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：模型优化与参数评估

摘 要

准确标记耕地及计算耕地面积能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。

在初赛中，我队基于 U-Net 卷积神经网络建立模型对题目数据进行处理，实现了精确度较高的遥感图像耕地地块分割与提取，成功计算出耕地面积比例并制作出较为清晰的耕地标签图。参考“集成学习”的思想，我们提出“多图平均二值法”用以辅助 U-Net 模型预测。该方法成功降低了预测结果中的噪声干扰，显著提升了预测的准确度。

本文对初赛模型进行优化，提升其对耕地边界的识别能力。同时，本文分析了预测模型中的关键参数并定量评估其对模型精度的影响。

针对问题一，在初赛模型的基础上，本文进一步探究初赛提出的“多图平均二值法”的合理性与使用条件，并依据其原理改进了数据增广方法和模型训练参数，对数据增广的随机旋转范围和训练迭代次数进行优化，使得训练出的 U-Net 模型预测精度显著提升。优化后的模型准确率达 99.02%，损失率 0.041，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 2%范围内的对耕地的精确分割。同时，为了在保证预测精度的同时提升耕地标签图的视觉效果，本文提出采用腐蚀膨胀操作对预测结果做进一步处理。结果显示，腐蚀与膨胀操作使得模型预测效果得到进一步提升。相比初赛模型，本文模型具有更为优秀的分割性能和更为良好的视觉效果。本文模型的预测图像耕地边缘更为清晰，噪声基本消失，道路连通性强。利用上述优化模型，本文预测 Test3，Test4 图像的耕地地块分布并制作标签图，计算耕地面积占比。

针对问题二，我们对问题一中的优化模型进行深度分析，详细描述了该识别系统中的关键参数(模型训练参数和腐蚀膨胀操作核参数),并定量评估了它们对本文模型识别精度的影响。

本文所述的优化算法、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 2%范围内的对耕地的精确分割，预测图像噪声低，耕地边缘完整，可为耕地遥感制图提供借鉴与参考。

关键词：遥感图像分割 深度学习　U-Net　卷积神经网络　优化模型　腐蚀膨胀

目录

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。

基于初赛建立的模型与数据处理方法，本题需解决以下问题：

(1)优化初赛模型，在提取耕地边界方面做出改进；

(2)利用优化后的模型，从给定测试图像(Test3.tif、Test4.tif)中提取出耕地；

(3)制作测试图像(Test3.tif、Test4.tif)耕地标签图并计算图像中耕地面积比例；

(4)总结模型中的关键参数，并定量评估其对模型识别精度的影响。

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。

(5)假设低于 400 平方米(100 像素点)的误差属于较小面积误差，对识别精度影响不大。

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}	最大像素值
x_{min}	最小像素值
p	比值
x	原始像素值
x'	归一化后像素值
x''	可视化后像素值
y	“整合图片”像素值
y_i	第 i 子图像素值
θ	测试图与训练图特征方向夹角

四、问题分析

[问题 1]

在问题一中，题目要求我们优化初赛模型，从给定的测试图像(Test3.tif、Test4.tif)中提取出耕地，制作耕地标签图并计算其面积。数据集仍采用初赛中给出的 8 张遥感图像和人工标签。我们需要对初赛模型的预测能力进行分析，寻找模型缺陷，重点关注其对耕地边界的提取情况。在初赛模型的基础上设计优化思路，改进模型，并用优化后的模型对测试图像进行预测，制作耕地标签图，计算耕地面积比例。另外，我们需要比较优化前后对不同类别图像的表现，分析模型优化的合理性并说明优化效果。难点在于提升模型预测精度，优化模型对噪声、边界的处理。

[问题 2]

在问题二中，题目要求我们总结模型中的关键参数，并定量评估其对识别精度的影响。我们需要分析模型，总结对模型识别精度影响较大的参数。我们适当调整参数测试模型，定量评估它们对模型识别精度的影响，寻找最优参数设置。

五、问题求解

5.1 求解说明

5.1.1 初赛回顾

在初赛中，我队基于 U-Net 卷积神经网络建立模型对题目数据进行处理，实现了遥感图像耕地地块分割与提取，成功计算出耕地面积比例并制作出较为清晰的耕地标签图。初赛中我队的主要求解步骤是：先通过训练好的 U-Net 网络模型对图像进行初步预测，再通过“多图平均二值法”对预测图像做进一步处理得到最终结果。

(1) U-Net 网络模型

1 过程简述

我们对数据集 8 张图像和标签进行处理，经过标准化、可视化、切割、数据扩增操作，得到模型训练集和测试集。通过 Python 编程、利用 U-Net 卷积网络建立了体积小、训练速度快的预测模型。

该模型经过 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。训练损失(loss)和准确率(accuracy)与迭代次数(epoch)的关系如图 1 所示。

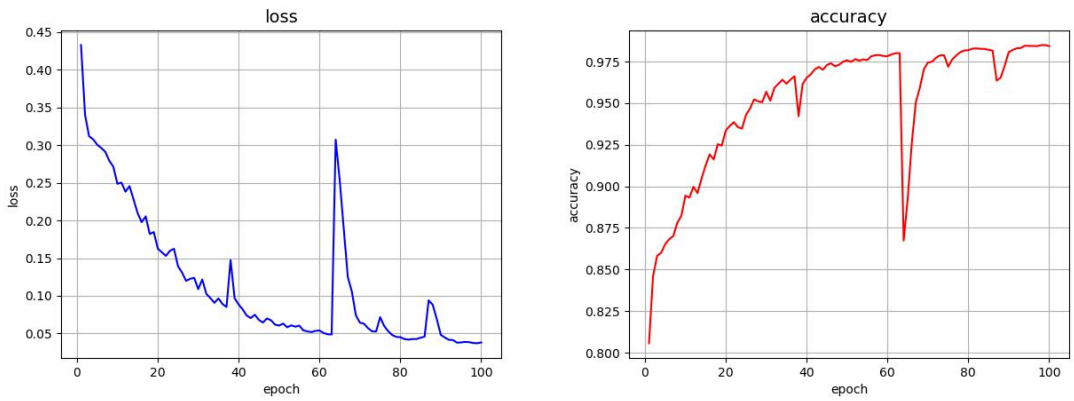


图 1 训练损失(loss)和准确率(accuracy)与迭代次数(epoch)的关系[初赛]

测试中该模型在遥感图像空间分辨率为 2m 的条件下可作出误差在 5%范围内的对耕地的精确分割。测试结果的相对误差在表 1 中给出。

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 1 48 图模型测试面积比例预测结果及相对误差[初赛]

2 结果分析

由图 1 可知，该 U-Net 网络模型的训练损失(loss)和准确率(accuracy)随迭代次数变化，最终归于平稳，训练结果较为理想。但该模型训练损失和准确率在前期变化较为稳定，后期(约从 63 次开始)产生突变。这说明该模型在训练中可能出现“过拟合”现象，模型泛化性和稳定性有待提升。

由表 2 数据可知，该模型的分割相对误差小于 5%，大部分测试结果相对误差在 1%以内，分割结果精度高。分析相应图像结果可知，该模型能较精确的分割耕地的边界，对于楼房住宅区、植被、池塘有较高的分割精确度且基本能实现道路连续分割，分割结果较为清晰。将预测结果与标签图比对可知，该模型仍存在问题：

(1)噪声问题。噪声明显，黑斑，白斑均有出现；

(2)边界问题。边界不圆滑，有毛刺；

(3)道路问题。部分道路间断，交叉处间断明显。

图 2-1 噪声问题 (左至右依次为遥感图、预测标签、人工标签)[初赛]

图 2-2 边界问题（左至右依次为遥感图、预测标签、人工标签)[初赛]

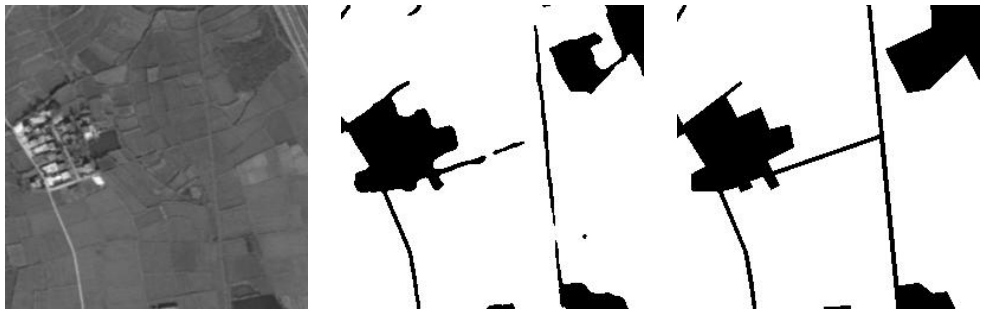


图 2-3 道路问题（左至右依次为遥感图、预测标签、人工标签)[初赛]

(2) 多图平均二值法

1 过程简述

在建立完整模型的基础上，我们在优化预测数据上提出创新方法：“多图平均二值法”。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结果进行提炼，得到较为准确的识别结果。初赛优化前后结果见表 2 和图 3。

结果显示，该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，噪声减少，边界清晰。采用“多图平均二值法”成功的降低了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。另外，我们提出在使用“多图平均二值法”时采用人机配合的选图方式进行耕地标记。这样既可以减少人工工作量，又能得到高质量的结果。

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

表 2 “多图平均二值法”优化前后结果与人工标签像素点差距[初赛]

图 3 “多图平均二值法”优化前后结果图对比(左至右，优化前，优化后，人工标签)[初赛]

2 结果分析

由表 2 和图 3 结果可知，“多图平均二值法”的使用可以明显提升预测精度，但由于最终结果依赖于 U-Net 模型的初步预测结果，所以优化效果仍受到限制。比如，图 3 中的优化使得结果与人工标签更为相似，但是仍未将道路完全连通。因此，“多图平均二值法”需要与 U-Net 网络模型相互协调配合。应进一步研究“多图平均二值法”原理及使用条件。

5.1.2 复赛目标

基于对初赛模型的结果分析与问题探究，我们复赛将围绕以下几个方面进行研究，对模型进行进一步优化：

(1)优化 U-Net 网络模型，提高模型泛化性和稳定性，一定程度上解决初步分割中的噪声、边界和道路问题；

(2)探究“多图平均二值法”原理其与原始图像的关系，明确其使用条件；

(3)寻找合适的方法对耕地标签图进行进一步优化，提高其精度与视觉效果。

另外，根据题目要求，我们需要对模型中重要参数和结果精度进行定量分析，评估参数的使用对模型和训练精度的影响。

5.2 问题一求解——优化模型并完成问题图像的耕地分割与面积计算

5.2.1 图像预处理

图像预处理方法沿用初赛方法，对原始图像数据进行标准化、归一化和分割处理，准备模型训练集图像与测试集图像。

(1)标准化

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：

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

(2) 可视化

题目提供的 tif 格式图片文件不适宜直接观察。图像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，在建立和测试模型时更易判断模型的合理性和准确性。同时，可视化使得预测结果更加直观。因此，本文应用可视化方法处理数据。

本文采用将归一化的数据扩大 255 倍的方法进行可视化，并将图片转存为 png 格式。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 $[0, 255]$。

可视化结果见图 4，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。

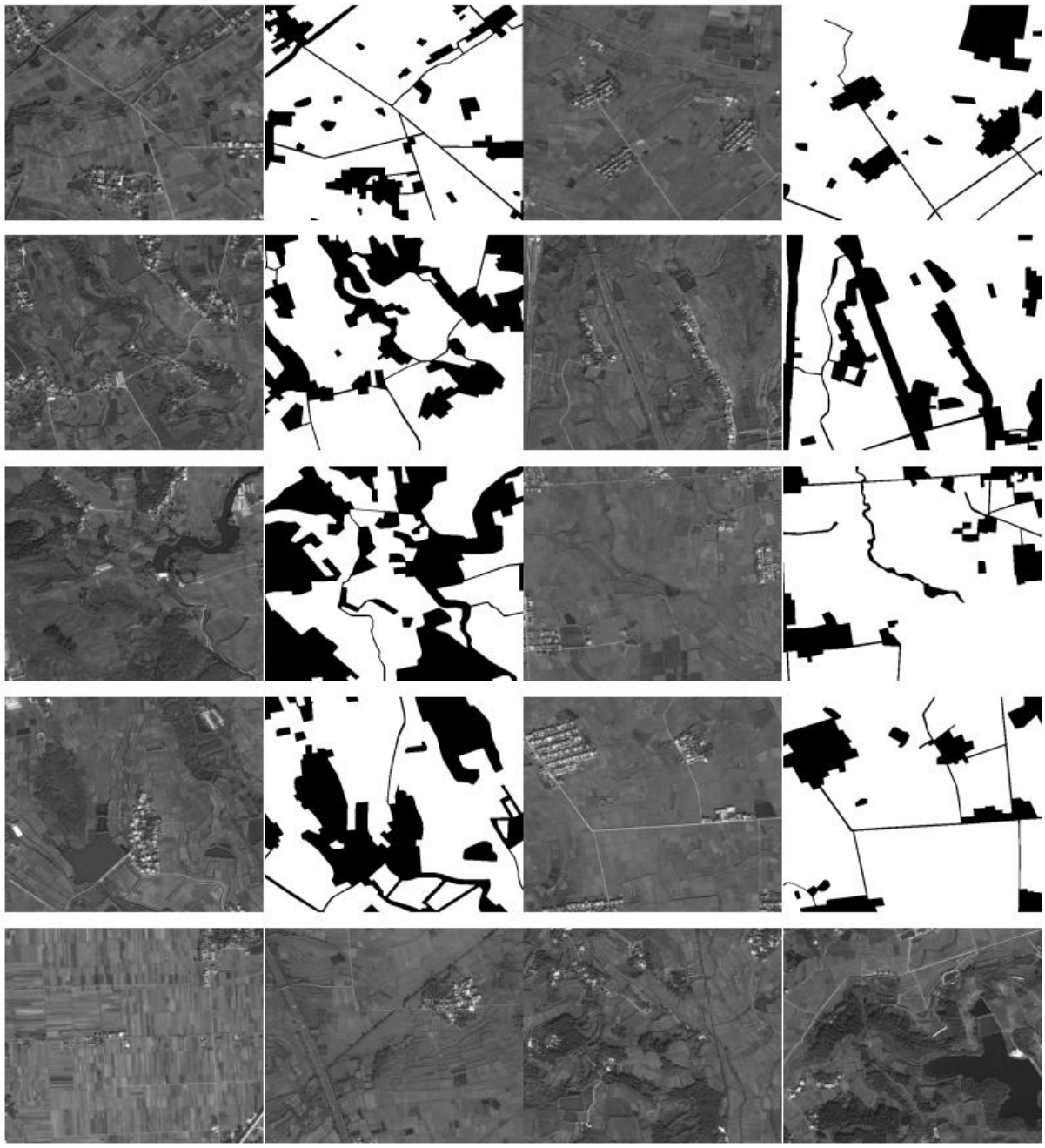


图 4 可视化结果(从左向右从上到下依次为 Data1-8，Test1-4)

(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】

为了同时保证模型训练集的完整性以及测试集与训练集图像不重复，本文采用不同于训练集的切割方式对数据进行处理,形成测试集。将一张像素为 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 5 图像切割结果

5.2.2 数据准备

数据准备上，本文对初赛方法进行了优化，将数据增强操作中的随机旋转角度范围由[0,10°]更改为[0,45°]。旋转范围的扩大使得训练集数据变化更加多样，一定程度上防止了“过拟合”的发生。后文中的测试结果和预测结果显示该修改提高模型泛化性和稳定性，一定程度上解决初步分割中的噪声、边界和道路问题。该优化选用 45°的原因与“多图平均法”的原理相关，将在后文中给出详细解释。

【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[0,45°]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。

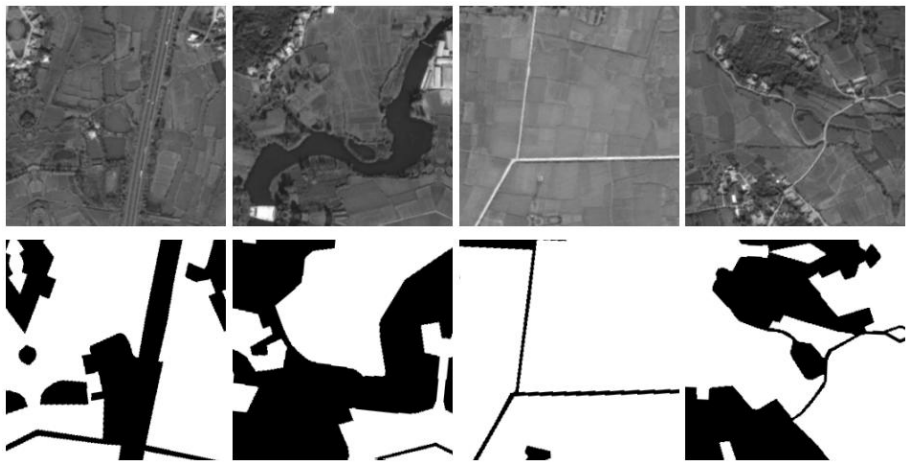


图 6 图像增广结果

【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。

5.2.3 模型建立

模型建立上，本文沿用初赛方法，建立 U-Net 网络模型，模型结构、损失函数、优化器的选取均与初赛相同。

(1)U-Net 神经网络模型

全卷积神经网络是图像分割和边缘检测的常用手段。其中，U-Net 由于结构体量精细、对线状图像特征提取准确而广泛被应用于细节处地物的分割，如遥感影像道路检测、医学影像血管提取等。

U-Net 网络结构清晰，呈现“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，膨胀阶段通过特征数量的拼接对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。这使其十分适合像本题这类原始数据量较小的情况。因此，本文采用 U-Net 网络结构设计模型用以提取图片信息。

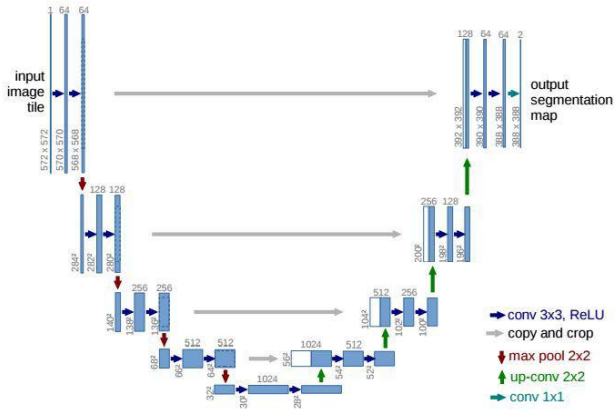


图 7 U-Net 卷积示例图[1]

(2)模型结构

本文运用的U-Net神经网络主要包含23个3×3的卷积层，4个2×2的下采样层，4个2×2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“收紧-膨胀”结构。

U-Net单次下采样中包括2个的3×3的的卷积层，一个2×2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(训练准确率)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。

5.2.4 模型训练

模型训练上，本文在初赛的基础上进行了优化。此前在数据准备上进行的扩大图像随机旋转角度的优化使得模型需要学习的数据变化更加多样，因此，训练的迭代次数应做适当增加。经过多次尝试,本文将初赛的迭代次数epochs = 100更改为300,以配合随机旋转范围[0,45°]的设定，防止“欠拟合”和“过拟合”的发生。训练结果显示，该优化合理，使得模型稳定性增加。

其它参数本文沿用初赛参数，为了保证求解思路完整，后文再次解释参数的选取理由。

(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		

表 3 batch_size 对训练的影响[3]

参考上图，我们选取 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】

考虑到初赛迭代次数 epochs = 100 与[0, 10°]的随机旋转范围相适应。在将旋转范围修改为[0, 45°]后，我们依次选取 100、150、250、300 进行尝试，结果显 100、150、250 次未饱和，300 次饱和且稳定，因此本文选取 epochs = 300。

(2) 训练结果

本文所述的模型经过 300 轮的训练，共训练 108000 组图像，最终模型准确率达 99.02%，损失率为 0.041。利用 ModelCheckpoint 函数将模型文件导出保存。相比于初赛进行的 100 轮训(loss0.037, acc98.49%，图 1)，本次训练结果损失率(loss)虽有下降但变化不大，准确率(acc)有明显上升。同时，考虑到本次训练并无类似初赛“突变”情况产生，基本可否定初赛论文中所提到的“白化过度造成突变”的猜测，表明初赛模型有可能发生“过拟合”。

综上所述，本文对模型的优化(将图像增强随机旋转角度由[0, 10°]更改为[0, 45°]；将迭代次数 epochs 由 100 次更改为 300 次）较为合理，成功提高了模型的稳定性。

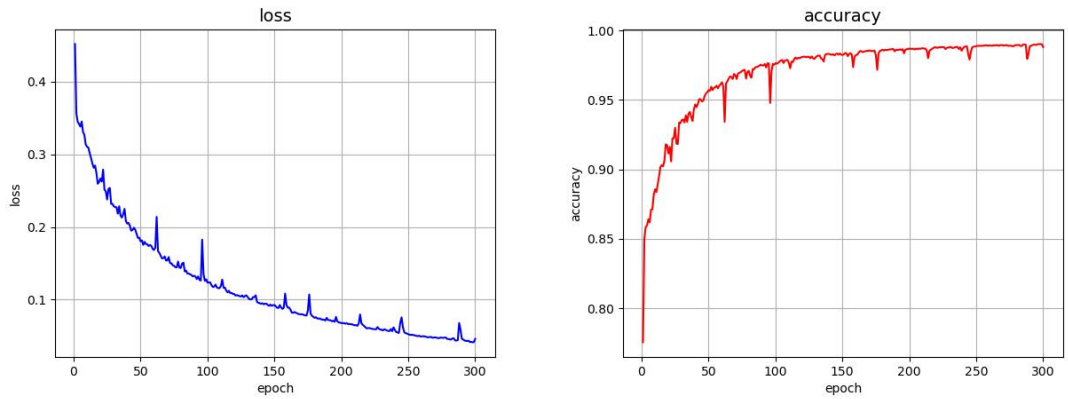


图 8 训练损失(loss)和准确率(accuracy)与迭代次数(epoch)的关系

5.2.5 模型测试

得到经过上述优化的完整模型后，我们利用模型对切割好的测试集进行预测，得到初步的耕地分割结果并与人工标签比较计算误差，结果见表 4。相对误差计算公式为：

$$相对误差 = \frac{测试结果 - 标签}{标签} \times 100\%$$

将表 4 中数据与表 1 中数据对比(即优化前后模型预测结果数据对比)可知，优化后的模型较优化前的初赛模型性能明显提升。优化前模型存在较多问题图像(相对误差大于等于 0.01)，优化后模型仅存在一个问题图像。同时，优化后模型的相对误差整体低于优化前模型，优化效果良好。另外，通过观察预测结果图像，我们发现新的模型相比于旧的模型在对噪声、边界和道路问题的处理上明显有所提升，见图 9。

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	0.81355286	-0.00600309
1	0.89433289	0.89398193	-0.00039243
2	0.68728638	0.68360901	-0.00535056
3	0.64765930	0.64588928	-0.00273295
4	0.47662354	0.47312927	-0.00733130
5	0.89802551	0.89938354	0.00151224
6	0.73742676	0.73658752	-0.00113807
7	0.70938110	0.70947266	0.00012907
8	0.90330505	0.90240479	-0.00099663
9	0.88595581	0.88478088	-0.00132617
10	0.64797974	0.64688110	-0.00169549
11	0.84605408	0.84407043	-0.00234459
12	0.50753784	0.50390625	-0.00715531
13	0.89022827	0.89152527	0.00145693
14	0.72434998	0.72315979	-0.00164311
15	0.88604736	0.88633728	0.00032721
16	0.83790588	0.84022522	0.00276802
17	0.66554260	0.66249084	-0.00458537
18	0.63362122	0.63047791	-0.00496087
19	0.83793640	0.83621216	-0.00205772
20	0.51921082	0.51724243	-0.00379112
21	0.74530029	0.74406433	-0.00165834
22	0.64778137	0.64349365	-0.00661909
23	0.82815552	0.82666016	-0.00180565
24	0.81906128	0.81735229	-0.00208652
25	0.89059448	0.89187622	0.00143920
26	0.75799561	0.74935913	-0.01139384
27	0.77224731	0.77180481	-0.00057300
28	0.51884460	0.51873779	-0.00020586
29	0.77928162	0.78125000	0.00252589
30	0.60072327	0.59919739	-0.00254007
31	0.87872314	0.87242126	-0.00717163
32	0.80784607	0.80752563	-0.00039666
33	0.90097046	0.90101624	0.00005081
34	0.76953125	0.76795959	-0.00204236
35	0.69288635	0.69262695	-0.00037438
36	0.60630798	0.60368347	-0.00432867
37	0.93629456	0.93644714	0.00016296
38	0.59843445	0.59674072	-0.00283027
39	0.95690918	0.95632935	-0.00060594
40	0.86485291	0.86305237	-0.00208190
41	0.82998657	0.83239746	0.00290473
42	0.77384949	0.77331543	-0.00069013
43	0.73242188	0.73306274	0.00087499
44	0.56666565	0.56251526	-0.00732423
45	0.95986938	0.96038818	0.00054049
46	0.75393677	0.75198364	-0.00259058
47	0.86323547	0.86335754	0.00014141

表 4 48 图模型测试面积比例预测结果及相对误差[复赛]

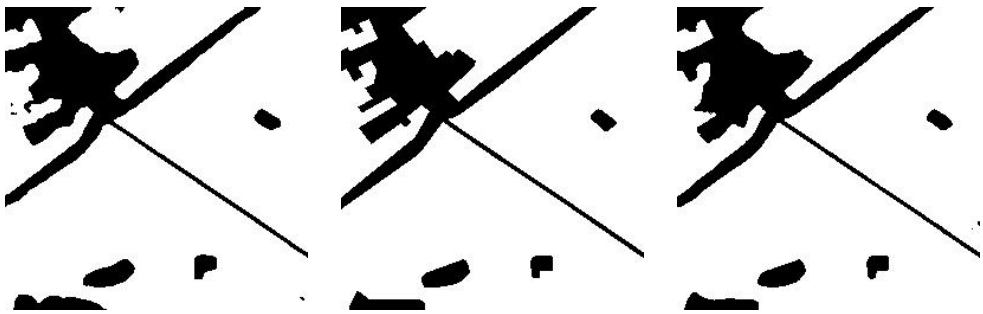


图 9-1 噪声问题优化（左至右依次为优化前、预测标签、优化后）

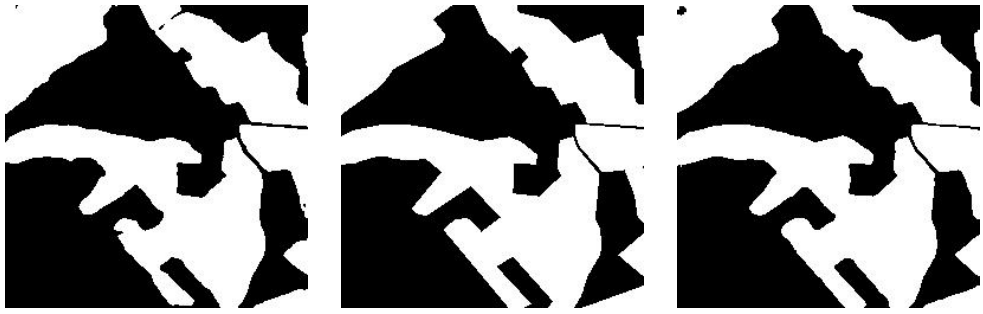


图 9-2 边界问题优化（左至右依次为优化前、预测标签、优化后）

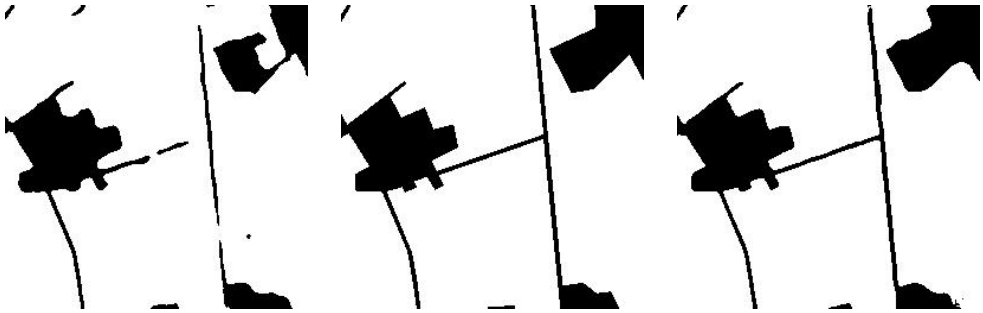


图 9-3 道路问题优化（左至右依次为优化前、预测标签、优化后）

{图 9}图 9-1 中旧模型预测图图右上角的黑色噪声得到优化；图 9-2 中旧模型预测图整体边界得到优化，中下部边界的毛刺消失；图 9-3 中旧模型预测图道路得到优化，交叉处连接明显，道路清晰。

需要注意的是，尽管相比于初赛结果，上述模型预测结果优化结果明显，但是该模型的预测结果依然会存在上述噪声、边界和道路问题，如图 10。因此，需要利用下文中的“多图平均二值法”和“腐蚀与膨胀”方法对预测结果进一步处理，得到更为理想的耕地标签图。

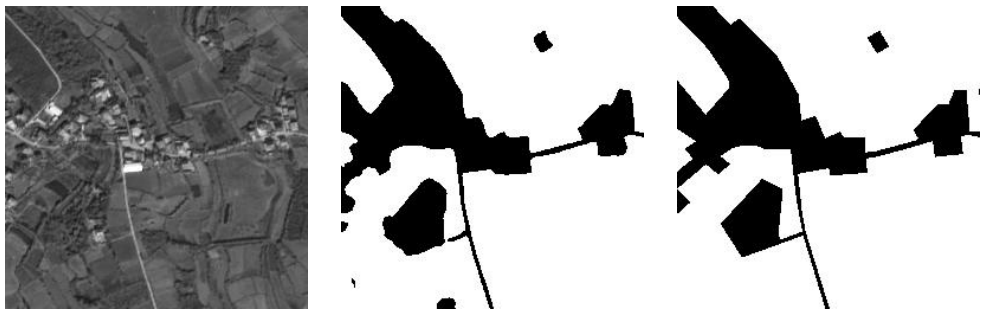


图 10 新模型问题图像（左至右依次为遥感图、预测标签、人工标签）

{图 10}从图中可以明显看出，预测标签左侧中部边缘有毛刺，左下角有黑色噪声，中间偏左下位置道路出现间断。最下方出现了可能的错误预测结果。

5.2.6 多图平均二值法：预测效果整体提升

“多图平均二值法”为我队针对卷积网络初步预测结果后续处理而提出的方法。该方法在我队的初赛问题求解中起到了重要的作用，极大地提升了预测标签的精确度。本文将继续使用“多图平均二值法”求解问题。在初赛的基础上，本文进一步研究了“多图平均二值法”的基本原理、“多图平均二值法”与数据准备和基本模型的关系、“多图平均二值法”的使用条件等问题，进一步完善“多图平均二值法”的相关内容。

(1)方法背景

1 卷积

卷积[5]作为分析数学中的一种重要运算，被广泛应用于图像处理。神经网络的卷积操作通常是通过使用卷积核完成的。卷积核[6]是一个函数，定义了图像某区域像素的加权权值。图像处理中的卷积操作通常是对输入图像进行顺序采样，将输入图像的小区域中像素加权平均，结果作为输出图像的对应像素值，这里的权值就由卷积核定义。由于卷积操作是顺序采样，权值与像素点对应，将一普通方形图片若以不同方向输入，会得到不同的采样结果。因此，当旋转、镜像后的图像仍适宜作为输入图像时，多次旋转输入图像得到的预测结果互不相同，但都属于对原图的合理预测，具有参考价值。基于以上特点，本文将同一图像经多次给定旋转和镜像操作输入上文中训练好的 U-Net 模型，得到多个不同预测结果。

2 集成学习

集成学习通过将多个学习器进行结合，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。[7]

本文在仅使用 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。以下说明优化过程及其原理。

(2) 过程及数学原理

1 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。

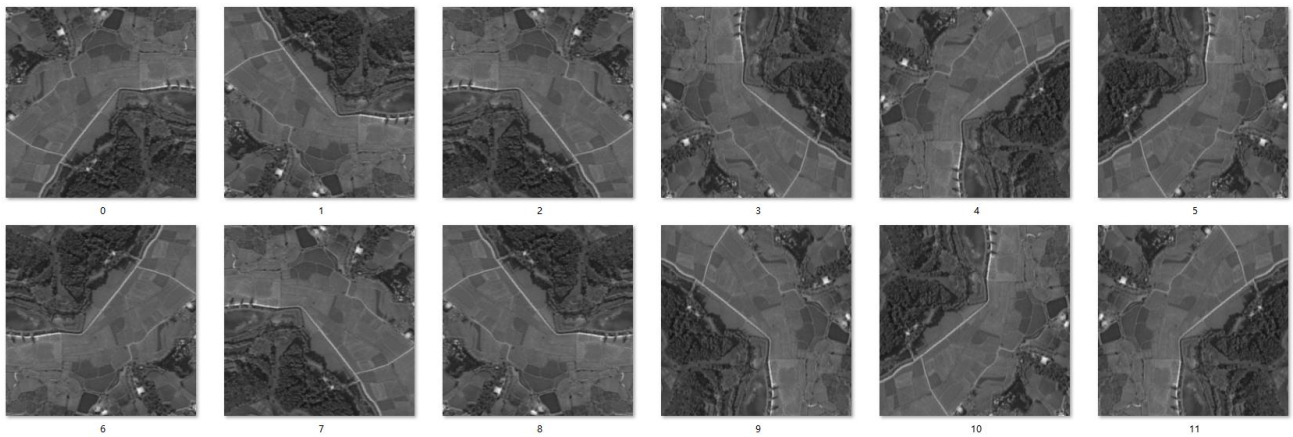


图 11-1 测试集 12 图处理结果[初赛模型]

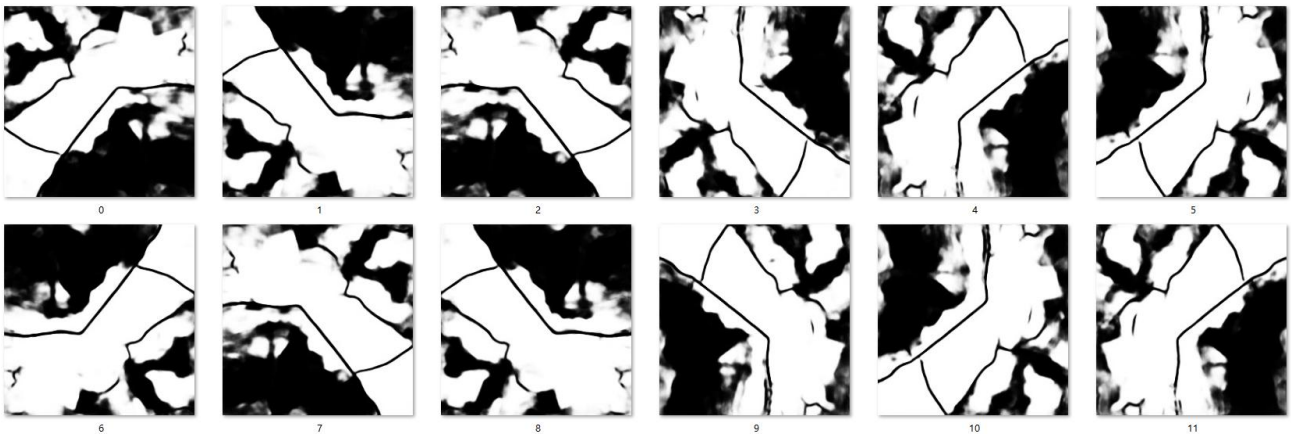


图 11-2 测试集 12 图预测结果[初赛模型]

{图 11}图像来自于初赛模型(旋转范围[0, 10°])对 Test4 的 12 图预测，该预测结果相比于后文新模型结果图 13 明显不理想。选取该结果是为了与图 13 构成对比，进一步说明随机旋转范围与“多图平均二值法”配合得到良好的预测结果。

需要注意的是，虽然称之为“12 图”操作，但实质上只生成了 8 张不同的图像。由于在模型训练时采用旋转、镜像操作，所以模型针对不同角度图像的预测能力参差不齐，12 张图的处理同样图片的重复出现正模拟了模型的训练操作，可看做不同角度图像结果平均化权值不同。另外，由于训练数据的旋转范围决定了模型对不同角度图像的预测能力，训练时的旋转范围需要与多图平均法相配合。原理如图 12。

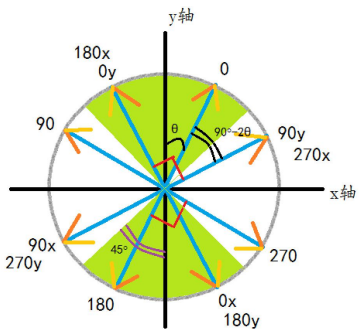


图 12 多图平均二值法原理图

{图 12}图中绿色区域代表了训练集旋转、镜像范围在[0, 45°]度时对某特征(假设该特征方向为 y 轴正向)处理精度较高的位置，即学习到的角度范围。彩色箭头代表了测试集图像某特征位置以及其旋转、镜像后对应的该特征位置。箭头旁的注释表示“旋转角度 镜像轴”，0 代表操作前该图像特征的位置。可以看出，处理后得到的 12 张图实际仅有 8 类。θ 代表了测试图像与训练图像的特征区别。范围取 45°时，当较小(与 y 轴正向夹角[-45°, 45°])，随 θ 变化总有 6 张图在绿色区域内。

分析图 12 可知，为了得到在模型预测精确范围(绿色内)的多张图的预测结果，即保证测试集旋转、镜像后的 12 张图中总有落在高精度区域内的，旋转范围最小选取[0, 45°]，即当 θ=45°，90°-2θ=0°，箭头刚刚好图像落在范围边缘。另外，从图中可以看到，范围取[0, 45°]时，当 θ 较小(与 y 轴正向夹角[-45°, 45°])，随 θ 变化总有 6 张图(0、0x、0y、180、180x、180y)在绿色区域内，预测结果优秀。这 6 个方位的图片的训练结果在初赛中经过观察，明显优于另外六个方位，为上述理论提供实验论据。因此，由于数据准备旋转范围越小模型需要学习到的特征相对越少，精度较高，我们选择与“多图平均二值法”相适应的最小范围[0, 45°]作为随机旋转范围。以光线特征(把图 12 中的箭头看做图中物体影子方向)为例，考虑数据训练集和测试集特征差别不大，到针对上述特点，本文选用 0、0x、0y、180、180x 和 180y 六张图进行多图平均二值操作。

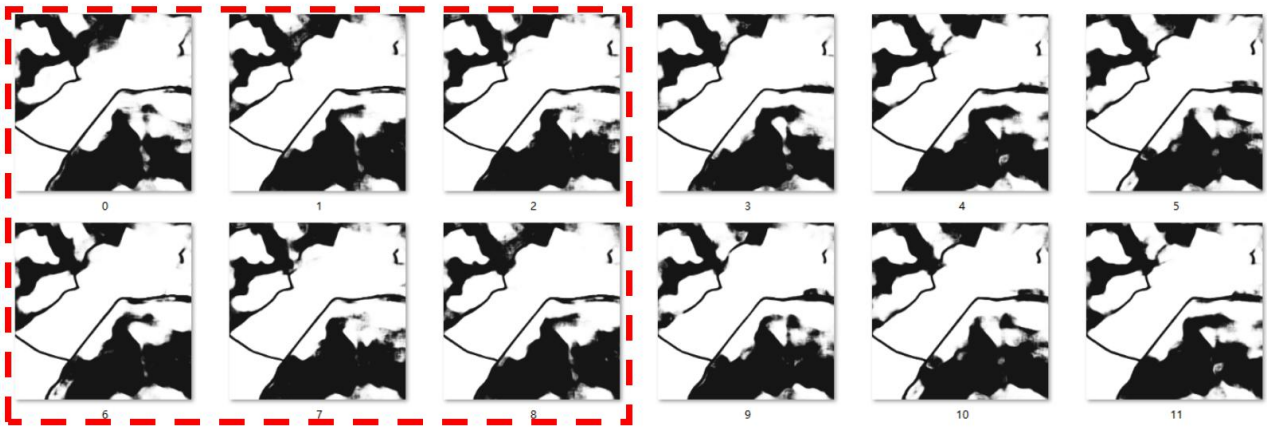


图 13 测试集 12 图预测结果及 6 图选取[复赛模型]

(上至下，左至右依次为 0 0x 0y 90 90x 90y 180 180x 180y 270 270x 270y)

{图 13}图像来自于复赛模型(旋转范围[0,45°])对 Test4 的 12 图预测，与图 11 对比可知，新模型的 12 张图从整体上明显优于旧模型，表现在图与图之间相似性高，结果较为清晰。红色虚线框内所选的六张图为下面多图平均二值操作用图。可以明显看出左边六张图较右边六张图较为清晰理想。

2 多图平均二值操作

普通的均值思想得出的结果只能保证结果优于最次，却不能做到优于最优，本文中提出的“多图平均二值”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。

本操作中，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最终的二值化判断。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，该方法对模型的预测精度要求低，明显减少了错误点数目，相比于集成学习方法减少了训练多个模型所消耗的时间。降低了损失(loss)，提高了准确率(accuracy)。结合人工技术性筛选(选图不同角度输入图的预测结果图)更能提高最终结果准确性。

① 多图平均融合

将处理好的数据（test3、test4 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张（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。

$$y_1 = x - 0.5$$
$$y_1 = \sum_{i=0}^{11} y_i / 12$$
$$y = y_1 + 0.5$$

其中 y、y_i 分别为“整合”图片的像素值和第 i 张子图的像素值，x 为归一化图片像素值。

② 二值化

将 y 图再次归一化并二值化输出结果。

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

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

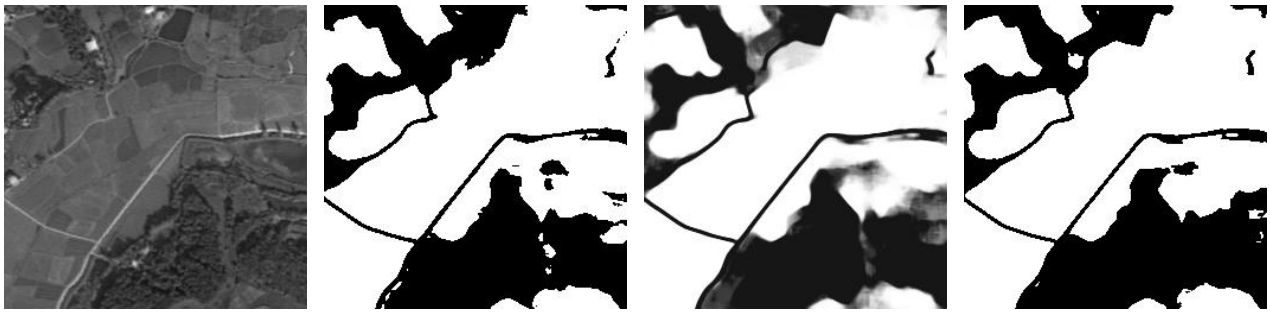


图 14 多图平均二值操作流程[复赛模型]

{图 14}图像是对复赛模型(旋转范围[0,45°])对 Test4 预测图的多图平均二值处理。从左至右，图像依次是遥感图、未进行多图平均二值优化的结果图、平均融合操作成图和对融合结果的二值化操作成图。观察图像可知，经过“多图平均二值法”处理生成的结果图效果要明显优于未经过“多图平均二值法”处理生成的结果图。噪声明显减少，边缘更加圆滑。

3 拼接成图

对进行过上述操作的小图(256x256)，根据其裁剪方式，将图片拼接成大图(500x600)。

(2) 配合新 U-Net 模型的实际运行测试

利用上述“多图平均二值法”我队对测试集在新模型的预测结果进行优化，优化提升后的数据的误差点数目和准确率，如表 5 所示。

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	3663	2989	0.98779000	0.99003667	-674	22.47
Data2	1935	1404	0.99355000	0.99532000	-531	17.70
Data3	3017	2219	0.98994333	0.99260333	-798	26.60
Data4	2947	2150	0.99017667	0.99283333	-797	26.57
Data5	2993	2293	0.99002333	0.99235667	-700	23.33
Data6	2136	1665	0.99288000	0.99445000	-471	15.70
Data7	2453	1931	0.99182333	0.99356333	-522	17.40
Data8	2240	1497	0.99253333	0.99501000	-743	24.77

表 5 “多图平均二值法”优化前后结果与人工标签像素点差距[复赛]

由表 5 可以得出，“多图平均二值法”的使用减少了错误像素点的数目，进而提高了预测精度。另外，将表 5 与表 2 对比可知，新模型在优化前的的预测结果精度高于旧模型优化后的精度，而且在“多图平均二值法”后精度再次上升且依然提高明显。这表明了“多图平均二值法”不仅仅是对精度低的模型提升明显，对于精度高的模型依然能起到较为优秀的优化效果。

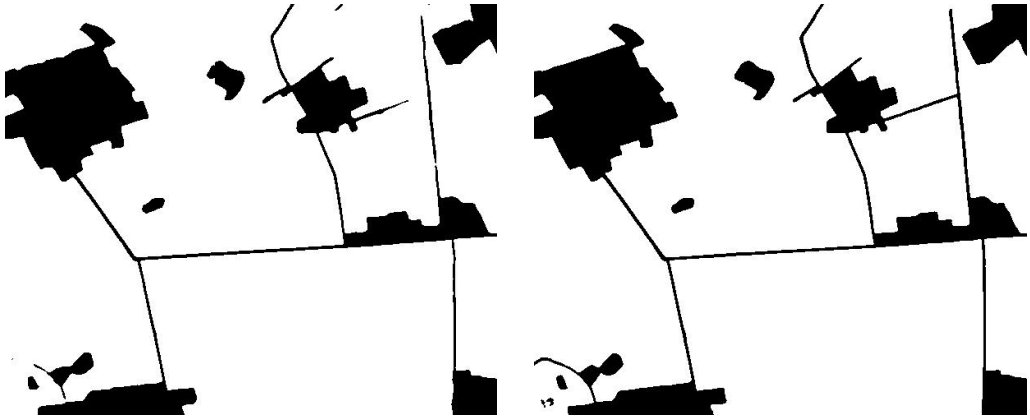


图 15 新旧模型“多图平均二值法”优化前后结果对比(左旧右新)

{图 15}由上图，很明显可以观察到噪声减少,边缘圆滑，道路更清晰连贯。左图左下和右上间断的道路在右图中明显联通，右图的边缘明显比左图更为圆滑、毛刺少，噪声更少。将模型训练数据准备与“多图平均二值法”配合优化使得预测结果明显提升。

值得注意的是，在图 15 右图左下角依然存在噪声问题，因此本文针对此类问题，利用腐蚀与膨胀的方法继续对模型进行优化。

5.2.7 腐蚀与膨胀的尝试：视觉效果提升(修补道路、圆滑边界、减少噪声)

腐蚀与膨胀是图片处理中用以去除图片噪声、圆滑边界线条的常用操作。腐蚀可以增加二值图像中 0 部分，膨胀可以增加二值图像中 1 部分。开操作和闭操作是腐蚀与膨胀在图像处理中的常见应用。开操作指先执行腐蚀操作，再执行膨胀操作；闭操作指先执行膨胀操作，再执行腐蚀操作。[9]本文尝试使用开操作和闭操作对上文预测结果进一步处理,但结果显示,该方法虽然使得测试集预测图视觉效果提升，但同时测试集预测面积的准确度有微弱下降。

在多图平均二值法基础上进行膨胀腐蚀效果						
	错误像素点个数		准确率		效果下降	
	平均二值	膨胀腐蚀	平均二值	膨胀腐蚀	误差点	准确率(*10^-4)
Data1	2989	2998	0.99003667	0.99000333	9	-0.33
Data2	1404	1419	0.99532000	0.99527000	15	-0.50
Data3	2219	2239	0.99260333	0.99253667	20	-0.67
Data4	2150	2169	0.99283333	0.99277000	19	-0.63
Data5	2293	2415	0.99235667	0.99195000	122	-4.07
Data6	1665	1673	0.99445000	0.99442333	8	-0.27
Data7	1931	1937	0.99356333	0.99354333	6	-0.20
Data8	1497	1503	0.99501000	0.99499000	6	-0.20

表 6 腐蚀与膨胀降低测试集预测准确率

{表 6}该测试是对测试集“多图平均二值法”结果进行统一腐蚀膨胀操作得出的。由表可知腐蚀膨胀对面积精度造成了影响，但影响较小，精确度依然高于 U-Net 模型的直接结果。而且，由于分辨率为 2m，每张图面积误差基本在 80 平方米以内，几乎可以忽略。另外，从上表数据得知，腐蚀膨胀程序的泛化能力并不优秀，不适合大批量操作。

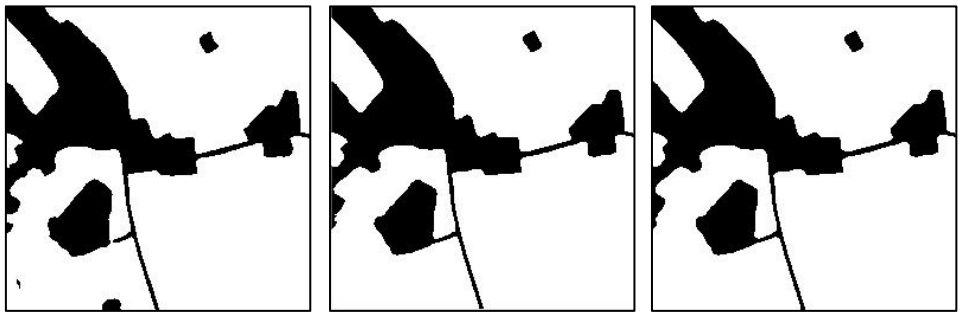


图 16 优化前后结果对比(左至右，模型结果、平均二值、腐蚀与膨胀)

从视觉效果看，腐蚀与膨胀后的结果较为优秀，边界较为清晰。但就面积精度而言，在平均二值的基础上进行腐蚀与膨胀降低了预测面积精度。

分析原因后我们认为，面积精度的降低可能由于人工标签标注没有精确到像素点或者前期模型测试准确度已经理想，难以提升。虽然腐蚀与膨胀的操作会对图像的面积精度造成一定的影响，但在可接受范围内。“多图平均二值法”后对于 Test 图的预测仍有明显的提升空间。针对性的进行腐蚀与膨胀操作可以使面积精度仅发生小幅度波动，同时显著提升标签图的视觉效果。另外，通过后文参数分析，我们发现上述面积精度变化可控。

鉴于以上考虑，我们选择在“多图平均二值法”基础上再针对问题图制定腐蚀与膨胀方案，优化标签图，在使面积精度提高或尽量少的避免面积精度降低的同时，提升视觉效果。

5.2.8 Test3、Test4 图片耕地标签图制作

本文利用上述模型和优化方法，对 Test3 和 Test4 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。相比于操作前的图，优化明显。

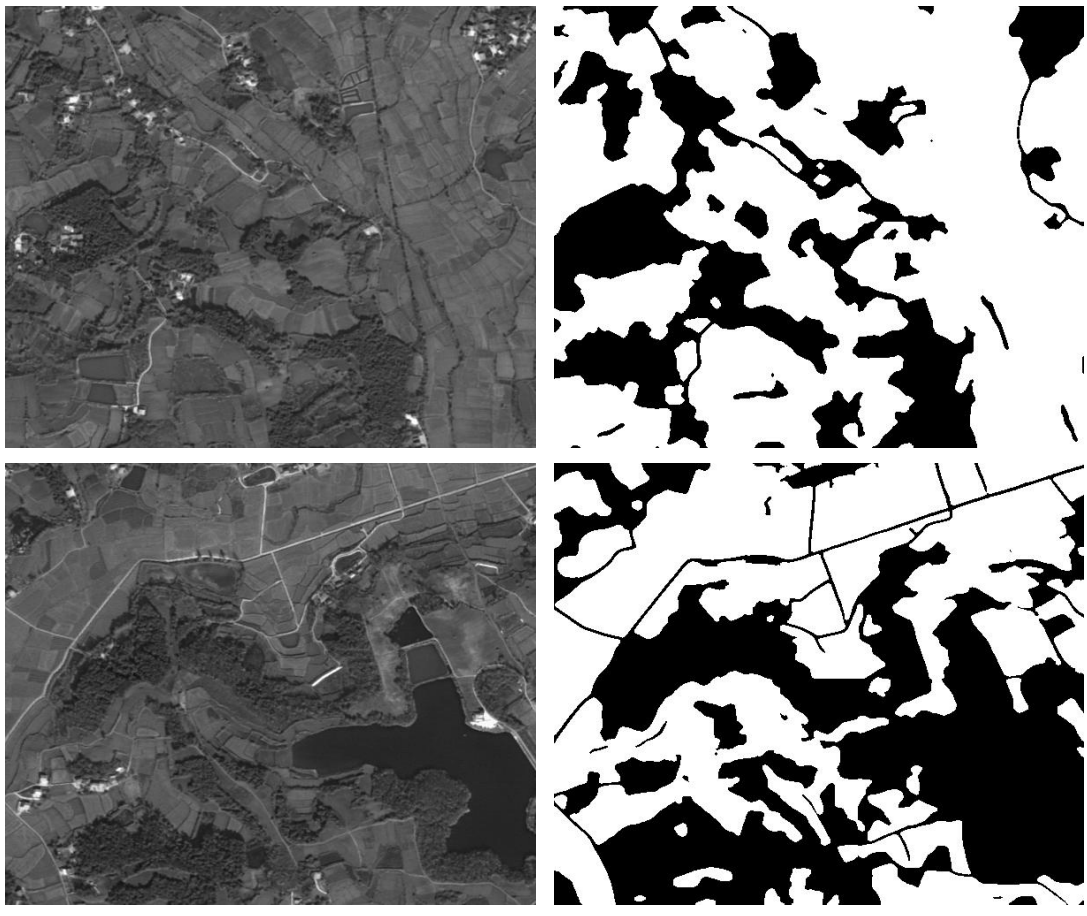


图 17 Test3 和 Test4 进行耕地标签图预测

5.2.9 Test3、Test4 图片耕地面积比例计算

得到较为理想的耕地标签图后我们利用初赛的方法计算 Test3 和 Test4 的耕地面积比例。

(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

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

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

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。

(2) 结果

数据编号	耕地所占比例
Test 3	0.69787333
Test 4	0.52884333

表 7 耕地在各图像中所占比例

5.3 问题二求解——参数评估

初步分析，模型训练参数和腐蚀膨胀操作的核参数对预测精度起关键作用。模型中训练参数对模型效果的影响主要体现在训练过程中损失(loss)和准确率(accuracy)随迭代次数的变化和最终结果。腐蚀膨胀操作核参数的尺寸、形状、锚点均对预测面积精度产生的影响。上述参数对精度的影响具体情况将在下文说明。

5.3.1 训练参数对模型预测精度的影响

随机旋转角度	迭代次数	损失	准确率
10°	100	0.03704853	0.98499188
90°	100	0.13277034	0.97246501
90°	150	0.04638607	0.98106054
90°	200	0.03628405	0.98496221
90°	250	0.05648167	0.98707241
90°	300	0.02831206	0.98822424
45°	300	0.04129387	0.99021984

表 8 Test3 和 Test4 进行耕地标签图预测

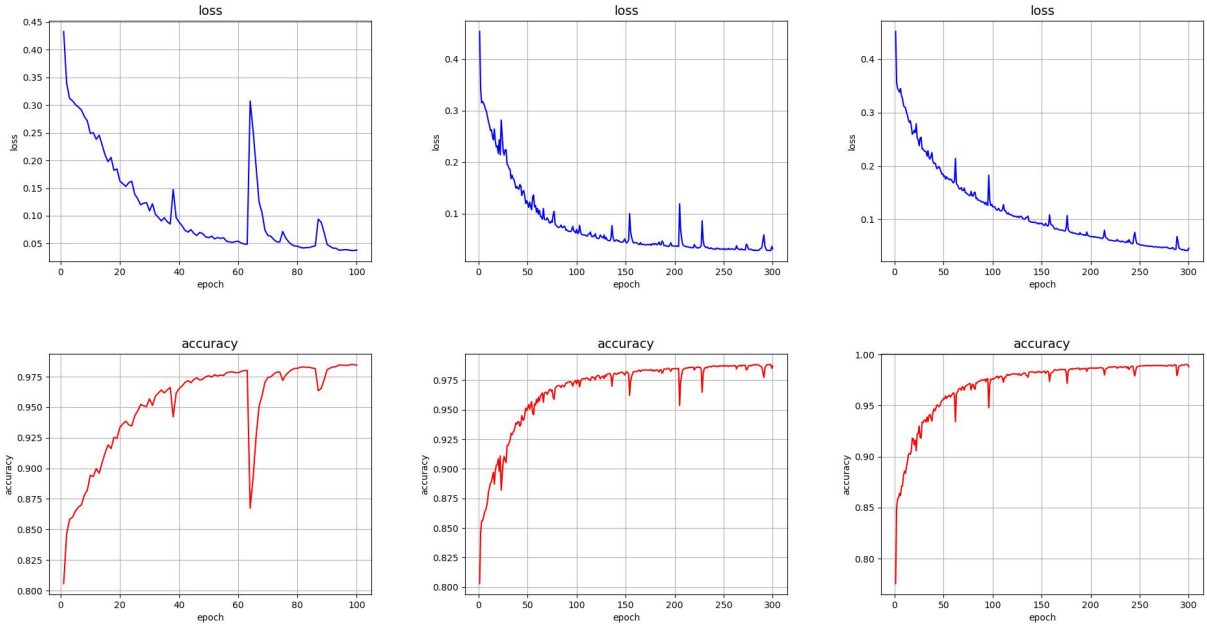


图 18 训练过程(左至右 10° 100 轮、90° 300 轮、45° 300 轮)

(1)数据准备随机旋转角度与训练精度的关系

为了分析上述关系，我们对随机旋转角度为 10°、45°、90°时的训练结果做定量分析，并对比相应模型的预测精度。

由表 8 可以看出随机旋转角度越大，训练越不容易达到饱和，损失较高，准确率较低。同时，不适合的随即旋转角度会使得模型发生“过拟合”或“欠拟合”。由图 18 可以看出，10° 100 轮训练时，模型在约 63 次位置存在明显突变，可能发生了“过拟合”。90° 300 轮训练中模型整体趋势还未完全趋于稳定，需要继续增加迭代次数才能使训练得到饱和，训练开销大。相较其余二者，45° 300 轮的训练过程较为合适，整体变化较为平稳，最终模型趋于稳定，即变化率趋于 0。同时，由于 45° 的参数选取是基于“多图平均二值法”原理(详细原因在上文中已给出)，45° 300 轮为本次问题求解较为合理的参数选择。

数据准备随机旋转角度与训练精度的关系可以总结为以下几点：

1 旋转范围越大，饱和所需迭代次数越多，越不易达到高的训练精度，容易欠拟合；

2 旋转范围过小会增加模型“过拟合”的可能性，且模型稳定性差；

3 合适的旋转范围可在保证模型具有良好预测精度、泛化性和稳定性的同时，节约训练开销。

(2)模型训练迭代次数与训练精度的关系

由表 8 可知，一定范围内，训练精度随迭代次数上升而上升，会有微小波动。结合图 18 的训练过程可知，过大的迭代次数会导致模型“过拟合”，进而导致精度突变降低。

从上所述，模型预测精度与迭代次数息息相关，能有效避免“欠拟合”和“过拟合”的发生是选择训练迭代次数合适的关键。

5.3.2 腐蚀与膨胀核参数对面积精度的影响

腐蚀、膨胀通过核的移动与加权平均计算来实现图片去除噪声、圆滑线条等操作。其中，核是一个行列通常为奇数(如 3x3，5x7 等)的矩阵，常用结构有矩形、椭圆和十字交叉形。核中存在锚点，用于定义核矩阵中心点的位置，规定了形态学处理结果的偏移量。本文的耕地标签问题不适宜形态学偏移处理，因此本文中锚点均定义在矩阵中心，即无形态学偏移。本文从模型 48 图预测结果中选出三张面积误差大的图像(见表 9)来测试不同腐蚀膨胀操作对面积精度的影响。下文中主要归纳了核的尺寸形状与开闭操作对预测面积精度的影响。[9]

48图test预测结果问题图像				
图片名称	标签	测试结果	相对误差	错误像素
41	0.82998657	0.83239746	0.00290473	718
16	0.83790588	0.84022522	0.00276802	946
26	0.75799561	0.74935913	-0.01139384	926

表 9 48 图预测主要问题图像

{表 9}表中给出了模型对 48 图的预测结果的主要问题图像，用于后文测试腐蚀与膨胀操作效果。

通过改变核的尺寸和形状、腐蚀与膨胀的实现手段，本文对十字结构(1, 3) (3, 1) (3, 3) (5, 5) (7, 7)、椭圆结构(3, 3) (5, 5) (5, 7) (7, 5) (7, 7)、矩形结构(3, 3) (5, 5) (7, 7)的 13 种核类型分别在开、闭操作下生成的对 41 号图、16 号图、26 号图的预测结果与人工标签对比的错误像素点个数和相对误差进行了计算，计算结果见表 10。

		开操作					
核		41		16		26	
形状	尺寸	错误像素	面积误差	错误像素	面积误差	错误像素	面积误差
十字	(1, 3)	709	0.00244512	952	0.00258591	928	-0.01163540
	(3, 1)	720	0.00264735	950	0.00262233	926	-0.01171592
	(3, 3)	706	0.00257381	945	0.00256770	932	-0.01159514
	(5, 5)	699	0.00193036	960	0.00193033	970	-0.01264192
	(7, 7)	699	0.00112145	986	0.00109264	1022	-0.01413157
椭圆	(3, 3)	706	0.00257381	945	0.00256770	932	-0.01159514
	(5, 5)	703	0.00148913	969	0.00180285	944	-0.01248088
	(5, 7)	687	0.00148913	971	0.00162075	965	-0.01258153
	(7, 5)	735	0.00023900	999	0.00107443	954	-0.01308479
	(7, 7)	691	0.00086407	981	0.00132938	963	-0.01298414
矩形	(3, 3)	708	0.00213259	956	0.00244022	928	-0.01195748
	(5, 5)	734	0.00033092	1019	0.00092874	963	-0.01278283
	(7, 7)	814	-0.00187521	1189	-0.00282265	1011	-0.01431275

		闭操作					
核		41		16		26	
形状	尺寸	错误像素	面积误差	错误像素	面积误差	错误像素	面积误差
十字	(1, 3)	723	0.00299665	941	0.00293191	925	-0.01117240
	(3, 1)	727	0.00314373	948	0.00316865	910	-0.01103148
	(3, 3)	721	0.00303342	939	0.00315044	909	-0.01097109
	(5, 5)	1057	0.00987241	1560	0.01595251	1313	-0.00046300
	(7, 7)	2932	0.04816708	2053	0.02675141	1493	0.00489170
椭圆	(3, 3)	721	0.00303342	939	0.00315044	909	-0.01097109
	(5, 5)	2226	0.03331250	1928	0.02298181	1299	-0.00082535
	(5, 7)	2881	0.04686179	1991	0.02489392	1467	0.00291891
	(7, 5)	2920	0.04802000	2039	0.02569519	1305	-0.00022143
	(7, 7)	2984	0.04912307	2061	0.02627793	1486	0.00418713
矩形	(3, 3)	749	0.00354819	1045	0.00526287	903	-0.01068927
	(5, 5)	2883	0.04719271	2050	0.02618688	1461	0.00360335
	(7, 7)	3053	0.05072251	2182	0.02950121	1530	0.00615992

表 10 核参数及开闭操作的选取对面积精度的影响

{表 10}上表给出了不同核参数、开闭操作处理图像面积精度。其中浅绿色标记操作结果优于表 9 中未处理结果。深绿色标记操作的结果表现最为优秀。黄色标记结果面积精度显著提升，面积误差降低，但错误像素点增加。

通过分析表 10 并将表 10 结果与表 9 中数据对比可知，整体效果上开操作比较适用于修复对耕地面积预测偏大的图片(41 号图和 16 号图)，闭操作适用于修复对耕地面积预测偏小的图片(26 号图)，而且原图预测偏差越大，操作后面积精度提升越明显。用同样的操作手段处理同一张图，不同的核预测结果不同。核尺寸越大，图像面积精度变化越大，不同长宽也对面积精度有影响。如表 10 椭圆 41 号图开操作中，随核尺寸增加，面积误差先降低后升高，椭圆(5, 7)表现最为优秀。不同核形状对预测结果影响不同，同样核大小，十字结构处理结果面积精度普遍变化较小，椭圆其次但结果常常最优秀，矩形变化最大。从宏观上看，空间分辨率在 2m 的情况下，大部分操作前后面积变化极其微弱，基本在 200 平方米以内。因此，鉴于

上述特点，在不确定优化是否合适时，应采用尺寸较小的椭圆或十字结构核，根据视觉效果选择具体操作。

由表 10 计算结果可知，椭圆(5,7)开操作处理 41 号图结果优秀，椭圆(3,3)开操作处理 16 号图结果优秀，矩形(3,3)开操作处理 41 号图结果优秀。本文对比操作前后的图像，分析腐蚀膨胀对视觉效果的改变。

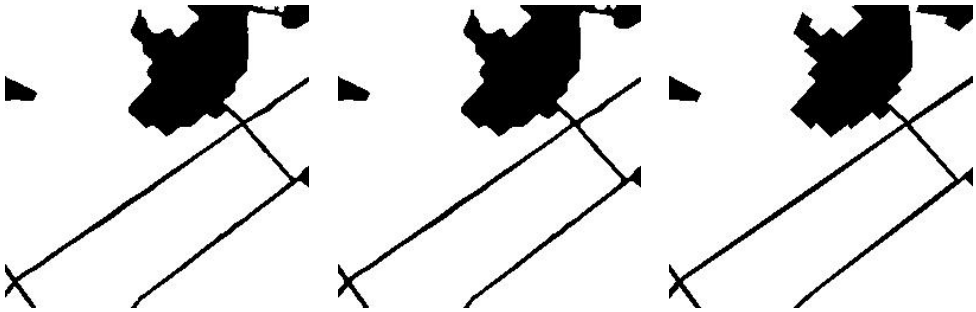


图 19-1 41 号图 椭圆(5,7)开操作结果(左至右 操作前、操作后、人工标签)

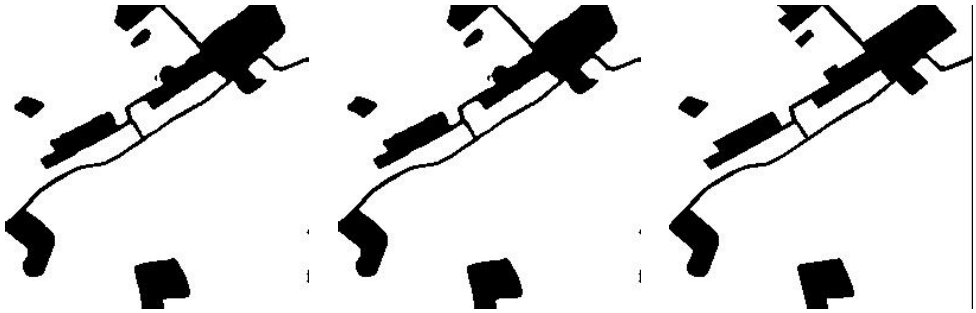


图 19-2 16 号图 椭圆(3,3)开操作结果(左至右 操作前、操作后、人工标签)

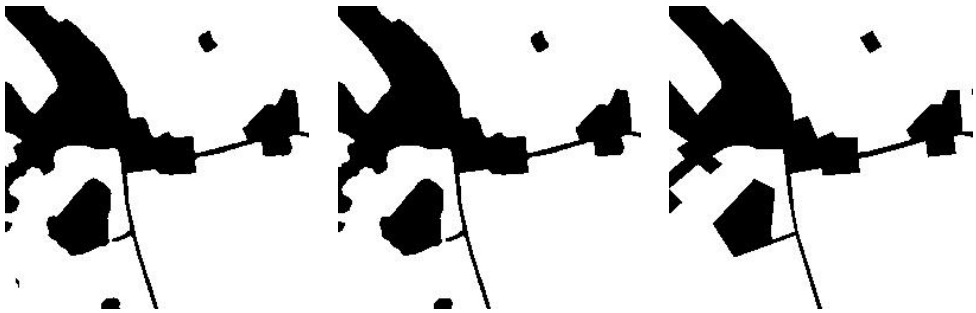


图 19-3 66 号图 矩形(3,3)闭操作结果(左至右 操作前、操作后、人工标签)

总的来讲,腐蚀与膨胀操作对面积精度的提升主要表现在使边界更加平直,去除部分噪声,可连接少数道路。图 19-1 右上角白斑减小，图 19-3 左下角噪声消失。操作后视觉效果明显提升，与人工绘制标签图较为相似。通过上述分析，核参数对面积精度的影响及其使用条件可以总结为以下几点：

1 预测图耕地面积偏大适合开操作，偏小适合闭操作；

2 核形状十字、椭圆对面积精度影响小，矩形影响大，实验效果显示椭圆最为合适，十字结构可能会将狭窄的道路错误抹去；

3 核尺寸越大，面积精度变化越大，一般选取尺寸较小的核；

4 腐蚀与膨胀操作针对性较强，不适合对大批量差别大图片同时使用相同操作；

5 原图效果越差，合适的腐蚀膨胀操作提升效果越好；

综合考虑上述关系及耕地图片特点(预测面积偏大、道路多、边界毛边多、白色噪声多)，本文认为处理耕地图片，使用小尺寸椭圆核进行开操作较为合适。处理前后效果见附录 4，视觉效果提升明显。

六、模型评价

6.1 优点和缺点分析

6.1.1 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，圆滑边界，修复道路上表现突出。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。配合合理的模型训练方法，效果更为理想。
- 本模型采用腐蚀膨胀操作。该操作显著地减少了黑白色块中细小的斑点，在道路纤细处连接两段路和平滑较大面积耕地边界发挥了重要的作用。该方法弥补了模型预测结果中出现细小的、难以去除的噪声的影响，让预测结果更符合实际。
- 优化后模型相比于优化前的模型分割精度、稳定性和泛化性均有所提升。在对噪声问题、边界问题和道路问题的处理上表现更为良好。模型消除了道路的部分断点、边界的毛刺和部分噪声点，得到视觉效果更为优秀的耕地标签图。

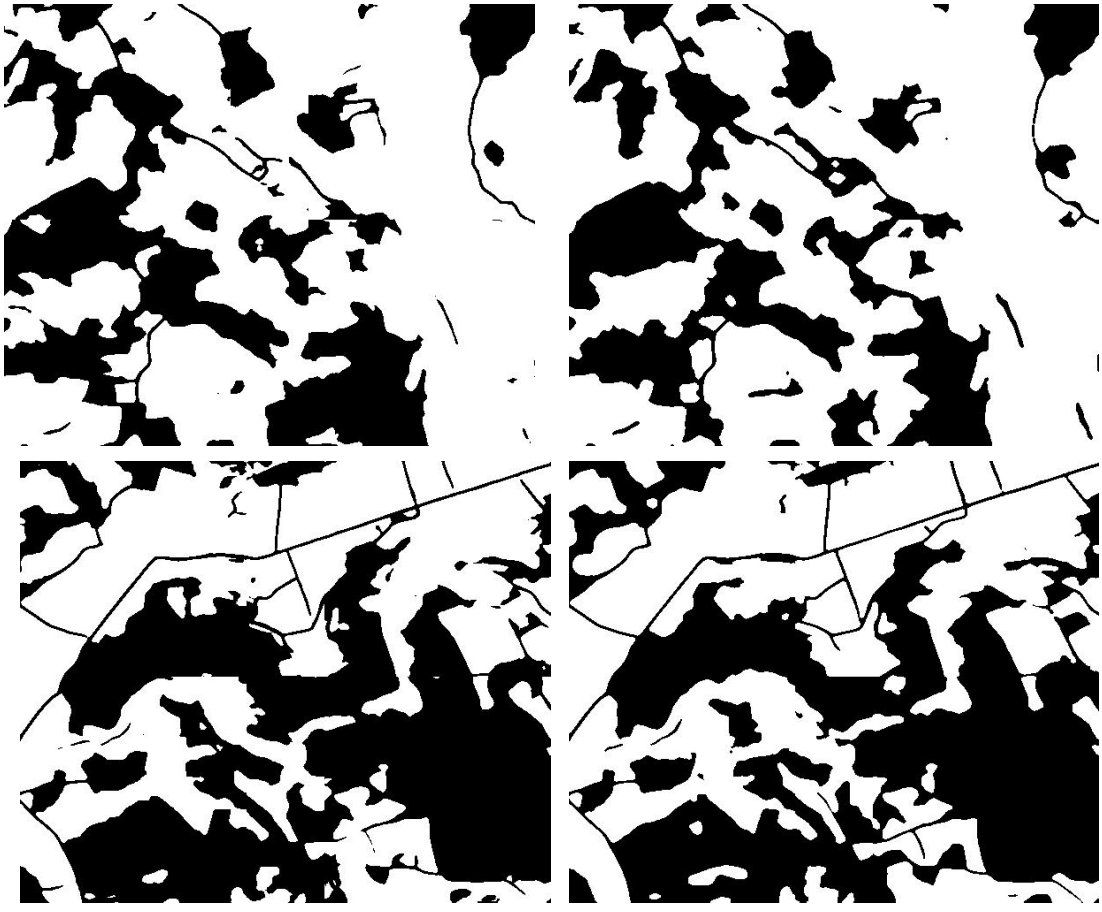


图 20 初赛复赛优化结果对比(左前右后)

{图 20}上图为初赛模型和经过全部优化的本文模型对 Test3 和 Test4 预测结果的对比。由上图可知，复赛模型相对于初赛模型优化明显，体现在噪声减少、边界圆滑、道路连接。

6.1.2 缺点

- 虽然优化后模型可以区分一些灰度和灰度变化与耕地地块均较为相近的色块,但对少数池塘等色块依然存在问题,需要更多的池塘与耕地混杂的数据集进行训练。对于种植有绿化树的细小的田间道路,因其与田埂较为相似,本文模型不能很精确的识别、分割,若在训练集中添加田间绿化树的图片能够得到进一步优化。
- 腐蚀与膨胀操作针对性强,程序泛化性差。需要根据图片具体特点设计程序,不合适的操作有降低面积精度的风险。

6.2 模型创新点及应用前景

本文模型创新了预测思路,提出“多图平均二值法”。将图像旋转多个方向输入模型进行预测并将预测结果做平均和二值化操作得到综合的预测结果。该方法快速、简便、高效,极大地降低了模型预测误差,也提升了预测速度。运用在医疗检测图像分析,该方法可极大减小机器学习对图像的判断误差,让医生可以更依赖于智能诊断[10];运用在工程领域,该方法也可以优化并精确目标点位置,并消除背景对物体轨迹的影响[11]。

基于“多图平均二值法”的原理,本文创新性地提出控制数据扩增时图片旋转角度范围([0,45°])以提高模型精度。该方法使训练集图像的光影信息得到保留,让模型更有针对性的学习,使模型对同一时段的耕地图片更加敏感,对道路、森林等易误识别区域预测效果更好。同时,保证较为合适的训练开销。对于具有特定光影信息或类似方向性特征的图片,这种训练方式极大地提高了模型的针对性和精准性。例如监控图像的分割和信息提取中,针对特定机位设置特定随机旋转角度训练模型可得到更为理想的效果。

为了在保证预测面积精度的同时提升耕地标签图的视觉效果,本文创新性地使用了腐蚀膨胀操作,消除了斑块的干扰且保持了耕地地块的原本形状。该操作可以让耕地地块形状趋于合理化,减少了噪声干扰,对国家土地合理规划具有极大的借鉴意义。在因天气等客观因素致使的图片质量不高、噪点密集的图像中,运用腐蚀膨胀操作可以使图像更为清晰,有利于对图像的进一步分析处理。[12]

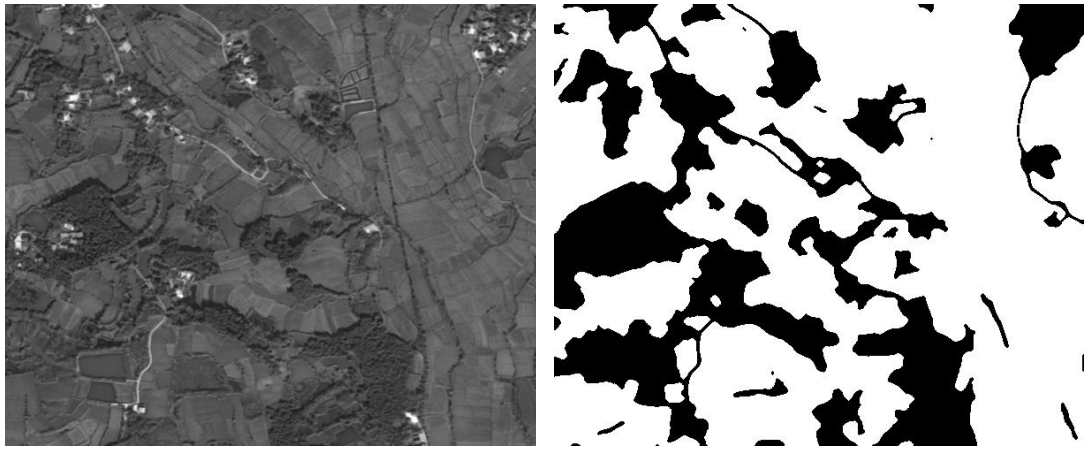
七、总结

在初赛模型的基础上,本文进一步探究初赛提出的“多图平均二值法”的合理性与使用条件,并依据其原理改进了数据增广方法和训练方法,对训练数据增广的随机旋转范围和训练迭代次数进行优化,使得训练出的U-Net模型预测精度显著提升。同时,为了在保证预测精度的同时提升耕地标签图的视觉效果,本文提出采用腐蚀膨胀操作对预测结果做进一步处理,详细分析了腐蚀膨胀操作的使用对面积精度的影响,选取合适操作处理图片,使得预测结果的视觉效果等得到显著提升。

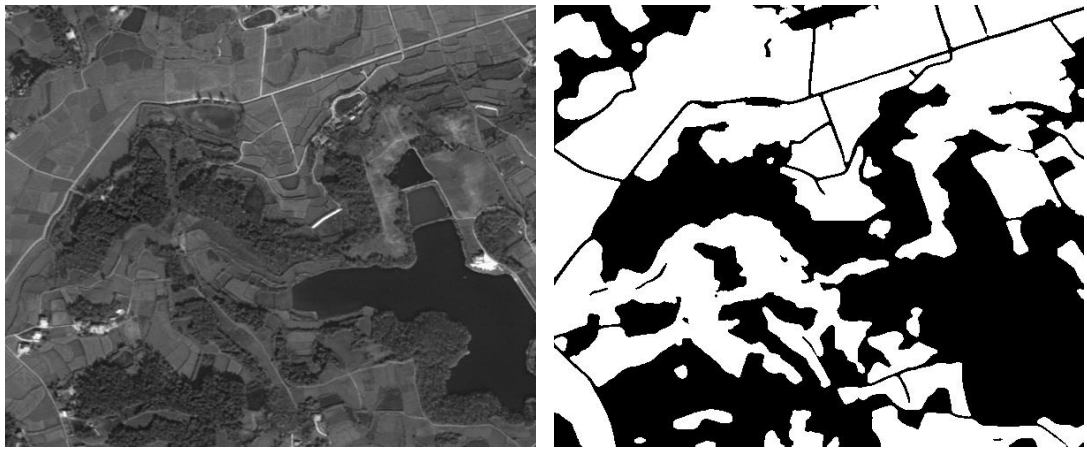
对比优化前后模型对Test3和Test4的耕地预测结果(图20)可知,经本文优化后的模型具有更为优秀的分割性能和更为良好的视觉效果。相比初赛模型,本文模型的预测图像耕地边缘更为清晰,噪声明显减少,部分道路间断处得到修复。除此外,本文模型具有良好的稳定性。因此,本文优化较为成功。

尽管本文优化使得模型性能得到显著提升,甚至可以区分一些灰度和灰度变化与耕地地块均较为相近的色块,但对少数池塘等色块的预测依然存在问题,需要强化原始图像信息增加训练数据集并对模型做进一步的改进。本文中训练的U-Net模型的准确率可达到99.02%,测试中准确率达98%,结合“多图平均二值法”和腐蚀膨胀操作,对Test3和Test4图片的预测结果如下图所示:

//Test3//



//Test4//



耕地面积比例计算结果如下表所示：

数据编号	耕地所占比例
Test 3	0.69787333
Test 4	0.52884333

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] 卷积

[EB/OL].https://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF/9411006?fr=aladdin&qq-pf

-to=pcqq.c2c
[6] 卷积核
[EB/OL].https://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF%E6%A0%B8/3377590?fr=aladdin&qq-pf-to=pcqq.c2c
[7] 周志华. 机器学习[M].北京:清华大学出版社, 2016: 182-183
[8] 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
[9] 形态学操作—膨胀与腐蚀（Dilation and Erosion）
[EB/OL].https://www.jianshu.com/p/a8ee3674c061, 2018-08-22
[10]吴茜,余永建,汪志.少样本条件下 CT 图像三维分割算法及其放疗应用[J].兰州文理学院学报(自然科学版),2021,35(01):65-70.
[11]周俊杰,陈黎,陈姚节.基于图像分割的水炮射流落点检测[J].计算机工程与设计,2020,41(11):3262-3268.
[12]付辉,吴斌,李林飞,张红英.基于暗原色先验的雾霾图像清晰度复原[J].计算机工程,2016,42(07):232-237.

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

<model.py>:unet 基本模型

<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

<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

<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。

<erode&dilate.py>:腐蚀膨胀操作程序

---

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=45, #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_300.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=300,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_300.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_300.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_300.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')
```

erode&dilate.py

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

for files in glob.glob('out1'+"/*_bi.png"):
    filepath,filename = os.path.split(files)
    img=cv2.imread(files)
```

```
    #GrayImage=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    ret,image=cv2.threshold(img,153,255,cv2.THRESH_BINARY)  #0,1 二值化
    print(image.shape)
#参数（核结构，核大小）
    kernel1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))  # 椭圆形结构
    #kernel2 = cv2.getStructuringElement(cv2.MORPH_CROSS, (1, 3)) # 十字结构
    #kernel3= cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))# 十字结构
#参数（开、闭操作，卷积核）
    image = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel1)#开操作除白噪声
    #cv2.imwrite('image1.png', image)

    #image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel2)#闭操作除黑噪声
    #cv2.imwrite('image2.png', image)

    #image = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel3)#开操作修路
    image = image[:, :, 1]
    print(image.shape)
    cv2.imwrite(filename, image)


#腐蚀函数
#cv2.erode(src, kernel, dst=None, anchor=None, iterations=None, borderType=None,
borderValue=None)
#膨胀函数
#cv2.dilate(src, kernel, dst=None, anchor=None, iterations=None, borderType=None,
borderValue=None)
#获取不同形状的结构元素，返回指定形状和尺寸的结构元素
#cv2.getStructuringElement(shape, ksize, anchor=None)
#参数 shape:表示内核的形状
#矩形：MORPH_RECT   十字形：MORPH_CROSS     椭圆形：MORPH_ELLIPSE;
#参数 ksize:是内核的尺寸(n,n)
#参数 anchor:锚点的位置
```

answer_tif.py

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

#将二值 png 转 tif
def tif_generator():
    print(' <.png to .tif >'+':')
    i=0
    for files in glob.glob("*.png"): #遍历 image 每个文件,后缀随前程序为_ori
```

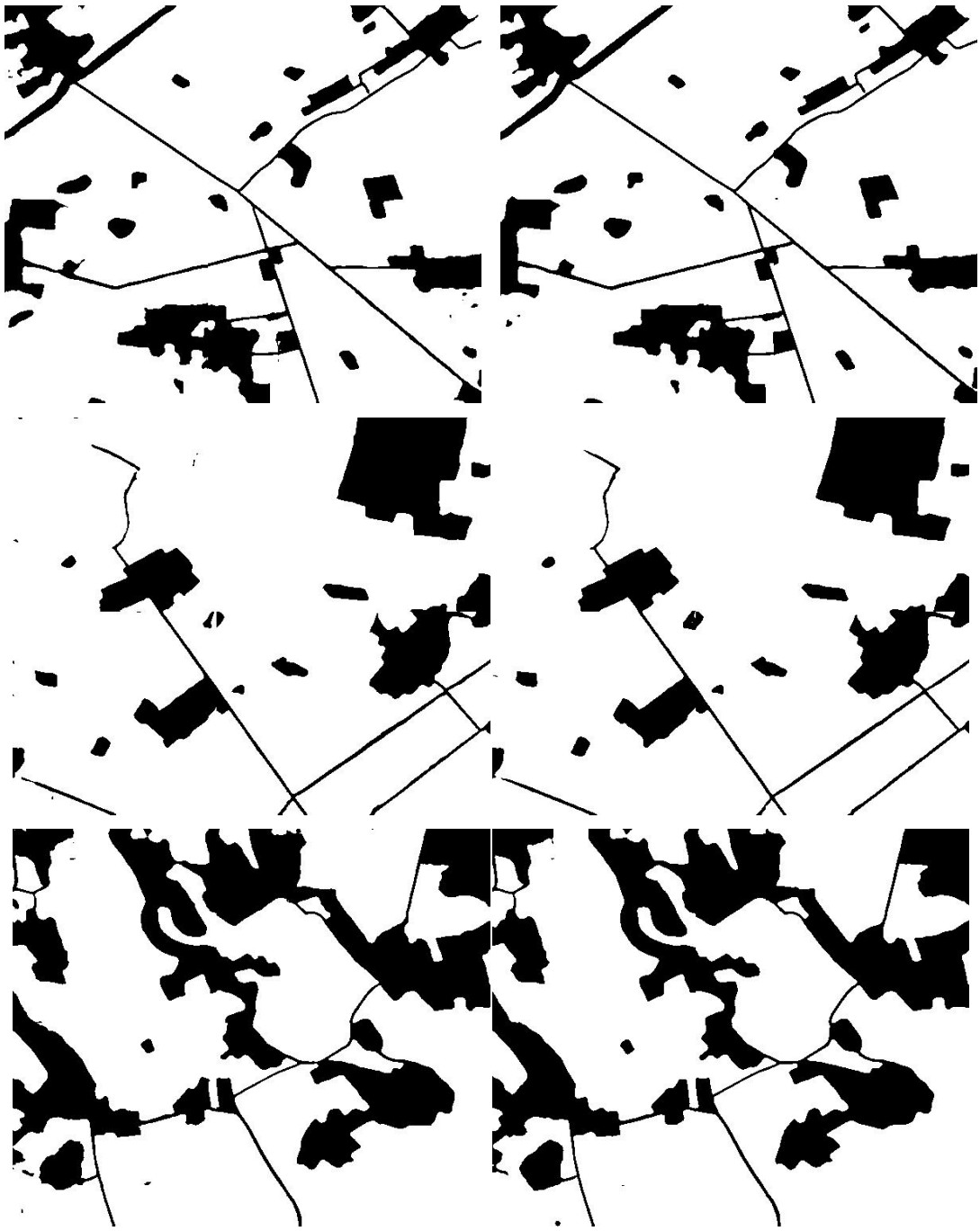
```
        i+=1
        filepath,filename = os.path.split(files)
        x = cv2.imread(files, -1)/255
        x[x>=0.5] = 1
        x[x<0.5] = 0
        print(x.shape)
        cv2.imwrite(filename+".tif", x) #转 png
    print('   new_tif_number = '+'%d'%i)

tif_generator()
```

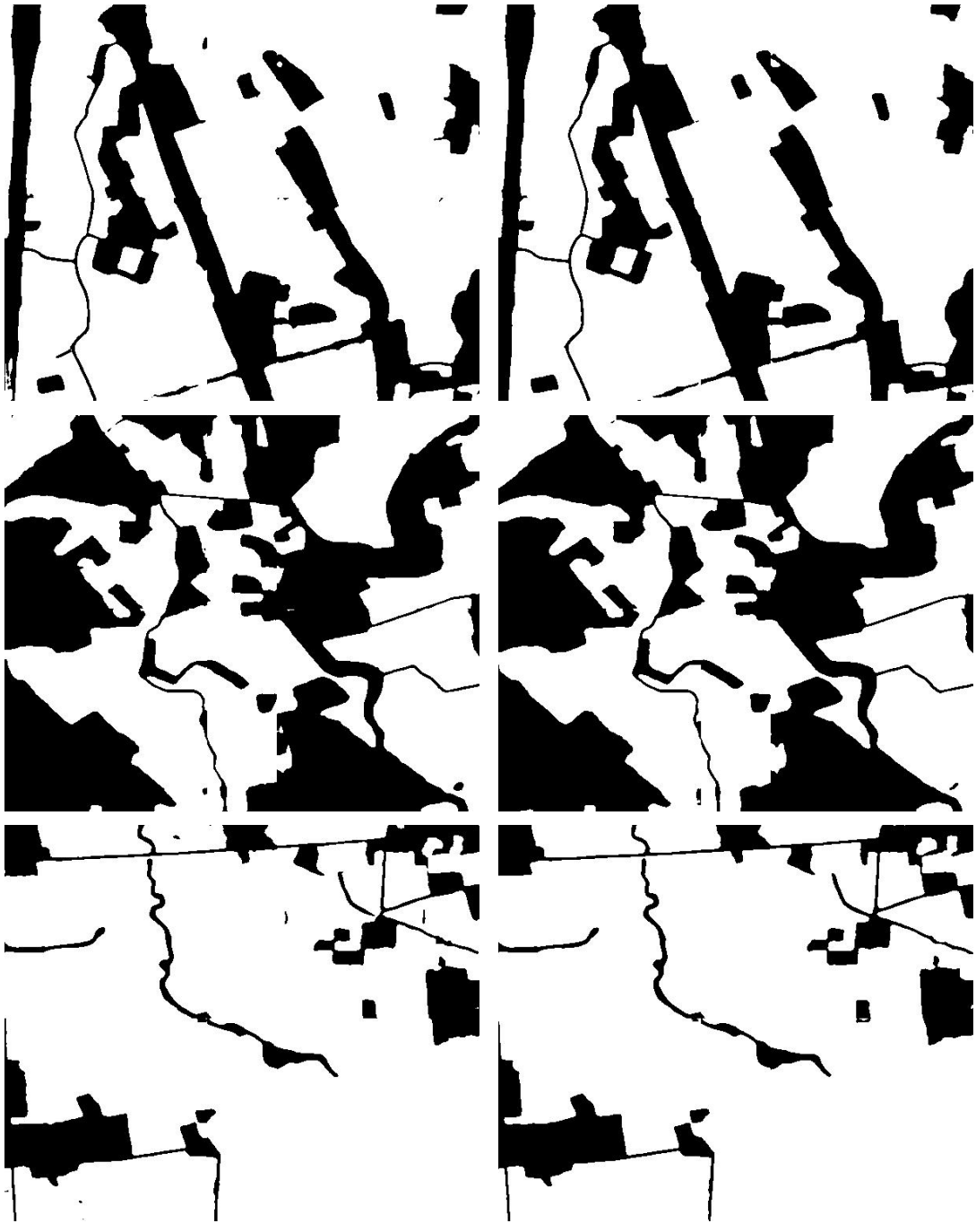
answer_S.py

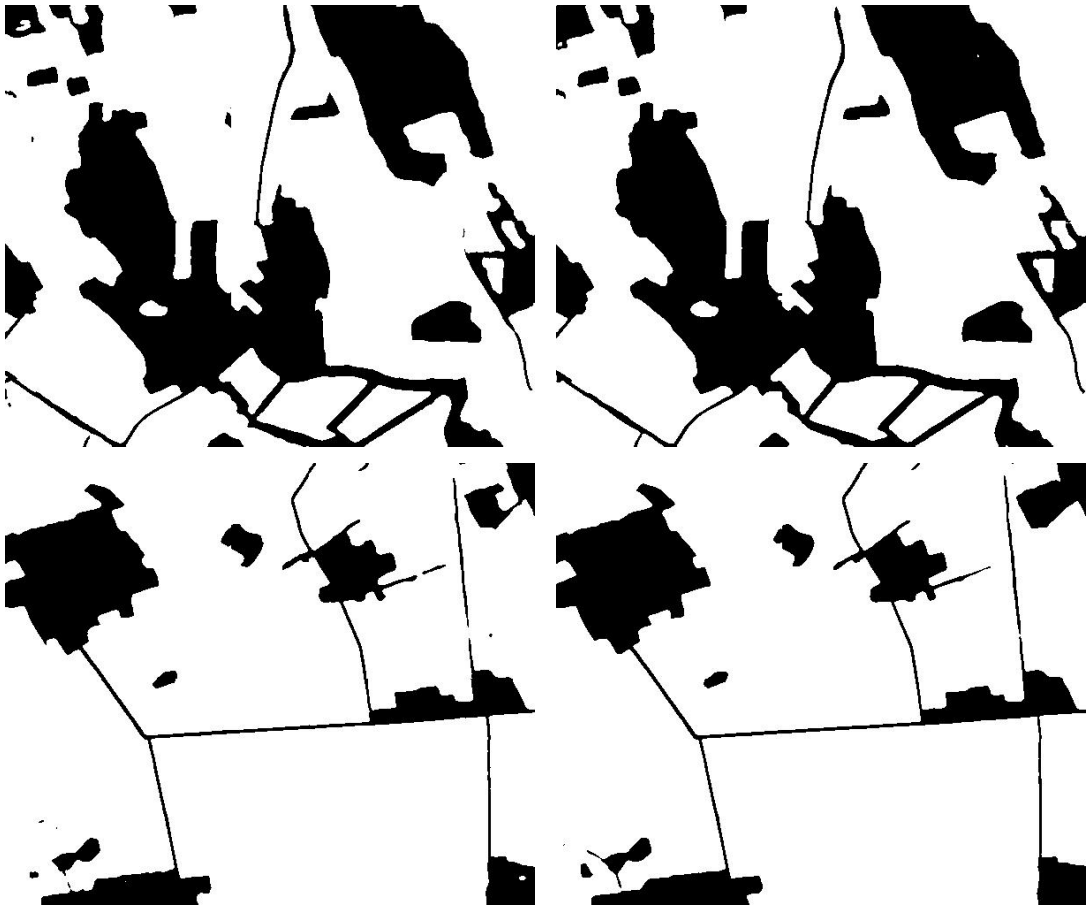
```
from PIL import Image
import numpy as np
#from train import mask
import glob
import os
import cv2

path1 = '*.tif'

path2 = '*.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):   #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(path2):  #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))
```

附录 2 多图平均二值法优化前后对比图[初赛模型]

（左前右后）

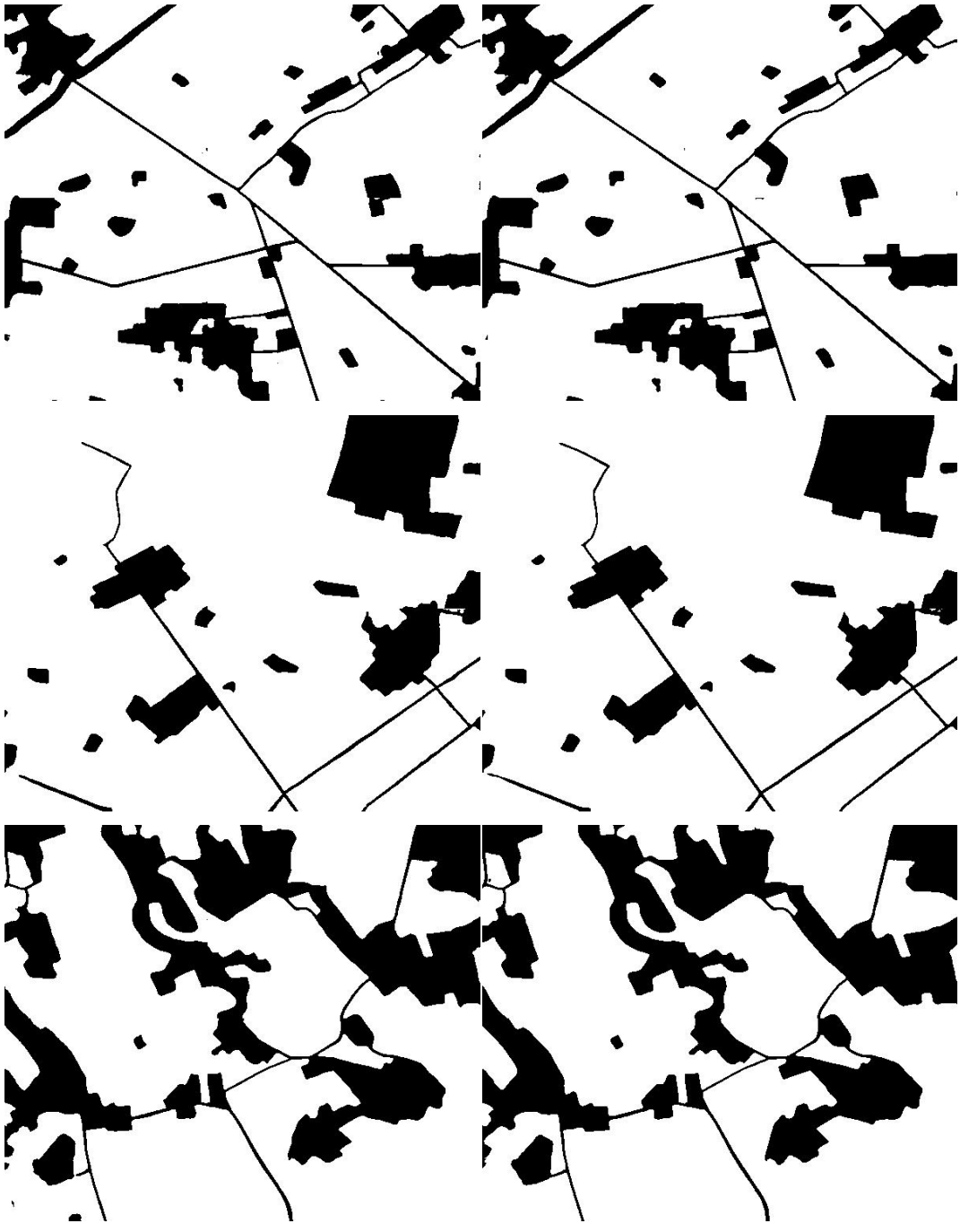


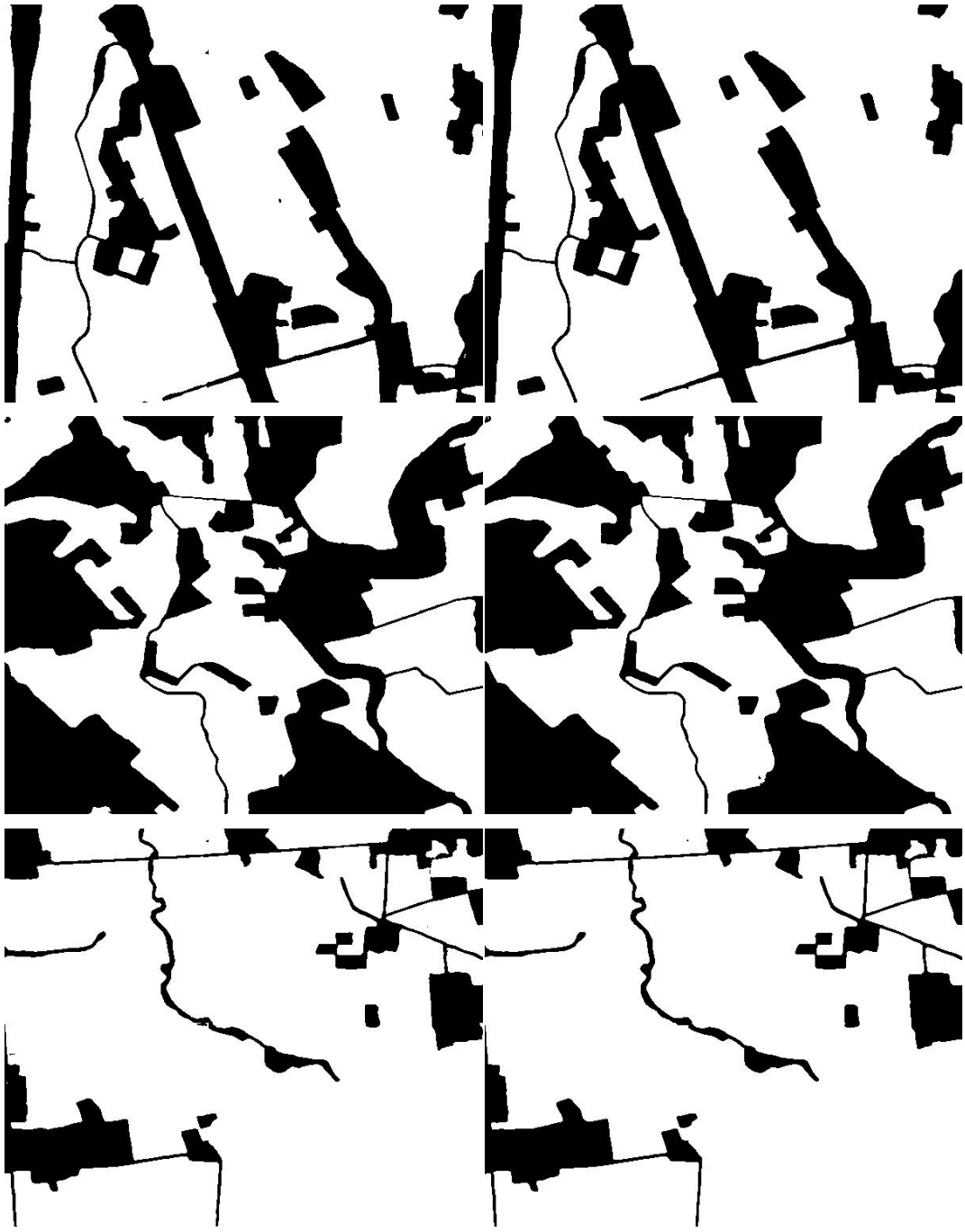


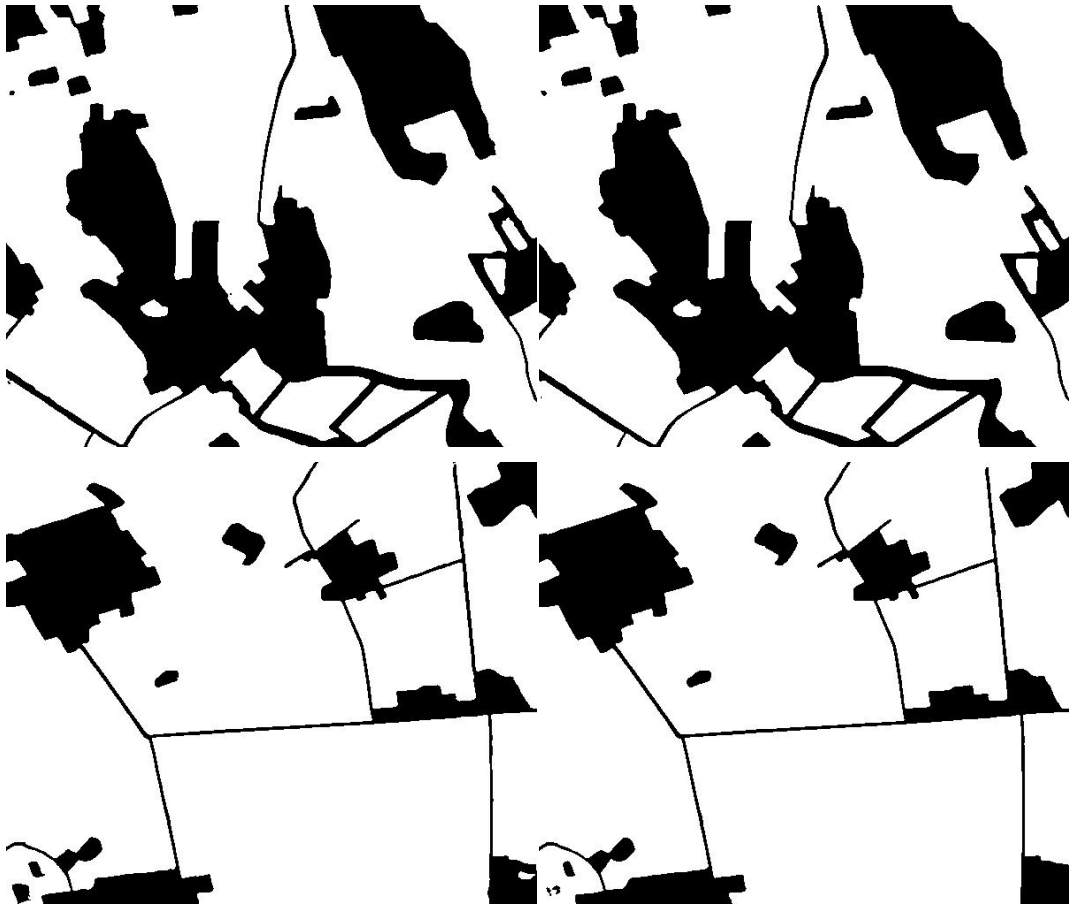


附录 3 多图平均二值法优化前后对比[复赛模型]

（左前右后）







附录 4 腐蚀与膨胀优化前后对比图

（左前右后）