《计算机视觉》

-图像分割



华为技术有限公司

目录

[1 实验介绍 2](#_Toc81361457)

[1.1 实验目的 2](#_Toc81361458)

[1.2 实验清单 2](#_Toc81361459)

[1.3 实验开发环境 2](#_Toc81361460)

[2 医学图像分割实验 3](#_Toc81361461)

[2.1 医学图像分割实验介绍 3](#_Toc81361462)

[2.2 实验总体设计 4](#_Toc81361463)

[2.3 实验详细设计与实现 4](#_Toc81361464)

[2.3.1 创建华为云Notebook 4](#_Toc81361465)

[2.3.2 下载数据集 4](#_Toc81361466)

[2.3.3 导入实验环境 4](#_Toc81361467)

[2.3.4 查看数据集 5](#_Toc81361468)

[2.3.5 使用大津阈值法进行图像分割 6](#_Toc81361469)

[2.3.6 基于神经网络的图像分割算法 8](#_Toc81361470)

[2.3.7 模型训练 19](#_Toc81361471)

[2.3.8 模型验证和测试 22](#_Toc81361472)

[2.4 实验总结 25](#_Toc81361473)

[2.5 思考题-汇总 25](#_Toc81361474)

[3 基于DeepLabv3的语义分割实验 27](#_Toc81361475)

[3.1 实验介绍 27](#_Toc81361476)

[3.2 实验环境要求 27](#_Toc81361477)

[3.3 背景知识 27](#_Toc81361478)

[3.3.1 网络介绍 27](#_Toc81361479)

[3.4 实验步骤 29](#_Toc81361480)

[3.5 实验总结 51](#_Toc81361481)

[4 附录：ModelArts开发环境搭建 52](#_Toc81361482)

# 实验介绍

计算机视觉属于人工智能核心应用领域，而图像分割是计算机视觉中的基本任务之一，在工业生产中有着广泛的应用。本章实验主要围绕基于深度学习的图像分割任务中典型的网络进行开发，如U-Net和DeepLabv3等，本章实验难度分为初级和高级。

初级：基于U-Net的医学图像分割实验。

高级：基于DeepLabv3的语义分割实验。

## 实验目的

本章实验的主要目的是掌握图像分割任务难点，了解如何使用深度学习解决相关问题。掌握不同图像分割神经网络架构的设计原理与核心思想，熟悉使用MindSpore深度学习框架实现深度学习实验的一般流程。

## 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 软件环境 | 开发环境 |
| 基于U-Net的医学图像分割实验 | 使用MindSpore，实现基于U-Net模型的医学图像分割实验。 | 高级 | Python3.7.5  MindSpore 1.5 | ModelArts |
| 基于DeepLabv3的语义分割实验 | 使用MindSpore实现基于微调DeepLabv3模型的语义分割实验。 | 高级 | Python3.7.5  MindSpore 1.5 | ModelArts |

## 实验开发环境

MindSpore-1.5

若选择在华为云ModelArts上快速搭建开发环境，可参考文末附录：ModelArts开发环境搭建。

# 医学图像分割实验

## 医学图像分割实验介绍

图像分割（Segmentation）是根据需要解决的问题，将图像细分为目标内容所构成的不同子区域。本次实验使用的数据集是ISBI会议在2012年进行的挑战竞赛提供的数据。该项竞赛希望训练出模型，能自动在果蝇一龄幼虫腹神经索(VNC)连续切片透射电镜(ssTEM)数据集的切片图像中，分割出神经组织。可以通过官网了解竞赛详情和获取数据集：<http://brainiac2.mit.edu/isbi_challenge/home>

数据集使用的图像是真实图像中的代表数据，存在一定的噪声和较小的图像对齐误差。这些问题对于人类神经解剖学专家进行手工的图像标注是没有任何困难的。但是对于传统的图像处理算法，并不能达到很好的自动分割效果。

本实验使用了深度学习的U-net网络模型进行图像分割。U-net是2015年菲兹保大学的Olaf Ronneberger等人在论文《U-Net: Convolutional Networks for Biomedical Image Segmentation》中提出的深度网络结构，是适用于生物图像分割的深度学习模型。该网络可以用非常少的图像进行端对端训练，并且速度非常快。U-Net是一个全卷积网络，输入和输出都是图像，没有全连接层。U-Net的网络结构和论文可以参考其官网：<https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>

【实验环境要求】:

1、python3.7.5

2、MindSpore1.5

3、ModelArts平台

## 实验总体设计

本实验将使用U-Net完成医学图像分割任务，通过Python语言与其各种强大的资源库如numpy, matplotlib, opencv, 等来实现复杂图像的分割效果。作为对照组的传统图像处理算法使用了opencv库。

通过本实验学员将了解如何使用MindSpore对少量的数据集进行数据增强，并学习如何定义和训练卷积神经网络的基本操作，来实现能分割医学图像的U-Net模型训练。

## 实验详细设计与实现

本节将详细介绍实验的设计与实现，本实验的实验步骤为：

1. 创建华为云Notebook
2. 下载数据集
3. 导入实验环境
4. 查看数据集
5. 使用传统算法进行图像分割，以大津阈值法为例
6. 定义Unet网络结构
7. 定义损失函数
8. 数据预处理
9. 使用数据增强后的数据训练神经网络
10. 使用训练好的模型进行预测
11. 显示结果

### 创建华为云Notebook

进入ModelArts开发环境

参考文末附录，创建ModelArts上的开发环境Notebook并进入。

### 下载数据集

通过moxing接口，从华为对象存储服务OBS桶中下载数据集到Notebook中的虚拟硬盘。

import moxing as mox

mox.file.copy\_parallel(src\_url="obs://ascend-professional-construction-dataset/ComputerVision/Unet",dst\_url="./")

### 导入实验环境

#导入实验所需要的库

import os

import argparse

import ast

import numpy as np

import cv2

import mindspore

import mindspore.nn as nn

import mindspore.ops.operations as F

from mindspore import Model, context

from mindspore.nn.loss.loss import \_Loss

from mindspore.communication.management import init, get\_group\_size

from mindspore.train.callback import CheckpointConfig, ModelCheckpoint

from mindspore.context import ParallelMode

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

from mindspore.common.initializer import TruncatedNormal

from mindspore.nn import CentralCrop

from PIL import Image, ImageSequence

import mindspore.dataset as ds

import mindspore.dataset.vision.c\_transforms as c\_vision

from mindspore.dataset.vision.utils import Inter

from mindspore.communication.management import get\_rank, get\_group\_size

from collections import deque

import time

from mindspore.train.callback import Callback

from mindspore.common.tensor import Tensor

from scipy.special import softmax

from matplotlib import pyplot as plt

device\_id = 2

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend", save\_graphs=False)

mindspore.set\_seed(1)

### 查看数据集

数据集说明

官网数据集（<http://brainiac2.mit.edu/isbi_challenge/home>）下载需要注册账号。

训练和测试数据集为两组30节果蝇一龄幼虫腹神经索（VNC）的连续透射电子显微镜（ssTEM）数据集。微立方体的尺寸约为2 x 2 x 1.5微米，分辨率为4x4x50纳米/像素。数据集大小为22.5 MB，共三个文件：train-volume.tif，train-labels.tif，test-volume.tif。第一个文件为训练集图像，该TIF文件共有30个通道，每个通道为一张灰度图像。第二个文件为训练集标签，该TIF文件共有30个通道，每个通道为一张灰度图像（像素值仅为0或255）。第三个文件为测试集图像，该TIF文件同样有30个通道，每个通道为一张灰度图像。

展示数据集图像和标签

#打印图像和标签的形状，并展示第一张图像和标签

image= np.array([np.array(p) for p in ImageSequence.Iterator(Image.open("./data/train-volume.tif"))])

label= np.array([np.array(p) for p in ImageSequence.Iterator(Image.open("./data/train-labels.tif"))])

print(image.shape)

print(label.shape)

#设置图像大小，单位为“英寸”

plt.figure(figsize=(10,10))

plt.subplot(2,2,1)

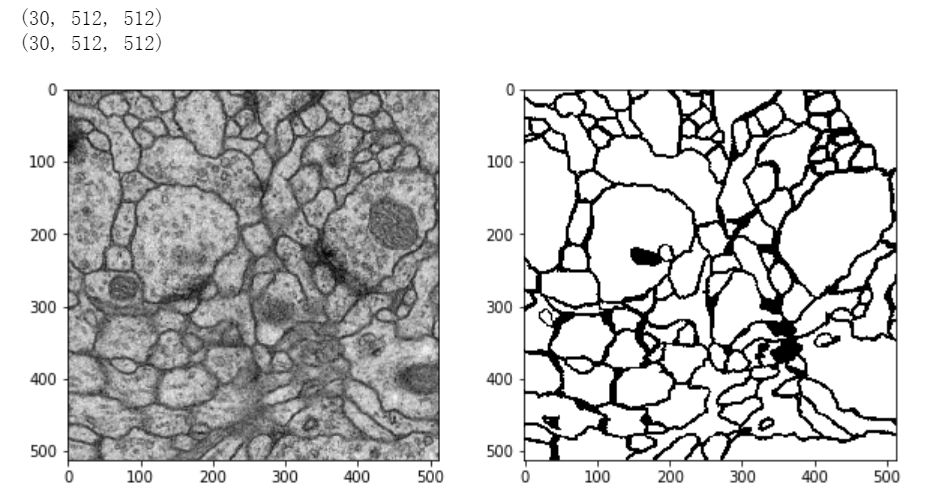
plt.imshow(image[0],cmap='gray')

plt.subplot(2,2,2)

plt.imshow(label[0],cmap='gray')

plt.show()

输出：



### 使用大津阈值法进行图像分割

#### OTSU算法介绍：

大津阈值法（OTSU）,又称作最大类间方差法，是一种图像二值化分割阈值的算法，由日本学者大津于1979年提出。大津阈值法是具有统计意义上的最佳分割阈值。其核心思想就是使类间方差最大，按照大津阈值法求得的阈值进行图像二值分割以区分前后背景，前景与背景图像的类间方差最大。该算法要求被分割的物体颜色纹理比较紧凑，类内方差小，对于一些文本图像的处理（比如车牌、指纹）效果很好。

大津阈值法是基于统计直方图的图像分割算法，首先我们绘制观察图像直方图。

试题1：请用OpenCV和matplotlib绘制一张训练集中的直方图。（初级）

解答1：

#显示原图的直方图

plt.hist(image[0].ravel(), 256)

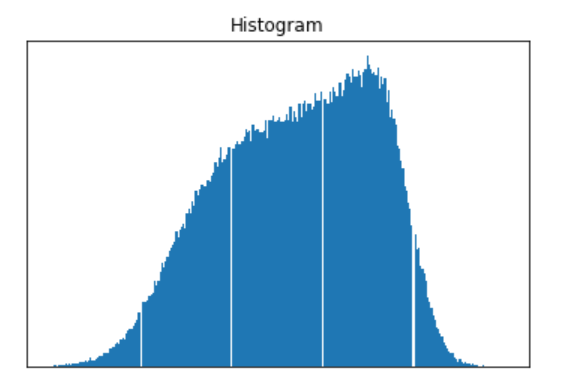
plt.title("Histogram")

plt.xticks([])

plt.yticks([])

plt.show()

输出：



直方图展示

我们发现该数据集的图像的直方图很连续，并没有出现明显的波谷，使用传统基于统计的阈值分割算法可能并不能取得好的效果。

试题2：请用OpenCV中的python接口实现基于大津阈值法的图像二值化分割。（初级）

解答2：

# 二值化处理，thesh=0代表其从0开始扫描

ret1, th1 = cv2.threshold(src=image[0], thresh=0,

maxval=255, type=cv2.THRESH\_OTSU)

#显示原图

plt.figure(figsize=(10,10))

plt.subplot(121)

plt.imshow(image[0],cmap='gray')

plt.title("source image")

plt.xticks([])

plt.yticks([])

#显示经过大津阈值法后的二值化图像

plt.subplot(122)

plt.imshow(th1, "gray")

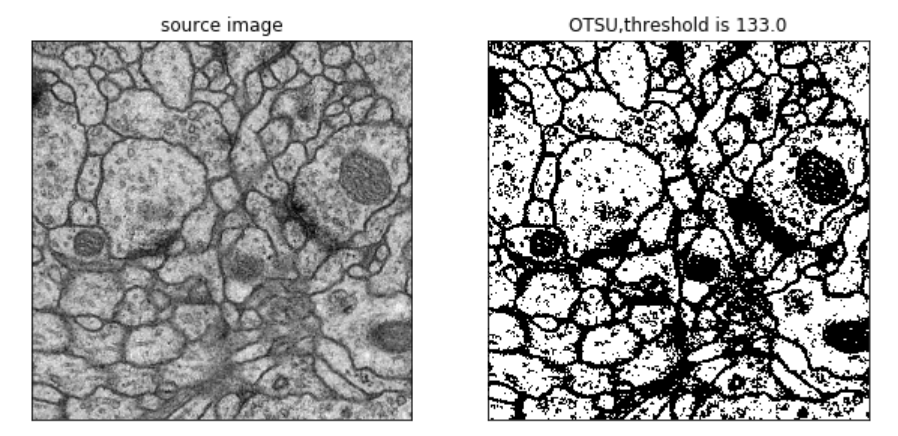
plt.title("OTSU,threshold is " + str(ret1))

plt.xticks([])

plt.yticks([])

plt.show()

输出：



大津阈值法后的二值化图像

通过分割结果我们观察到，分割结果并不理想。下面我们尝试用深度学习的方法解决以上的图像分割问题。

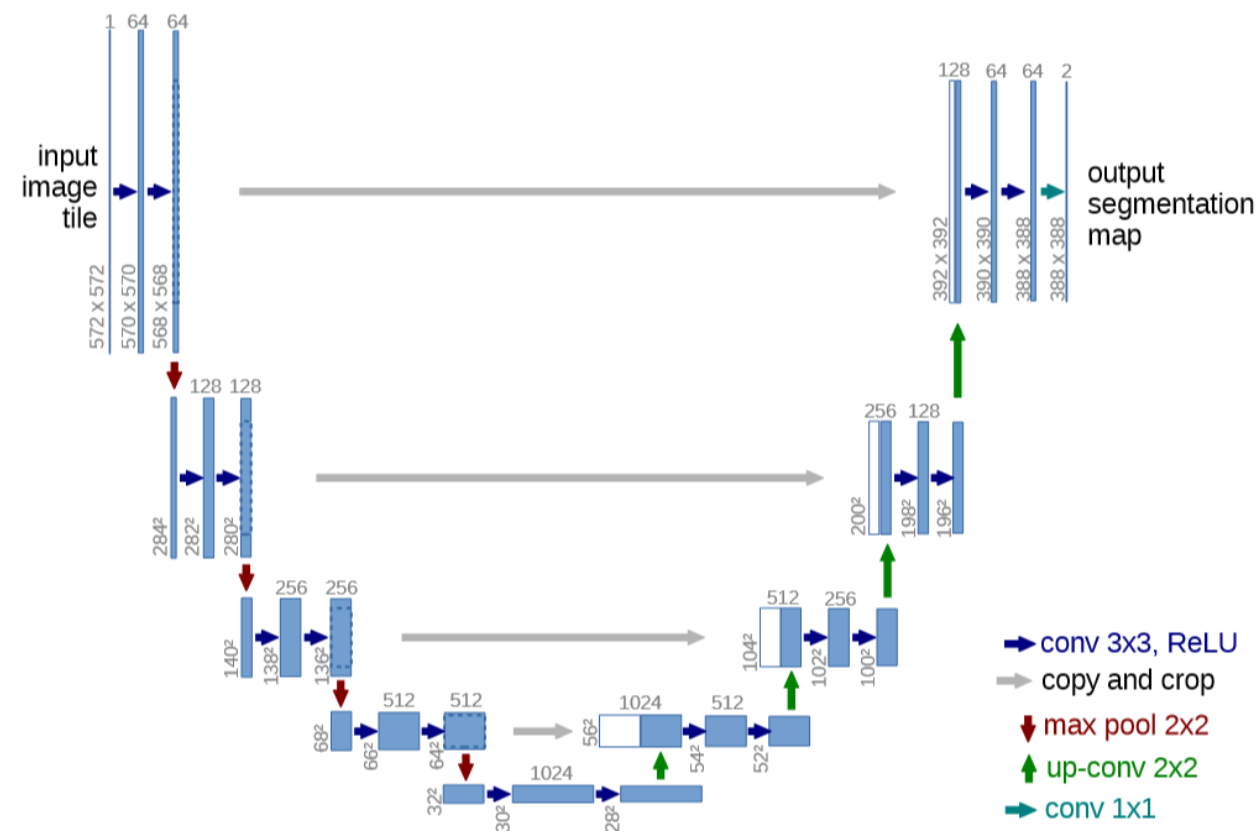
### 基于神经网络的图像分割算法

使用深度学习进行图像分割，采用了分类的思路，对每个像素点进行分类，判断像素点是属于目标前景还是背景。 传统卷积神经网络做分类的步骤是，首先单个图像进来之后经过多层卷积得到降维之后的特征图，这个特征图经过全连接层变成一个分类器，最后输出一个类别的向量，这就是分类的结果。 对于基于神经网络的图像分割问题来说，图像中的每一个像素都会输出一个分类结果，传统神经网络中分类的向量，就变成了一个分类的特征图，通道数等于类别数量。

#### 定义Unet网络结构

Unet是一种改进的全卷积网络（FCN）用于图像分割任务。由Olaf Ronneberger等人在论文《U-Net: Convolutional Networks for Biomedical Image Segmentation》中提出。U-Net的网络结构和论文可以参考其官网：<https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>

U-Net得名于它的网络结构图，类似一个英文字母“U”。左半边是一个从上到下，一步一步从原始图像抽取特征（即原始图像本质信息）的过程；右半边是一个从下到上，一步一步从图像特征还原目标信息的过程。U-Net网络主要分为encoder和decoder两个部分。网络结构如下所示：



Unet网络结构

##### 编码

第一部分网络与普通卷积网络相同，通过叠加2个3x3卷积和最大池化的模块，获取多尺度特征图以抓住图像中的上下文信息（也即像素间的关系）。网络中特征图的尺寸共压缩了4次。

卷积层：原始U-Net网络中卷积层统一为3x3的卷积核，padding为0 ，striding为1。没有padding所以每次卷积之后feature map的尺寸变小。因此在copy and crop时要注意feature map的尺寸。且我们会发现输入的图像尺寸为572x572，输出尺寸为388x388，输入的尺寸是要大于输出的尺寸的。

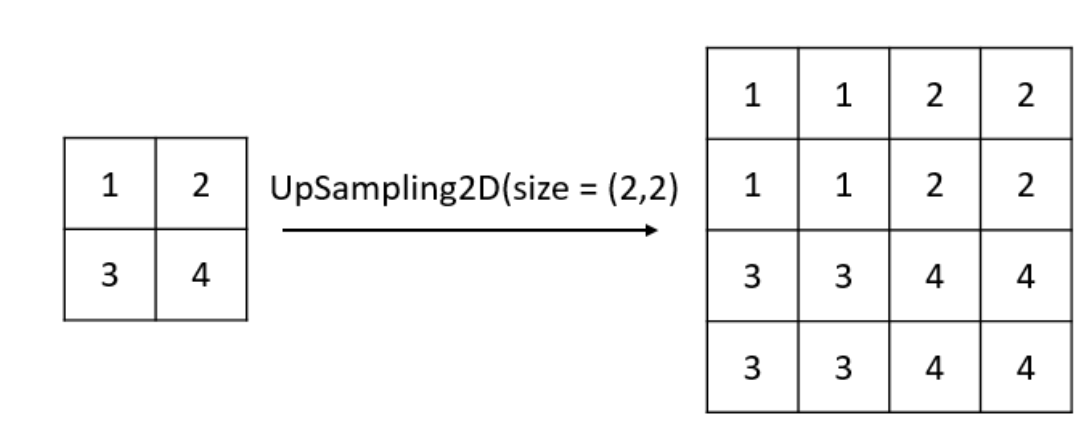
池化层：两次卷积之后是一个窗口尺寸为2，步长为2的max pooling，输出大小变为原来的二分之一，池化的主要作用为减小特征图尺寸。

##### 解码

第二部分与前面基本对称，共4个上采样模块，每个模块有2个3x3卷积和转置卷积，3x3的卷积核同样采用padding为0 ，striding为1，转置卷积考虑到棋盘效应（https://distill.pub/2016/deconv-checkerboard/），采用的是窗口大小为2，步长为2的转置卷积。每上采样一次，通过copy and crop，即skip-connection操作与encoder中对应的特征图进行拼接。网络最后输出时利用1\*1卷积调整特征图输出数量。该过程除了卷积比较关键的步骤就是转置卷积与skip-connection。

上采样的目的是恢复图像分辨率，上采样采用的方式有简单像素重复、转置卷积和差值（最邻近差值和双线性差值法等）。

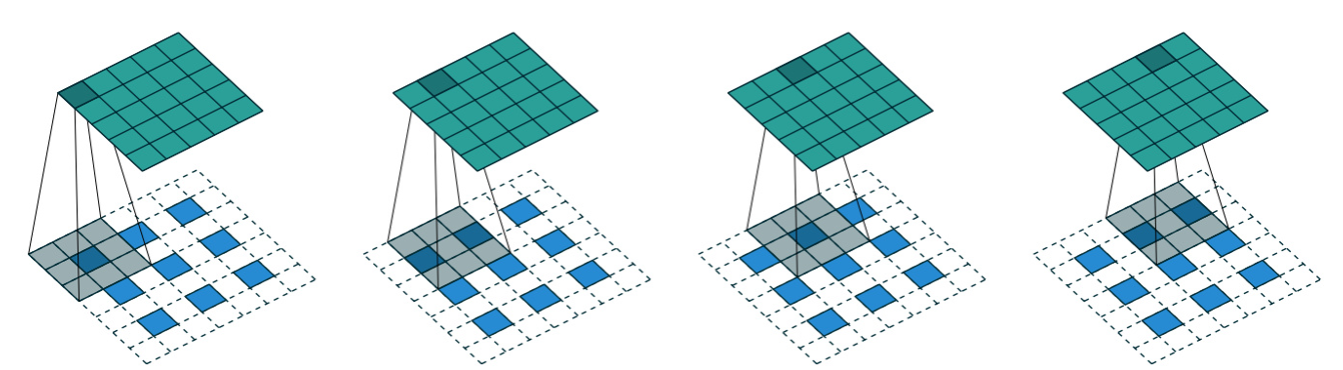
简单的像素重复(UpSampling2D)：



UpSampling2D展示

同样地，UpSampling2D的方式是没有参数的。

转置卷积：转置卷积操作先按照一定的比例通过补0来扩大输入图像的尺寸，接着旋转卷积核，再进行正向卷积。



转置卷积展示

mindspore.nn.Conv2dTranspose参考：<https://www.mindspore.cn/docs/api/zh-CN/r1.5/api_python/nn/mindspore.nn.Conv2dTranspose.html?highlight=conv2dtranspose#mindspore.nn.Conv2dTranspose>

U-Net中skip-connection通过特征图拼接的方式融合不同层次特征图。其目的是避免decoder只利用encoder中较深卷积层的high-level特征进行图像分辨率恢复和损失计算，而是结合了低级feature map中的特征，从而可以使得最终所得到的feature map中既包含了high-level 的feature，也包含很多的low-level的feature，实现了不同scale下feature的融合，提高模型的结果精确度。实现方式为特征图拼接。

concatenate参考：<https://www.mindspore.cn/docs/api/zh-CN/r1.5/api_python/ops/mindspore.ops.Concat.html?highlight=concat#mindspore.ops.Concat>

##### 超参数配置

#设置模型的超参数

cfg\_unet = {

'name': 'Unet',

'lr': 0.0001,

'epochs': 400,

'distribute\_epochs': 1600,

'batchsize': 16,

'cross\_valid\_ind': 1,

'num\_classes': 2,

'num\_channels': 1,

'keep\_checkpoint\_max': 10,

'weight\_decay': 0.0005,

'loss\_scale': 1024.0,

'FixedLossScaleManager': 1024.0,

'resume': False,

'resume\_ckpt': './',

}

##### 各个模块搭建

分别搭建了两层卷积，下采样，上采样，输出层模块，用于搭建Unet网络。

class DoubleConv(nn.Cell):

#定义两个卷积层，由于所有的模块都要用到，因此为了简便，定义了这个类

def \_\_init\_\_(self, in\_channels, out\_channels, mid\_channels=None):

super().\_\_init\_\_()

#截断高斯分布初始化

init\_value\_0 = TruncatedNormal(0.06)

init\_value\_1 = TruncatedNormal(0.06)

if not mid\_channels:

mid\_channels = out\_channels

#定义两个卷积层，根据原论文，激活函数用ReLU,卷积核大小为3，不用padding，因此每经过一个卷积层，特征图大小会减小2像素。

self.double\_conv = nn.SequentialCell(

[nn.Conv2d(in\_channels, mid\_channels, kernel\_size=3, has\_bias=True,

weight\_init=init\_value\_0, pad\_mode="valid"),

nn.ReLU(),

nn.Conv2d(mid\_channels, out\_channels, kernel\_size=3, has\_bias=True,

weight\_init=init\_value\_1, pad\_mode="valid"),

nn.ReLU()]

)

def construct(self, x):

return self.double\_conv(x)

class Down(nn.Cell):

"""根据原论文，下采样用一个最大池化接两个卷积层"""

def \_\_init\_\_(self, in\_channels, out\_channels):

super().\_\_init\_\_()

self.maxpool\_conv = nn.SequentialCell(

[nn.MaxPool2d(kernel\_size=2, stride=2),

DoubleConv(in\_channels, out\_channels)]

)

def construct(self, x):

return self.maxpool\_conv(x)

class Up1(nn.Cell):

"""根据原论文，第一个上采样模块采用一个转置卷积接两个卷积"""

def \_\_init\_\_(self, in\_channels, out\_channels, bilinear=True):

super().\_\_init\_\_()

#U-Net网络需要拼接编码器和解码器的特征图

self.concat = F.Concat(axis=1)

#根据原论文，编码器中的特征图经过中心裁剪后，进行拼接。编码器的特征图大小为64，解码器为56。

self.factor = 56.0 / 64.0

#中心裁剪

self.center\_crop = CentralCrop(central\_fraction=self.factor)

self.print\_fn = F.Print()

self.conv = DoubleConv(in\_channels, out\_channels, in\_channels // 2)

#转置卷积，常用的上采样方式。

self.up = nn.Conv2dTranspose(in\_channels, in\_channels // 2, kernel\_size=2, stride=2)

self.relu = nn.ReLU()

def construct(self, x1, x2):

x1 = self.up(x1)

x1 = self.relu(x1)

x2 = self.center\_crop(x2)

x = self.concat((x1, x2))

return self.conv(x)

class Up2(nn.Cell):

"""Upscaling then double conv"""

def \_\_init\_\_(self, in\_channels, out\_channels, bilinear=True):

super().\_\_init\_\_()

#U-Net网络需要拼接编码器和解码器的特征图

self.concat = F.Concat(axis=1)

#根据原论文，编码器中的特征图经过中心裁剪后，进行拼接。编码器的特征图大小为136，解码器为104。

self.factor = 104.0 / 136.0

#中心裁剪

self.center\_crop = CentralCrop(central\_fraction=self.factor)

self.conv = DoubleConv(in\_channels, out\_channels, in\_channels // 2)

#转置卷积，常用的上采样方式。

self.up = nn.Conv2dTranspose(in\_channels, in\_channels // 2, kernel\_size=2, stride=2)

self.relu = nn.ReLU()

def construct(self, x1, x2):

x1 = self.up(x1)

x1 = self.relu(x1)

x2 = self.center\_crop(x2)

x = self.concat((x1, x2))

return self.conv(x)

class Up3(nn.Cell):

"""Upscaling then double conv"""

def \_\_init\_\_(self, in\_channels, out\_channels, bilinear=True):

super().\_\_init\_\_()

#U-Net网络需要拼接编码器和解码器的特征图

self.concat = F.Concat(axis=1)

#根据原论文，编码器中的特征图经过中心裁剪后，进行拼接。编码器的特征图大小为280，解码器为200。

self.factor = 200 / 280

self.center\_crop = CentralCrop(central\_fraction=self.factor)

self.print\_fn = F.Print()

self.conv = DoubleConv(in\_channels, out\_channels, in\_channels // 2)

self.up = nn.Conv2dTranspose(in\_channels, in\_channels // 2, kernel\_size=2, stride=2)

self.relu = nn.ReLU()

def construct(self, x1, x2):

x1 = self.up(x1)

x1 = self.relu(x1)

x2 = self.center\_crop(x2)

x = self.concat((x1, x2))

return self.conv(x)

class Up4(nn.Cell):

"""Upscaling then double conv"""

def \_\_init\_\_(self, in\_channels, out\_channels, bilinear=True):

super().\_\_init\_\_()

self.concat = F.Concat(axis=1)

#根据原论文，编码器中的特征图经过中心裁剪后，进行拼接。编码器的特征图大小为568，解码器为392

self.factor = 392 / 568

self.center\_crop = CentralCrop(central\_fraction=self.factor)

self.conv = DoubleConv(in\_channels, out\_channels, in\_channels // 2)

self.up = nn.Conv2dTranspose(in\_channels, in\_channels // 2, kernel\_size=2, stride=2)

self.relu = nn.ReLU()

def construct(self, x1, x2):

x1 = self.up(x1)

x1 = self.relu(x1)

x2 = self.center\_crop(x2)

x = self.concat((x1, x2))

return self.conv(x)

class OutConv(nn.Cell):

#最后的输出层，通道数为2，意味着二分类。

def \_\_init\_\_(self, in\_channels, out\_channels):

super(OutConv, self).\_\_init\_\_()

init\_value = TruncatedNormal(0.06)

self.conv = nn.Conv2d(in\_channels, out\_channels, kernel\_size=1, has\_bias=True, weight\_init=init\_value)

def construct(self, x):

x = self.conv(x)

return x

试题3：使用MindSpore搭建Unet类，用于构建网络。要求：网络结构和原论文保持一致。

解答3：

#根据原论文搭建U-Net网络

class UNet(nn.Cell):

def \_\_init\_\_(self, n\_channels, n\_classes):

super(UNet, self).\_\_init\_\_()

self.n\_channels = n\_channels

self.n\_classes = n\_classes

self.inc = DoubleConv(n\_channels, 64)

self.down1 = Down(64, 128)

self.down2 = Down(128, 256)

self.down3 = Down(256, 512)

self.down4 = Down(512, 1024)

self.up1 = Up1(1024, 512)

self.up2 = Up2(512, 256)

self.up3 = Up3(256, 128)

self.up4 = Up4(128, 64)

self.outc = OutConv(64, n\_classes)

def construct(self, x):

x1 = self.inc(x)

x2 = self.down1(x1)

x3 = self.down2(x2)

x4 = self.down3(x3)

x5 = self.down4(x4)

x = self.up1(x5, x4)

x = self.up2(x, x3)

x = self.up3(x, x2)

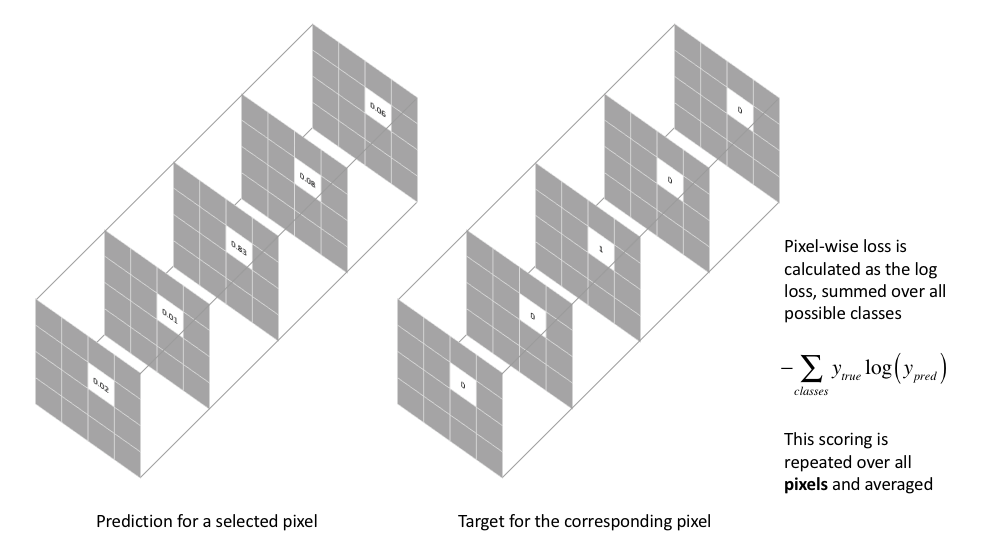
x = self.up4(x, x1)

logits = self.outc(x)

return logits

#### 损失函数

图像分割实现的是像素级别的分类。对应本实验来说，每个像素有两种可能分类，即前景和背景。损失函数会对所有位置的多分类损失求平均，示意图如下：



试题4：请重新构建一个类，将MindSpore的nn.SoftmaxCrossEntropyWithLogits损失函数用于Unet，计算输出特征图各个位置平均的损失值。

解答4：

class CrossEntropyWithLogits(\_Loss):

#重写损失函数。

def \_\_init\_\_(self):

super(CrossEntropyWithLogits, self).\_\_init\_\_()

self.transpose\_fn = F.Transpose()

self.reshape\_fn = F.Reshape()

self.softmax\_cross\_entropy\_loss = nn.SoftmaxCrossEntropyWithLogits()

self.cast = F.Cast()

def construct(self, logits, label):

# NCHW->NHWC

logits = self.transpose\_fn(logits, (0, 2, 3, 1))

logits = self.cast(logits, mindspore.float32)

label = self.transpose\_fn(label, (0, 2, 3, 1))

#损失函数计算所有像素点的交叉熵损失，取平均后就得到了总损失。

loss = self.reduce\_mean(self.softmax\_cross\_entropy\_loss(self.reshape\_fn(logits, (-1, 2)),

self.reshape\_fn(label, (-1, 2))))

return self.get\_loss(loss)

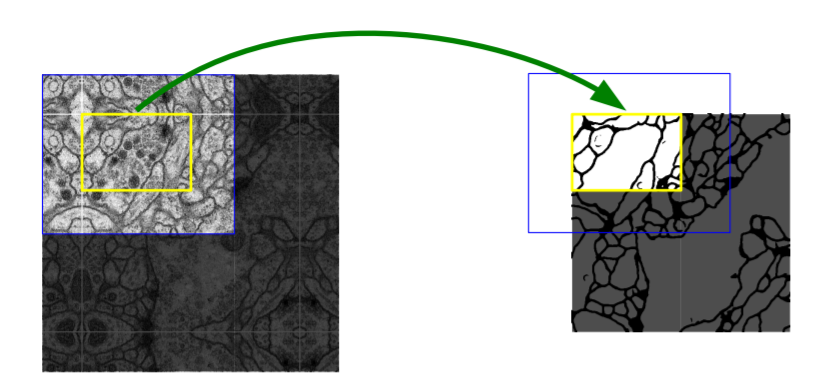
* + - 1. 数据预处理

要使用深度学习算法来处理此问题，首先需要解决数据的问题。深度学习的模型需要大量的数据驱动。在深度学习项目中，数据的收集和处理会花费相当多的时间。但在很多实际的项目中特别是医学图像，数据的获取和标注成本是很高的，我们难以找到充足的数据来完成任务。为了防止深度学习模型出现严重的过拟合现象，当数据量不足时，我们通常会做数据增强，扩大数据集样本量，提升数据样本空间的复杂度。

在原论文中，作者通过移动窗口的方式将一张图像切分成多个子图以及弹性形变（http://cognitivemedium.com/assets/rmnist/Simard.pdf）达到数据增强的目的。在该实验中，我们没有使用弹性形变，而是使用普通的仿射变换（即翻转，旋转，裁剪）和改变亮度来达到数据增强的目的。

此外，由于Unet网络的输入是大于输出的，因此，在输入图像前，我们还需要对原图进行放大。如下图，实际预测的区域为黄色的区域（输出层对应的区域），实际输入的区域为蓝色的区域。左图中间白色线框为训练集中的原图，外面一圈是通过对边缘部分进行翻转，旋转变换得到的。

其次，我们也需要对图像和标签进行归一化，来帮助模型训练。



归一化效果展示

def \_load\_multipage\_tiff(path):

"""Load tiff images containing many images in the channel dimension"""

return np.array([np.array(p) for p in ImageSequence.Iterator(Image.open(path))])

def \_get\_val\_train\_indices(length, fold, ratio=0.8):

#将训练数据分为训练集和验证集

assert 0 < ratio <= 1, "Train/total data ratio must be in range (0.0, 1.0]"

np.random.seed(0)

indices = np.arange(0, length, 1, dtype=np.int)

#打乱数据，利于模型训练

np.random.shuffle(indices)

if fold is not None:

#使用deque方式切分数据

indices = deque(indices)

indices.rotate(fold \* round((1.0 - ratio) \* length))

indices = np.array(indices)

train\_indices = indices[:round(ratio \* len(indices))]

val\_indices = indices[round(ratio \* len(indices)):]

else:

train\_indices = indices

val\_indices = []

return train\_indices, val\_indices

def data\_post\_process(img, mask):

img = np.expand\_dims(img, axis=0)

mask = (mask > 0.5).astype(np.int)

mask = (np.arange(mask.max() + 1) == mask[..., None]).astype(int)

mask = mask.transpose(2, 0, 1).astype(np.float32)

return img, mask

def train\_data\_augmentation(img, mask):

#原数据只有30张训练数据，因此需要数据增强

#生成随机数，如果随机数大于0.5就旋转图像。

h\_flip = np.random.random()

if h\_flip > 0.5:

img = np.flipud(img)

mask = np.flipud(mask)

v\_flip = np.random.random()

#生成随机数，如果随机数大于0.5就翻转图像。

if v\_flip > 0.5:

img = np.fliplr(img)

mask = np.fliplr(mask)

#生成随机数用于裁剪图像

left = int(np.random.uniform()\*0.3\*572)

right = int((1-np.random.uniform()\*0.3)\*572)

top = int(np.random.uniform()\*0.3\*572)

bottom = int((1-np.random.uniform()\*0.3)\*572)

#裁剪

img = img[top:bottom, left:right]

mask = mask[top:bottom, left:right]

#改变亮度

brightness = np.random.uniform(-0.2, 0.2)

img = np.float32(img+brightness\*np.ones(img.shape))

img = np.clip(img, -1.0, 1.0)

return img, mask

def create\_dataset(data\_dir, repeat=400, train\_batch\_size=16, augment=False, cross\_val\_ind=1, run\_distribute=False):

#创建数据集

images = \_load\_multipage\_tiff(os.path.join(data\_dir, 'train-volume.tif'))

masks = \_load\_multipage\_tiff(os.path.join(data\_dir, 'train-labels.tif'))

train\_indices, val\_indices = \_get\_val\_train\_indices(len(images), cross\_val\_ind)

train\_images = images[train\_indices]

train\_masks = masks[train\_indices]

train\_images = np.repeat(train\_images, repeat, axis=0)

train\_masks = np.repeat(train\_masks, repeat, axis=0)

val\_images = images[val\_indices]

val\_masks = masks[val\_indices]

train\_image\_data = {"image": train\_images}

train\_mask\_data = {"mask": train\_masks}

valid\_image\_data = {"image": val\_images}

valid\_mask\_data = {"mask": val\_masks}

ds\_train\_images = ds.NumpySlicesDataset(data=train\_image\_data, sampler=None, shuffle=False)

ds\_train\_masks = ds.NumpySlicesDataset(data=train\_mask\_data, sampler=None, shuffle=False)

if run\_distribute:

rank\_id = get\_rank()

rank\_size = get\_group\_size()

ds\_train\_images = ds.NumpySlicesDataset(data=train\_image\_data,

sampler=None,

shuffle=False,

num\_shards=rank\_size,

shard\_id=rank\_id)

ds\_train\_masks = ds.NumpySlicesDataset(data=train\_mask\_data,

sampler=None,

shuffle=False,

num\_shards=rank\_size,

shard\_id=rank\_id)

ds\_valid\_images = ds.NumpySlicesDataset(data=valid\_image\_data, sampler=None, shuffle=False)

ds\_valid\_masks = ds.NumpySlicesDataset(data=valid\_mask\_data, sampler=None, shuffle=False)

c\_resize\_op = c\_vision.Resize(size=(388, 388), interpolation=Inter.BILINEAR)

c\_pad = c\_vision.Pad(padding=92)

c\_rescale\_image = c\_vision.Rescale(1.0/127.5, -1)

c\_rescale\_mask = c\_vision.Rescale(1.0/255.0, 0)

c\_trans\_normalize\_img = [c\_rescale\_image, c\_resize\_op, c\_pad]

c\_trans\_normalize\_mask = [c\_rescale\_mask, c\_resize\_op, c\_pad]

c\_center\_crop = c\_vision.CenterCrop(size=388)

train\_image\_ds = ds\_train\_images.map(input\_columns="image", operations=c\_trans\_normalize\_img)

train\_mask\_ds = ds\_train\_masks.map(input\_columns="mask", operations=c\_trans\_normalize\_mask)

train\_ds = ds.zip((train\_image\_ds, train\_mask\_ds))

train\_ds = train\_ds.project(columns=["image", "mask"])

if augment:

augment\_process = train\_data\_augmentation

c\_resize\_op = c\_vision.Resize(size=(572, 572), interpolation=Inter.BILINEAR)

train\_ds = train\_ds.map(input\_columns=["image", "mask"], operations=augment\_process)

train\_ds = train\_ds.map(input\_columns="image", operations=c\_resize\_op)

train\_ds = train\_ds.map(input\_columns="mask", operations=c\_resize\_op)

train\_ds = train\_ds.map(input\_columns="mask", operations=c\_center\_crop)

post\_process = data\_post\_process

train\_ds = train\_ds.map(input\_columns=["image", "mask"], operations=post\_process)

train\_ds = train\_ds.shuffle(repeat\*24)

train\_ds = train\_ds.batch(batch\_size=train\_batch\_size, drop\_remainder=True)

valid\_image\_ds = ds\_valid\_images.map(input\_columns="image", operations=c\_trans\_normalize\_img)

valid\_mask\_ds = ds\_valid\_masks.map(input\_columns="mask", operations=c\_trans\_normalize\_mask)

valid\_ds = ds.zip((valid\_image\_ds, valid\_mask\_ds))

valid\_ds = valid\_ds.project(columns=["image", "mask"])

valid\_ds = valid\_ds.map(input\_columns="mask", operations=c\_center\_crop)

post\_process = data\_post\_process

valid\_ds = valid\_ds.map(input\_columns=["image", "mask"], operations=post\_process)

valid\_ds = valid\_ds.batch(batch\_size=1, drop\_remainder=True)

return train\_ds, valid\_ds

### 模型训练

#### 定义类用于打印损失和速度

class StepLossTimeMonitor(Callback):

#创建callback用于监控训练。

def \_\_init\_\_(self, batch\_size, per\_print\_times=1):

super(StepLossTimeMonitor, self).\_\_init\_\_()

if not isinstance(per\_print\_times, int) or per\_print\_times < 0:

raise ValueError("print\_step must be int and >= 0.")

self.\_per\_print\_times = per\_print\_times

self.batch\_size = batch\_size

def step\_begin(self, run\_context):

self.step\_time = time.time()

def step\_end(self, run\_context):

step\_seconds = time.time() - self.step\_time

step\_fps = self.batch\_size\*1.0/step\_seconds

cb\_params = run\_context.original\_args()

loss = cb\_params.net\_outputs

if isinstance(loss, (tuple, list)):

if isinstance(loss[0], Tensor) and isinstance(loss[0].asnumpy(), np.ndarray):

loss = loss[0]

if isinstance(loss, Tensor) and isinstance(loss.asnumpy(), np.ndarray):

loss = np.mean(loss.asnumpy())

cur\_step\_in\_epoch = (cb\_params.cur\_step\_num - 1) % cb\_params.batch\_num + 1

if isinstance(loss, float) and (np.isnan(loss) or np.isinf(loss)):

raise ValueError("epoch: {} step: {}. Invalid loss, terminating training.".format(

cb\_params.cur\_epoch\_num, cur\_step\_in\_epoch))

if self.\_per\_print\_times != 0 and cb\_params.cur\_step\_num % self.\_per\_print\_times == 0:

# TEST

print("step: %s, loss is %s, fps is %s" % (cur\_step\_in\_epoch, loss, step\_fps), flush=True)

#### 定义类用于训练模型

def train\_net(data\_dir, cross\_valid\_ind=1, epochs=400, batch\_size=16, lr=0.0001, run\_distribute=False, cfg=None):

if run\_distribute:

init()

group\_size = get\_group\_size()

parallel\_mode = ParallelMode.DATA\_PARALLEL

context.set\_auto\_parallel\_context(parallel\_mode=parallel\_mode,

device\_num=group\_size,

gradients\_mean=False)

net = UNet(n\_channels=cfg['num\_channels'], n\_classes=cfg['num\_classes'])

if cfg['resume']:

param\_dict = load\_checkpoint(cfg['resume\_ckpt'])

load\_param\_into\_net(net, param\_dict)

criterion = CrossEntropyWithLogits()

train\_dataset, \_ = create\_dataset(data\_dir, epochs, batch\_size, True, cross\_valid\_ind, run\_distribute)

train\_data\_size = train\_dataset.get\_dataset\_size()

print("dataset length is:", train\_data\_size)

ckpt\_config = CheckpointConfig(save\_checkpoint\_steps=train\_data\_size,

keep\_checkpoint\_max=cfg['keep\_checkpoint\_max'])

ckpoint\_cb = ModelCheckpoint(prefix='ckpt\_unet\_medical\_adam',

directory='./ckpt\_{}/'.format(device\_id),

config=ckpt\_config)

optimizer = nn.Adam(params=net.trainable\_params(), learning\_rate=lr, weight\_decay=cfg['weight\_decay'],

loss\_scale=cfg['loss\_scale'])

loss\_scale\_manager = mindspore.train.loss\_scale\_manager.FixedLossScaleManager(cfg['FixedLossScaleManager'], False)

model = Model(net, loss\_fn=criterion, loss\_scale\_manager=loss\_scale\_manager, optimizer=optimizer, amp\_level="O3")

print("============== Starting Training ==============")

model.train(2, train\_dataset, callbacks=[StepLossTimeMonitor(batch\_size=batch\_size), ckpoint\_cb],

dataset\_sink\_mode=False)

print("============== End Training ==============")

#### 训练模型

data\_url = './data'

run\_distribute = False

epoch\_size = cfg\_unet['epochs'] if not run\_distribute else cfg\_unet['distribute\_epochs']

train\_net(data\_dir=data\_url, cross\_valid\_ind=cfg\_unet['cross\_valid\_ind'], epochs=epoch\_size,

batch\_size=cfg\_unet['batchsize'], lr=cfg\_unet['lr'], run\_distribute=run\_distribute,

cfg=cfg\_unet)

输出：

dataset length is: 600

============== Starting Training ==============

step: 1, loss is 0.7004798, fps is 0.12990480406685767

step: 2, loss is 0.68976885, fps is 64.92636426522448

step: 3, loss is 0.6817631, fps is 65.16214492050005

step: 4, loss is 0.66404736, fps is 65.24837192298793

step: 5, loss is 0.6231464, fps is 65.20215731367617

step: 6, loss is 0.54606646, fps is 65.20861961577666

step: 7, loss is 0.5840454, fps is 65.26512424021395

step: 8, loss is 0.55446315, fps is 65.24469264183742

step: 9, loss is 0.53775936, fps is 65.21400584613471

step: 10, loss is 0.5526323, fps is 65.20811272225887

step: 11, loss is 0.57306504, fps is 65.27832909550388

step: 12, loss is 0.5498034, fps is 65.26740931362538

step: 13, loss is 0.5426476, fps is 65.25706428673531

step: 14, loss is 0.5266376, fps is 65.17505494473477

step: 15, loss is 0.54430926, fps is 65.24171145973388

step: 16, loss is 0.5315207, fps is 65.2474837826486

......

……

step: 598, loss is 0.19874662, fps is 65.3874733759059

step: 599, loss is 0.18141083, fps is 65.31644997965822

step: 600, loss is 0.1776892, fps is 65.39129621365221

============== End Training ==============

### 模型验证和测试

#### 定义类用于评估模型性能

图像分割算法的评估指标通常使用Dice coefficient，Dice coefficient和目标检测任务中的IoU指标，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：

在该实验中，通过“(2x标签和预测值相同的数量)/(2x 总像素数量)”得到一张图的Dice值。

试题5: 定义一个名为dice\_coeff的类，用于计算每张验证集图像的Dice以及返回验证集中Dice的均值。

解答5：

class dice\_coeff(nn.Metric):

#计算dice用于评估模型

def \_\_init\_\_(self):

super(dice\_coeff, self).\_\_init\_\_()

self.clear()

def clear(self):

self.\_dice\_coeff\_sum = 0

self.\_samples\_num = 0

def update(self, \*inputs):

if len(inputs) != 2:

raise ValueError('Mean dice coeffcient need 2 inputs (y\_pred, y), but got {}'.format(len(inputs)))

y\_pred = self.\_convert\_data(inputs[0])

y = self.\_convert\_data(inputs[1])

self.\_samples\_num += y.shape[0]

y\_pred = y\_pred.transpose(0, 2, 3, 1)

y = y.transpose(0, 2, 3, 1)

y\_pred = softmax(y\_pred, axis=3)

#计算交集

inter = np.dot(y\_pred.flatten(), y.flatten())

#计算并集

union = np.dot(y\_pred.flatten(), y\_pred.flatten()) + np.dot(y.flatten(), y.flatten())

#计算交并比

single\_dice\_coeff = 2 \* float(inter) / float(union + 1e-6)

print("single dice coeff is:", single\_dice\_coeff)

self.\_dice\_coeff\_sum += single\_dice\_coeff

def eval(self):

if self.\_samples\_num == 0:

raise RuntimeError('Total samples num must not be 0.')

return self.\_dice\_coeff\_sum / float(self.\_samples\_num)

#### 测试模型效果

定义函数用于在验证集上计算Dice值：

def test\_net(data\_dir, ckpt\_path, cross\_valid\_ind=1, cfg=None):

#用验证集测试模型表现。

net = UNet(n\_channels=cfg['num\_channels'], n\_classes=cfg['num\_classes'])

param\_dict = load\_checkpoint(ckpt\_path)

load\_param\_into\_net(net, param\_dict)

criterion = CrossEntropyWithLogits()

\_, valid\_dataset = create\_dataset(data\_dir, 1, 1, False, cross\_valid\_ind, False)

model = Model(net, loss\_fn=criterion, metrics={"dice\_coeff": dice\_coeff()})

print("============== Starting Evaluating ============")

dice\_score = model.eval(valid\_dataset, dataset\_sink\_mode=False)

print("Cross valid dice coeff is:", dice\_score)

调用函数进行预测：

ckpt\_path = './ckpt\_2/ckpt\_unet\_medical\_adam-2\_600.ckpt'

test\_net(data\_dir=data\_url, ckpt\_path=ckpt\_path, cross\_valid\_ind=cfg\_unet['cross\_valid\_ind'],

cfg=cfg\_unet)

试题6：请在test\_net原函数基础上调用测试数据集进行预测，并可视化预测结果，注意：Unet的输入图像尺寸是大于输出图像的尺寸的，需要对原图进行预处理。

解答6：

def test\_net(data\_dir, ckpt\_path, cross\_valid\_ind=1, cfg=None):

net = UNet(n\_channels=cfg['num\_channels'], n\_classes=cfg['num\_classes'])

param\_dict = load\_checkpoint(ckpt\_path)

load\_param\_into\_net(net, param\_dict)

criterion = CrossEntropyWithLogits()

\_, valid\_dataset = create\_dataset(data\_dir, 1, 1, False, cross\_valid\_ind, False)

model = Model(net, loss\_fn=criterion, metrics={"dice\_coeff": dice\_coeff()})

print("============== Starting Evaluating ============")

dice\_score = model.eval(valid\_dataset, dataset\_sink\_mode=False)

print("Cross valid dice coeff is:", dice\_score)

testimage=np.array([np.array(p) for p in ImageSequence.Iterator(Image.open("./data/test-volume.tif"))])

#读取一张测试集图像

testdata=testimage[10]

image = Image.fromarray(testdata)

#对图像进行缩放

image = image.resize((388, 388))

testdata = np.asarray(image)

#根据原论文对原图进行扩充，通过numpy的pad函数，将原图像边缘像素“外翻”，将388\*388的图像扩充至572\*572。

testdata = np.pad(testdata, ((92, 92),(92, 92) ), 'symmetric')

#和训练时一样进行归一化处理

testdata = testdata/127.5-1

testdata = testdata.astype(np.float32)

testdata = testdata.reshape(1,1,572,572)

output = model.predict(Tensor(testdata))

pred = np.argmax(output.asnumpy(), axis=1)

pred = pred.reshape(388, 388)

#可视化测试图像和模型推理结果。

plt.figure(figsize=(10,10))

plt.subplot(2,2,1)

plt.imshow(testimage[10],cmap='gray')

plt.subplot(2,2,2)

plt.imshow(pred,cmap='gray')

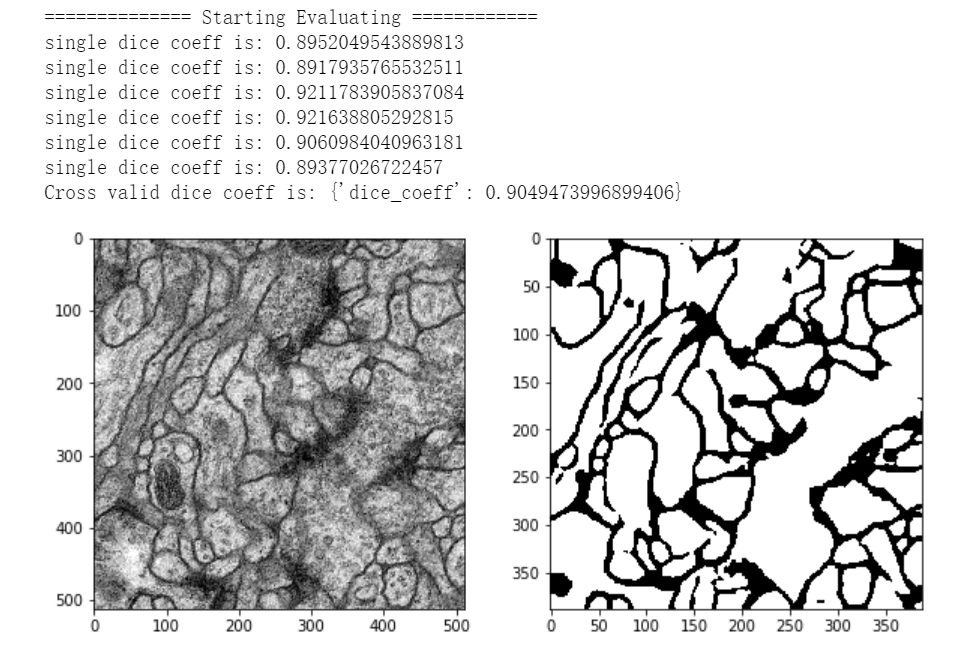
plt.show()

ckpt\_path = './ckpt\_2/ckpt\_unet\_medical\_adam-2\_600.ckpt'

test\_net(data\_dir=data\_url, ckpt\_path=ckpt\_path, cross\_valid\_ind=cfg\_unet['cross\_valid\_ind'],

cfg=cfg\_unet)

输出：



验证结果展示

## 实验总结

本次实验介绍对算机视觉当中利用深度学习U-Net网络完成图像分割任务，并和传统算法的效果进行了对照。实验过程中，需学习深度学习如何进行数据增强，如何实现定义复杂的网络结构，如何训练和提升模型效果及微调训练等操作技巧。