队伍编号	604
赛道	B

遥感图像地块分割与提取

摘要

本文用改进后的遥感图像地块模型进行耕地分割、提取与面积计算；并介绍该识别系统中的重要参数，通过实验对相关参数进行定量评估.

针对问题一，需改进初赛的遥感图像耕地分割模型，并优化耕地边界的提取，从而更准确地计算 Test3、Test4 中耕地在图像中所占比例并制作耕地标签图. 在结构方面，该模型用九网络集成代替了三网络集成；在参数选取上，模型将 BCE、focal loss、Dice loss 等损失函数进行最优组合，并在训练中使用带重启的余弦退火学习率提升效率与准确率；对于耕地边界的细节处理处，模型改善了预测图边缘的毛躁，并将融合结果通过中值滤波去除噪声点. 改进后模型准确率从初赛的 90.68%升至 93.13%.

针对问题二，需描述该改进的遥感图像耕地分割模型的关键参数，并定量评估这个参数对识别精度的影响. 本部分包含了学习率、batch size、损失函数中的部分超参数、输入图像尺寸和网络个数等参数的描述与定量实验.

最后，在模型评价部分，分点概述了复赛模型相较初赛模型的改进，并对模型优点与其他有可能的改进进行简要评估.

关键词：遥感图像分割；数据增强；网络融合；中值滤波；余弦退火.

目录

一、问题重述

1.1 问题背景

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.

1.2 问题提出

赛题在初赛的基础上提出. 题目提供与初赛相同的 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. 原图像预览图如图 1 所示，标签图预览图如图 2 所示，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.

图 1.1 遥感数据预览图

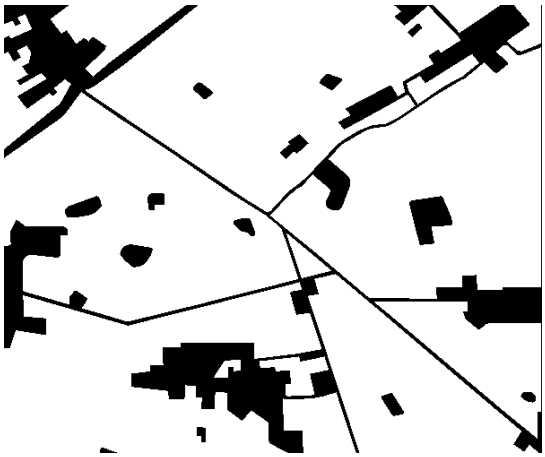


图 1.2 标注后的参考预览图（白色区域为耕地）

问题 1：利用改进后的模型，从给定的 2 幅测试图像(Test3.tif、Test4.tif)中提取出耕地，计算耕地在图像中所占比例，并制作标签图；说明相对于初赛模型，在优化提取耕地边界方面做出了哪些改进.

问题 2：描述改进的遥感图像耕地分割模型中的关键参数，并定量评估其对识别精度的影响.

二、模型假设

1.不考虑耕地的变化，标签结果仅由给出的图片所得.

2.不考虑对耕地不同种类的细分.

3.忽略照片本身的光照、大气等影响.

三、符号说明

符号	含义
α	不均衡样本权重
γ	难学习样本权重
y	真实值
$\hat{y}$	预测值
lr	学习率
BCE	交叉熵

四、问题一模型介绍及求解

4.1 问题一的分析

问题一的题目要求：a.计算给定的 2 幅测试图像在图像中所占比例，并制作耕地标签图; b.说明相对于初赛模型，在优化提取耕地边界方面做出了哪些改进.

针对问题一，首先需要改进初赛的遥感图像耕地分割模型，并优化耕地边界的提取. 因此，团队在初赛模型的基础上，分别对模型结构、损失函数、学习率等进行了新的尝试. 在结构上，用九网络集成代替了三网络集成; 在参数选取上，我们对 BCE、focal loss、Dice loss 等损失函数进行最优组合，并使用了带重启的余弦退火学习率; 对于耕地边界的细节处理，模型的融合结果利用中值滤波去除小的噪声点，提高验证集的准确率.

4.2 数据集的处理

复赛与初赛提供的数据集相同. 同样地, 鉴于图片尺寸太大, 训练集太小, 我们根据模型的输入大小, 首先对训练集进行 128*128, 224*224 的重叠分割. 再通过概率翻转、仿射变化、随机区域的像素丢失、部分图片弹性形变、添加高斯噪声、明亮度改变等方式对数据集进行数据增强[1], 将数据集扩展为原来的 3 倍.

相关数据增强实例（以 data1 为例）：

图 4.1 对 data1 进行 128*128 分割及数据增强操作的部分图片数据

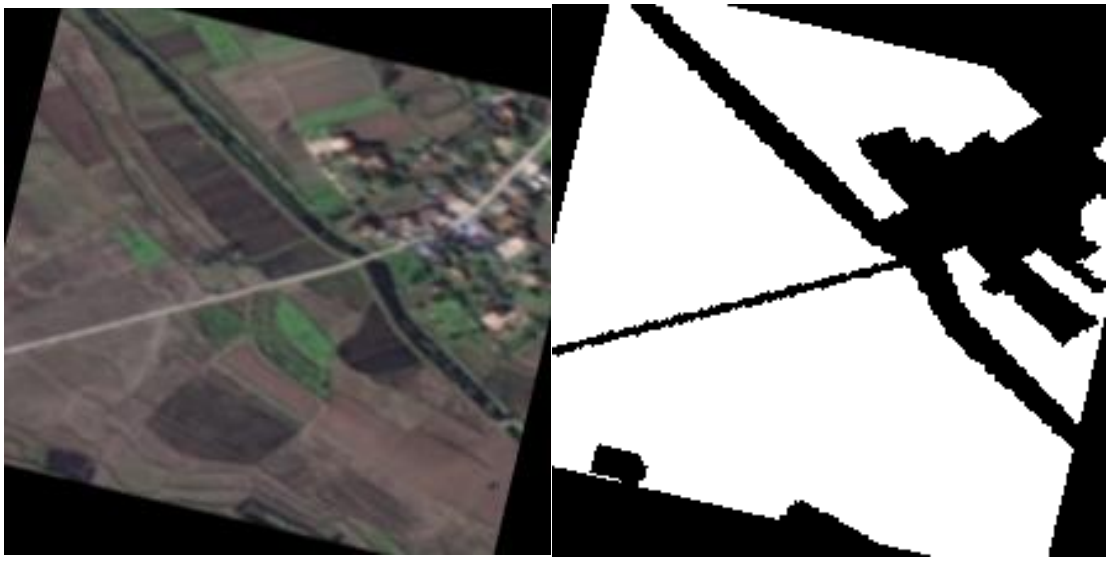


图 4.2 对 data1 进行 224*224 分割及数据增强操作的图片与其对应标签

对于模型不同的输入大小, 我们选取相应尺寸的第 4、7 幅图为测试图用于评估各模型准确率, 剩余六张图作为训练集. 获得最优模型后, 恢复 4、7 为训练图像, 用全部数据进行模型训练.

4.3 模型网络结构

4.3.1 网络选择

在初赛的模型基础上，我们尝试了新的模型：以 vgg16 为 backbone 的 Unet、以 Resnet50 为 backbone 的 Unet[2]以及 Unet++模型，但这些模型在验证集的准确率都不如以 Mobilenet[3]为 backbone 的 Unet（Mobilenet-Unet），所以我们最后选择了 Mobilenet-Unet.

4.3.2 Mobilenet-Unet

相比初赛，模型通过实验优化了损失函数，最终确定了九个损失函数与输入大小的组合. 本小节主要介绍 Mobilenet-Unet 的模块.

为了学习到鲁棒的特征，同时减少可训练参数的数量，我们对原来的 Unet 模型进行改进，使用预训练的 MobileNetV2 模型的中间输出层作为编码器，解码器仍采用原来的模块.

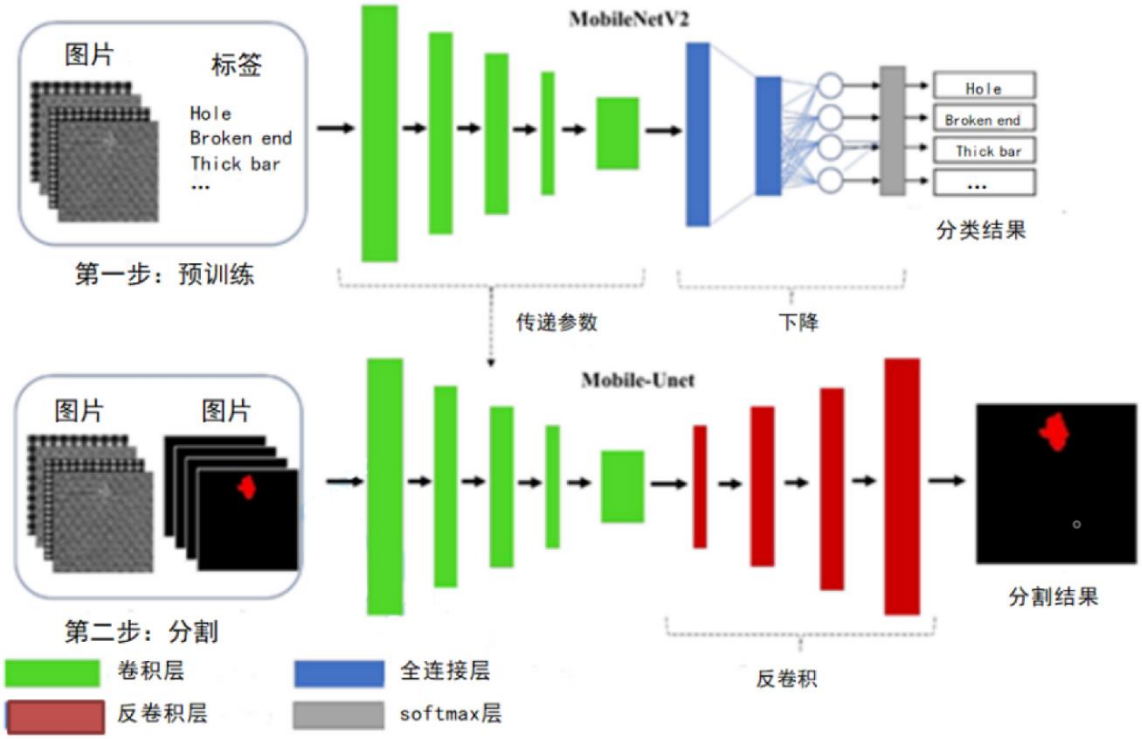


图 4.3 Mobilenet-Unet 的模型结构

网络指定编码器网络的输入维度为（128, 128, 3），为了保持编码器与解码器

的对称结构，只使用 MobileNetV2 模型中间层的一些特定输出来构成编码器.

如下所示，只使用模型中的这五层，他们是带有 relu 激活函数的卷积层，可以使特征维度不断减少，起到下采样的作用.

```
layer_names = [
'block_1_expand_relu',   # 64x64
'block_3_expand_relu',   # 32x32
'block_6_expand_relu',   # 16x16
'block_13_expand_relu',   # 8x8
'block_16_project',   # 4x4
]
```

解码器仍采用原来的结构，包含四组 3x3 卷积和通过反卷积实现的上采样层，最后再添加一个 sigmoid 层来输出像素预测概率.

4.3.3 损失函数

对于损失函数，我们实验了 Focal loss[5] , Dice loss[6]和带标签平滑的二分类交叉熵损失函数.

a. BinaryCrossentropy Loss

$$Loss = -\sum_{i=1}^{n} \hat{y}_i \log y_i + (1-\hat{y}_i) log(1-\hat{y}_i)$$

该函数时用于二分类的最经典损失函数，预测值与真实值相差越大，loss 越大；当且仅当 $\hat{y}$ 和 y 是相等时，loss 为 0，否则 loss 为正数. 交叉熵损失函数适用于大多数的语义分割场景中. 但其有一个明显缺点：针对只分割前景和背景的问题，当前景像素的数量远远小于背景像素的数量（即 $y=0$ 的数量远大于 $y=1$ 的数量），此时数据样本严重不均衡，损失函数中 $y=0$ 的背景占据主导作用，导致模型严重偏向于背景，分割效果将下降.

b. Focal Loss

$$Loss = -\alpha(1-\hat{y})^{\gamma} \times y \log(\hat{y}) - (1-\alpha)\hat{y}^{\gamma} \times (1-y) log(1-\hat{y})$$

此函数对不同类别的像素数量不均衡提出了改进方法，并将像素分为难学习和容易学习这两种样本，让模型更加关注难学习的样本.

对于公式中的参数 α，改变其大小可以调整不同类样本的权重，降低类不平衡的影响. 通过在二分类交叉熵损失函数前加入$(1-\hat{y})^{\gamma}$这一权重项，可使容易样本对损失函数的影响减小，困难样本对损失函数的影响增大，从而对难学习的像素赋予更高的权重.

c. dice soft loss

dice loss 针对前景比例太小的问题提出，适用于解决本次比赛数据集中非耕地区域比例较小的问题. dice 系数源于二分类，本质上通过衡量两个样本的重叠部分进行计算. 度量范围为 0 至 1，当且仅当完全重叠时 Dice 系数为 1.

$$Dice = \frac{2|A \cap B|}{|A| + |B|}$$

对于在预测的分割掩码上评估 Dice 系数，我们可以将$|A \cap B|$近似为预测掩码和标签掩码之间的逐元素乘法，再对结果矩阵求和. 1-Dice 作为可以最小化的损失函数，则最终的 Dice 损失为：

$$Loss = 1 - \frac{2\sum_{pixels} y * \hat{y}}{\sum_{pixels}(y^2 + \hat{y}^2)}$$

将每个类的 Dice 损失求和取平均，得到最后的 Dice soft loss.

d. 损失函数的选择

这三种损失函数针对不同情况，各有优点和缺点，因此在实际应用中，我们损失函数组合来进行互补. 如将 dice soft loss 和 Crossentropy Loss 结合，编写了 Dice_SoftCELoss，将三种损失函数结合在一起编写 Focal_Dice_Bce Loss.

通过对这些损失函数和输入大小进行不同的组合实验，准确率最优的组合如下：

	模型名称	损失函数	输入大小
Net1	Mobilenet-Unet	交叉熵(BCE)	128
Net2	Mobilenet-Unet	Focal loss(alpha=0.3)	128
Net3	Mobilenet-Unet	Focal loss(alpha=0.4)	128
Net4	Mobilenet-Unet	Dice loss+BCE	128
Net5	Mobilenet-Unet	Focal+dice+BCE	128
Net6	Mobilenet-Unet	交叉熵(BCE)	224
Net7	Mobilenet-Unet	Focal loss	224
Net8	Mobilenet-Unet	Dice loss+BCE	224
Net9	Mobilenet-Unet	Focal+dice+BCE	224

4.3.4 网络融合

如上表，我们选择了 9 个网络用于模型集成. 在每个像素点位置采用投票策略决定像素的最终预测值.

其中在对每个模型进行预测时，我们对模型的输出采用更严格的判定策略，对于每个像素点，当模型预测其属于耕地的概率大于某个阈值时我们判定其为耕地. 我们尝试了一系列阈值，通过观察其对验证集准确率的影响（如下图），可以看到当阈值在 0.75 到 0.8 之间比较合适，考虑到模型的泛化能力，保守起见我们将阈值定为 0.75.

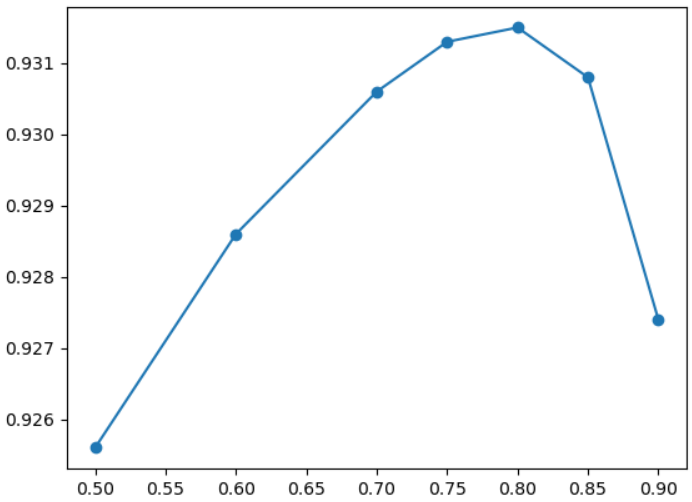


图 4. 4 不同阈值对验证集准确率的影响

对于投票法，前文的 9 个组合分别对应 9 个网络，运行后针对同一张遥感图

片会生成 9 张标签图，对这 9 张标签图逐像素进行投票来决定每一像素最终的标签，投票方法为少数服从多数，如果有 5 个或 5 个以上的模型判定当前像素的标签为土地（用像素值为 255 的像素点表示），则最终标签为土地，否则为非土地（用像素值为 0 的像素点表示）.

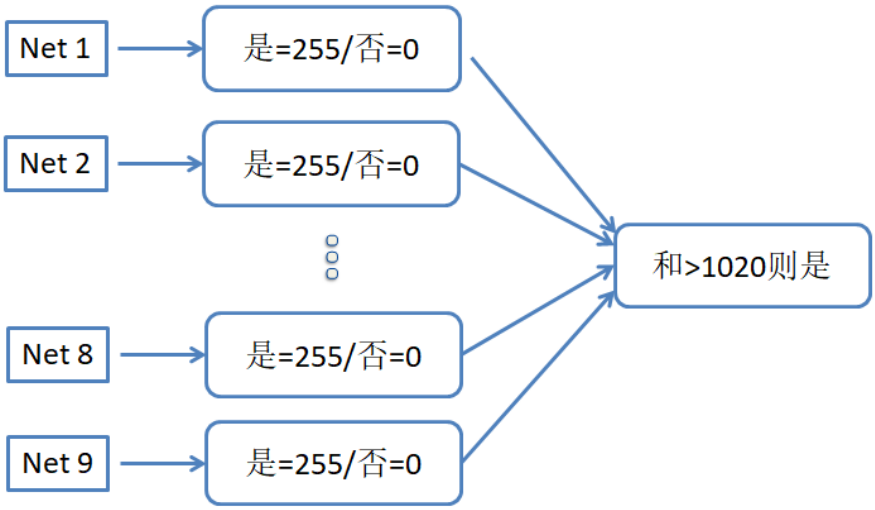


图 4. 5 投票法原理图

4.4 模型训练

相比初赛，本文在模型训练上做了改进. 在训练中，我们使用了带重启的余弦退火学习率[4]下降，间隔 2, 4, 8, 16 个 epoch 分别重启一次，最后一次 epoch 为 32. 优化器我们选择了 Adam 优化器. 对 128 和 224 两种输入尺寸，batchsize 均设为 16. 每个 epoch 评估一次准确率和损失值. 以 Net1 为例，下附训练过程中损失变化可视化图像示例：

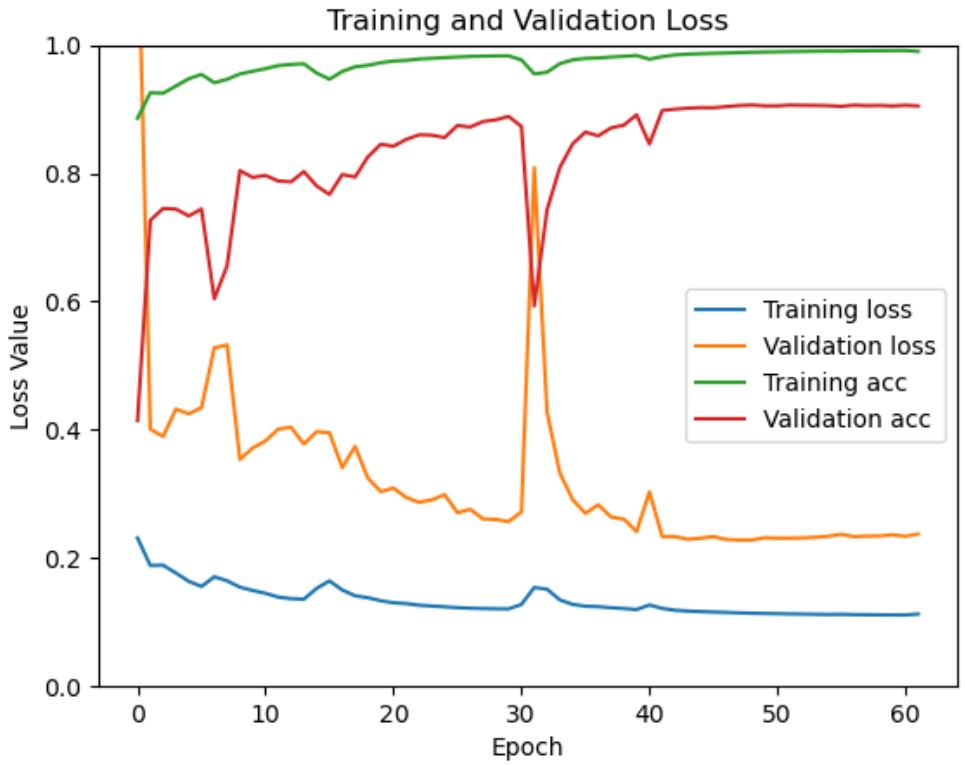


图 4.6 Net1 网络损失变化

其中蓝线代表训练集损失变化；黄线代表验证集损失变化；绿线代表训练集准确率变化；红线代表验证集准确率.

4.5 结果分析与精度检验

对于不同输入与损失函数的九网络，以及基于投票法的九网络融合模型的评估结果如下:

模型名称	损失函数	输入大小	Batch size	验证集准确率
Mobilenet-Unet	交叉熵(BCE)	128	16	91.77%
Mobilenet-Unet	Focal loss(alpha=0.3)	128	16	90.28%
Mobilenet-Unet	Focal loss(alpha=0.4)	128	16	90.35%
Mobilenet-Unet	Dice loss+BCE	128	16	91.94%
Mobilenet-Unet	Focal+dice+BCE	128	16	92.01%
Mobilenet-Unet	交叉熵(BCE)	224	16	91.67%
Mobilenet-Unet	Focal loss	224	16	91.03%
Mobilenet-Unet	Dice loss+BCE	224	16	91.51%
Mobilenet-Unet	Focal+dice+BCE	224	16	91.63%

其中，评估结果的计算方式为：准确率=预测正确的像素个数/总像素个数.

与初赛的比较：

	单网络最高	单网络最低	融合准确率
改进后	92.01%	90.28%	93.13%
改进前	89.83%	87.82%	90.68%

可见改进后的模型在准确率方面获得了大幅度的提升.

4.6 标签图的预测与面积计算

4.6.2 标签图的预测与优化

在初赛的模型中，通过观察预测结果和真实标签我们发现网络对耕地边界的学习效果较差，预测图的边缘毛刺比较多，这主要是因为耕地边缘的像素比较难学习. 相关可视化效果如下：

4.7 Data7 局部预测结果

4.8 Data7 局部真实标签

因此在初赛的基础上，我们尝试使用 Focal loss 作为损失函数. Focal loss 公式如下：

$$Focal\ loss = -\alpha(1-y_{pred})^{\gamma} \times y_{true} log(y_{pred}) - (1-\alpha) y_{pred}^{\gamma} \times (1-y_{true}) log(1-y_{pred})$$

对于预测结果概率高的像素点被视为容易样本，与之对立的是困难样本，模型可以轻易的预测正确容易样本，所以应该把更多的注意力放在对困难样本的学习上. 通过在二分类交叉熵损失函数前加入 $(1-y_{pred})^{\gamma}$ 这一权重项，可以使容易样本对损失函数的影响减小，困难样本对损失函数的影响增大，从而对难学习的像素赋予更高的权重.

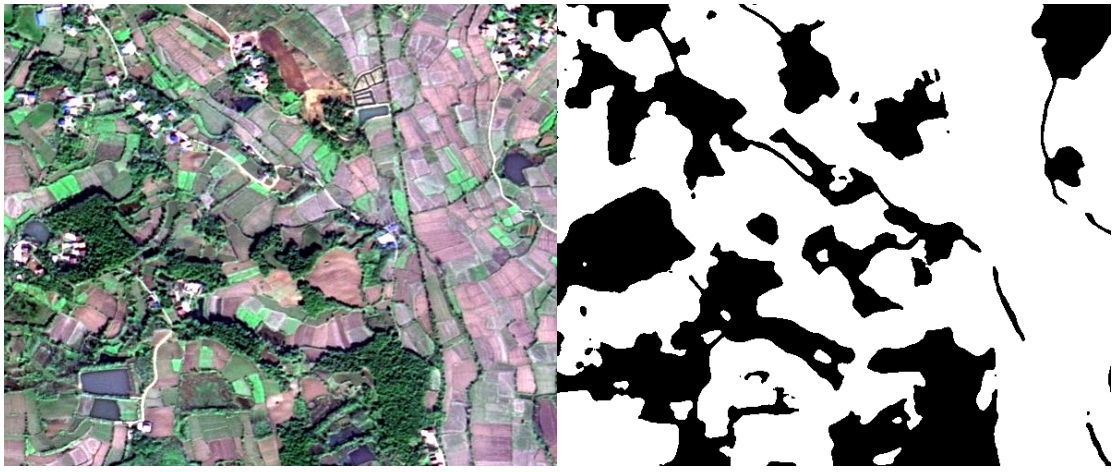
下面是使用二分类交叉熵损失函数的预测结果和使用 Focal loss 损失函数对 Data4.tif 进行预测的结果的对比，可以看到右图比左图预测结果更完整，图像左边的细长道路更连续，但是同时也带来等多的噪声，导致准确率下降. 因此我们用多模型融合方式降低其带来的负面影响，同时提升准确率和优化预测图.

4.9 使用交叉熵损失函数的预测结果

4.10 使用 Focal loss 的预测图

最终，相比初赛模型，本文选取了准确率和标签预测最优化的九网络融合模型，并带入所有数据重新训练了改进后的模型，然后进行模型集成对测试集进行预测.

其次，我们对融合的结果进行了中值滤波[7]，去除小的噪声点；对预测结果进行后处理，剔除小的像素块. 最终可视化结果如下：



4.11 test3 原图及其标签预测图

4.12 test4 原图及其标签预测图

4.6.1 面积计算

获取标签图后，再利用初赛中的 histogram 函数计算面积. histogram 函数模型主要是对 RGB 图像进行处理，它的原理为利用区间为[0, 255]的数组 hist 统计图中各像素值所拥有的像素个数. 因此，若要利用 histogram 函数求解耕地面积占比，步骤为：a.标签图的读取; b.函数模型计算.

a.读取标签图

因为生成的标签图是 tif 格式的，无法直接用 histogram 函数统计各像素值个数，所以需要先转换成 8 位 RGB 图. 16 位图的像素值在[0,65535]，将它映射到[0,255]区间，并尽可能的减少精度的损失，$image'_{[i][j]} = \frac{image_{[i][j]} * 255}{65535}$（$image_{[i][j]}$ 表示第 *i* 张图第 *j* 个像素点的像素值）

b.计算土地面积比例

将 a 得到的转换格式后的标签图输入面积计算模型得到土地面积比例. 该模型的具体计算步骤为：用 histogram 函数统计出该图片像素值在区间[0,255]范围内各像素值所拥有的像素个数; histogram 函数通过设置一个长度为 256 的数组 hist，数组区间为[0,255]，遍历图中所有像素点的像素值，设当前像素值为 x，则 hist[x]中的值加 1，相当于是一个计数器.

由于标签图只包含 255 和 0 这两种像素值，且标签图尺寸为 600*500，共 300000 个像素，只需将像素值 255 的像素个数除以 300000 即可得到耕地比例.

即 $percent = \frac{hist[255]}{300000}$.

最终面积计算结果为：

编号	耕地所占比例
Test3	68.62%
Test4	53.18%

4.7 模型优化总结

在初赛基础上的改进如下表：

变化/问题解决	初赛	复赛
融合个数	3	9
网络选取	Unet、Mobilenet-Unet	Mobilenet-Unet
损失函数	BCE	BCE、Focal loss、dice soft loss 组合
学习率	普通学习率下降	带重启的余弦退火学习率下降
阈值	0.5	0.75
去除边缘毛刺	CRF	BCE 与 Focal loss 的组合
标签图去噪	无	中值滤波

模型的融合准确率从初赛的 90.68%升至 93.13%.

五、问题二模型参数描述与评估

5.1 学习率

在训练过程中，为加快网络学习的速度，可随时间慢慢减少学习率，我们称此方法为学习率衰减. 为此，我们先后实验了两种学习率下降的方法.

第一种是普通的学习率下降：

$$\text{reduce\_lr} = \text{ReduceLROnPlateau}(\text{monitor} = \text{'val\_loss'}, \text{factor} = 0.1, \text{patience} = 6)$$

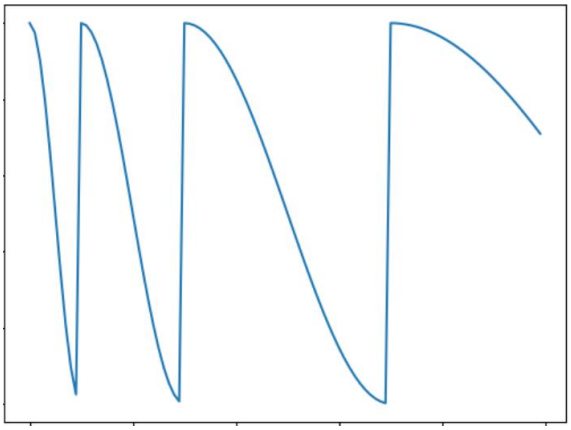
如代码所示，如果连续 6 个 epoch 的准确率没有提升，模型会将学习率降低为 $new\_lr = lr * factor$.

但这种下降方法容易使模型陷入局部最优解，搭配早停止方法后，在迭代了

40 余次后模型即会自动停止训练. 因此相比初赛，在复赛中我们又使用了第二种学习率下降方法，即带重启的余弦退火学习率下降. 其原理如下

$$\eta_t = \eta_{min}^i + \frac{1}{2}(\eta_{max}^i - \eta_{min}^i)(1 + cos(\frac{T_{cur}}{T_i}\pi))$$

使用梯度下降算法来优化目标函数，当越来越接近 Loss 的全局最小值点时，余弦退火通过余弦函数来降低学习率，从而让模型尽可能接近这一点. 余弦函数中随着 x 的增加余弦值首先缓慢下降，后加速下降，再缓慢下降. 这种下降模式和学习率配合，产生更好的效果. 不止如此，在训练时梯度下降算法可能陷入局部最小值，此时可以通过突然提高学习率，来“跳出”局部最小值并找到通向全局最小值的路径.



5.1 带重启的余弦退火学习率变化图

在定量评估过程中，我们在 Net 1 上进行了初始学习率的实验，实验结果如下表：

不同学习率下降方法和数值	验证集准确率
普通学习率下降(initial_lr = 0.01)	89.66%
普通学习率下降(initial_lr = 0.001)	89.96%
带重启的余弦退火学习率下降(initial_lr = 0.001)	90.28%

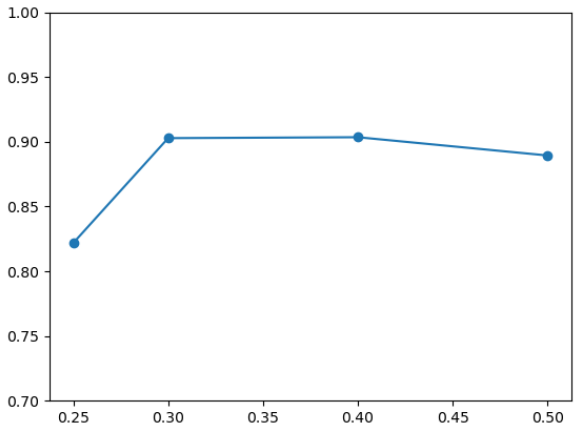
5.2 Loss 函数选择

如问题一所述, BinaryCrossentropy Loss、Focal Loss、dice soft loss 三种损失函数针对不同情况，各有优点和缺点. 在定量评估实验中，对 BinaryCrossentropy Loss、Dice_SoftCELoss、Focal Loss、Focal_Dice_Bce Loss 这四种损失函数进行了实验，结果如下表：

损失函数	验证集准确率
BinaryCrossentropy Loss	91.77%
Dice_SoftCELoss	91.94%
Focal Loss（α=0.3）	90.28%
Focal_Dice_Bce Loss	92.01%

5.3Loss 函数中部分参数

Focal Loss 中调节 α 参数可调整不同类样本的权重，缓解类别不平衡的影响. α 参数大小对模型在验证集上的准确率影响如下图，当 α 在 0.3 到 0.4 区间效果最好.



5. 2 参数大小验证集准确率的影响

如图，当 α 在 0.3 到 0.4 区间效果最好. 因此，实验针对 α 进行定量评估，在验证集上对应准确率如下表：

α 值	准确率
α=0.3	90.28%
α=0.4	90.35%

5.4 Batch size

Batch Size 为一次训练所选取的样本数，其大小影响模型的优化程度和速度. Batch Size 通过并行化提高内存的利用率可提高训练速度；但同时，适当减小 Batch Size 使得梯度下降方向更加准确.

对此，实验对 Net1 网络进行不同的 Batch size 进行评估，结果如下表所示：

Batch size	准确率
32	90.50%
16	91.61%
8	91.66%

可见，适当的减小 Batch Size 的值可提高模型准确率. 但由于 Batch size 为 16 和 8 时模型准确率相差不大，故综合时间因素，最终选择 Batch Size 为 16.

5.5 网络个数

为了实验网络个数对分割效果的影响，我们设置了三组实验，固定模型输入大小为 128*128，选取 Net1 为第一组，模型融合 Net3、Net4、Net5 作为第二组，模型融合 Net1-Net5 作为第三组，实验结果如下表：

网络选择	准确率
第一组（Net1）	91.77%
第二组（Net3、Net4、Net5 融合）	92.93%
第三组（Net1、Net2、Net3、Net4、Net5 融合）	92.13%

由表，适当增加网络个数可以增加准确率，继续增加可能会导致过拟合的问题，如果结合不同的输入尺寸，让模型学习到更多的特征，则可以进一步提升准确率.

5.6 输入图像尺寸

从直观来看，输入图片越大，其纹理和上下文就越多，越能捕捉到更好的特征. 但是，当尺寸变大到一定程度，分类性能可能反而会不变，甚至变坏，计算的开销也会相应变大. 对此，我们选取了除输入图像尺寸不同外，其余设置均相同的两组网络进行了实验，结果如下：

模型名称	输入图像尺寸	准确率
Mobilenet-Unet（交叉熵）	128*128	91.77%
	224*224	91.67%
Mobilenet-Unet（Dice loss+BCE）	128*128	91.94%
	224*224	91.51%
Mobilenet-Unet（Focal loss）	128*128	90.35%
	224*224	91.03%

从表中可以看出，对于不同的损失函数，两种输入尺寸得到的结果均不相同，前两组实验中，均是输入尺寸为 128*128 的模型效果较好，第三组则是 224*224 的模型效果更好. 为了使模型学习到更好的图片特征，同时考虑到运行效率的因素，在最终的模型里同时结合了这两种输入尺寸.

六、模型总结与评价

6.1 改进总结

相比初赛，复赛在模型结构、损失函数、预测标签图细节处理等方面都进行了改进. 最终融合模型的准确率从初赛的 90.68%升至 93.13%. 改进主要如下:

6.1.1 融合网络个数

如初赛文中提到的投票法不局限与三种网络，故复赛中，我们进行了一个、三个、五个、九个网络的实验，实验得九网络在准确率上获得显著提高.

6.1.2 网络选取

初赛的融合了两个 Unet 和一个 Mobilenet-Unet. 在复赛中，通过不断的调整参数，得到不同损失及输入下的 Mobilenet-Unet 性能超过 Unet，故网络选取修改为不同类型的 Mobilenet-Unet.

6.1.3 损失函数组合

初赛仅仅使用了 BCE，复赛中将 BCE、Focal loss、dice soft loss 组合，有效结合不同损失函数的优势.

6.1.4 改进学习率

初赛仅使用普通的学习率下降法，复赛改进为带重启的余弦退火学习率下降，有效加速了训练效率，且训练中避免了可能陷入局部最小值的问题.

6.1.5 阈值调整

初赛中阈值为 0.5，复赛中在对每个模型进行预测时，我们对模型的输出采用更严格的判定策略，实验后将阈值定为 0.75，显著提升了准确率.

6.1.6 预测标签图细节处理

初赛中的预测标签图未能有效改善边缘毛刺和噪声等问题. 复赛中，我们在训练时利用 BCE 与 Focal loss 的组合，有效改善预测标签图的边缘毛刺问题. 在网络融合后，利用中值滤波去除预测图噪声.

6.2 模型评价

6.1.1.优点

a.模型的准确率显著提升;

b.预测标签图边缘处更加平滑;

c.预测标签图噪声点减少.

6.2.2 可改进之处

a. 保证准确率的同时尝试轻量化网络;

b. 数据集处理部分添加其他数据增强方法;

c. 尝试改进优化器.

七、参考文献

[1] Shorten, Connor, and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. Journal of Big Data 6.1 (2019): 60.

[2] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

[3] Jing, Junfeng, et al. Mobile-Unet: An Efficient Convolutional Neural Network for Fabric Defect Detection. Textile Research Journal, May 2020,doi:10.1177/0040517520928604.

[4] Loshchilov I , Hutter F . SGDR: Stochastic Gradient Descent with Warm Restarts[J]. 2016.

[5] Lin T Y , Goyal P , Girshick R , et al. Focal Loss for Dense Object Detection[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2017, PP(99):2999-3007.

[6] Fausto Milletari, Nassir Navab, et al.V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation[J].2016.

[7] S. Perreault and P. Hebert, "Median Filtering in Constant Time",IEEE Transactions on Image Processing, September 2007.

附录：

程序（Python）：

训练：train.py ; 测试：test.py ; 计算面积：calculate.py

train.py：

```
# -- coding: utf-8 --
import cv2 as cv
import numpy as np
import os
from model.Unet import unet,unet2
from model.Segnet import SegNet,segnet
from model.unetplusplus import Xnet
import matplotlib.pyplot as plt
import tensorflow as tf
from prepocessing import load_dataset
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from sklearn.model_selection import train_test_split
from Config import config_

from losses import Dice_SoftCELoss,Focal_Loss,Focal_Dice_Bce

config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)


def train_model(use_all=False):
    #load dataset
    config_ob=config_()
    if use_all:
        train_x,train_y,val_x,val_y=load_dataset()
        train_x.extend(val_x)
        train_y.extend(val_y)
        x=train_x
        y=train_y
        sample_num=len(y)
        # index=list(range(len(x)))
        # np.random.shuffle(index)
        # x=x[index]
        # y=y[index]
        print(np.asarray(x).shape)
        # sample_num=int(np.asarray(x).shape[0]*(1.-0.15))
```

```
        # #split dataset into train_dataset and test_dataset
        # x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15)
        traindataset=tf.data.Dataset.from_tensor_slices((x,y))
        # valdataset=tf.data.Dataset.from_tensor_slices((x_test,y_test))
    else:
        train_x,train_y,val_x,val_y=load_dataset()
        sample_num=np.asarray(train_x).shape[0]
        traindataset = tf.data.Dataset.from_tensor_slices((train_x, train_y))
        valdataset = tf.data.Dataset.from_tensor_slices((val_x, val_y))


    traindataset=traindataset.shuffle(buffer_size=1024).batch(config_ob.batch_size)
    valdataset=valdataset.shuffle(buffer_size=1024).batch(config_ob.batch_size)

    #train

    if config_ob.model=='unet':
        model=unet()
    elif config_ob.model=='unet2':
        model=unet2()
    elif config_ob.model=='segnet':
        model=SegNet()
    elif config_ob.model=='xnet':

model=Xnet(input_shape=(config_ob.width,config_ob.width,config_ob.channels))
    else:
        model=None

    lr_schedule = tf.keras.experimental.CosineDecayRestarts(1e-3, 2 *
int(sample_num/config_ob.batch_size), alpha=0.1, t_mul=2.0)
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
                  loss=tf.keras.losses.BinaryCrossentropy(),
                  #loss=Dice_SoftCELoss(loss_weights=(0.5,0.5)),
                  #loss=Focal_Loss(),
                  #loss=Focal_Dice_Bce(),
                  metrics=['accuracy'])

    reduce_lr = tf.keras.callbacks.ReduceLROnPlateau('val_loss', factor=0.1,
                                          patience=6,
                                          verbose=1)
    csvlogger =
tf.keras.callbacks.CSVLogger('output/{}_{}.log'.format(config_ob.model,config_ob.ind
ex))
    # lrscheduler=WarmUpCosineDecayScheduler(learning_rate_base=1e-4,
```

```
    #
total_steps=config.epochs*config.steps_each_epoch,
    #                                                   warmup_learning_rate=1e-5,
    #
warmup_steps=int(0.33*config.epochs)*config.steps_each_epoch,
    #                                                   hold_base_rate_steps=100,
    #                                                   verbose=0)
    earlystopping = tf.keras.callbacks.EarlyStopping('val_loss', patience=10,
verbose=1)
    modelcheckpoint =
tf.keras.callbacks.ModelCheckpoint('output/{}_{}_best'.format(config_ob.model,confi
g_ob.index),

monitor='val_loss',

save_best_only=True,

save_weights_only=True,
                                                                    )

    History=model.fit(traindataset,
                      epochs=config_ob.epochs,
                      validation_data=valdataset,
                      callbacks=[csvlogger]
                     )

    model.save('output/{}_{}.h5'.format(config_ob.model,config_ob.index))
    #model.save('output/{}_{}'.format(config_ob.model,config_ob.index))

    #plot the training loss and accuracy
    # print(History.history.keys())
    if not use_all:
        loss = History.history['loss']
        val_loss = History.history['val_loss']
        acc=History.history['accuracy']
        val_acc=History.history['val_accuracy']
        epochs = range(len(loss))
        plt.figure()
        plt.plot(epochs, loss,    label='Training loss')
        plt.plot(epochs, val_loss, label='Validation loss')
        plt.plot(epochs,acc,label='Training acc')
        plt.plot(epochs,val_acc,label='Validation acc')
        plt.title('Training and Validation Loss')
        plt.xlabel('Epoch')
```

```
        plt.ylabel('Loss Value')
        plt.ylim([0, 1])
        plt.legend()
        plt.savefig('output/{}_{}.png'.format(config_ob.model,config_ob.index))
        plt.show()

import time
start=time.time()
train_model(use_all=False)
print('训练时间：',time.time()-start)
#unet2 1995s 0.9599
#583    0.9477
#746s 0.9222
```

test.py：

```
# -- coding: utf-8 --
import cv2 as cv
import numpy as np
from tensorflow.keras.models import load_model
from prepocessing import read_image
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from losses import Dice_SoftCELoss,Focal_Loss
#from CRF import crf
from Config import config_
from postprocessing import mor,transform

config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)

config_ob=config_()

#用来预测两张测试图片，将结果保存
def
predict_and_save(model_index_list,test_images=None,test_labels=None,model_inp
ut_width=None,flag=None):

    model_num=len(model_index_list)

    if test_images is None:
        test1_path='data/data/Test3.tif'
```

```
        test2_path='data/data/Test4.tif'

        test1=read_image(test1_path)/255.0
        test2=read_image(test2_path)/255.0
        test_images=[test1,test2]

    res_4=[]
    res_7=[]
    model_id=0
    for model_name,index in model_index_list:
        model = load_model('output/{}_{}.h5'.format(model_name, index),
compile=False)
        for id,image in enumerate(test_images):
            mask=predict(model,image,model_input_width[model_id])
            if id==0:
                res_4.append(mask)
            else:
                res_7.append(mask)
        model_id+=1

    mask4=np.sum(res_4,axis=0)
    print(np.array(mask4).shape)
    h, w = np.array(mask4).shape[:2]
    for i in range(h):
        for j in range(w):
            if mask4[i][j] >= 255*(model_num//2):
                mask4[i][j] = 1
            else:
                mask4[i][j] = 0
    result = mask4.astype(np.uint16)
    cv.imwrite('data/results/4_16bit_{}.tif'.format(flag), result)
    result = mask4.astype(np.uint8)
    cv.imwrite('data/results/4_8bit_{}.tif'.format(flag), result)
    result *= 255
    result = result.astype(np.uint8)
    result=mor(result)
    cv.imwrite('data/results/4_{}.png'.format(flag), result)

    if test_labels is not None:
        compare = cv.bitwise_xor(result, test_labels[0]) / 255
        print(1 - np.sum(compare) / (500 * 600))


    mask7 = np.sum(res_7, axis=0)
```

```
        print(np.array(mask7).shape)
        h, w = np.array(mask7).shape[:2]
        for i in range(h):
            for j in range(w):
                if mask7[i][j] >= 255 * (model_num // 2):
                    mask7[i][j] = 1
                else:
                    mask7[i][j] = 0
        result = mask7.astype(np.uint16)
        cv.imwrite('data/results/7_16bit_{}.tif'.format(flag), result)
        result = mask7.astype(np.uint8)
        cv.imwrite('data/results/7_8bit_{}.tif'.format(flag), result)
        result *= 255
        result = result.astype(np.uint8)
        result=mor(result)
        cv.imwrite('data/results/7_{}.png'.format(flag), result)

        if test_labels is not None:
            compare = cv.bitwise_xor(result, test_labels[1]) / 255
            print(1 - np.sum(compare) / (500 * 600))


#采用覆盖预测
def evaluate_cover_128():
        test1_path = 'data/data/Data4.tif'
        test2_path = 'data/data/Data7.tif'
        mask1_path= 'data/data/Data4_reference.tif'
        mask2_path= 'data/data/Data7_reference.tif'

        test1 = read_image(test1_path) / 255.0
        test2 = read_image(test2_path) / 255.0
        mask1=cv.imread(mask1_path,-1).astype(np.uint8)*255
        mask2=cv.imread(mask2_path,-1).astype(np.uint8)*255
        test_images = [test1, test2]
        test_masks=[mask1,mask2]
        print(test1.shape,mask1.shape)
        # stride_w = 118
        # stride_h = 93
        stride=64
        flag = True
        model = load_model('model/segnet_17.h5')
        acc=0
        for image, mask_true in zip(test_images, test_masks):
            h, w = image.shape[:2]
```

```
            padding_h = (h // stride + 2) * stride
            padding_w = (w // stride + 1) * stride
            padding_img = np.zeros((padding_h, padding_w, 3), dtype=np.float32)
            padding_img[0:h, 0:w, :] = image[:, :, :]
            mask = np.zeros((padding_h, padding_w), dtype=np.float32)
            for i in range(padding_h//stride-1):
                for j in range(padding_w//stride-1):
                    crop = padding_img[i * stride:i * stride + 128, j * stride:j * stride +
128, :]
                    crop = np.expand_dims(crop, axis=0)
                    pred = model.predict(crop)
                    pred = pred.reshape((128, 128))

                    if j==0 and i==0:
                        mask[i * stride:i * stride + 128-32, j * stride:j * stride +
128-32] = pred[:(128-32),:(128-32)]
                    elif j==0 and i!=0:
                        mask[i * stride+32:i * stride + 128 - 32, j * stride:j * stride +
128 - 32] = pred[32:(128 - 32), :(128 - 32)]
                    elif j!=0 and i==0:
                        mask[i * stride:i * stride + 128 - 32, j * stride+32:j * stride +
128 - 32] = pred[:(128 - 32), 32:(128 - 32)]
                    else:
                        mask[i * stride+32:i * stride + 128 - 32, j * stride+32:j *
stride + 128 - 32] = pred[32:(128 - 32), 32:(128 - 32)]
            mask=mask[:h,:w]

            ret, mask = cv.threshold(mask, 0.5, 255, cv.THRESH_BINARY)
            mask=mask.astype(np.uint8)
            # cv.imshow('mask',mask)
            # cv.imshow('ori',mask_true)
            # cv.waitKey(0)
            compare=cv.bitwise_xor(mask,mask_true)/255
            print(1-np.sum(compare)/(500*600))
            acc+=np.sum(compare)

    acc=1-acc/(500*600*2)
    print(acc)

    #使用学习率下降+数据增强  0.8760  unet_13.h5                0.8756(使用重
叠测试)
    #使用学习率下降+数据增强(plus)  0.8647  unet_14.h5          0.8693(使用重
叠测试)
    #使用学习率下降+数据增强(plus)  0.8564  segnet_14.h5     0.8611(使用重叠
```

```
测试)
    #0.9061

def evaluate_128():
    test1_path = 'data/data/Data4.tif'
    test2_path = 'data/data/Data7.tif'
    mask1_path = 'data/data/Data4_reference.tif'
    mask2_path = 'data/data/Data7_reference.tif'

    test1 = read_image(test1_path) / 255.0
    test2 = read_image(test2_path) / 255.0
    mask1 = cv.imread(mask1_path, -1).astype(np.uint8) * 255
    mask2 = cv.imread(mask2_path, -1).astype(np.uint8) * 255
    test_images = [test1, test2]
    test_masks=[mask1,mask2]

    stride_w = 118
    stride_h = 93


#model=load_model('output/{}_{}.h5'.format(config_ob.model,config_ob.index))
    #model =
load_model('output/{}_1.h5'.format(config_ob.model),custom_objects={'Dice_SoftC
ELoss':Dice_SoftCELoss(smooth=0.1,loss_weights=(0.5,0.5))},compile=False)
    model =
load_model('output/{}_{}.h5'.format(config_ob.model,config_ob.index),custom_obje
cts={'Dice_SoftCELoss':Focal_Loss()},compile=False)
    #model = load_model('output/vgg_37.h5',compile=False)

#model=load_model('output/{}'.format(config_ob.model),custom_objects={'Dice_So
ftCELoss':Dice_SoftCELoss(smooth=0.1,loss_weights=(0.5,0.5))},compile=False)

    acc=0
    acc_=0
    ff = 0
    index=[4,7]
    for image,mask_true in zip(test_images,test_masks):
        h, w = image.shape[:2]
        mask = np.zeros((h, w), dtype=np.float32)
        for i in range((h - 128) // stride_h + 1):
            for j in range((w - 128) // stride_w + 1):
                crop = image[i * stride_h:i * stride_h + 128, j * stride_w:j *
stride_w + 128, :]
                crop = np.expand_dims(crop, axis=0)
```

```
                    pred = model.predict(crop)
                    pred = pred.reshape((128, 128))
                    mask[i * stride_h:i * stride_h + 128, j * stride_w:j * stride_w +
128] = pred[:, :]
            ret, mask = cv.threshold(mask, 0.5, 255, cv.THRESH_BINARY)
            mask=mask.astype(np.uint8)

cv.imwrite('data/label_compare/{}_{}.png'.format(index[ff],config_ob.index),mask)

            compare = cv.bitwise_xor(mask, mask_true) / 255
            print(1 - np.sum(compare) / (500 * 600))
            acc_ += np.sum(compare)



            mask=mor(mask)
            cv.imwrite('data/label_compare/{}_{}_post.png'.format(index[ff],
config_ob.index), mask)




            compare=cv.bitwise_xor(mask,mask_true)/255
            print(1 - np.sum(compare) / (500 * 600))
            acc+=np.sum(compare)
            ff+=1

        acc=1-acc/(500*600*2)
        acc_ = 1 - acc_ / (500 * 600 * 2)
        print(acc,acc_)

#评估输入为 224*224 的模型
def evaluate_224():
        test1_path = 'data/data/Data4.tif'
        test2_path = 'data/data/Data7.tif'
        mask1_path = 'data/data/Data4_reference.tif'
        mask2_path = 'data/data/Data7_reference.tif'

        test1 = read_image(test1_path) / 255.0
        test2 = read_image(test2_path) / 255.0
        mask1 = cv.imread(mask1_path, -1).astype(np.uint8) * 255
        mask2 = cv.imread(mask2_path, -1).astype(np.uint8) * 255
        test_images = [test1, test2]
        test_masks = [mask1, mask2]
```

```
    stride_w = 188
    stride_h = 138

    #
model=load_model('output/{}_{}.h5'.format(config_ob.model,config_ob.index))
    #                                                  model                                                  =
load_model('output/{}_1.h5'.format(config_ob.model),compile=False)
    model            =            load_model('output/{}_{}.h5'.format(config_ob.model,
config_ob.index),compile=False)
    # model=load_model('output/{}'.format(config_ob.model),compile=False)
    acc = 0
    acc_ = 0
    ff = 0
    index = [4, 7]
    for image, mask_true in zip(test_images, test_masks):
        h, w = image.shape[:2]
        mask = np.zeros((h, w), dtype=np.float32)
        for i in range((h - 224) // stride_h + 1):
            for j in range((w - 224) // stride_w + 1):
                crop = image[i * stride_h:i * stride_h + 224, j * stride_w:j *
stride_w + 224, :]
                crop = np.expand_dims(crop, axis=0)
                pred = model.predict(crop)
                pred = pred.reshape((224, 224))
                mask[i * stride_h:i * stride_h + 224, j * stride_w:j * stride_w +
224] = pred[:, :]
        ret, mask = cv.threshold(mask, 0.7, 255, cv.THRESH_BINARY)
        mask = mask.astype(np.uint8)
        cv.imwrite('data/label_compare/{}_{}_4.png'.format(index[ff],
config_ob.index), mask)

        compare = cv.bitwise_xor(mask, mask_true) / 255
        print(1 - np.sum(compare) / (500 * 600))
        acc_ += np.sum(compare)

        mask = mor(mask)
        cv.imwrite('data/label_compare/{}_{}_post4.png'.format(index[ff],
config_ob.index), mask)

        compare = cv.bitwise_xor(mask, mask_true) / 255
        print(1 - np.sum(compare) / (500 * 600))
        acc += np.sum(compare)
```

```
            ff += 1

        acc = 1 - acc / (500 * 600 * 2)
        acc_ = 1 - acc_ / (500 * 600 * 2)
        print(acc, acc_)

#投票
# def vote_per_image2():
#         test1_path = 'data/data/Data4.tif'
#         test2_path = 'data/data/Data7.tif'
#         mask1_path = 'data/data/Data4_reference.tif'
#         mask2_path = 'data/data/Data7_reference.tif'
#
#         test1 = read_image(test1_path) / 255.0
#         test2 = read_image(test2_path) / 255.0
#         mask1 = cv.imread(mask1_path, -1).astype(np.uint8) * 255
#         mask2 = cv.imread(mask2_path, -1).astype(np.uint8) * 255
#         test_images = [test1, test2]
#         test_masks = [mask1, mask2]
#
#         model1=load_model('model/unet_16.h5')
#         model2=load_model('model/unet_17.h5')
#         model3=load_model('model/unet_18.h5')
#
#
#         for image,mask_true in zip(test_images,test_masks):
#             h, w = image.shape[:2]
#             mask1=predict(model1,image,128)
#             mask2=predict(model2,image,128)
#             mask3=predict(model3,image,224)
#
#             for i in range(h):
#                 for j in range(w):
#                     temp=mask1[i][j]+mask2[i][j]+mask3[i][j]
#                     if temp>=500:
#                         mask1[i][j]=255
#                     else:
#                         mask1[i][j] = 0
#
#             result=mask1.astype(np.uint8)
#             compare = cv.bitwise_xor(result, mask_true) / 255
#             acc = np.sum(compare)
#             print(1-acc/(500*600))
```

```
#返回预测图像
def predict(model,image,block_size=224):

    #def predict_one(model,image,block_size=224):

    if block_size==128:
        stride_w=118
        stride_h=93
    else:
        stride_w = 188
        stride_h = 138

    h,w=image.shape[:2]
    mask=np.zeros((h,w),dtype=np.float32)
    for i in range((h-block_size)//stride_h+1):
        for j in range((w-block_size)//stride_w+1):

crop=image[i*stride_h:i*stride_h+block_size,j*stride_w:j*stride_w+block_size,:]
                crop=np.expand_dims(crop,axis=0)
                pred=model.predict(crop)
                pred=pred.reshape((block_size,block_size))

mask[i*stride_h:i*stride_h+block_size,j*stride_w:j*stride_w+block_size]=pred[:,:]
    ret,mask=cv.threshold(mask,0.75,255,cv.THRESH_BINARY)

    return mask

    # image_transformed=[image]+transform(image)
    # labels=[]
    # for img in image_transformed:
    #         labels.append(predict_one(model,img,block_size=block_size))
    # labels[1:]=transform(labels[1:])
    # label_num=len(labels)
    # label=np.sum(labels,axis=0)
    # ret, mask = cv.threshold(label, 255*(label_num//2), 255, cv.THRESH_BINARY)
    #
    # return mask

def vote_pred():
    test1_path = 'data/data/Data4.tif'
    test2_path = 'data/data/Data7.tif'
    mask1_path = 'data/data/Data4_reference.tif'
    mask2_path = 'data/data/Data7_reference.tif'
```

```
        test1 = read_image(test1_path) / 255.0
        test2 = read_image(test2_path) / 255.0
        mask1 = cv.imread(mask1_path, -1).astype(np.uint8) * 255
        mask2 = cv.imread(mask2_path, -1).astype(np.uint8) * 255

#model_index_list=[('unet',2000),('unet',1),('unet',3),('unet',5),('unet',8),('unet',11),('
unet',12),('unet',13),('unet',14)]

model_index_list=[('unet',21),('unet',22),('unet',26),('unet',23),('unet',24),('unet',35),
('unet',33),('unet',34),('unet',32)]
        test_images = [test1,test2]
        test_labels = [mask1,mask2]
        model_input_width =[128,128,128,128,128,224,224,224,224]

        predict_and_save(model_index_list,None,None,model_input_width,flag=(1))
        #9 post(中值滤波) 10（无后处理）  11（open）

#predict_and_save(model_index_list,test_images,test_labels,model_input_width,fla
g=str(5))


if __name__ =='__main__':
        evaluate_128()
        #vote_pred()

        # a=[0.5,0.6,0.7,0.75,0.8,0.85,0.9]
        # b=[0.9256,0.9286,0.9306,0.9313,0.9315,0.9308,0.9274]
        # plt.plot(a,b,'o-')
        #
        # plt.show()
```

calculate.py：

```
import cv2 as cv
import numpy as np
import os
def read_image(path):

        image=cv.imread(path,-1)
        image = image.astype(np.uint8)
        image *= 255
        image=np.expand_dims(image,-1)
        return image
```

```
label_path =    ['data/results/4']
for file in os.listdir(label_path):
        image=read_image(os.path.join(label_path, file))/255.0
        hist, _ = np.histogram(image, bins=256, range=None, weights=None)
        percent = hist[255]/300000.0
        print(file+":", percent)
```