《计算机视觉》第二次实验：图像分割

一、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。

二、实验环境

- Python 3.7.5
- Mindspore 1.5
- Matplotlib 3.2.2
- Numpy 1.18.5
- 平台：华为AI平台
  数据：医学图像数据集

三、实验内容

任务一：按照华为平台实验手册进行操作

> 要求：熟悉实验环境，掌握Unet图像分割原理及程序流程
> 具体：记录并观察实验结果，如损失随迭代轮数变化等

任务二：Unet-5模型性能分析

> 调节实验参数(至少1种)，进行实验对比及分析
> 实验参数包括：训练数据比例，批次大小，迭代轮数，学习速率等

任务三：完成华为平台实验手册思考题

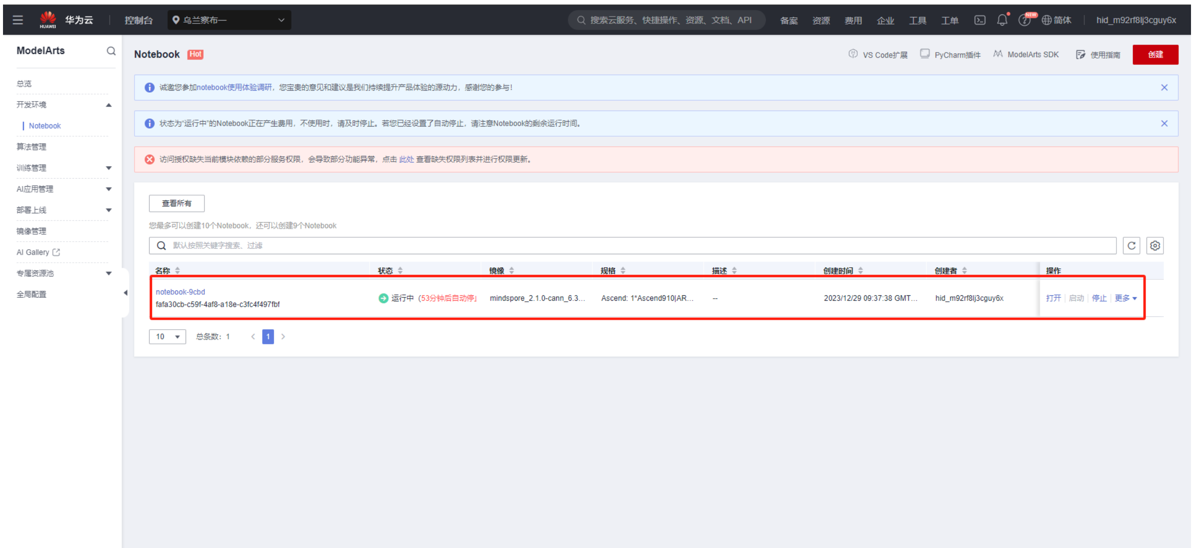
> 要求：尝试自动动手编写代码实现，至少完成试题1和试题2

四、实验过程及结果

1、任务一：按照华为平台实验手册进行操作

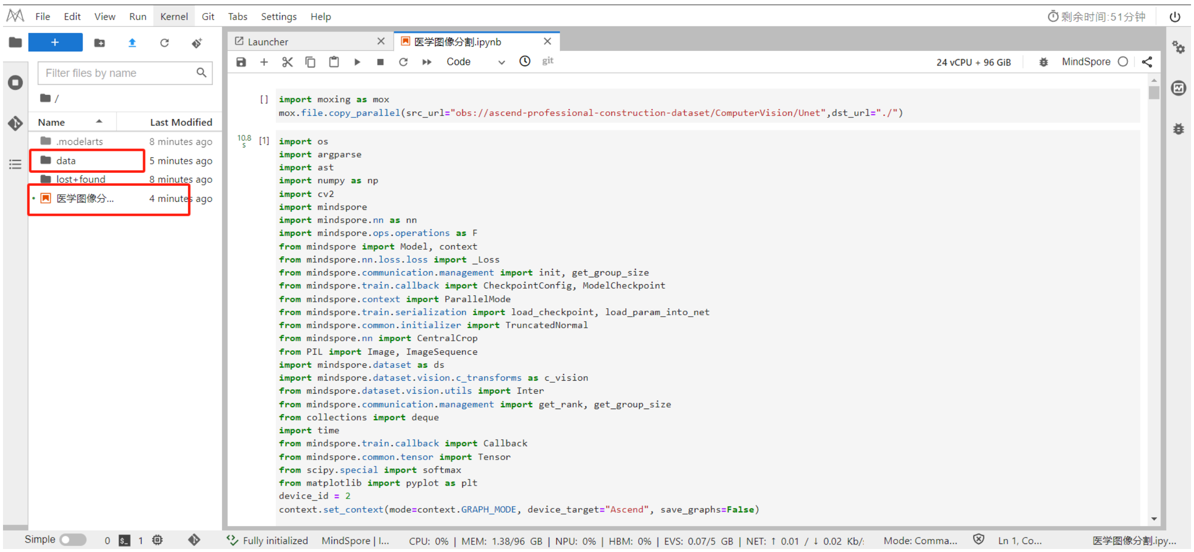
（1）创建华为云Notebook

按照助教老师的示范，创建并启动Notebook，如下图所示：



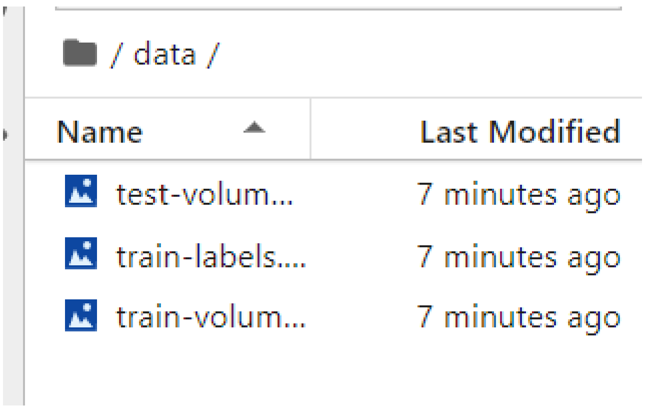
（2）导入相关实验模块

在Notebook中导入实验二所给的代码和数据集，如图所示：



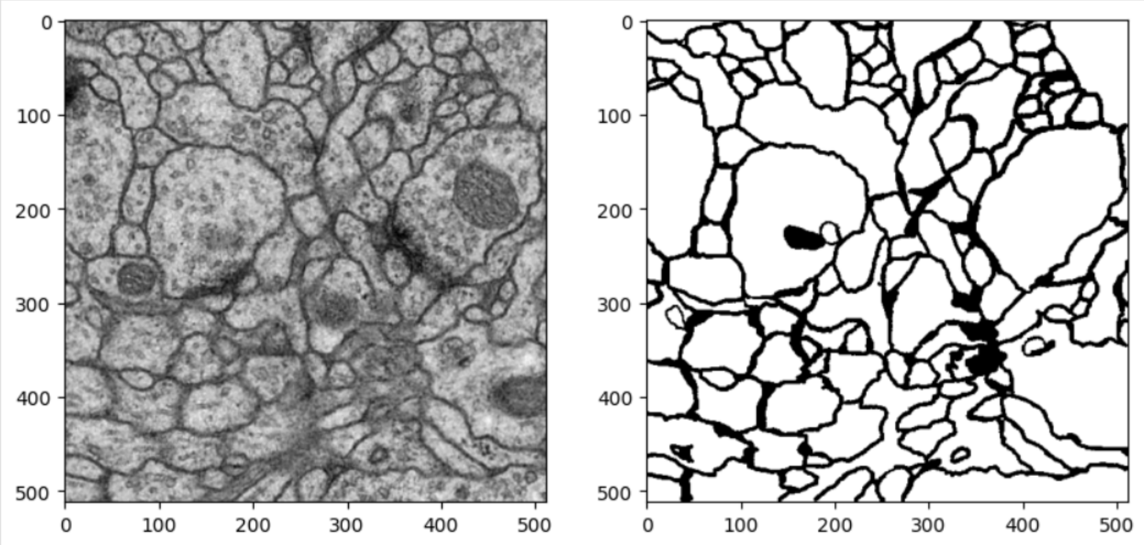
（3）下载和查看数据集

由于数据集下载网站打开不了，数据集采用手动导入，如下图所示：



打印图像和标签的形状，并展示第一张图像和标签：

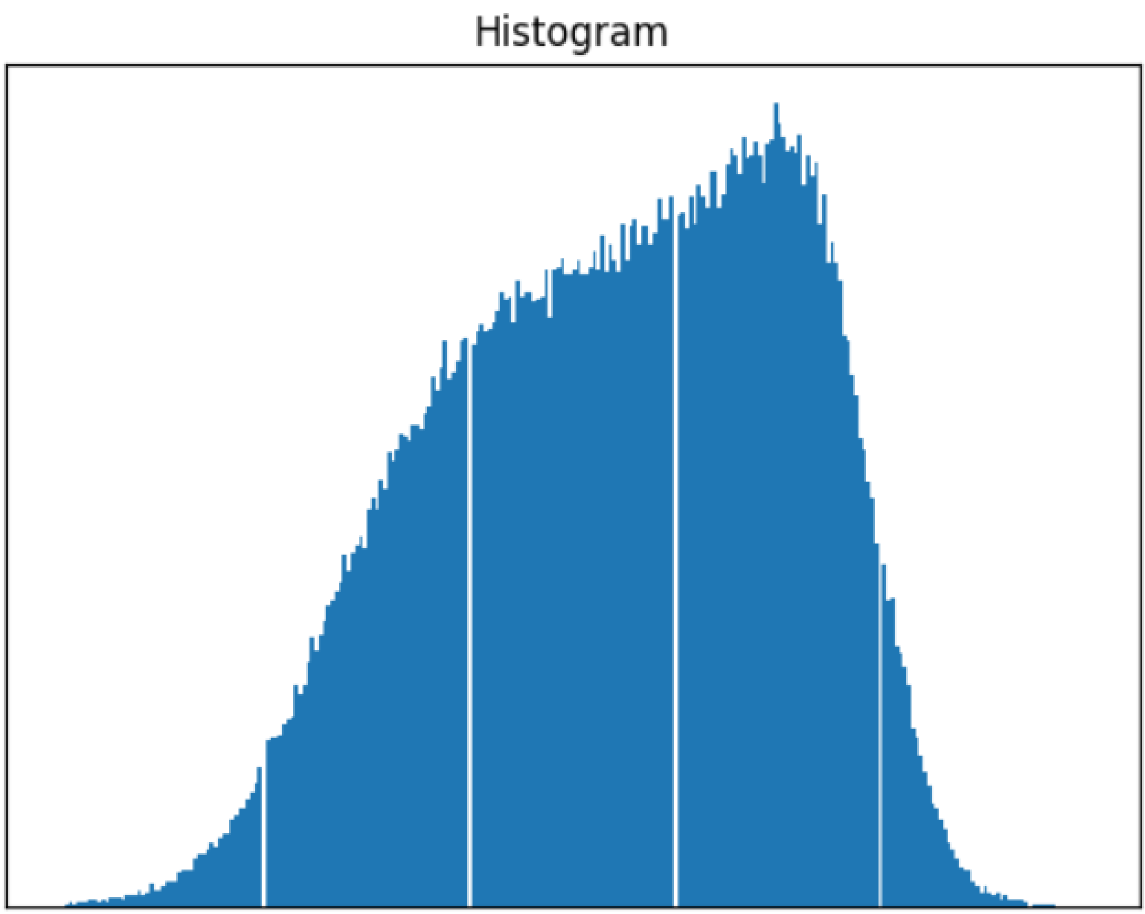
```
(30, 512, 512)
(30, 512, 512)
```



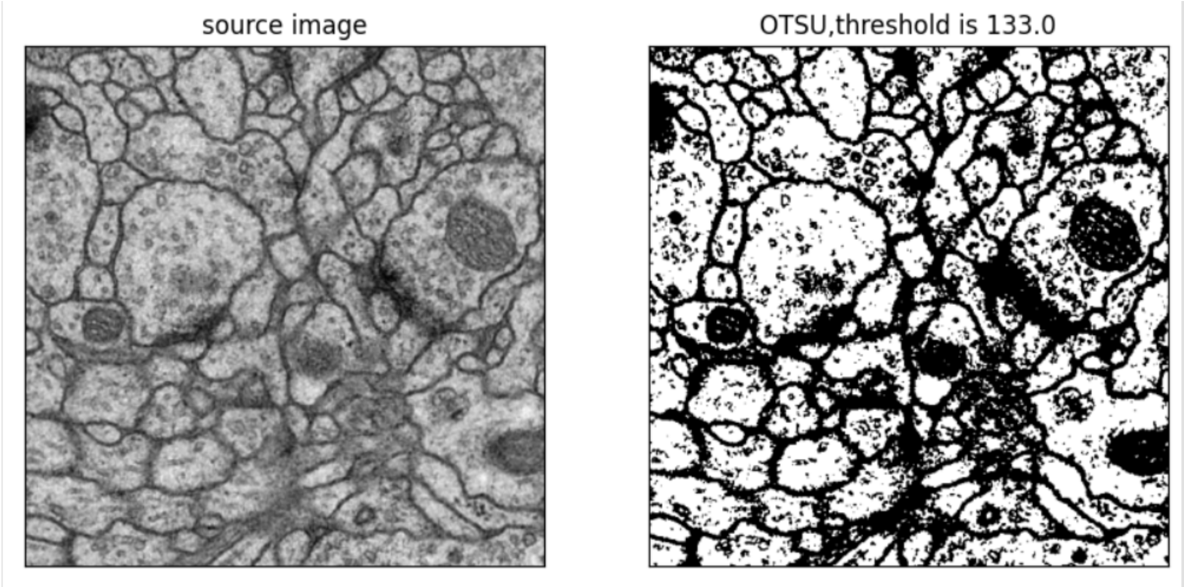
> 数据集一共共三个文件：train-volume.tif，train-labels.tif，test-volume.tif
>
> 第一个文件为训练集图像，该TIF文件共有30个通道，每个通道为一张灰度图像。
>
> 第二个文件为训练集标签，该TIF文件共有30个通道，每个通道为一张灰度图像（像素值仅为0或255）。
>
> 第三个文件为测试集图像，该TIF文件同样有30个通道，每个通道为一张灰度图像。

（4）使用大津阈值法进行图像分割

用OpenCV和matplotlib绘制一张训练集中的直方图：



用OpenCV中的python接口实现基于大津阈值法的图像二值化分割，结果如下：



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。

（5）基于神经网络的图像分割算法

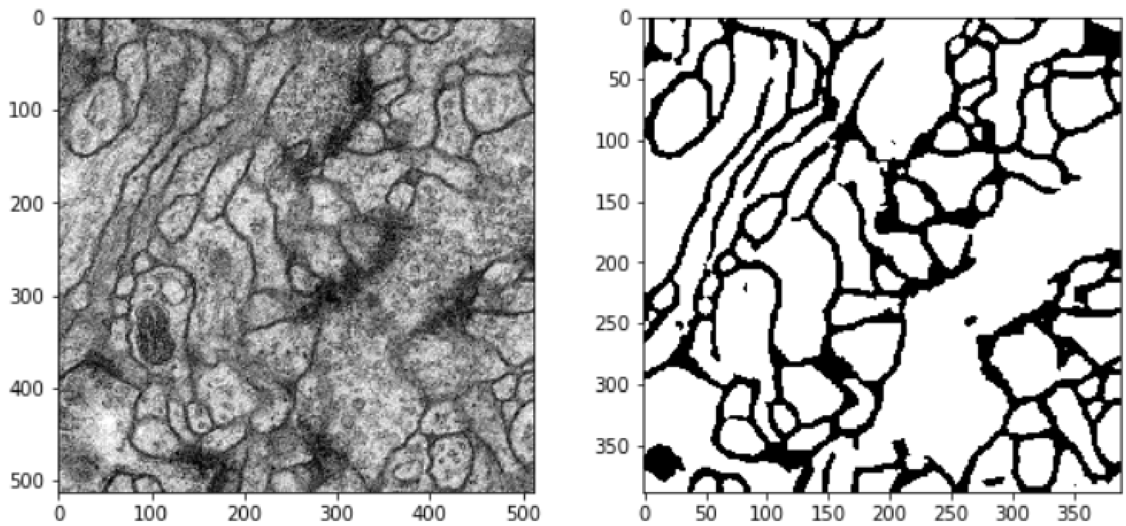
训练模型：

```
dataset length is: 600
============== Starting Training ==============
step: 1, loss is 0.7011225, fps is 0.1296080562471482
step: 2, loss is 0.68978643, fps is 64.62294842017279
step: 3, loss is 0.681577, fps is 64.95458038688845
step: 4, loss is 0.6631299, fps is 65.10183939334264
step: 5, loss is 0.6250401, fps is 65.19031308588248
……
step: 596, loss is 0.19244952, fps is 64.00372336113456
step: 597, loss is 0.199357, fps is 63.69034824812988
step: 598, loss is 0.19228645, fps is 63.73172014928917
step: 599, loss is 0.18943003, fps is 64.52291462755677
step: 600, loss is 0.19491342, fps is 58.610975835554605

step: 1, loss is 0.19198747, fps is 60.99641341198498
step: 2, loss is 0.20046772, fps is 64.90520256259478
step: 3, loss is 0.19927795, fps is 64.98250642476872
step: 4, loss is 0.19961753, fps is 64.91524794155885
step: 5, loss is 0.18876144, fps is 64.82895916411636
……
step: 596, loss is 0.17452283, fps is 65.2429165856504
step: 597, loss is 0.1757165, fps is 65.34729691700821
step: 598, loss is 0.17769071, fps is 65.18429761297686
step: 599, loss is 0.16920947, fps is 64.87257654146269
step: 600, loss is 0.17143147, fps is 64.84192544453366
============== End Training ==============
```

测试模型效果：

```
============== Starting Evaluating ============
single dice coeff is: 0.9053852793871535
single dice coeff is: 0.9105223777160442
single dice coeff is: 0.9271962117878177
single dice coeff is: 0.928957563919879
single dice coeff is: 0.9209360657011245
single dice coeff is: 0.9092419754966059
Cross valid dice coeff is: {'dice_coeff': 0.9170399123347709}
```

图像分割结果:



2、任务二：Unet-5模型性能分析

(1) 调整训练数据比例

```python
def _get_val_train_indices(length, fold, ratio=0.6):
    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)
    np.random.shuffle(indices)
```

- 0.6

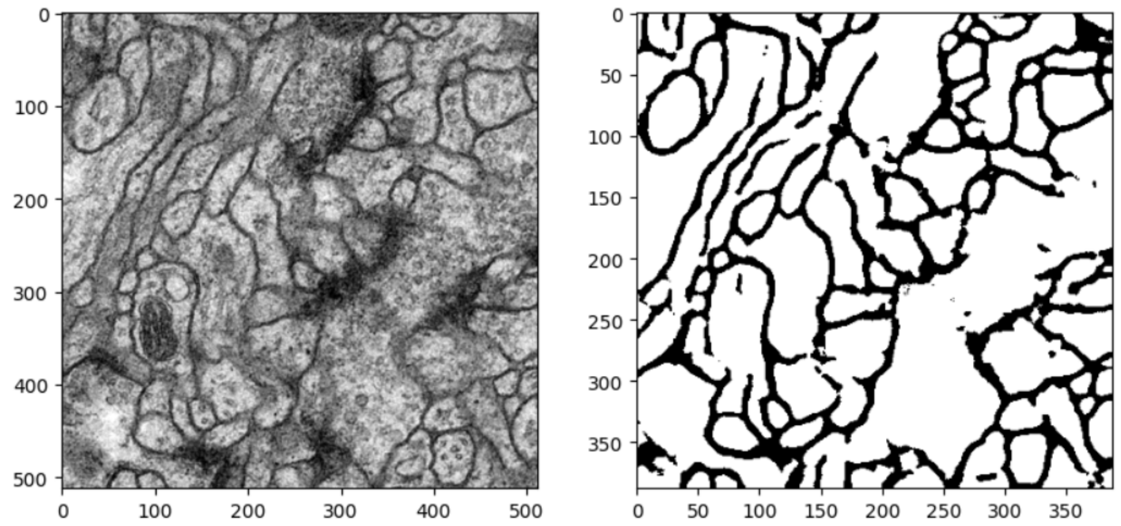
```
dataset length is: 450
============== Starting Training ==============
step: 1, loss is 6.229542, fps is 0.8478258804247878
step: 2, loss is 1.6254226, fps is 82.8969125905137
step: 3, loss is 0.7565664, fps is 82.45841560678798
step: 4, loss is 0.65023875, fps is 80.63643179851965
step: 5, loss is 0.6430286, fps is 83.62339800102927
......
step: 446, loss is 0.23220548, fps is 79.34478143459117
step: 447, loss is 0.23096965, fps is 73.57774583617025
step: 448, loss is 0.23930849, fps is 78.1748228776184
step: 449, loss is 0.22903346, fps is 75.21863651444097
step: 450, loss is 0.23570503, fps is 78.67642234865582

step: 1, loss is 0.22504225, fps is 75.33710754944256
```

```
step: 2, loss is 0.2375894, fps is 83.39415870633103
step: 3, loss is 0.26833087, fps is 83.31526628188622
step: 4, loss is 0.260363, fps is 83.62287699465931
step: 5, loss is 0.23059039, fps is 82.30560658946578
……
step: 446, loss is 0.23394093, fps is 83.71989600642725
step: 447, loss is 0.19775192, fps is 83.2252713455166
step: 448, loss is 0.2431247, fps is 84.4934825224835
step: 449, loss is 0.20444164, fps is 84.33007785993254
step: 450, loss is 0.23221028, fps is 51.15441562084568
============== End Training ==============
```

测试模型效果：

```
============== Starting Evaluating ============
single dice coeff is: 0.9221514753646984
single dice coeff is: 0.9220598962929997
single dice coeff is: 0.9256643246337737
single dice coeff is: 0.928297699006741
single dice coeff is: 0.894786875717176
single dice coeff is: 0.9310753021286923
single dice coeff is: 0.9031060043141785
single dice coeff is: 0.9313656853912232
single dice coeff is: 0.93335024372403
single dice coeff is: 0.9140544493627842
single dice coeff is: 0.9154395397825226
single dice coeff is: 0.9223629799639241
Cross valid dice coeff is: {'dice_coeff': 0.9203095396402285}
```

图像分割结果：



- 0.8

  见任务一

- 0.9

  训练模型：

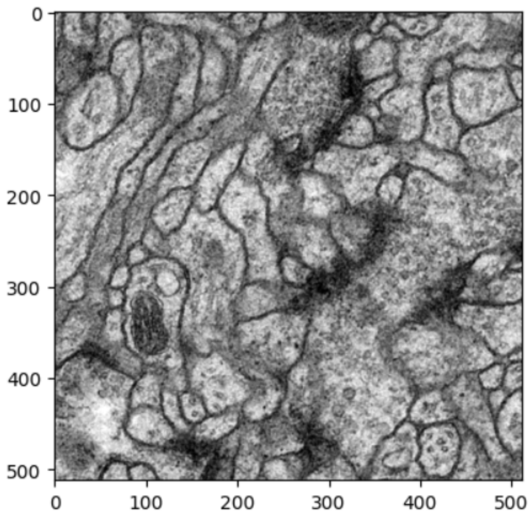
```
dataset length is: 675
============== Starting Training ==============
step: 1, loss is 4.7443357, fps is 0.6725245269417375
```

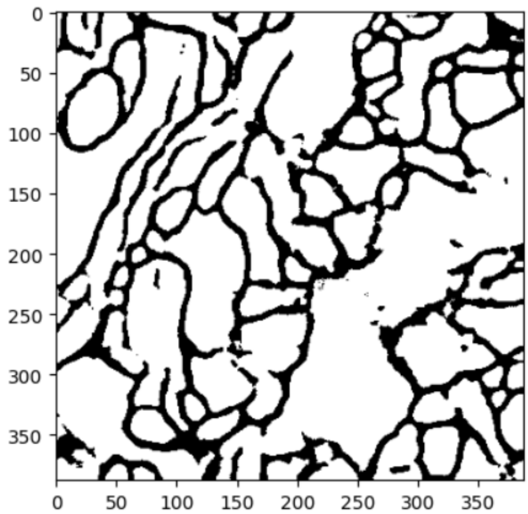
```
step: 2, loss is 1.356718, fps is 80.35209444614067
step: 3, loss is 1.0659987, fps is 81.94010493297306
step: 4, loss is 0.88847524, fps is 77.06239736897103
step: 5, loss is 0.70947105, fps is 81.75314056271867
……
step: 671,  loss is 0.2172605,  fps is 79.97690867509709
step: 672, loss is 0.20997886,  fps s 83.5978763213823
step: 673, loss is .21342328,  fps is 75.07743253432857
step: 674, loss is .21979848,  fps is 73.83712427534199
step: 675, loss is 0.21255963,  fps is 79.05484102769499

step: 1, loss is 0.21080986,  fps is 75.32991942671512
step: 2, loss is 0.1956191, fps is 66.06757298222712
step: 3, loss is 0.2050862,  fps is 80.97795904576883
step: 4, loss is .22827275,  fps is 81.9097544029242
step: 5, loss is 0.2115972,  fps is 77.48736695525263
……
step: 671, loss is 0.1914054, fps is 83.49906184833297
step: 672, loss is 0.18279818, fps is 83.68127009145117
step: 673, loss is 0.1815741, fps is 83.69066230352128
step: 674, loss is 0.18357044, fps is 37.67904389842937
step: 675, loss is 0.18302467, fps is 57.81812699181262
============== End Training ==============
```

测试模型效果：

```
============== Starting Evaluating ============
single dice coeff is: 0.9322946157655563
single dice coeff is: 0.921598020537904
single dice coeff is: 0.9056918040544771
Cross valid dice coeff is: {'dice_coeff': 0.9198614801193125}
```

图像分割结果：





训练数据比例从0.8调节为0.6或0.9时，训练损失均小幅增加。Dice coefficient相比原来小幅增加。

（2）调节训练批次大小

```
cfg_unet = {
    'name': 'Unet',
    'lr': 0.0001,
    'epochs': 400,
    'distribute_epochs': 1600,
    'batchsize': 32,
    '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': './',
}
```

- 16

  见任务一

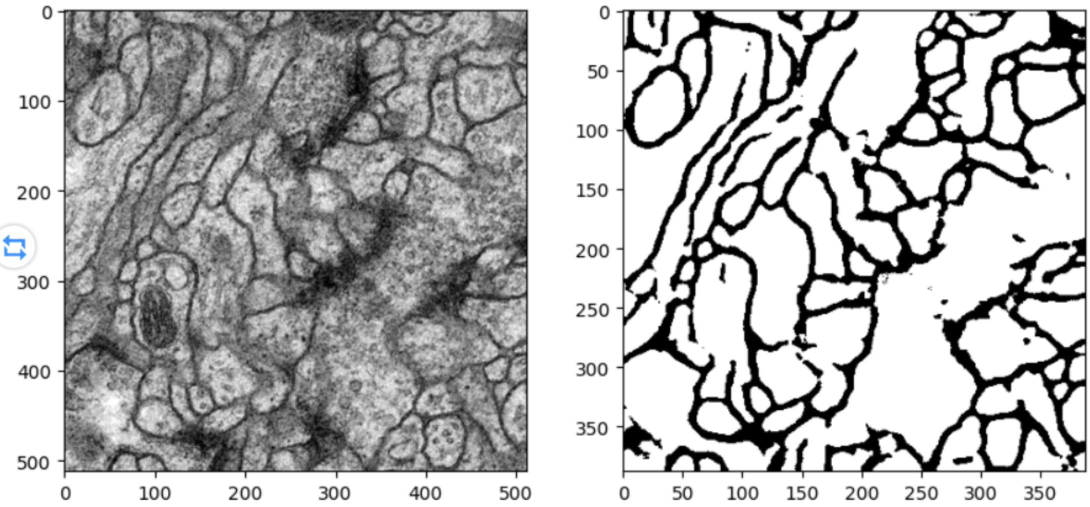
- 32

```
dataset length is: 300
============== Starting Training ==============
step: 1, loss is 4.4729686, fps is 0.5828683780626811
step: 2, loss is 2.7984843, fps is 83.73953581232843
step: 3, loss is 1.5922848, fps is 81.94085530802212
step: 4, loss is 1.1246264, fps is 81.96422386101877
step: 5, loss is 0.86831105, fps is 81.79185078066774
……
step: 296, loss is 0.25754672, fps is 78.28859243220795
step: 297, loss is 0.28150916, fps is 81.53323468021567
step: 298, loss is 0.2521141, fps is 73.5042078612827
step: 299, loss is 0.26311678, fps is 81.03247089223561
step: 300, loss is 0.2561699, fps is 74.73384623008637

step: 1, loss is 0.26409972, fps is 75.54314255111856
step: 2, loss is 0.26712176, fps is 82.87500493973523
step: 3, loss is 0.26626617, fps is 82.06440409974338
step: 4, loss is 0.2696544, fps is 82.97824612597798
step: 5, loss is 0.28020397, fps is 81.96442407750294
……
step: 296, loss is 0.21209514, fps is 82.92871976081129
step: 297, loss is 0.22208183, fps is 82.93645756092738
step: 298, loss is 0.21220288, fps is 82.95609037929165
step: 299, loss is 0.21597756, fps is 82.70280955233004
step: 300, loss is 0.20603812, fps is 82.85229054472815
============== End Training ==============
```

测试模型效果：

```
============== Starting Evaluating ============
single dice coeff is: 0.9022660878410403
single dice coeff is: 0.9054967466012757
single dice coeff is: 0.9268221539778871
single dice coeff is: 0.9295463737871804
single dice coeff is: 0.9148295769374601
single dice coeff is: 0.9050554122576601
Cross valid dice coeff is: {'dice_coeff': 0.9140027252337507}
```

图像分割结果：



batch_size扩大一倍后，训练损失小幅增加，产出的图片边缘更加平滑

(3) 调节训练迭代轮数

```
cfg_unet = {
    'name': 'Unet',
    'lr': 0.0001,
    'epochs': 40,
    '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': './',
}
```

- 400

  见任务一

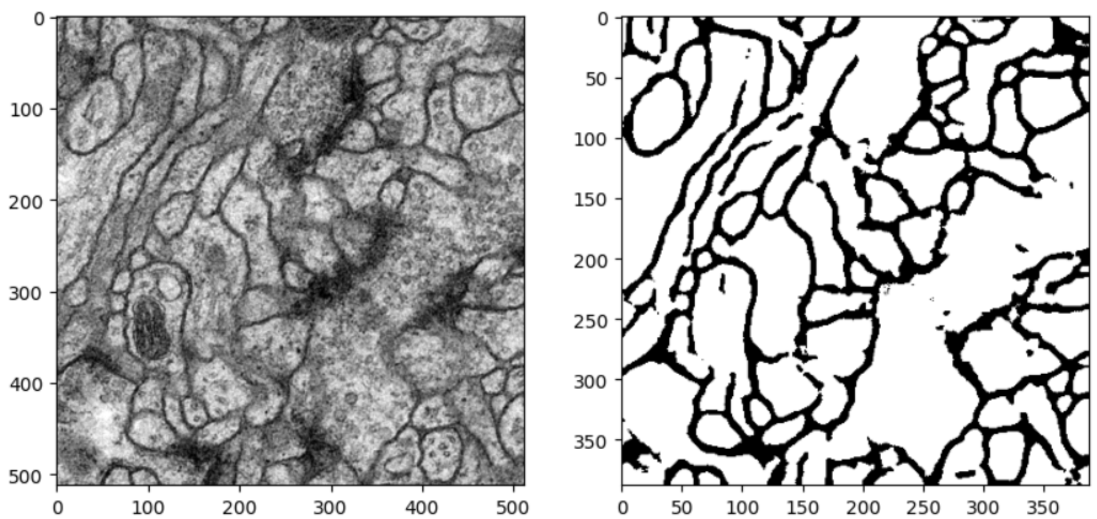
- 40

```
dataset length is: 60
============== Starting Training ==============
step: 1, loss is 4.849551, fps is 0.8277288914181513
step: 2, loss is 1.3416855, fps is 82.60335317931668
step: 3, loss is 0.98884016, fps is 82.46540719750499
step: 4, loss is 0.8747877, fps is 83.78072573738524
step: 5, loss is 0.73593897, fps is 83.62548209143257
……
step: 56, loss is 0.5035998, fps is 83.98339575983671
step: 57, loss is 0.48886293, fps is 83.23910406366517
step: 58, loss is 0.5124778, fps is 83.8329508673878
step: 59, loss is 0.49857304, fps is 83.4799500430406
step: 60, loss is 0.49186188, fps is 83.52026125570004

step: 1, loss is 0.4829633, fps is 83.35334791513738
step: 2, loss is 0.48049444, fps is 83.40773663042451
step: 3, loss is 0.47794884, fps is 83.45970523090163
step: 4, loss is 0.47608435, fps is 83.22723242134452
step: 5, loss is 0.47743502, fps is 83.41406068410383
……
step: 56, loss is 0.40164235, fps is 82.83245122047555
step: 57, loss is 0.4064544, fps is 83.06087025402626
step: 58, loss is 0.39093438, fps is 82.59369515506118
step: 59, loss is 0.194237, fps is 82.89885822214701
step: 60, loss is 0.16719997, fps is 82.5323432400058
============== End Training ==============
```

测试模型效果：

```
============== Starting Evaluating ============
single dice coeff is: 0.9022660878410403
single dice coeff is: 0.9054967466012757
single dice coeff is: 0.9268221539778871
single dice coeff is: 0.9295463737871804
single dice coeff is: 0.9148295769374601
single dice coeff is: 0.9050554122576601
Cross valid dice coeff is: {'dice_coeff': 0.9140027252337507}
```

图像分割结果：



迭代轮数减少为40后，训练时的损失明显增大。

（4）调节learning-rate

```
cfg_unet = {
    'name': 'Unet',
    'lr': 0.0003,
    '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': './',
}
```

- 0.0001

  见任务一

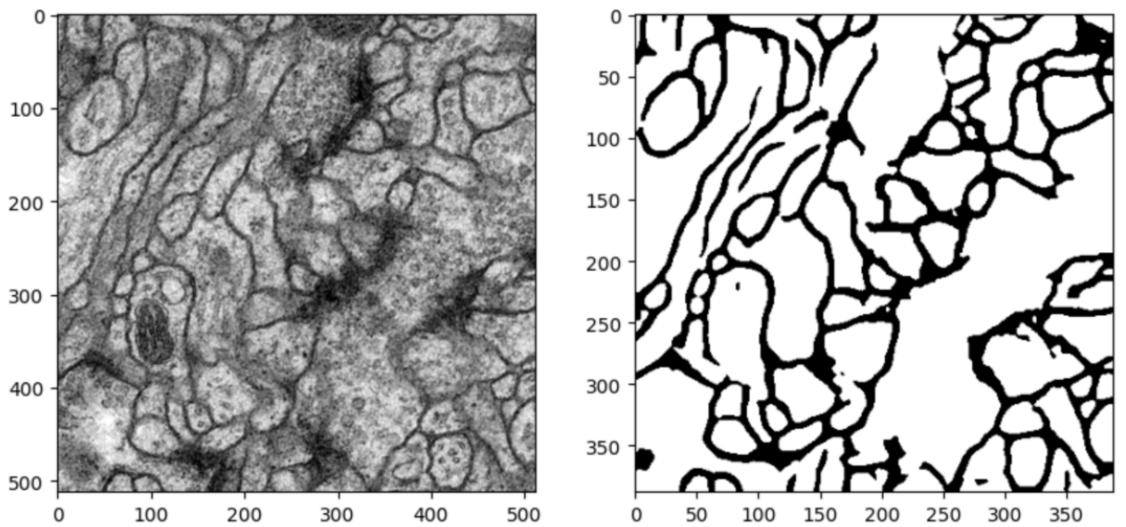
- 0.0003

```
dataset length is: 600
============== Starting Training ==============
step: 1, loss is 0.7011225, fps is 0.1296080562471482
step: 2, loss is 0.68978643, fps is 64.62294842017279
step: 3, loss is 0.681577, fps is 64.95458038688845
step: 4, loss is 0.6631299, fps is 65.10183939334264
step: 5, loss is 0.6250401, fps is 65.19031308588248
……
step: 596, loss is 0.19244952, fps is 64.00372336113456
step: 597, loss is 0.199357, fps is 63.69034824812988
step: 598, loss is 0.19228645, fps is 63.73172014928917
step: 599, loss is 0.18943003, fps is 64.52291462755677
step: 600, loss is 0.19491342, fps is 58.610975835554605

step: 1, loss is 0.19198747, fps is 60.99641341198498
step: 2, loss is 0.20046772, fps is 64.90520256259478
step: 3, loss is 0.19927795, fps is 64.98250642476872
step: 4, loss is 0.19961753, fps is 64.91524794155885
step: 5, loss is 0.18876144, fps is 64.82895916411636
……
step: 596, loss is 0.21452283, fps is 65.2429165856504
step: 597, loss is 0.2157165, fps is 65.34729691700821
step: 598, loss is 0.20769071, fps is 65.18429761297686
step: 599, loss is 0.21920947, fps is 64.87257654146269
step: 600, loss is 0.20143147, fps is 64.84192544453366
============== End Training ==============
```

测试模型效果:

```
============== Starting Evaluating ============
single dice coeff is: .9012603640237051
single dice coeff is: .9038953956373993
single dice coeff is: .9260539281205189
single dice coeff is: .9278957406702245
single dice coeff is: .9148563916487551
single dice coeff is: .9042405301736068
Cross valid dice coeff is: {'dice coeff': 0.9130337250457017}
```

图像分割结果:



lr越大，训练时的损失有所增加，但增加的幅度较小，效果越差。从产出的图片来看，细节有所丢失。

3、任务三：完成华为平台实验手册思考题

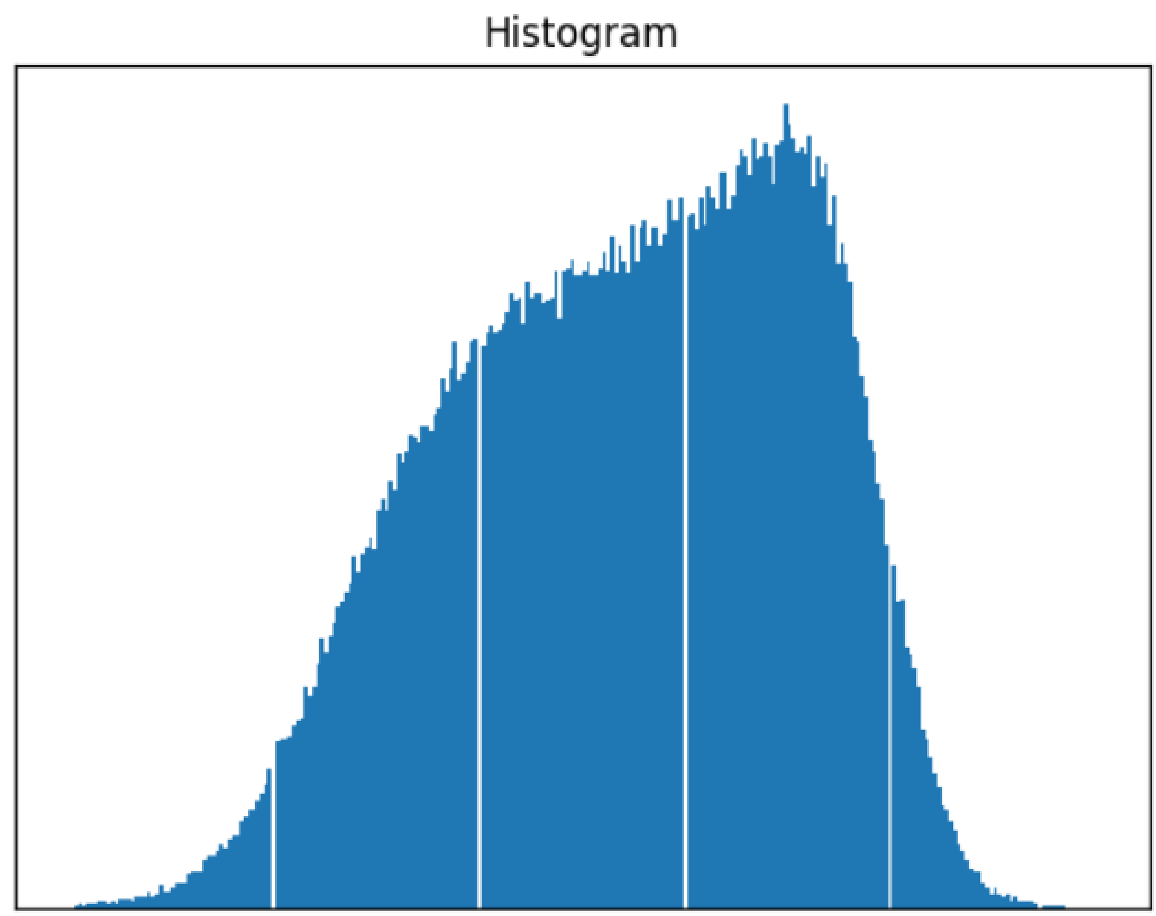
（1）试题1：请用OpenCV和matplotlib绘制一张训练集中的直方图。

加载图像并绘制直方图:

```
import cv2
import matplotlib.pyplot as plt

# 读取图像
image_path = './data/test-volume.tif'  #图像文件路径
image = cv2.imread(image_path, 0)  # 使用0表示以灰度模式读取图像

# 绘制直方图
plt.hist(image.ravel(), 256, [0, 256])
plt.title("Histogram")
plt.xticks([])
plt.yticks([])

# 显示图像和直方图
plt.figure(figsize=(10, 5))
plt.subplot(121), plt.imshow(image, cmap='gray'), plt.title('Image')
plt.subplot(122), plt.plot(plt.hist(image.ravel(), 256, [0, 256])),
plt.title('Histogram')
```

```
plt.show()
```

结果图：



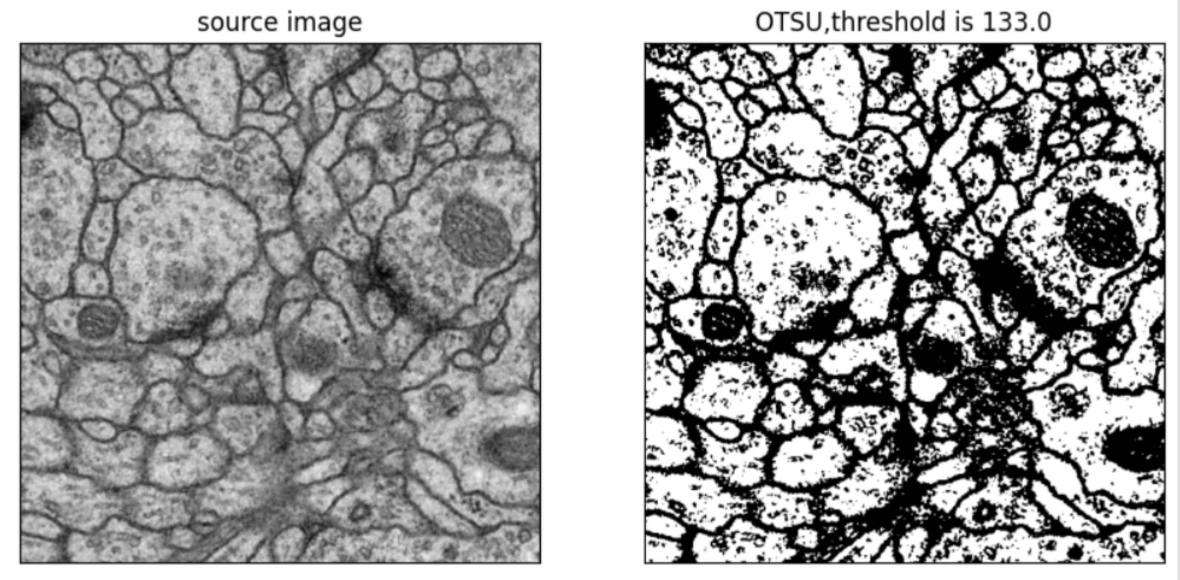
（2）试题2：请用OpenCV中的python接口实现基于大津阈值法的图像二值化分割。

使用OpenCV中的Python接口实现基于大津阈值法的图像二值化分割，代码如下：

```
import cv2
import matplotlib.pyplot as plt

# 读取图像
image_path = 'your_image_path.jpg'  # 将 'your_image_path.jpg' 替换为你的图像文件路径
image = cv2.imread(image_path, 0)  # 使用0表示以灰度模式读取图像

# 大津阈值法二值化
ret1, th1 = cv2.threshold(src=image, thresh=0, maxval=255, type=cv2.THRESH_OTSU)

# 显示原图和经过大津阈值法后的二值化图像
plt.figure(figsize=(10, 5))

# 显示原图
plt.subplot(121)
plt.imshow(image, 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, cmap='gray')
plt.title("OTSU, Threshold is " + str(ret1))
plt.xticks([])
plt.yticks([])

plt.show()
```

结果图：



（3）试题3：使用MindSpore搭建Unet类，用于构建网络。

包含U-Net网络的相应模块，并使用MindSpore构建UNet类：

```
import mindspore.nn as nn
from mindspore.ops import operations as P

class DoubleConv(nn.Cell):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=3,
stride=1, padding=1)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
stride=1, padding=1)

    def construct(self, x):
        x = self.conv(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.relu(x)
        return x

class Down(nn.Cell):
    def __init__(self, in_channels, out_channels):
        super(Down, self).__init__()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.double_conv = DoubleConv(in_channels, out_channels)

    def construct(self, x):
```

```python
        x = self.maxpool(x)
        x = self.double_conv(x)
        return x

class Up(nn.Cell):
    def __init__(self, in_channels, out_channels):
        super(Up, self).__init__()
        self.up = nn.Upsample(scale_factor=2, mode='bilinear',
align_corners=True)
        self.double_conv = DoubleConv(in_channels, out_channels)

    def construct(self, x1, x2):
        x1 = self.up(x1)
        x = P.Concat(1)(x1, x2)
        x = self.double_conv(x)
        return x

class Up1(Up):
    def __init__(self, in_channels, out_channels):
        super(Up1, self).__init__(in_channels, out_channels)

class Up2(Up):
    def __init__(self, in_channels, out_channels):
        super(Up2, self).__init__(in_channels, out_channels)

class Up3(Up):
    def __init__(self, in_channels, out_channels):
        super(Up3, self).__init__(in_channels, out_channels)

class Up4(Up):
    def __init__(self, in_channels, out_channels):
        super(Up4, self).__init__(in_channels, out_channels)

class OutConv(nn.Cell):
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def construct(self, x):
        x = self.conv(x)
        return x

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
```

（4）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。

将MindSpore的`nn.SoftmaxCrossEntropyWithLogits`损失函数用于U-Net，并计算输出特征图各个位置平均的损失值的代码：

```
import mindspore.nn as nn
from mindspore.ops import operations as P
from mindspore import dtype as mstype

class UNetWithSoftmaxCrossEntropy(nn.Cell):
    def __init__(self, n_channels, n_classes):
        super(UNetWithSoftmaxCrossEntropy, self).__init__()
        self.unet = UNet(n_channels, n_classes)
        self.transpose_fn = P.Transpose()
        self.reshape_fn = P.Reshape()
        self.softmax_cross_entropy_loss = nn.SoftmaxCrossEntropyWithLogits()

    def construct(self, x, label):
        logits = self.unet(x)

        # NCHW -> NHWC
        logits = self.transpose_fn(logits, (0, 2, 3, 1))
        label = self.transpose_fn(label, (0, 2, 3, 1))

        # Flatten logits and labels
        logits_flat = self.reshape_fn(logits, (-1, 2))
        label_flat = self.reshape_fn(label, (-1,1))

        # Compute softmax cross entropy loss
        loss = self.softmax_cross_entropy_loss(logits_flat, label_flat)

        # Calculate mean loss
        mean_loss = P.ReduceMean()(loss, (0,1))

        return mean_loss

n_channels = 3  # 通道数
n_classes = 2   # 类别数
```

```
net = UNetWithSoftmaxCrossEntropy(n_channels, n_classes)
```

（5）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。

计算每张验证集图像的Dice并返回验证集中Dice的均值，代码如下：

```
import mindspore.nn as nn
from mindspore import Tensor
import numpy as np
from mindspore.ops import operations as P

class DiceCoefficient(nn.Metric):
    def __init__(self):
        super(DiceCoefficient, self).__init__()
        self.clear()

    def clear(self):
        self._dice_coeff_sum = 0
        self._samples_num = 0
        self.transpose_fn = P.Transpose()

    def update(self, *inputs):
        if len(inputs) != 2:
            raise ValueError('Dice coefficient needs 2 inputs (y_pred, y), but
got {}'.format(len(inputs)))

        y_pred = self.transpose_fn(inputs[0], (0, 2, 3, 1))
        y = self.transpose_fn(inputs[1], (0, 2, 3, 1))

        y_pred = P.Softmax(axis=-1)(y_pred)

        # 计算交集
        inter = np.sum(y_pred * y)
        # 计算并集
        union = np.sum(y_pred) + np.sum(y)
        # 计算Dice系数
        single_dice_coeff = 2 * float(inter) / float(union + 1e-6)

        print("Single Dice coefficient is:", single_dice_coeff)
        self._dice_coeff_sum += single_dice_coeff
        self._samples_num += 1

    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)

# 使用示例
dice_metric = DiceCoefficient()

# 模拟输入数据
y_pred_tensor = Tensor(np.random.rand(2, 1, 256, 256).astype(np.float32))
y_tensor = Tensor(np.random.randint(0, 2, size=(2, 1, 256,
256)).astype(np.float32))
```

```
# 更新Dice系数
dice_metric.update(y_pred_tensor, y_tensor)

# 计算并打印Dice系数均值
mean_dice_coeff = dice_metric.eval()
print("Mean Dice coefficient is:", mean_dice_coeff)
```

（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。

在原有的`test_net`函数基础上，对输入图像进行预处理，然后进行模型推理并可视化预测结果，代码如下：

```
import numpy as np
from PIL import Image, ImageSequence
import matplotlib.pyplot as plt
from mindspore import Tensor
from mindspore.train.serialization import load_checkpoint, load_param_into_net
from mindspore.ops import operations as P
from mindspore.train import Model


def preprocess_image(image_path):
    # 读取一张测试集图像
    testimage = np.array([np.array(p) for p in
ImageSequence.Iterator(Image.open(image_path))])
    # 选择一张测试图像
    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)
    return testdata

def visualize_results(original_image, predicted_image):
    plt.figure(figsize=(10, 10))
    plt.subplot(2, 2, 1)
    plt.imshow(original_image, cmap='gray')
    plt.title('Original Image')
    plt.subplot(2, 2, 2)
    plt.imshow(predicted_image, cmap='gray')
    plt.title('Predicted Image')
    plt.show()

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)

    # 预处理测试图像
    testdata = preprocess_image("./data/test-volume.tif")

    # 模型推理
    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
    visualize_results(testdata[0, 0, 92:460, 92:460], pred)

# 使用示例
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)
```