

# 《计算机视觉》

## -图像分割



华为技术有限公司

版权所有 © 华为技术有限公司 2022。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

#### 商标声明

 HUAWEI 和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

#### 注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<http://e.huawei.com>

# 目录

---

<b>1 实验介绍 .....</b>	<b>2</b>
1.1 实验目的 .....	2
1.2 实验清单 .....	2
1.3 实验开发环境 .....	2
<b>2 医学图像分割实验 .....</b>	<b>3</b>
2.1 医学图像分割实验介绍 .....	3
2.2 实验总体设计 .....	4
2.3 实验详细设计与实现 .....	4
2.3.1 创建华为云 Notebook .....	4
2.3.2 下载数据集 .....	4
2.3.3 导入实验环境 .....	5
2.3.4 查看数据集 .....	5
2.3.5 使用大津阈值法进行图像分割 .....	6
2.3.6 基于神经网络的图像分割算法 .....	8
2.3.7 模型训练 .....	19
2.3.8 模型验证和测试 .....	22
2.4 实验总结 .....	25
2.5 思考题-汇总 .....	25
<b>3 基于 DeepLabv3 的语义分割实验 .....</b>	<b>26</b>
3.1 实验介绍 .....	26
3.2 实验环境要求 .....	26
3.3 背景知识 .....	26
3.3.1 网络介绍 .....	26
3.4 实验步骤 .....	28
3.5 实验总结 .....	50
<b>4 附录：ModelArts 开发环境搭建 .....</b>	<b>51</b>

# 1 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。

## 1.1 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。

## 1.2 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

## 1.3 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。

## 2 医学图像分割实验

---

### 2.1 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](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

## 2.2 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。

## 2.3 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

### 2.3.1 创建华为云 Notebook

步骤 1 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。

### 2.3.2 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=".")
```

上述下载如果无效，可以下载我们提供给大家的 Unet 数据，并将数据集放在./data/路径，如下所示：



### 2.3.3 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)
```

### 2.3.4 查看数据集

#### 步骤 1 数据集说明

官网数据集 ([http://brainiac2.mit.edu/isbi\\_challenge/home](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  $2 \times 2 \times 1.5$  微米，分辨率为  $4 \times 4 \times 50$  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。

## 步骤 2 展示数据集图像和标签

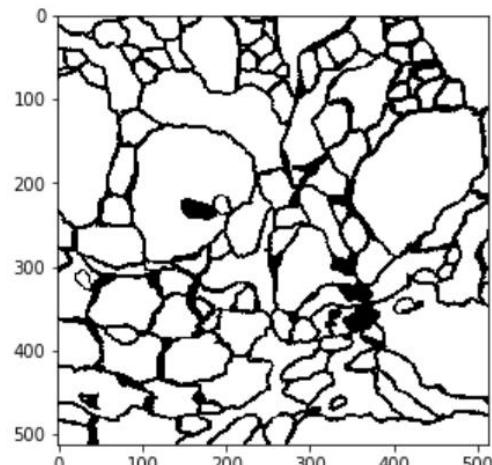
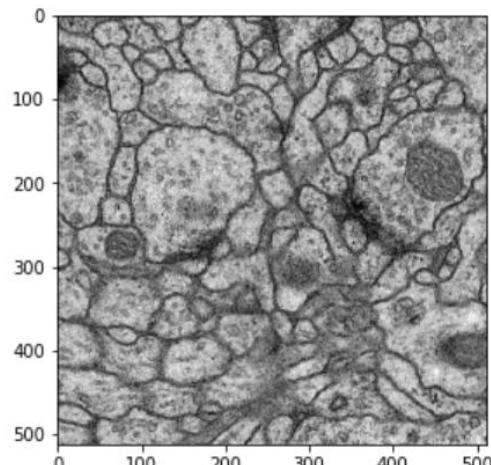
```
#打印图像和标签的形状，并展示第一张图像和标签
image= np.array([np.array(p) for p in ImageSequence.Iterator(Image.open("./data/train-volume.tif"))])

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：

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



## 2.3.5 使用大津阈值法进行图像分割

### 2.3.5.1 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：

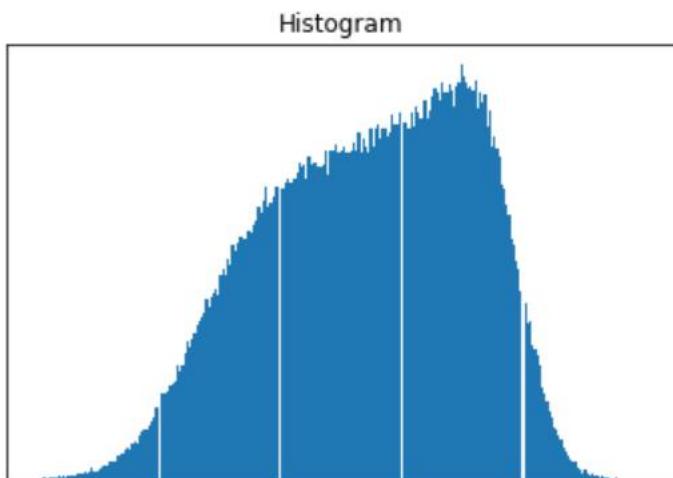


图 2-1 直方图展示

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，thresh=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：

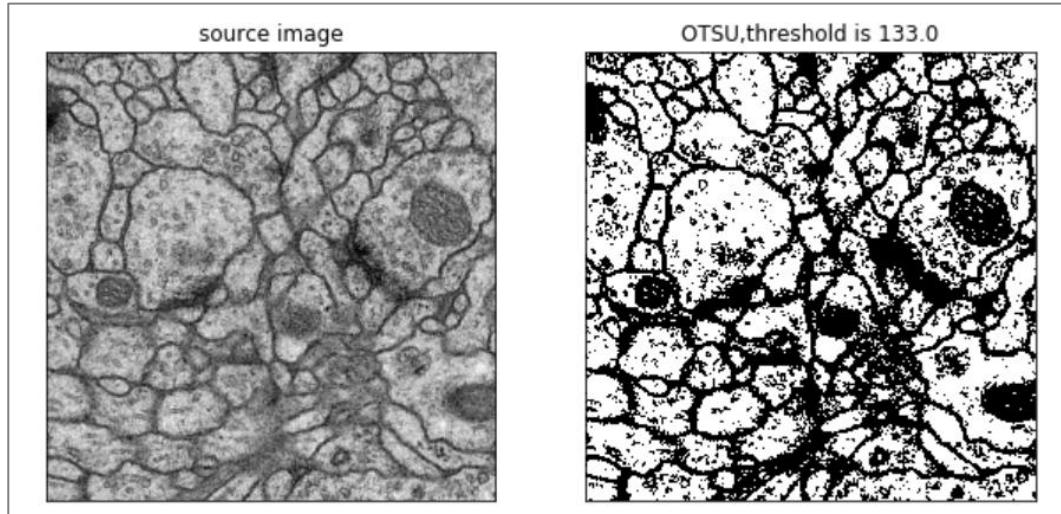


图 2-2 大津阈值法后的二值化图像

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。

### 2.3.6 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。

#### 2.3.6.1 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：

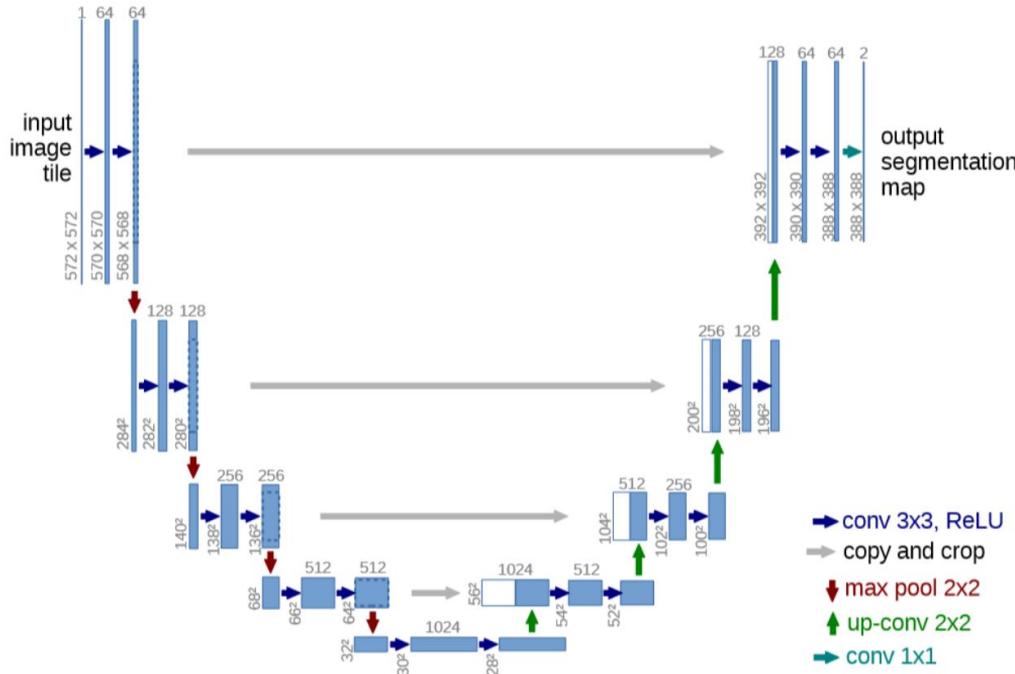


图 2-3 Unet 网络结构

#### 2.3.6.1.2 编码

第一部分网络与普通卷积网络相同，通过叠加 2 个  $3 \times 3$  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  $3 \times 3$  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。

#### 2.3.6.1.3 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  $3 \times 3$  卷积和转置卷积， $3 \times 3$  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  $1 \times 1$  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):

1	2		
3	4		
		1	1
		2	2
		3	3
		4	4
		3	3
		4	4

图 2-4 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。

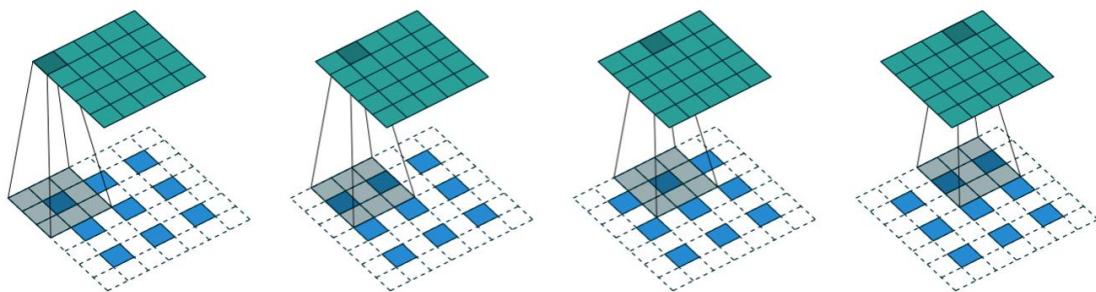


图 2-5 转置卷积展示

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](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](https://www.mindspore.cn/docs/api/zh-CN/r1.5/api_python/ops/mindspore.ops.Concat.html?highlight=concat#mindspore.ops.Concat)

#### 2.3.6.1.4 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': './',
}
```

### 2.3.6.1.5 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_o = 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_o, 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：

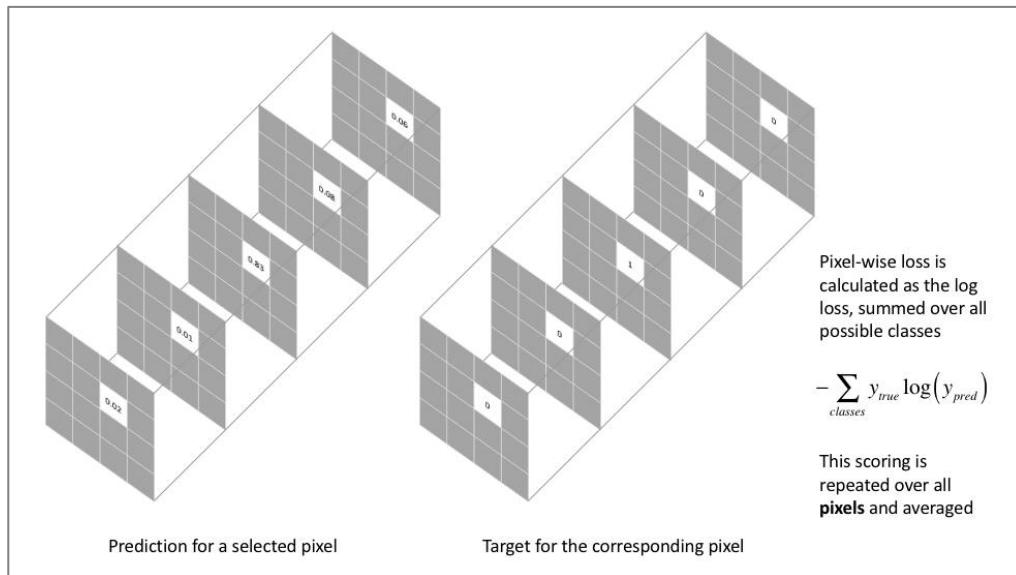
```
#根据原论文搭建 U-Net 网络
class UNet(nn.Cell):
    def __init__(self, n_channels, n_classes):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.inc = DoubleConv(n_channels, 64)
        self.down1 = Down(64, 128)
        self.down2 = 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
```

### 2.3.6.2 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)
```

### 2.3.6.3 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。

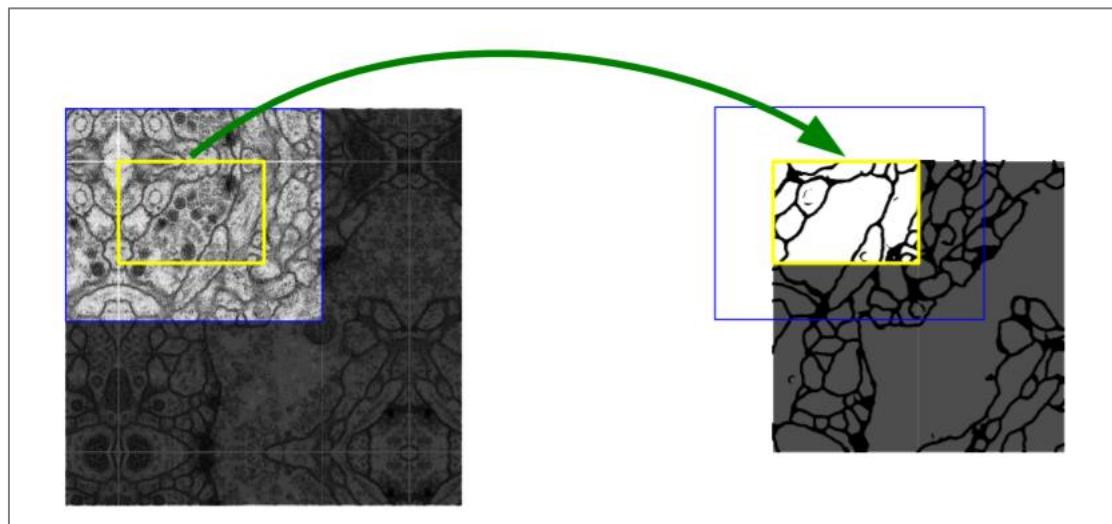


图 2-6 归一化效果展示

```
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.fliplr(img)  
        mask = np.fliplr(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.flipud(img)  
        mask = np.flipud(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
```

## 2.3.7 模型训练

### 2.3.7.1 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)
```

### 2.3.7.2 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 =====")
```

### 2.3.7.3 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 =====
```

## 2.3.8 模型验证和测试

### 2.3.8.1 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：

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

在该实验中，通过 “ $(2 \times \text{标签和预测值相同的数量}) / (2 \times \text{总像素数量})$ ” 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 coefficient 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)
```

### 2.3.8.2 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：

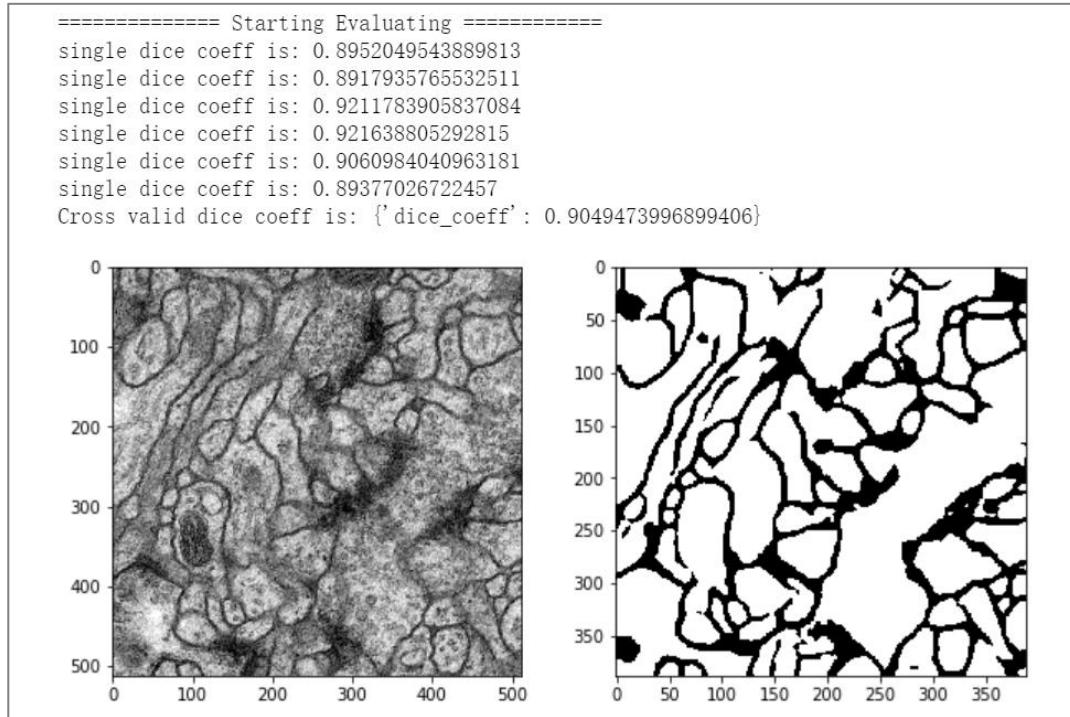


图 2-7 验证结果展示

## 2.4 实验总结

本次实验介绍对计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。

## 2.5 思考题-汇总

- 试题 1：请用 OpenCV 和 matplotlib 取出一张训练集中的图像，并绘制图像直方图。（初级）
- 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）
- 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。
- 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。
- 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。
- 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。

# 3 基于 DeepLabv3 的语义分割实验

## 3.1 实验介绍

语义分割是基于像素点级别的物体识别问题。目标是用对应的所表示的类来标记图像的每个像素。因为我们正在预测图像中的每个像素，所以此任务通常被称为密集预测。语义分割有着广泛的应用场景，包括自动驾驶，人机交互，医学图像诊断，计算摄影学和增强现实等。

本实验主要介绍使用 MindSpore 深度学习框架在 PASCAL VOC 2012 数据集上实现 Deeplabv3 网络模型的图像语义分割。图像的语义分割是将输入图像中的每个像素分配一个语义类别，以得到像素化的密集分类。

## 3.2 实验环境要求

- ModelArts 平台：Ascend: 1\*Ascend910|CPU:24 核 96GB

## 3.3 背景知识

### 3.3.1 网络介绍

- 语义分割网络介绍

一般的语义分割架构可以被认为是一个编码器-解码器网络。编码器通常是一个预训练的分类网络，像 VGG、ResNet，然后是一个解码器网络。这些架构不同的地方主要在于解码器网络。解码器的任务是将编码器学习到的可判别特征（较低分辨率）从语义上投影到像素空间（较高分辨率），以获得密集分类。语义分割不仅需要在像素级有判别能力，还需要有能将编码器在不同阶段学到的可判别特征投影到像素空间的机制。不同的架构采用不同的机制（跳跃连接、金字塔池化等）作为解码机制的一部分。

- FCN 网络介绍

自 2014 年 Long 等人首次使用全卷积神经网络 (FCN) 对自然图像进行端到端分割，语义分割才产生了大的突破。FCN 网络将当前分类网络 (AlexNet, VGG net 等) 修改为全卷积网络，通过对分割任务进行微调，将它们学习的表征转移到网络中。然后，定义了一种新的架构，将深的、粗糙的网络层的语义信息和浅的、精细的网络层的表层信息结合起来，来生成精确和详细的分割。FCN 网络结构：FCN 网络结构图如下所示。网络为卷积层和池化层的堆叠，特征图大小不断地减小，最后通过上采样的方式恢复图像分辨率。网络分为 FCN-32s、FCN-16s、FCN-8s，分别代表上采样 32、16、8 倍的网络。以 FCN-8s 为例，将 pool3 输出的特征图上采样 8 倍、pool4

输出的特征图上采样 16 倍、conv7 的特征图上采样 32 倍。最后将三者的结果拼接在一起，以使用多个尺度的特征图。综合浅层特征的位置信息和深层特征的语义信息，这样在分割时就有足够的上下文信息(context information)，同时也有目标的细节信息。

**FCN 问题：**在 CNN 中，低层的卷积中空间位置信息较为准确，但是感受野较小，只能关注附近像素点的信息。高层的卷积中感受野扩大，可以获取全局信息，但是目标空间位置信息损失严重，不能精确定位目标。而图像分割相较于分类任务需要更多的上下文信息和更精确的目标定位信息。

论文：<https://arxiv.org/abs/1411.4038>

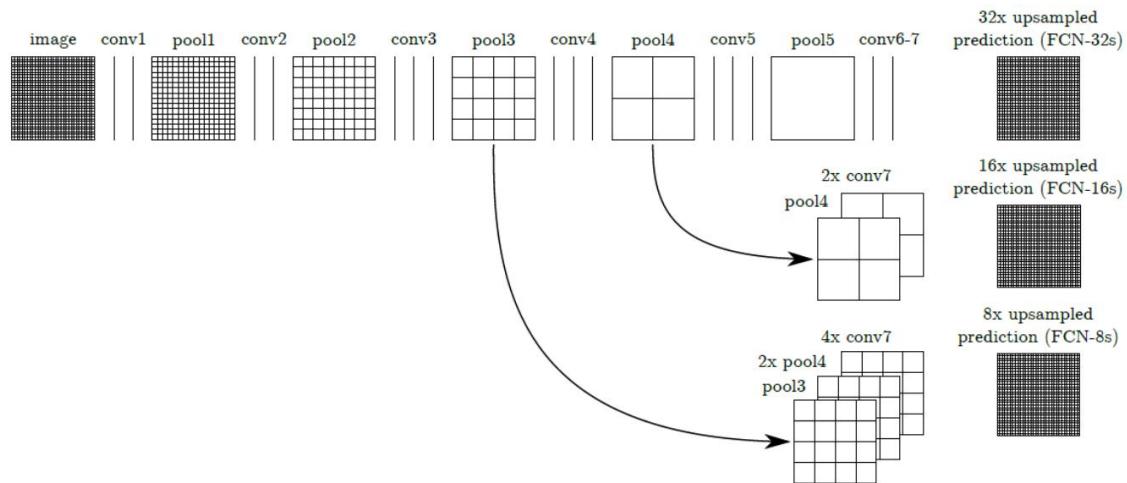


图 3-1 网络结构展示

- **Deeplab 网络介绍**

近来，深度卷积网络（DCNN）在高级视觉任务（图像分类和目标检测）中展示了优异的性能。Deeplabv1 结合 DCNN 和概率图模型来解决像素级分类任务（即语义分割）。其关键特点：

1. 提出空洞卷积（atrous convolution）。
2. 在最后两个最大池化操作中不降低特征图的分辨率，并在倒数第二个最大池化之后的卷积中使用空洞卷积。
3. 使用 CRF（条件随机场）作为后处理，恢复边界细节，达到精确定位效果。
4. 附加输入图像和前四个最大池化层的每个输出到一个两层卷积，然后拼接到主网络的最后一层，达到多尺度预测效果。

- **Deeplabv2 关键特点：**

1. 强调上采样过滤器的卷积，或“空洞卷积”，在密集预测任务中是一个强大的工具。空洞卷积允许显式地控制在深度卷积神经网络中计算的特征响应的分辨率。它还允许有效地扩大过滤器的视野，在不增加参数数量或计算量的情况下引入更大的上下文。
2. 提出了一种空洞空间金字塔池化（ASPP）的多尺度鲁棒分割方法。ASPP 使用多个采样率的过滤器和有效的视野探测传入的卷积特征层，从而在多个尺度上捕获目标和图像上下文。

3. 结合 DCNNs 方法和概率图形模型，改进了目标边界的定位。DCNNs 中常用的最大池化和下采样的组合实现了不变性，但对定位精度有一定影响。通过将 DCNN 最后一层的响应与一个全连接条件随机场(CRF)相结合来克服这个问题。

Deeplabv3 相较于之前的 Deeplab 有很大的改进，在 Pascal VOC 2012 图像语义分割基准上获得了 state-of-art 的性能（论文参考：<http://arxiv.org/abs/1706.05587>）。其关键特点：

1. 为了解决多尺度目标的分割问题，串行/并行设计了能够捕捉多尺度上下文的模块，模块中采用不同的空洞率。
2. 增强了先前提出的空洞空间金字塔池化（ASPP）模块，增加了图像级特征来编码全局上下文，使得模块可以在多尺度下探测卷积特征。并在没有 CRF 作为后处理的情况下显著提升了性能。

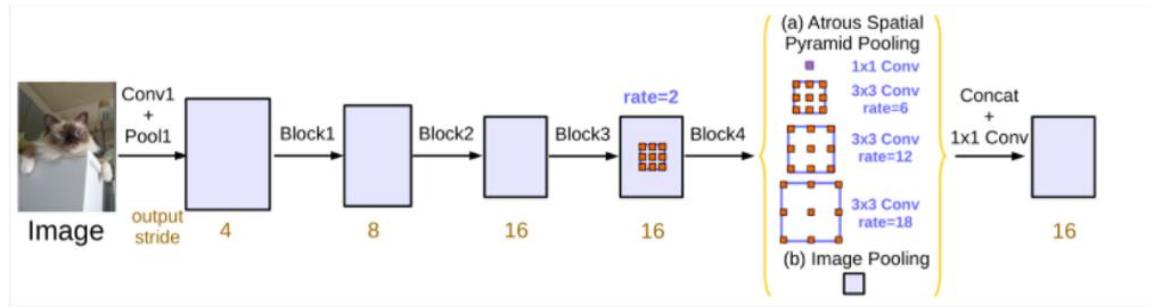


Figure 5. Parallel modules with atrous convolution (ASPP), augmented with image-level features.

图 3-2 DeepLab v3 网络结构

DeepLab v3 使用 ResNet 作为主干网络。

## 3.4 实验步骤

该实验属于 fine-tune 训练过程，即对预训练的 deeplabv3 模型进行微调。

### 步骤 1 进入 ModelArts 开发环境

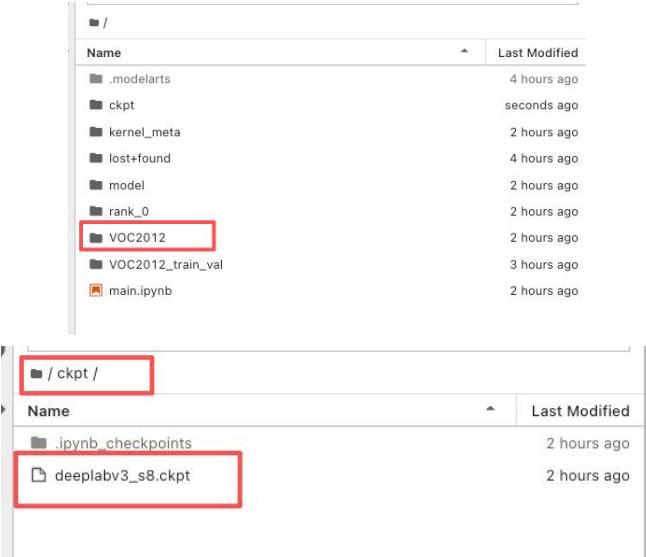
参考文末附录，创建 ModelArts 上的开发环境 Notebook 并进入。

### 步骤 2 准备数据集和预训练模型

使用 Modelarts 的 Moxing 华为对象存储服务 OBS 桶中的 VOC2012 数据集和预训练的 DeepLabv3 模型下载到 Notebook 环境中。

```
import moxing as mox
mox.file.copy_parallel(src_url="obs://ascend-professional-construction-dataset/deep-learning/deeplabv3-mindspore/VOC2012", dst_url=".//VOC2012")
mox.file.copy_parallel(src_url="obs://ascend-professional-construction-dataset/deep-learning/deeplabv3-mindspore/ckpt", dst_url=".//ckpt")
```

上述下载如果无效，可以下载我们提供给大家的 VOC2012 数据集和 deeplabv3 预训练权重，并放在相应的路径下。如果显示空间不足，需要重新创建一个 notebook，并将云硬盘设置成 10G。



并更新代码中相应的路径名称：

```
shutil.rmtree(cfg.train_dir)  
data_path = './VOC2012'  
cfg.data_file = data_path  
ckpt_path = './ckpt/deeplab_v3_s8-300_11.ckpt'  
cfg.ckpt_file = ckpt_path  
train(cfg)
```

### 步骤 3 构建数据读取相关函数

以下代码主要构建一个数据集分割类，通过这个类可以完成生成 Mindrecord 文件，预处理数据集等操作。

```
import os  
import numpy as np  
import scipy.io  
import pickle  
from PIL import Image  
import shutil  
  
import cv2  
  
from mindspore.mindrecord import FileWriter  
import mindspore.dataset as de
```

```
cv2.setNumThreads(0)

class SegDataset:
    def __init__(self,
                 image_mean,
                 image_std,
                 data_file='',
                 batch_size=32,
                 crop_size=512,
                 max_scale=2.0,
                 min_scale=0.5,
                 ignore_label=255,
                 num_classes=21,
                 num_readers=2,
                 num_parallel_calls=4,
                 shard_id=None,
                 shard_num=None):
        self.data_file = data_file
        self.batch_size = batch_size
        self.crop_size = crop_size
        self.image_mean = np.array(image_mean, dtype=np.float32)
        self.image_std = np.array(image_std, dtype=np.float32)
        self.max_scale = max_scale
        self.min_scale = min_scale
        self.ignore_label = ignore_label
        self.num_classes = num_classes
        self.num_readers = num_readers
        self.num_parallel_calls = num_parallel_calls
        self.shard_id = shard_id
        self.shard_num = shard_num
        self.voc_img_dir = os.path.join(self.data_file, 'JPEGImages')
        self.voc_anno_dir = os.path.join(self.data_file, 'SegmentationClass')
        self.voc_train_lst = os.path.join(self.data_file, 'ImageSets/Segmentation/train.txt')
        self.voc_val_lst = os.path.join(self.data_file, 'ImageSets/Segmentation/val.txt')

        self.voc_anno_gray_dir = os.path.join(self.data_file, 'SegmentationClassGray')
        self.mindrecord_save = os.path.join(self.data_file, 'VOC_mindrecord')

    assert max_scale > min_scale

def preprocess_(self, image, label):
    # bgr image
    image_out = cv2.imdecode(np.frombuffer(image, dtype=np.uint8), cv2.IMREAD_COLOR)
    label_out = cv2.imdecode(np.frombuffer(label, dtype=np.uint8), cv2.IMREAD_GRAYSCALE)

    sc = np.random.uniform(self.min_scale, self.max_scale)
```

```
new_h, new_w = int(sc * image_out.shape[0]), int(sc * image_out.shape[1])
image_out = cv2.resize(image_out, (new_w, new_h), interpolation=cv2.INTER_CUBIC)
label_out = cv2.resize(label_out, (new_w, new_h), interpolation=cv2.INTER_NEAREST)

image_out = (image_out - self.image_mean) / self.image_std
h_, w_ = max(new_h, self.crop_size), max(new_w, self.crop_size)
pad_h, pad_w = h_ - new_h, w_ - new_w
if pad_h > 0 or pad_w > 0:
    image_out = cv2.copyMakeBorder(image_out, 0, pad_h, 0, pad_w, cv2.BORDER_CONSTANT,
value=0)
    label_out = cv2.copyMakeBorder(label_out, 0, pad_h, 0, pad_w, cv2.BORDER_CONSTANT,
value=self.ignore_label)
    offset_h = np.random.randint(0, h_ - self.crop_size + 1)
    offset_w = np.random.randint(0, w_ - self.crop_size + 1)
    image_out = image_out[offset_h: offset_h + self.crop_size, offset_w: offset_w + self.crop_size, :]
    label_out = label_out[offset_h: offset_h + self.crop_size, offset_w: offset_w + self.crop_size]

if np.random.uniform(0.0, 1.0) > 0.5:
    image_out = image_out[:, ::-1, :]
    label_out = label_out[:, ::-1]

image_out = image_out.transpose((2, 0, 1))
image_out = image_out.copy()
label_out = label_out.copy()
return image_out, label_out

def get_gray_dataset(self):
    if os.path.exists(self.voc_anno_gray_dir):
        print('the gray file is already exists! ')
        return
    os.makedirs(self.voc_anno_gray_dir)

    # convert voc color png to gray png
    print('converting voc color png to gray png ...')
    for ann in os.listdir(self.voc_anno_dir):
        ann_im = Image.open(os.path.join(self.voc_anno_dir, ann))
        ann_im = Image.fromarray(np.array(ann_im))
        ann_im.save(os.path.join(self.voc_anno_gray_dir, ann))
    print('converting done')

def get_mindrecord_dataset(self, is_training, num_shards=1, shuffle=True):
    datas = []
    if is_training:
        data_lst = self.voc_train_lst
        self.mindrecord_save = os.path.join(self.mindrecord_save, 'train')
    else:
        data_lst = self.voc_val_lst
```

```
self.mindrecord_save = os.path.join(self.mindrecord_save,'eval')

if os.path.exists(self.mindrecord_save):
    #shutil.rmtree(self.mindrecord_save)
    print('mindrecord file is already exists ! ')
    self.mindrecord_save = os.path.join(self.mindrecord_save,'VOC_mindrecord')
    return

with open(data_lst) as f:
    lines = f.readlines()
if shuffle:
    np.random.shuffle(lines)

print('creating mindrecord dataset...')
os.makedirs(self.mindrecord_save)
self.mindrecord_save = os.path.join(self.mindrecord_save,'VOC_mindrecord')
print('number of samples:', len(lines))
seg_schema = {"file_name": {"type": "string"}, "label": {"type": "bytes"}, "data": {"type": "bytes"}}
writer = FileWriter(file_name=self.mindrecord_save, shard_num=num_shards)
writer.add_schema(seg_schema, "seg_schema")
cnt = 0
for l in lines:
    id_ = l.strip()
    img_path = os.path.join(self.voc_img_dir, id_ + '.jpg')
    label_path = os.path.join(self.voc_anno_gray_dir, id_ + '.png')

    sample_ = {"file_name": img_path.split('/')[-1]}
    with open(img_path, 'rb') as f:
        sample_['data'] = f.read()
    with open(label_path, 'rb') as f:
        sample_['label'] = f.read()
    datas.append(sample_)
    cnt += 1
    if cnt % 1000 == 0:
        writer.write_raw_data(datas)
        print('number of samples written:', cnt)
        datas = []
if datas:
    writer.write_raw_data(datas)
writer.commit()
print('number of samples written:', cnt)
print('Create Mindrecord Done')

def get_dataset(self, repeat=1):
    data_set = de.MindDataset(dataset_files=self.mindrecord_save, columns_list=["data", "label"],
                             shuffle=True, num_parallel_workers=self.num_readers,
```

```
        num_shards=self.shard_num, shard_id=self.shard_id)
transforms_list = self.preprocess_
data_set = data_set.map(operations=transforms_list, input_columns=["data", "label"],
                       output_columns=["data", "label"],
                       num_parallel_workers=self.num_parallel_calls)
data_set = data_set.shuffle(buffer_size=self.batch_size * 10)
data_set = data_set.batch(self.batch_size, drop_remainder=True)
data_set = data_set.repeat(repeat)
return data_set
```

## 步骤 4 构建网络

以下代码主要完成 Deeplabv3 主体网络的构建，通过定义多个 resnet cell 结构组成整体的 Deeplabv3 网络。最终的网络是以类的方式进行定义，通过实例化即可创建对应的网络对象。

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


def conv1x1(in_planes, out_planes, stride=1):
    return nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride, weight_init='xavier_uniform')


def conv3x3(in_planes, out_planes, stride=1, dilation=1, padding=1):
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride, pad_mode='pad', padding=padding,
                   dilation=dilation, weight_init='xavier_uniform')


class Resnet(nn.Cell):
    def __init__(self, block, block_num, output_stride, use_batch_statistics=True):
        super(Resnet, self).__init__()
        self.inplanes = 64
        self.conv1 = nn.Conv2d(3, self.inplanes, kernel_size=7, stride=2, pad_mode='pad', padding=3,
                            weight_init='xavier_uniform')
        self.bn1 = nn.BatchNorm2d(self.inplanes, use_batch_statistics=use_batch_statistics)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, pad_mode='same')
        self.layer1 = self._make_layer(block, 64, block_num[0], use_batch_statistics=use_batch_statistics)
        self.layer2 = self._make_layer(block, 128, block_num[1], stride=2,
                                      use_batch_statistics=use_batch_statistics)

        if output_stride == 16:
            self.layer3 = self._make_layer(block, 256, block_num[2], stride=2,
                                          use_batch_statistics=use_batch_statistics)
            self.layer4 = self._make_layer(block, 512, block_num[3], stride=1, base_dilation=2, grids=[1, 2, 4],
                                          use_batch_statistics=use_batch_statistics)
        elif output_stride == 8:
            self.layer3 = self._make_layer(block, 256, block_num[2], stride=1, base_dilation=2,
                                          use_batch_statistics=use_batch_statistics)
```

```
        self.layer4 = self._make_layer(block, 512, block_num[3], stride=1, base_dilation=4, grids=[1, 2, 4],
                                      use_batch_statistics=use_batch_statistics)

    def _make_layer(self, block, planes, blocks, stride=1, base_dilation=1, grids=None, use_batch_statistics=True):
        if stride != 1 or self.inplanes != planes * block.expansion:
            downsample = nn.SequentialCell([
                conv1x1(self.inplanes, planes * block.expansion, stride),
                nn.BatchNorm2d(planes * block.expansion, use_batch_statistics=use_batch_statistics)
            ])

        if grids is None:
            grids = [1] * blocks

        layers = [
            block(self.inplanes, planes, stride, downsample, dilation=base_dilation * grids[0],
                  use_batch_statistics=use_batch_statistics)
        ]
        self.inplanes = planes * block.expansion
        for i in range(1, blocks):
            layers.append(
                block(self.inplanes, planes, dilation=base_dilation * grids[i],
                      use_batch_statistics=use_batch_statistics))

        return nn.SequentialCell(layers)

    def construct(self, x):
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.maxpool(out)

        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        return out

class Bottleneck(nn.Cell):
    expansion = 4

    def __init__(self, inplanes, planes, stride=1, downsample=None, dilation=1, use_batch_statistics=True):
        super(Bottleneck, self).__init__()
        self.conv1 = conv1x1(inplanes, planes)
        self.bn1 = nn.BatchNorm2d(planes, use_batch_statistics=use_batch_statistics)

        self.conv2 = conv3x3(planes, planes, stride, dilation, dilation)
```

```
self.bn2 = nn.BatchNorm2d(planes, use_batch_statistics=use_batch_statistics)

self.conv3 = conv1x1(planes, planes * self.expansion)
self.bn3 = nn.BatchNorm2d(planes * self.expansion, use_batch_statistics=use_batch_statistics)

self.relu = nn.ReLU()
self.downsample = downsample

self.add = P.TensorAdd()

def construct(self, x):
    identity = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    out = self.conv2(out)
    out = self.bn2(out)
    out = self.relu(out)

    out = self.conv3(out)
    out = self.bn3(out)

    if self.downsample is not None:
        identity = self.downsample(x)

        out = self.add(out, identity)
        out = self.relu(out)
    return out

class ASPP(nn.Cell):
    def __init__(self, atrous_rates, phase='train', in_channels=2048, num_classes=21,
                 use_batch_statistics=True):
        super(ASPP, self).__init__()
        self.phase = phase
        out_channels = 256
        self.aspp1 = ASPPConv(in_channels, out_channels, atrous_rates[0],
                             use_batch_statistics=use_batch_statistics)
        self.aspp2 = ASPPConv(in_channels, out_channels, atrous_rates[1],
                             use_batch_statistics=use_batch_statistics)
        self.aspp3 = ASPPConv(in_channels, out_channels, atrous_rates[2],
                             use_batch_statistics=use_batch_statistics)
        self.aspp4 = ASPPConv(in_channels, out_channels, atrous_rates[3],
                             use_batch_statistics=use_batch_statistics)
        self.aspp_pooling = ASPPPooling(in_channels, out_channels)
        self.conv1 = nn.Conv2d(out_channels * (len(atrous_rates) + 1), out_channels, kernel_size=1,
```

```
        weight_init='xavier_uniform')
    self.bn1 = nn.BatchNorm2d(out_channels, use_batch_statistics=use_batch_statistics)
    self.relu = nn.ReLU()
    self.conv2 = nn.Conv2d(out_channels, num_classes, kernel_size=1, weight_init='xavier_uniform',
has_bias=True)
    self.concat = P.Concat(axis=1)
    self.drop = nn.Dropout(0.3)

def construct(self, x):
    x1 = self.aspp1(x)
    x2 = self.aspp2(x)
    x3 = self.aspp3(x)
    x4 = self.aspp4(x)
    x5 = self.aspp_pooling(x)

    x = self.concat((x1, x2))
    x = self.concat((x, x3))
    x = self.concat((x, x4))
    x = self.concat((x, x5))

    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    if self.phase == 'train':
        x = self.drop(x)
    x = self.conv2(x)
    return x

class ASPPPooling(nn.Cell):
    def __init__(self, in_channels, out_channels, use_batch_statistics=True):
        super(ASPPPooling, self).__init__()
        self.conv = nn.SequentialCell([
            nn.Conv2d(in_channels, out_channels, kernel_size=1, weight_init='xavier_uniform'),
            nn.BatchNorm2d(out_channels, use_batch_statistics=use_batch_statistics),
            nn.ReLU()
        ])
        self.shape = P.Shape()

    def construct(self, x):
        size = self.shape(x)
        out = nn.AvgPool2d(size[2])(x)
        out = self.conv(out)
        out = P.ResizeNearestNeighbor((size[2], size[3]), True)(out)
        return out

class ASPPConv(nn.Cell):
```

```
def __init__(self, in_channels, out_channels, atrous_rate=1, use_batch_statistics=True):
    super(ASPPConv, self).__init__()
    if atrous_rate == 1:
        conv = nn.Conv2d(in_channels, out_channels, kernel_size=1, has_bias=False,
weight_init='xavier_uniform')
    else:
        conv = nn.Conv2d(in_channels, out_channels, kernel_size=3, pad_mode='pad', padding=atrous_rate,
dilation=atrous_rate, weight_init='xavier_uniform')
    bn = nn.BatchNorm2d(out_channels, use_batch_statistics=use_batch_statistics)
    relu = nn.ReLU()
    self.aspp_conv = nn.SequentialCell([conv, bn, relu])

def construct(self, x):
    out = self.aspp_conv(x)
    return out

class DeepLabV3(nn.Cell):
    def __init__(self, phase='train', num_classes=21, output_stride=16, freeze_bn=False):
        super(DeepLabV3, self).__init__()
        use_batch_statistics = not freeze_bn
        self.resnet = Resnet(Bottleneck, [3, 4, 23, 3], output_stride=output_stride,
use_batch_statistics=use_batch_statistics)
        self.aspp = ASPP([1, 6, 12, 18], phase, 2048, num_classes,
use_batch_statistics=use_batch_statistics)
        self.shape = P.Shape()

    def construct(self, x):
        size = self.shape(x)
        out = self.resnet(x)
        out = self.aspp(out)
        out = P.ResizeBilinear((size[2], size[3]), True)(out)
        return out
```

## 步骤 5 定义不同的学习率

以下代码定义不同的学习率函数。

```
#定义不同的学习率
def cosine_lr(base_lr, decay_steps, total_steps):
    for i in range(total_steps):
        step_ = min(i, decay_steps)
        yield base_lr * 0.5 * (1 + np.cos(np.pi * step_ / decay_steps))

def poly_lr(base_lr, decay_steps, total_steps, end_lr=0.0001, power=0.9):
    for i in range(total_steps):
        step_ = min(i, decay_steps)
        yield (base_lr - end_lr) * ((1.0 - step_ / decay_steps) ** power) + end_lr
```

```
def exponential_lr(base_lr, decay_steps, decay_rate, total_steps, staircase=False):
    for i in range(total_steps):
        if staircase:
            power_ = i // decay_steps
        else:
            power_ = float(i) / decay_steps
        yield base_lr * (decay_rate ** power_)
```

## 步骤 6 定义损失函数

以下代码定义了损失函数。

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


class SoftmaxCrossEntropyLoss(nn.Cell):
    def __init__(self, num_cls=21, ignore_label=255):
        super(SoftmaxCrossEntropyLoss, self).__init__()
        self.one_hot = P.OneHot(axis=-1)
        self.on_value = Tensor(1.0, mstype.float32)
        self.off_value = Tensor(0.0, mstype.float32)
        self.cast = P.Cast()
        self.ce = nn.SoftmaxCrossEntropyWithLogits()
        self.not_equal = P.NotEqual()
        self.num_cls = num_cls
        self.ignore_label = ignore_label
        self.mul = P.Mul()
        self.sum = P.ReduceSum(False)
        self.div = P.RealDiv()
        self.transpose = P.Transpose()
        self.reshape = P.Reshape()

    def construct(self, logits, labels):
        labels_int = self.cast(labels, mstype.int32)
        labels_int = self.reshape(labels_int, (-1,))
        logits_ = self.transpose(logits, (0, 2, 3, 1))
        logits_ = self.reshape(logits_, (-1, self.num_cls))
        weights = self.not_equal(labels_int, self.ignore_label)
        weights = self.cast(weights, mstype.float32)
        one_hot_labels = self.one_hot(labels_int, self.num_cls, self.on_value, self.off_value)
        loss = self.ce(logits_, one_hot_labels)
        loss = self.mul(weights, loss)
        loss = self.div(self.sum(loss), self.sum(weights))
        return loss
```



## 步骤 7 构建训练网络的函数

```
"""train deeplabv3."""
import os
import sys
sys.path.insert(0,'./deeplabv3/deeplabv3_2/')      # your code path
from easydict import EasyDict as edict
import shutil

# import moxing as mox
from mindspore import context
from mindspore.train.model import ParallelMode, Model
import mindspore.nn as nn
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig
from mindspore.train.serialization import load_checkpoint, load_param_into_net
from mindspore.communication.management import init, get_rank, get_group_size
from mindspore.train.callback import LossMonitor, TimeMonitor
from mindspore.train.loss_scale_manager import FixedLossScaleManager
from mindspore.common import set_seed

set_seed(1)
context.set_context(mode=context.GRAPH_MODE, enable_auto_mixed_precision=True, save_graphs=False,
                    device_target="Ascend")

class BuildTrainNetwork(nn.Cell):
    def __init__(self, network, criterion):
        super(BuildTrainNetwork, self).__init__()
        self.network = network
        self.criterion = criterion

    def construct(self, input_data, label):
        output = self.network(input_data)
        net_loss = self.criterion(output, label)
        return net_loss

def train(args):
    # init multicards training
    if args.is_distributed:
        init()
        args.rank = get_rank()
        args.group_size = get_group_size()

        parallel_mode = ParallelMode.DATA_PARALLEL
        context.set_auto_parallel_context(parallel_mode=parallel_mode, gradients_mean=True,
device_num=args.group_size)
```

```
# dataset
dataset = SegDataset(image_mean=args.image_mean,
                     image_std=args.image_std,
                     data_file=args.data_file,
                     batch_size=args.batch_size,
                     crop_size=args.crop_size,
                     max_scale=args.max_scale,
                     min_scale=args.min_scale,
                     ignore_label=args.ignore_label,
                     num_classes=args.num_classes,
                     num_readers=2,
                     num_parallel_calls=4,
                     shard_id=args.rank,
                     shard_num=args.group_size)
dataset.get_gray_dataset()
dataset.get_mindrecord_dataset(is_training=True)
dataset = dataset.get_dataset(repeat=1)

# network
if args.model == 'deeplab_v3_s16':
    network = DeepLabV3('train', args.num_classes, 16, args.freeze_bn)
elif args.model == 'deeplab_v3_s8':
    network = DeepLabV3('train', args.num_classes, 8, args.freeze_bn)
else:
    raise NotImplementedError('model [{:s}] not recognized'.format(args.model))

# loss
loss_ = SoftmaxCrossEntropyLoss(args.num_classes, args.ignore_label)
loss_.add_flags_recursive(fp32=True)
train_net = BuildTrainNetwork(network, loss_)

# load pretrained model
param_dict = load_checkpoint(args.ckpt_file)
load_param_into_net(train_net, param_dict)

# optimizer
iters_per_epoch = dataset.get_dataset_size()
total_train_steps = iters_per_epoch * args.train_epochs
if args.lr_type == 'cos':
    lr_iter = cosine_lr(args.base_lr, total_train_steps, total_train_steps)
elif args.lr_type == 'poly':
    lr_iter = poly_lr(args.base_lr, total_train_steps, total_train_steps, end_lr=0.0, power=0.9)
elif args.lr_type == 'exp':
    lr_iter = exponential_lr(args.base_lr, args.lr_decay_step, args.lr_decay_rate,
                            total_train_steps, staircase=True)
else:
```

```
raise ValueError('unknown learning rate type')

opt = nn.Momentum(params=train_net.trainable_params(), learning_rate=lr_iter, momentum=0.9,
weight_decay=0.0001,
                loss_scale=args.loss_scale)

# loss scale
manager_loss_scale = FixedLossScaleManager(args.loss_scale, drop_overflow_update=False)
model = Model(train_net, optimizer=opt, amp_level="O3", loss_scale_manager=manager_loss_scale)

# callback for saving ckpts
time_cb = TimeMonitor(data_size=iters_per_epoch)
loss_cb = LossMonitor()
cbs = [time_cb, loss_cb]

if args.rank == 0:
    config_ck = CheckpointConfig(save_checkpoint_steps=iters_per_epoch,
                                keep_checkpoint_max=args.keep_checkpoint_max)
    ckpoint_cb = ModelCheckpoint(prefix=args.model, directory=args.train_dir, config=config_ck)
    cbs.append(ckpoint_cb)

model.train(args.train_epochs, dataset, callbacks=cbs, dataset_sink_mode=True)
```

## 步骤 8 设定相关参数并训练网络

```
#设定相关参数
cfg = edict({
    "batch_size": 16,
    "crop_size": 513,
    "image_mean": [103.53, 116.28, 123.675],
    "image_std": [57.375, 57.120, 58.395],
    "min_scale": 0.5,
    "max_scale": 2.0,
    "ignore_label": 255,
    "num_classes": 21,
    "train_epochs": 3,

    "lr_type": 'cos',
    "base_lr": 0.0,
    "lr_decay_step": 3*91,
    "lr_decay_rate": 0.1,
    "loss_scale": 2048,
    "model": 'deeplab_v3_s8',
    'rank': 0,
    'group_size': 1,
    'keep_checkpoint_max': 1,
```

```
'train_dir': 'model',  
  
'is_distributed':False,  
'freeze_bn':True  
})  
  
if os.path.exists(cfg.train_dir):  
    shutil.rmtree(cfg.train_dir)  
  
data_path = './VOC2012'  
  
cfg.data_file = data_path  
  
ckpt_path = './ckpt/deeplab_v3_s8-300_11.ckpt'  
  
cfg.ckpt_file = ckpt_path  
  
train(cfg)
```

Fine-tune 好的模型将放在“./model”文件夹中， 默认的模型名为“deeplab\_v3\_s8-3\_91.ckpt”。

输出：

```
converting voc color png to gray png ...  
converting done  
creating mindrecord dataset...  
number of samples: 1464  
number of samples written: 1000  
number of samples written: 1464  
Create Mindrecord Done  
  
epoch: 1 step: 91, loss is 0.0029648119  
epoch time: 220090.707 ms, per step time: 2418.579 ms  
epoch: 2 step: 91, loss is 0.004314194  
epoch time: 47422.051 ms, per step time: 521.121 ms  
epoch: 3 step: 91, loss is 0.0039475723  
epoch time: 47430.172 ms, per step time: 521.211 ms
```

## 步骤 9 验证网络

以下代码构建验证网络。

```
"""eval deeplabv3."""  
import os  
import sys  
sys.path.insert(0,'./deeplabv3/deeplabv3_2/')      # your code path  
from easydict import EasyDict as edict  
from PIL import Image  
import PIL
```

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.colors as colors
import numpy as np
import cv2
# import moxing as mox

from mindspore import Tensor
import mindspore.common.dtype as mstype
import mindspore.nn as nn
from mindspore import context
from mindspore.train.serialization import load_checkpoint, load_param_into_net

context.set_context(mode=context.GRAPH_MODE, device_target="Ascend", save_graphs=False)

def cal_hist(a, b, n):
    k = (a >= 0) & (a < n)
    return np.bincount(n * a[k].astype(np.int32) + b[k], minlength=n ** 2).reshape(n, n)

def resize_long(img, long_size=513):
    h, w, _ = img.shape
    if h > w:
        new_h = long_size
        new_w = int(1.0 * long_size * w / h)
    else:
        new_w = long_size
        new_h = int(1.0 * long_size * h / w)
    imo = cv2.resize(img, (new_w, new_h))
    return imo

class BuildEvalNetwork(nn.Cell):
    def __init__(self, network):
        super(BuildEvalNetwork, self).__init__()
        self.network = network
        self.softmax = nn.Softmax(axis=1)

    def construct(self, input_data):
        output = self.network(input_data)
        output = self.softmax(output)
        return output

def pre_process(args, img_, crop_size=513):
```

```
# resize
img_ = resize_long(img_, crop_size)
resize_h, resize_w, _ = img_.shape

# mean, std
image_mean = np.array(args.image_mean)
image_std = np.array(args.image_std)
img_ = (img_ - image_mean) / image_std

# pad to crop_size
pad_h = crop_size - img_.shape[0]
pad_w = crop_size - img_.shape[1]
if pad_h > 0 or pad_w > 0:
    img_ = cv2.copyMakeBorder(img_, 0, pad_h, 0, pad_w, cv2.BORDER_CONSTANT, value=0)

# hwc to chw
img_ = img_.transpose((2, 0, 1))
return img_, resize_h, resize_w

def eval_batch(args, eval_net, img_lst, crop_size=513, flip=True):
    result_lst = []
    batch_size = len(img_lst)
    batch_img = np.zeros((args.batch_size, 3, crop_size, crop_size), dtype=np.float32)
    resize_hw = []
    for l in range(batch_size):
        img_ = img_lst[l]
        img_, resize_h, resize_w = pre_process(args, img_, crop_size)
        batch_img[l] = img_
        resize_hw.append([resize_h, resize_w])

    batch_img = np.ascontiguousarray(batch_img)
    net_out = eval_net(Tensor(batch_img, mstype.float32))
    net_out = net_out.asnumpy()

    if flip:
        batch_img = batch_img[:, :, :, ::-1]
        net_out_flip = eval_net(Tensor(batch_img, mstype.float32))
        net_out += net_out_flip.asnumpy()[:, :, :, ::-1]

    for bs in range(batch_size):
        probs_ = net_out[bs][:, :resize_hw[bs][0], :resize_hw[bs][1]].transpose((1, 2, 0))
        ori_h, ori_w = img_lst[bs].shape[0], img_lst[bs].shape[1]
        probs_ = cv2.resize(probs_, (ori_w, ori_h))
        result_lst.append(probs_)

    return result_lst
```

```
def eval_batch_scales(args, eval_net, img_lst, scales,
                     base_crop_size=513, flip=True):
    sizes_ = [int((base_crop_size - 1) * sc) + 1 for sc in scales]
    probs_lst = eval_batch(args, eval_net, img_lst, crop_size=sizes_[0], flip=flip)
    #print(sizes_)
    for crop_size_ in sizes_[1:]:
        probs_lst_tmp = eval_batch(args, eval_net, img_lst, crop_size=crop_size_, flip=flip)
        for pl_ in enumerate(probs_lst):
            probs_lst[pl] += probs_lst_tmp[pl]

    result_msk = []
    for i in probs_lst:
        result_msk.append(i.argmax(axis=2))
    return result_msk

# The color source: print(list(colors.cnames.keys()))
#print(list(colors.cnames.keys()))
num_class = {0: 'background', 1: 'aeroplane', 2: 'bicycle', 3: 'bird', 4: 'boat', 5: 'bottle', 6: 'bus', 7: 'car', 8: 'cat',
             9: 'chair', 10: 'cow', 11: 'diningtable', 12: 'dog', 13: 'horse', 14: 'motorbike', 15: 'person', 16: 'pottedplant',
             17: 'sheep', 18: 'sofa', 19: 'train', 20: 'tvmonitor', 21: 'edge'}

num_color = {0:'aliceblue', 1:'grey', 2:'red', 3:'green', 4:'darkorange', 5:'lime', 6:'bisque',
             7:'black', 8:'blanchedalmond', 9:'blue', 10:'blueviolet', 11:'brown', 12:'burlywood', 13:'cadetblue',
             14:'darkorange', 15:'tan', 16:'darkviolet', 17:'cornflowerblue', 18:'yellow', 19:'crimson', 20:'darkcyan'}

color_dic = [num_color[k] for k in sorted(num_color.keys())]
bounds = list(range(21))
cmap = mpl.colors.ListedColormap(color_dic)
norm = mpl.colors.BoundaryNorm(bounds, cmap.N)

def num_to_ClassAndColor(num_list):
    color_ = []
    class_ = []
    for num in num_list:
        color_.append(num_class[num])
        class_.append(num_color[num])
    return color_,class_

def net_eval(args):
    # network
    if args.model == 'deeplab_v3_s16':
        network = DeepLabV3('eval', args.num_classes, 16, args.freeze_bn)
    elif args.model == 'deeplab_v3_s8':
        network = DeepLabV3('eval', args.num_classes, 8, args.freeze_bn)
    else:
```

```
raise NotImplementedError('model [{:s}] not recognized'.format(args.model))

eval_net = BuildEvalNetwork(network)

# load model
param_dict = load_checkpoint(args.ckpt_file)
load_param_into_net(eval_net, param_dict)
eval_net.set_train(False)

# data list
with open(args.data_lst) as f:
    img_lst = f.readlines()

# evaluate
hist = np.zeros((args.num_classes, args.num_classes))
batch_img_lst = []
batch_msk_lst = []
bi = 0
image_num = 0
for i, line in enumerate(img_lst):
    id_ = line.strip()
    img_path = os.path.join(cfg.voc_img_dir, id_ + '.jpg')
    msk_path = os.path.join(cfg.voc_anno_gray_dir, id_ + '.png')

    img_ = cv2.imread(img_path)
    msk_ = cv2.imread(msk_path, cv2.IMREAD_GRAYSCALE)
    batch_img_lst.append(img_)
    batch_msk_lst.append(msk_)
    if args.if_png:
        batch_res = eval_batch_scales(args, eval_net, batch_img_lst, scales=args.scales,
                                      base_crop_size=args.crop_size, flip=args.flip)

        height, weight = batch_res[0].shape
        batch_msk_lst[0][batch_msk_lst[0]==args.ignore_label] = 0

        plt.figure(figsize=(3 * weight/1024*10, 2 * height/1024*10))
        plt.subplot(1,3,1)
        image = Image.open(img_path)
        plt.imshow(image)

        plt.subplot(1,3,2)
        plt.imshow(image)
        plt.imshow(batch_res[0],alpha=0.8,interpolation='none', cmap=cmap, norm=norm)

        plt.subplot(1,3,3)
        plt.imshow(image)
        plt.imshow(batch_msk_lst[0],alpha=0.8,interpolation='none', cmap=cmap, norm=norm)
```

```
plt.show()

prediction_num = np.unique(batch_res[o])
real_num = np.unique(batch_msk_lst[o])

prediction_color,prediction_class = num_to_ClassAndColor(prediction_num)
print('prediction num:',prediction_num)
print('prediction color:',prediction_color)
print('prediction class:',prediction_class)
real_color,real_class = num_to_ClassAndColor(real_num)
print('groundtruth num:',real_num)
print('groundtruth color:',real_color)
print('groundtruth class:',real_class)
batch_img_lst = []
batch_msk_lst = []
if i < args.num_png-1:
    continue
else:
    return

bi += 1
if bi == args.batch_size:
    batch_res = eval_batch_scales(args, eval_net, batch_img_lst, scales=args.scales,
                                  base_crop_size=args.crop_size, flip=args.flip)
    for mi in range(args.batch_size):
        hist += cal_hist(batch_msk_lst[mi].flatten(), batch_res[mi].flatten(), args.num_classes)

    bi = o
    batch_img_lst = []
    batch_msk_lst = []
    if (i+1)%100 == o:
        print('processed {} images'.format(i+1))
    image_num = i

if bi > o:
    batch_res = eval_batch_scales(args, eval_net, batch_img_lst, scales=args.scales,
                                  base_crop_size=args.crop_size, flip=args.flip)
    for mi in range(bi):
        hist += cal_hist(batch_msk_lst[mi].flatten(), batch_res[mi].flatten(), args.num_classes)
    if (i+1) % 100 == o:
        print('processed {} images'.format(image_num + 1))

iu = np.diag(hist) / (hist.sum(1) + hist.sum(0) - np.diag(hist))
print('mean IoU', np.nanmean(iu))
```

以下代码是从数据集读取图片进行网络验证，不显示图片。

```
# test 1
cfg = edict({}
```

```
"batch_size": 1,
"crop_size": 513,
"image_mean": [103.53, 116.28, 123.675],
"image_std": [57.375, 57.120, 58.395],
"scales": [1.0],           # [0.5,0.75,1.0,1.25,1.75]
'flip': True,

'ignore_label': 255,
'num_classes': 21,

'model': 'deeplab_v3_s8',
'freeze_bn': True,

'if_png': False,
'num_png': 10
})

data_path = './VOC2012'
# if not os.path.exists(data_path):
#     mox.file.copy_parallel(src_url="s3://share-course/dataset/voc2012_raw/", dst_url=data_path)
cfg.data_file = data_path

# dataset
dataset = SegDataset(image_mean=cfg.image_mean,
                      image_std=cfg.image_std,
                      data_file=cfg.data_file)
dataset.get_gray_dataset()
cfg.data_lst = os.path.join(cfg.data_file, 'ImageSets/Segmentation/val.txt')
cfg.voc_img_dir = os.path.join(cfg.data_file, 'JPEGImages')
cfg.voc_anno_gray_dir = os.path.join(cfg.data_file, 'SegmentationClassGray')

ckpt_path = './model'
# if not os.path.exists(ckpt_path):
#     mox.file.copy_parallel(src_url="s3://yyq-3/DATA/code/deeplabv3/model", dst_url=ckpt_path)    #if yours
model had saved
cfg.ckpt_file = os.path.join(ckpt_path, 'deeplab_v3_s8-3_91.ckpt')
print('loading checkpoing:', cfg.ckpt_file)

net_eval(cfg)
```

输出：

```
the gray file is already exists!
loading checkpoing: ./model/deeplab_v3_s8-3_91.ckpt

processed 100 images
processed 200 images
processed 300 images
```

```
processed 400 images
processed 500 images
processed 600 images
processed 700 images
processed 800 images
processed 900 images
processed 1000 images
processed 1100 images
processed 1200 images
processed 1300 images
processed 1400 images
mean IoU 0.7759366796455764
```

以下代码进行网络验证，显示最终图片和结果。

```
# test 2
cfg = edict({
    "batch_size": 1,
    "crop_size": 513,
    "image_mean": [103.53, 116.28, 123.675],
    "image_std": [57.375, 57.120, 58.395],
    "scales": [1.0],           # [0.5,0.75,1.0,1.25,1.75]
    'flip': True,
    'ignore_label': 255,
    'num_classes': 21,
    'model': 'deeplab_v3_s8',
    'freeze_bn': True,
    'if_png': True,
    'num_png': 3
})

# import moxing as mox
data_path = './VOC2012'
# if not os.path.exists(data_path):
#     mox.file.copy_parallel(src_url="s3://share-course/dataset/voc2012_raw/", dst_url=data_path)
cfg.data_file = data_path

# dataset
dataset = SegDataset(image_mean=cfg.image_mean,
                      image_std=cfg.image_std,
                      data_file=cfg.data_file)

dataset.get_gray_dataset()
cfg.data_lst = os.path.join(cfg.data_file, 'ImageSets/Segmentation/val.txt')
cfg.voc_img_dir = os.path.join(cfg.data_file, 'JPEGImages')
cfg.voc_anno_gray_dir = os.path.join(cfg.data_file, 'SegmentationClassGray')
```

```
ckpt_path = './model'  
# if not os.path.exists(ckpt_path):  
#     mox.file.copy_parallel(src_url="s3://yyq-3/DATA/code/deeplabv3/model", dst_url=ckpt_path)      #if yours  
model had saved  
cfg.ckpt_file = os.path.join(ckpt_path,'deeplab_v3_s8-3_91.ckpt')  
print('loading checkpoing:',cfg.ckpt_file)  
  
net_eval(cfg)
```

输出：

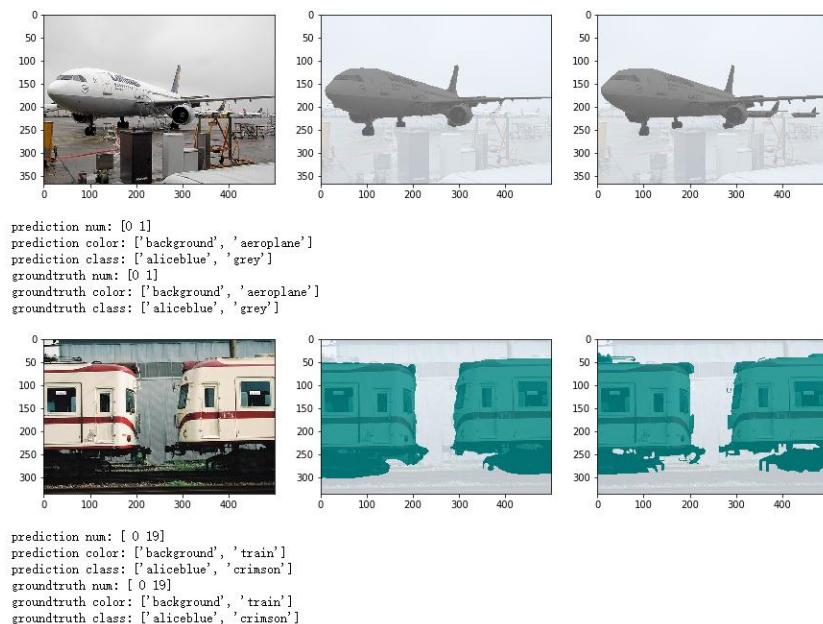


图 3-3 模型效果展示

## 3.5 实验总结

本实验主要介绍如何使用 MindSpore 在 voc2012 数据集上训练和推理 deeplabv3 网络模型，从而实现图像语义分割任务。通过本实验学员将了解如何处理图像分割数据标签，定义和训练卷积神经网络等的基本操作。

# 4

## 附录：ModelArts 开发环境搭建

- ModelArts 平台：Mindspore-1.5

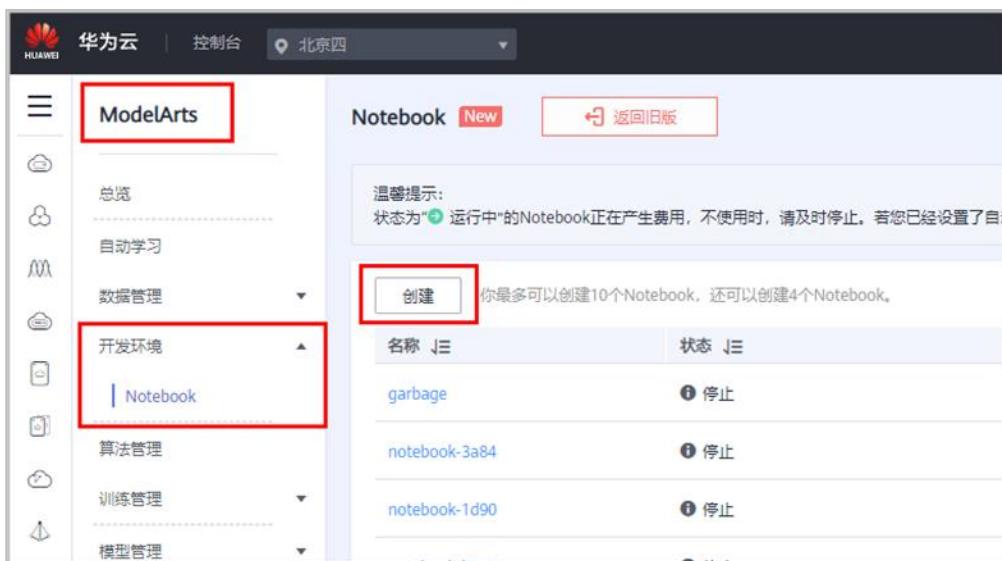
### 步骤 1 进入 ModelArts

在[华为云](#)主页搜索 Modelarts，点击“AI 开发平台 ModelArts”中的“进入控制台”。



### 步骤 2 选择训练作业

选择“北京四”地区，在左侧下拉框中点击“开发环境”中的“Notebook”：



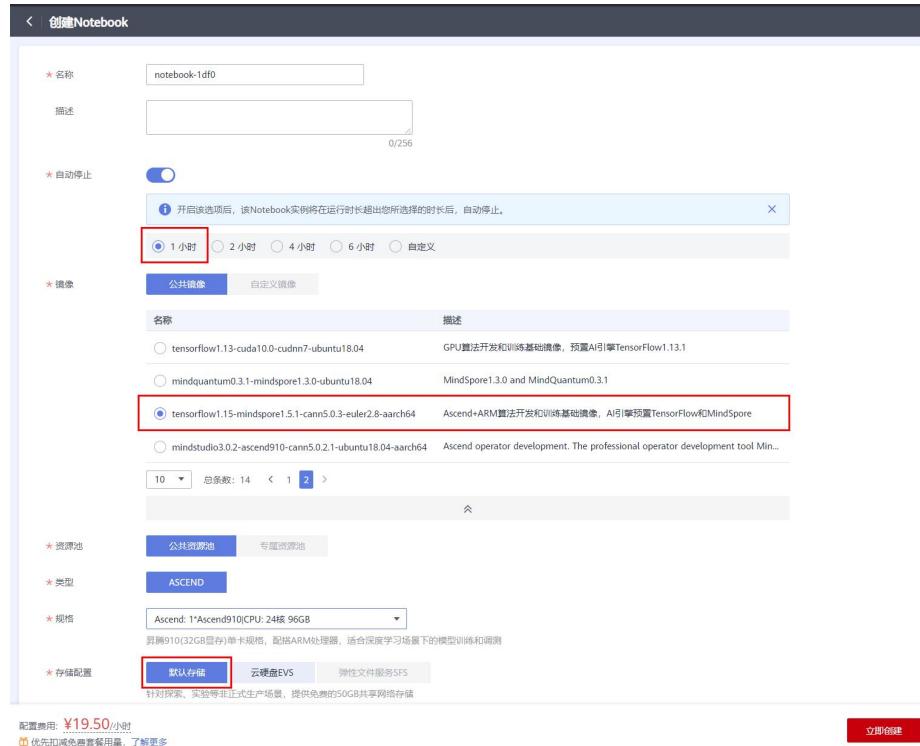
名称	状态
garbage	停止
notebook-3a84	停止
notebook-1d90	停止

### 步骤 3 创建 Notebook

点击创建按钮来创建一个新的 Notebook，选择如下配置：

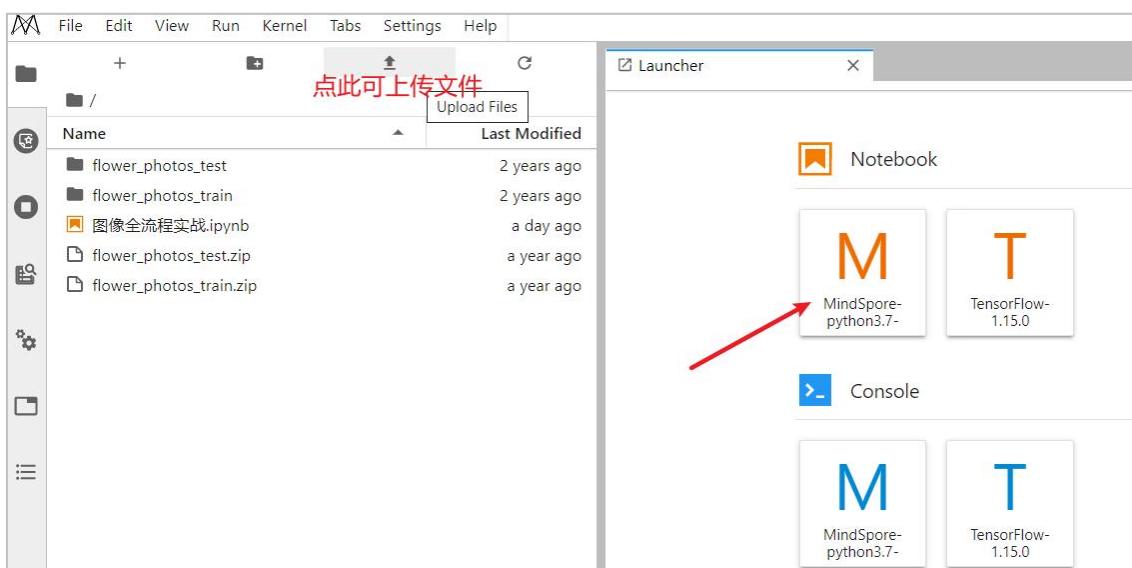
- 名称：自定义。
- 工作环境：Ascend+ARM 算法开发和训练基础镜像。
- 存储配置：默认存储。

点击“下一步”，确认规格如下后选择提交：



#### 步骤 4 启动 Notebook 进入开发环境

当 Notebook 状态变为“运行中”时，点击右侧“打开”按钮打开 Notebook。打开后选择右侧“MindSpore-python3.7-aarch64”按钮，进入 Notebook 环境：



#### 步骤 5 停止实验环境

试验完成之后请及时停止实验环境，避免资源浪费，如下图：



The screenshot shows the ModelArts interface with the 'Notebook' tab selected. A message at the top left says: '温馨提示：状态为“运行中”的Notebook正在产生费用，不使用时，请及时停止。若也已经设置了自动停止，请注意Notebook的剩余运行时间。' Below this, a table lists a single notebook entry:

名称	状态	镜像	规格	描述	创建时间	操作
dp	运行中 (40分钟)	tensorflow1.15-mindspore1.3.0-carm5.0.2-euler2.8...	Ascend: 1*Ascend910 CPU: 24核 96GB ...		2021/09/22 10:46:54 GMT+08:00	<a href="#">打开</a> <a href="#">后台</a> <a href="#">停止</a> <a href="#">更多</a>

图 4-1 停止实验环境