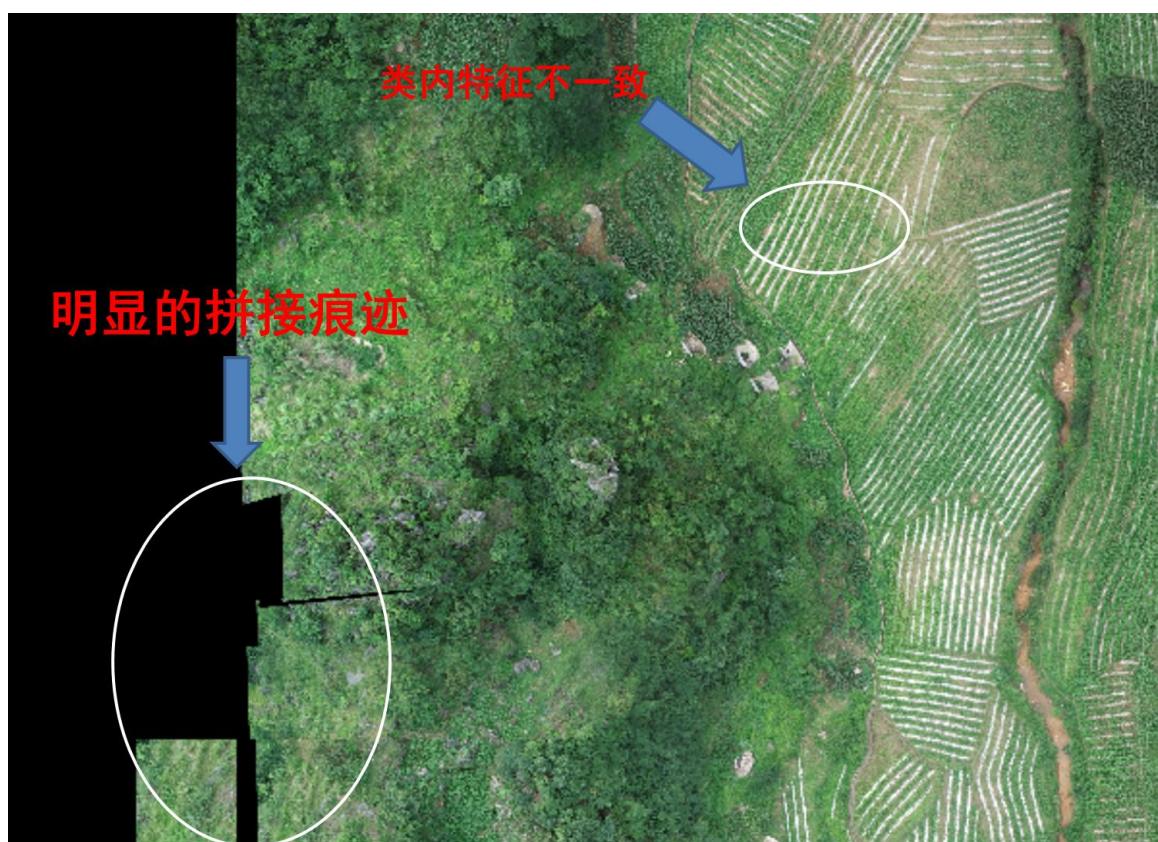


图 3-1 农作物遥感图像特点

#### 3.2 DeepLab-v3+网络模型基本结构

本实验采用 DeepLab-v3+网络, 输入部分最后决定图片大小为 1024\*1024, 采用 RandomCrop 的方式随机选取图片内的像素点, 保证实验数据获取的随机性, 提高模型泛

化能力。Backbone 最后使用的是 ResNet-101，在低级特征获取结束后使用 ASPP（空间金字塔池化模型）获取高维信息，高维信息经过全局平均和双线性 4 倍上采样后与低级信息合并，最终合并的特征再经过一次双线性 4 倍上采样恢复原图片维度，如图 3-2 所示。

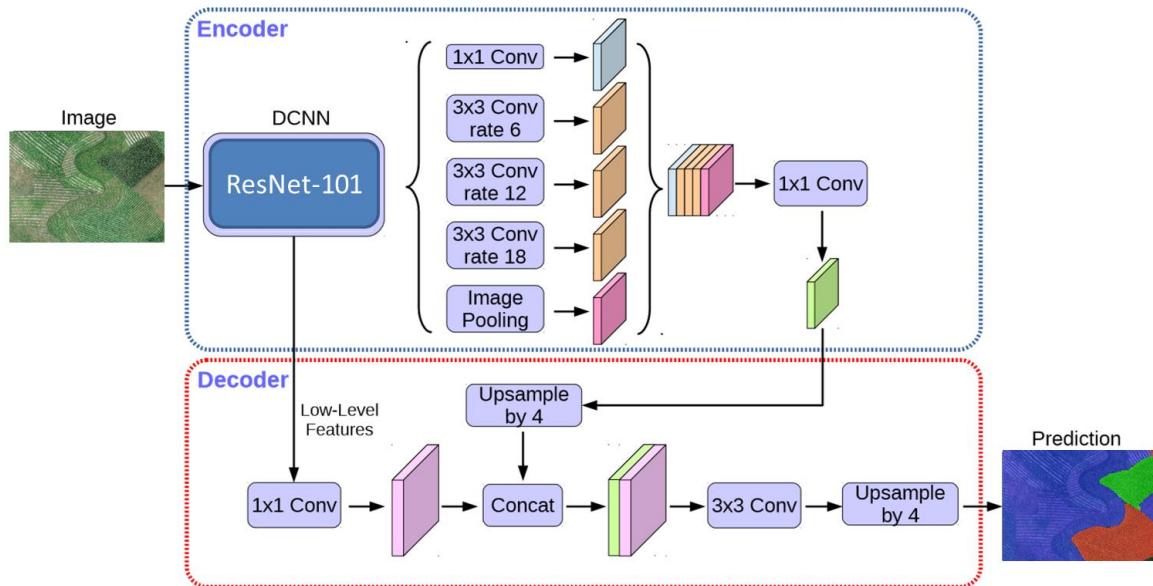


图 3-2 实验采用的 DeepLab-v3+模型

### 3.3 网络模型实现

本实验在 Linux 操纵系统下，运用 Python 语言开发，采用 Pytorch 框架进行训练。Pytorch 是 Torch 的 Python 版本，是由 Facebook 开源的神经网络框架，专门针对 GPU 加速的深度神经网络（DNN）编程。Torch 是一个经典的对多维矩阵数据进行操作的张量（Tensor）库，在机器学习和其他数学密集型应用有广泛应用。与 Tensorflow 的静态计算图不同，Pytorch 计算图是动态的，可以根据计算需要实时改变计算图，利于模型迭代，并且 Pytorch 是面向 Python 语言开发，内部语法规则基本与 Python 一致，有利于对框架的理解、学习。

由于本实验是要处理高分辨率图像，最大的图像几乎达到了 50000\*50000 像素，家里普通的笔记本根本无法胜任实验计划，因此，最终决定通过服务器去运行网络，实验的详细硬件配置请看图 3-3。

图 3-4 表示的是本实验设计的 DeepLab-v3+网络的代码实现。

序号	部件编码	型号	描述	单套数量	总数	成交单价 (CNY)	成交总价 (CNY)
1	FusionServer G5500 站点1						
	02311XTW	IT21SPCA10	G530 V5 Purley 2P通用计算模块(2*CPU插槽,24*DIMM 槽, 支持2*NVMe/SAS/SATA,支持2*10GE 板载网卡)	1	1		
	41020679	BC6M81CPU	英特尔至强银牌4110(2.1GHz/8-core/11MB/85W)处理器	2	2		
	06200241	N26DDR402	DDR4 RDIMM内存-32GB-2666MT/s-2Rank(2G*4bit)-1.2V-ECC	2	2		
	02312RBT	N300S1210W4	通用硬盘-300GB-SAS 12Gb/s-10K rpm-128MB及以上-2.5英寸(2.5英寸托架)	2	2		
	02311XUJ	IT2M10ESML	(LSI3108) SAS/SATA RAID卡-RAID0,1,5,6,10,50,60-12Gb/s-1GB Cache-G5500服务器	1	1		
	03032VXT	IT21GHEA01	GP308 异构计算模块(8*PCIe x16全高单槽位或 4*PCIe x16全高双槽位,支持4*3.5" SAS/SATA HDD)	1	1		
	02311PVP	N4000NS127W3	通用硬盘-4000GB-NL SAS 12Gb/s-7.2K rpm-128MB-3.5英寸(3.5英寸托架)	1	1	319,042.88	319,042.88
	06320130	NTV100G32	视频卡-GPU卡-Tesla V100 PCIe-PN:900-2G500-0010-000/32GB HBM2 Memory/900GB/s Bandwidth/ PCIE 3.0 x16-10DE-1DB6-1,250W/ DualSlot/PassiveCooling-无	2	2		
	02312GAS	IT2M01GP308	GP308配套双槽位GPU卡必配附件包(拉手条和电源线等)	1	1		
	03022PKQ	CN21ITGBA	以太网卡-1Gb 电口(Intel I350)-双端口-RJ45-PCIe 2.0 x4	2	2		
	02301293	IT2K01G5500	G5500 机箱(4U,支持半宽节点,带6风扇)-带必发附件	1	1		
	02311XUA	IT21SMMA10	MM510 机框管理模块	1	1		
	02311XUH	IT21RIOA10	EX500 后IO扩展模块(2*PCIe x16半高半长槽位)	2	2		
	02312EGU	PAC2000S12-1	Atlas 2000W交流电源模块备件	2	2		
	21240142	EGUIDER00	4U静态滑轨套件	1	1		
		设备总价					319,042.88
		项目总价					319,042.88

图 3-3 实验硬件环境配置

```

# DeepLabv3+网络
class deeplabv3plus_resnet101(nn.Module):
    def __init__(self,nc=5):
        super(deeplabv3plus_resnet101, self).__init__()
        self.nc = nc
        self.backbone = ResNet_101()
        self.assp1 = ASSP(in_channels=1024)
        self.out1 = nn.Sequential(nn.Conv2d(in_channels=256,
                                           out_channels=256,kernel_size=1,stride=1),nn.ReLU())
        self.dropout1 = nn.Dropout(0.5)
        self.up4 = nn.Upsample(scale_factor=4)
        self.up2 = nn.Upsample(scale_factor=2)

        self.conv1x1 = nn.Sequential(nn.Conv2d(1024,256,1,bias=False),
                                   nn.ReLU())
        self.conv3x3 = nn.Sequential(nn.Conv2d(512,self.nc,1),nn.ReLU())
        self.dec_conv = nn.Sequential(nn.Conv2d(256,256,3,padding=1),
                                     nn.ReLU())

    def forward(self,x):
        x = self.backbone(x)
        out1 = self.assp1(x)
        out1 = self.out1(out1)
        out1 = self.dropout1(out1)
        out1 = self.up4(out1)

        dec = self.conv1x1(x)
        dec = self.dec_conv(dec)
        dec = self.up4(dec)
        contact = torch.cat((out1,dec),dim=1)
        out = self.conv3x3(contact)
        out = self.up4(out)
        return out

```

图 3-4 本实验设计的网络代码实现

## 3.4 网络训练过程

### 3.4.1 设定实验必要参数与超参数

本实验设置网络的初始学习率为 1e-4。本实验使用 Adam 优化器，并通过 CosineAnnealingLR（余弦周期退火，即 Snapshot Ensemble 的核心概念）对后续学习率进行优化，本实验对于 CosineAnnealingLR 的参数设置为：T\_MAX = 12；最小学习率：1e-8；Optimizer: Adam，如图 3-5 所示。

```

# 5.2 LR_SCHEDULER
SOLVER.ENABLE_LR_SCHEDULER = True
SOLVER.LR_SCHEDULER = "CosineAnnealingLR"
# SOLVER.LR_SCHEDULER = "ReduceLROnPlateau"
# SOLVER.LR_SCHEDULER = "StepLR"
SOLVER.LR_SCHEDULER_PATIENCE = 5
SOLVER.LR_SCHEDULER_FACTOR = 1/3
SOLVER.LR_SCHEDULER_REPEAT = 6
SOLVER.T_max = 12

# 5.3 OPTIMIZER
SOLVER.OPTIMIZER_NAME = "Adam"
SOLVER.LEARNING_RATE = 1e-4
SOLVER.SGD_MOMENTUM = 0.9
SOLVER.Adam_weight_decay = 0

```

图 3-5 设置实验参数

设置模型的度量为 Label\_Accuracy\_delete\_edge，用于验证集评估模型，详细代码见图 3-6。

```

class Label_Accuracy_delete_edge(Metric):
    def __init__(self, n_class=5, edge_size=64):
        super(Label_Accuracy_delete_edge, self).__init__()
        self.n_class = n_class
        self.edge_size = edge_size
    # epoch开始之前调用一次
    def reset(self):
        self.step = 0
        self.mean_iu = 0
    # 每次iteration结束时调用
    def update(self, output):
        label_preds, label_trues = output
        # 删除edge
        _, _, h, w = label_preds.shape
        label_preds = label_preds[:, :, :self.edge_size:h-self.edge_size, :self.edge_size:w-self.edge_size]
        label_trues = label_trues[:, :, :self.edge_size:h-self.edge_size, :self.edge_size:w-self.edge_size]
        label_preds = label_preds.max(dim=1)[1].data.cpu().numpy()
        label_preds = [i for i in label_preds]

        label_trues = label_trues.data.cpu().numpy()
        label_trues = [i for i in label_trues]

        hist = np.zeros((self.n_class, self.n_class))
        for lt, lp in zip(label_trues, label_preds):
            hist += _fast_hist(lt.flatten(), lp.flatten(), self.n_class)
        with np.errstate(divide='ignore', invalid='ignore'):
            iu = np.diag(hist) / (
                hist.sum(axis=1) + hist.sum(axis=0) - np.diag(hist)
            )
        mean_iu = np.nanmean(iu)
        self.mean_iu += mean_iu
        self.step += 1
    # epoch结束时调用
    def compute(self):
        return self.mean_iu / self.step

```

图 3-6 模型度量代码实现

对于实验损失函数，使用标签平滑的方法对其进行调整计算，考虑到输入数据有以下几种情况，一是在进行卷积时零填充太多，造成有用的信息缺少，增大分类难度；二是不同类之间的交界点不好确定，训练时梯度不稳定；三是不同类之间在交界处存在只相差几个像素的情况，但对机器来说这两者的输入数据是相似的，但是训练模型时的标签却不一样，造成训练过程的不稳定。

对于这几种情况，采取在图像边缘与类交界处设置过滤带，如图 3-7 所示，这是一个超参数，在本实验中取  $w = 11$ ，在过滤带内的像素视为硬标签，对这些标签做平滑处理，平滑的程度取决于训练的过程中每个 batch 中属于硬标签的像素所占输入数据的比例。

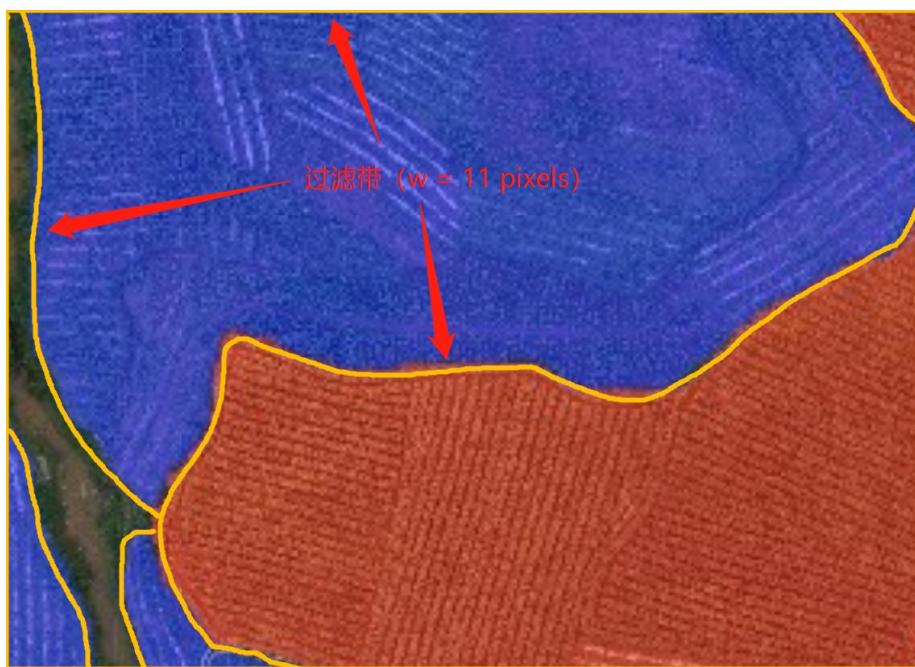


图 3-7 实验中对图像边缘与类交界处设置的过滤带

标签平滑的交叉熵损失函数见公式 2，代码实现见图 3-8。

$$\begin{aligned} H(\mathbf{y}, \mathbf{p}) &= \sum_{k=1}^K -\mathbf{y}_k^{easy} \log(\mathbf{p}_k^{easy}) + \sum_{k=1}^K -\mathbf{y}_k^{hard} \log(\mathbf{p}_k^{hard}) \\ \mathbf{y}_k^{hard} &= \mathbf{y}_k(1 - \alpha) + \alpha/K \end{aligned} \quad (2)$$

式中，参数 K 为分类总数；

$\mathbf{P}_k$  为 Softmax 后的结果；

$\alpha$  来控制标签的平滑程度，其取值为每次输入数据中硬标签占输入数据的比例。

### 3.4.2 网络训练流程

实验中网络训练采用 Pytorch 框架，其 Engine 类的网络训练思维导图如图 3-9 所示。

```

class LabelSmoothing(nn.Module):
    def __init__(self, win_size=11, num_classes=5, smoothing=0.05):
        super(LabelSmoothing, self).__init__()
        assert (win_size%2) == 1
        self.smoothing = smoothing / (num_classes-1)
        self.win_size = win_size
        self.num_classes = num_classes
        self.find_edge_Conv = nn.Conv2d(in_channels=1, out_channels=1,
                                       kernel_size=win_size,
                                       padding=(win_size-1)//2,
                                       stride=1, bias=False)

        new_state_dict = OrderedDict()
        weight = torch.zeros(1,1,win_size,win_size)
        weight = weight - 1
        weight[:, :, win_size//2, win_size//2] = win_size*win_size - 1
        new_state_dict['weight'] = weight
        self.find_edge_Conv.load_state_dict(new_state_dict)
        self.find_edge_Conv.weight.requires_grad=False

    def to_categorical(self, y, alpha=0.05, num_classes=None):
        y = np.array(y, dtype='int')
        input_shape = y.shape
        if input_shape and input_shape[-1] == 1 and len(input_shape) > 1:
            input_shape = tuple(input_shape[:-1])
        y = y.ravel()
        if not num_classes:
            num_classes = np.max(y) + 1
        n = y.shape[0]
        categorical = np.zeros((n, num_classes))
        categorical = categorical + alpha
        categorical[np.arange(n), y] = (1-alpha) + (alpha/self.num_classes)
        output_shape = input_shape + (num_classes,)
        categorical = np.reshape(categorical, output_shape)
        categorical = categorical.transpose(0,3,1,2)
        return categorical

    def forward(self, x, target):
        assert x.size(1) == self.num_classes
        log_p = nn.functional.log_softmax(x, dim=1)
        self.find_edge_Conv.cuda(device=target.device)
        edge_mask = self.find_edge_Conv(target)
        edge_mask = edge_mask.data.cpu().numpy()
        edge_mask[edge_mask!=0] = 1
        self.smoothing = np.mean(edge_mask)
        if self.smoothing > 0.2:
            self.smoothing = 0.2

        target = target.squeeze(dim=1)
        target = target.data.cpu().numpy()
        onehot_mask = self.to_categorical(target, 0, num_classes=self.num_classes)
        onehot_mask = onehot_mask*(1-edge_mask)
        softlabel_mask = self.to_categorical(target, alpha=self.smoothing,
                                             num_classes=self.num_classes)
        softlabel_mask = softlabel_mask*edge_mask
        onehot_mask = torch.from_numpy(onehot_mask).cuda(device=log_p.device).float()
        softlabel_mask = torch.from_numpy(softlabel_mask).cuda(device=log_p.device).float()
        loss = torch.sum(onehot_mask*log_p+softlabel_mask*log_p, dim=1).mean()
        return -loss

```

图 3-8 标签平滑代码实现

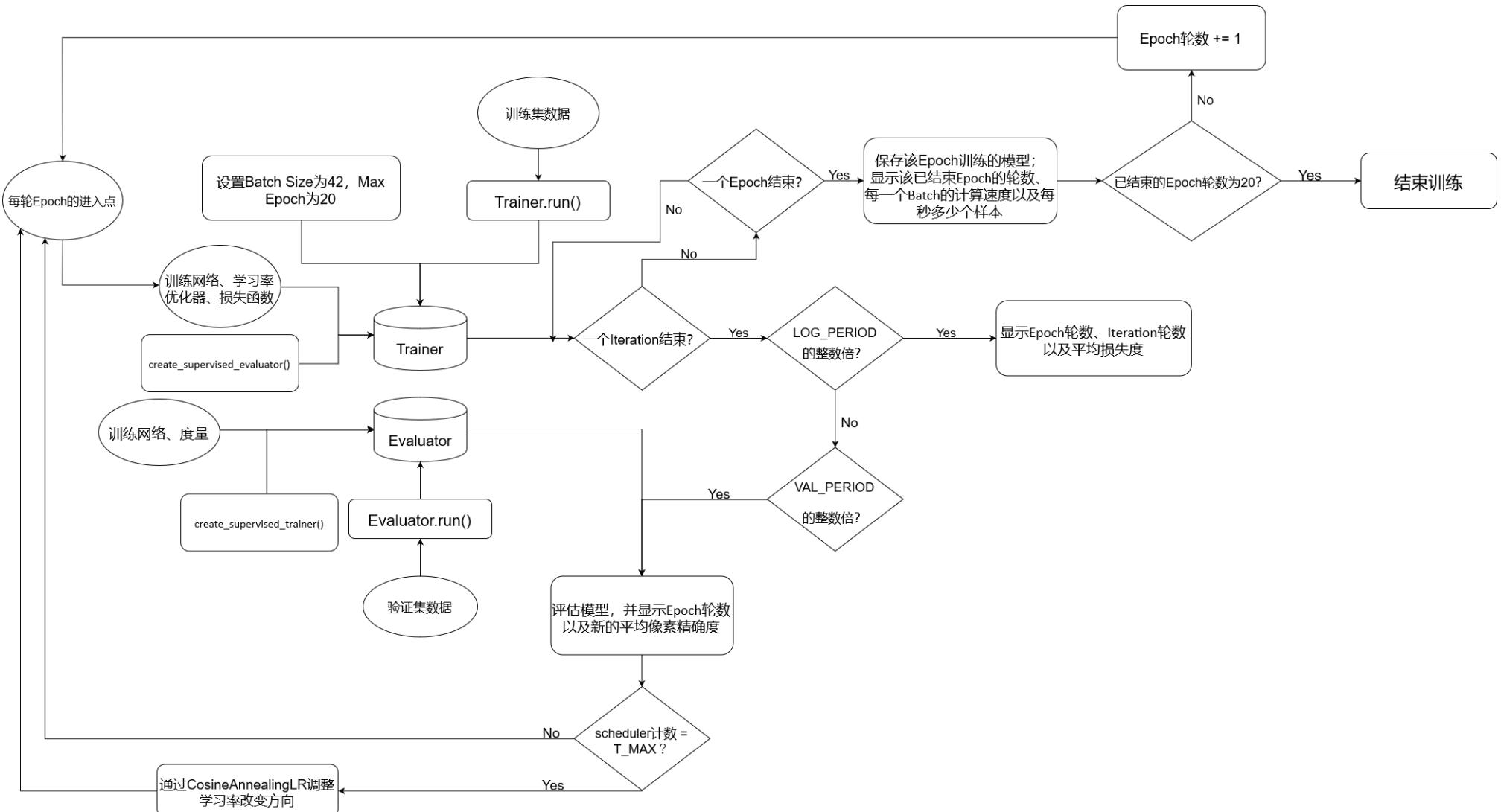


图 3-9 Engine 类的训练模块思维导图

本实验是在 Linux 操作系统上，基于 Pytorch 进行的语义分割项目，首先通过 argparse 库将实验预先设定好的参数输入程序，如图 3-10 所示，具体的内容在代码部分的 config/default.py 文件下。

```
# DATA
# Batch大小是在更新模型之前处理的多个样本，Epoch数是通过训练数据集的完整传递次数
parser.add_argument("--num_workers", "--DATA.DATALOADER.NUM_WORKERS", type=int,
                    help='Num of data loading threads.')
parser.add_argument("--train_batch_size", "--DATA.DATALOADER.TRAIN_BATCH_SIZE", type=int,
                    help="input batch size for training (default:64)")
parser.add_argument("--val_batch_size", "--DATA.DATALOADER.VAL_BATCH_SIZE", type=int,
                    help="input batch size for validation (default:128)")
parser.add_argument("--train_csv_file", "--DATA.DATASET.train_csv_file", type=str)
parser.add_argument("--train_root_dir", "--DATA.DATASET.train_root_dir", type=str)
parser.add_argument("--train_mask_dir", "--DATA.DATASET.train_mask_dir", type=str)
parser.add_argument("--val_csv_file", "--DATA.DATASET.val_csv_file", type=str)
parser.add_argument("--val_root_dir", "--DATA.DATASET.val_root_dir", type=str)
parser.add_argument("--val_mask_dir", "--DATA.DATASET.val_mask_dir", type=str)
```

图 3-10 通过 argparse 编写命令行接口，利于后面程序调用参数

载入实验参数、超参数，创建 Dataloader，形成网络，准备进行迭代运算，如图 3-11 所示。

```
def train(cfg):
    #建立学习网络使用deeplabv3resnet101
    model = build_model(cfg)
    # 通过nn.dataParallel并行加速计算
    model = nn.DataParallel(model)
    # 开启GPU并行加速计算
    torch.backends.cudnn.benchmark = True
    # 确定学习率的优化器
    optimizer = make_optimizer(cfg, model)
    # 确定损失函数
    criterion = make_criterion(cfg)
    # 调整学习率
    scheduler = make_lr_scheduler(cfg, optimizer)
    # 设置模型评估标准
    metrics = make_metrics(cfg)
    # 载入数据
    train_loader = make_dataloader(cfg, is_train=True)
    val_loader = make_dataloader(cfg, is_train=False)

    do_train(cfg, model=model, train_loader=train_loader,
             val_loader=val_loader, optimizer=optimizer,
             scheduler=scheduler, loss_fn=criterion, metrics=metrics)
```

图 3-11 载入实验参数、超参数

在训练过程中，通过函数 create\_supervised\_evaluator() 和 create\_supervised\_trainer() 创建 trainer 和 evaluator。通过装饰器为一些事件注册函数，首先我们提前设置一个 Batch size 的大小为 21，最大 Epoch 为 20，我们为了让实验结果更容易阅读以及对学习率进行调整，我们设置了两个参数，分别是 LOG\_PERIOD = 0.03 和 VAL\_PERIOD = 1.00。

当一个 Iteration 结束时，网络首先判断该轮运算是否计算到了上面两个参数的整数倍，若为 LOG\_PERIOD 的整数倍，则返回 Epoch 轮数、Iteration 轮数以及平均损失度，代码实现如图 3-12 所示：

```
@trainer.on(Events.ITERATION_COMPLETED)
def log_training_loss(engine):
    len_train_loader = len(train_loader)
    log_period = int(cfg.LOG_PERIOD*len_train_loader)
    iter = (engine.state.iteration-
1)%len_train_loader + 1 + engine.state.epoch*len_train_loader
    if iter % log_period == 0:
        iter = (engine.state.iteration-1)%len_train_loader + 1
        logger.info("Epoch[{}] Iteration[{} / {}] Loss {:.7f}".format(engine.state.epoch,
                                                               iter, len_train_loader,
                                                               engine.state.metrics['avg_loss']))
```

图 3-12 每一组 Iteration 结束，终端返回数据代码实现

若为 VAL\_PERIOD 的整数倍，则用验证集评估模型，并打印出新的学习率，而 scheduler.step() 会进行一次计数，如果计数总数达到 12 次，即 T\_MAX 的值，则通过 CosineAnnealingLR 调整学习率改变的方向（为了防止模型沉降到局部极小），返回 Epoch 轮数以及该 Epoch 的平均像素精确度，之后等待该 Epoch 结束进入下一个 Epoch，代码实现见图 3-13。

```
# 验证集评估模型
evaluator.run(val_loader)
metrics = evaluator.state.metrics
avg_loss = metrics["pixel_error"]
logger.info("Validation Result -
Epoch: {} Avg Pixel Accuracy: {:.7f} ".format(engine.state.epoch,
                                                avg_loss))

# 调整学习率
elif cfg.SOLVER.LR_SCHEDULER == "CosineAnnealingLR":
    lr = optimizer.state_dict()['param_groups'][0]['lr']
    scheduler.step()
    new_lr = optimizer.state_dict()['param_groups'][0]['lr']
```

图 3-13 学习率调整及验证集评估代码实现

当一个 Epoch 结束时，由于使用 Snapshot Ensemble 的方法来进行预测推理，需要在训练集训练的时候进行模型自融合，所以我们不得不保存每一个 Epoch 的模型，代码实现如图 3-14，并且返回该已结束 Epoch 的轮数、每一个 Batch 的计算速度以及每秒多少个样本。

```

@trainer.on(Events.EPOCH_COMPLETED)
def save(engine):
    epoch = engine.state.epoch
    print("epoch: "+str(epoch))
    if epoch%1 == 0:
        model_name=os.path.join(cfg.OUTPUT.DIR_NAME+"model/",
                               "epoch_"+str(engine.state.epoch) +
                               "_" +cfg.TAG+"_" +
                               cfg.MODEL.NET_NAME+".pth")
        torch.save(model.module.state_dict(),model_name)

```

图 3-14 保存每轮 Epoch 模型

当第 40 轮 Epoch 计算结束时，结束训练，取得实验模型。

### 3.5 评价指标

本实验采用平均交并比（Mean Intersection over Union）作为测试结果的评价标准，即求出每一类的 IOU 取平均值。IOU 指的是，真实标签和预测结果的两块区域交集/并集。评估只考虑“烤烟”、“玉米”、“薏仁米”、“人造建筑”四种类型。针对每种作物所有的预测结果，统计每个类别的真实标签和预测结果，计算 IOU，最后取平均，MIOU 的计算公式见公式 3，代码实现如图 3-15。

```

# a是转化成一维数组的标签，形状(H×W,); b是转化成一维数组的标签，形状(H×W,); n是类别数目，实数（在这里为5）
def fast_hist(a, b, n):
    # k是一个一维bool数组，形状(H×W,); 目的是找出标签中需要计算的类别（去掉了背景类数据）
    k = (a >= 1) & (a < n)
    # np.bincount计算了从0到n**2-1这n**2个数中每个数出现的次数，返回值形状(n, n)
    return np.bincount(n * a[k].astype(int) + b[k], minlength=n ** 2).reshape(n, n)

# 分别为每个类别（在这里是4类）计算mIoU，hist的形状(n, n)
def per_class_iu(hist):
    # 矩阵的对角线上的值组成的一维数组/矩阵的所有元素之和，返回值形状(n,)
    return np.diag(hist) / (hist.sum(1) + hist.sum(0) - np.diag(hist))

```

图 3-15 MIOU 代码实现

$$\text{MIOU} = \text{TP} / (\text{FP} + \text{FN} + \text{TP}) \quad (3)$$

式中，

**TP:** T (预测对了 true) P(预测为正样本 positive); 真的正值，说明被预测为正样本，预测是真的，即真实值为正样本。

**TN:** T (预测对了 true) N(预测为负样本 negative); 真的负值，说明被预测为负样本，预测是真的，即真实值为负样本。

**FP:** F (预测错了 false) P(预测为正样本 positive); 假的正直：说明被预测为正样本，但预测是假的，即真实值为负样本。

**FN:** F (预测错了 false) N(预测为负样本 negative); 假的负值，说明被预测为负样本，但预测是假的，即真实值为正样本。

### 3.6 本章小结

由于农作物遥感图像有着类间分布不均和类内时间不统一的特点，因此最终决定通过当前比较先进的 Deeplab-v3+语义分割网络进行训练。本实验在 Linux 系统环境下，基于 Pytorch 提供的框架，通过 argparse 库输入网络训练必要的参数与超参数，将 Snapshot Ensemble 与 DeepLab-v3+相结合，以补偿由于数据过于庞大的模型计算效率问题，对数据的边缘部分采用标签平滑的方式，以提升模型精度。在训练过程中，实时记录每个 Iteration 的平均损失度与每个 Epoch 的训练时长等关键信息，通过验证集评估每个 Epoch 结束时的模型并保存，根据余弦退火原理在 Epoch 结束时调整学习率改变方向，同时反馈当前 Epoch 的平均像素精确度。最终选定的实验模型将通过 MIOU（平均交并比）评价指标来评估模型的预测结果，即求出每一类的 IOU 取平均值。IOU 指的是，真实标签和预测结果的两块区域交集/并集。其中评估内容只考虑“烤烟”、“玉米”、“薏仁米”、“人造建筑”四种类型。

## 4 实验过程与结果分析

### 4.1 准备数据集

本文首先选取了具有独特的地理环境、气候条件以及人文特色的贵州省兴仁市作为研究区域，以当地的优势产业和支柱产业作为实验对象，通过无人机航拍的地面影像获取实验数据，进行语义分割，识别薏仁米、玉米、烤烟、人造建筑四大类型，实验所提供的航拍影像的标签为与原始图像 1:1 大小的单通道图像，其中“烤烟”的标签值为 1，“玉米”为 2，“薏仁米”为 3，“人造建筑”为 4，背景类为 0。

首先将无人机航拍的图像进行切分，这样有利于机器学习。采用以 1024\*1024 的大小进行图像切割，步长为 512，如果切割的边界大小不足 1024 则填充至步长的整数倍，并且对每一个切分出来的图像填充二分之一长度的外边框，用于预测推理时的膨胀预测，代码实现如图 4-1 所示，最后对分割好的数据按照 6:2:2 进行归类，见表 4-1 所示，分别为训练集、验证集、测试集，图 4-2 为完成数据分割后的每张图像的坐标信息，从左到右依次为切分后的图片名称、切分后图片左上角的纵坐标，切分后图片左上角的横坐标、补足至 1024 长度后图片右下角的纵坐标以及横坐标。切分后的图片如图 4-3 所示。

```
# 填充外边界至步长整数倍
target_w,target_h = target_size
h,w = image.shape[0],image.shape[1]
new_w = (w//target_w)*target_w if (w//target_w == 0) else (w//target_w+1)*target_w
new_h = (h//target_h)*target_h if (h//target_h == 0) else (h//target_h+1)*target_h
image = cv.copyMakeBorder(image,0,new_h-h,0,new_w-w,cv.BORDER_CONSTANT,0)
label = cv.copyMakeBorder(label,0,new_h-h,0,new_w-w,cv.BORDER_CONSTANT,0)

# 填充外边框至二分之一stride长度
h,w = image.shape[0],image.shape[1]
new_w,new_h = w + stride,h + stride
image = cv.copyMakeBorder(image,stride//2,stride//2,stride//2,stride//2,cv.BORDER_CONSTANT,0)
label = cv.copyMakeBorder(label,stride//2,stride//2,stride//2,stride//2,cv.BORDER_CONSTANT,0)
```

图 4-1 数据切割部分关键代码

表 4-1 数据集汇总

数据集	数量（单位：张）	比例 <sup>1)</sup>
训练集	8299	6
验证集	2915	2
测试集	2539	2

注：1) 比例已四舍五入至整数。

	A	B	C	D	E
1	image_2_0.png	0	1024	1024	2048
2	image_2_1.png	0	1536	1024	2560
3	image_2_2.png	0	2048	1024	3072
4	image_2_3.png	0	2560	1024	3584
5	image_2_4.png	0	3072	1024	4096
6	image_2_5.png	0	3584	1024	4608
7	image_2_6.png	0	4096	1024	5120
8	image_2_7.png	0	4608	1024	5632
9	image_2_8.png	0	5120	1024	6144
10	image_2_9.png	0	5632	1024	6656
11	image_2_10.png	0	6144	1024	7168
12	image_2_11.png	0	6656	1024	7680
13	image_2_12.png	512	0	1536	1024
14	image_2_13.png	512	512	1536	1536
15	image_2_14.png	512	1024	1536	2048
16	image_2_15.png	512	1536	1536	2560
17	image_2_16.png	512	2048	1536	3072
18	image_2_17.png	512	2560	1536	3584
19	image_2_18.png	512	3072	1536	4096
20	image_2_19.png	512	3584	1536	4608
21	image_2_20.png	512	4096	1536	5120

图 4-2 完成数据分割后每张图像的坐标信息

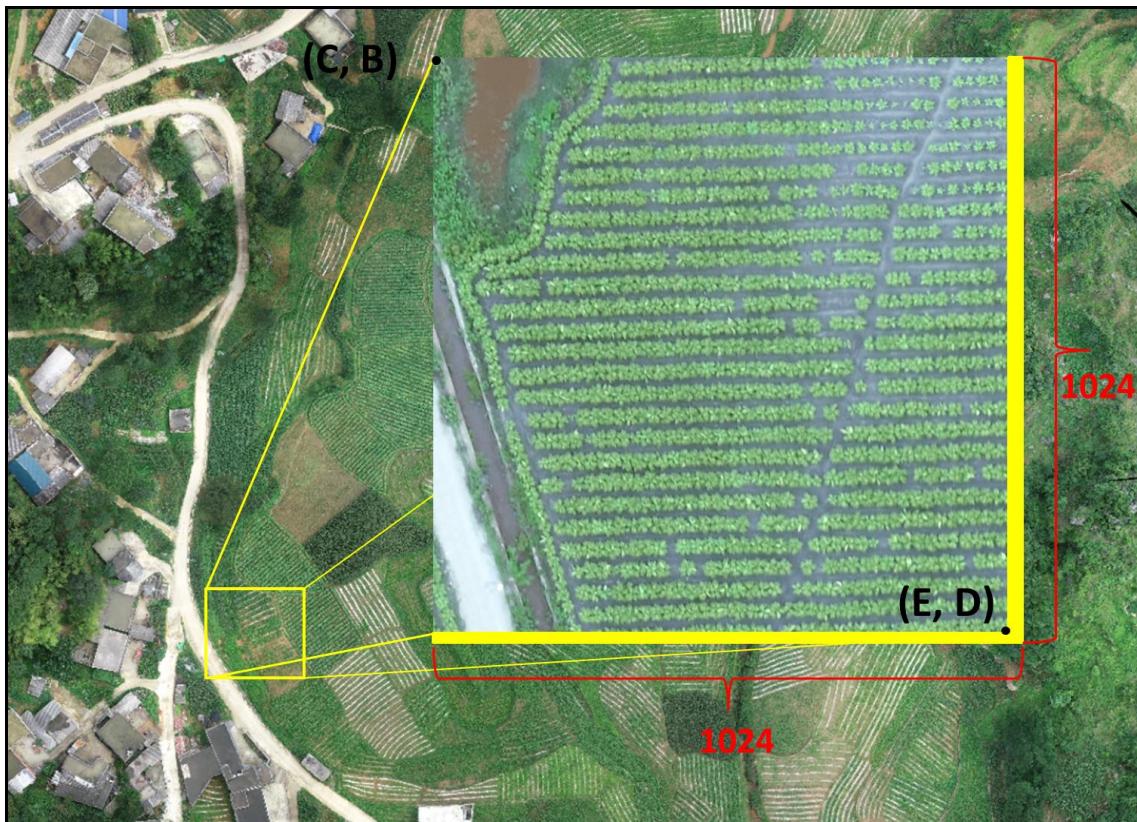


图 4-3 切分后的图片

## 4.2 实验过程

将数据集载入成功后，运行网络等待每轮 Epoch 结束，在训练的过程中观察每轮 Epoch 结束后验证集验证模型的反馈像素精确度的结果与每轮训练过程中每组 Iteration 结束后的平均损失度，以及每轮 Epoch 后计算机对学习率的调整，结合这三者挑选出最优的每轮 Epoch 结束后保存的模型。

挑选出最优模型后，便进入了测试阶段。

当取得网络训练出来的模型后，跟训练模型时一样，通过 argparse 库编写命令行接口，输入测试必要的参数与环境，如图 4-4 所示。

```
def load_arg():
    parser = ArgumentParser(description="Pytorch Hand Detection Training")
    parser.add_argument("-config_file", "--CONFIG_FILE", type=str,
                        help="Path to config file")
    parser.add_argument('-model', '--MODEL.NET_NAME', type=str,
                        help="Net to build")
    parser.add_argument('-path', '--MODEL.LOAD_PATH', type=str,
                        help="path/file of a pretrain model(state_dict)")
    parser.add_argument("-device", "--MODEL.DEVICE", type=str,
                        help="cuda:x (default(cuda:0))")
    parser.add_argument("-image_n", "--TOOLS.image_n", type=int, default=3)
    parser.add_argument("-save_path", "--TOOLS.save_path", type=str, required=True)
    arg = parser.parse_args()
    return arg
```

图 4-4 通过 argparse 库对参数初始化

将训练好的模型和测试集载入后，由于使用 Snapshot Ensemble，余弦周期退火的学习率调整策略，训练网络对每个收敛到局部最小的模型进行保存，最终通过模型自融合不断训练，提升最终效果。

#### 4.2.1 增强测试

为了提高预测结果的精确度，通过对图像进行旋转、翻转等多次操作，对不同角度的图像进行预测，最后对预测结果进行总和，获取最优的结果，代码实现如图 4-5 所示。

```
# 通过模型自融合提升模型效果
for model in model_list:
    # 通过对图像水平翻转，垂直翻转，水平垂直翻转等多次预测，提高预测精度
    predict_1 = model(image)
    predict_list = predict_1
    predict_2 = model(torch.flip(image, [-1]))
    predict_2 = torch.flip(predict_2, [-1])

    predict_3 = model(torch.flip(image, [-2]))
    predict_3 = torch.flip(predict_3, [-2])

    predict_4 = model(torch.flip(image, [-1, -2]))
    predict_4 = torch.flip(predict_4, [-1, -2])

    predict_list += (predict_1 + predict_2 + predict_3 + predict_4)
predict_list = torch.argmax(predict_list.cpu(), 1).byte().numpy()
```

图 4-5 对训练集进行预测推理后的增强测试

#### 4.2.2 膨胀预测

由于直接做不重叠的滑窗预测拼接，会造成预测结果的拼接痕迹明显，这是由于在卷

积网络计算时，网络为了保持图像的分辨率绘制了大量的Zero-padding，这使得网络对边缘像素的预测精度下降。本实验中，为了缓解这种问题的发生，采用膨胀预测的方法，即在预测时，只保留预测结果的中心区域，舍弃预测不准的边缘图像，这也是在实验一开始处理数据的时候对每张图像进行扩充二分之一个步长的原因，在此处扩充的部分将被舍弃，如图 4-7 所示。

具体实现方法，代码实现见图 4-6：

- (1) 填充右下边界至滑窗预测窗口大小的整数倍，方便对初始数据的图像分割
- (2) 填充二分之一步长大小的外边框，为膨胀预测做准备
- (3) 以 1024\*1024 作为滑窗的大小，步长为 512，每次预测只保留图像中心的 512\*512 大小的窗口的预测结果。

```
# 获取Batch的大小
batch_size = predict_list.shape[0]
for i in range(batch_size):
    predict = predict_list[i]
    pos = pos_list[i,:]
    [topleft_x,topleft_y,buttonright_x,buttonright_y] = pos

    if(buttonright_x-topleft_x)==1024 and (buttonright_y-topleft_y)==1024:
        png[topleft_y+256:buttonright_y-256,topleft_x+256:buttonright_x-256] = predict[256:768,256:768]
    else:
        raise ValueError("target_size!=512, Got {},{}".format(buttonright_x-topleft_x,buttonright_y-topleft_y))
```

图 4-6 膨胀预测代码实现

之后等待计算机计算完成，获取现阶段测试结果。

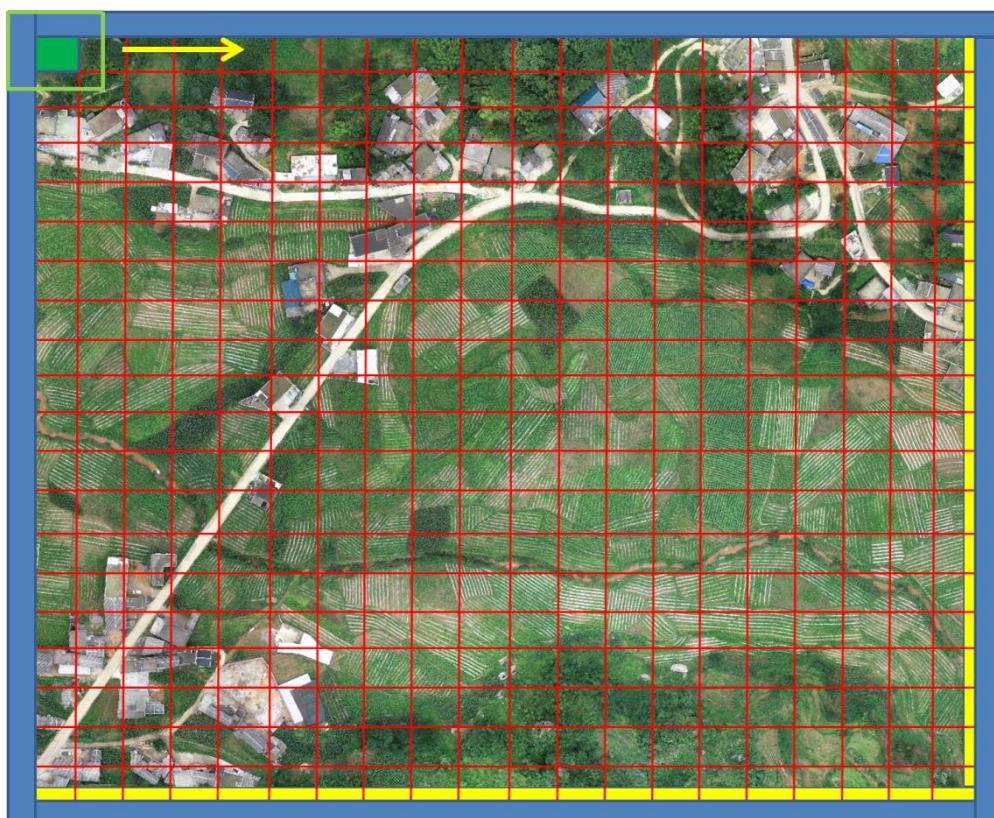


图 4-7 膨胀预测直观图

### 4.3 实验结果

训练模型的过程中，对不同 Epoch 反馈的模型结果进行了记录，见表 4-2。

**表 4-2 模型训练过程数据记录**

Epoch 轮数	平均像素精度	花费时间	调整后的学习率	Epoch 结束时的学习率
1	0.6654702	00:11:25	9.829646168532197e-05	0.0001
2	0.6629608	00:12:07	9.330194006220303e-05	9.829646168532197e-05
3	0.6746247	00:11:58	8.535680352542144e-05	9.330194006220303e-05
4	0.6896188	00:12:05	7.50025e-05	8.535680352542144e-05
5	0.7030398	00:12:50	6.294465815990053e-05	7.50025e-05
6	0.7079460	00:12:43	5.000500000000001e-05	6.294465815990053e-05
7	0.7159000	00:12:00	3.706534184009949e-05	5.000500000000001e-05
8	0.7066874	00:12:06	2.500750000000002e-05	3.706534184009949e-05
9	0.7121209	00:12:03	1.4653196474578565e-05	2.500750000000002e-05
10	0.7088854	00:11:57	6.7080599377969894e-06	1.4653196474578565e-05
11	0.7045086	00:12:20	1.713538314678036e-06	6.7080599377969894e-06
12	0.7051369	00:12:14	1e-08	1.713538314678036e-06
13	0.7054096	00:12:43	1.7135383146780298e-06	1e-08
14	0.6984439	00:12:23	6.708059937796982e-06	1.7135383146780298e-06
15	0.7012622	00:12:21	1.4653196474578538e-05	6.708059937796982e-06
16	0.7015889	00:12:46	2.500749999999998e-05	1.4653196474578538e-05
17	0.6911366	00:12:10	3.7065341840099486e-05	2.500749999999998e-05
18	0.6946755	00:12:29	5.000499999999999e-05	3.7065341840099486e-05
19	0.6830981	00:12:24	6.29446581599005e-05	5.000499999999999e-05
20	0.6869380	00:12:02	7.500249999999999e-05	6.29446581599005e-05
21	0.6507225	00:12:26	8.535680352542143e-05	7.500249999999999e-05
22	0.7266806	00:12:42	9.330194006220299e-05	8.535680352542143e-05
23	0.6546131	00:12:03	9.829646168532195e-05	9.330194006220299e-05
24	0.6724493	00:12:51	9.99999999999998e-05	9.829646168532195e-05
25	0.6898540	00:12:11	9.829646168532194e-05	9.99999999999998e-05
26	0.6985153	00:12:17	9.330194006220299e-05	9.829646168532194e-05
27	0.6825399	00:12:14	8.535680352542143e-05	9.330194006220299e-05

28	0.7052590	00:12:08	7.50025e-05	8.535680352542143e-05
29	0.6700831	00:12:04	6.294465815990051e-05	7.50025e-05
30	0.6696351	00:12:13	5.000500000000005e-05	6.294465815990051e-05
31	0.6829142	00:12:23	3.70653418400995e-05	5.000500000000005e-05
32	0.6785885	00:12:08	2.5007500000000038e-05	3.70653418400995e-05
33	0.6715707	00:12:27	1.4653196474578536e-05	2.5007500000000038e-05
34	0.6663265	00:12:16	6.708059937796981e-06	1.4653196474578536e-05
35	0.6739585	00:12:12	1.7135383146780298e-06	6.708059937796981e-06
36	0.6660800	00:12:07	1e-08	1.7135383146780298e-06
37	0.6826406	00:12:07	1.7135383146780298e-06	1e-08
38	0.6712157	00:12:25	6.708059937796987e-06	1.7135383146780298e-06
39	0.6727166	00:12:01	1.4653196474578559e-05	6.708059937796987e-06
40	0.6798265	00:12:24	2.5007500000000096e-05	1.4653196474578559e-05

综合考虑，最终选取了第 22 轮 Epoch 保存的模型，因为该模型反馈的像素精确度是最高的，而且根据 Snapshot Ensemble 原理，训练出的模型不易由于沉降到局部极小而不具有泛化能力，并且该轮中 Iteration 反馈的平均损失度较低并趋于稳定，图 4-8 为训练保存的模型。

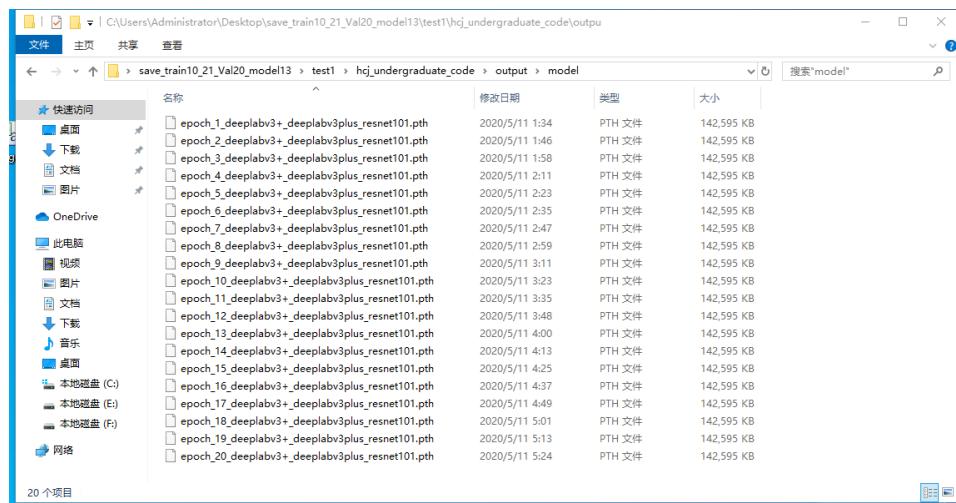


图 4-8 训练模型保存结果

在选择了模型之后，我们把选好的模型与测试集载入计算机，进行推理预测，等待计算机完成预测获取现阶段实验结果，如图 4-9 所示。

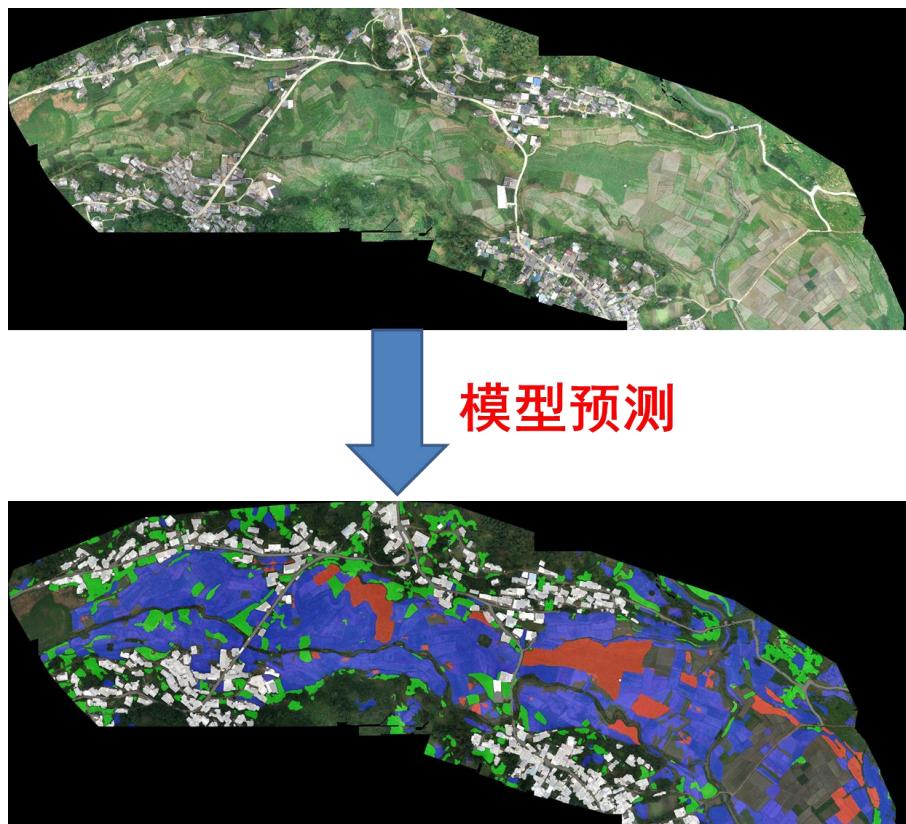


图 4-9 现阶段实验结果

到此处为止，对实验对象的预测大体上已经完成，但是预测的实验结果仍然存在一些问题，如在同一个类内存在孔洞以及小连通域，本实验仅仅采取了 skimage.morphology 库内所提供的移除小连通域以及填充孔洞的方法，见图 4-10、4-11、4-12。等待计算机线程处理事件完成，获取最终测试结果，如图 4-13 所示。

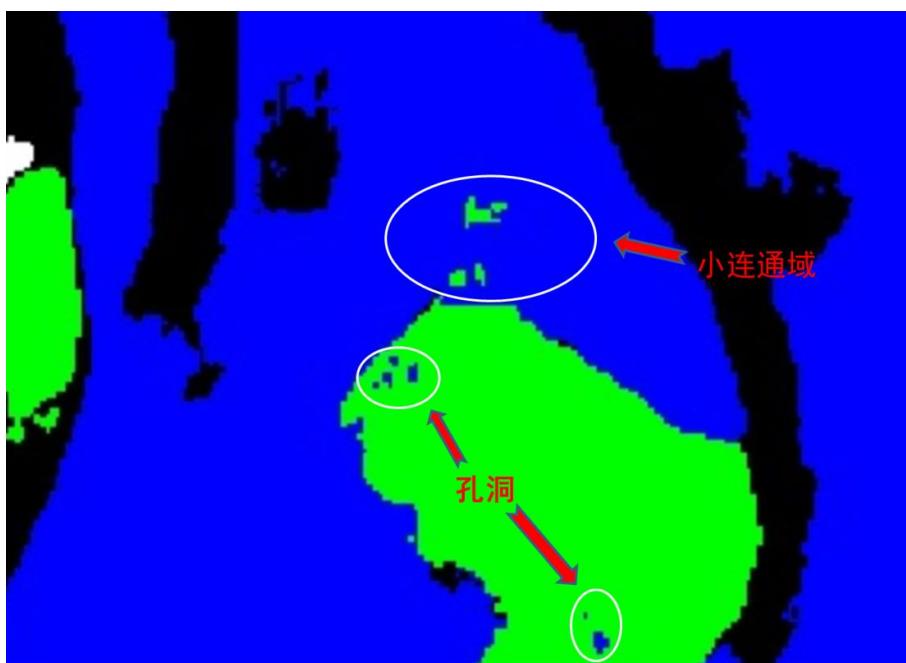


图 4-10 预测结果的小连通域及孔洞

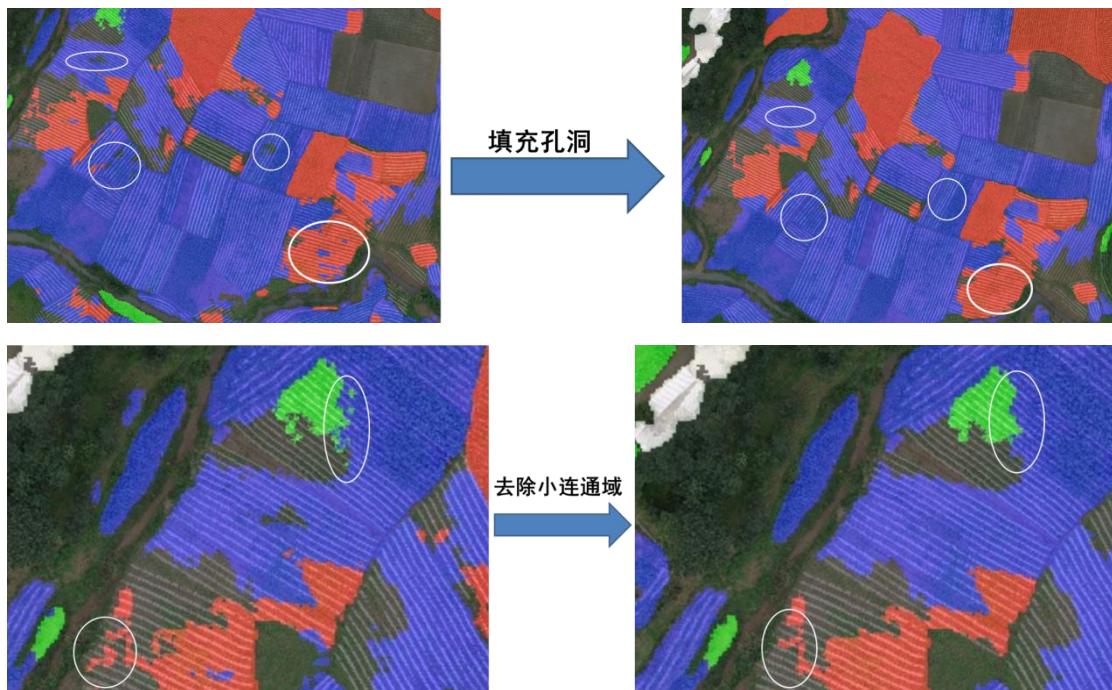


图 4-11 对预测结果的小连通域及孔洞进行处理

```

def remove_small_objects_and_holes(class_type,label,min_size,area_threshold,in_place=True):
    print("----- class_n : {} start -----".format(class_type))
    label = remove_small_objects(label==1,min_size=min_size,
                                 connectivity=1,
                                 in_place=in_place)
    label = remove_small_holes(label==1,area_threshold=area_threshold,
                               connectivity=1,
                               in_place=in_place)
    print("----- class_n : {} finished -----".format(class_type))
    return label

```

图 4-12 代码实现填充孔洞和去除小连通域

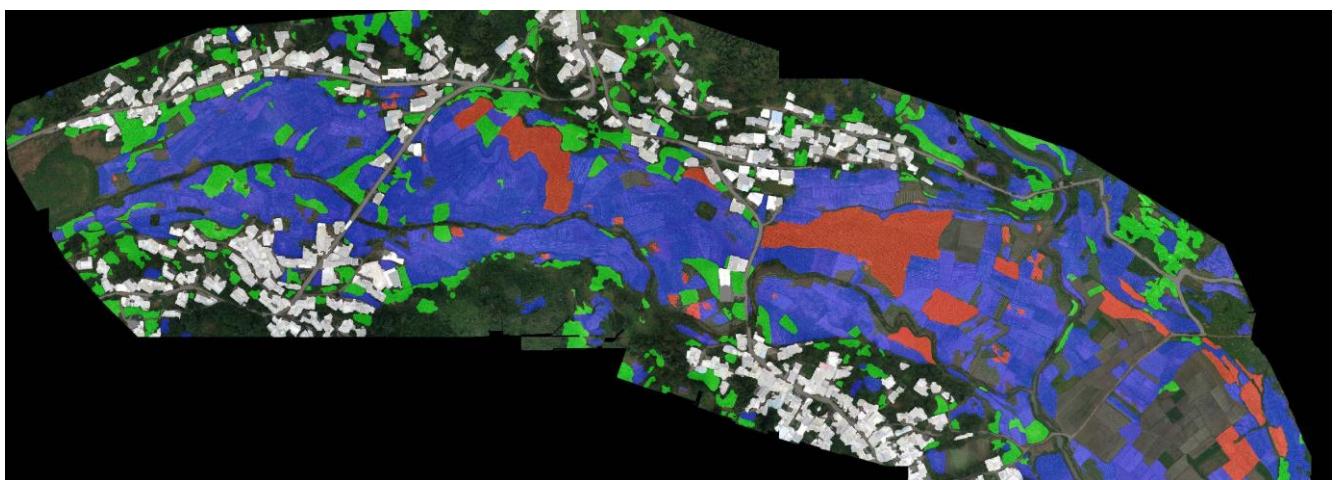


图 4-13 实验最终预测结果

## 4.4 实验结果分析

### 4.4.1 MIOU 评价结果

本实验结果的 MIOU 值为 78.66，其中对薏米以及烤烟的预测率达到了 83.2 和 80.42

的较高水准，如图 4-14 所示，总体上说本次实验非常成功，达到了预期的目标。

```
(db) [root@localhost MIOU]# python MIOU_test.py
====> Smoke:      80.42
====> Corn:       76.39
====> BarleyRice: 83.2
====> Building:   74.63
====> mIoU: 78.66
```

图 4-14 实验 MIOU 评价结果

#### 4.4.2 本实验方法的优点

本实验采取的将 DeepLab-v3+网络、Snapshot Ensemble、Label Smoothing 这三方面相结合的方法，相比于当前比较流行的深度学习网络，例如，ResNet、AlexNet、GoogleLeNet 等等，该方法收敛速度更快，对农作物语义分割的判断更加精准。其中，在 2019 年提出的 DeepLab-v3+网络本身就是集成了各个深度学习网络的优势与精髓，通过空间金字塔模块（ASPP）与编码-解码结构相结合，既保证了模型训练的速度又在一定程度上提高模型识别精度。而 Snapshot Ensemble 方法通过余弦退火进一步提高模型收敛速度并且确保了模型沉降到全局最小的概率。最后 Label Smoothing 方法则根据农作物遥感语义分割的特点对边缘数据进行标签平滑处理，使得这些对计算机来说不易识别的数据更加凝聚可分，提高语义分割的精度。正是这些算法的优点汇聚在一起才铸造了本次农作物遥感语义分割实验的成功。

## 5 总结与展望

### 5.1 结论

本实验通过 DeepLab-v3+ 网络对贵州省兴仁市农作物地区的高分辨率图像进行了语义分割任务。其中在训练模型过程中，将 Snapshot Ensemble 与 DeepLab-v3+ 网络相结合，通过余弦退火的方法调整网络学习率，实验模型收敛速度，并利用将实验损失函数加入 Label Smoothing 方法，对训练数据设定过滤带，将图像边缘与类间交界处数据设为硬标签，进行标签平滑操作以提高模型精度。在测试模型过程中，采用了膨胀预测方法，仅仅对实验数据中心附近位置保留预测结果，周边预测不准边缘数据舍弃，提升预测精度；并对预测图像采用水平翻转、垂直翻转、水平垂直翻转等多方位预测的方式进行测试增强。最终实验采用 MIOU 评价标准，总体评价结果为 78.66。

本实验探究了 DeepLab-v3+ 网络与 Snapshot Ensemble、Label Smoothing 这两种算法相结合对农作物遥感图像语义分割的可行性，其中 Snapshot Ensemble 算法对模型的收敛速度有了一定的提升，获取最优模型仅仅花费了四个多小时，并且由于该算法使用余弦退火的方法调整学习率，使模型更容易脱离局部最小，沉降到全局最小，相比于没有使用该算法的训练网络，取得的模型更具有稳定性，免去了长时间的比较模型的准确度、收敛速度以及泛化能力。而 Label Smoothing 算法对训练集的边缘数据进行了标签平滑的操作，减少了边缘数据预测不准的情况，提高预测精度，尤其对训练集中薏仁米的预测 IOU 分数达到了 83.2，这是因为当地的主要农作物为薏仁米，是支柱产业，对薏仁米种植的规划较为详细，在种植过程中人工干预，使得该种类的边缘数据较为整齐，有利于进行标签平滑和网络学习。

本实验仍然有许多可以改进的地方，比如说将第一次训练出来的模型去测试一组没有标签的数据，将返回的结果通过多组不同置信度阀值过滤数据，再让这组数据以及预测出的标签结合训练集重新训练模型，即将预测出的数据设定为伪标签，在第二次训练的过程中对所有伪标签进行标签平滑处理，该方法可以用来缓解由于伪标签内的错误数据对训练过程的影像，并且实验通过多个不同的 Snapshot Ensemble 方法自融合训练出来的模型，提高模型的泛化能力，不断重复上面的步骤直到模型的精确度基本稳定，这个方法在没有使模型过拟合的情况下，可以使其提分显著。

总体来说实验非常成功，达到了实验预期效果，证明了 DeepLab-v3+ 网络、Snapshot Ensemble、Label Smoothing 这三方面相结合对农作物遥感图像语义分割的可行性。

## 5.2 农作物遥感图像语义分割存在的问题

### 5.2.1 农业遥感技术仍存在不足

农业遥感检测技术历经了几十年的探索，逐渐取得了不少的技术成就。然而，农作物遥感分类精度仍然极大的影响着农作物种植面积检测精度，这与农作物分类特征密不可分。目前为止，农作物遥感分类特征变量有两个问题有待解决，一是对农作物遥感分类特征变量基础理论不足；二是对农作物遥感分类特征变量应用仍有问题<sup>[4]</sup>。对于标签传播现在有两种主要的传播方法：补丁传播匹配、对光学的跟踪。但是，第一种方法对补丁大小与阈值比较敏感，在一定的条件下，假设对类统计有一定知识积累；第二种方法对精确的光学跟踪计算要求极高，难以达到预期效果。在错误的跟踪计算条件下，这会导致传播的标签与图像相对位置的帧无法对齐<sup>[16]</sup>。

### 5.2.2 DeepLab-v3+缺陷

在原始的 DeepLab 模型内，其参数量非常多，其中在比较靠后的卷积层中尤为明显，512 个卷积核被使用在 conv4 和 conv5 的卷积层中，然后每一个卷积核的参数数量有 236 万个，这其中还不包括 conv4\_1 层内的 118 万个参数。如此庞大的参数量大大减缓了网络训练过程中的运算速度，而且在遥感数据集内有标签的数据有限，因此在训练样本较少，但网络参数庞大的条件下，过拟合的现象经常发生<sup>[5]</sup>。

## 5.3 农作物遥感图像语义分割未来展望

在农业遥感图像语义分割的技术中，遥感技术有着多波段、多维度、多时相的特点，这可以对观测数据提供大量支持，并且可以实时获取地表特征，比如说植被指数、地表亮度指数和地表辐射量等等，并在定量反演的过程中，进一步得到一些更详尽的参数，例如，地表反射率、植被面积指数、植被叶绿素含量、水分含量等等。但是在农作物遥感监测中，农作物高度、种植面积、农作物数量等属性是连续变化的过程，仅仅依靠遥感数据是无法保证数据的连续性。因此，现在人们在将农业专业模型如农作物成长模型、地表能量均衡模型等模型与遥感监测取得的数据进行同化，来弥补遥感监测中对时间连续变化无法精确观测的缺陷<sup>[6]</sup>，在这一方面人类已经取得了长足的进展，在未来农作物遥感语义分割必将对农作物种植等各个方面有着举足轻重的作用。

## 致 谢

大学四年转瞬即逝，转眼就要离开学校踏入社会。在重庆理工大学的这几年，我十分快乐充实，并有满满得到收获感。首先，我十分感谢学校众多老师对我的栽培与谆谆教诲，我将时刻铭记在心。在校期间我始终用功学习、认真做人、乐观面对生活，我的人生观、价值观日趋成熟。

如今，论文写作已经接近尾声，我特别要感谢我的导师苟光磊老师，在大学四年内，苟老师给了我莫大的帮助，其对于学术的严谨态度深深地影响了我。在我的毕业设计过程中，不管是开题报告还是代码设计，甚至到论文写作都不厌其烦地指出我的问题，并教我一步一步改正，让我的论文达到更高的水平。在此，我再次向苟老师说声谢谢。

其次，我要感谢学校给我们提供的良好的学习环境、舒适的住宿条件以及大力营造的浓厚学习氛围；感谢班级各位同学、身边的朋友们给我提供的无私帮助，是这些让我在大学生活中度过了一个充实而快乐的求学生涯，学校的一室、一桌、一花、一树都将成为我生命中的永恒记忆。其中我要特别感谢我的女朋友赵艺萱，在我大学生活的方方面面给我提供了许多帮助，让我对即将结束的大学生涯充满怀念，愿执子之手，遍览世间繁华。

同时，我要向我的父母表达深深的感激之情。他们在我二十多年的人生旅途中，不断鼓励我，激励我坚持和努力，不要轻言放弃，没有他们就没有今天的我。在此，我要向他们说一声，谢谢！

最后，再次向所有关心我的亲人、老师和同学们表达我深深的谢意，祝您们健康快乐！

## 参考文献

- [1] 袁立,袁吉收,张德政.基于 DeepLab-v3+的遥感影像分类[J].激光与光电子学进展,2019,56(15):236-243.
- [2] 陈宏伟, 鄢来仪, 叶志伟. 基于烟花算法的 Otsu 多阈值图像分割法[J]. 湖北工业大学学报, 2018(1):55-58.
- [3] 屈彬, 王景熙, 郑昌琼,等. 一种基于区域生长规则的快速边缘跟踪算法[J]. 四川大学学报(工程科学版), 2002, 34(2):100-103.
- [4] 贾坤, 李强子. 农作物遥感分类特征变量选择研究现状与展望[J]. 资源科学, 2013, 35(12): 2507-2516.
- [5] 陈天华,郑司群,于峻川.采用改进 DeepLab 网络的遥感图像分割[J].测控技术, 2018, 37(11): 34-39.
- [6] 史舟, 梁宗正, 杨媛媛, 等. 农业遥感研究现状与展望[J]. 农业机械学报, 2015, 46(2): 247-260.
- [7] Liang-Chieh Chen, Yukun Zhu, George Papandreou, et al. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation[EB/OL].ArXiv abs/1802.02611(2018):n.pag.
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition[C]. 2015 IEEE transactions on pattern analysis and machine intelligence, 2015,37 (9): 1904-1916.
- [9] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation[C].2017 IEEE transactions on pattern analysis and machine intelligence,2017,39 (12): 2481-2495.
- [10]Chen L C , Papandreou G , Kokkinos I , et al. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs[C]. 2018 IEEE Transactions on Pattern Analysis and Machine Intelligence,2018,40 (4): :834-848.
- [11]Liang-Chieh Chen, George Papandreou, Florian Schroff, et al. Rethinking Atrous Convolution for Semantic Image Segmentation[EB/OL].ArXiv abs/1706.05587 (2017): n. pag.
- [12]François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions[C]. 2017 IEEE Conference on Computer Vision and Pattern Recognition, 2017,1251-1258.

- [13] Gao Huang, Yixuan Li, Geoff Pleiss, et al. Snapshot Ensembles: Train 1, get M for free[C].  
2017 international conference on learning representations, 2017:1-14.
- [14] Hinton G , Vinyals O , Dean J . Distilling the Knowledge in a Neural Network[J].  
Computer Science, 2015, 14(7):38-39.
- [15] Rafael Müller, Simon Kornblith, Geoffrey Hinton. When Does Label Smoothing Help? [C].  
33rd Conference on Neural Information Processing Systems (NeurIPS 2019), 2019, 1-10.
- [16] Yi Zhu, Karan Sapra, Fitsum A. Reda, et al. Improving Semantic Segmentation via Video  
Propagation and Label Relaxation[C]. 2019 IEEE/CVF Conference on Computer Vision  
and Pattern Recognition, 2019, 8856-8865.