

队伍编号	224
赛道	B

## 基于图像语义分割的遥感地块分割与提取

### 摘 要

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获取耕地区域，准确的耕地分布能够为国家决策部门提供重要支撑。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。本文通过建立深度学习语义分割网络模型对遥感地块分割和提取展开了相关研究。

**针对问题 1：**本文首先利用 **OpenCV** 对附件所给的图像数据进行读取并显示，观察所给图像和标签大小一致，然后针对 8 幅训练图像，由于其给出了相对应的标签且是**灰度图像**，直接计算数值 1 所占的比例即可；针对 2 幅测试图像，利用问题 2 建立的图像语义分割模型对测试图像进行预测，对于预测出的标签，进一步计算数值 1 的比例。

**针对问题 2：**为了好的解决问题，首先查阅相关了文献了解常用的遥感地块分割方法，如基于边缘、形态学、区域的分割方法以及传统的 **CNN** 方法。由于传统的图像算法主要依靠设计者的先验知识，常需要针对具体的问题人为地设计特征，且模型的鲁棒性不好，而深度学习可以从大数据中自动学习好的特征的表示，**图像语义分割**可以用完成像素级别的分类，是一项更高层次的图像解析任务。因此，本文建立深度学习的图像语义分割模型完成遥感地块的分割和提取任务。

首先，考虑到给定的数据集样本较少，会造成模型训练困难，因此采用不同的**数据增强**方式，通过旋转 90°、旋转 180°、旋转 270°、翻转、光照调整、模糊操作、增加噪声等方式，将训练数据集扩充到 **3000 张**。为了直观地看到所给图片在数据增强前后的对比效果，并对生成的图像进行了可视化对比。进一步，分别建立了**基于注意力机制的 ResNet 的 FCN1s 网络、Attention U-Net 网络**对问题进行了求解，在具体训练中，为了使训练的过程与我们的目标一致，将损失函数设计为 **CELoss** 和 **DiceLoss** 二者的加权和，并以 **Dice 系数**为评价指标引导模型进行更好的训练，并将其训练过程进行了可视化，最后损失均收敛到了 **0.50** 附近，然后使用验证集上评价分数最高的模型对测试图片进行预测。

为了改善模型的预测能力，提高模型的鲁棒性，本文采用**集成学习**方法对进行三个不同网络的**模型融合**，对每张结果图的每个像素点，采取少数服从多数的投票表决策策略，从而去掉一些明显分类错误的像素点，并得到最后的测试集图片预测结果。

**针对问题 3：**本文通过对于前面问题的进一步延伸，给出了几点可以能快速、精准的识别出田块新的研究：1) 采用面向对象的多尺度分割技术；2) 综合地利用田块以及周边道路和水渠等特殊地物的几何特征、辐射特征、拓扑特征及上下文特征；3) 选择高质量的数据源，所选择的数据要具有较高的空间分辨率，需要有较高光谱分辨率的光谱信息。

**关键词：**遥感地块分割 FCN ResNet101 数据增强 注意力机制 Attention U-Net

目录

一、问题重述..... 1

    1.1 问题背景 ..... 1

    1.2 需要解决的问题 ..... 1

二、符号说明..... 2

三、问题 1..... 3

    3.1 问题分析 ..... 3

    3.2 模型建立与问题求解 ..... 3

        3.2.1 工具与模块介绍..... 3

        3.2.2 图片读取并计算比例..... 4

四、问题 2..... 5

    4.1 问题分析 ..... 5

    4.2 模型建立与问题求解 ..... 5

        4.2.1 遥感图像分割方法概述..... 5

        4.2.2 数据增强介绍..... 7

        4.2.3 模型一：基于注意力机制的 ResNet 的 FCN1s 网络..... 10

        4.2.4 模型二：Attention U-Net 网络..... 18

        4.2.5 模型融合..... 22

五、问题 3..... 23

六、模型总结和改进..... 24

    6.1 模型的优点 ..... 24

    6.2 模型的缺点 ..... 24

    6.3 未来的工作 ..... 24

七、参考文献..... 25

附录..... 26

# 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获取耕地区域，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。

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。原图像预览图如图 1-1 所示，标签图预览图如图 1-2 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

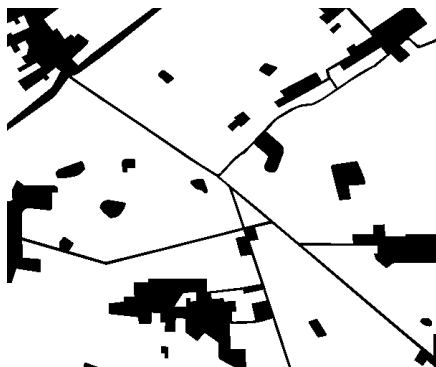


图 1-2 标注后的参考预览

## 1.2 需要解决的问题

问题 1：计算 10 幅图中耕地在各图像中所占比例，并填写表 1-1，数据请呈现在论文中；

表 1-1 耕地在各图像中所占比例

数据编号	耕地所占比例
Data 1	
Data 2	
Data 3	
Data 4	
Data 5	
Data 6	
Data 7	
Data 8	
Test 1	
Test 2	

问题 2：从给定的 2 幅测试图像(Test1.tif、Test2.tif)中提取出耕地，制作耕地标签图，并将标签图分别上传到竞赛平台中（注意田块间的边界是否清晰）；

问题 3：我国土地辽阔，地貌复杂，你有什么创新的思路能够快速、精准的识别出田块。

## 二、符号说明

符号	含义
$X$	图片真实标注的标签
$Y$	网络预测的结果标签
$X_i$	样本 $i$
$x_i$	样本 $i$ 的预测概率
$y_i$	样本 $i$ 的真实标签，取值 0 或 1
$w_i$	样本 $i$ 的权重

## 三、问题 1

### 3.1 问题分析

题目附件提供了由专业遥感解译人员解译的 8 幅图像和相应耕地标签，用于模型的训练。其中，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。题目要求我们计算出所给 10 幅图中耕地在所占的比例并填写相应表格。

针对附件所给的图像数据，首先利用 OpenCV 进行读取并显示，观察所给图像和标签大小是否一致。然后针对 8 幅训练图像，由于其给出了相对应的标签且是灰度图像，直接计算数值 1 所占的比例即可；针对 2 幅测试图像，利用深度学习的图像语义分割模型进行求解（问题二中给出），对于预测出的标签，进一步计算数值 1 的比例。

### 3.2 模型建立与问题求解

#### 3.2.1 工具与模块介绍

##### 3.2.1.1 Python 语言

Python 是一种代表简单思想的语言，语法简单，而且免费、开源、可移植性强，可以直接用源代码运行。在计算机内部，Python 解释器把源代码转换为字节码的中间形式，然后再把它翻译成计算机使用的机器语言。在国内外数据科学领域的研究中，Python 的使用率呈线性增长。不仅如此，它还集成了与数据分析相关的库，使算法实现变得简单。

##### 3.2.1.2 NumPy 模块

Python 实现的开源科学计算包。它可以定义高维数组对象、矩阵计算和随机数生成等函数。

##### 3.2.1.3 pandas 模块

Pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 提供了大量快速便捷地处理数据的函数和方法，它是使 Python 成为强大而高效的数据分析环境的重要因素之一。Pandas 的数据结构主要有 Series(一维数组)和 Data Frame(二维数组)。

##### 3.2.1.4 matplotlib 模块

Python 实现的作图包。使用 matplotlib 能够非常简单的可视化数据，仅需要几行代码，便可以生成直方图、功率谱图、条形图、散点图等。

##### 3.2.1.5 OpenCV 模块

OpenCV (Open Source Computer Vision Library) 是一个跨平台的开源的计算机视觉库，

它提供了很多高效地实现了计算机视觉算法的函数，可用于开发实时的图像处理、计算机视觉以及模式识别程序。

3.2.1.6 PyTorch 框架

PyTorch 是一个开源的 Python 机器学习库，基于 Torch 框架，底层由 C++实现，应用于人工智能领域。主要有两大特征：1) 类似于 NumPy 的张量计算，可使用 GPU 加速；2) 基于带自动微分系统的深度神经网络。主要包括 torch.nn、torch.optim 等子模块。

3.2.2 图片读取并计算比例

首先利用 cv2.imread()函数对图片进行读取，由于读取的标注文件是三通道的灰度图，因此取一个通道进行计算耕地所占的比例。以读取 Data1.png 为例，如图 3-1 所示。

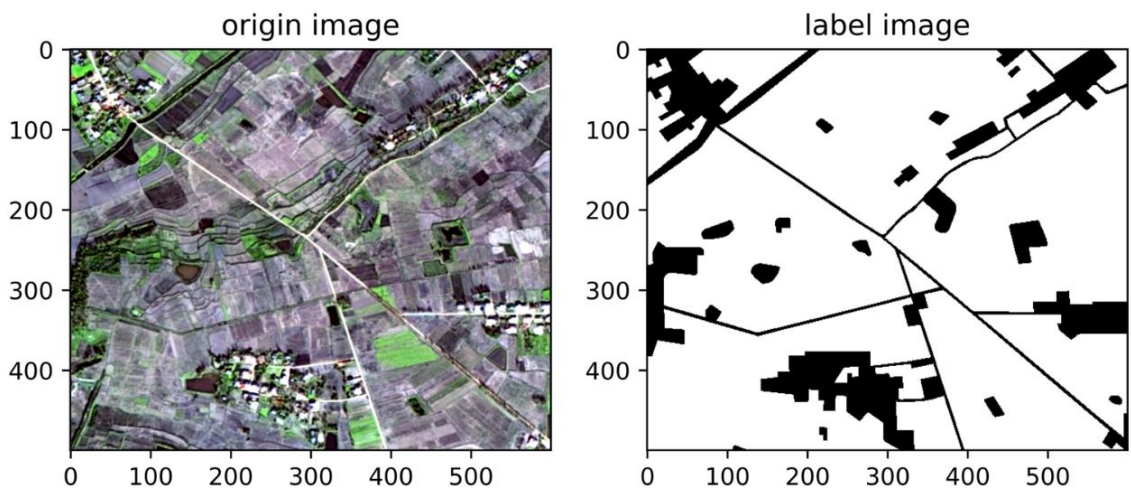


图 3-1 遥感图中的图片 Data1 和其对应的标注标签图

然后针对 8 幅训练图像，由于其给出了相对应的标签且是灰度图像，直接计算数值 1 所占的比例即可；对于 2 张测试图像，由问题 2 模型完成预测后，再进一步计算数值 1 所占的比例。将计算结果填入下表 3-1。

表 3-1 问题 1 中所占比例计算结果表

数据编号	耕地所占比例
Data 1	0.8329
Data 2	0.8423
Data 3	0.7004
Data 4	0.7468
Data 5	0.5534
Data 6	0.8556
Data 7	0.6802
Data 8	0.8389
Test 1	0.8505
Test 2	0.8097

## 四、问题 2

### 4.1 问题分析

题目要求从给定的 2 幅测试图像(Test1.tif、Test2.tif)中提取出耕地,制作耕地标签图,并且要注意田块间的边界是否清晰;

我们的目的是从给定的测试数据图中准确的提取出耕地区域,即对图像进行特定分割并获取感兴趣的区域。为了好的解决问题,通过查阅相关文献了解常用的遥感地块分割方法。由于传统的图像算法主要依靠设计者的先验知识,常需要针对具体的问题人为地设计特征,且模型的鲁棒性不好。而深度学习可以从大数据中自动学习好的特征表示,图像语义分割从图像级别的分类进一步扩展到像素级别,是一项更高层次的任务,为全面理解场景铺平了道路。因此,本文建立深度学习的图像语义分割模型完成遥感地块的分割和提取任务。首先,考虑到给定的数据集样本较少,会造成模型训练困难,因此采用数据增强的方法对数据集作进行扩充,并且考虑不同的数据增强方式对于遥感分割任务的影响。进一步,利用不同的图像语义分割模型 FCN、U-Net 对问题进行求解,由于要考虑到田块间的边界是否清晰,将注意力机制嵌入到网络模型中验证方法的有效性。在下面的描述中,我们将详细介绍实验过程和实验结果。

### 4.2 模型建立与问题求解

#### 4.2.1 遥感图像分割方法概述

图像分割是把一幅图像分割为一定数量的子区域,使得这些区域内部的属性或者特征相同,而相邻区域之间这些特性或属性是不同的。

遥感图像分割是一种面向对象的图像特征识别与信息提取技术,同时也包含对某些特征信息的检测等内容,它与遥感图像分类技术的最大不同是:分类技术得到的是输入图像的分类结果,而分割技术得到的结果则是输入图像的描述与解释。常用的遥感图像分割方法可以分为传统图像分割算法和基于深度学习的图像语义分割算法两大类。

传统的图像分割算法:

##### 1. 基于边缘的分割方法

边缘是图像中灰度值存在突变的地方,是两个不同区域的交界处,主要表现为图像中灰度值局部的不连续性。边缘检测算法的优点是边缘位置定位准确,但也有缺点,对噪声敏感。如果不能保证边界是封闭连续和单像素宽的,则处理所得图像还需要边缘链接和跟踪等方法才能得到闭合的边界。

##### 2. 基于形态学的分割方法

近年来,大量的图像分割算法采用形态学算子对图像进行预处理或后处理,基于数学形态学的分割算法飞速发展,已经成为一种流行的图像处理方法和理论,在这些基于形态学理论的图像分割算法中,比较有代表性的是分水岭算法( Watered )。因为图像中存在噪声和纹理,尤其是高分辨率遥感图像,地物表面细节更加清晰,纹理性更强,用分水岭算法处理时,必然会产生大量“伪盆地”,导致过分割问题,也就是一些同区域被错误分割成一定数量的小区域。

##### 3. 基于区域的分割方法

同一区域中的像素必然具有某些相似的特性,比如光谱,不同区域的像素必然具有某些不同的特性,因此可以利用这种性质将目标分离出来,本质就是将具有相似特

质的像素连接成区域,这种分割方法主要有两大类:区域生长法和分裂合并法。这种方法能够较为充分地组合利用全局与局部信息,具有层次性的合理结构,但是也有自身的不足。一方面,分裂如果不能达到最大限度,算法的分割精度就不会达到最高,而另一方面,如果分裂程度过高,可能会破坏一些同质区域。

#### 4. 基于马尔科夫随机场的分割方法

基于随机场的方法以统计学为理论依据,将图像中的像元灰度值看做是某种随机变量,然后选择合适数学模型对图像进行建模,通过这个模型分析图像特性,依靠分析图像中相邻像素点之间的关系来确定像素与其相邻像素是否属于同一区域,如果是,则将其合并到同一区域,最后得到分割结果。在这些方法中,应用最广泛的是基于马尔科夫随机场的图像分割方法。所以,在具体实践中常常需要结合实际问题,使用马尔科夫的基本理论设计更为合理的图像分割算法。

深度学习图像分割算法:

##### 1. 传统 CNN

传统的基于 CNN 的分割方法:为了对一个像素分类,使用该像素周围的一个图像块作为 CNN 的输入用于训练和预测。这种方法有几个缺点:

一是存储开销很大。例如对每个像素使用的图像块的大小为  $15 \times 15$ ,然后不断滑动窗口,每次滑动的窗口给 CNN 进行判别分类,因此则所需的存储空间根据滑动窗口的次数和大小急剧上升。

二是计算效率低下。相邻的像素块基本上是重复的,针对每个像素块逐个计算卷积,这种计算也有很大程度上的重复。

三是像素块大小的限制了感知区域的大小。通常像素块的大小比整幅图像的大小小很多,只能提取一些局部的特征,从而导致分类的性能受到限制。

通常 CNN 网络在卷积层之后会接上若干个全连接层,将卷积层产生的特征图(feature map)映射成一个固定长度的特征向量,最后都期望得到整个输入图像的一个数值描述(概率)。这个概率信息是 1 维的,即只能标识整个图片的类别,不能标识每个像素点的类别,所以这种全连接方法不适用于图像分割。

##### 2. 图像语义分割

图像语义分割(Semantic Segmentation)是图像处理和是机器视觉技术中关于图像理解的重要一环。语义分割即是对图像中每一个像素点进行分类,确定每个点的类别(如属于背景、人或车等),从而进行区域划分。目前,语义分割已经被广泛应用于自动驾驶、无人机落点判定等场景中。

图像语义分割网络采取全卷积神经网络,从抽象的特征中恢复出每个像素所属的类别,即相对于传统 CNN 算法来说,从图像级别的分类进一步延伸到像素级别的分类。遵循编码器/解码器结构,在这个结构中,编码器阶段对输入的空间分辨率下采样,产生分辨率更低的特征图,通过学习这些特征图可以更高效地分辨类别,解码器阶段为了解决卷积和池化对图像尺寸的影响,则是将这些特征表征上采样至完整分辨率的分割图。深度学习语义分割方向的代表性论文有 FCN、SegNet、U-Net、PSPNet、Deeplab 系列、OCRNet 等。

传统的图像算法主要依靠设计者的先验知识,常需要针对具体的问题人为地设计特征,很难利用大数据的优势,而且依赖手工调参数,特征的设计中只允许出现少量的参数,就会造成模型的鲁棒性不好。而深度学习可以从大数据中自动学习好的特征表示,从而极大提高模型的性能。图像语义分割从图像级别的分类进一步扩展到像素级别,是一项更高层次的任务,为全面理解场景铺平了道路。因此,本文建立深度学习的图像语义分割模型完成遥感地块的分割和提取任务。



4.2.2 数据增强介绍


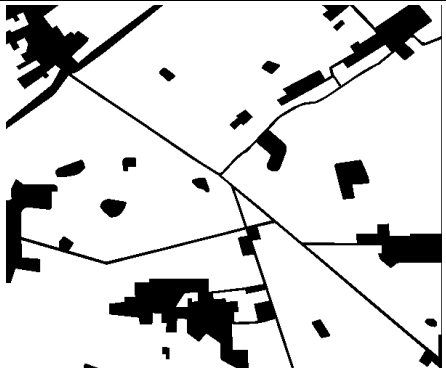

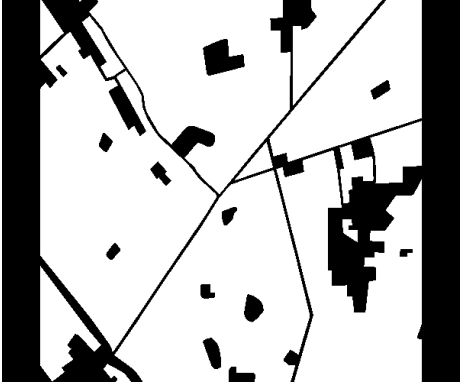
在深度学习图像任务中，为了更好的提取图像特征，一般需要设计具有强大表达能力的深层卷积神经网络。随着深度、宽度的不断增加，网络模型的参数量都是数以百万计的，因此需要大量的图像数据进行训练得到更好的模型，而实际情况中数据并没有我们想象中的那么多，所以需要丰富图像训练集。通常采用两种方法，一种方法是获得新的数据，这种方法比较麻烦，需要大量的成本，而第二种方法则是对数据进行增强。一方面，它可以增加训练的数据量，提高模型的泛化能力；另一方面，它可以增加噪声数据，提升模型的鲁棒性。


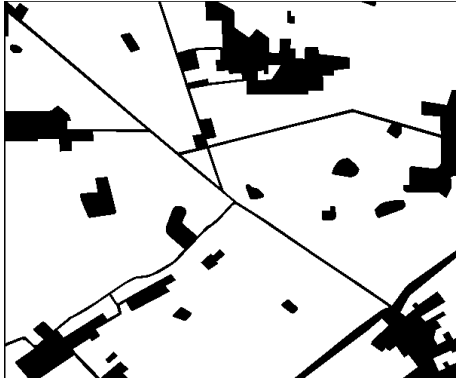

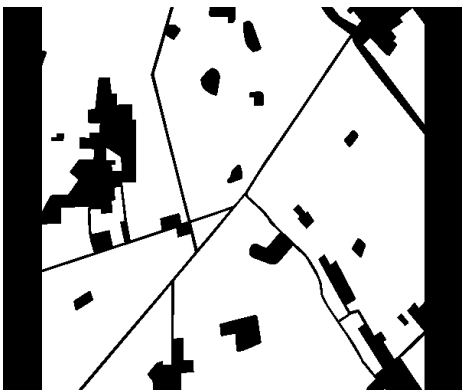

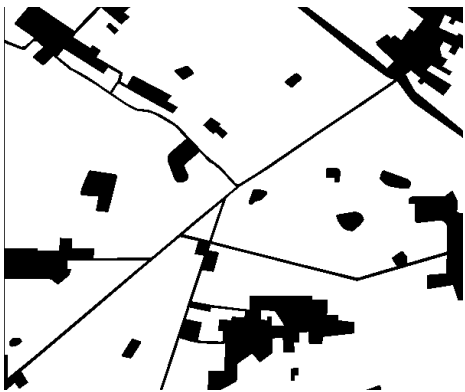

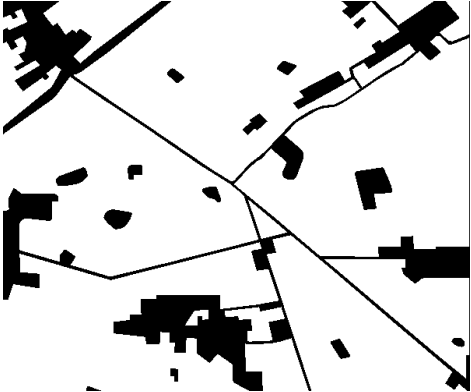
数据增强可以分为两类，一类是离线增强，一类是在线增强。1)离线增强：直接对数据集进行处理，数据的数目会变成增强因子乘以原数据集的数目，这种方法常用于数据集很小的时候。2)在线增强：这种增强的方法用于，获得 batch 数据之后，然后对这个 batch 的数据进行增强，如旋转、平移、翻折等相应的变化，由于有些数据集不能接受线性级别的增长，这种方法常用于大的数据集，很多机器学习框架已经支持了这种数据增强方式，并且可以使用 GPU 优化计算。

数据增强最常用的方式有翻转、旋转、剪切、平移图像，改变图像色差,扭曲图像特征，改变图像尺寸大小，增强图像噪音（一般使用高斯噪音，盐椒噪音）等，但是需要注意，不要加入其他图像轮廓的噪音。




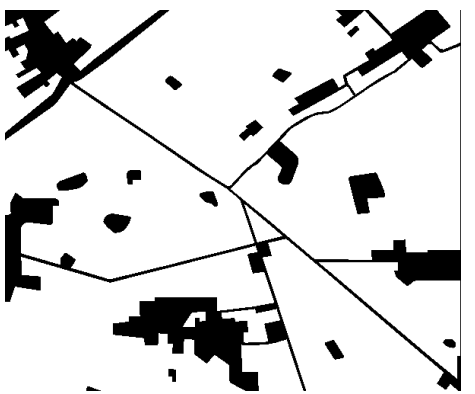

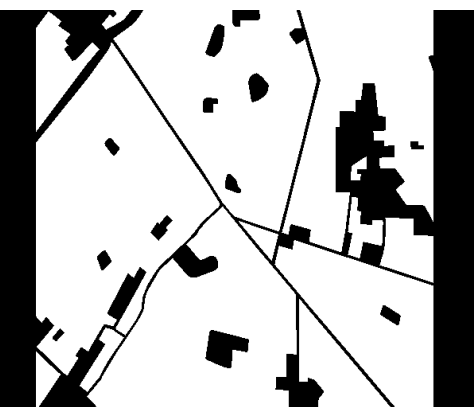


在本题目中，附件提供的训练数据集有 8 张，为了更好的训练语义分割的深度学习模型，本文利用 OpenCV 对所给的 8 张图像进行了离线数据增强，通过旋转 90°、旋转 180°、旋转 270°、翻转、光照调整、模糊操作、增加噪声等方式，将训练数据集扩充到 3000 张。为了直观地看到所给图片在数据增强前后的对比效果，下面对使用的数据增强操作生成的图像进行了可视化。如表格 4-1 所示

表 4-1 数据增强效果

	Image	label
原图		
旋转90°		

<p>旋转180°</p>		
<p>旋转270°</p>		
<p>翻转</p>		
<p>光照调整</p>		



模糊		
增加噪声		
多种方式 组合 1		
多种方式 组合 2		

## 4.2.3 模型一：基于注意力机制的 ResNet 的 FCN1s 网络

### 4.2.3.1 骨干网络 ResNet

深度卷积网络通过多层端到端的方式，集成了低中高三个层次的特征和分类器，并且这些特征的数量还可以通过堆叠层数来增加，从而在图像分类任务上取得了一系列突破。随着网络层数的增加，训练的问题随之凸显。比较显著的问题有梯度消失/爆炸，它会在一开始就影响收敛，但是收敛的问题可以通过正则化来得到部分的解决。

在深层网络能够收敛的前提下，随着网络深度的增加，正确率开始饱和甚至下降，称之为网络的退化(degradation)问题。示例可见图 4-1。显然，56 层的网络相对于 20 层的网络，不管是训练误差还是测试误差都显著增大。

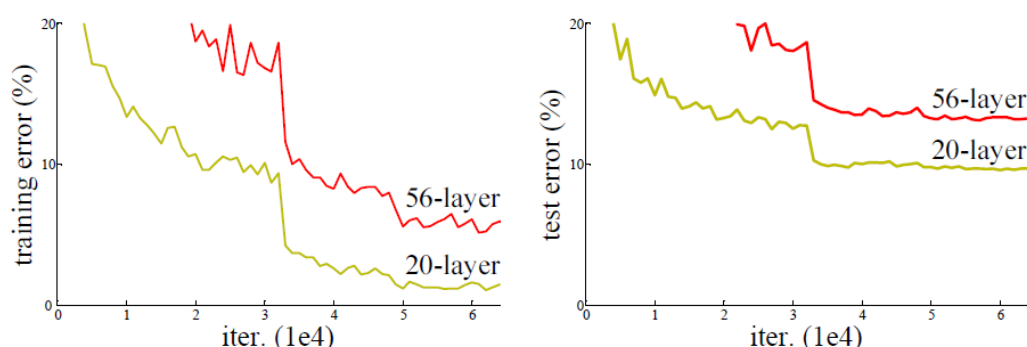


图 4-1 深层网络下训练误差和测试误差变化图

深度残差网络的设计是为了克服由于网络深度加深而产生的学习效率变低与准确率无法有效提升的问题。考虑一个浅层的网络架构和在它基础上构建的深层网络，在极端条件下，如果增加的所有层都是前一层的直接复制（即  $y=x$ ），这种情况下深层网络的训练误差应该和浅层网络相等。因此，网络退化的根本原因还是优化问题。为了解决优化的难题，提出了残差网络。在残差网络中，不是让网络直接拟合原先的映射，而是拟合残差映射。假设原始的映射为  $H(x)$ ，残差网络拟合的映射为： $F(x) := H(x)$ 。实验结果显示，残差网络更容易优化，并且加深网络层数有助于提高正确率。

Residual net(残差网络)重新构建了一种新的网络结构，它将靠前若干层的某一层数据输出直接跳过多层引入到后面数据层的输入部分。意味着后面的特征层的内容会有一部分由其前面的某一层线性贡献，以便学习包含推理的残差函数，而不是学习未经过推理的函数。其结构如下图所示：

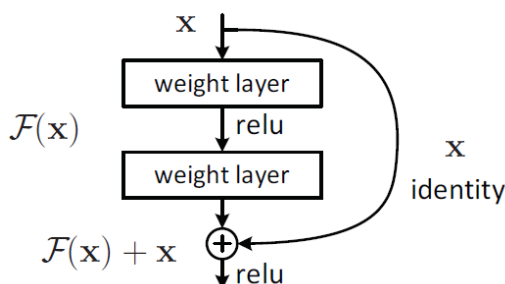


图 4-2 残差网络结构图

ResNet 有两个基本的块，分别名为 Conv Block 和 Identity Block，其中 Conv Block 输入和输出的维度是不一样的，所以不能连续串联，它的作用是改变网络的维度；Identity Block 输入维度和输出维度相同，可以串联，用于加深网络的。

Conv Block 的结构如下：

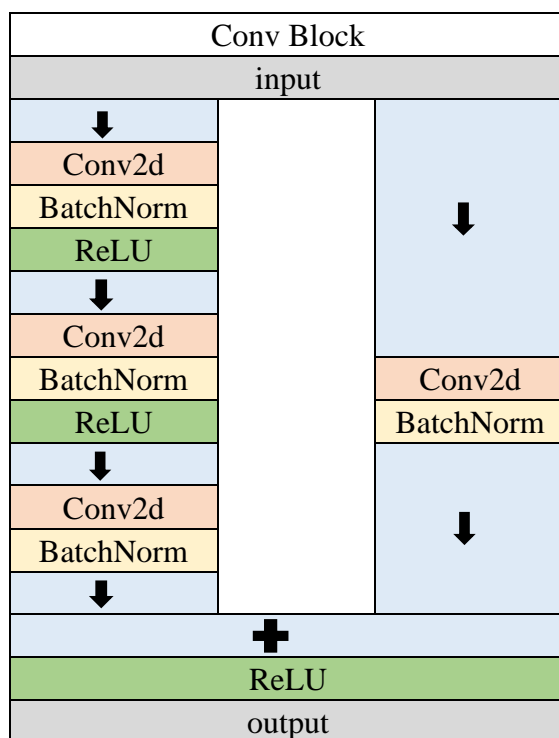


图 4-3 Conv Bloc 结构图

Identity Block 的结构如下：

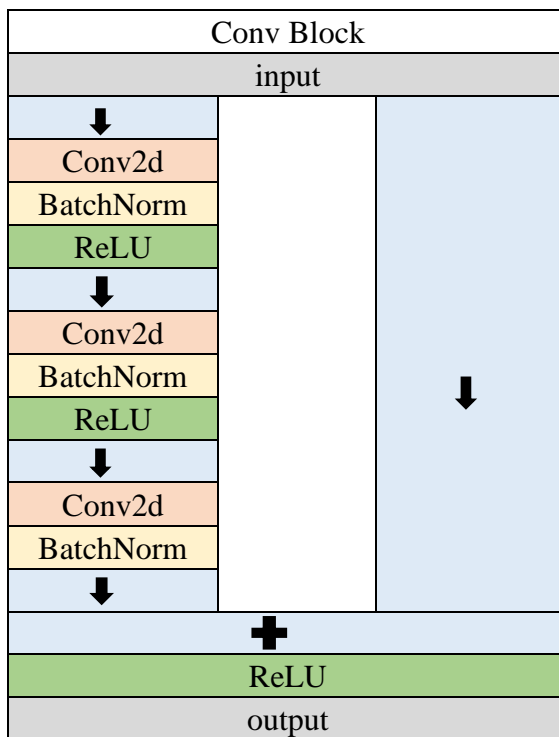


图 4-4 Identity Block 结构图

图 4-5 列出了不同层数下的 ResNet 架构。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

图 4-5 不同深度的 Resnet 架构图

#### 4.3.2.2 注意力机制 SENet

SENet 的全称是 Squeeze-and-Excitation Networks(压缩和激励网络),是 ImageNet 2017 的冠军模型,和 ResNet 的出现类似,都在很大程度上减小了之前模型的错误率,并且复杂度低,新增参数和计算量小。主要由两部分组成:

1. Squeeze 部分。即为压缩部分,原始 feature map 的维度为  $H \times W \times C$ ,其中  $H$  是高度 (Height), $W$  是宽度 (width), $C$  是通道数 (channel)。Squeeze 做的事情是把  $H \times W \times C$  压缩为  $1 \times 1 \times C$ ,相当于把  $H \times W$  压缩成一维了,实际中一般是用 global average pooling 实现的。 $H \times W$  压缩成一维后,相当于这一维参数获得了之前  $H \times W$  全局的视野,感受区域更广。

2. Excitation 部分。得到 Squeeze 的  $1 \times 1 \times C$  的表示后,加入一个 FC 全连接层 (Fully Connected),对每个通道的重要性进行预测,得到不同 channel 的重要性大小后再作用 (激励) 到之前的 feature map 的对应 channel 上,再进行后续操作。

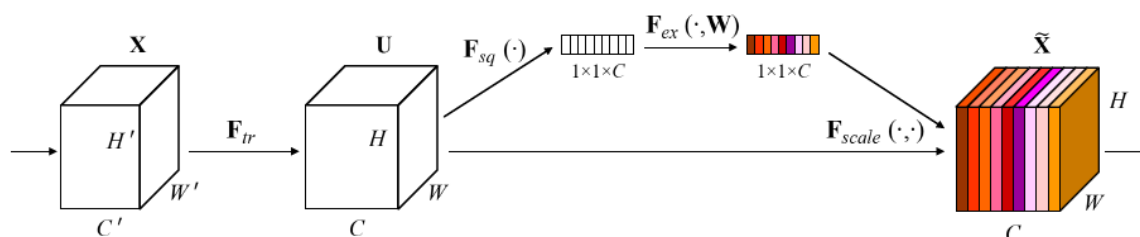


图 4-6 SENet 示意图

可以看出,SENet 和 ResNet 很相似,如下图 4-7,但比 ResNet 做得更多。ResNet 只是增加了一个 skip connection,而 SENet 在相邻两层之间加入了处理,使得 channel 之间的信息交互成为可能,进一步提高了网络的准确率。SENet 可以随意插入到任何网络中,可以一定程度上减少误差,提升效果也是比较显著的。

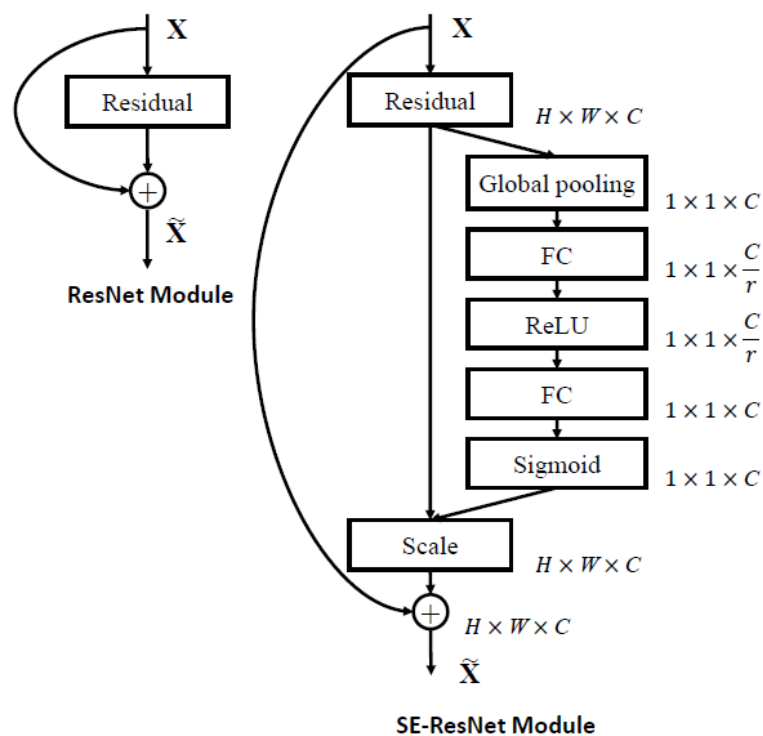


图 4-7 SENet 和 ResNet 对比图

#### 4.2.3.3 语义分割模型 FCN

全卷积网络(FCN)则是从抽象的特征中恢复出每个像素所属的类别。即从图像级别的分类进一步延伸到像素级别的分类。

FCN 将传统卷积网络后面的全连接层换成了卷积层，这样网络输出不再是类别而是 heatmap；同时为了解决因为卷积和池化对图像尺寸的影响，提出使用上采样的方式恢复。

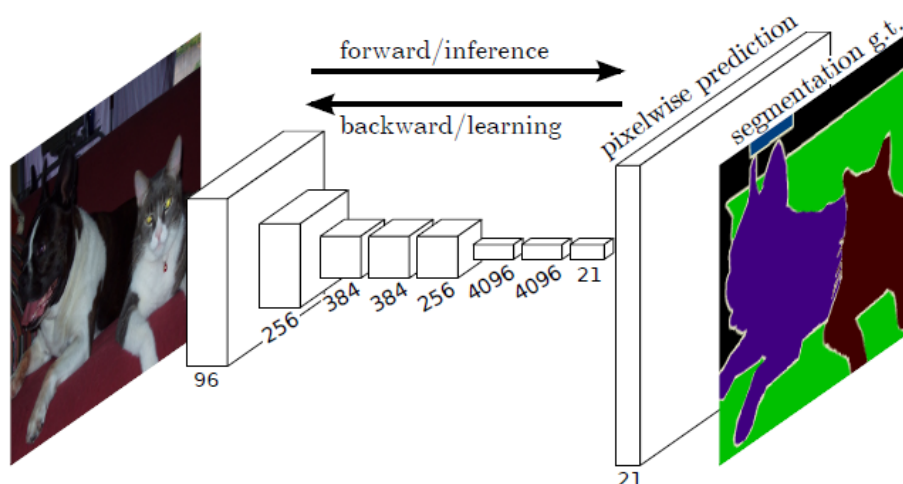


图 4-8 全卷积网络结构图

##### 1.全连接层→全卷积

通常 CNN 网络在卷积层之后会接上若干个全连接层，将卷积层产生的特征图



(feature map)映射成一个固定长度的特征向量。与经典的 CNN 不同, FCN 可以接受任意尺寸的输入图像, 采用反卷积层对最后一个卷积层的 feature map 进行上采样, 使它恢复到输入图像相同的尺寸, 从而可以对每个像素都产生了一个预测, 同时保留了原始输入图像中的空间信息, 最后在上采样的特征图上进行逐像素分类。最后逐个像素计算 softmax 分类的损失, 相当于每一个像素对应一个训练样本。

### 2.上采样-反卷积(deconvolution)

经过多次卷积和 pooling 以后, 得到的图像越来越小, 分辨率越来越低。其中图像小的一层时 (下图的  $H/32 \times W/32$ ), 所产生图叫做 heatmap(热图), 热图就是我们最重要的高维特征图, 得到高维特征的 heatmap 之后就是最重要的一步也是最后的一步对原图像进行 upsampling, 把图像进行不断放大到原图像的大小。最后的输出是  $n$ (类别数)个通道 heatmap 经过 upsampling 变为原图大小的图片, 为了对每个像素进行分类预测 label 成最后已经进行语义分割的图像, 通过逐个像素地求其在  $n$  个通道上该像素位置的最大数值描述 (概率) 作为该像素的分类, 由此产生了一张已经分类好的图片。

### 3.跳跃结构(skip)

经过前面操作, 基本就能实现语义分割了, 但是直接将全卷积后的 heatmap 结果进行反卷积, 得到的结果往往比较粗糙。具体来说, 就是将不同池化层的结果进行上采样, 然后结合这些结果来优化输出, 分为 FCN-32s, FCN-16s, FCN-8s 三种, 第一行对应 FCN-32s, 第二行对应 FCN-16s, 第三行对应 FCN-8s。

具体结构如下:

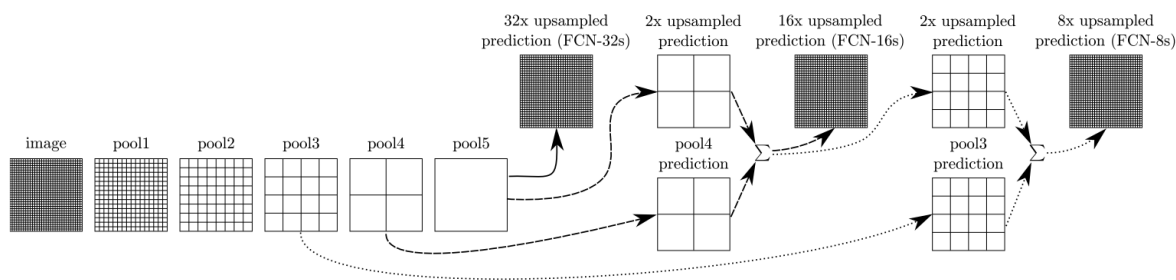


图 4-9 跳跃结构

#### 4.2.3.4 评价指标 Dice 系数

Dice 系数是一种集合相似度度量函数, 通常用于计算两个样本的相似度(值范围为  $[0,1]$ ):

$$s = \frac{2|X \cap Y|}{|X| + |Y|}$$

$|X \cap Y|$  表示  $X$  和  $Y$  之间的交集;  $|X|$  和  $|Y|$  分别表示  $X$  和  $Y$  的元素个数。其中, 分子中的系数 2, 是因为分母存在重复计算  $X$  和  $Y$  之间的共同元素的原因。

#### 4.2.3.5 损失函数 CELoss 和 DiceLoss

BCELoss 用于计算预测值和真实值之间的二元交叉熵损失, 只能用于二分类问题, 而 CrossEntropyLoss 可以用于二分类, 也可以用于多分类。

使用 nn.BCELoss 需要在该层前面加上 Sigmoid 函数。公式如下:



$$loss(X_i, y_i) = -w_i[y_i \log x_i + (1 - y_i) \log(1 - x_i)]$$

使用 nn.CrossEntropyLoss 会自动加上 Softmax 层。公式如下：

$$loss(X_i, y_i) = -w_{label} \log \frac{e^{x_{label}}}{\sum_{j=1}^N e^{x_j}}$$

Dice 系数差异函数(Dice loss):

$$s = 1 - \frac{2|X \cap Y|}{|X| + |Y|}$$

#### 4.2.3.6 SENet+ResNet50+FCN 模型训练

训练配置参数如下表 4-2 所示：

表 4-2 训练配置参数表

SENet+ResNet50+FCN			
基本配置		学习率与优化器	
backbone network	SENet+ResNet50	loss	CELoss、DiceLoss
classes_num	2		
batchsize	24	optimizer	Adam
total_images	3000		
num_train	2550	lr	0.001
num_val	450		
epochs	100	lr_scheduler	StepLR、step_size=5、gamma=0.9
image_shape	[480,480,3]		
笔记本配置: 2 张 GeForce RTX 3090 24G			

模型训练过程如下图 4-10 所示：

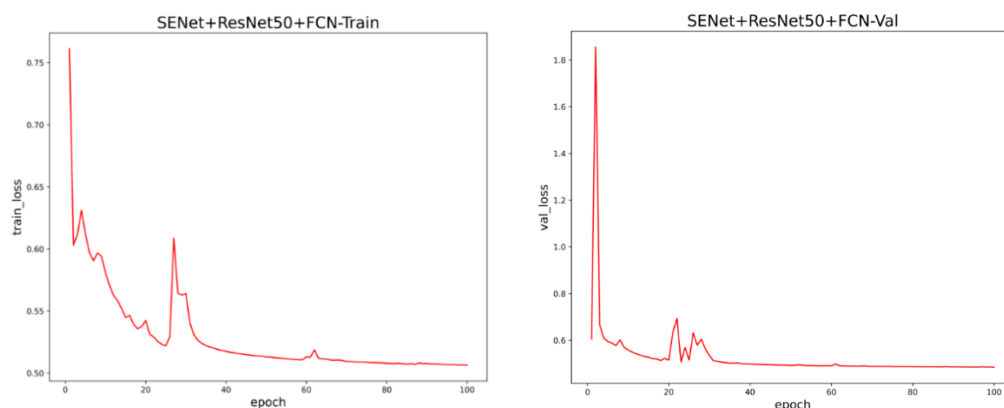



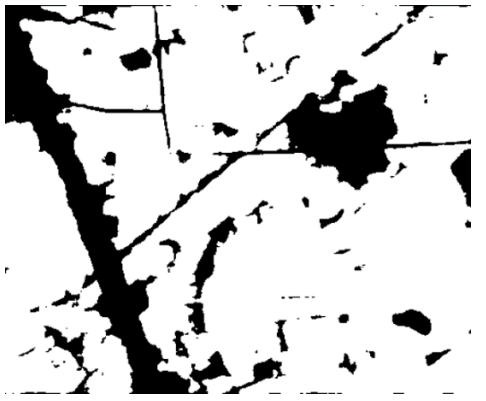


图 4-10 模型训练过程

#### 4.2.3.7 SENet+ResNet50+FCN 模型预测

选取在验证集上 Dice 系数最高的模型文件加载并进行预测，测试集预测结果如表 4-3:

表 4-3 预测结果

SENet+ResNet50+FCN		
Test1		
Test2		

#### 4.2.3.8 SENet+ResNet101+FCN 模型训练

训练配置参数如下表 4-4:

表 4-4 训练配置参数表

SENet+ResNet101+FCN			
基本配置		学习率与优化器	
backbone network	SENet+ResNet101	loss	CELoss、DiceLoss
classes_num	2		
batchsize	16	optimizer	Adam
total_images	3000		
num_train	2550	lr	0.0008
num_val	450		
epochs	100	lr_scheduler	StepLR、step_size=5、gamma=0.9
image_shape	[480,480,3]		

模型训练过程如下图 4-11 所示:

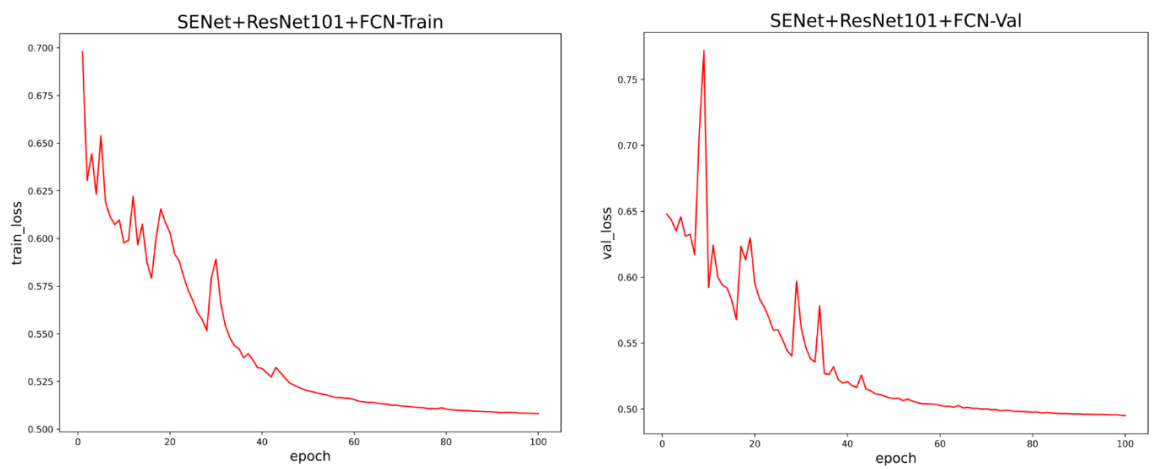




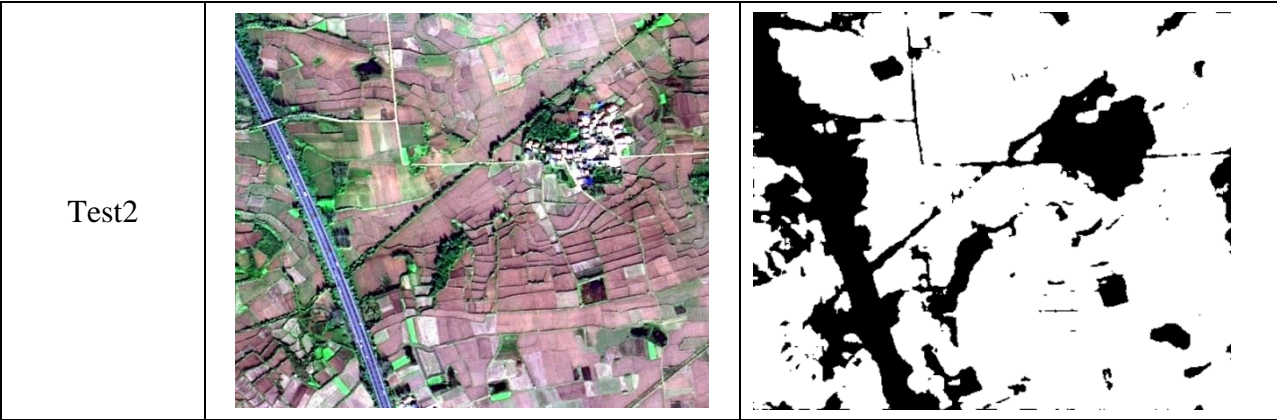
图 4-11 损失函数变化图

4.2.3.9 SENet+ResNet101+FCN 模型预测

选取在验证集上 Dice 系数最高的模型文件加载并进行预测，测试集预测结果如下表 4-5:

表 4-5 测试集预测结果

SENet+ResNet101+FCN		
Test1		



#### 4.2.4 模型二：Attention U-Net 网络

##### 4.2.4.1 U-Net

U-Net 是医学图像处理方面著名的图像分割网络，通过使用包含压缩路径和扩展路径的对称 U 形结构在当时非常具有创新性，且一定程度上影响了后面若干个分割网络的设计。整个网络结构如下图：

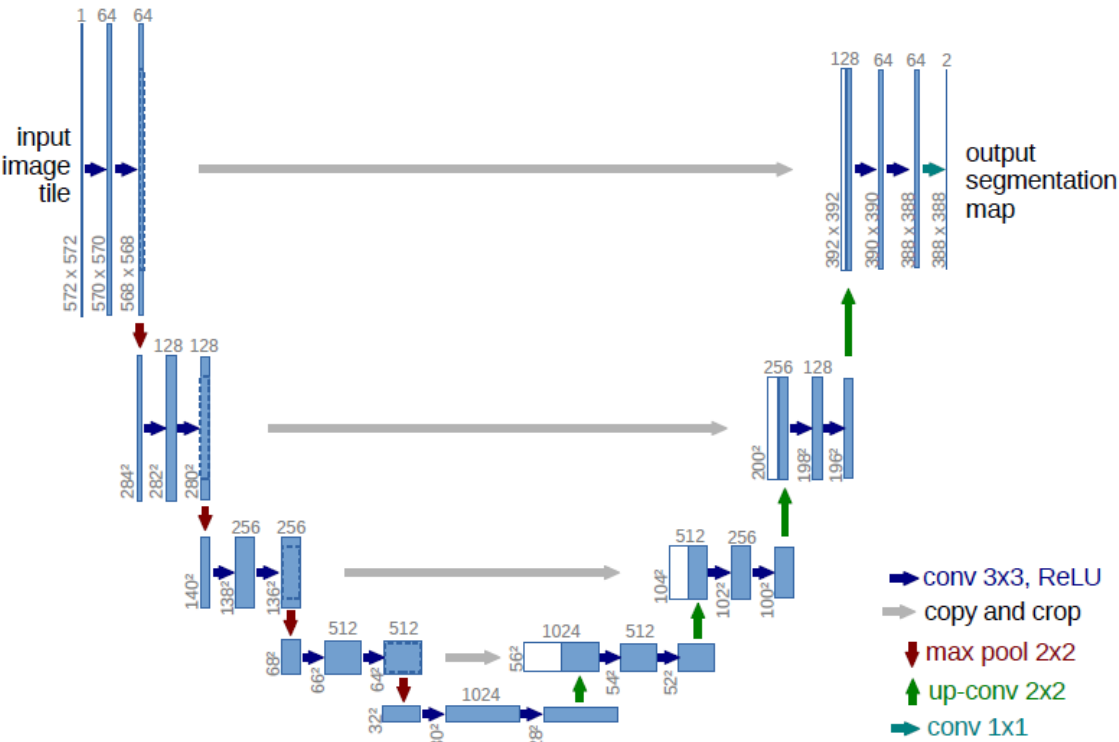


图 4-12 UNet 结构图

U-Net 网络结构如其名呈现一个 U 字形，蓝色代表卷积和激活函数，灰色代表复制，红色代表下采样，绿色代表上采样然后在卷积，conv 1X1 代表核为 1X1 的卷积操作，可以看出这个网络没有全连接，只有卷积和下采样。这也是一个端到端的图像，即输入是一

幅图像，输出也是一副图像。较浅的高分辨率层用来解决像素定位的问题，较深的层用来解决像素分类的问题。

可以看出，该网络结构主要分为三部分：下采样，上采样以及跳跃连接。首先将该网络分为左右部分来分析，左边是压缩的过程，即 **Encoder**。通过卷积和下采样来降低图像尺寸，提取一些浅显的特征。右边部分是解码的过程，即 **Decoder**。通过卷积和上采样来获取一些深层次的特征。其中卷积采用的 **valid** 的填充方式来保证结果都是基于没有缺失上下文特征得到的，因此每次经过卷积后，图像的大小会减小。中间通过 **concat** 的方式，将编码阶段获得的 **feature map** 同解码阶段获得的 **feature map** 结合在一起，由于这里两层的 **feature map** 大小不同，因此需要经过切割。最后一层通过 **1x1** 的卷积做分类。结合深层次和浅层次的特征，细化图像，根据得到的 **feature map** 进行预测分割。

#### 4.2.4.2 Attention U-Net

本文提出了基于注意门（AGs）的 Attention U-Net 通过抑制不相关区域的激活值从而提升模型的敏感度和精度，用来自学习图像显著性特征，文将 AGs 引入 U-Net 网络用于遥感图像地块的分割。如下图：

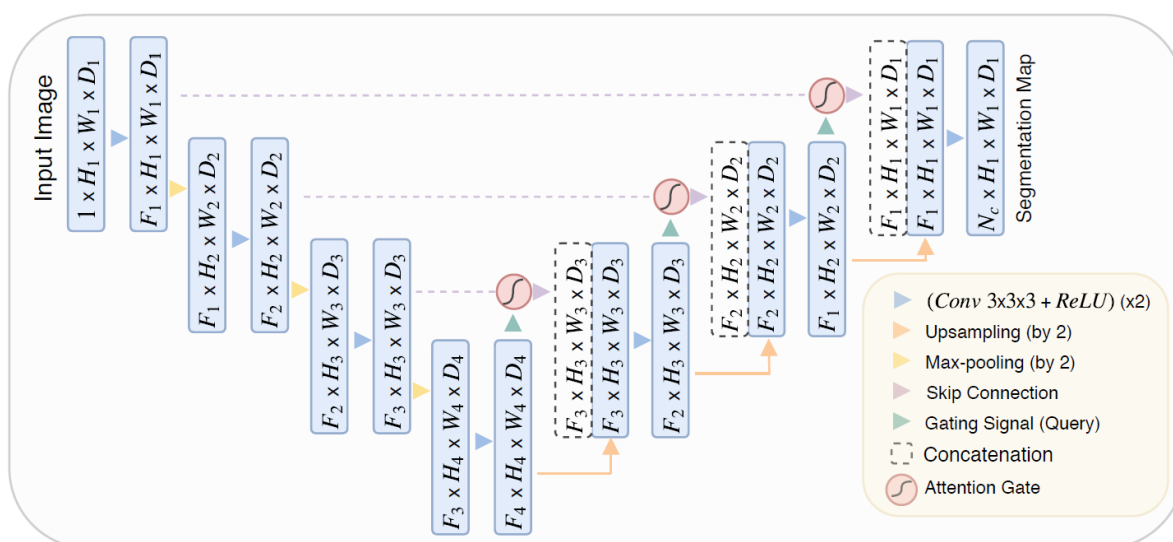


图 4-13 Attention U-Net 结构图

**Attention Gate** 注意门可以用于学习不同形状/大小的目标，通过有选择性的学习输入图像中相互关联的区域，抑制不相关区域的显著性，可以避免在网络搭建过程中引入额外人为的监督。如图 4-14 所示，另一方面，注意门（AGs）可以作为一种即插即用的模块引入各种网络从而提升模型的敏感度和精度。

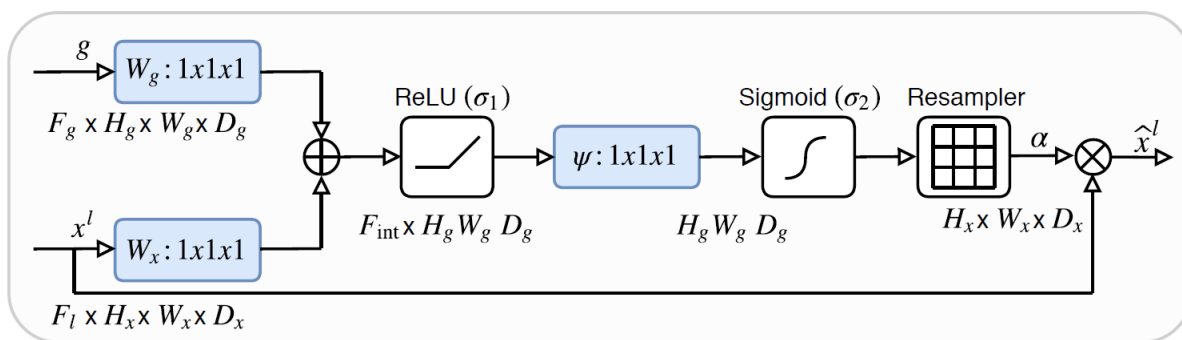


图 4-13 Attention Gate 注意门结构图

首先，下采样层同层的特征图  $g_i(F_g \times H_g \times W_g \times D_g)$  进行  $1*1*1$  卷积运算得到  $W_g^T g_i$ ，上采样层上一层的特征图  $x_i^l(F_l \times H_x \times W_x \times D_x)$ ，进行  $1*1*1$  卷积运算得到  $W_x^T x_i^l$ ，将上两步得到的特征图  $W_g^T g_i$  和  $W_x^T x_i^l$  进行相加后在进行 ReLU 得到  $\sigma_1(W_x^T x_i^l + W_g^T g_i + b_g)(F_{int} \times H_g \times W_g \times D_g)$ ， $\sigma_1$  为 ReLU 激活函数。随后在使用  $1*1*1$  卷积运算得到  $q_{att}^l = \psi^T(\sigma_1(W_x^T x_i^l + W_g^T g_i + b_g)) + b_\psi$ ，最后对  $q_{att}^l$  进行 sigmoid 激活函数  $\alpha_i^l = \sigma_2(q_{att}^l(x_i^l, g_i; \Theta_{att}))$ ，得到最终的 attention coefficients。这个重要度系数是根据我们的目标来学习出来的，也就是我们引入 Attention 来学习到各个元素与目标之间的重要度。

#### 4.2.4.3 Attention U-Net 模型训练

训练配置参数如下表 4-6 所示：

表 4-6 模型训练参数设置

Attention U-Net			
基本配置		学习率与优化器	
backbone network	/	loss	CELoss、DiceLoss
classes_num	2		
batchsize	16	optimizer	Adam
total_images	3000		
num_train	2550	lr	0.001
num_val	450		
epochs	100	lr_scheduler	StepLR、step_size=2、gamma=0.9
image_shape	[480,480,3]		
笔记本配置: 2 张 GeForce RTX 3090 24G			



模型训练过程如下图 4-14 所示：

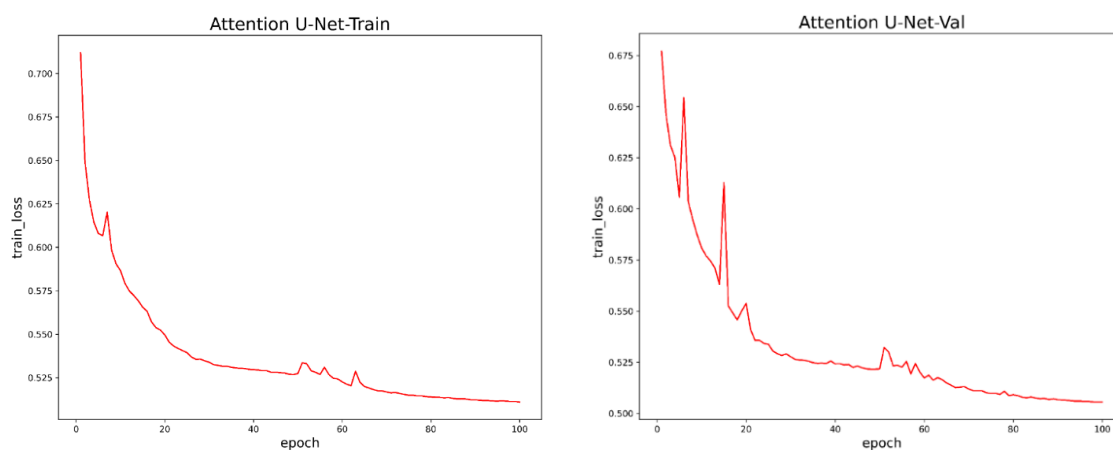






图 4-14 损失函数变化图

#### 4.2.4.4 Attention U-Net 模型预测

选取在验证集上 Dice 系数最高的模型文件加载并进行预测，测试集预测结果：


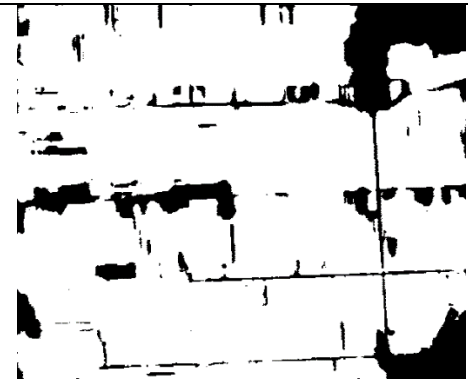


表 4-7 预测结果

Attention U-Net		
Test1		
Test2		

4.2.5 模型融合

在上述求解中，我们先后设计了SENet+ResNet50+FCN1s、SENet+ResNet101+FCN1s、Attention U-Net三个不同的深度学习语义分割模型，不同的模型采取了不同参数去进行训练和预测，就会得到很多预测MASK图。为了改善模型的预测能力，可以采取模型融合的思路，对每张结果图的每个像素点采取少数服从多数的投票表决策略，对每张图相应位置的像素点的类别进行预测，票数最多的类别即为该像素点的类别。正所谓“三个臭皮匠，胜过诸葛亮”，采用集成学习方法，这种ensemble的思路可以很好地去掉一些明显分类错误的像素点，从而提高模型的鲁棒性。模型融合后的结果见表4-8：

表 4-8 预测结果

模型融合		
Test1		
Test2		



## 五、问题 3

利用高分辨率遥感影像进行地块边界及其空间分布的提取,不仅时效性强、精度高,而且符合中国农村高度分散条件下的精准农业的实施。

传统的遥感农田道路的信息提取主要有三种方法: 1)基于像元尺度的影像自动分类技术,该方法效率较高,但受异物同谱等因素的影响,精度较低; 2)人机交互模式下的人工解译提取技术,该方法精度较高,但对解译人员有较高的要求,且效率较低; 3)自动识别跟踪方法,该方法是介于自动与半自动之间的方法,在自动识别提取线性地物后再进行人工取舍,应用较为广泛。依靠人工数字化的方法来提取耕地地块不仅费时费力,而且需要解译人员必须具有丰富的经验,而深度学习中的图像语义分割技术则能利用高分辨率遥感影像来自动提取耕地地块,通过前面的问题求解可以得知,它取得较好的效果,并逐渐成为耕地地块遥感提取的主要方法。

为了能快速、精准的识别出田块,可以从以下几点进行新的研究:

1) 采用面向对象的多尺度分割技术,它可以有效地利用所要提取对象的形态特征,从而取得更好的效果。

2) 综合地利用田块以及周边道路和水渠等特殊地物的几何特征、辐射特征、拓扑特征及上下文特征,以提高自动提取的效率和精度。

3) 选择高质量的数据源,所选择的数据要具有较高的空间分辨率,以确保地块边界的准确性;需要有较高光谱分辨率的光谱信息,以便区分不同地块在土壤类型和质地上的差别,新型星载遥感数据(如 RapidEye、WorldView-2)的出现将提高地块提取的精度和效率。

## 六、模型总结和改进

### 6.1 模型的优点

1) 针对附件所给数据集过少的问题，本文利用了数据增强技术，如翻转、旋转、剪裁、光亮度调整等角度，将数据量扩大到3000，便于神经网络模型进行更好的训练；

2) 针对遥感地块分割任务存在耕地和背景类别不均衡的问题，本文将损失函数设计为CELoss和DiceLoss二者的加权和，并以Dice系数为评价指标引导模型进行更好的训练，使其训练的过程与我们的目标是一致的；

3) 针对存在的连通域问题，即田块间的边界是否清晰，本文先后将注意力机制加入不同的模型中，并取得好的效果；

4) 本文先后设计了SENet+ResNet50+FCN1s、SENet+ResNet101+FCN1s、Attention U-Net、三个不同的深度学习语义分割模型，并做个模型的融合，多模型投票机制使预测结果的鲁棒性更好。

### 6.2 模型的缺点

1) 没有深入探究数据量、不同骨干网络对于语义分割模型预测结果的对比分析；

2) 在本任务中，没有利用传统图像分割算法进行分割和提取。

### 6.3 未来的工作

尝试新型图像语义分割模型，如OCRNet、图神经网络等，将不同类型的注意力机制嵌入网络进行训练，对于模型预测的结果，可以利用行业领域的专业知识对其进行后处理，以此达到更好的预测效果，从而更好的完成任务。

## 七、参考文献

- [1] 胡潭高, et al. "高分辨率遥感图像耕地地块提取方法研究." 光谱学与光谱分析 10 (2009): 2703-2707.
- [2] 庞新华, et al. "基于高分辨率遥感影像的耕地地块提取方法研究." 测绘科学 34.1 (2009): 48-49.
- [3] 张涛, et al. "基于多尺度图像库的遥感影像分割参数优选方法." 国土资源遥感 28.4 (2016): 59-63.
- [4] 付卓, et al. "遥感应用分析中影像分割方法." 遥感技术与应用 21.5 (2006): 456-462.
- [5] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
- [6] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
- [7] Hu, Jie, Li Shen, and Gang Sun. "Squeeze-and-excitation networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
- [8] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).
- [9] Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
- [10] Howard, Andrew, et al. "Searching for mobilenetv3." Proceedings of the IEEE International Conference on Computer Vision. 2019.
- [11] Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." IEEE transactions on pattern analysis and machine intelligence 39.12 (2017): 2481-2495.
- [12] Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
- [13] 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.
- [14] Zhao, Hengshuang, et al. "Pyramid scene parsing network." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
- [15] Oktay, Ozan, et al. "Attention u-net: Learning where to look for the pancreas." arXiv preprint arXiv:1804.03999 (2018).
- [16] Zhou, Zongwei, et al. "Unet++: A nested u-net architecture for medical image segmentation." Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support. Springer, Cham, 2018. 3-11.

## 附录

### 代码 1: 问题一

```
# -*- coding: UTF-8 -*-
import os
import numpy as np
from matplotlib import pyplot as plt
import cv2
%matplotlib inline

img00 = cv2.imread("work/dataset/dataset/预览图/Data8.png")
img01 = cv2.imread("work/dataset/dataset/预览图/Data8_reference.png")

plt.figure(figsize=(8,8))

plt.subplot(1,2,1)
plt.title('origin image')
plt.imshow(img00)

plt.subplot(1,2,2)
plt.title('label image')
plt.imshow(img01)

# 显示是三通道
print(img01.shape)
# 取一个通道
img01 = img01[:, :, 0]
# # 像素值个数
# print(len(img01.flatten()))

# # 方式一:
# from collections import Counter
# print(Counter(img01.flatten()))
# 方式二:
mask = np.unique(img01)
tmp = {}
for v in mask:
    tmp[v] = [np.sum(img01 == v)] + [np.sum(img01 == v)/len(img01.flatten
())]
print("mask 值为: ")
print(mask)
print("统计结果: ")
print(tmp)
```

## 代码 2：问题二—数据增强

```
# -*- coding: UTF-8 -*-
import cv2
import random
import os
from tqdm import tqdm

img_w, img_h = 600, 500

def rotate(src_img, label_img, angle):
    """
    旋转
    """
    M_rotate = cv2.getRotationMatrix2D((img_w/2, img_h/2), angle, 1)
    src_img = cv2.warpAffine(src_img, M_rotate, (img_w, img_h))
    label_img = cv2.warpAffine(label_img, M_rotate, (img_w, img_h))
    return src_img, label_img

def gamma_transform(src_img, gamma):
    """
    光照增强
    """
    gamma_table = [np.power(x / 255.0, gamma) * 255.0 for x in range(256)]
    gamma_table = np.round(np.array(gamma_table)).astype(np.uint8)
    return cv2.LUT(src_img, gamma_table)

def random_gamma_transform(src_img, gamma_vari):
    log_gamma_vari = np.log(gamma_vari)
    alpha = np.random.uniform(-log_gamma_vari, log_gamma_vari)
    gamma = np.exp(alpha)
    return gamma_transform(src_img, gamma)

def blur(src_img):
    """
    模糊--均值滤波
    """
    src_img = cv2.blur(src_img, (3, 3));
    return src_img

def add_noise(src_img):
    """
    添加点噪声
    """
```

```

    for i in range(200):
        temp_x = np.random.randint(0,src_img.shape[0])
        temp_y = np.random.randint(0,src_img.shape[1])
        src_img[temp_x][temp_y] = 255
    return src_img

def data_augment(src_img, label_img):
    # 旋转90°
    if np.random.random() < 0.20:
        src_img,label_img = rotate(src_img,label_img,90)
    # 旋转180°
    if np.random.random() < 0.35:
        src_img,label_img = rotate(src_img,label_img,180)
    # 旋转270°
    if np.random.random() < 0.25:
        src_img,label_img = rotate(src_img,label_img,270)
    # flipcode > 0: 沿y轴翻转
    if np.random.random() < 0.50:
        src_img = cv2.flip(src_img, 1)
        label_img = cv2.flip(label_img, 1)
    # 光照调整
    if np.random.random() < 0.25:
        src_img = random_gamma_transform(src_img,1.0)
    # 模糊
    if np.random.random() < 0.25:
        src_img = blur(src_img)
    # 增加噪声 (高斯噪声, 椒盐噪声)
    if np.random.random() < 0.2:
        src_img = add_noise(src_img)
    return src_img, label_img

def creat_dataset(image_num=3000):
    print('creating dataset...')
    image_each = image_num / len(image_sets)
    g_count = 0
    for i in tqdm(range(len(image_sets))):
        count = 0
        # image -> 3 channels
        src_img = cv2.imread('dataset/src/' + image_sets[i])
        # label -> single channel
        label_img = cv2.imread('dataset/label/' + image_sets[i], cv2.IMREAD_
GRAYSCALE)
        X_height, X_width, _ = src_img.shape
        while count < image_each:
            src_aug, label_aug = data_augment(src_img, label_img)

```

```

        visualize = np.zeros((256,256)).astype(np.uint8)
        visualize = label_aug * 50
        cv2.imwrite('/home/wp/pytorch/FCN/VOCdevkit/VOC2012/JPEGImages
/%d.png' % g_count), src_aug)
        cv2.imwrite('/home/wp/pytorch/FCN/VOCdevkit/VOC2012/Segmentatio
nClass/%d.png' % g_count), label_aug)
        count += 1
        g_count += 1

if __name__ == '__main__':
    image_sets = ['Data1.png', 'Data2.png', 'Data3.png', 'Data4.png', 'Data5.png',
'Data6.png', 'Data7.png', 'Data8.png']
    creat_dataset()

```

### 代码 3：问题二—SENet+ResNet50+FCN1s

```

# -*- coding: UTF-8 -*-
import torch
import torch.nn as nn
import torchvision
import torch.nn.functional as F
import torch.optim as optim
from torchvision import models
from torch.nn import init
import numpy as np

def Conv1(in_planes, places):
    return nn.Sequential(
        nn.Conv2d(in_channels=in_planes, out_channels=places, kernel_size=
7, stride=1, padding=3, bias=False),
        nn.BatchNorm2d(places),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=3, stride=2, padding=1))

class SE_Module(nn.Module):
    def __init__(self, channel, ratio=16):
        super(SE_Module, self).__init__()
        self.squeeze = nn.AdaptiveAvgPool2d(1)
        self.excitation = nn.Sequential(
            nn.Linear(in_features=channel, out_features=channel // rati
o),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=channel // ratio, out_features=channe
l),

```

```

        nn.Sigmoid())
def forward(self, x):
    b, c, _, _ = x.size()
    y = self.squeeze(x).view(b, c)
    z = self.excitation(y).view(b, c, 1, 1)
    return x * z.expand_as(x)

class SE_ResNetBlock(nn.Module):
    def __init__(self, in_places, places, stride=1, downsampling=False, expansion=4):
        super(SE_ResNetBlock, self).__init__()
        self.expansion = expansion
        self.downsampling = downsampling

        self.bottleneck = nn.Sequential(
            nn.Conv2d(in_channels=in_places, out_channels=places, kernel_size=1, stride=1, bias=False),
            nn.BatchNorm2d(places),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=places, out_channels=places, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(places),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=places, out_channels=places*self.expansion, kernel_size=1, stride=1, bias=False),
            nn.BatchNorm2d(places*self.expansion))

        if self.downsampling:
            self.downsample = nn.Sequential(
                nn.Conv2d(in_channels=in_places, out_channels=places*self.expansion, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(places*self.expansion))
            self.relu = nn.ReLU(inplace=True)
            self.sigmoid = nn.Sigmoid()
            # SE layers
            self.fc1 = nn.Conv2d(places*self.expansion, places*self.expansion//16, kernel_size=1) # Use nn.Conv2d instead of nn.Linear
            self.fc2 = nn.Conv2d(places*self.expansion//16, places*self.expansion, kernel_size=1)

    def forward(self, x):
        residual = x
        out = self.bottleneck(x)
        # Squeeze

```



```

w = F.avg_pool2d(out, out.size(2))
w = self.relu(self.fc1(w))
w = self.sigmoid(self.fc2(w))
# Excitation
out = out * w # New broadcasting feature from v0.2!
if self.downsampling:
    residual = self.downsample(x)

out += residual
out = self.relu(out)
return out

class SE_ResNet(nn.Module):
    def __init__(self, blocks, num_classes=1000, expansion=4):
        super(SE_ResNet, self).__init__()
        self.expansion = expansion

        self.conv1 = Conv1(in_planes=3, places=64)

        self.layer1 = self.make_layer(in_places=64, places=64, block=blocks
[0], stride=2)
        self.layer2 = self.make_layer(in_places=256, places=128, block=block
s[1], stride=2)
        self.layer3 = self.make_layer(in_places=512, places=256, block=block
s[2], stride=2)
        self.layer4 = self.make_layer(in_places=1024, places=512, block=bloc
ks[3], stride=2)

        self.avgpool = nn.AvgPool2d(7, stride=1)
        self.fc = nn.Linear(2048, num_classes)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearit
y='relu')
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

        def make_layer(self, in_places, places, block, stride):
            layers = []
            layers.append(SE_ResNetBlock(in_places, places, stride, downsamplin
g=True))
            for i in range(1, block):
                layers.append(SE_ResNetBlock(places*self.expansion, places))

```

```

    return nn.Sequential(*layers)

def forward(self, x):
    out_put = []
    x = self.conv1(x)
    out_put.append(x)
    x = self.layer1(x)
    out_put.append(x)
    x = self.layer2(x)
    out_put.append(x)
    x = self.layer3(x)
    out_put.append(x)
    x = self.layer4(x)
    out_put.append(x)

    #x = self.avgpool(x)
    #x = x.view(x.size(0), -1)
    #x = self.fc(x)
    return out_put

def SE_ResNet50():
    return SE_ResNet([3, 4, 6, 3])
    #return SE_ResNet([3, 4, 23, 3])

class FCN1s(nn.Module):
    def __init__(self, pretrained_net, n_class):
        super().__init__()
        self.n_class = n_class
        self.pretrained_net = pretrained_net
        self.relu = nn.ReLU(inplace=True)
        self.deconv1 = nn.ConvTranspose2d(2048, 1024, kernel_size=3, stride=
2, padding=1, dilation=1, output_padding=1)
        self.bn1 = nn.BatchNorm2d(1024)
        self.deconv2 = nn.ConvTranspose2d(1024, 512, kernel_size=3, stride=
2, padding=1, dilation=1, output_padding=1)
        self.bn2 = nn.BatchNorm2d(512)
        self.deconv3 = nn.ConvTranspose2d(512, 256, kernel_size=3, stride=2,
padding=1, dilation=1, output_padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.deconv4 = nn.ConvTranspose2d(256, 64, kernel_size=3, stride=2,
padding=1, dilation=1, output_padding=1)
        self.bn4 = nn.BatchNorm2d(64)
        self.deconv5 = nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, p

```

```

adding=1, dilation=1, output_padding=1)
    self.bn5 = nn.BatchNorm2d(32)
    self.classifier = nn.Conv2d(32, n_class, kernel_size=1)

    def forward(self, x):
        output = self.pretrained_net.forward(x)
        x5 = output[4] # [None, 2048, 15, 15]
        x4 = output[3] # [None, 1024, 30, 30]
        x3 = output[2] # [None, 512, 60, 60]
        x2 = output[1] # [None, 256, 120, 120]
        x1 = output[0] # [None, 64, 240, 240]
        score = self.relu(self.deconv1(x5))
        score = self.bn1(score+x4)
        #print(score.shape)
        score = self.relu(self.deconv2(score))
        score = self.bn2(score+x3)
        #print(score.shape)
        score = self.relu(self.deconv3(score))
        score = self.bn3(score+x2)
        #print(score.shape)
        score = self.relu(self.deconv4(score))
        score = self.bn4(score+x1)
        #print(score.shape)
        score = self.relu(self.deconv5(score))
        #print(score.shape)
        score = self.classifier(score)
        return score

```

#### 代码 4： 问题二—Attention U-Net

```

# -*- coding: UTF-8 -*-
from torch import nn
from torch.nn import functional as F
import torch
from torchvision import models
import torchvision

class conv_block(nn.Module):
    def __init__(self, ch_in, ch_out):
        super(conv_block, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(ch_in, ch_out, kernel_size=3, stride=1, padding=1, bias=True),
            nn.BatchNorm2d(ch_out),

```

```

        nn.ReLU(inplace=True),
        nn.Conv2d(ch_out, ch_out, kernel_size=3, stride=1, padding=1, bias=True),
        nn.BatchNorm2d(ch_out),
        nn.ReLU(inplace=True)
    )
    def forward(self, x):
        x = self.conv(x)
        return x

class up_conv(nn.Module):
    def __init__(self, ch_in, ch_out):
        super(up_conv, self).__init__()
        self.up = nn.Sequential(
            nn.Upsample(scale_factor=2),
            nn.Conv2d(ch_in, ch_out, kernel_size=3, stride=1, padding=1, bias=True),
            nn.BatchNorm2d(ch_out),
            nn.ReLU(inplace=True)
        )
    def forward(self, x):
        x = self.up(x)
        return x

class Attention_block(nn.Module):
    def __init__(self, F_g, F_l, F_int):
        super(Attention_block, self).__init__()
        self.W_g = nn.Sequential(
            nn.Conv2d(F_g, F_int, kernel_size=1, stride=1, padding=0, bias=True),
            nn.BatchNorm2d(F_int)
        )
        self.W_x = nn.Sequential(
            nn.Conv2d(F_l, F_int, kernel_size=1, stride=1, padding=0, bias=True),
            nn.BatchNorm2d(F_int)
        )
        self.psi = nn.Sequential(
            nn.Conv2d(F_int, 1, kernel_size=1, stride=1, padding=0, bias=True),
            nn.BatchNorm2d(1),
            nn.Sigmoid()
        )
        self.relu = nn.ReLU(inplace=True)
    def forward(self, g, x):

```

```

# 下采样的gating signal 卷积
g1 = self.W_g(g)
# 上采样的 l 卷积
x1 = self.W_x(x)
# concat + relu
psi = self.relu(g1 + x1)
# channel 减为1, 并Sigmoid, 得到权重矩阵
psi = self.psi(psi)
# 返回加权的 x
return x * psi

```

```

class AttU_Net(nn.Module):
    def __init__(self, img_ch=3, output_ch=2):
        super(AttU_Net, self).__init__()
        self.Maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.Conv1 = conv_block(ch_in=img_ch, ch_out=64)
        self.Conv2 = conv_block(ch_in=64, ch_out=128)
        self.Conv3 = conv_block(ch_in=128, ch_out=256)
        self.Conv4 = conv_block(ch_in=256, ch_out=512)
        self.Conv5 = conv_block(ch_in=512, ch_out=1024)
        self.Up5 = up_conv(ch_in=1024, ch_out=512)
        self.Att5 = Attention_block(F_g=512, F_l=512, F_int=256)
        self.Up_conv5 = conv_block(ch_in=1024, ch_out=512)
        self.Up4 = up_conv(ch_in=512, ch_out=256)
        self.Att4 = Attention_block(F_g=256, F_l=256, F_int=128)
        self.Up_conv4 = conv_block(ch_in=512, ch_out=256)
        self.Up3 = up_conv(ch_in=256, ch_out=128)
        self.Att3 = Attention_block(F_g=128, F_l=128, F_int=64)
        self.Up_conv3 = conv_block(ch_in=256, ch_out=128)
        self.Up2 = up_conv(ch_in=128, ch_out=64)
        self.Att2 = Attention_block(F_g=64, F_l=64, F_int=32)
        self.Up_conv2 = conv_block(ch_in=128, ch_out=64)
        self.Conv_1x1 = nn.Conv2d(64, output_ch, kernel_size=1, stride=1, padding=0)

        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        # encoding path
        x1 = self.Conv1(x)
        x2 = self.Maxpool(x1)
        x2 = self.Conv2(x2)
        x3 = self.Maxpool(x2)
        x3 = self.Conv3(x3)
        x4 = self.Maxpool(x3)
        x4 = self.Conv4(x4)

```

```

x5 = self.Maxpool(x4)
x5 = self.Conv5(x5)
# decoding + concat path
d5 = self.Up5(x5)
x4 = self.Att5(g=d5, x=x4)
d5 = torch.cat((x4, d5), dim=1)
d5 = self.Up_conv5(d5)
d4 = self.Up4(d5)
x3 = self.Att4(g=d4, x=x3)
d4 = torch.cat((x3, d4), dim=1)
d4 = self.Up_conv4(d4)
d3 = self.Up3(d4)
x2 = self.Att3(g=d3, x=x2)
d3 = torch.cat((x2, d3), dim=1)
d3 = self.Up_conv3(d3)
d2 = self.Up2(d3)
x1 = self.Att2(g=d2, x=x1)
d2 = torch.cat((x1, d2), dim=1)
d2 = self.Up_conv2(d2)
d1 = self.Conv_1x1(d2)
#d1 = self.sigmoid(d1)
return d1

```

```

if __name__ == '__main__':
    model = AttU_Net(img_ch=3, output_ch=2)
    print(model)

    input = torch.randn(1, 3, 480, 480)
    out = model(input)
    print(out.shape)

```

## 代码 5：问题二—模型训练

```

# -*- coding: UTF-8 -*-
import time
import torch
import torch.optim as optim
import torch.backends.cudnn as cudnn
import torchvision.models as models
import numpy as np
from tqdm import tqdm
from torchvision import models
from torch.autograd import Variable
from PIL import Image

```

```

from torch import nn
import models
from fcn_training import CE_Loss, Dice_loss
from utils.metrics import f_score
from torch.utils.data import DataLoader
from utils.dataloader import fcn_dataset_collate, FCNDataset
from fcn_senet import SE_ResNet50, FCN1s
from unet_senet import AttU_Net
from unet import UNet
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

def get_lr(optimizer):
    """
    学习率
    """
    for param_group in optimizer.param_groups:
        return param_group['lr']

def fit_one_epoch(net, epoch, epoch_size_train, epoch_size_val, gen_train,
gen_val, Epoch, cuda):
    """
    训练一个世代 epoch
    net: 网络模型
    epoch: 一个世代 epoch
    epoch_size_train: 训练迭代次数 iters
    epoch_size_val: 验证迭代次数 iters
    gen_train: 训练数据集
    gen_val: 验证数据集
    Epoch: 总的迭代次数 Epoch
    cuda: 是否使用 GPU
    """
    train_total_loss = 0
    train_total_f_score = 0
    val_total_loss = 0
    val_total_f_score = 0
    # 开启训练模式
    net.train()
    print('Start Training')
    start_time = time.time()
    with tqdm(total=epoch_size_train, desc=f'Epoch {epoch + 1}/{Epoch}', po
stfix=dict, mininterval=0.3) as pbar:
        for iteration, batch in enumerate(gen_train):

```

```

    if iteration >= epoch_size_train:
        break
    imgs, pngs, labels = batch
    with torch.no_grad():
        imgs = Variable(torch.from_numpy(imgs).type(torch.FloatTensor
r))
        pngs = Variable(torch.from_numpy(pngs).type(torch.FloatTensor
r)).long()
        labels = Variable(torch.from_numpy(labels).type(torch.FloatTensor
ensor))

        if cuda:
            imgs = imgs.cuda()
            pngs = pngs.cuda()
            labels = labels.cuda()
        # 梯度初始化置零
        optimizer.zero_grad()
        # 前向传播, 网络输出
        outputs = net(imgs)
        # 计算损失 一次iter 即一个batchsize 的loss
        loss = CE_Loss(outputs, pngs, num_classes = NUM_CLASSES)
        if dice_loss:
            main_dice = Dice_loss(outputs, labels)
            loss = loss + main_dice
        # 计算f_score
        with torch.no_grad():
            _f_score = f_score(outputs, labels)
        # loss 反向传播求梯度
        loss.backward()
        # 更新所有参数
        optimizer.step()
        train_total_loss += loss.item()
        train_total_f_score += _f_score.item()
        waste_time = time.time() - start_time
        pbar.set_postfix(**{'train_loss': train_total_loss / (iteration
+ 1),
                           'f_score'   : train_total_f_score / (iteration
+ 1),
                           's/step'   : waste_time,
                           'lr'       : get_lr(optimizer)})
        pbar.update(1)
        start_time = time.time()
    print('Finish Training')
    # 开启验证模式
    net.eval()
    print('Start Validation')

```



```

    with tqdm(total=epoch_size_val, desc=f'Epoch {epoch + 1}/{Epoch}', post
fix=dict, mininterval=0.3) as pbar:
        for iteration, batch in enumerate(gen_val):
            if iteration >= epoch_size_val:
                break
            imgs, pngs, labels = batch
            with torch.no_grad():
                imgs = Variable(torch.from_numpy(imgs).type(torch.FloatTensor
r))
                pngs = Variable(torch.from_numpy(pngs).type(torch.FloatTensor
r)).long()
                labels = Variable(torch.from_numpy(labels).type(torch.FloatT
ensor))

                if cuda:
                    imgs = imgs.cuda()
                    pngs = pngs.cuda()
                    labels = labels.cuda()
                outputs = net(imgs)
                val_loss = CE_Loss(outputs, pngs, num_classes = NUM_CLASSES)
                if dice_loss:
                    main_dice = Dice_loss(outputs, labels)
                    val_loss = val_loss + main_dice
                # 计算f_score
                _f_score = f_score(outputs, labels)
                val_total_loss += val_loss.item()
                val_total_f_score += _f_score.item()
            pbar.set_postfix(**{'val_loss' : val_total_loss / (iteration +
1),
                                'f_score' : val_total_f_score / (iteration +
1),
                                'lr' : get_lr(optimizer)})
            pbar.update(1)
        print('Finish Validation')
        print('Epoch:' + str(epoch+1) + '/' + str(Epoch))
        print('Train Loss: %.4f || Val Loss: %.4f ' % (train_total_loss/(epoch_s
ize_train+1), val_total_loss/(epoch_size_val+1)))
        print('Saving state, epoch:', str(epoch+1))
        torch.save(model.state_dict(), 'logs/Epoch%d-Train_Loss%.4f-Val_Loss%.4
f.pth'%((epoch+1), train_total_loss/(epoch_size_train+1), val_total_loss/(e
poch_size_val+1)))

if __name__ == "__main__":
    inputs_size = [480,480,3]
    log_dir = "logs/"

```

```

#-----#
# 分类个数+1
#-----#
NUM_CLASSES = 2
#-----#
# 建议选项:
# 种类少 (几类) 时, 设置为 True
# 种类多 (十几类) 时, 如果 batch_size 比较大 (10 以上), 那么设置为 True
# 种类多 (十几类) 时, 如果 batch_size 比较小 (10 以下), 那么设置为 False
#-----#

dice_loss = True
#-----#
#-----#
# Cuda 的使用
#-----#
Cuda = True
# 模型加载
#-----ResNet50+SENet+FCN-----
#backbone = SE_ResNet50()
#model = FCN1s(pretrained_net=backbone, n_class=NUM_CLASSES)
# -----unet-----
#model = UNet(num_classes=2)
# -----AttU_Net-----
model = AttU_Net(img_ch=3, output_ch=2)
#-----#
# 权值文件的下载请看 README, 权值和主干特征提取网络一定要对应
#-----#
model_path = "/home/wp/pytorch/FCN/logs/Epoch81-Train_Loss0.5137-Val_Loss0.5087.pth"
# 加快模型训练的效率
print('正在加载预训练模型权重...')
model_dict = model.state_dict()
pretrained_dict = torch.load(model_path)
pretrained_dict = {k: v for k, v in pretrained_dict.items() if np.shape(model_dict[k]) == np.shape(v)}
model_dict.update(pretrained_dict)
model.load_state_dict(model_dict)
print('预训练模型权重加载完成!')
# 使用多 gpu 训练
if Cuda:
    net = torch.nn.DataParallel(model)
    cudnn.benchmark = True
    net = net.cuda()
# 打开训练数据集的 txt

```

```

with open(r"VOCdevkit/VOC2012/ImageSets/Segmentation/train.txt","r") as
f:
    train_lines = f.readlines()
    # 打开验证集数据集的txt
with open(r"VOCdevkit/VOC2012/ImageSets/Segmentation/val.txt","r") as
f:
    val_lines = f.readlines()

if True:
    lr = 1.03e-5
    Interval_Epoch = 81
    Epoch = 100
    Batch_size = 8
    optimizer = optim.Adam(model.parameters(),lr)
    lr_scheduler = optim.lr_scheduler.StepLR(optimizer,step_size=2,gamm
a=0.9)
    # 加载数据集
    train_dataset = FCNDataset(train_lines, inputs_size, NUM_CLASSES, Tr
ue)
    val_dataset = FCNDataset(val_lines, inputs_size, NUM_CLASSES, False)

    gen_train = DataLoader(train_dataset, shuffle=True, batch_size=Batch
_size, num_workers=4, pin_memory=True,
                           drop_last=True, collate_fn=fcn_dataset_collat
e)
    gen_val = DataLoader(val_dataset, shuffle=True, batch_size=Batch_siz
e, num_workers=4,pin_memory=True,
                       drop_last=True, collate_fn=fcn_dataset_collat
e)

    epoch_size_train = max(1, len(train_lines)//Batch_size)
    epoch_size_val = max(1, len(val_lines)//Batch_size)
    # 打开骨干网络
    # for param in model.backbone.parameters():
    #     param.requires_grad = True
    for epoch in range(Interval_Epoch,Epoch):
        fit_one_epoch(model, epoch, epoch_size_train, epoch_size_val, ge
n_train, gen_val, Epoch, Cuda)
        lr_scheduler.step()

import torch
import torch.nn.functional as F

def f_score(inputs, target, beta=1, smooth = 1e-5, threshold = 0.5):
    n, c, h, w = inputs.size()

```

```

nt, ht, wt, ct = target.size()
if h != ht and w != wt:
    inputs = F.interpolate(inputs, size=(ht, wt), mode="bilinear", align
_corners=True)
    temp_inputs = torch.softmax(inputs.transpose(1, 2).transpose(2, 3).cont
iguous().view(n, -1, c), -1)
    temp_target = target.view(n, -1, ct)
    # 计算dice 系数
    temp_inputs = torch.gt(temp_inputs, threshold).float()
    tp = torch.sum(temp_target[..., :-1] * temp_inputs, axis=[0, 1])
    fp = torch.sum(temp_inputs, axis=[0, 1]) - tp
    fn = torch.sum(temp_target[..., :-1], axis=[0, 1]) - tp
    score = ((1 + beta ** 2) * tp + smooth) / ((1 + beta ** 2) * tp + beta **
2 * fn + fp + smooth)
    score = torch.mean(score)
    return score

```

## 代码 6：问题二—模型预测

```

# -*- coding: UTF-8 -*-

from fcn_senet import FCN1s as fcn
from fcn_senet import SE_ResNet50
from unet_senet import AttU_Net
from unet import UNet
from torch import nn
from PIL import Image
from torch.autograd import Variable
import models
import torch.nn.functional as F
import numpy as np
import colorsys
import torch
import copy
import os
os.environ["CUDA_VISIBLE_DEVICES"] = '1'

class FCN1s(object):
    #-----#
    # 注意修改 model_path、num_classes 和 backbone, 使其符合自己的模型
    #-----#
    _defaults = {
        "model_path" : '/home/wp/pytorch/FCN/logs50/Epoch100-Train_

```

```

Loss0.5064-Val_Loss0.4850.pth',
    "model_image_size" : (480, 480, 3),
    "num_classes"      : 2,
    "cuda"             : True,
    "blend"            : False, #True
}

def __init__(self, **kwargs):
    self.__dict__.update(self._defaults)
    self.generate()

def generate(self):
    # 获得所有的分类
    backbone = SE_ResNet50()
    self.net = fcn(pretrained_net=backbone, n_class=self.num_classes)
    #self.net = AttU_Net(img_ch=3, output_ch=2)
    #self.net = UNet(num_classes=2)
    self.net = self.net.eval()
    # 加载训练好的模型
    state_dict = torch.load(self.model_path)
    self.net.load_state_dict(state_dict, strict=False)
    if self.cuda:
        self.net = nn.DataParallel(self.net)
        self.net = self.net.cuda()
    print('{} model, anchors, and classes loaded.'.format(self.model_path))

    # 画框设置不同的颜色
    if self.num_classes <= 21: # (128, 0, 0)
        self.colors = [(0, 0, 0), (255, 255, 255), (0, 128, 0), (128, 128, 0), (0, 0, 128), (128, 0, 128), (0, 128, 128),
            (128, 128, 128), (64, 0, 0), (192, 0, 0), (64, 128, 0),
            (192, 128, 0), (64, 0, 128), (192, 0, 128),
            (64, 128, 128), (192, 128, 128), (0, 64, 0), (128, 64, 0), (0, 192, 0), (128, 192, 0), (0, 64, 128), (128, 64, 12)]
    else:
        # 画框设置不同的颜色
        hsv_tuples = [(x / len(self.class_names), 1., 1.) for x in range(len(self.class_names))]
        self.colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
        self.colors = list(map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)), self.colors))

def letterbox_image(self, image, size):
    iw, ih = image.size

```

```

w, h = size
scale = min(w/iw, h/ih)
nw = int(iw*scale)
nh = int(ih*scale)
image = image.resize((nw,nh), Image.BICUBIC)
new_image = Image.new('RGB', size, (128,128,128))
new_image.paste(image, ((w-nw)//2, (h-nh)//2))
return new_image,nw,nh

def detect_image(self, image):
    """
    检测图片
    """
    old_img = copy.deepcopy(image)
    orininal_h = np.array(image).shape[0]
    orininal_w = np.array(image).shape[1]
    image, nw, nh = self.letterbox_image(image, (self.model_image_size
[1], self.model_image_size[0]))
    images = [np.array(image)/255] # 增加batchsize 维度
    images = np.transpose(images,(0,3,1,2))
    with torch.no_grad():
        images = Variable(torch.from_numpy(images).type(torch.FloatTensor
r))
        if self.cuda:
            images =images.cuda()
            pr = self.net(images)[0]
            pr = F.softmax(pr.permute(1,2,0),dim = -1).cpu().numpy().argmax
(axis=-1)
            pr = pr[int((self.model_image_size[0]-nh)//2):int((self.model_im
age_size[0]-nh)//2+nh), int((self.model_image_size[1]-nw)//2):int((self.mod
el_image_size[1]-nw)//2+nw)]
            cv2.imwrite("./img_out/"+jpg.split('.')[0]+" .png",np.uint8(pr),[cv
2.IMWRITE_PNG_COMPRESSION, 0])
            seg_img = np.zeros((np.shape(pr)[0],np.shape(pr)[1],3))

            for c in range(self.num_classes):
                seg_img[:, :,0] += ((pr[:, : ] == c )*( self.colors[c][0] )).astyp
e('uint8')
                seg_img[:, :,1] += ((pr[:, : ] == c )*( self.colors[c][1] )).astyp
e('uint8')
                seg_img[:, :,2] += ((pr[:, : ] == c )*( self.colors[c][2] )).astyp
e('uint8')
            image = Image.fromarray(np.uint8(seg_img)).resize((orininal_w,orini
nal_h))
            if self.blend:

```

```

        image = Image.blend(old_img,image,0.5)
    return image

#-----#
#     对单张图片进行预测
#-----#

from fcn import FCN1s
from PIL import Image

fcn = FCN1s()

while True:
    img = input('Input image filename:')
    try:
        image = Image.open(img)
    except:
        print('Open Error! Try again!')
        continue
    else:
        r_image = fcn.detect_image(image)
        #r_image.show()
        r_image.save("SENet+ResNet50+FCN 2.png")

```

## 代码 7：问题二—模型融合

```

import numpy as np
import cv2
import argparse

RESULT_PREFIXX = ['./result1/', './result2/', './result3/']

# each mask has 2 classes: 0~2

def vote_per_image(image_id):
    result_list = []
    for j in range(len(RESULT_PREFIXX)):
        im = cv2.imread(RESULT_PREFIXX[j]+str(image_id)+'.png',0)
        im = cv2.resize(im,(600,500))
        result_list.append(im)
    print(result_list[0].shape)
    # each pixel
    height,width = result_list[0].shape
    vote_mask = np.zeros((height,width))
    for h in range(height):
        for w in range(width):

```

```

record = np.zeros((1,2))
for n in range(len(result_list)): # 3
    mask = result_list[n]
    pixel = mask[h,w]
    print('pix:',pixel)
    record[0,pixel]+=1

label = record.argmax()
#print(label)
vote_mask[h,w] = label

cv2.imwrite('vote_mask'+str(image_id)+'.png',vote_mask)

#####im = cv2.imread('result1/q1.png',0)
# print(im.shape)
#####print(im[1,:])
for q in range(1,3):
    vote_per_image(q)

```