队伍编号	1248
赛道	B

基于 U-Net 卷积神经网络的遥感图像耕地地块分割与提取研究

摘 要

准确地分划耕地分布及计算耕地面积能够为国家决策部门提供重要支撑。通过计算机识别卫星遥感影像中的耕地面积能够加速耕地信息的提取，减少人力、财力消耗。但是遥感影像中非耕地的区域环境复杂,且其分布位置、大小形状、颜色等方面存在许多差异,导致其信息提取存在一定困难。

针对问题一，我们首先设计相应算法，利用 python 编程对 8 张标签集进行计算，得到相应耕地面积比例。

针对问题二，我们通过题目中已知条件和要求，首先对数据集 8 张图像和标签进行处理，经过切割、数据扩增操作，生成包含训练集和包含测试集。通过 Python 编程、利用 U-Net 卷积网络成功建立体积小、训练速度快的预测模型。本文所述的模型经过 100 轮的训练，共训练 36000 组图像，准确率达到 98.49%，损失率为 0.037，可实现精确度较高遥感图像地块分割与提取。本文利用上述模型，根据 Test1，Test2 图像，预测耕地地块分布标签图，并利用上述程序计算耕地面积占比，完善问题一。

在建立完整模型的基础上，我们优化预测数据上提出了创新点：参考“集成学习”的思想，以“少数服从多数”的原则对同一图片的多次预测进行提炼，得到较为准确的识别结果；采用“多图平均法”降低预测结果的噪声干扰，简化阈值的设置和二值化过程，能够将耕地区域的“斑点”基本完全消除。

本文所述的算法、模型，在遥感图像空间分辨率为 2m 的条件下，测试中可作出误差在 5%范围内的对耕地的精确分割，可为耕地遥感制图提供借鉴与参考。

针对问题三，我们将结合我国耕地分布时间、空间复杂性提出创新思路。首先，我们提出以同一地块不同时间（如，春夏秋冬）的遥感图像做输入数据集（每一数据包含时间和空间特征），以此提高模型预测的精确性。另外，我们提出采用人机配合的方式进行耕地标记，既可以减少人工工作量，又能得到高质量的结果。除此之外，通过资料查找和分析思考，我们提出将应用于分割生物组织的插值法[6]，以及多尺度特征提取方法[7][8]用来优化耕地预测，提高耕地地块的分割效率和准确性。

关键词： 遥感图像　图像分割　深度学习　U-Net　卷积神经网络　优化模型

目录

一、问题重述.....1
二、模型假设.....1
三、符号说明.....1
四、问题分析.....2
五、问题求解.....2
5.1 图像预处理.....2
5.1.1 标准化.....2
5.1.2 可视化.....3
5.1.3 切割.....4
5.2 问题一求解——耕地比例计算.....4
5.2.1 求解过程.....4
5.2.2 结果.....5
5.3 问题二求解——模型预测耕地标签图.....5
5.3.1 数据准备.....5
5.3.2 模型建立.....6
5.3.3 模型训练.....7
5.3.4 模型测试.....9
5.3.5 模型优化.....10
5.3.6 Test1、Test2 图片耕地标签图制作.....13
5.3.7 完善问题一.....14
5.4 问题三的求解——创新思路.....14
5.4.1 增加时间维度特征训练.....14
5.4.2 人机配合.....14
5.4.3 插值法优化模型.....15
5.4.4 多尺度特征提取.....15
六、模型评价.....15
6.1 优点.....15
6.2 缺点.....15
七、总结.....16
八、参考文献.....17
附录 1 程序.....18
附录 2 多图平均二值法优化前后对比图.....45

一、问题重述

耕地的数量和质量是保持农业可持续发展的关键，利用卫星遥感影像可以识别并提取耕地，并对耕地进行遥感制图，准确的耕地分布能够为国家决策部门提供重要支撑。目前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，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。

资源三号（ZY-3）卫星是中国第一颗自主的民用高分辨率立体测绘卫星，通过立体观测，可以测制 1：5 万比例尺地形图，为国土资源、农业、林业等领域提供服务，资源三号填补了中国立体测图这一领域的空白。图片来源于资源三号卫星获取的遥感图像数据，空间分辨率为 2 米，光谱为可见光波段（红，绿，蓝）。

题目提供 8 幅图像和相应耕地标签，用于参赛者模型训练和测试，图像为 tif 格式。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。另提供 2 幅图像作为测试实例。

问题：

(1)计算 10 幅图中耕地在各图像中所占比例。

(2)从给定的 2 幅测试图像(Test1.tif、Test2.tif)中提取出耕地。

(3)针对我国土地辽阔，地貌复杂的特点，提出创新性思路快速、精准的识别出田块。

二、模型假设

(1)假设已知专业人工标记标签图足够准确划分耕地与非耕地;

(2)假设在神经网络预测中，局部图像特征可代表整体特征;

(3)假设耕地面积、像素点误差可以作为耕地标签预测模型的评价标准;

(4)假设灰度、形状大小、分布体现耕地特性。

三、符号说明

此处仅列出部分通用符号，个别符号使用时会在下文说明。

符号	说明
$x_{\max}$	最大值
$x_{\min}$	最小值
p	比值

四、问题分析

[问题 1]

在问题一中，题目要求我们计算 10 幅图中耕地在各图像中所占比例，其中 8 幅图像提供标注图像，标注图像为专业遥感解译人员人工解译，可作为参考。标注图像是二值标签图，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。利用上述特征，我们将标签图转化为二维矩阵，通过计算值为 1 的像素点占全部像素点个数的比例，进行运算求解。由于最后 2 张测试图片未提供标签，我们选择先完成问题二，建立模型，得到测试图像的标签结果后，再进行耕地面积占比的运算。

[问题 2]

在问题二中，题目要求我们从给定的 2 幅测试图像(Test1.tif、Test2.tif)中提取出耕地，制作耕地标签图。运用所给数据集，运用合适的网络进行训练得到图像分割模型，用训练好的模型来分割测试图像的耕地和非耕地区域，得到较为精确的标签标记结果，田块间的边界清晰。我们需要利用已知标签的 8 幅遥感图像，生成训练集和测试集，建立模型，训练模型，利用上述模型，由预测 Test1，Test2 图像中耕地地块分布，形成标签图，同时利用问题一程序算法计算 Test1，Test2 图像中耕地面积占比，完善问题一。难点在于扩增数据集和精确地预测耕地标签图，保证田块间的边界清晰准确。

[问题 3]

在问题三中，题目要求我们针对我国土地辽阔，地貌复杂的特点，提出创新性思路快速、精准的识别出田块。我们需要从我国田块分布的时间和空间复杂度出发，提出合理的思路以快速精准识别田块。难点在于需要克服我国地广、地貌复杂的特点，完成大量田块的精准识别。

五、问题求解

5.1 图像预处理

5.1.1 标准化

在众多标准化方法中,本文选用了 min-max 标准化(Min-Max Normalization)方法，也称为离差标准化，是对原始数据的线性变换，使结果值映射到[0 - 1]之间。

转换函数如下:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

其中 x,x’为转化前后的像素值; x_{max}， x_{min} 为样本数据的最大，最小像素值。

本题中图片像素值均为正,故 x_{min} 将设为 0。使用的归一化转换函数如下:

$$x' = \frac{x}{x_{\max}}$$

5.1.2 可视化

由于题目提供 tif 图片文件不适宜之间观察，而数据特点分析与分类需要以可视化为前提。将训练、测试的图片可视化，在建立、测试模型时能更容易判断模型的合理性和正确性。同时，预测结果更加直观。

本文采用将归一化的数据扩大 255 倍的方法进行可视化，可视化的数据转换函数如下：

$$x'' = x' \times 255$$

其中 x’,x”为转化前后的像素值。生成的图片像素值范围为[0,255]。

可视化结果中，图片中各部分的深浅关系由肉眼观察，耕地非耕地间边界清晰。同时，各数据集整体灰度差距不大，符合训练模型要求。

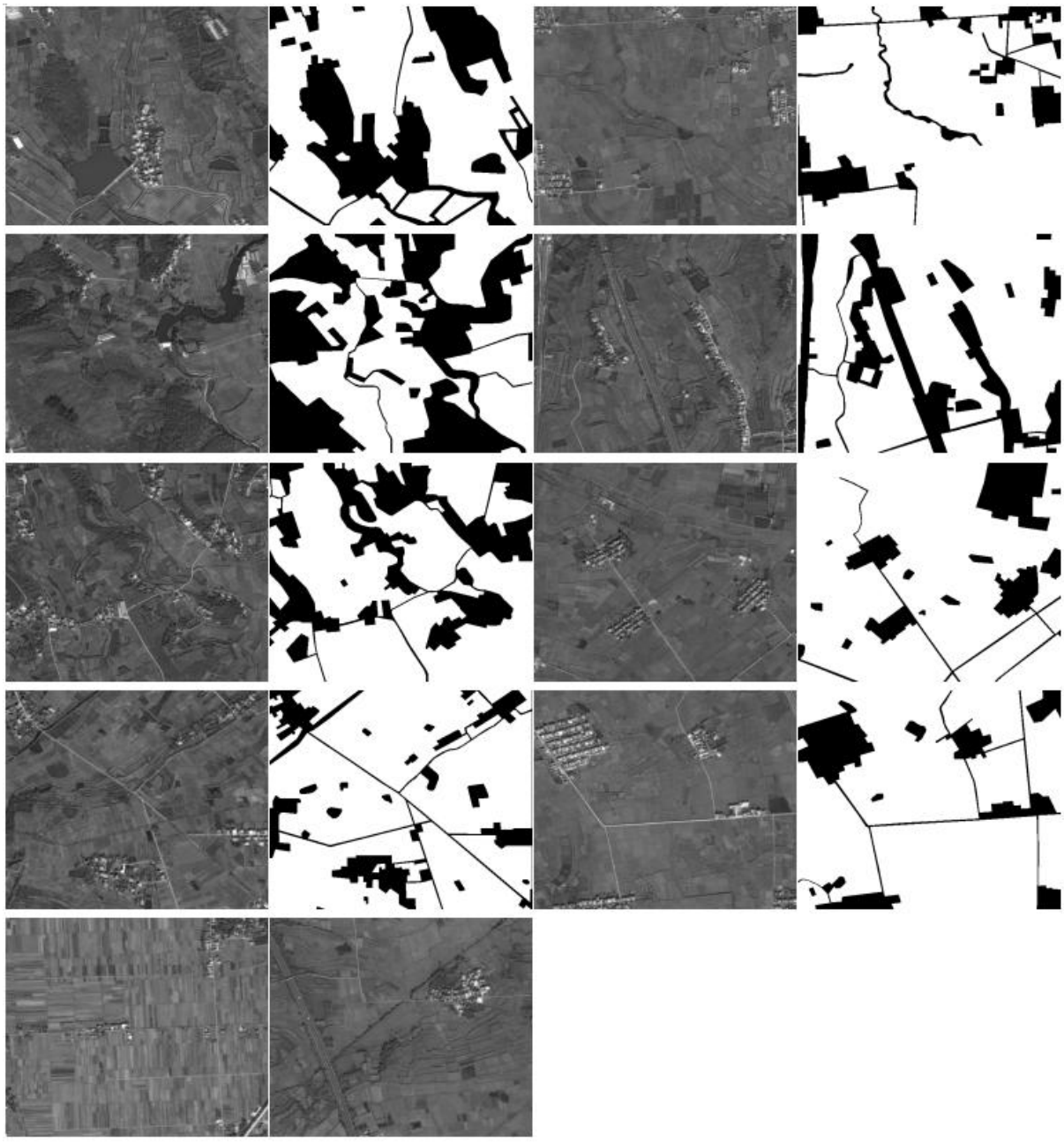


图 1 可视化结果

从左向右从上到下依次为 Data1-8，Test1-2

5.1.3 切割

【训练集准备】

将像素为 600×500 原图切割成大小为 256×256 像素的图片，每次移动横向步长 43，纵向步长 61，将一张 600×500 的图片可分割为 5*9=45 张图片，即数据集中 8 张原图被切割成 8*45=360 张图片，扩大了数据集，同时较好的保留数据集原本特征。将经过归一化和切割的图片编码（image 和 mask 编号对应）后保存为 png 格式的图片。

【测试集准备】

测试集采用不同于训练集的切割方式对数据进行处理，将一张 600×500 的图片可分割为 2*3=6 张图片，即数据集中 8 张原图被切割成 8*6=48 张图片。将经过归一化和切割的图片编码（image 和 mask 编号对应）后保存为 png 格式的图片。

图 2 图像切割结果

5.2 问题一求解——耕地比例计算

5.2.1 求解过程

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。利用上述特征，我们用 python 编程，将标签图转化为二维矩阵，通过计算值为 1 的像素点占全部像素点个数的比例，进行运算求解。由于最后 2 张测试图片未提供标签，我们选择先完成问题二，建立模型，得到测试图像的标签结果后，再进行耕地面积占比的运算。

计算公式为

$$\mathrm{n}=\sum_{0}^{j=499}\left(\sum_{0}^{i=599}x_{ij}\right)$$

$$\mathrm{p}=\frac{n}{500\times 600}$$

其中 n 为全部像素点和，也就是值为 1 的像素点数目；

500x600 是图片像素点总数。

5.2.2 结果

数据编号	耕地所占比例
Data 1	0.83292667
Data 2	0.84226667
Data 3	0.70042000
Data 4	0.74683667
Data 5	0.55335333
Data 6	0.85561667
Data 7	0.68023667
Data 8	0.83893667
Test 1	0.89282667
Test 2	0.84388000

表 1 耕地在各图像中所占比例

5.3 问题二求解——模型的建立、图像地块分割、提取

5.3.1 数据准备

【训练集】

采用 python 的 keras 库的数据增广函数，编写训练数据生产函数，对图像预处理中切割产生的 360 张训练集进行批量图像增强处理。每批产生 4 组图片（遥感图和标签图为 1 组），输入至网络进行训练，每个周期训练 90 批，同时每组图片保存于指定位置以供查阅。保证了训练数据的随机性和特征的全面性。

- 数据增强操作（每组遥感图和标签图共用 seed，执行操作相同）

(1)在[0,10°]内随机旋转一个角度;

(2)在[0, 0.1]内水平位置平移和上下位置平移一个距离;

(3)错切变换,让点 x 坐标(或者 y 坐标)保持不变,而对应的 y 坐标(或者 x 坐标)则按 0.5 的比例发生平移，且平移的大小和该点到 x 轴(或 y 轴)的垂直距离成正比;

(4)在长或宽方向放大。参数控制图片同时在长宽两个方向进行同等程度放缩操作;

(5)随机执行水平翻转和上下翻转操作;

(6)图片的空缺用 reflect 模式进行补全，符合耕地分布正常状态。

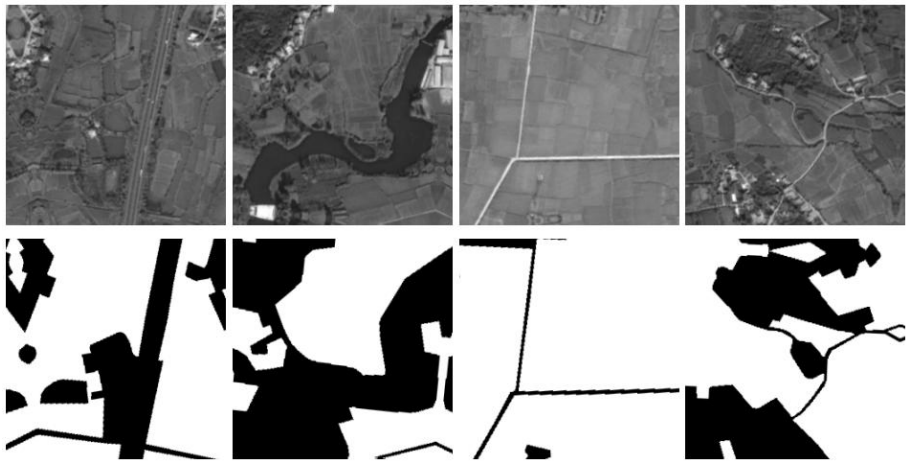


图 3 图像增广结果

【测试集】

直接采用图像预处理中切割产生的 48 张测试集图片。

5.3.2 模型建立

(1)U-Net 神经网络模型

处理图像的边缘细节检测分割时，全卷积神经网络是常用手段。其中，U-Net 由于其结构体量的精细、对线状图像特征的准确提取而广泛应用于细节处地物分割当中，如遥感影像中道路检测、医学影像中血管提取等领域。

U-Net 网络结构清晰，呈现的是收紧-膨胀结构。收紧阶段主要通过池化逐步减少特征维度和参数数量，在膨胀阶段通过特征数量的拼接对于图像的细节和维度进行恢复，形成了 U 形结构，如图所示。[1]

U-Net 最大的优点在于在小样本数据集上具有较快的收敛速度和较好的分割效果，对于本题原始数据量较小的情况十分适合。故本文采用 U-Net 网络结构设计模型提取图片信息。

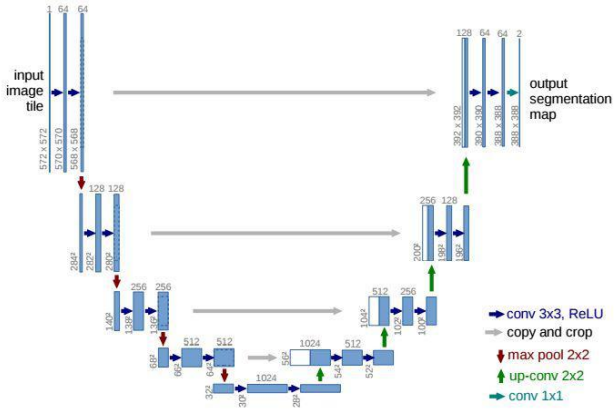


图 4 U-Net 卷积示例图[1]

(2)模型结构

本文运用的 U-Net 神经网络主要包含 23 个 3 x 3 的卷积层，4 个 2 x 2 的下采样层，4 个 2 x 2 的上采样层，采用 4 次 skip connection 对深层网络和浅层网络产生的特征图进行连接，呈现的是收紧-膨胀结构。

U-Net 单次下采样中包括 2 个的 3 x 3 的的卷积层，一个 2 x 2 的池化层，激活函数为 Relu（activation = 'relu'），补偿采用缺少补零的方式（padding = 'same'），权值初始化参数模式设定为'he_normal'（kernel_initializer = 'he_normal'）[4]。

在 U 型结构中，利用 Concatente 函数将深层网络和浅层网络的特征信息进行拼接，保持图像维度不发生改变。[1]

(3)损失函数

经运行尝试，本文采用交叉熵代价函数(Binary Crossentropy)计算损失，保存最小损失模型用于预测。交叉熵代价函数相比于 sigmoid 型函数不易发生饱和，梯度更新较快。其次，同时，利用交叉熵作为损失函数进行梯度下降计算时可一定程度上梯度弥散问题，保证学习速率适宜。

(4)Adam 优化器

本文采用 Adam 优化器，依照经验和运行结果，选取学习率为‘1e-4’。

Adam 优化器,基于随机梯度下降算法(SGD),结合自适应优化(AdaGrad)和 RMSProp 两种算法的优点，对梯度均值和梯度的未中心化方差综合考虑，计算更新步长。

其主要优点有：

1. 实现简单，计算高效，对内存需求少；
2. 参数的更新不受梯度的伸缩变换影响；
3. 超参数具有很好的解释性，且通常无需调整或仅需很少的微调；
4. 更新的步长能够被限制在大致的范围内（初始学习率）；
5. 能自然地实现步长退火过程（自动调整学习率）；
6. 很适合应用于大规模的数据及参数的场景；
7. 适用于不稳定目标函数；
8. 适用于梯度稀疏或梯度存在很大噪声的问题；[2]

综合 Adam 在很多情况下算作默认工作性能比较优秀的优化器。对于耕地边界，Adam 优化器可改善边界模糊问题，提高模型准确度。通过学习率控制可使得模型不易过拟合，acc 稳步提高。除此之外，本文运行主要在内存 4G 或 8G 的笔记本电脑上实现，无法提供过大内存。结合实际情况考虑，Adam 优化器较为适用于本文问题。

5.3.3 模型训练

(1) 参数选取

【Batch_size 批尺寸】

Batch 的选择决定了梯度下降的方向。虽然由全数据集（Full Batch Learning）确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向，但每次载入全数据集的方法不易完成。同时，在线学习（Online Learning） 会导致样本的梯度方向频繁修正，训练难以收敛。因此，我们选用批梯度下降法（Mini-batches Learning）。[3]

Batch_Size	5000	2000	1000	500	256	100	50	20	10	5	2	1
Total Epoches	200	200	200	200	200	200	200	200	200	200	200	200
Total Iterations	1999	4999	9999	19999	38999	99999	199999	499999	999999	1999999	cannot converge	
Time of 200 Epoches	1	1.068	1.16	1.38	1.75	3.016	5.027	8.513	13.773	24.055		
Achieve 0.99 Accuracy at Epoch	-	-	135	78	41	45	24	9	9	-		
Time of Achieve 0.99 Accuracy	-	-	2.12	1.48	1	1.874	1.7	1.082	1.729	-		
Best Validation Score	0.015	0.011	0.01	0.01	0.01	0.009	0.0098	0.0084	0.01	0.032		
Best Score Achieved at Epoch	182	170	198	100	93	111	38	49	51	17		
Best Test Score	0.014	0.01	0.01	0.01	0.01	0.008	0.0083	0.0088	0.008	0.0262		
Final Test Error (200 epoches)	0.0134	0.01	0.01	0.01	0.01	0.009	0.0082	0.0088	0.008	0.0662		

图 5 batch_size 对训练的影响[3]

参考上图，我们首先选取 batch_size = 2 和 batch_size = 4 进行尝试，最终结果呈现出 batch_size = 4 效果较优。

【steps_per_epoch 每次迭代批数】

这个参数的选择我们根据训练及大小（360）和批尺寸（4），选择 90 作为每次迭代批数，保证训练数据利用的全面性。

【epochs 迭代次数】

迭代次数的选取我们根据经验首先选取 50、100 和 200 进行尝试，结果显示 50 次未饱和，200 次有过饱和现象，因此本文选取 epochs = 100。

(2) 训练结果

本文所述的模型经过 100 轮的训练，共训练 36000 组图像，最终模型准确率达 98.49%，损失率为 0.037。利用 ModelCheckpoint 函数将模型文件导出保存。

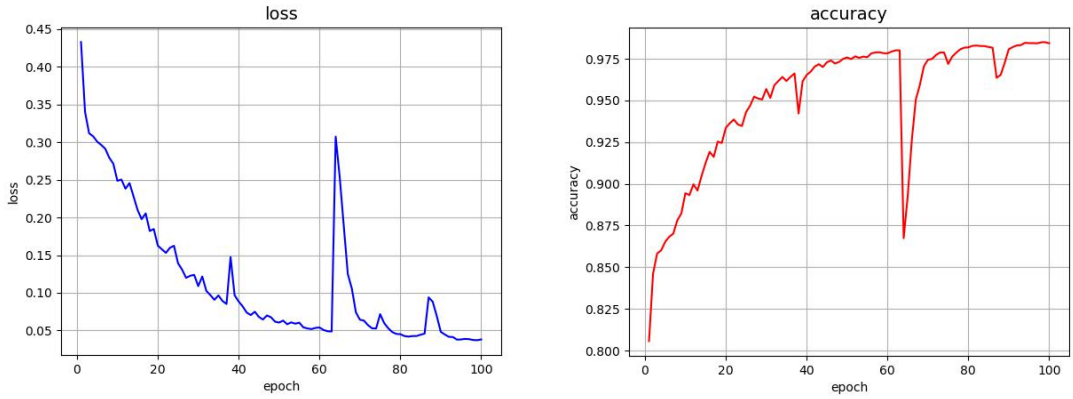


图 6 训练损失(loss)和准确率(accuracy)与迭代次数(epoch)的关系

训练中训练损失(loss)和准确率(accuracy)与迭代次数(epoch)的关系如图 6 所示，训练前期变化较为稳定，后期(从 63 次开始)产生突变。经分析可能原因如下：

1 训练集生成器每次产生图片不同，可能在某个 epoch 与之前相差较大，模型还未学到其特征；

2 出现过拟合现象；

通过查找图片生成器的运行记录，我们发现后期突变附近图像‘白化’特征较明显（如图 7），猜测原因 1 可能性较大。但由于时间关系，且模型最终准确率达到要求，对此问题未做过多讨论。

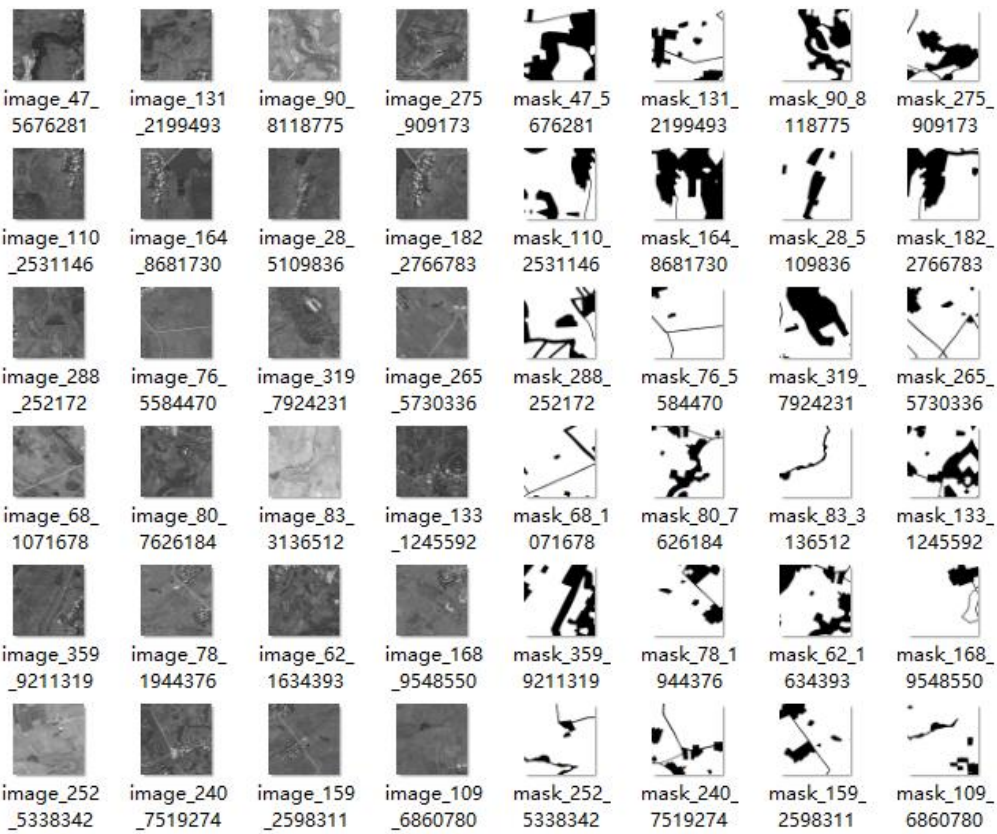


图 7 后期突变(epoch = 63)附近部分训练图片

5.3.4 模型测试

得到完整模型后，我们利用模型对切割好的测试集进行预测，并对模型进行优化，比较优化前后结果准确性，如表 2。

相对误差色谱对照表			
误差x绝对值	正差距	负差距	图片标注
0.01<=x<0.05			x>=0.02
0.005<=x<0.01			0.01<=x<0.02
x<0.005			x<0.01

48图test预测结果相对误差			
图片名称	标签	测试结果	相对误差
0	0.81846619	0.81433105	-0.00505230
1	0.89433289	0.89245605	-0.00209859
2	0.68728638	0.67791748	-0.01363173
3	0.64765930	0.64164734	-0.00928260
4	0.47662354	0.46109009	-0.03259061
5	0.89802551	0.89833069	0.00033983
6	0.73742676	0.73919678	0.00240027
7	0.70938110	0.70834351	-0.00146267
8	0.90330505	0.89814758	-0.00570956
9	0.88595581	0.88504028	-0.00103338
10	0.64797974	0.63972473	-0.01273961
11	0.84605408	0.84005737	-0.00708786
12	0.50753784	0.49801636	-0.01876014
13	0.89022827	0.88836670	-0.00209112
14	0.72434998	0.71899414	-0.00739399
15	0.88604736	0.88455200	-0.00168768
16	0.83790588	0.83979797	0.00225812
17	0.66554260	0.66168213	-0.00580049
18	0.63362122	0.62907410	-0.00717640
19	0.83793640	0.83448792	-0.00411544
20	0.51921082	0.51539612	-0.00734711
21	0.74530029	0.73970032	-0.00751371
22	0.64778137	0.63806152	-0.01500483
23	0.82815552	0.84423828	0.01941998
24	0.81906128	0.81149292	-0.00924029
25	0.89059448	0.89570618	0.00573965
26	0.75799561	0.74565125	-0.01628553
27	0.77224731	0.77235413	0.00013832
28	0.51884460	0.51068115	-0.01573390
29	0.77928162	0.78271484	0.00440562
30	0.60072327	0.59983826	-0.00147324
31	0.87872314	0.88430786	0.00635549
32	0.80784607	0.80561829	-0.00275768
33	0.90097046	0.89881897	-0.00238797
34	0.76953125	0.76567078	-0.00501665
35	0.69288635	0.68693542	-0.00858861
36	0.60630798	0.59558105	-0.01769221
37	0.93629456	0.93704224	0.00079855
38	0.59843445	0.59379578	-0.00775134
39	0.95690918	0.95549011	-0.00148297
40	0.86485291	0.86495972	0.00012350
41	0.82998657	0.82870483	-0.00154429
42	0.77384949	0.77052307	-0.00429854
43	0.73242188	0.72541809	-0.00956251
44	0.56666565	0.55035400	-0.02878532
45	0.95986938	0.96141052	0.00160557
46	0.75393677	0.74586487	-0.01070634
47	0.86323547	0.86224365	-0.00114896

表 2 48 图模型测试面积比例预测结果及相对误差

分析表 2 数据，我们发现模型能较精确的分割耕地的边界，对于植被、住宅区、池塘有较高的分割精确度，边界清晰且能连续分割整条道路，但仍存在以下问题：

(1)问题较大图片主要来自 Data3 和 Data5；

(2)问题图片含有大量非耕地，以大面积森林为主；

(3)部分问题图片存在‘弱标签’问题，即没有细致表明田块间的分隔，如图 8-1；

(4)部分图片呈现中心残缺问题，即错误的将非耕地内部划分成小片耕地，如图 8-2；

(5)噪声较多，黑斑，白斑均有出现，以黑斑为主；

(6)水平、竖直道路判断准确性不一。

图 8-1 标签弱化问题

左至右依次为遥感图、预测标签、人工标签

图 8-2 标签弱化问题
左至右依次为遥感图、预测标签、人工标签

针对上述问题我们在(4)中对模型进行优化，得到了较优的测试结果。

5.3.5 模型优化

(1)集成学习

集成学习通过将多个学习器进行结合，常可获得比单一学习器显著优越的泛化性能，对“弱学习器”(weak learner) 尤为明显。其潜在的思想是即便某一个弱分类器得到了错误的预测，其他的弱分类器也可以将错误纠正。[5]

本题在仅使用 U-Net 单一神经网络的前提下，借鉴集成学习的思想，通过旋转、镜像将模型对同一 test 图片进行多次判断，不同情境下的判断，具有一定的独立性。设计算法分析结果，将同一 test 图片的不同结果融合，生成标签图，使准确率大大提升。参考测试结果，我们对模型进行优化，比较优化前后结果准确性。以下说明优化过程。

(2)优化过程

【测试集处理】

结合卷积神经网络的特点，我们发现模型对不同方向图片预测结果不同，且对不同情境下的判断，具有一定的独立性。因此，本文改变了算法中直接输入预测图像的思路。将测试集图片经过预处理，每张图切分成 2*3=6 张 256*256 的子图后，分别进行：

1 旋转 90°，180°，270°；

2 将旋转后的图片分别关于 x 轴、y 轴镜像；

经过上述步骤，每张子图（包括仅切割未进行上述操作的 3 张子图）共生成 6*12=72 张图片做为新的测试集。

【多图平均二值法】

普通的均值思想得出的结果只能保证优于最次，却不能做到优于最优，本文中提出的‘多图均值’思想，在一定条件下，可通过二值化达到‘1+1>2’的效果。

本方法采用将归一化后值（0,1 间）处理，使得标签图中黑色非耕地（-0.5,0 间）和白色耕地（0，0.5 间）分居 0 值两侧，再将各点求和取平均。最后，恢复至原状态（0,1 间）。进行二值化，得到标签图。

平均的是非二值图，黑白部分在不同图片求和时相互抵消，强度随深浅而定。值的微波动不会影响二值化判断。因此，可做到类似“集成学习”模型取比例最高分类的效果。同时，该方法减少了训练模型所消耗的时间，明显减少了错误点数目。降低了损失，提高了准确率。结合人工技术性筛选更能提高最终结果准确性。

【结果融合】

将处理好的数据（test1、test2 各扩增了 72 张图片）和作为输入进入训练好的模型，每张测试图得到的 72 个预测结果。由于输入的图片方向、大小与原图不一致，故得到的结果标签图不能直接相加。将结果的标签图做预处理的逆操作，使每张图恢复原始图片的方向。

利用上述多图均值法，将同一张切割后的原图对应的多（12）张标签图处理，求和，得到复合多重判断的“整合”图片，再平均，复原。得到融合后的图片。

$$y_1 = x - 0.5$$

$$y_1 = \frac{\sum_{i=0}^{11} y_i}{12}$$

$$y = y_1 + 0.5$$

其中 y、yi 分别为“整合”图片的像素值和第 i 张子图的像素值，x 为经过归一化的图片。最后将 y 图再次归一化并二值化输出结果即可。

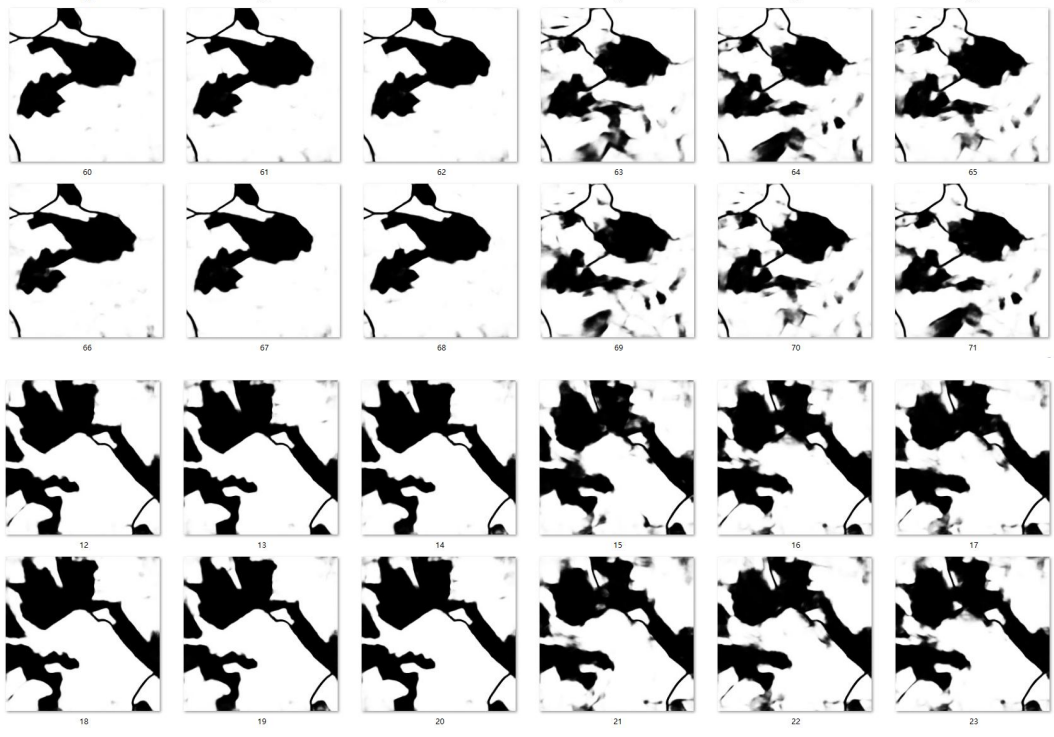
$$y = y / y_{\max} \qquad \begin{matrix} y[y >= 0.5] = 1 \\ y[y < 0.5] = 0 \end{matrix}$$

其中 $y_{\max}$ 为像素值最值，若生成 png 可视化格式，则需要将上述公式中的‘1’替换为‘255’。

(3) 优化结果

优化后，本文对测试集再次进行测试，发现模型对 90 度和 270 度操作的测试集（图右）表现明显劣于 0 度和 180 度的旋转和镜像产生的 6 张测试图（图左），不适宜参与平均，如图 9。同时预测用 Test2 图片也表现出相同的效果，Test1 不太明显。分析原因可能是模型学习到了训练集的光线特征，与本文求解关系不紧密，不做过多说明。

【注】12 图选 6 图的方式依照实际数据而定。



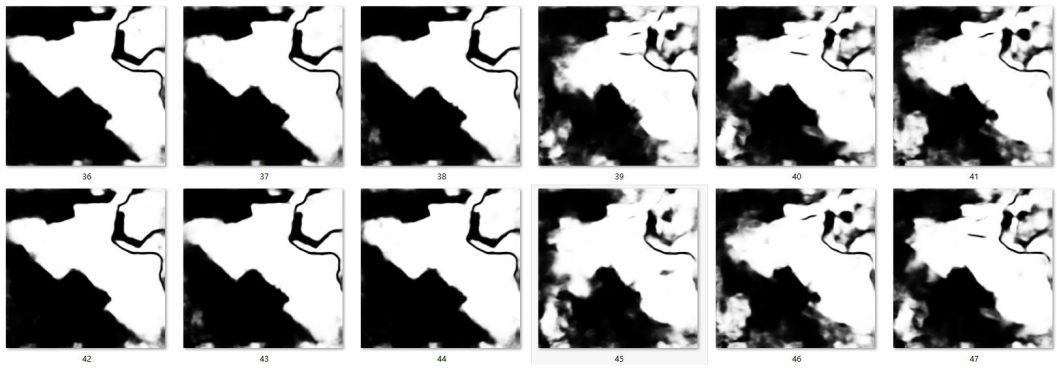


图 9-1 训练集

上至下，左至右依次为(0, 0x, 0y) (90, 90x, 90y) (180, 180x, 180y) (270, 270x, 270y)
（旋转角度镜像方向）

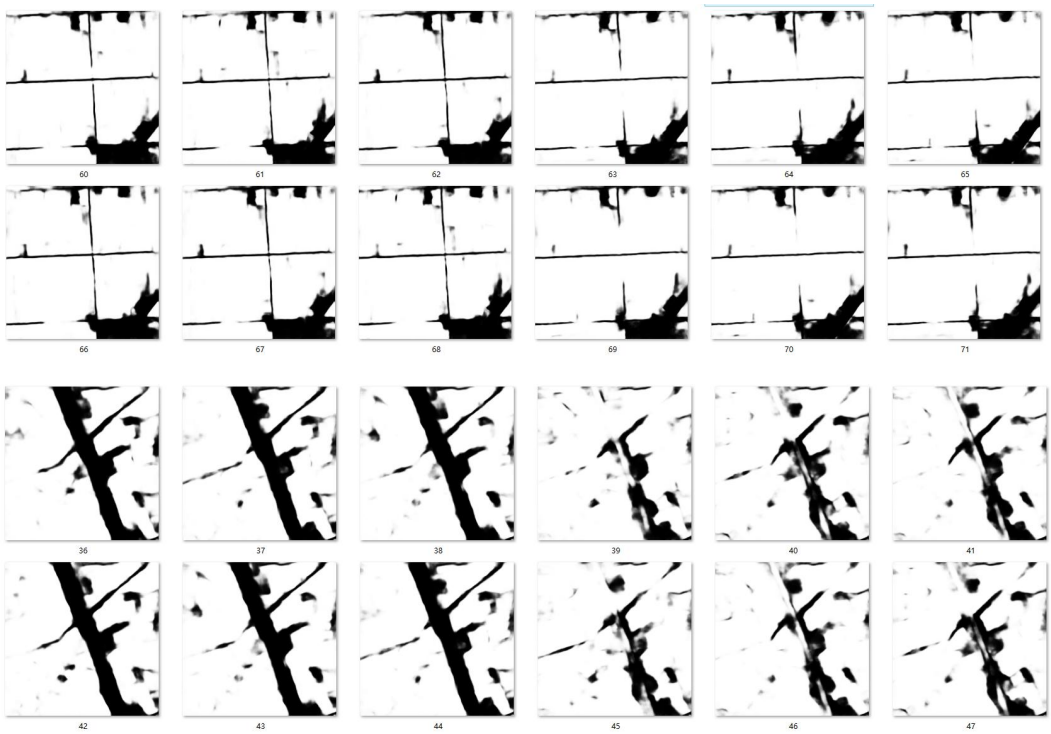


图 9-1 Test

上至下，左至右依次为(0, 0x, 0y) (90, 90x, 90y) (180, 180x, 180y) (270, 270x, 270y)
（旋转角度镜像方向）

{图 9}通过观察可以看到，其中六张预测结果标签图显著优于另外六张。对于预测结果的差异，我们猜测，由于经过预处理后的测试图片的光线与训练时处理的图片光线有差异（光强、光线方向等），造成模型不能在某些情况下泛化。但预处理后的测试图片中的六个角度的子图，较符合模型对于输入数据的要求，其预测效果会优于其他角度的子图。

因此，考虑图像特点和模型特点，本文使用“6 图平均法”，即仅保留 0 度和 180 度的旋转和镜像产生的 6 张图融合，再次进行测试，测试结果显示，模型表现良好，如图 10。

优化前后测试结果评价						
	错误像素点个数		准确率		优化提升	
	优化前	优化后	优化前	优化后	误差点	准确率(*10^-4)
Data1	5482	4415	0.98172667	0.98528333	-1067	35.57
Data2	3006	2423	0.98998000	0.99192333	-583	19.43
Data3	4950	4135	0.98349667	0.98621667	-815	27.20
Data4	4967	4088	0.98344333	0.98637333	-879	29.30
Data5	5991	5090	0.98003000	0.98303333	-901	30.03
Data6	3451	2842	0.98849667	0.99052667	-609	20.30
Data7	5120	3809	0.98293000	0.98730333	-1311	43.73
Data8	3824	2585	0.98725333	0.99138000	-1239	41.27

图 10-1 优化前后结果与人工标签像素点差距

{图 10-1}由上表可以得出，本文上述优化方法可以使预测结果质量明显提升，表现为误差像素点个数减少。

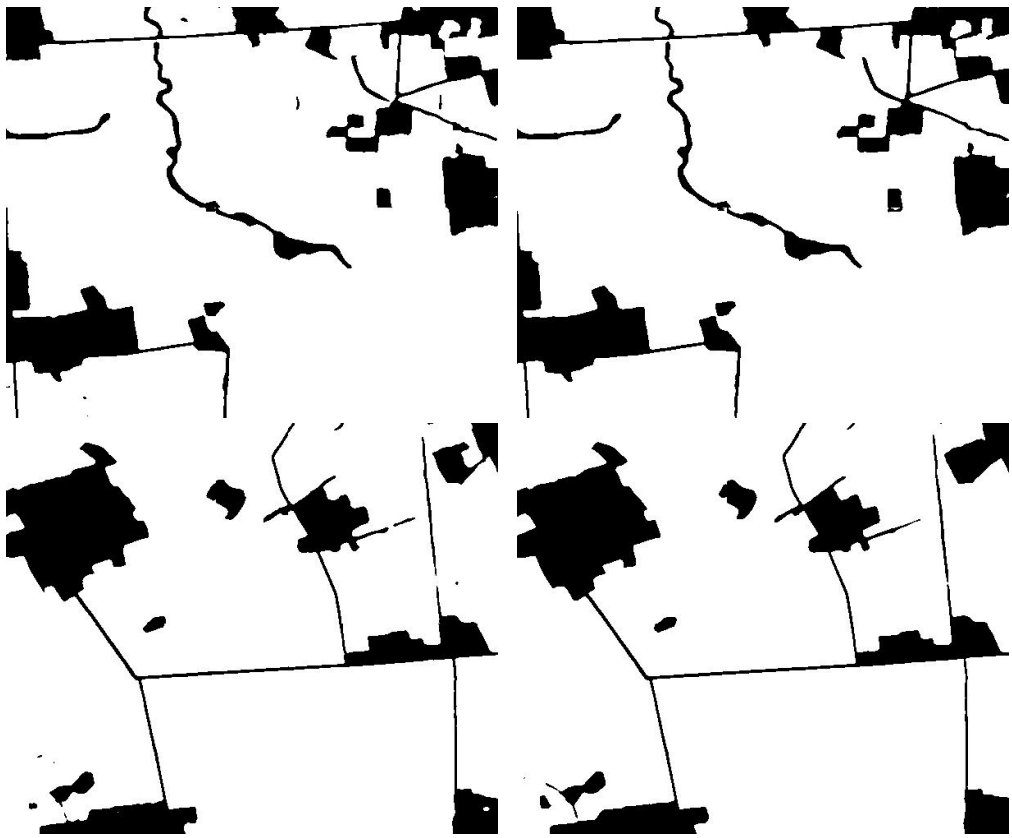


图 10-2 优化前后结果对比(左优化前，有优化后)

很明显可以观察到噪声减少,道路更清晰连贯

{图 10-2}由上图，左至右上至下 1-4 图，图 1 右上部分黑色斑点噪声在图 2 中消失，图 3 右侧和左下角间断的道路在图 4 中明显链接。优化提升效果明显。

5.3.6 Test1、Test2 图片耕地标签图制作

本文利用上述模型和优化方法，对 Test1 和 Test2 进行耕地标签图预测，得到如下耕地标签图结果（右侧为标签图），分割较为理想。

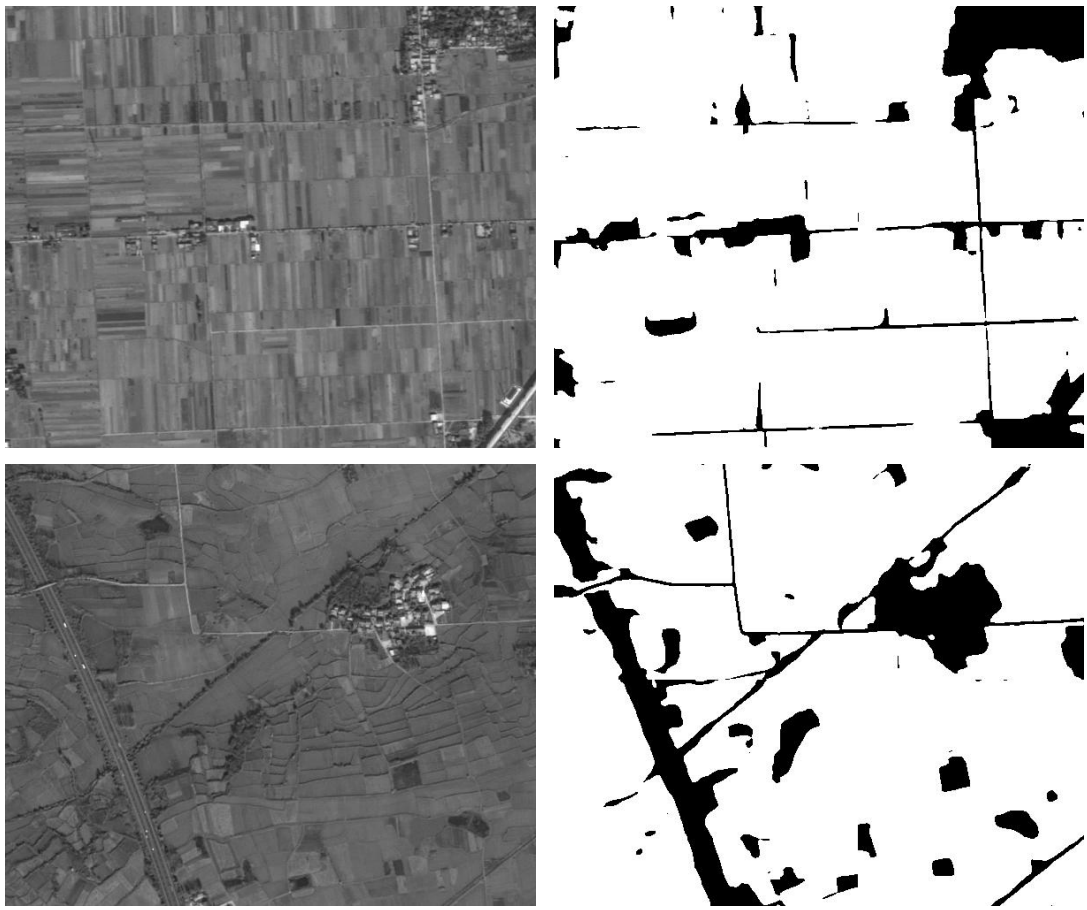


图 11 Test1 和 Test2 进行耕地标签图预测

5.3.7 完善问题一

得到较为理想的耕地标签图后，我们利用问题一中的计算方法得到 Test1 和 Test2 中的耕地面积，数据间问题一下表格（表 1）。

5.4 问题三的求解----创新思路

5.4.1 增加时间维度特征训练

以同一地块不同时间（如，春夏秋冬）的遥感图像做输入数据集（每一数据包含时间和空间特征），使模型学习到耕地**时间变化**，以此提高模型预测的精确性。训练集包含不同地点不同种类的耕地地块；着重训练耕地与非耕地间的边界，获得田间小路等精确信息。

5.4.2 人机配合

采用多图平均二值法，配合人工优中选优进行地块划分，步骤如下：

1）训练模型，并寻找模型优势及缺陷；

2）和优化模型中方法相同，进行同一预测图各 12 类输入预测；

3）从 2)各组 12 图中人工查找最优图（或多个较优图），进行融合拼接。

人机配合，可以让工作更加具有目的性，符合各种需求，如道路清晰连贯等。结合人工技术性筛选后的平均二值法结果性能增强，最终结果准确性提升。

正常情况下，训练集与测试集在多特征上具有足够多的相似性，直接预测即可得到理想结果，但若测试集在光线，更地块种类和训练图差距大，如 Test1 的情形，则需适当结合手动校准。

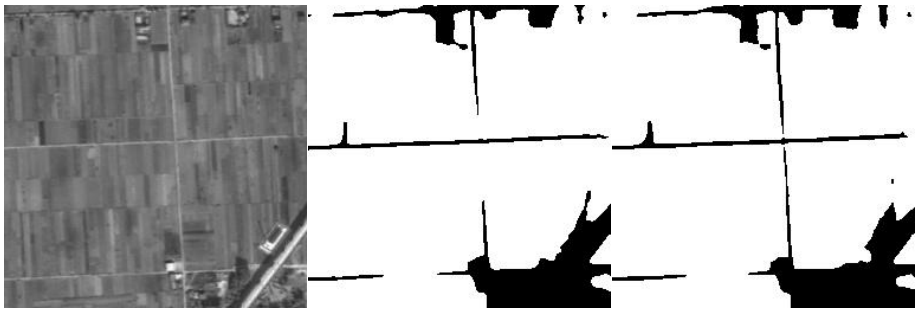


图 11 人机配合前(左)后(右)耕地标签图预测

{图 11}人机配合后的结果中对十字交叉道路的预测明显优于单独及其判断

5.4.3 插值法优化模型

因为食管癌的病例图片与耕地类似，均为块状分布，且与周围其他组织有灰度差异。故可采用优化食管癌病灶的分割模型的方法来优化耕地图像的分割。

因道路、河流某些部分宽度过窄或像素值与耕地相近，U-Net 在分割道路、河流时常会出现分离点，以至于不能精确分割。采用插值算法[6]，在分离部分寻找符合图形（曲线）走势的数值，补齐中间细小的分离点，使其分割线更完善，优化模型预测结果。

拟合函数是尽可能地寻找使新的数据点逼近其他样本点的过程，而插值函数则是要经过原本样本数据点，插值函数相较于拟合函数更能还原图片信息。[6]

5.4.4 多尺度特征提取[7][8]

提取图像多尺度特征时采用 Atrous 卷积，通过引入不同的扩张率参数，在不增加参数量的情况下扩大感受野，使得每个卷积输出都包含了较大范围的信息，在避免过拟合的同时，能减少图片特征信息的遗失。在训练耕地分割网络时能加入更多的周围环境影响因素，提升其分类正确率。

Atrous 卷积计算公式为：

$$y(i)=\sum_{k=1}^{K}x(i+r\cdot k)\omega(k)$$

其中 x(i)表示输入信号，w(k)表示长度为 k 的滤波器，r 为对输入信号进行采样的步幅，y(i)为 Atrous 卷积的输出。

六、模型评价

6.1 优点

- U-Net 网络预测模型结构体量的精细、对线状图像特征的准确提取，提取信息充分且分割能力强，细节处地物分割效果良好。相较于其他神经网络，U-Net 网络的参数较少，训练的时候不容易发生过拟合。同时，该网络层数较少，模型体积小，训练速度快。
- 本模型中采用的多图平均二值法，在减少噪声，连贯道路上表现突出。人工配合后，成果更为显著。该方法可以降低对模型训练的要求，对与训练图相差较大的测试图也可以得到较为良好的预测结果。

6.2 缺点

- 题目标签中仅将耕地与非耕地做了区分，故模型只能学习到耕地与非耕地间特征的区别，导致训练出的模型对环境的各组成成分区分能力不一，生成的标签图上会存在难以去除的

噪声（即使运用了集成学习的思想）。例如，当河流、道路某部分像素值与周围耕地像素值较接近时，生成的标签图上，该部分会被识别成耕地造成河流、道路的中断。位置若将分类细化，即将耕地与周围环境的要素一一区分，如河/耕地，路/耕地，森林/耕地，则分割精度会相应提高，下一步的优化应分类后对模型加强缺陷类训练。

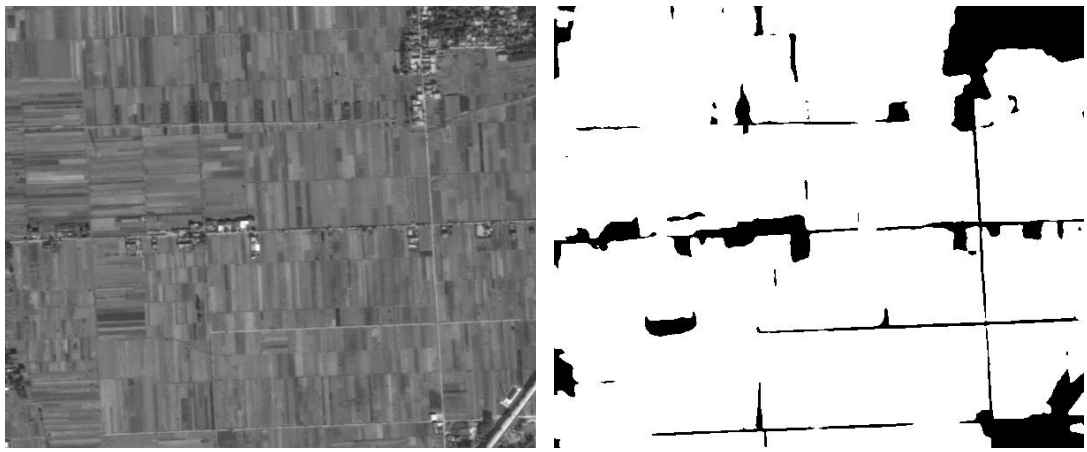
- 本文模型对细小的田间道路不能很精确的识别、分割，应在训练用的标签图里切割出细小道路的标签部分专门训练，再复合其他部分。一些田间小路没有标签且图像中存在人眼难以分辨的区域，下一步的模型优化应添加对弱标签优化。在训练前整理数据将标签进一步优化。

七、总结

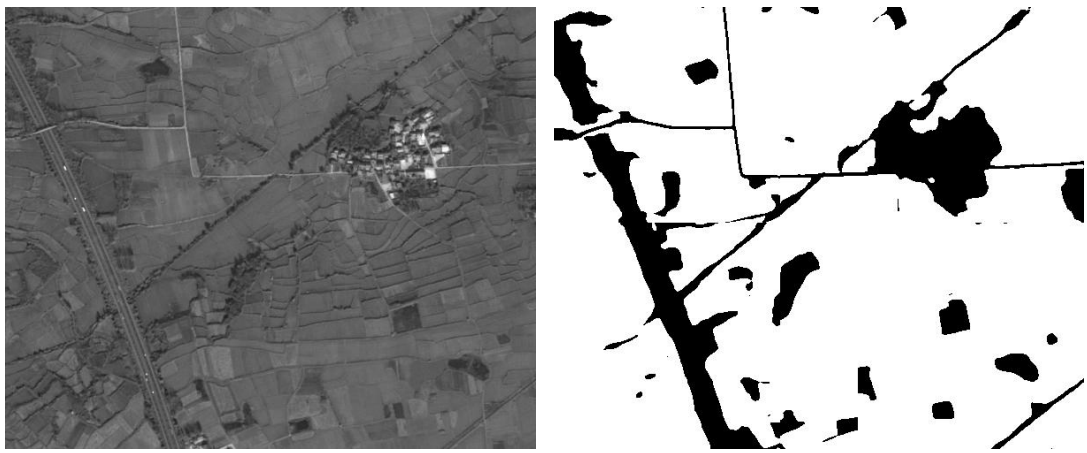
本文结合“集成学习”的思想，提出一种基于 U-Net 神经网络的遥感图像耕地地块分割模型。U-Net 网络预测模型在耕地地块分割与提取中表现良好。结合多图平均二值化和人工参与后，预测效果更为突出。随机数据增广方法在利用较少原始图像训练中作用明显。

通过预测结果可以看出，本文的模型取得了良好的分割性能，实现了对耕地和非耕地区域较为准确的分割，且具有很好的泛化作用，但是对于颜色、灰度与耕地相似的非耕地区域偶尔会被分割成耕地区域，需要进一步的改进。本文中训练的模型的准确率可达到 98.49%，测试中准确率达 95%，对 test 图片的预测结果如下图所示：

//Test1//



//Test2//



耕地面积计算结果如下表所示：

数据编号	耕地所占比例
Data 1	0.83292667
Data 2	0.84226667
Data 3	0.70042000
Data 4	0.74683667
Data 5	0.55335333
Data 6	0.85561667
Data 7	0.68023667
Data 8	0.83893667
Test 1	0.89282667
Test 2	0.84388000

八、参考文献

[1] 张哲晗. 基于编解码卷积神经网络的遥感图像分割研究[D].中国科学技术大学,2020.

[2] 范仁义.简单认识 Adam 优化器
[EB/OL].https://www.cnblogs.com/Renyi-Fan/p/13374682.html,2020-7-24.

[3] 程引.深度学习中的 batch 的大小对学习效果有何影响
[EB/OL].https://www.zhihu.com/question/32673260,2015-11-8

[4] Xnion.深度学习--keras 实现 unet 代码及数据
[EB/OL].https://blog.csdn.net/Xnion/article/details/105797671?utm_source=app,2020-4-27

[5] 周志华. 机器学习[M].北京:清华大学出版社, 2016: 182-183

[6] 仵晨阳,何瑶.基于 U-Net 网络的食管癌病灶的分割研究[J].计算机与数字工程,2020,48(11):2734-2738.

[7] 王雪.基于 U-Net 多尺度和多维度特征融合的皮肤病变分割方法[J].吉林大学学报(理学版),2021,59(01):123-127.

[8] Chen L C , Papandreou G , Kokkinos I , et al. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs[J]. 2017.40（4）: 834-848Xnion.

[9] 深度学习--keras 实现 unet 代码及数据
[EB/OL].https://blog.csdn.net/Xnion/article/details/105797671?utm_source=app,2020-4-27

附录 1 程序

代码类型： python
软件： Spider、PyCharm

【程序文件总体概况】
<data.py>:模型训练(train)和简单测试(test)时用的基本函数库
//部分引自[1]

<model.py>:unet 基本模型
//部分引自[1]

<pre-data.py>:准备数据，用于将不可视的 tif 转化成可视的 png 处理，将 image 和 mask 剪切生成原始训练数据集
转化理由：过程可视化，方便寻找模型缺陷，设计合理应用方法，提高最终结果准确性
操作文件夹：data、train、test

<train.py>:训练网络，调用 data.py 和 model.py
操作文件夹：train
//部分引自[1]

<test.py>:模型效果检测，测试集 image 和 truelabel 不同于原始训练数据集剪切，提供未经过优化的预测方法
操作文件夹：test
//部分引自[1]

<predict.py>:（详细见 predict 文件夹内 readme 文件）
---1 利用训练模型结果，完整进行由卫星 tif 至标签 tif 的转化，中间过程 png 格式可视化显示。
---2 提供优化后的预测方法：
a)将 500*600 切为 256*256;
b)每 256*256 图通过 180 度旋转以及相应 x，y 镜像生成 6 张图输入模型;
c)将 6 图输出结果对比，利用适当算法求取平均，整合为 1 张 256*256 图;
d)将 6 张 256*256 图拼接成 500*600 的结果图;
操作文件夹：predict
tif_question--->in1--->in6--->in12--->out12--->out12change--->out6--->out1--->tif_result
【注】上述优化方法是通过观察下述方法生成的 12 图选定的 6 种合适图片处理方式完成。
<predict(12).py>:
a)将 500*600 切为 256*256;

b)每 256*256 图通过 90，180，270 旋转以及相应 x，y 镜像生成 12 张图输入模型；
c)将 12 图输出结果对比，利用适当算法求取平均，整合为 1 张 256*256 图；
d)将 6 张 256*256 图拼接成 500*600 的结果图；
操作文件夹：predict
tif_question--->in1--->in6--->in12--->out12--->out12change--->out6--->out1--->tif_result

<S_calculation>：利用 tif 或 png 标签图计算每张图耕地占比。

---

pre_data.py 数据预处理

```
import os
import glob
import cv2
import numpy as np

#将原始图片 if(500,600)转为可视图片 png(500,600)
def png_generator(file_path, file_savepath):
    print(' <.tif to .png >'+file_path+'-->'+file_savepath+':')
    i=0
    for files in glob.glob(file_path+"/*.tif"): #遍历 image 每个文件
        i+=1
        #filepath,filename = os.path.split(files)
        x = cv2.imread(files, -1)
        m = x.max()
        q = (x*255)/m #标准化
        cv2.imwrite(file_savepath+'/'+'%d'%(i-1)+".png", q) #转 png
    print('   new_png_number = '+'%d'%i)

def data_cut(file_path, file_savepath):
    print(' pre_train_cut:')
    print(file_path+'-->'+file_savepath+':')
    k=0
    for i in range(0, 245, 61):  #竖着 5 张
        for j in range(0, 345, 43):  #横着 9 张
            for files in glob.glob(file_path+"/*.png"): #遍历 image 每个文件
                k+=1
                #filepath,filename = os.path.split(files)
                x = cv2.imread(files, -1)
                q = x[i : (256+i), j : (256+j) ]
                cv2.imwrite(file_savepath+"/"+'%d'%(k-1)+".png", q)
    print('   generater_number = '+'%d'%k) #5*9*8=360;5*9*2=90
```

```
def pre_test_cut(file_path, savepath):
    print('pre_test_cut :')
    print(file_path+'-->'+savepath+':')
    k=0  #顺序是依次右上每张图，再左切每张图。。。
    a = [0, 244]
    b = [0, 214, 344]
    for p in range(0,2):  #竖着 2 张
        for q in range(0,3):  #横着 3 张
            for files in glob.glob(file_path+"/*.png"): #遍历 image 每个文件
                k+=1
                filepath,filename = os.path.split(files)
                i = a[p]
                j = b[q]
                x = cv2.imread(files, -1)
                m = x[i : (256+i), j : (256+j) ]
                cv2.imwrite(savepath+"/"+'%d'%(k-1)+".png", m)
    print('   generater_number = '+'%d'%(k)) #2*3 = 6, 6*8=48

#路径是数据文件夹相对路径

#tif path
image1 = r"data/tif/train_image"
mask1 =  r"data/tif/train_mask"
test1 = r"data/tif/test_image"

#tif-->png
#png save path
image2 =r"data/png/train_image"
mask2 = r"data/png/train_mask"
test2 = r"data/png/test_image"

#cut save path
image3 = r"train/image"
mask3 = r"train/mask"
test3 = r"test/image"
test4 =  r"test/true_label"

png_generator(image1, image2)
png_generator(mask1,  mask2)
png_generator(test1, test2)

data_cut(image2, image3)
data_cut(mask2,  mask3)
```

```
pre_test_cut(image2, test3)
pre_test_cut(mask2, test4)
```

data.py 训练测试时图片处理 [9]

```
from __future__ import print_function
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import os
import glob
import skimage.io as io
import skimage.transform as trans

#adjustData()函数主要是对训练集的数据和标签的像素值进行归一化
def adjustData(img,mask):
    if(np.max(img) > 1):
        img = img / 255.0
        mask = mask /255.0
        mask[mask > 0.5] = 1
        mask[mask <= 0.5] = 0
    return (img,mask)

    '''
    can generate image and mask at the same time
    use the same seed for image_datagen and mask_datagen to ensure the transformation
for image and mask is the same
    if you want to visualize the results of generator, set save_to_dir = "your path"
    '''
# trainGenerator()函数主要是产生一个数据增强的图片生成器, 不断生成图片
def trainGenerator(batch_size , train_path, image_folder, mask_folder, aug_dict,
image_color_mode = "grayscale",
                   mask_color_mode = "grayscale", image_save_prefix  = "image",
mask_save_prefix  = "mask",
                   flag_multi_class = False, num_class = 2, save_to_dir = None,
target_size = (256,256),seed = 1):

    image_datagen = ImageDataGenerator(**aug_dict)
    mask_datagen = ImageDataGenerator(**aug_dict)#aug_dict 控制处理范围和方法
    image_generator = image_datagen.flow_from_directory(
        train_path,#训练数据文件夹路径
        classes = [image_folder],#类别文件夹,对哪一个类进行增强
        class_mode = None,#不返回标签
        color_mode = image_color_mode,#灰度,单通道模式
        target_size = target_size,#转换后的目标图片大小
```

```
        batch_size = batch_size,#每次产生的进行转换后的图片张数
        save_to_dir = save_to_dir,#图片保存路径
        save_prefix  = image_save_prefix,#生成图片的前缀，提供 save_to_dir 时有效
        seed = seed)
    mask_generator = mask_datagen.flow_from_directory(
        train_path,
        classes = [mask_folder],
        class_mode = None,
        color_mode = mask_color_mode,
        target_size = target_size,
        batch_size = batch_size,
        save_to_dir = save_to_dir,
        save_prefix  = mask_save_prefix,
        seed = seed)
    train_generator = zip(image_generator, mask_generator)#组合成一个生成器
    for (img,mask) in train_generator:#由于 batch 是 4，所以一次返回两张，即 img 是一个 4 张灰度图片的数组，[4,256,256]
        img,mask = adjustData(img,mask)#数据和标签的像素值进行归一化，返回的 img 依旧是[4,256,256]
        yield (img,mask)#每次分别产出两张图片和标签


# testGenerator()函数主要是对测试图片进行规范，使其尺寸和维度上和训练图片保持一致
def testGenerator(test_path, num_image = 30, target_size = (256,256), flag_multi_class = False, as_gray = True):
    for i in range(num_image):
        img = io.imread(os.path.join(test_path,"%d.png"%i),as_gray = as_gray)
        img = img / 255.0
        img = trans.resize(img,target_size)
        img = np.reshape(img,img.shape+(1,)) if (not flag_multi_class) else img
        img = np.reshape(img,(1,)+img.shape) #(1,)+(2,3) = (1,2,3)
       #将测试图片扩展一个维度，与训练时的输入[4,256,256]保持一致
        yield img


# geneTrainNpy()函数主要是分别在训练集文件夹下和标签文件夹下搜索图片，
# 然后扩展一个维度后以 array 的形式返回，是为了在没用数据增强时的读取文件夹内自带的数据
def geneTrainNpy(image_path,mask_path,flag_multi_class = False,num_class = 2,image_prefix = "image",mask_prefix = "mask",image_as_gray = True,mask_as_gray = True):
    image_name_arr = glob.glob(os.path.join(image_path,"%s*.png"%image_prefix))
   #相当于文件搜索，搜索某路径下与字符匹配的文件
    image_arr = []
```

```
    mask_arr = []
    for index,item in enumerate(image_name_arr):#enumerate 是枚举，输出
[(0,item0),(1,item1),(2,item2)]
        img = io.imread(item,as_gray = image_as_gray)
        img = np.reshape(img,img.shape + (1,)) if image_as_gray else img
        mask =
io.imread(item.replace(image_path,mask_path).replace(image_prefix,mask_prefix),a
s_gray = mask_as_gray)
       #重新在 mask_path 文件夹下搜索带有 mask 字符的图片（标签图片）
        mask = np.reshape(mask,mask.shape + (1,)) if mask_as_gray else mask
        img,mask = adjustData(img,mask,flag_multi_class,num_class)
        image_arr.append(img)
        mask_arr.append(mask)
    image_arr = np.array(image_arr)
    mask_arr = np.array(mask_arr)#转换成 array
    return image_arr,mask_arr

# 生成二值图片/灰度图： 1/255
def saveResult(save_path,npyfile):
    for i, item in enumerate(npyfile):
        img = item[:,:,0]
        print(np.max(img), np.min(img))

        #img = img/255
        #img[img>=0.5]=1#此时 1 是浮点数，下面的 0 也是，灰度改成 255
        #img[img<0.5]=0
        #print(np.max(img), np.min(img))
        io.imsave(os.path.join (save_path, "%d.png"%i), img)
```

model.py [9]

```
import numpy as np
import os
import skimage.io as io
import skimage.transform as trans
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras import backend as keras

def unet(pretrained_weights = None,input_size = (256,256,1)):
    inputs = Input(input_size)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
```

```
= 'he_normal')(inputs)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(pool1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(pool2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(pool3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv4)
    drop4 = Dropout(0.5)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(pool4)
    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv5)
    drop5 = Dropout(0.5)(conv5)

    up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(UpSampling2D(size = (2,2))(drop5))
    merge6 = concatenate([drop4,up6], axis = 3)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(merge6)
    conv6 = 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))
    merge7 = concatenate([conv3,up7], axis = 3)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(merge7)
    conv7 = 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))
    merge8 = concatenate([conv2,up8], axis = 3)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(merge8)
    conv8 = 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))
    merge9 = concatenate([conv1,up9], axis = 3)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(merge9)
    conv9 = 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(1, 1, activation = 'sigmoid')(conv9)

    #model = Model( input = inputs, output = conv10)
    model = Model( inputs, conv10)
    model.compile(optimizer=Adam(lr = 1e-4), loss = 'binary_crossentropy', metrics
= ['accuracy'])
    #'mse'均方误差
    model.summary()

    if(pretrained_weights):
       model.load_weights(pretrained_weights)

    return model
```

train.py [9]

```
from model import *
from data import *

import matplotlib as plt
# #os.environ["CUDA_VISIBLE_DEVICES"] = "0"
# data_gen_args = dict() : 为 keras 自带的图像增强方法
data_gen_args = dict(rotation_range=10, #整数。随机旋转的度数范围。
                     width_shift_range=0.1, #浮点数、一维数组或整数
                     height_shift_range=0.1, #浮点数。剪切强度（以弧度逆时针方向剪
切角度）。
                     shear_range=0.05,
                     zoom_range=0.1, #浮点数 或 [lower, upper]。随机缩放范围
```

```
                    fill_mode='reflect',
                    horizontal_flip=True,
                    vertical_flip=True) # {"constant", "nearest", "reflect" or
"wrap"} 之一。默认为 'nearest'。输入边界以外的点根据给定的模式填充:
# 建立测试集，样本和标签分别放在同一个目录下的两个文件夹中，文件夹名字为：
'image','label'
#得到一个生成器，以batch=2的速率无限生成增强后的数据
myGene    =    trainGenerator(4,'train','image','mask',data_gen_args,save_to_dir
=r'train/mid') # data

# 调用模型，默认模型输入图像size=(256,256,1),样本位深为8位
model = unet() # model
# 保存训练的模型参数到指定的文件夹，格式为.hdf5；检测的值是'loss'使其更小
model_checkpoint = ModelCheckpoint('unet_model_100.hdf5', monitor='loss',verbose=1,
save_best_only=True) # keras
# 开始训练，steps_per_epoch为迭代次数，epochs:
h                                                                                =
model.fit_generator(myGene,steps_per_epoch=90,epochs=100,callbacks=[model_checkp
oint]) # keras

history = h.history

f = open("history.text", 'w')
f.write(str(history))
f.close()
print("save history successfully")
print(history)
```

python test.py [9]

```
from model import *
from data import *
"""
1 target_size()为图片尺寸，要求测试集图像尺寸设置和model输入图像尺寸保持一致，
2 如果不设置图片尺寸，会对输入图片做resize为处理，输入网络和输出图像尺寸默认均为
（256,256），
3 且要求图片位深为8位，24/32的会报错！！
4 测试集数据名称需要设置为：0.png……
5model.predict_generator( ,n, ):n为测试集中样本数量，需要手动设置，不然会报错！！
"""
# 输入测试数据集，
testGene = testGenerator(r"test/image",48, target_size = (256,256)) # data
```

```
# 导入模型
model = unet(input_size = (256,256,1)) # model
# 导入训练好的模型
model.load_weights("unet_model_100.hdf5")
# 预测数据
results = model.predict_generator(testGene,48,verbose=1) # keras
#print(results)
saveResult(r"test/mask",results) # data
print("over")
```

predict.py(6 图)

```
#import matplotlib.gridspec as gridspec

#目标：将 test 图片切分成不重合的小块，分别进行：
#   正、左、右、下和它们的镜像 8 种操作，
#   再进入模型预测，结果还原求最大概率：
#       结果求和，>=4 置 1，<4 置 0

def Mirror_x(img):                  #关于 x 轴镜像
   new  = cv2.flip(img,  0)
   return new

def Mirror_y(img):                  #关于 y 轴镜像
    new  = cv2.flip(img,  1)
    return new

def  AntiClock90(img):                  #逆时针旋转 90 度
    trans = cv2.transpose(img) #矩阵转置
    new  = cv2.flip(trans,  0)  #关于 x 轴镜像
    return new

def  Clock90(img):                  #顺时针旋转 90 度
    trans = cv2.transpose(img) #矩阵转置
    new  = cv2.flip(trans,  1)  #关于 y 轴镜像
    return new

def png_generator(file_path, file_savepath):
    print(' <.tif to .png >'+file_path+'-->'+file_savepath+':')
    i=0
    for files in glob.glob(file_path+"/*.tif"): #遍历 image 每个文件
        i+=1
        filepath,filename = os.path.split(files)
        print(filename+'    '+'%d'%(i-1)+".png")
```

```
        x = cv2.imread(files, -1)
        m = x.max()
        q = (x*255)/m #标准化
        cv2.imwrite(file_savepath+'/'+'%d'%(i-1)+".png", q) #转 png
    print('   new_png_number = '+'%d'%i)
    return i
#将图片(500,600)转为(256,256)2*3=6 张

def test_cut(img, savepath):
    print('test_cut')
    print(img+'-->'+savepath+':')
    k=0
    a = [0, 244]
    b = [0, 214, 344]
    for p in range(0,2):  #竖着 5 张
        for q in range(0,3):  #横着 9 张
                k+=1
                #filepath,filename = os.path.split(files)
                i = a[p]
                j = b[q]
                x = cv2.imread(img, -1)
                m = x[i : (256+i), j : (256+j) ]
                cv2.imwrite(savepath+"/"+'%d'%(k-1)+".png", m)
    print('   number = '+'%d'%k) #2*3 = 6

def test_generator(imgpath, savepath):   #6---->(6*6)
   k=0
   for files in glob.glob(imgpath+"/*.png"):
      #filepath,filename = os.path.split(files)
       img = cv2.imread(files, -1)

       cv2.imwrite(savepath+"/"+"%d"%(0 + 6*k)+".png", img)
       cv2.imwrite(savepath+"/"+"%d"%(1 + 6*k)+".png", Mirror_x(img))
       cv2.imwrite(savepath+"/"+"%d"%(2 + 6*k)+".png", Mirror_y(img))

       img = AntiClock90(img)
       img = AntiClock90(img)
       cv2.imwrite(savepath+"/"+"%d"%(3 + 6*k)+".png", img)
       cv2.imwrite(savepath+"/"+"%d"%(4 + 6*k)+".png", Mirror_x(img))
       cv2.imwrite(savepath+"/"+"%d"%(5 + 6*k)+".png", Mirror_y(img))
#优化后保留 0 和 180 旋转以及其 x,y 镜像
       #img = AntiClock90(img)
       #cv2.imwrite(savepath+"/"+ "%d"%(6 + 12*k)+".png", img)
       #cv2.imwrite(savepath+"/"+"%d"%(7 + 12*k)+".png", Mirror_x(img))
```

```
        #cv2.imwrite(savepath+"/"+"%d"%(8 + 12*k)+".png", Mirror_y(img))

        #img = AntiClock90(img)
        #cv2.imwrite(savepath+"/"+"%d"%(9 + 12*k)+".png", img)
        #cv2.imwrite(savepath+"/"+"%d"%(10 + 12*k)+".png", Mirror_x(img))
        #cv2.imwrite(savepath+"/"+"%d"%(11 + 12*k)+".png", Mirror_y(img))

        k+= 1
    print('   number = '+'%d'%(k*6)) #12*6 = 72

from model import *
from data import *
def test_result(path, savepath, n = 36):
    # python test.py
    """
    注:
            A: target_size()为图片尺寸, 要求测试集图像尺寸设置和model输入图像尺寸保持一致,
                    如果不设置图片尺寸, 会对输入图片做resize为处理, 输入网络和输出图像尺寸默认均为(256,256),
            B: 且要求图片位深为8位, 24/32的会报错!!
            C: 测试集数据名称需要设置为: 0.png……
            D: model.predict_generator( ,n, ):n为测试集中样本数量, 需要手动设置, 不然会报错!!
    """
    # 输入测试数据集,
    testGene = testGenerator(path , n,target_size = (256,256)) # data
    # 导入模型
    model = unet(input_size = (256,256,1)) # model
    # 导入训练好的模型
    model.load_weights("unet_model_100.hdf5")
    # 预测数据
    results = model.predict_generator(testGene, n, verbose=1) # keras
    #print(results)
    saveResult(savepath, results) # data
    print("over")

def mask_together(maskpath, savepath):  # (6*12)---->6
    k = 0
    a = [ ]
    for k in range (0,36):
        file = maskpath+"/%d"%k+".png"
        i = k%6
        filepath,filename = os.path.split(file)
```

```
        print(filename)
        x = cv2.imread(file, -1)
        a.append(x)   #图片进表
        if (i==5) : #将图片回归原方向
           a[1] = Mirror_x(a[1])
           a[2] = Mirror_y(a[2])

           a[3] = Clock90(a[3])
           a[3] = Clock90(a[3])

           a[4] = Mirror_x(a[4])
           a[4] = Clock90(a[4])
           a[4] = Clock90(a[4])

           a[5] = Mirror_y(a[5])
           a[5] = Clock90(a[5])
           a[5] = Clock90(a[5])


           for o in range(0,6):
               u = a[o]
               cv2.imwrite("predict/out12change"+"/"+ "%d"%(o+k-5)+".png", a[o])
#中间过程

               u =a[o]
               u = u/(u.max())   #每张图最大值归一化
               #u[u<=0.5] *= 0.5
               #u[u>0.5] *= 1.5
               u = u-0.5     #为了求和黑的更黑白的更白
               a[o] = u
               #u[u>=0] = 1   #此除优先二值化，可以保留细节特征，但是整体图像效果
下降
               #u[u<0] = -1

           mask = a[0]+a[1]+a[2]+a[3]+a[4]+a[5]
           mask = mask/6 + 0.5    #取平均，恢复（0,1）

           maskmax = mask.max()   #最大值归一化
           mask = mask/maskmax

           write1 = mask*255
           name = (k-5)/6 #文件名

           cv2.imwrite(savepath+"/"+ "%d"%name+"_ori.png", write1)
```

```
            mask[mask>0.5]=255
            mask[mask<=0.5]=0
            cv2.imwrite(savepath+"/"+ "%d"%name+"_bi.png", mask)
            a = [ ] #列表更新
            print('\n')
            print('   finish  '+'%d'%(name)+'/6')

def result(maskpath, savepath, typename, namenumber):  #6---->1
    i = 0
    x = [ ]
    for k in range(0,6):
        file = maskpath+"/%d"%k+typename+".png"
        filepath,filename = os.path.split(file)
        print(filename)
        x.append(cv2.imread(file, -1))
        i+=1

    y = x[1]
    x[1] = y[: , 42 : 130]
     #左右合并
    line1 = np.concatenate([x[0], x[1]], axis = 1)
    line1 = np.concatenate([line1, x[2]], axis = 1)
    line1 = line1[0 : 244, :]

    y = x[4]
    x[4] = y[: , 42 : 130]
    #左右合并
    line2 = np.concatenate([x[3], x[4]], axis = 1)
    line2 = np.concatenate([line2, x[5]], axis = 1)
    mask = np.vstack((line1, line2))   #上下合并

    print(mask.shape)
    cv2.imwrite(savepath+"/%d"%namenumber +typename+ ".png", mask)
    print('\n')

#将二值 png 转 tif
def tif_generator(file_path, file_savepath):
     print(' <.png to .tif >'+file_path+'-->'+file_savepath+':')
     i=0
     for files in glob.glob(file_path+"/*_bi.png"): #遍历 image 每个文件,后缀随前程
序为_bi
         i+=1
         filepath,filename = os.path.split(files)
```

```
        x = cv2.imread(files, -1)/255
        x[x>=0.5] = 1
        x[x<0.5] = 0
        cv2.imwrite(file_savepath+'/'+filename+".tif", x) #转 png
    print('   new_tif_number = '+'%d'%i)


#路径是数据文件夹相对路径

tifpath1 =  r"predict/tif_question"  #存放原始卫星 tif 文件夹
in1 = r"predict/in1" #
in6 =  r"predict/in6"
in12 = r"predict/in12"

out12 = r"predict/out12"
out6 =  r"predict/out6"
out1 = r"predict/out1"

tifpath2 = r"predict/tif_result"  #存放标注结果 tif 文件夹

n = png_generator(tifpath1, in1 )#tif 图片数目

for number in range(0,n):
    test_cut(in1+'/%d.png'%number, in6)
    test_generator(in6,  in12)
    test_result(in12, out12, 36)
    mask_together(out12, out6)
    result(out6, out1, '_ori', number) #灰度图结果
    result(out6, out1, '_bi', number) #二值图预测结果
    print('finish_predict %d / %d'%(number+1, n))

tif_generator(out1, tifpath2)
```

predict.py(12 图)

```
import os
import glob
import cv2
import numpy as np
#import matplotlib.pyplot as plt
#import matplotlib.gridspec as gridspec

#目标: 将 test 图片切分成不重合的小块, 分别进行:
```

```
#   正、左、右、下和它们的镜像 8 种操作，
#   再进入模型预测，结果还原求最大概率：
#       结果求和，>=4 置 1，<4 置 0

def Mirror_x(img):                  #关于 x 轴镜像
   new  = cv2.flip(img,  0)
   return new

def Mirror_y(img):                  #关于 y 轴镜像
    new  = cv2.flip(img,  1)
    return new

def  AntiClock90(img):                  #逆时针旋转 90 度
    trans = cv2.transpose(img) #矩阵转置
    new  = cv2.flip(trans,  0)  #关于 x 轴镜像
    return new

def  Clock90(img):                  #顺时针旋转 90 度
    trans = cv2.transpose(img) #矩阵转置
    new  = cv2.flip(trans,  1)  #关于 y 轴镜像
    return new

def png_generator(file_path, file_savepath):
    print(' <.tif to .png >'+file_path+'-->'+file_savepath+':')
    i=0
    for files in glob.glob(file_path+"/*.tif"): #遍历 image 每个文件
        i+=1
        filepath,filename = os.path.split(files)
        print(filename+'    '+'%d'%(i-1)+".png")
        x = cv2.imread(files, -1)
        m = x.max()
        q = (x*255)/m #标准化
        cv2.imwrite(file_savepath+'/'+'%d'%(i-1)+".png", q) #转 png
    print('   new_png_number = '+'%d'%i)
    return i
#将图片(500,600)转为(256,256)2*3=6 张

def test_cut(img, savepath):
    print('test_cut')
    print(img+'-->'+savepath+':')
    k=0
    a = [0, 244]
    b = [0, 214, 344]
    for p in range(0,2):  #竖着 5 张
```

```
        for q in range(0,3):  #横着 9 张
                k+=1
                #filepath,filename = os.path.split(files)
                i = a[p]
                j = b[q]
                x = cv2.imread(img, -1)
                m = x[i : (256+i), j : (256+j) ]
                cv2.imwrite(savepath+"/"+'%d'%(k-1)+".png", m)
    print('   number = '+'%d'%k) #2*3 = 6

def test_generator(imgpath, savepath):   #6---->(6*12)
   k=0
   for files in glob.glob(imgpath+"/*.png"):
      #filepath,filename = os.path.split(files)
      img = cv2.imread(files, -1)

      cv2.imwrite(savepath+"/"+"%d"%(0 + 12*k)+".png", img)
      cv2.imwrite(savepath+"/"+"%d"%(1 + 12*k)+".png", Mirror_x(img))
      cv2.imwrite(savepath+"/"+"%d"%(2 + 12*k)+".png", Mirror_y(img))

      img = AntiClock90(img)
      cv2.imwrite(savepath+"/"+"%d"%(3 + 12*k)+".png", img)
      cv2.imwrite(savepath+"/"+"%d"%(4 + 12*k)+".png", Mirror_x(img))
      cv2.imwrite(savepath+"/"+"%d"%(5 + 12*k)+".png", Mirror_y(img))

      img = AntiClock90(img)
      cv2.imwrite(savepath+"/"+ "%d"%(6 + 12*k)+".png", img)
      cv2.imwrite(savepath+"/"+"%d"%(7 + 12*k)+".png", Mirror_x(img))
      cv2.imwrite(savepath+"/"+"%d"%(8 + 12*k)+".png", Mirror_y(img))

      img = AntiClock90(img)
      cv2.imwrite(savepath+"/"+"%d"%(9 + 12*k)+".png", img)
      cv2.imwrite(savepath+"/"+"%d"%(10 + 12*k)+".png", Mirror_x(img))
      cv2.imwrite(savepath+"/"+"%d"%(11 + 12*k)+".png", Mirror_y(img))

      k+=  1
   print('   number = '+'%d'%(k*12)) #12*6 = 72

from model import *
from data import *
def test_result(path, savepath, n = 72):
   # python test.py
   """
   注:
```

A: target_size()为图片尺寸，要求测试集图像尺寸设置和model输入图像尺寸保持一致，

如果不设置图片尺寸，会对输入图片做resize为处理，输入网络和输出图像尺寸默认均为（256,256），

B: 且要求图片位深为8位，24/32的会报错！！

C: 测试集数据名称需要设置为：0.png……

D: model.predict_generator(,n,):n为测试集中样本数量，需要手动设置，不然会报错！！

"""

输入测试数据集,

testGene = testGenerator(path , n,target_size = (256,256)) # data

导入模型

model = unet(input_size = (256,256,1)) # model

导入训练好的模型

model.load_weights("unet_model_100.hdf5")

预测数据

results = model.predict_generator(testGene, n, verbose=1) # keras

#print(results)

saveResult(savepath, results) # data

print("over")

```
def mask_together(maskpath, savepath):  # (6*12)---->6
    k = 0
    a = [ ]
    for k in range (0,72):
        file = maskpath+"/%d"%k+".png"
        i = k%12
        filepath,filename = os.path.split(file)
        print(filename)
        x = cv2.imread(file, -1)
        a.append(x)   #图片进表
        if (i==11) : #将图片回归原方向
            a[1] = Mirror_x(a[1])
            a[2] = Mirror_y(a[2])

            a[3] = Clock90(a[3])

            a[4] = Mirror_x(a[4])
            a[4] = Clock90(a[4])

            a[5] = Mirror_y(a[5])
            a[5] = Clock90(a[5])

            a[6] = Clock90(a[6])
```

```
            a[6] = Clock90(a[6])

            a[7] = Mirror_x(a[7])
            a[7] = Clock90(a[7])
            a[7] = Clock90(a[7])

            a[8] = Mirror_y(a[8])
            a[8] = Clock90(a[8])
            a[8] = Clock90(a[8])

            a[9] = AntiClock90(a[9])

            a[10] = Mirror_x(a[10])
            a[10] = AntiClock90(a[10])

            a[11] = Mirror_y(a[11])
            a[11] = AntiClock90(a[11])

            for o in range(0,12):
                u = a[o]
                cv2.imwrite("predict/out12change"+"/"+ "%d"%(o+k-11)+".png", a[o])
#中间过程

                u =a[o]
                u = u/(u.max())   #每张图最大值归一化
                #u[u<=0.5] *= 0.5
                #u[u>0.5] *= 1.5
                u = u-0.5     #为了求和黑的更黑白的更白
                a[o] = u
                #u[u>=0] = 1   #此除优先二值化，可以保留细节特征，但是整体图像效果
下降
                #u[u<0] = -1

            mask = a[0]+a[1]+a[2]+a[3]+a[4]+a[5]+a[6]+a[7]+a[8]+a[9]+a[10]+a[11]
            mask = mask/12 + 0.5    #取平均，恢复（0,1）

            maskmax = mask.max()   #最大值归一化
            mask = mask/maskmax

            write1 = mask*255
            name = (k-11)/12 #文件名

            cv2.imwrite(savepath+"/"+ "%d"%name+"_ori.png", write1)
```

```
            mask[mask>0.5]=255
            mask[mask<=0.5]=0
            cv2.imwrite(savepath+"/"+ "%d"%name+"_bi.png", mask)
            a = [ ] #列表更新
            print('\n')
            print('   finish  '+'%d'%(name)+'/6')

def result(maskpath, savepath, typename, namenumber):  #6---->1
    i = 0
    x = [ ]
    for k in range(0,6):
        file = maskpath+"/%d"%k+typename+".png"
        filepath,filename = os.path.split(file)
        print(filename)
        x.append(cv2.imread(file, -1))
        i+=1

    y = x[1]
    x[1] = y[: , 42 : 130]
     #左右合并
    line1 = np.concatenate([x[0], x[1]], axis = 1)
    line1 = np.concatenate([line1, x[2]], axis = 1)
    line1 = line1[0 : 244, :]

    y = x[4]
    x[4] = y[: , 42 : 130]
    #左右合并
    line2 = np.concatenate([x[3], x[4]], axis = 1)
    line2 = np.concatenate([line2, x[5]], axis = 1)
    mask = np.vstack((line1, line2))   #上下合并

    print(mask.shape)
    cv2.imwrite(savepath+"/%d"%namenumber +typename+ ".png", mask)
    print('\n')

#将二值png转tif
def tif_generator(file_path, file_savepath):
     print(' <.png to .tif >'+file_path+'-->'+file_savepath+':')
     i=0
     for files in glob.glob(file_path+"/*_bi.png"): #遍历 image 每个文件,后缀随前程
序为_ori
         i+=1
         filepath,filename = os.path.split(files)
         x = cv2.imread(files, -1)/255
```

```
        x[x>=0.5] = 1
        x[x<0.5] = 0
        cv2.imwrite(file_savepath+'/'+filename+".tif", x) #转 png
    print('   new_tif_number = '+'%d'%i)


#路径是数据文件夹相对路径

tifpath1 =  r"predict/tif_question"  #存放原始卫星 tif 文件夹
in1 = r"predict/in1" #
in6 =  r"predict/in6"
in12 = r"predict/in12"

out12 = r"predict/out12"
out6 =  r"predict/out6"
out1 = r"predict/out1"

tifpath2 = r"predict/tif_result"  #存放标注结果 tif 文件夹

n = png_generator(tifpath1, in1 )#tif 图片数目

for number in range(0,n):
    test_cut(in1+'/%d.png'%number, in6)
    test_generator(in6,  in12)
    test_result(in12, out12, 72)
    mask_together(out12, out6)
    result(out6, out1, '_ori', number) #灰度图结果
    result(out6, out1, '_bi', number) #二值图预测结果
    print('finish_predict %d / %d'%(number+1, n))

tif_generator(out1, tifpath2)
```

S_calculation.py(计算耕地面积)

```
from PIL import Image
import numpy as np
#from train import mask
import glob
import os
import cv2

path1 = 'data/tif/train_mask/*.tif'
```

```
path2 = 'predict/tif_result/*.tif'
path3 = 'predict/out1/*_bi.png'  #人工核实 png 成功生成, png 格式要'/255'

print('求耕地面积占比\n')

def whitepercent(a):
    c=2*np.abs(a-0.5)
    s=c.sum()  #s=sum(sum(c))
    n=a.sum()   #n=sum(sum(a))
    p=n/s
    return p

for files in glob.glob(path1):    #原始耕地面积生成
    filepath,filename = os.path.split(files)
    a = cv2.imread(files, -1) #np.asarray(Image.open(files))
    print(filename+'      ' +' %.8f \n' %whitepercent(a))

for files in glob.glob(path2):   #tif 耕地面积计算
    filepath,filename = os.path.split(files)
    a = cv2.imread(files, -1) #np.asarray(Image.open(files))
    print(filename+'      ' +' %.8f \n' %whitepercent(a))

for files in glob.glob(path3):  #png 耕地面积计算
    filepath,filename = os.path.split(files)
    a = cv2.imread(files, -1)/255 #np.asarray(Image.open(files))
    print(filename+'      ' +' %.8f \n' %whitepercent(a))
```

手动优中选优.py

```
import os
import glob
import cv2
import numpy as np


def mask_together(maskpath, savepath):  # (6*6)---->6
   k = 0
   a = [ ]
   for k in range (0,36):
        file = maskpath+"/%d"%k+".png"
        i = k%6
        filepath,filename = os.path.split(file)
        print(filename)
```

```
        x = cv2.imread(file, -1)
        #print(x)
        a.append(x)   #图片进表
        if (i==5) : #将图片回归原方向
           for o in range(0,6):
               u = a[o]
               #cv2.imwrite("predict/out12change"+"/"+ "%d"%(o+k-5)+".png", a[o])
#中间过程
               u =a[o]
               u = u/(u.max())   #每张图最大值归一化
               #u[u<=0.5] *= 0.5
               #u[u>0.5] *= 1.5
               u = u-0.5     #为了求和黑的更黑白的更白
               a[o] = u
               #u[u>=0] = 1   #此除优先二值化，可以保留细节特征，但是整体图像效果
下降
               #u[u<0] = -1

           mask = a[0]+a[1]+a[2]+a[3]+a[4]+a[5]
           mask = mask/6 + 0.5    #取平均，恢复（0,1）

           maskmax = mask.max()   #最大值归一化
           mask = mask/maskmax

           write1 = mask*255
           name = (k-5)/6 #文件名

           cv2.imwrite(savepath+"/"+ "%d"%name+"_ori.png", write1)

           mask[mask>0.5]=255
           mask[mask<=0.5]=0
           cv2.imwrite(savepath+"/"+ "%d"%name+"_bi.png", mask)
           a = [ ] #列表更新
           print('\n')
           print('   finish  '+'%d'%(name)+'/6')

def result(maskpath, savepath, typename, namenumber):  #6---->1
   i = 0
   x = [ ]
   for k in range(0,6):
      file = maskpath+"/%d"%k+typename+".png"
      filepath,filename = os.path.split(file)
      print(filename)
      x.append(cv2.imread(file, -1))
```

```
        i+=1


    y = x[0]
    x[0] =y[:,  :214]    #这里修改
    y = x[1]
    x[1] = y[: ,  : 130]
     #左右合并
    line1 = np.concatenate([x[0], x[1]], axis = 1)
    line1 = np.concatenate([line1, x[2]], axis = 1)
    line1 = line1[0 : 244, :]

    y = x[3]
    x[3] =y[:,  :214]
    y = x[4]
    x[4] = y[: ,  : 130]#这里修改
    #左右合并
    line2 = np.concatenate([x[3], x[4]], axis = 1)
    line2 = np.concatenate([line2, x[5]], axis = 1)
    mask = np.vstack((line1, line2))   #上下合并

    print(mask.shape)
    cv2.imwrite(savepath+"/%d"%namenumber +typename+ ".png", mask)
    print('\n')

#将二值 png 转 tif
def tif_generator(file_path, file_savepath):
    print(' <.png to .tif >'+file_path+'-->'+file_savepath+':')
    i=0
    for files in glob.glob(file_path+"/*_bi.png"): #遍历 image 每个文件,后缀随前程
序为_ori
        i+=1
        filepath,filename = os.path.split(files)
        x = cv2.imread(files, -1)/255
        x[x>=0.5] = 1
        x[x<0.5] = 0
        cv2.imwrite(file_savepath+'/'+filename+".tif", x) #转 png
    print('   new_tif_number = '+'%d'%i)


#路径是数据文件夹相对路径

#tifpath1 =  r"predict/tif_question"  #存放原始卫星 tif 文件夹
#in1 = r"predict/in1"
```

```
#in6 =  r"predict/in6"
#in12 = r"predict/in12"

out12 = r"hand_out12"
out6 =  r"hand_out6"
out1 = r"hand_out1"

tifpath2 = r"hand_tif_result"  #存放标注结果 tif 文件夹

#n = png_generator(tifpath1, in1 )#tif 图片数目

#for number in range(0,n):
    #test_cut(in1+'/%d.png'%number, in6)
    #test_generator(in6,  in12)
    #test_result(in12, out12, 72)
mask_together(out12, out6)
result(out6, out1, '_ori', 0) #灰度图结果
result(out6, out1, '_bi', 0) #二值图预测结果
print('finish_predict %d / %d'%(1, 1))

tif_generator(out1, tifpath2)
```

reault 未优化.py

```
import numpy as np
import cv2
import os

def result(maskpath, savepath,  namenumber):  #6---->1
   i = 0
   x = [ ]
   for k in range(0,6):
      file = maskpath+"/%d"%(k*8+namenumber)+".png"
      filepath,filename = os.path.split(file)
      print(filename)
      x.append(cv2.imread(file, -1))
      i+=1

   y = x[1]
   x[1] = y[: , 42 : 130]
    #左右合并
   line1 = np.concatenate([x[0], x[1]], axis = 1)
   line1 = np.concatenate([line1, x[2]], axis = 1)
   line1 = line1[0 : 244, :]
```

```
    y = x[4]
    x[4] = y[: , 42 : 130]
    #左右合并
    line2 = np.concatenate([x[3], x[4]], axis = 1)
    line2 = np.concatenate([line2, x[5]], axis = 1)
    mask = np.vstack((line1, line2))   #上下合并

    print(mask.shape)
    cv2.imwrite(savepath+"/%d"%namenumber+ ".png", mask)
    print('\n')

for i in range(0, 8):
    result('bi', 'result 未优化', i)
```

S_calculation(test).py

```
from PIL import Image
import numpy as np
#from train import mask
import glob
import os
import cv2

path1 = 'mask/*.png'  #测试

path2 = 'true_label/*.png'   #手绘标签

print('求耕地面积占比,测试分析模型特点\n')

def whitepercent(a):
    c=2*np.abs(a-0.5)
    s=c.sum()  #s=sum(sum(c))
    n=a.sum()   #n=sum(sum(a))
    p=n/s
    return p

for files in glob.glob(path1):    #原始耕地面积生成
    filepath,filename = os.path.split(files)
    a = cv2.imread(files, -1)
    m  = a.max()
    a = a/m   #a = a/255
    a[a>=0.5] = 1
    a[a<0.5] = 0
```

```
    print(filename+'      ' +' %.8f \n' %whitepercent(a))

for files in glob.glob(path2):    #tif耕地面积计算
    filepath,filename = os.path.split(files)
    a = cv2.imread(files, -1) /255#np.asarray(Image.open(files))
    print(filename+'      ' +' %.8f \n' %whitepercent(a))
```

loss&acc_draw.py

```
import matplotlib.pylab as pl
import matplotlib.pyplot as plt
import numpy as np

history = open('history_100.text', 'r')
dict = eval(history.read())
history.close()
#print(dict)
loss = dict['loss']
acc = dict['acc']

x = [ ]
for i in range(1,101):
    x.append(i)

def drew_lines(x, y, x_label, y_label, color):

    pl.plot(x,y, color)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(y_label, size = 14)
    pl.grid(True)   #要不要线
    #pl.scatter(x,y)  #要不要点
    pl.show()

if __name__=="__main__":  #运行本程序作图
    drew_lines(x, loss, 'epoch', 'loss', 'b')
    drew_lines(x, acc, 'epoch', 'accuracy', 'r')
```

附录 2 多图平均二值法优化前后对比图

（左前右后）