| | |
|---|---|
| 队伍编号 | 382 |
| 赛道 | B |

# 探索基于 UNet 模型的遥感图像地块分割与提取

## 摘要

针对问题一，基于初赛的 UNet 模型，我们首先从图像的概率阈值、卷积-池化层数和学习率等参数对遥感图像提取耕地边界进行了优化探索，但这些方法训练效果不佳。随后，我们提出了能够有效增加卷积时感受野大小的 Dilated-UNet（D-UNet）模型，其中将最底层的 4 个普通卷积调整为 4 个空洞率分别为 3、5、7、9 的空洞卷积；以及提出基于初赛 UNet 模型对 batch size 和 dropout 两个参数进行改进和优化的 Advanced-UNet 模型，其中 batch size 从 2 改为 8，dropout 参数从 0.5 改为 0.2。Dilated-UNet 模型训练结果表明，除了 train-loss 下降 75%以外，val-loss、val-acc 和 Kappa（D-UNet）都较劣于初赛模型，可知 D-UNet 模型与初赛结果相比效果欠佳，但其对优化提取耕地边界方面具有一定的优化作用，这为以后深入探索空洞卷积效果起到了积极的作用，拓展了思路方向。而 Advanced UNet 的模型结果表明，train-loss、val-loss 和 val-acc 分别下降了 18.6%、4.1%、0.6%，Kappa（Advanced-UNet）=0.9587 比原来的 Kappa（UNet）=0.9274 提高了 3.4%，并且通过 Test3、Test4 预测结果说明 Advanced-UNet 模型相比于初赛的 UNet 模型对提取耕地边界、分割精度上有所优化和提高，且得到的效果更佳。最后我们运用 Advanced-UNet 模型预测出 Test3、Test4 图像耕地比例分别为 0.635，0.5642。

针对问题二，基于问题一的实践探索我们选取了阈值、卷积-池化层数、学习率、卷积空洞率作为系统关键参数并予以简单描述。对于系统关键参数对识别精度的影响，我们采用了 Kappa 系数进行定量评估。即把 Advanced-UNet 模型的上述 4 个系统关键参数值作为初始参数值，在初始参数值的基础上对关键参数分别进行微调，采用控制变量法进行实验，将初始参数模型和调整参数模型得到的 Kappa 系数进行横向比较。结果表明，调整阈值、卷积-池化层数、学习率和空洞卷积率后模型对应的 Kappa 系数和初始的相比分别下降了 8.06%、10.86%、9.97%、7.56%。结果进一步显示 Advanced-UNet 模型的分割效果较好。

以上两问的结果说明 Advanced-UNet 模型比初赛使用的 U-Net 模型在图像提取耕地的边界上进行了优化，且 Advanced-UNet 模型训练得到的关键参数对所给图像的耕地提取上较为适用。

**关键词：**遥感图像；二值化；UNet；Advanced-UNet；D-UNet；空洞卷积；Kappa 系数

# 目录

# 1 问题重述

## 1.1 问题背景

从古至今，我们一直利用土地满足需要。当下我国的耕地面积仅占国土的 16%左右[1]，耕地资源不多，其作为农业可持续发展的关键，耕地的数量和耕地质量的保证应是有关部门深思的问题。因此，获得准确的耕地分布尤为重要。现如今高精度的耕地信息提取主要还是依靠人工解译，耗费大量人力、财力且效率较低。由于利用卫星遥感影像识别可提取耕地并对耕地进行遥感制图，因此，遥感图像的耕地识别算法研究将对耕地遥感制图提供重要帮助，使耕地识别效率大大增高。

## 1.2 问题提出

现已提供 8 幅图像和相应耕地标签，用于模型训练和测试。标签图中白色（值为 1）代表的是耕地类，黑色（值为 0）代表的是背景类。并提供 2 幅图像（Test3、Test4）作为测试实例。现要我们解决下面 2 个问题。

问题 1：利用模型对附件数据进行处理，从给定的 2 幅测试图像(Test3.tif、Test4.tif)中提取出耕地，计算耕地在图像中所占比例,并将标签图分别上传到竞赛平台中；并说明相对于初赛模型，在优化提取耕地边界方面做出了哪些改进。

问题 2：描述识别系统中的关键参数，并定量评估其对识别精度的影响。

# 2 问题分析

## 2.1 问题一的分析

通过查阅文献资料和初赛中使用的 UNet 模型的求解过程，我们发现概率阈值、卷积-池化层数、学习率这三个参数对模型对图像耕地的提取有着密切的关系。我们可以从这三个参数的改进和优化入手对初始模型进行优化。同时我们关注到在 UNet 模型上采样的过程中，由于池化会导致一些有用信息的损失，经查阅文献资料我们采用了能够在不增加网络参数和不缩小图像尺寸的前提下有效地增加卷积感受野大小的空洞卷积 UNet 模型（Dilated-UNet），从而避免因池化带来的信息损失。基于初赛和上述的探索过程，我们基于 UNet 模型又提出了一种经改进后的模型 Advanced-UNet，该模型的网络架构与 UNet 相同，但对 batch size 和 dropout 这两个参数进行改进。对上述几种思路付诸于实践，并对各调整模型的精度进行分析，从而确定较优化的最终模型。最后算耕地比例时，预测出两张图像的标签图中属于耕地的像素点数（即 1 值）占总像素点数的比例即为所求。本问题的探索思路如下图 2-1 所示。

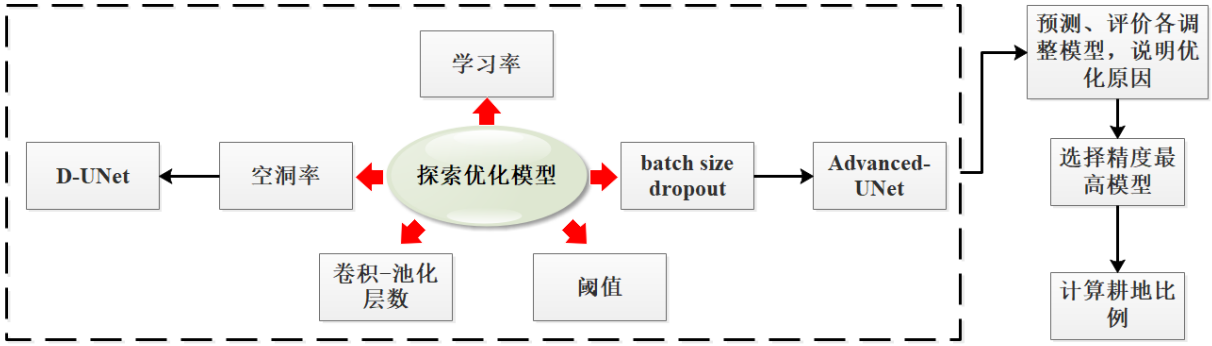



图 2-1 问题一的探索思路

### 2.2 问题二的分析

基于问题一，我们选取概率阈值、卷积-池化层数、学习率和卷积空洞率作为系统的关键参数。以问题一中 Advanced-UNet 模型的上述 4 个系统关键参数值作为初始参数值，在初始参数值的基础上对关键参数分别进行微调，采用控制变量法进行实验，用 Kappa 系数作为模型精度进行调整前后模型效果的横向比较并定量评估，即可得知关键参数的调整对识别精度的影响。本问题的分析思路如下图 2-2 所示。

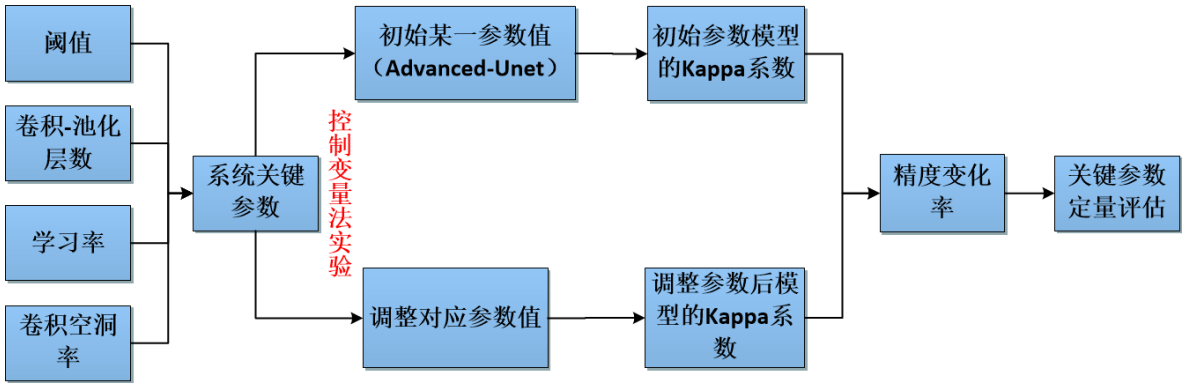



图 2-2 问题二分析思路

## 3 符号说明

| 符号 | 说明 |
|---|---|
| $N$ | 0-1 值化后的标签图像素点数 |
| $N_i$ | 0-1 值化后第$i$张标签图像素矩阵中“1”值个数 |
| $A_i$ | 第$i$张图片中的耕地占比 |

注：这里只列出论文各部分通用符号，个别单独使用的符号在首次引用时会进行说明。

## 4 模型建立

### 4.1 全卷积神经网络与 U-Net 模型

本部分简要介绍全卷积神经网络的架构及初赛中使用的 U-Net 模型。

#### 4.1.1 全卷积神经网络

对于分割问题，实际上是对每个像素点进行分类。而传统卷积神经网络无法对每个

像素点分类，需要在待分类像素周围指定图像块才能输入网络，这种重复利用像素的方法会导致训练速度很慢、分类效果变差[2]。

2014 年，为了改进这些问题进而提出了全卷积网络（FCN），FCN 即是用卷积层代替 CNN 中的全连接层，可解决根本性问题：像素点分类。FCN 可以接受任意尺寸的输入图像，先下采样提取图像高维信息，此时图像变小，然后对下采样最后一个卷积层的输出图像进行上采样，获取高层语义特征，使它恢复到与输入图像相同的尺寸。上采样是 FCN 的一大特点，也可称为反卷积（deconvolution），反卷积与卷积类似，都是相乘相加运算，不过卷积是多对一，而反卷积是一对多[3]。接着在上采样的特征图上进行逐像素分类，最后逐个像素计算 softmax 分类的损失并求和，得到交叉熵损失函数。通过损失函数梯度下降不断对模型参数优化至训练结束。

（1）上采样层（Upsamping）

下采样结束后图像尺寸变小，使用上采样层可将图像恢复成原来大小，特征细节也会更丰富。本文介绍该层中上采样的方法，此方法未使用最大池化时的位置信息，而是直接将内容复制来扩充特征图，示意图如图 4-1 所示。

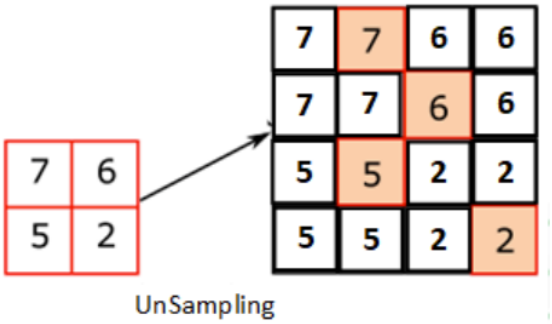



图 4-1 上采样过程

（2）级联组合层（Concatenate）

本文在上采样层之后加入级联组合层，将上采样输出的特征图与对称路径相应卷积层输出特征图进行级联组合，这两个图像的尺寸需一致，目的是为了融合图像的浅层和深层信息，使最终图像更清晰、准确性更高。

（3）Batch Normalization 层[2]

网络的每一层需要输入数据，因为各层参数不同，输出数据会有不同分布状态。批标准化法（Batch Normalization）可为每一层输入数据设置初始化方式、统一标准；整体而言，批标准化使得数据分布相对均匀且变化敏感，使网络提高训练效率。

（4）Dropout 层

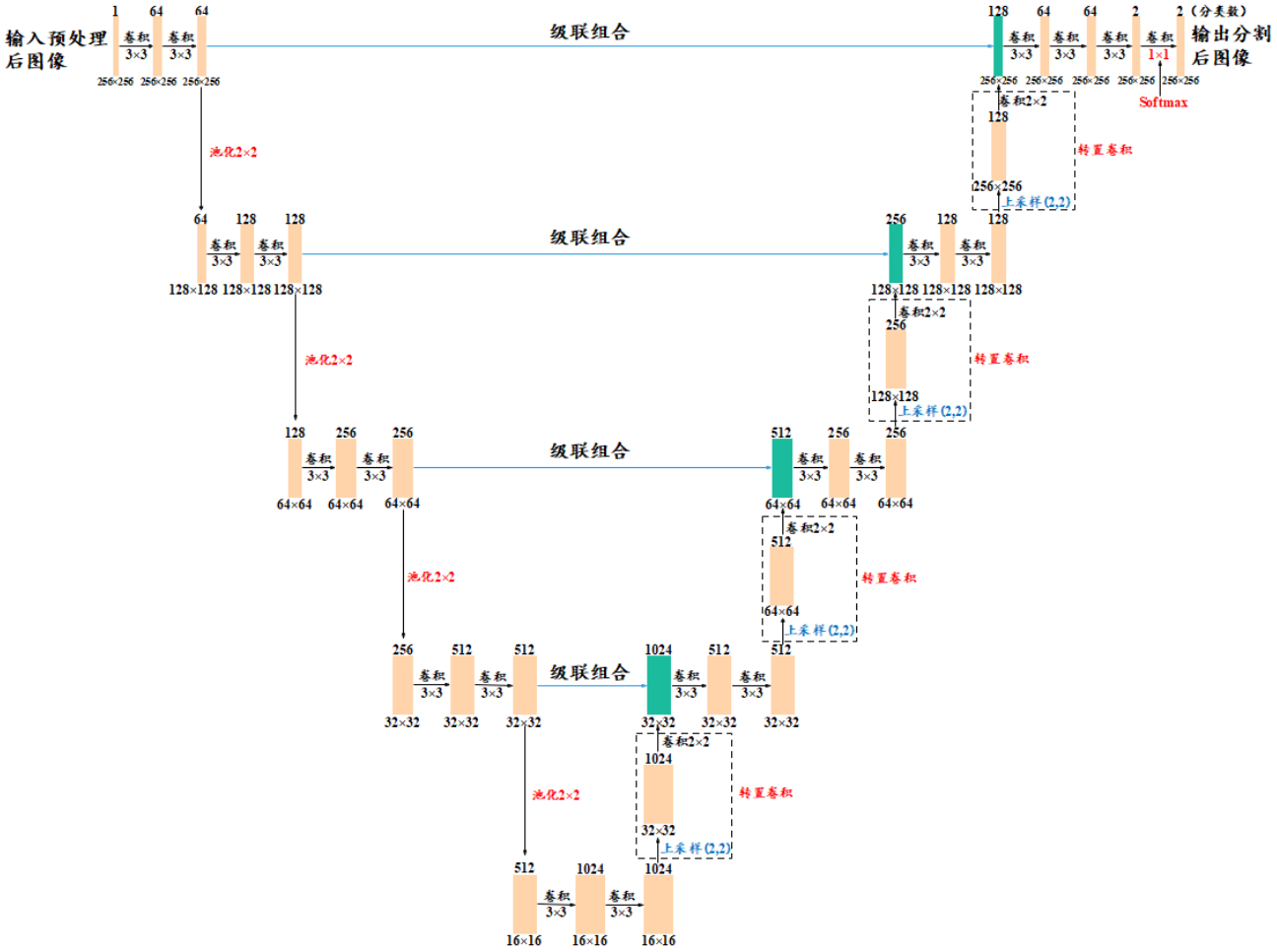
该层不是框架中必需的。但由于过拟合是深度模型中普遍存在的问题，一般都是通过融合训练过的多个模型加以解决，但会产生耗时很长的弊端。基于此，增加 Dropout

层灭活部分神经元，当网络训练时，这些灭活的神经元不会被丢弃，只是暂时不通过此神经元传递信息，即可使模型变简单，计算量变少，防止过拟合。

### 4.1.2 初赛中使用的 UNet 模型

查阅相关资料[2]可知，UNet 网络是对 FCN 的优化，最先应用于医学图像分割，对于二分类、背景单一的图像具有良好的分割效果。其次，UNet 网络针对训练数据较少的分割能力依旧很强，基于此我们最初选择了 UNet 网络。为了降低网络层间的复杂度，我们增加了 Dropout 层。本模型中我们在第 4、5 个卷积层后添加了概率为 0.5 的 Dropout 层，即暂时灭掉一半的神经元防止过拟合。

基于本问题实际，对于遥感图像地物类型丰富、光谱范围广等特点，我们相应地在基础 UNet 模型[4]的基础上增加模型的卷积次数以提取更丰富、深层次的特征，得到了初赛的模型。初赛构建的 UNet 模型架构图如图 4-2 所示。




注：①图中每层输出宽度仅表示输出图像的尺寸；

②图中转置卷积后得到的输出图与对称路径下采样中得到特征图的通道数相同，二者级联后通道数加倍，得到特征图（绿色矩形）

图 4-2 初赛采用的 UNet 模型架构图

由上图，初赛采用的 UNet 网络也为编码—解码结构，所有卷积过程的 padding=same，可使输入输出图像尺寸相同，均为 256×256。左半部分为下采样过程，由 5 组卷积与 4 个最大池化层组成，卷积核大小为 3×3，各组卷积的卷积核数量分别为 64、128、256、512、1024，且卷积层输出采用 Relu 激活函数。最大池化层大小为 2×2，可获取输入图像浅层局部特征。右半部分为上采样过程，虚框部分先进行一次上采样再进行一次卷积，此过程称为转置卷积，得到的特征图与对称路径下采样中得到特征图级联组合，此举可融合上采样输出的深层抽象特征和下采样局部特征，以助于恢复一些细节信息。在输出分割图像前采用 softmax 激活函数，转换为两张概率图。本问题为二分类问题，因此这两张概率图对应元素之和为 1。

## 4.2 模型探索

初赛模型总体在分割效果上较令人满意，训练停止时 train-loss、val-loss 分别为 0.10211 和 0.1760，Kappa 系数为 0.9274。但不难从预测结果中看出来一些问题，下图 4-3 是使用初赛模型预测 Test2 的预览图。

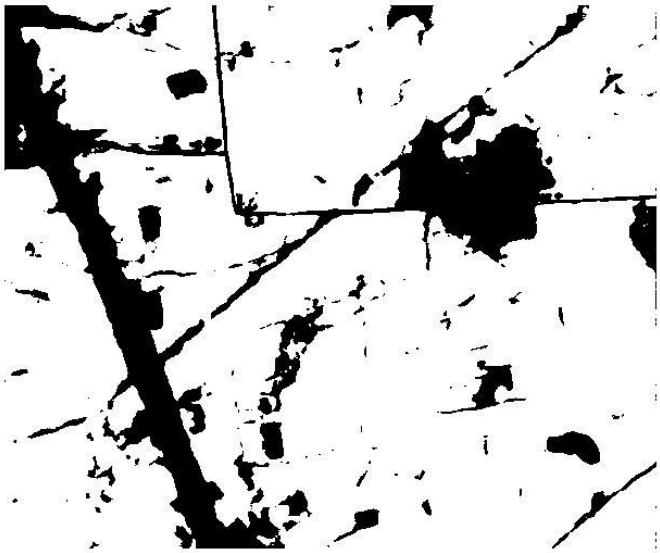


图 4-3 初赛模型预测 Test 2 的预览图

上图可以看出，存在诸多小孔及斑点等错分的情况，导致耕地边界比较模糊。经过对概率阈值、卷积-池化层数、学习率等参数的调整进行训练取得的效果均不佳后拟采用下述两种方法进行优化探索。基于问题实际，同时已知遥感图像地物类型丰富、光谱范围广等特点，我们通过查阅资料、不断探索与验证，得到对提取耕地边界、分割效果具有一定优化作用的网络模型，下述两个模型均对以后的研究起到一定积极的作用，故下面

仅详述具有一定成效的探索过程，而精度远小于初赛模型的探索过程不再赘述。

4.2.1 Dilated U-Net

在探索过程中，我们发现了一种基于空洞 UNet（Dilated UNet，D-UNet）的图像分割方法。

**4.2.1.1 空洞卷积**

上述的 UNet 模型中，卷积层后面是池化层，卷积层提取图像特征后通过池化层对其进行降维，可有效降低网络参数的规模以及获得更大的感受野[5]。如此操作会使得特征图尺寸越来越小，图 4-4 显示网络最底层特征图尺寸为 16×16。使用 UNet 模型时需要让输入图像与输出图像的尺寸相同，因此需要上采样还原图像尺寸，而特征图缩小再放大的过程中由于池化会造成信息的损失，因此我们采用空洞卷积来解决该问题。空洞卷积可在不增加网络参数和不缩小图像尺寸的前提下有效增加卷积得感受野[5]。

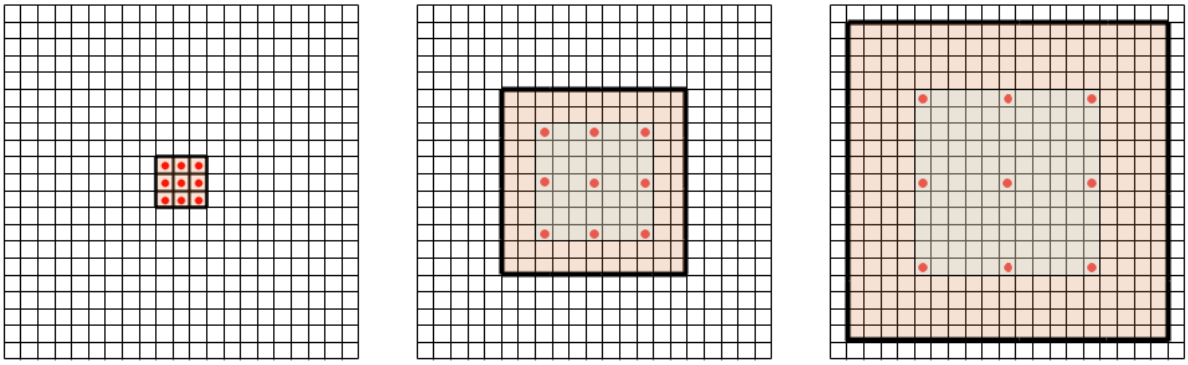
下面详细介绍空洞卷积的原理。空洞卷积在卷积时会在卷积核元素之间塞入空格，同时引入一个新的参数空洞率（dilation），卷积核元素塞入空格的数目为空洞率减 1，设原来卷积核的大小为$k$，则空洞率为$r$的卷积核实际大小为：

$$n = (k-1)\cdot(r-1)+k \tag{1}$$

在 D-UNet 模型中，我们设置空洞卷积的 padding=same，由此可以计算出卷积时空洞卷积的感受野大小为：

$$m = 2(r-1)\cdot(k-1)+k \tag{2}$$

下图 4-11a 为 3×3 的普通卷积，实际上即为空洞率为 1 的卷积。图 4-11b 是空洞率为 3 的空洞卷积，由（1）式可知其实际卷积核大小为 7×7 见浅绿色部分，由（2）式可知感受野为 11×11，相当于与 11×11 的感受野相同（橙色区域），在图中仅红色的元素参与卷积运算，其余元素不参与；图 4-11c 是空洞率为 5 的空洞卷积，由（1）式可知其实际卷积核大小为 11×11 见浅绿色部分，由（2）式可知感受野为 19×19，相当于与 19×19 的感受野相同（橙色区域），同理在图中仅红色的元素参与卷积运算。显然，空洞卷积可以在不增加卷积网络参数的情况下增大感受野，避免池化带来的信息损失[5]。

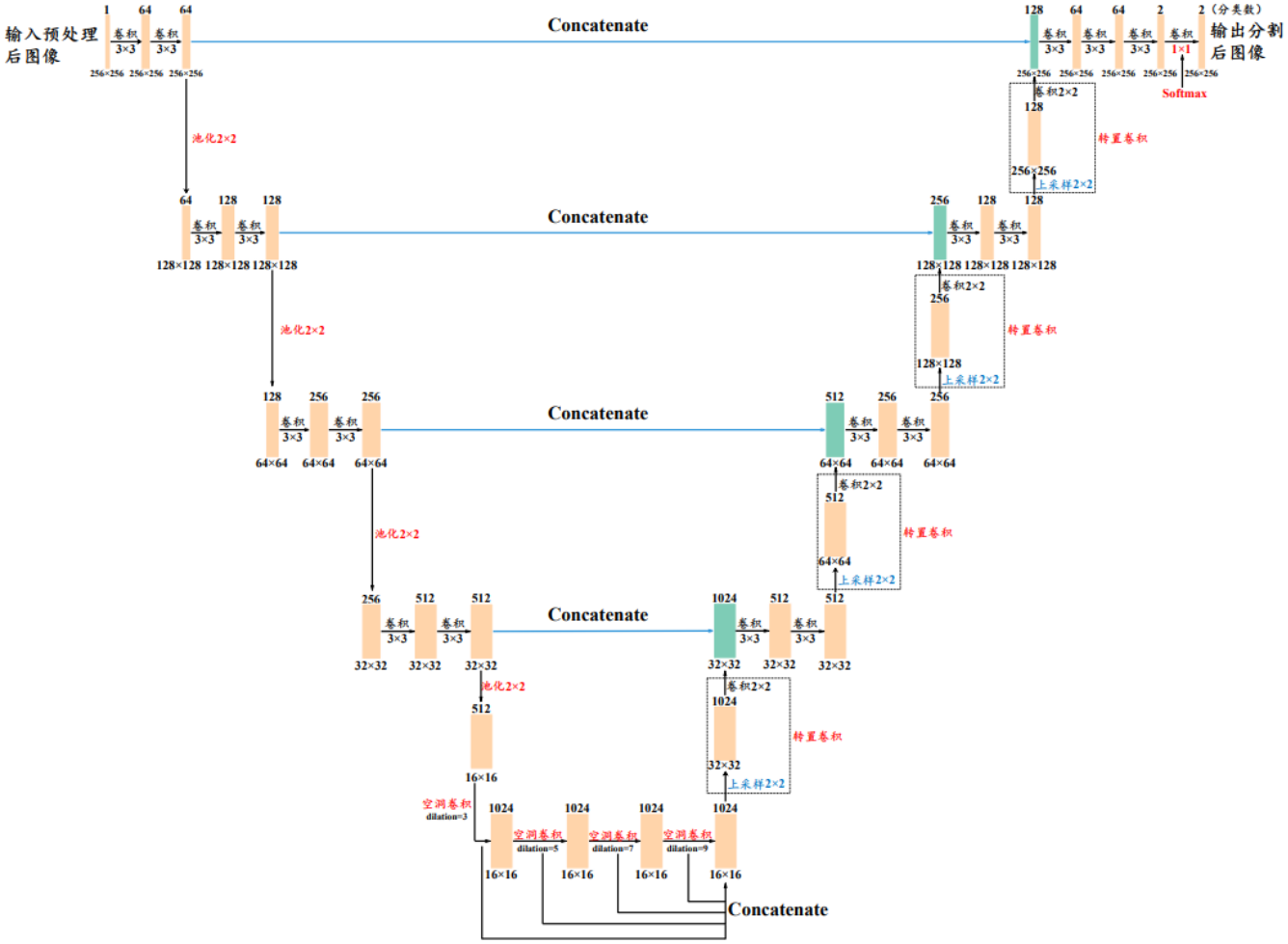


a 空洞率为 1 的普通卷积　　b 空洞率为 3 的空洞卷积　　c 空洞率为 5 的空洞卷积

图 4-4 空洞卷积、实际卷积核与感受野

#### 4.2.1.2 基于初赛模型调整的 D-Unet 模型

我们对初赛 UNet 模型进行了一些调整，从而得到了一种能增大感受野的 D-Unet 模型。D-Unet 模型架构如图 4-5 所示。




注：①图中每层输出宽度仅表示输出图像的尺寸；

②图中转置卷积后得到的输出图与对称路径下采样中得到特征图的通道数相同，二者级联后通道数加倍，得到特征图（绿色矩形）

图 4-5 调整的 D-Unet 模型架构图

相对于初赛模型，我们对网络最深卷积层做了一系列调整。网络最底层由 3 个普通卷积构成的卷积层改成 4 个空洞卷积构成的空洞卷积层，空洞率分别为 3、5、7、9，以此对下采样的特征图进行不同尺度的信息提取，同时将各提取的结果通过 Concatenate 方式进行融合得到更加丰富的特征，最后再将结果上采样输出。其余的架构与初赛采用的 UNet 模型一致。

4.2.2 Advanced-UNet

在不断调整相关参数的探索过程中，我们基于初赛 U-Net 模型提出一种改进的模型（Advanced U-Net）。网络架构同初赛模型（见 4-2），仅对少量参数进行改进优化。

**（1）改变 batch size**

原来的模型我们设置该参数为 2。为了提高运算速度及精度，经过反复实验我们在改进的 U-Net 模型中将 batch size 调整为 8。

**（2）改变 dropout**

初赛的模型我们设置两个 dropout 层，参数为 0.5，分别放在第 4、5 卷积层结束的位置。为避免抛弃较多的数据，经过反复实验，在改进的 UNet 模型中我们同样设置了两个 dropout 层，分别放在第 4、5 卷积层结束的位置，而参数调整为 0.2。

### 4.3 框架选取

本文选用 Keras 框架，其使用 python 编写，封装有 RNN、CNN 等算法，具有代码较简单、易操作等优点，适合初学者使用。

## 5 问题一实验设置与结果

### 5.1 实验数据处理与环境

实验数据为初赛中给出的 8 张 data1-8 图片及对应标签，以及 2 张 test3-4 图片。实验的硬件环境为：GPU GeForce GTX1080Ti、显存 10G、软件 Python3.6 和 Tensorflow2.4.3。数据预处理可分为如下操作：

Step1.**图像裁剪与数据增强**

由于供训练的样本仅有 8 张（不包括标签），因此我们先分别将原始图像和对应的标签图像切割成尺寸为 256×256 的图片，再分别将切割后的原始图像和标签图像进行水平、垂直翻转和对角镜像的数据增强处理，以提高模型的泛化能力。代码见附录 2。

Step2.**数据归一化与编码**

数据增强后的图片不直接输入到网络中，先对原始图像和标签进行归一化处理，映射到

统一标准，再对标签进行 one-hot 编码。而且在这个环节中，我们定义了训练数据生成器用来在模型训练中产生训练集与验证集，比例为 4：1，以及测试数据生成器用来对测试图片进行预处理。代码见附录 3。

## 5.2 实验流程

数据预处理后接下来将样本数据分为 80%训练集、20%验证集训练 D-UNet 模型、Advanced U-Net 模型，后者将关键参数设置为 dropout=0.2，learning rate=1e-4，batch size=8，epoch=40，训练结束后将验证集样本进行预测得到三个精度指标：交并比（Iou）、召回率（Recall）、Kappa 系数并进行精度评估，最后分别用调整后的两个模型预测 Test3、Test4 并做结果分析。实验流程如下图 4-6 所示。

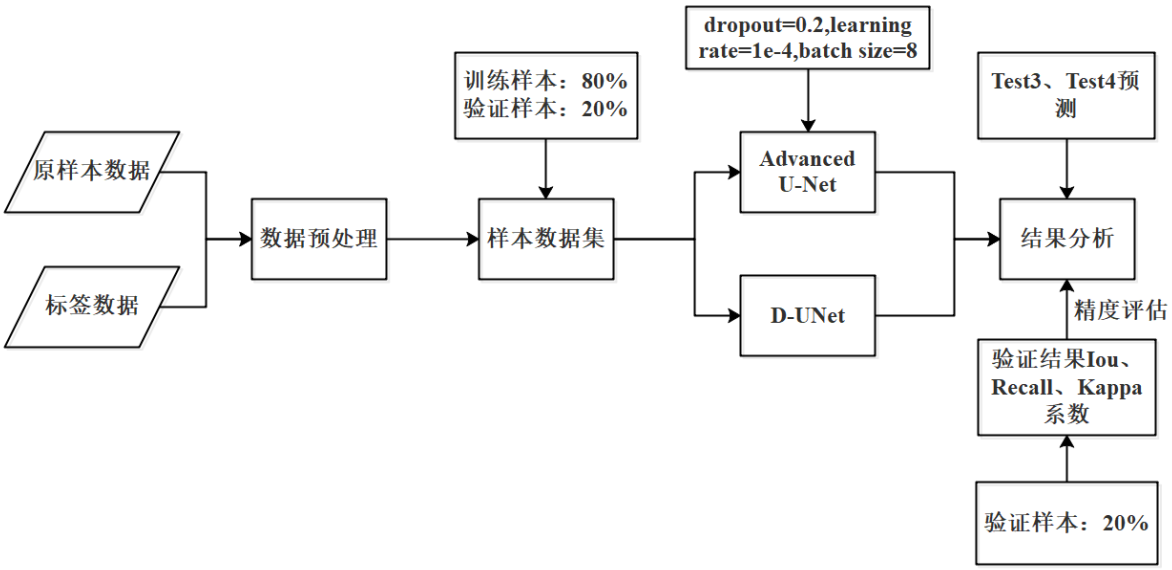



图 4-6 实验流程

## 5.3 实验结果分析

### 5.3.1 模型训练结果

#### 5.3.1.1 D-UNet

训练过程中每批生成 2 个样本用来训练。每迭代完一次若 train-loss 减小则保存更新模型并更改权重，迭代次数为 40 次，当 train-loss 连续 3 代未下降终止训练。模型训练过程耗时约 7 小时，最终迭代 40 代，此时 train-loss 为 0.02553，val-loss 为 0.2210，val-acc 为 0.9425。模型的 acc 和 loss 曲线如下图 4-7 所示。

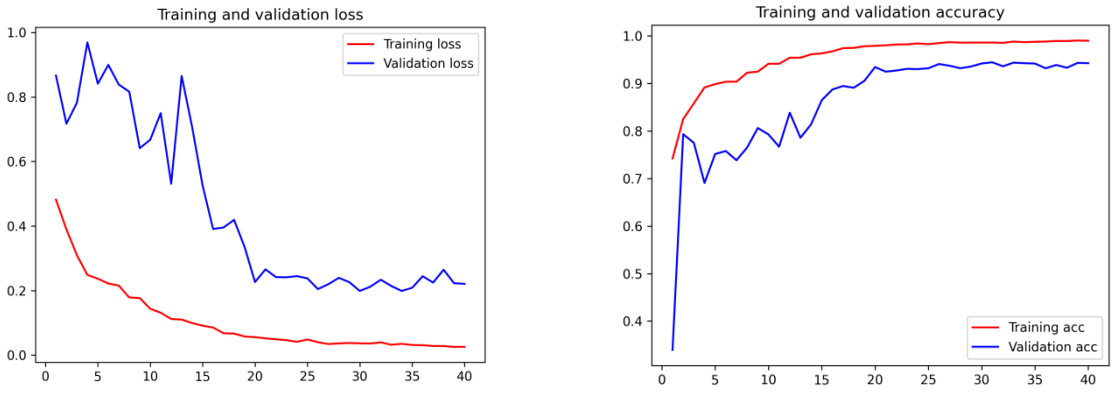



图 4-7 D-UNet 模型 acc 和 loss 曲线图

初赛使用的模型训练结束后 train-loss 为 0.10211，val-loss 为 0.1760，val-acc 为 0.9746。与初赛相比，调整后的 D-UNet 模型的 train-loss 下降 75%，val-loss 上升 25.6%，val-acc 下降 3.3%。对比并结合上图 4-7 中 val-acc 和 val-loss 曲线走势，可知模型训练效果与初赛模型近似。

#### 5.3.1.2 Advanced-UNet

训练过程中每批生成 8 个样本用来训练。每迭代完一次若 train-loss 减小则保存更新模型并更改权重，迭代次数为 100 次，当 train-loss 连续 3 代未下降终止训练。模型训练过程耗时约 5 小时，最终迭代 98 代，此时训练结果与初赛模型极度近似，train-loss 为 0.08314，val-loss 为 0.1687，val-acc 为 0.9801。模型的 acc 和 loss 曲线如下图 4-8 所示。

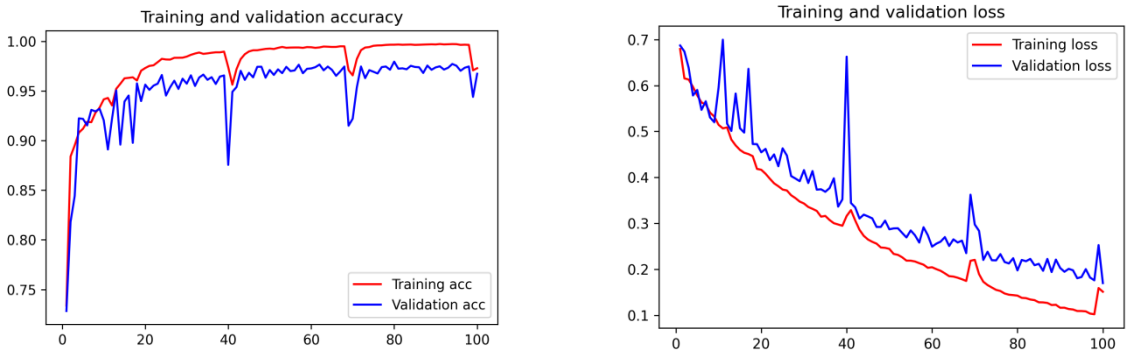



图 4-8 Advanced-UNet 模型 acc 和 loss 曲线图

调整后的 Advanced-UNet 模型的 train-loss 下降 18.6%，val-loss 下降 4.1%，val-acc 上升 0.6%。对比并结合上图 4-15 中 val-acc 和 val-loss 曲线走势，可知模型训练效果优于初赛模型，同时也侧面说明初赛采取的模型及设置参数比较合理准确。Advanced-UNet 模型定义代码见附录 4，模型训练代码见附录 5。

### 5.3.2 验证集精度指标

本文采用的精度指标是交并比（Iou）、召回率（Recall）、Kappa 系数。Iou 为语义分

割中常用的衡量标准，表示预测值与真实值的交集与预测值与真实值之间的并集的比值[5]。计算公式为：

$$Iou = \frac{TP}{FP + TP + FN} \tag{3}$$

Recall 为被正确预测为耕地占总耕地的比例[5]。计算公式为：

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

上两式中，TP 为网络预测结果为正样本，实际也是正样本的特征数；FP 为网络预测为正样本，而实际是负样本的特征数；FN 为预测为负样本，而实际为正样本的特征数。对验证集样本预测结果进行计算可以得出 Iou（D-UNet）=0.9471，Recall（D-UNet）=0.9660，Iou（Advanced-UNet）=0.9802，Recall（Advanced-UNet）=0.9878。另外，我们查阅遥感图像分割的相关评价指标[6]，学术界比较有代表性的为 Kappa 系数，因此我们通过计算 Kappa 系数来检验模型的准确性。

Kappa 系数的计算需要依赖混淆矩阵。混淆矩阵也称误差矩阵，是表示精度评价的一种标准格式。由于本文解决二分类问题，故该矩阵表达为 2×2。每行代表该类正确和误分到其他类的像素数。我们将验证组预测与真实数据输入 python 软件中得到混淆矩阵。之后计算 Kappa 系数。Kappa 系数取值为（-1，1），系数大于 0.8 意味着好的分类[6]。假设混淆矩阵大小为$a \times a$，$M$ 为像素总数，则 Kappa 系数表达式为：

$$Kappa = \frac{M\sum x_{aa} - \sum x_{a\sum a} \cdot x_{a\sum a}}{M^2 - \sum x_{a\sum a} \cdot x_{a\sum a}} \tag{5}$$

将混淆矩阵表中相应元素代入(5)式中得到 Kappa（D-UNet）＝0.886，Kappa（Advanced-UNet）=0.9587。对于初赛模型，Kappa（UNet）=0.9274。由于 Kappa（Advanced-UNet）=0.9587 更接近于 1，则可认为我们优化改进的 Advanced-UNet 模型精度最高。

### 5.3.3 耕地所占比例

通过问题一的分析可知，将 Test3、Test4 放入模型进行预测后得到预测标签图，设预测标签图中耕地所占的比例$A_i$计算公式如下：

$$A_i = \frac{N_i}{N} \tag{6}$$

根据 5.3.2 计算出的精度，我们采用精度最高的 Advanced-UNet 模型计算耕地比例。我

们运用 Matlab 软件依次计算这两幅图的耕地比例，得到 Test3、Test4 耕地比例分别为 0.635，0.5642。代码见附录 1。

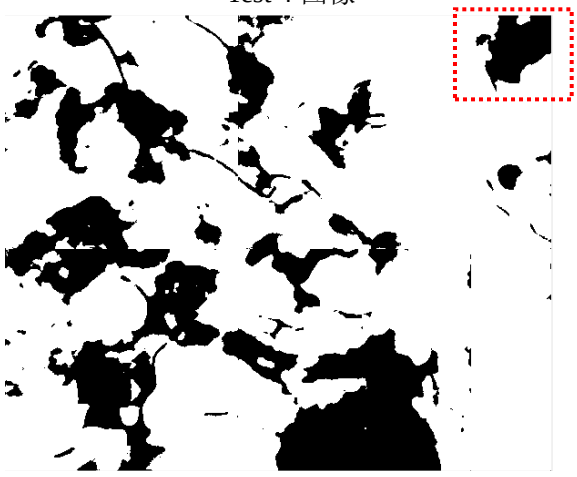
**（4）预测结果与分析**

为了直观展示上述两种调整方法预测的效果，我们对 Test3、Test4 分别运用两个模型进行预测。调用测试数据生成器对测试数据预处理，并结合忽略边缘预测法[7]，先裁剪后拼接、并对标签可视化得到测试图像的最终结果如下组图 4-16。代码见附录 6。

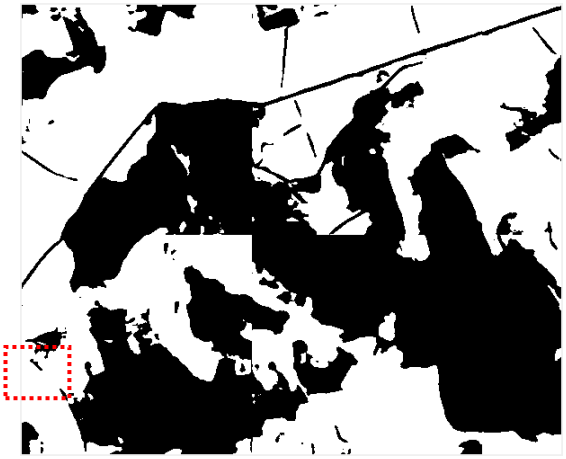
Test 3 图像

Test 4 图像

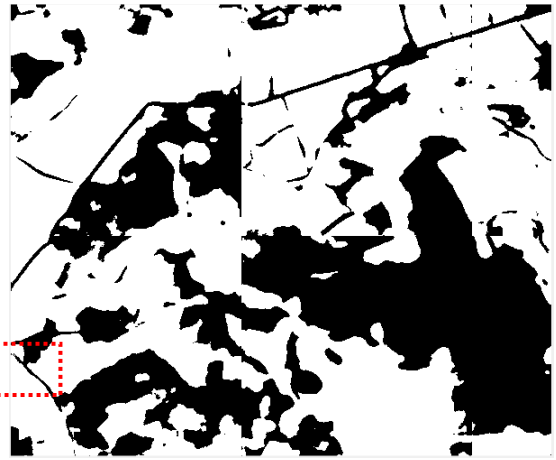
Test3 Advanced-UNet 预测



Test3 D-UNet 预测



Test3 Advanced-UNet 预测



Test4 D-UNet 预测

图 4-16 Test3、Test4 预测标签组图

基于提取耕地边界、分割精度来说，我们基于初赛模型不断尝试调整各种参数得到的 Advanced-UNet 模型在一定程度上进行了优化。可以看到，D-UNet 模型虽然理论上具有较大的优势性，但实际效果远不如 Advanced-UNet 模型，我们猜测这可能与空洞卷积感受野不连续有关。但值得注意的是，D-UNet 模型对耕地边界提取的效果有所提升（如上图红色框区域等），说明我们努力的方向是正确的，在参数设置上需要继续实验探索，以得到分割效果更好的模型。而对于 Advanced-UNet 模型而言，虽然精度和边界提取清晰度均提升了，但仍然存在孔洞等错分现象，需要在未来继续加以优化提升。

# 6 问题二的求解

## 6.1 系统关键参数

### 6.1.1 阈值

在网络最后需要通过 softmax 函数输出概率分布图，而交叉熵损失目标函数$y_i$与概率分布图中每一个像素点对应是否为耕地的概率$p_i$有关，因此确定一个合适的概率阈值对分割结果尤为重要。

### 6.1.2 卷积-池化层数

卷积、池化层可以提取输入图像的特征，从低维逐渐至高维。当网络层数越深时，所“学习”到的特征就会越多，上采样时通过 Concatenate 方式进行高、低维特征融合，训练效果就会更佳。但随着网络层数越深，运行时间成本就会越高。

### 6.1.3 学习率

学习率越大，输出误差对参数的影响就越大，参数更新的就越快，但同时受到异常数据的影响也就越大，很容易发散[8]。而且学习率不是固定不变的，它是随着训练次数增加进行衰减变化的值。初始学习率设置会对分割效果产生较大的影响。

### 6.1.4 卷积空洞率

一般来说，卷积都是指的普通卷积，即空洞率为 1 的卷积。空洞卷积可以增大卷积的感受野，学习特征的能力更强，在不需要改变图像尺寸的条件下可有效避免池化带来的信息缺失。但空洞卷积的缺点是感受野不连续，它依赖于网络的设计，因此空洞率的设计是空洞卷积网络模型中需要研究的首要问题。

## 6.2 精度影响因子评估

为使精度评估分割效果更为科学、严谨，我们仅采用遥感学界普遍认同的 Kappa 系数作为模型的精度。同时我们基于上述提出的精度最高的 Advanced-UNet 模型，采用控制变量法，对 4 个因子进行实验。仅改变上述 4 个因子中其中一个的值，分别定量计算

精度。由于时间有限，我们仅考虑了上述四个因素进行调整。计算精度采用的数据集均为验证数据生成器中 20%的验证集。计算结果如下表 6-1 所示。

表 6-1 系统参数对 Kappa 系数的定量影响

| 阈值 | | 卷积-池化层数 | | 学习率 | | 卷积空洞率 | |
|---|---|---|---|---|---|---|---|
| 模型 | | 模型 | | 模型 | | 模型 | |
| Ad-UNet | 调参后 | Ad-UNet | 调参后 | Ad-UNet | 调参后 | Ad-UNet | 调参后 |
| 0.5 | 0.6 | 4 | 3 | 1e-4 | 1e-5 | 1 1 1 1 | 3 5 7 9 |
| Kappa 系数 | | | | | | | |
| 0.9587 | 0.8814 | 0.9587 | 0.8546 | 0.9587 | 0.8631 | 0.9587 | 0.8862 |
| 调整因子数值后相比较优模型精度变化率 | | | | | | | |
| \ | -8.06% | \ | -10.86% | \ | -9.97% | \ | -7.56% |

注：Ad-UNet 代表 Advanced-UNet 模型

根据上表，在一定范围内调整后的关键系统参数对精度均产生较大的负影响，即效果不如初始设定值。**对于阈值而言：**阈值的设定对不同分割任务均会有影响，而在训练过程中即应寻找最佳分割阈值，这就需要不断地尝试、试错；**对于卷积-池化层数而言：**显然层数更小学习到的特征会越少，精度自然会降低；**对于学习率而言：**同阈值一样，需要找到最适合本分割任务的学习步长。在实际研究中，对模型学习率数值探索是必需的；**对于卷积空洞率而言：**通常来说，空洞卷积感受野比普通卷积大，理论上精度应更高。但实际显示并非如此，这可能与空洞卷积对不同分割任务的适应性有关，也可能与空洞率有关，这需要大量的实验验证。

# 7 模型的评估与推广

## 7.1 模型的优点

本文基于初赛提出的 UNet 模型，不断探索、优化得到了较好的 Advanced-UNet 模型。另外拓展了思路，探索出空洞卷积的强大性，为之后的进一步探索奠定坚实的基础。我们通过验证集验证了 Advanced-UNet 模型的可行性，并计算出评价遥感图像分类好坏的 kappa 系数为 0.9587，以及实际预测效果综合分析可知该模型分类效果较好。在改进的过程中，我们调整了 Dropout 层、batch_size 的参数，优化了分割效果。为了更好训练模型，我们将原始图像和标签均做切割、数据增强处理，并以 4：1 分为训练集和验证集，极大提高模型的收敛速度。

### 7.2 模型的缺点

本文方法的分类结果虽较接近真实情况，但仍有一些碎图斑和孔洞，以至于测试图切割预测后拼接得到的结果存在较多碎图斑等错分类的现象。对于 D-UNet 模型，空洞卷积在理论上虽具有可行性，实际探索中也确实得出提取边界效果较好的结论，但整体分割精度较低，需要不断调整空洞率等继续实验探索。

### 7.3 模型的改进

我们针对碎图斑等错分类现象的产生，下面提出几种改进方法：

（1）**图像后处理**：对于孔洞，采取孔洞融合的方式改进错分类现象。或采取 CRFs 条件随机场处理进行改进。我们的构想是采用带有 CRFs 的 D-UNet 模型进行训练。

（2）**模型融合**：即训练出几个模型随后按照某种方法融合集成得到最终模型。我们构想将 Advanced-UNet 模型、D-UNet 模型分别采取不同参数训练和预测，最终得到各自的预测图。然后采取“少数服从多数”的思路，对每个模型的各预测图的每个像素点的类别分别进行预测，属于某类频数最多的类别即为该像素点的类别。

（3）**提高数据样本数量**：在数据增强操作中，我们仅对原始数据进行 90 度、270 度旋转、对角与镜像旋转。我们构想采用更多数据增强操作，如增加对比度、图像加噪等以扩充数据量。

### 7.4 模型的推广

本问题目标为遥感图像二分类，亦可将本模型运用于遥感图像多分类任务中。

## 8 参考文献


[1] 王说情感，世界各国耕地占国土比例，https://www.sohu.com/a/405531113_120672689，2021 年 1 月 8 日.

[2] 彭晓迪，基于改进 UNet 网络的高分二号遥感影像分类研究.北京：中国地质大学，2020.

[3] qq-34672597，天池语义分割 task03，https://blog.csdn.net/qq_34672597/article/details/114156281，2021 年 3 月 12 日.

[4] Ronneberger O，Fischer P，Brox T，U-Net: Convolutional Networks for Biomedic-al I-mage Segmentation//International Conference on Medical Image Computing an-d Compu-ter-Assisted Intervention. Springer，Cham，2015.

[5] 吕道双，林娜，冯丽蓉，张小青，基于改进型 U-Net 网络的高分辨率遥感影像建筑物提取，地理空间信息，2021，19（01）：007.

[6] 苏健民，杨岚心，景维鹏，基于 U-Net 的高分辨率遥感图像语义分割方法，计算机工程与应用，2019，55(07)：207-213.

[7] Wang Z，Zhou Y，Wang S，Wang F，Xu Z，House building extraction from high re-solution remote sensing image based on IEU-Net. *J. Remote Sens.* 2021, in press.

[8] 追蜗牛的 coder，学习率（learning rate）的选择，https://blog.csdn.net/jinxiaonian11/article/details/83105439，2021 年 3 月 13 日.

# 9 附录

**附录 1**

介绍：依次计算 test3、test4 两幅图耕地比例的 Matlab 代码

```
clc;
clear;
p=[]
for i = 3:4
   path = ['C:\Desktop\B382附件\Result\Test',num2str(i),'.tif']
   x=imread(path);
   x = uint8(x)
   T=500*600
   V=sum(sum(x))
   p(i)=V/T
end
```

**附录 2**

介绍：分别将原始图像和对应的标签图像切割成尺寸为 256×256 的图片，再分别将切割后的原始图像和标签图像进行水平、垂直翻转和对角镜像的数据增强处理的 python 代码

```
#图像切割
import os
import gdal
import numpy as np
#  读取 tif 数据集
def readTif(fileName):
    dataset = gdal.Open(fileName)
    if dataset == None:
        print(fileName + "文件无法打开")
    return dataset
#  保存 tif 文件函数
def writeTiff(im_data, im_geotrans, im_proj, path):
    if 'int8' in im_data.dtype.name:
        datatype = gdal.GDT_Byte
    elif 'int16' in im_data.dtype.name:
        datatype = gdal.GDT_UInt16
    else:
        datatype = gdal.GDT_Float32
    if len(im_data.shape) == 3:
        im_bands, im_height, im_width = im_data.shape
    elif len(im_data.shape) == 2:
        im_data = np.array([im_data])
        im_bands, im_height, im_width = im_data.shape
```

```
    # 创建文件
    driver = gdal.GetDriverByName("GTiff")
    dataset = driver.Create(path, int(im_width), int(im_height), int(im_bands), datatype)
    if (dataset != None):
        dataset.SetGeoTransform(im_geotrans)  # 写入仿射变换参数
        dataset.SetProjection(im_proj)  # 写入投影
    for i in range(im_bands):
        dataset.GetRasterBand(i + 1).WriteArray(im_data[i])
    del dataset
"""
滑动窗口裁剪函数
TifPath 影像路径
SavePath 裁剪后保存目录
CropSize 裁剪尺寸
RepetitionRate 重复率
'''
def TifCrop(TifPath, SavePath, CropSize, RepetitionRate):
    dataset_img = readTif(TifPath)
    width = dataset_img.RasterXSize
    height = dataset_img.RasterYSize
    proj = dataset_img.GetProjection()
    geotrans = dataset_img.GetGeoTransform()
    img = dataset_img.ReadAsArray(0, 0, width, height)  # 获取数据
    #  获取当前文件夹的文件个数 len,并以 len+1 命名即将裁剪得到的图像
    new_name = len(os.listdir(SavePath)) + 1
    #  裁剪图片,重复率为 RepetitionRate
    for i in range(int((height - CropSize * RepetitionRate) / (CropSize * (1 - RepetitionRate)))):
        for j in range(int((width - CropSize * RepetitionRate) / (CropSize * (1 - RepetitionRate)))):
            #  如果图像是单波段
            if (len(img.shape) == 2):
                cropped = img[
                          int(i * CropSize * (1 - RepetitionRate)): int(i * CropSize * (1 - RepetitionRate)) + Cro
pSize,
                          int(j * CropSize * (1 - RepetitionRate)): int(j * CropSize * (1 - RepetitionRate)) + Cro
pSize]
            #  如果图像是多波段
            else:
                cropped = img[:,
                          int(i * CropSize * (1 - RepetitionRate)): int(i * CropSize * (1 - RepetitionRate)) + Cro
pSize,
                          int(j * CropSize * (1 - RepetitionRate)): int(j * CropSize * (1 - RepetitionRate)) + Cro
pSize]
            #  写图像
            writeTiff(cropped, geotrans, proj, SavePath + "/%d.tif" % new_name)
            #  文件名 + 1
            new_name = new_name + 1
    #  向前裁剪最后一列
    for i in range(int((height - CropSize * RepetitionRate) / (CropSize * (1 - RepetitionRate)))):
        if (len(img.shape) == 2):
            cropped = img[int(i * CropSize * (1 - RepetitionRate)): int(i * CropSize * (1 - RepetitionRate))
+ CropSize,
                      (width - CropSize): width]
        else:
            cropped = img[:,
                      int(i * CropSize * (1 - RepetitionRate)): int(i * CropSize * (1 - RepetitionRate)) + CropSi
ze,
```

```
                    (width - CropSize): width]
        # 写图像
        writeTiff(cropped, geotrans, proj, SavePath + "/%d.tif" % new_name)
        new_name = new_name + 1
    # 向前裁剪最后一行
    for j in range(int((width - CropSize * RepetitionRate) / (CropSize * (1 - RepetitionRate)))):
        if (len(img.shape) == 2):
            cropped = img[(height - CropSize): height,
                    int(j * CropSize * (1 - RepetitionRate)): int(j * CropSize * (1 - RepetitionRate)) + CropSi
ze]
        else:
            cropped = img[:,
                    (height - CropSize): height,
                    int(j * CropSize * (1 - RepetitionRate)): int(j * CropSize * (1 - RepetitionRate)) + CropSi
ze]
        writeTiff(cropped, geotrans, proj, SavePath + "/%d.tif" % new_name)
        # 文件名 + 1
        new_name = new_name + 1
    # 裁剪右下角
    if (len(img.shape) == 2):
        cropped = img[(height - CropSize): height,
                (width - CropSize): width]
    else:
        cropped = img[:,
                (height - CropSize): height,
                (width - CropSize): width]
    writeTiff(cropped, geotrans, proj, SavePath + "/%d.tif" % new_name)
    new_name = new_name + 1
# 将影像 1 裁剪为重复率为 0.1 的 256×256 的数据集
# TifCrop(r"Data\data2\tif\data2.tif",
#         r"Data\train\image1", 256, 0.1)
# TifCrop(r"Data\data2\label\label.tif",
#         r"data\train\label1", 256, 0.1)
imageList = os.listdir("pic\\image")
for i in range(len(imageList)):
    TifCrop(r"pic\\image\\Data%d.tif"%(i+1),
            r"Data\\cutimg", 256, 0.1)
    TifCrop(r"pic\\label\\Data%d_reference.tif"%(i+1),
            r"Data\\cutlab", 256, 0.1)
#
# TifCrop(r"pic\\image\\Data1.tif",
#             r"Data\\cutimg", 256, 0.1)
# TifCrop(r"pic\\label\\Data1_reference.tif",
#         r"Data\\cutlab", 256, 0.1)
#图像水平、垂直翻转、对角镜像
import gdal
import numpy as np
import os
import cv2
# 读取 tif 数据集
def readTif(fileName, xoff=0, yoff=0, data_width=0, data_height=0):
    dataset = gdal.Open(fileName)
    if dataset == None:
        print(fileName + "文件无法打开")
    # 栅格矩阵的列数
    width = dataset.RasterXSize
    # 栅格矩阵的行数
```

```
    height = dataset.RasterYSize
    # 波段数
    bands = dataset.RasterCount
    # 获取数据
    if (data_width == 0 and data_height == 0):
        data_width = width
        data_height = height
    data = dataset.ReadAsArray(xoff, yoff, data_width, data_height)
    # 获取仿射矩阵信息
    geotrans = dataset.GetGeoTransform()
    # 获取投影信息
    proj = dataset.GetProjection()
    return width, height, bands, data, geotrans, proj
# 保存 tif 文件函数
def writeTiff(im_data, im_geotrans, im_proj, path):
    if 'int8' in im_data.dtype.name:
        datatype = gdal.GDT_Byte
    elif 'int16' in im_data.dtype.name:
        datatype = gdal.GDT_UInt16
    else:
        datatype = gdal.GDT_Float32
    if len(im_data.shape) == 3:
        im_bands, im_height, im_width = im_data.shape
    elif len(im_data.shape) == 2:
        im_data = np.array([im_data])
        im_bands, im_height, im_width = im_data.shape
    # 创建文件
    driver = gdal.GetDriverByName("GTiff")
    dataset = driver.Create(path, int(im_width), int(im_height), int(im_bands), datatype)
    if (dataset != None):
        dataset.SetGeoTransform(im_geotrans)  # 写入仿射变换参数
        dataset.SetProjection(im_proj)  # 写入投影
    for i in range(im_bands):
        dataset.GetRasterBand(i + 1).WriteArray(im_data[i])
    del dataset
# train_image_path = r"Data\train\image1"
# train_label_path = r"Data\train\label"
train_image_path = r"Data\\cutimg"
train_label_path = r"Data\\cutlab"
# 进行几何变换数据增强
imageList = os.listdir(train_image_path)
labelList = os.listdir(train_label_path)
tran_num = len(imageList) + 1
for i in range(len(imageList)):
    # 图像
    img_file = train_image_path + "\\" + imageList[i]
    im_width, im_height, im_bands, im_data, im_geotrans, im_proj = readTif(img_file)
    # 标签
    label_file = train_label_path + "\\" + labelList[i]
    label = cv2.imread(label_file)
    # 图像水平翻转
    im_data_hor = np.flip(im_data, axis=2)
    hor_path = train_image_path + "\\" + str(tran_num) + imageList[i][-4:]
    writeTiff(im_data_hor, im_geotrans, im_proj, hor_path)
    # 标签水平翻转
    Hor = cv2.flip(label, 1)
```

```
    hor_path = train_label_path + "\\" + str(tran_num) + labelList[i][-4:]
    cv2.imwrite(hor_path, Hor)
    tran_num += 1
    # 图像垂直翻转
    im_data_vec = np.flip(im_data, axis=1)
    vec_path = train_image_path + "\\" + str(tran_num) + imageList[i][-4:]
    writeTiff(im_data_vec, im_geotrans, im_proj, vec_path)
    # 标签垂直翻转
    Vec = cv2.flip(label, 0)
    vec_path = train_label_path + "\\" + str(tran_num) + labelList[i][-4:]
    cv2.imwrite(vec_path, Vec)
    tran_num += 1
    # 图像对角镜像
    im_data_dia = np.flip(im_data_vec, axis=2)
    dia_path = train_image_path + "\\" + str(tran_num) + imageList[i][-4:]
    writeTiff(im_data_dia, im_geotrans, im_proj, dia_path)
    # 标签对角镜像
    Dia = cv2.flip(label, -1)
    dia_path = train_label_path + "\\" + str(tran_num) + labelList[i][-4:]
    cv2.imwrite(dia_path, Dia)
     tran_num += 1
```

**附录 3**

介绍：进行数据预处理，并定义训练数据生成器和测试数据生成器的 python 代码

```
import numpy as np
import os
import random
import gdal
import cv2
# 读取图像像素矩阵 1
# fileName 图像文件名
def readTif(fileName):
    dataset = gdal.Open(fileName)
    width = dataset.RasterXSize
    height = dataset.RasterYSize
    GdalImg_data = dataset.ReadAsArray(0, 0, width, height)
    return GdalImg_data
# 数据预处理：图像归一化+标签 onehot 编码      2
# img 图像数据
# label 标签数据
# classNum 类别总数(含背景)
# colorDict_GRAY 颜色字典
def dataPreprocess(img, label, classNum, colorDict_GRAY):
    # 归一化
    img = img / 255.0
    for i in range(colorDict_GRAY.shape[0]):
        label[label == colorDict_GRAY[i][0]] = i
    # 将数据厚度扩展到 classNum 层
    new_label = np.zeros(label.shape + (classNum,))
    # 将平面的 label 的每类，都单独变成一层
    for i in range(classNum):
        new_label[label == i, i] = 1
    label = new_label
    return (img, label)
```

```
# 获取颜色字典    3
# labelFolder 标签文件夹,之所以遍历文件夹是因为一张标签可能不包含所有类别颜色
# classNum 类别总数(含背景)
def color_dict(labelFolder, classNum):
    colorDict = []
    # 获取文件夹内的文件名
    ImageNameList = os.listdir(labelFolder)
    for i in range(len(ImageNameList)):
        ImagePath = labelFolder + "/" + ImageNameList[i]
        img = cv2.imread(ImagePath).astype(np.uint32)
        # 如果是灰度，转成 RGB
        if (len(img.shape) == 2):
            img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB).astype(np.uint32)
        # 为了提取唯一值，将 RGB 转成一个数
        img_new = img[:, :, 0] * 1000000 + img[:, :, 1] * 1000 + img[:, :, 2]
        unique = np.unique(img_new)
        # 将第 i 个像素矩阵的唯一值添加到 colorDict 中
        for j in range(unique.shape[0]):
            colorDict.append(unique[j])
        # 对目前 i 个像素矩阵里的唯一值再取唯一值
        colorDict = sorted(set(colorDict))
        # 若唯一值数目等于总类数(包括背景)ClassNum，停止遍历剩余的图像
        if (len(colorDict) == classNum):
            break
    # 存储颜色的 RGB 字典，用于预测时的渲染结果
    colorDict_RGB = []
    for k in range(len(colorDict)):
        # 对没有达到九位数字的结果进行左边补零(eg:5,201,111->005,201,111)
        color = str(colorDict[k]).rjust(9, '0')
        # 前 3 位 R,中 3 位 G,后 3 位 B
        color_RGB = [int(color[0: 3]), int(color[3: 6]), int(color[6: 9])]
        colorDict_RGB.append(color_RGB)
    # 转为 numpy 格式
    colorDict_RGB = np.array(colorDict_RGB)
    # 存储颜色的 GRAY 字典，用于预处理时的 onehot 编码
    colorDict_GRAY = colorDict_RGB.reshape((colorDict_RGB.shape[0], 1, colorDict_RGB.shape[1])).astype(np.uint8)
    colorDict_GRAY = cv2.cvtColor(colorDict_GRAY, cv2.COLOR_BGR2GRAY)
    return colorDict_RGB, colorDict_GRAY
# 训练数据生成器
# batch_size 批大小
# train_image_path 训练图像路径
# train_label_path 训练标签路径
# classNum 类别总数(含背景)
# colorDict_GRAY 颜色字典
# resize_shape resize 大小
def trainGenerator(batch_size, train_image_path, train_label_path, classNum, colorDict_GRAY, resize_shape=None):
    imageList = os.listdir(train_image_path)
    labelList = os.listdir(train_label_path)
    img = readTif(train_image_path + "\\" + imageList[0])
    # GDAL 读数据是(BandNum,Width,Height)要转换为->(Width,Height,BandNum)
    img = img.swapaxes(1, 0)
    img = img.swapaxes(1, 2)
```

```
    # 无限生成数据
    while (True):
        img_generator = np.zeros((batch_size, img.shape[0], img.shape[1], img.shape[2]), np.uint8)
        label_generator = np.zeros((batch_size, img.shape[0], img.shape[1]), np.uint8)
        if (resize_shape != None):
            img_generator = np.zeros((batch_size, resize_shape[0], resize_shape[1], resize_shape[2]), np.uint8)
            label_generator = np.zeros((batch_size, resize_shape[0], resize_shape[1]), np.uint8)
        # 随机生成一个 batch 的起点
        rand = random.randint(0, len(imageList) - batch_size)
        for j in range(batch_size):
            img = readTif(train_image_path + "\\" + imageList[rand + j])
            img = img.swapaxes(1, 0)
            img = img.swapaxes(1, 2)
            # 改变图像尺寸至特定尺寸(
            # 因为 resize 用的不多，我就用了 OpenCV 实现的，这个不支持多波段，需要的话可以用 np 进行 resize
            if (resize_shape != None):
                img = cv2.resize(img, (resize_shape[0], resize_shape[1]))
            img_generator[j] = img
            label = readTif(train_label_path + "\\" + labelList[rand + j]).astype(np.uint8)
            # 若为彩色，转为灰度
            if (len(label.shape) == 3):
                label = label.swapaxes(1, 0)
                label = label.swapaxes(1, 2)
                label = cv2.cvtColor(label, cv2.COLOR_RGB2GRAY)
            if (resize_shape != None):
                label = cv2.resize(label, (resize_shape[0], resize_shape[1]))
            label_generator[j] = label
        img_generator, label_generator = dataPreprocess(img_generator, label_generator, classNum, colorDict_GRAY)
        yield (img_generator, label_generator)
# 测试数据生成器
# test_iamge_path 测试数据路径
# resize_shape resize 大小
def testGenerator(test_iamge_path, resize_shape=None):
    imageList = os.listdir(test_iamge_path)
    for i in range(len(imageList)):
        img = readTif(test_iamge_path + "\\" + imageList[i])
        img = img.swapaxes(1, 0)
        img = img.swapaxes(1, 2)
        # 归一化
        img = img / 255.0
        if (resize_shape != None):
            # 改变图像尺寸至特定尺寸
            img = cv2.resize(img, (resize_shape[0], resize_shape[1]))
        # 将测试图片扩展一个维度,与训练时的输入[batch_size,img.shape]保持一致
        img = np.reshape(img, (1,) + img.shape)
        yield img
# 保存结果
# test_iamge_path 测试数据图像路径
# test_predict_path 测试数据图像预测结果路径
# model_predict 模型的预测结果
# color_dict 颜色词典
def saveResult(test_image_path, test_predict_path, model_predict, color_dict, output_size):
```

```
    imageList = os.listdir(test_image_path)
    for i, img in enumerate(model_predict):
        channel_max = np.argmax(img, axis=-1)
        img_out = np.uint8(color_dict[channel_max.astype(np.uint8)])
        # 修改差值方式为最邻近差值
        img_out = cv2.resize(img_out, (output_size[0], output_size[1]), interpolation=cv2.INTER_NEAREST)
        # 保存为无损压缩 png
        cv2.imwrite(test_predict_path + "\\" + imageList[i][:-4] + ".png",
```

**附录 4**

介绍：UNet 模型定义的 python 代码

```
from keras.models import Model
from keras.layers import Input, BatchNormalization, Conv2D, MaxPooling2D, Dropout, concatenate, merge, UpSampling2D
from keras.optimizers import Adam
def unet(pretrained_weights=None, input_size=(256, 256, 4), classNum=2, learning_rate=1e-5):
    inputs = Input(input_size)
    # 2D 卷积层
    conv1 = BatchNormalization()(
        Conv2D(64, 3, activation='relu', padding='same', kernel_initializer='he_normal')(inputs))
    conv1 = BatchNormalization()(
        Conv2D(64, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv1))
    # 对于空间数据的最大池化
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = BatchNormalization()(
        Conv2D(128, 3, activation='relu', padding='same', kernel_initializer='he_normal')(pool1))
    conv2 = BatchNormalization()(
        Conv2D(128, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv2))
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = BatchNormalization()(
        Conv2D(256, 3, activation='relu', padding='same', kernel_initializer='he_normal')(pool2))
    conv3 = BatchNormalization()(
        Conv2D(256, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv3))
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv4 = BatchNormalization()(
        Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(pool3))
    conv4 = BatchNormalization()(
        Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv4))
    # Dropout 正规化，防止过拟合
    drop4 = Dropout(0.2)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)
    conv5 = BatchNormalization()(
        Conv2D(1024, 3, activation='relu', padding='same', kernel_initializer='he_normal')(pool4))
    conv5 = BatchNormalization()(
        Conv2D(1024, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv5))
    drop5 = Dropout(0.2)(conv5)
    # 上采样之后再进行卷积，相当于转置卷积操作
    up6 = Conv2D(512, 2, activation='relu', padding='same', kernel_initializer='he_normal')(
        UpSampling2D(size=(2, 2))(drop5))
    try:
        merge6 = concatenate([drop4, up6], axis=3)
    except:
        merge6 = merge([drop4, up6], mode='concat', concat_axis=3)
    conv6 = BatchNormalization()(
        Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(merge6))
```

```
    conv6 = BatchNormalization()(
        Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv6))
    up7 = Conv2D(256, 2, activation='relu', padding='same', kernel_initializer='he_normal')(
        UpSampling2D(size=(2, 2))(conv6))
    try:
        merge7 = concatenate([conv3, up7], axis=3)
    except:
        merge7 = merge([conv3, up7], mode='concat', concat_axis=3)
    conv7 = BatchNormalization()(
        Conv2D(256, 3, activation='relu', padding='same', kernel_initializer='he_normal')(merge7))
    conv7 = BatchNormalization()(
        Conv2D(256, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv7))
    up8 = Conv2D(128, 2, activation='relu', padding='same', kernel_initializer='he_normal')(
        UpSampling2D(size=(2, 2))(conv7))
    try:
        merge8 = concatenate([conv2, up8], axis=3)
    except:
        merge8 = merge([conv2, up8], mode='concat', concat_axis=3)
    conv8 = BatchNormalization()(
        Conv2D(128, 3, activation='relu', padding='same', kernel_initializer='he_normal')(merge8))
    conv8 = BatchNormalization()(
        Conv2D(128, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv8))
    up9 = Conv2D(64, 2, activation='relu', padding='same', kernel_initializer='he_normal')(
        UpSampling2D(size=(2, 2))(conv8))
    try:
        merge9 = concatenate([conv1, up9], axis=3)
    except:
        merge9 = merge([conv1, up9], mode='concat', concat_axis=3)
    conv9 = BatchNormalization()(
        Conv2D(64, 3, activation='relu', padding='same', kernel_initializer='he_normal')(merge9))
    conv9 = BatchNormalization()(
        Conv2D(64, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv9))
    conv9 = Conv2D(2, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv9)
    conv10 = Conv2D(classNum, 1, activation='softmax')(conv9)
    model = Model(inputs=inputs, outputs=conv10)
    # 用于配置训练模型（优化器、目标函数、模型评估标准）
    model.compile(optimizer=Adam(lr=learning_rate), loss='categorical_crossentropy', metrics=['accura
cy'])
    # 如果有预训练的权重
    if (pretrained_weights):
        model.load_weights(pretrained_weights)
    return model
```

**附录 5**

介绍：UNet 模型训练的 python 代码

```
import os
import sys
sys.path.append(os.path.dirname(os.path.abspath(__file__)))
from seg_unet import unet
# from Model.seg_hrnet import seg_hrnet
from dataprocess import trainGenerator, color_dict
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
import datetime
import xlwt
import os
"""
```

```
数据集相关参数
'''
# 训练数据图像路径
train_image_path = "Data\\imag\\train"
# 训练数据标签路径
train_label_path = "Data\\label\\train"
# 验证数据图像路径
validation_image_path = "Data\\imag\\yanz"
# 验证数据标签路径
validation_label_path = "Data\\label\\yanz"
"""
模型相关参数
'''
# 批大小
batch_size = 8
# 类的数目(包括背景)
classNum = 2
# 模型输入图像大小
input_size = (256, 256, 1)
# 训练模型的迭代总轮数
epochs = 100
# 初始学习率
learning_rate = 1e-4
# 预训练模型地址
premodel_path = None
# 训练模型保存地址
model_path = "Model\\unet_model.hdf5"
# 训练数据数目
train_num = len(os.listdir(train_image_path))
# 验证数据数目
validation_num = len(os.listdir(validation_image_path))
# 训练集每个 epoch 有多少个 batch_size
steps_per_epoch = train_num / batch_size
# 验证集每个 epoch 有多少个 batch_size
validation_steps = validation_num / batch_size
# 标签的颜色字典,用于 onehot 编码
colorDict_RGB, colorDict_GRAY = color_dict(train_label_path, classNum)
# 得到一个生成器，以 batch_size 的速率生成训练数据
train_Generator = trainGenerator(batch_size,
                                 train_image_path,
                                 train_label_path,
                                 classNum,
                                 colorDict_GRAY,
                                 input_size)
# 得到一个生成器，以 batch_size 的速率生成验证数据
validation_data = trainGenerator(batch_size,
                                 validation_image_path,
                                 validation_label_path,
                                 classNum,
                                 colorDict_GRAY,
                                 input_size)
# 定义模型
model = unet(pretrained_weights=premodel_path,
             input_size=input_size,
             classNum=classNum,
```

```
            learning_rate=learning_rate)
# model = seg_hrnet(pretrained_weights = premodel_path,
#                 input_size = input_size,
#                 classNum = classNum,
#                 learning_rate = learning_rate)
# 打印模型结构
model.summary()
# 回调函数
# val_loss 连续 10 轮没有下降则停止训练
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
# 当 3 个 epoch 过去而 val_loss 不下降时，学习率减半
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, verbose=1)
model_checkpoint = ModelCheckpoint(model_path,
                                   monitor='loss',
                                   verbose=1,  # 日志显示模式:0->安静模式,1->进度条,2->每轮一行
                                   save_best_only=True)
# 获取当前时间
start_time = datetime.datetime.now()
# 模型训练
history = model.fit_generator(train_Generator,
                              steps_per_epoch=steps_per_epoch,
                              epochs=epochs,
                              callbacks=[early_stopping, model_checkpoint, model_checkpoint],
                              validation_data=validation_data,
                              validation_steps=validation_steps)
# 训练总时间
end_time = datetime.datetime.now()
log_time = "训练总时间: " + str((end_time - start_time).seconds / 60) + "m"
time = datetime.datetime.strftime(datetime.datetime.now(), '%Y%m%d-%H%M%S')
print(log_time)
with open('TrainTime_%s.txt' % time, 'w') as f:
    f.write(log_time)
# 保存并绘制 loss,acc
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
book = xlwt.Workbook(encoding='utf-8', style_compression=0)
sheet = book.add_sheet('test', cell_overwrite_ok=True)
for i in range(len(acc)):
    sheet.write(i, 0, acc[i])
    sheet.write(i, 1, val_acc[i])
    sheet.write(i, 2, loss[i])
    sheet.write(i, 3, val_loss[i])
book.save(r'AccAndLoss_%s.xls' % time)
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.savefig("accuracy_%s.png" % time, dpi=300)
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.savefig("loss_%s.png" % time, dpi=300)
```

```
125. plt.show()
```

**附录 6**

介绍：将测试图裁剪后通过模型预测并拼接出最终标签结果图的 python 代码

```
1.  import gdal
2.  import numpy as np
3.  from keras.models import load_model
4.  from keras import losses
5.  import datetime
6.  import math
7.  import sys
8.  # 读取 tif 数据集
9.  def readTif(fileName, xoff = 0, yoff = 0, data_width = 0, data_height = 0):
10.     dataset = gdal.Open(fileName)
11.     if dataset == None:
12.         print(fileName + "文件无法打开")
13.     # 栅格矩阵的列数
14.     width = dataset.RasterXSize
15.     # 栅格矩阵的行数
16.     height = dataset.RasterYSize
17.     # 波段数
18.     bands = dataset.RasterCount
19.     # 获取数据
20.     if(data_width == 0 and data_height == 0):
21.         data_width = width
22.         data_height = height
23.     data = dataset.ReadAsArray(xoff, yoff, data_width, data_height)
24.     # 获取仿射矩阵信息
25.     geotrans = dataset.GetGeoTransform()
26.     # 获取投影信息
27.     proj = dataset.GetProjection()
28.     return width, height, bands, data, geotrans, proj
29. # 保存 tif 文件函数
30. def writeTiff(im_data, im_geotrans, im_proj, path):
31.     if 'int8' in im_data.dtype.name:
32.         datatype = gdal.GDT_Byte
33.     elif 'int16' in im_data.dtype.name:
34.         datatype = gdal.GDT_UInt16
35.     else:
36.         datatype = gdal.GDT_Float32
37.     if len(im_data.shape) == 3:
38.         im_bands, im_height, im_width = im_data.shape
39.     elif len(im_data.shape) == 2:
40.         im_data = np.array([im_data])
41.         im_bands, im_height, im_width = im_data.shape
42.     #创建文件
43.     driver = gdal.GetDriverByName("GTiff")
44.     dataset = driver.Create(path, int(im_width), int(im_height), int(im_bands), datatype)
45.     if(dataset!= None):
46.         dataset.SetGeoTransform(im_geotrans) #写入仿射变换参数
47.         dataset.SetProjection(im_proj) #写入投影
48.     for i in range(im_bands):
49.         dataset.GetRasterBand(i+1).WriteArray(im_data[i])
50.     del dataset
51. # tif 裁剪（tif 像素数据，裁剪边长）
```

```
def TifCroppingArray(img, SideLength):
    # 裁剪链表
    TifArrayReturn = []
    # 列上图像块数目
    ColumnNum = int((img.shape[0] - SideLength * 2) / (256 - SideLength * 2))
    # 行上图像块数目
    RowNum = int((img.shape[1] - SideLength * 2) / (256 - SideLength * 2))
    for i in range(ColumnNum):
        TifArray = []
        for j in range(RowNum):
            cropped = img[i * (256 - SideLength * 2) : i * (256 - SideLength * 2) + 256,
                          j * (256 - SideLength * 2) : j * (256 - SideLength * 2) + 256]
            TifArray.append(cropped)
        TifArrayReturn.append(TifArray)
    # 考虑到行列会有剩余的情况，向前裁剪一行和一列
    # 向前裁剪最后一列
    for i in range(ColumnNum):
        cropped = img[i * (256 - SideLength * 2) : i * (256 - SideLength * 2) + 256,
                      (img.shape[1] - 256) : img.shape[1]]
        TifArrayReturn[i].append(cropped)
    # 向前裁剪最后一行
    TifArray = []
    for j in range(RowNum):
        cropped = img[(img.shape[0] - 256) : img.shape[0],
                      j * (256-SideLength*2) : j * (256 - SideLength * 2) + 256]
        TifArray.append(cropped)
    # 向前裁剪右下角
    cropped = img[(img.shape[0] - 256) : img.shape[0],
                  (img.shape[1] - 256) : img.shape[1]]
    TifArray.append(cropped)
    TifArrayReturn.append(TifArray)
    # 列上的剩余数
    ColumnOver = (img.shape[0] - SideLength * 2) % (256 - SideLength * 2) + SideLength
    # 行上的剩余数
    RowOver = (img.shape[1] - SideLength * 2) % (256 - SideLength * 2) + SideLength
    return TifArrayReturn, RowOver, ColumnOver
# 标签可视化，即为第 n 类赋上 n 值
def labelVisualize(img):
    img_out = np.zeros((img.shape[0],img.shape[1]))
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            # 为第 n 类赋上 n 值
            img_out[i][j] = np.argmax(img[i][j])
    return img_out
# 对测试图片进行归一化，并使其维度上和训练图片保持一致
def testGenerator(TifArray):
    for i in range(len(TifArray)):
        for j in range(len(TifArray[0])):
            img = TifArray[i][j]
            # 归一化
            img = img / 255.0
            # 在不改变数据内容情况下，改变 shape
            img = np.reshape(img,(1,)+img.shape)
            yield img
# 获得结果矩阵
def Result(shape, TifArray, npyfile, num_class, RepetitiveLength, RowOver, ColumnOver):
```

```
    result = np.zeros(shape, np.uint8)
    # j 来标记行数
    j = 0
    for i,item in enumerate(npyfile):
        img = labelVisualize(item)
        img = img.astype(np.uint8)
        # 最左侧一列特殊考虑，左边的边缘要拼接进去
        if(i % len(TifArray[0]) == 0):
            # 第一行的要再特殊考虑，上边的边缘要考虑进去
            if(j == 0):
                result[0 : 256 - RepetitiveLength, 0 : 256-RepetitiveLength] = img[0 : 256 - RepetitiveLength, 0 : 256 - RepetitiveLength]
            # 最后一行的要再特殊考虑，下边的边缘要考虑进去
            elif(j == len(TifArray) - 1):
                # 原来错误的
                #result[shape[0] - ColumnOver : shape[0], 0 : 256 - RepetitiveLength] = img[0 : ColumnOver, 0 : 256 - RepetitiveLength]
                # 后来修改的
                result[shape[0] - ColumnOver - RepetitiveLength: shape[0], 0 : 256 - RepetitiveLength] = img[256 - ColumnOver - RepetitiveLength : 256, 0 : 256 - RepetitiveLength]
            else:
                result[j * (256 - 2 * RepetitiveLength) + RepetitiveLength : (j + 1) * (256 - 2 * RepetitiveLength) + RepetitiveLength,
                       0:256-RepetitiveLength] = img[RepetitiveLength : 256 - RepetitiveLength, 0 : 256 - RepetitiveLength]
        # 最右侧一列特殊考虑，右边的边缘要拼接进去
        elif(i % len(TifArray[0]) == len(TifArray[0]) - 1):
            # 第一行的要再特殊考虑，上边的边缘要考虑进去
            if(j == 0):
                result[0 : 256 - RepetitiveLength, shape[1] - RowOver: shape[1]] = img[0 : 256 - RepetitiveLength, 256 -  RowOver: 256]
            # 最后一行的要再特殊考虑，下边的边缘要考虑进去
            elif(j == len(TifArray) - 1):
                result[shape[0] - ColumnOver : shape[0], shape[1] - RowOver : shape[1]] = img[256 - ColumnOver : 256, 256 - RowOver : 256]
            else:
                result[j * (256 - 2 * RepetitiveLength) + RepetitiveLength : (j + 1) * (256 - 2 * RepetitiveLength) + RepetitiveLength,
                       shape[1] - RowOver : shape[1]] = img[RepetitiveLength : 256 - RepetitiveLength, 256 - RowOver : 256]
            # 走完每一行的最右侧，行数+1
            j = j + 1
        # 不是最左侧也不是最右侧的情况
        else:
            # 第一行的要特殊考虑，上边的边缘要考虑进去
            if(j == 0):
                result[0 : 256 - RepetitiveLength,
                       (i - j * len(TifArray[0])) * (256 - 2 * RepetitiveLength) + RepetitiveLength : (i - j * len(TifArray[0]) + 1) * (256 - 2 * RepetitiveLength) + RepetitiveLength
                       ] = img[0 : 256 - RepetitiveLength, RepetitiveLength : 256 - RepetitiveLength]
            # 最后一行的要特殊考虑，下边的边缘要考虑进去
            if(j == len(TifArray) - 1):
                result[shape[0] - ColumnOver : shape[0],
                       (i - j * len(TifArray[0])) * (256 - 2 * RepetitiveLength) + RepetitiveLength : (i - j * len(TifArray[0]) + 1) * (256 - 2 * RepetitiveLength) + RepetitiveLength
                       ] = img[256 - ColumnOver : 256, RepetitiveLength : 256 - RepetitiveLength]
```

```
            else:
                result[j * (256 - 2 * RepetitiveLength) + RepetitiveLength : (j + 1) * (256 - 2 * RepetitiveLength) + RepetitiveLength,
                       (i - j * len(TifArray[0])) * (256 - 2 * RepetitiveLength) + RepetitiveLength : (i - j * len(TifArray[0]) + 1) * (256 - 2 * RepetitiveLength) + RepetitiveLength,
                       ] = img[RepetitiveLength : 256 - RepetitiveLength, RepetitiveLength : 256 - RepetitiveLength]
    return result
area_perc = 0.5
TifPath = r"pic\\image\\Test3.tif"#Test3.tif
ModelPath = r"Model\\unet_model.hdf5"
ResultPath = r"Result\\Result.tif"
RepetitiveLength = int((1 - math.sqrt(area_perc)) * 256 / 2)
# 记录测试消耗时间
testtime = []
# 获取当前时间
starttime = datetime.datetime.now()
im_width, im_height, im_bands, im_data, im_geotrans, im_proj = readTif(TifPath)
im_data = im_data.swapaxes(1, 0)
im_data = im_data.swapaxes(1, 2)
TifArray, RowOver, ColumnOver = TifCroppingArray(im_data, RepetitiveLength)
endtime = datetime.datetime.now()
text = "读取 tif 并裁剪预处理完毕,目前耗时间: " + str((endtime - starttime).seconds) + "s"
print(text)
testtime.append(text)
model = load_model(ModelPath, custom_objects = {'ignore_edges_loss': 'ignore_edges_loss'})
testGene = testGenerator(TifArray)
results = model.predict_generator(testGene,
                                  len(TifArray) * len(TifArray[0]),
                                  verbose = 1)
endtime = datetime.datetime.now()
text = "模型预测完毕,目前耗时间: " + str((endtime - starttime).seconds) + "s"
print(text)
testtime.append(text)
#保存结果
result_shape = (im_data.shape[0], im_data.shape[1])
result_data = Result(result_shape, TifArray, results, 2, RepetitiveLength, RowOver, ColumnOver)
writeTiff(result_data, im_geotrans, im_proj, ResultPath)
endtime = datetime.datetime.now()
text = "结果拼接完毕,目前耗时间: " + str((endtime - starttime).seconds) + "s"
print(text)
testtime.append(text)
time = datetime.datetime.strftime(datetime.datetime.now(), '%Y%m%d-%H%M%S')
with open('timelog_%s.txt'%time, 'w') as f:
    for i in range(len(testtime)):
        f.write(testtime[i])
        f.write("\r\n")
```

**附录 7**

介绍：将已切割、数据增强并预处理的原始数据按照 80%和 20%比例随机分成训练集和验证集，并对验证集进行预测进而求得模型的混淆矩阵、Kappa 系数等精度指标的 python 代码

```
1. #分成训练和验证集
2. import os
3. import random
```

```
import shutil
train_image_path = r"Data\cutimg"
train_label_path = r"Data\cutlab"
imageList = os.listdir(train_image_path)
labelList = os.listdir(train_label_path)
tran_num = len(imageList) + 1
print(len(imageList))
#得到训练数据下标
k=len(imageList)*0.8
s=[]
while(len(s)<k):
    x=random.randint(1,tran_num-1)
    if x not in s:
        s.append(x)
# 得到验证数据下标
s1=[]
for i in range(1,tran_num):
    if i not in s:
        s1.append(i)
for i in s:
    print(train_image_path + '\\' + str(i)+".tif")
    print("imag" + '\\'+"train"+'\\' +  str(i)+".tif")
    shutil.move(train_image_path + '\\' + str(i)+".tif", "Data\\imag" + '\\'+"train")
    shutil.move(train_label_path + '\\' +  str(i)+".tif", "Data\\label" + '\\' + "train" )
for i in s1:
    shutil.move(train_image_path + '\\' +  str(i)+".tif", "Data\\imag" + '\\' + "yanz" )
    shutil.move(train_label_path + '\\' +  str(i)+".tif", "Data\\label" + '\\' + "yanz" )
#验证集预测
from keras.models import load_model
from dataprocess import testGenerator, saveResult, color_dict
import os
# 训练模型保存地址
model_path = r"Model\\unet_model.hdf5"
# 测试数据路径
test_iamge_path = r"Data\imag\yanz"
# 结果保存路径
save_path = r"test_result"
# 测试数据数目
test_num = len(os.listdir(test_iamge_path))
# 类的数目(包括背景)
classNum = 2
# 模型输入图像大小
input_size = (256, 256, 3)
# 生成图像大小
output_size = (256, 256)
# 测试数据标签路径
train_label_path = "Data\label\yanz"
# 标签的颜色字典
colorDict_RGB, colorDict_GRAY = color_dict(train_label_path, classNum)
model = load_model(model_path)
testGene = testGenerator(test_iamge_path, input_size)
# 预测值的 Numpy 数组
results = model.predict_generator(testGene,
                                  test_num,
                                  verbose = 1)
# 保存结果
saveResult(test_iamge_path, save_path, results, colorDict_GRAY, output_size)
```

```
#计算精度指标
import numpy as np
import cv2
import os
"""
混淆矩阵
P\L     P    N
P      TP    FP
N      FN    TN
"""
# 获取颜色字典
# labelFolder 标签文件夹,之所以遍历文件夹是因为一张标签可能不包含所有类别颜色
# classNum 类别总数(含背景)
def color_dict(labelFolder, classNum):
    colorDict = []
    # 获取文件夹内的文件名
    ImageNameList = os.listdir(labelFolder)
    for i in range(len(ImageNameList)):
        ImagePath = labelFolder + "/" + ImageNameList[i]
        img = cv2.imread(ImagePath).astype(np.uint32)
        # 如果是灰度，转成 RGB
        if (len(img.shape) == 2):
            img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB).astype(np.uint32)
        # 为了提取唯一值，将 RGB 转成一个数
        img_new = img[:, :, 0] * 1000000 + img[:, :, 1] * 1000 + img[:, :, 2]
        unique = np.unique(img_new)
        # 将第 i 个像素矩阵的唯一值添加到 colorDict 中
        for j in range(unique.shape[0]):
            colorDict.append(unique[j])
        # 对目前 i 个像素矩阵里的唯一值再取唯一值
        colorDict = sorted(set(colorDict))
        # 若唯一值数目等于总类数(包括背景)ClassNum，停止遍历剩余的图像
        if (len(colorDict) == classNum):
            break
    # 存储颜色的 BGR 字典，用于预测时的渲染结果
    colorDict_BGR = []
    for k in range(len(colorDict)):
        # 对没有达到九位数字的结果进行左边补零(eg:5,201,111->005,201,111)
        color = str(colorDict[k]).rjust(9, '0')
        # 前 3 位 B,中 3 位 G,后 3 位 R
        color_BGR = [int(color[0: 3]), int(color[3: 6]), int(color[6: 9])]
        colorDict_BGR.append(color_BGR)
    # 转为 numpy 格式
    colorDict_BGR = np.array(colorDict_BGR)
    # 存储颜色的 GRAY 字典，用于预处理时的 onehot 编码
    colorDict_GRAY = colorDict_BGR.reshape((colorDict_BGR.shape[0], 1, colorDict_BGR.shape[1])
).astype(np.uint8)
    colorDict_GRAY = cv2.cvtColor(colorDict_GRAY, cv2.COLOR_BGR2GRAY)
    return colorDict_BGR, colorDict_GRAY
def ConfusionMatrix(numClass, imgPredict, Label):
    # 返回混淆矩阵
    mask = (Label >= 0) & (Label < numClass)
    label = numClass * Label[mask] + imgPredict[mask]
    count = np.bincount(label, minlength=numClass ** 2)
    confusionMatrix = count.reshape(numClass, numClass)
    return confusionMatrix
```

```
def OverallAccuracy(confusionMatrix):
    # 返回所有类的整体像素精度 OA
    # acc = (TP + TN) / (TP + TN + FP + TN)
    OA = np.diag(confusionMatrix).sum() / confusionMatrix.sum()
    return OA
def Precision(confusionMatrix):
    # 返回所有类别的精确率 precision
    precision = np.diag(confusionMatrix) / confusionMatrix.sum(axis=1)
    return precision
def Recall(confusionMatrix):
    # 返回所有类别的召回率 recall
    recall = np.diag(confusionMatrix) / confusionMatrix.sum(axis=0)
    return recall
def F1Score(confusionMatrix):
    precision = np.diag(confusionMatrix) / confusionMatrix.sum(axis=1)
    recall = np.diag(confusionMatrix) / confusionMatrix.sum(axis=0)
    f1score = 2 * precision * recall / (precision + recall)
    return f1score
def IntersectionOverUnion(confusionMatrix):
    # 返回交并比 IoU
    intersection = np.diag(confusionMatrix)
    union = np.sum(confusionMatrix, axis=1) + np.sum(confusionMatrix, axis=0) - np.diag(confusionM
atrix)
    IoU = intersection / union
    return IoU
def MeanIntersectionOverUnion(confusionMatrix):
    # 返回平均交并比 mIoU
    intersection = np.diag(confusionMatrix)
    union = np.sum(confusionMatrix, axis=1) + np.sum(confusionMatrix, axis=0) - np.diag(confusionM
atrix)
    IoU = intersection / union
    mIoU = np.nanmean(IoU)
    return mIoU
def Frequency_Weighted_Intersection_over_Union(confusionMatrix):
    # 返回频权交并比 FWIoU
    freq = np.sum(confusionMatrix, axis=1) / np.sum(confusionMatrix)
    iu = np.diag(confusionMatrix) / (
            np.sum(confusionMatrix, axis=1) +
            np.sum(confusionMatrix, axis=0) -
            np.diag(confusionMatrix))
    FWIoU = (freq[freq > 0] * iu[freq > 0]).sum()
    return FWIoU
########################################################################
# 标签图像文件夹
LabelPath = r"Data\label\yanz"#pic\label
# 预测图像文件夹
PredictPath = r"test_result"
# 类别数目(包括背景)
classNum = 2
########################################################################
# 获取类别颜色字典
colorDict_BGR, colorDict_GRAY = color_dict(LabelPath, classNum)
# 获取文件夹内所有图像
labelList = os.listdir(LabelPath)
PredictList = os.listdir(PredictPath)
# 读取第一个图像，后面要用到它的 shape
Label0 = cv2.imread(LabelPath + "//" + labelList[0], 0)
```

```
# 图像数目
label_num = len(labelList)
# 把所有图像放在一个数组里
label_all = np.zeros((label_num,) + Label0.shape, np.uint8)
predict_all = np.zeros((label_num,) + Label0.shape, np.uint8)
for i in range(label_num):
    Label = cv2.imread(LabelPath + "//" + labelList[i])
    Label = cv2.cvtColor(Label, cv2.COLOR_BGR2GRAY)
    label_all[i] = Label
    Predict = cv2.imread(PredictPath + "//" + PredictList[i])
    Predict = cv2.cvtColor(Predict, cv2.COLOR_BGR2GRAY)
    predict_all[i] = Predict
# 把颜色映射为 0,1,2,3...
for i in range(colorDict_GRAY.shape[0]):
    label_all[label_all == colorDict_GRAY[i][0]] = i
    predict_all[predict_all == colorDict_GRAY[i][0]] = i
# 拉直成一维
label_all = label_all.flatten()
predict_all = predict_all.flatten()
# 计算混淆矩阵及各精度参数
confusionMatrix = ConfusionMatrix(classNum, predict_all, label_all)
precision = Precision(confusionMatrix)
recall = Recall(confusionMatrix)
OA = OverallAccuracy(confusionMatrix)
IoU = IntersectionOverUnion(confusionMatrix)
FWIOU = Frequency_Weighted_Intersection_over_Union(confusionMatrix)
mIOU = MeanIntersectionOverUnion(confusionMatrix)
f1ccore = F1Score(confusionMatrix)
for i in range(colorDict_BGR.shape[0]):
        import webcolors
        rgb = colorDict_BGR[i]
        rgb[0], rgb[2] = rgb[2], rgb[0]
        print(webcolors.rgb_to_name(rgb), end="  ")
print("")
print("混淆矩阵:")
print(confusionMatrix)
print("精确度:")
print(precision)
print("召回率:")
print(recall)
print("F1-Score:")
print(f1ccore)
print("整体精度:")
print(OA)
print("IoU:")
print(IoU)
print("mIoU:")
print(mIOU)
print("FWIoU:")
print(FWIOU)
a=0
sum=0
r=[]
lie=[]
for i in range(2):
    for j in range(2):
        if j==1:
```

```
            r.append(confusionMatrix[i][j]+confusionMatrix[i][j-1])
        if i ==1:
            lie.append(confusionMatrix[i][j]+confusionMatrix[i-1][j])
        if i == j :
            a=a+confusionMatrix[i][j]
# ***********计算 kappa 值
a=0
sum=0
r=[]
lie=[]
for i in range(2):
    for j in range(2):
        if j==1:
            r.append(confusionMatrix[i][j]+confusionMatrix[i][j-1])
        if i ==1:
            lie.append(confusionMatrix[i][j]+confusionMatrix[i-1][j])
        sum=sum+confusionMatrix[i][j]
        if i == j :
            a=a+confusionMatrix[i][j]
sum=r[0]+r[1]
po=a/sum
dd=(r[0]*lie[0]+r[1]*lie[1])
qq=sum*sum
pe=dd/qq
k=(po-pe)/(1-pe)
print("kappa 值为:",k)
```