**计算机视觉课程设计-华为内容三**

**目录**

实验3-1

实验3-2

选做1

**实验3-1**

**实验要求：**

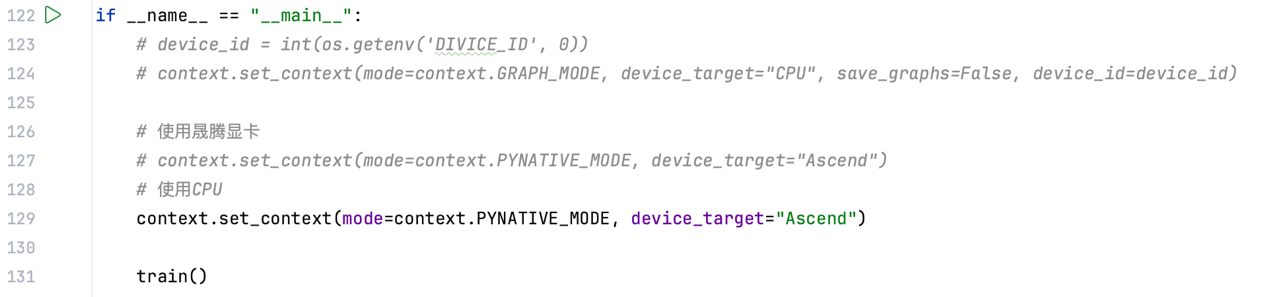
在modelarts上跑通肺实质分割训练及推理代码。

**代码及结果如下：**

import multiprocessing  
import os  
import mindspore.dataset as ds  
from mindspore import Tensor  
import numpy as np  
import mindspore.dataset.vision.py\_transforms as py\_vision  
from mindspore.dataset.transforms.py\_transforms import Compose  
from mydataset import DatasetGenerator  
from mindspore import nn  
from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net  
from mindspore.train.model import Model  
from mindspore.train.loss\_scale\_manager import FixedLossScaleManager  
from mindspore.train.callback import LossMonitor, TimeMonitor, ModelCheckpoint, CheckpointConfig  
from mindspore import context  
from nestedunet import NestedUNet  
from FWIoU\_metric import EvalCallBack, FWIoU  
from loss import WeightedBCELoss  
from UNet import UNet  
from attention\_Unet import AttU\_Net  
import argparse  
*# import moxing as mox*import learning\_rates  
from config import cfg  
  
*# 训练代码*def train():  
  
 *# 训练图像集* imgtrain\_list = []  
 train\_dataset\_generator = DatasetGenerator(os.path.join(cfg.DATA\_DIR, cfg.TRAIN\_DIR), cfg.IMAGE\_DIR, cfg.LABEL\_DIR, imgtrain\_list)  
 *# 测试图像集* imgtest\_list = []  
 valid\_dataset\_generator = DatasetGenerator(os.path.join(cfg.DATA\_DIR, cfg.VALID\_DIR), cfg.IMAGE\_DIR, cfg.LABEL\_DIR, imgtest\_list)  
  
 train\_dataset = ds.GeneratorDataset(train\_dataset\_generator,["image", "label"], shuffle=False)  
 valid\_dataset = ds.GeneratorDataset(valid\_dataset\_generator,["image", "label"], shuffle=False)  
  
 train\_dataset = train\_dataset.batch(cfg.BATCH\_SIZE, num\_parallel\_workers=1)  
 valid\_dataset = valid\_dataset.batch(cfg.BATCH\_SIZE, num\_parallel\_workers=1)  
  
 *# 损失函数  
 # loss = WeightedBCELoss(w0=1.39, w1=1.69)* loss = nn.BCEWithLogitsLoss()  
 *# loss = nn.DiceLoss()* loss.add\_flags\_recursive(fp32=True)  
  
 *# 网络模型  
 # train\_net = UNet()* train\_net = NestedUNet()  
 *# 不同的网络在这里进行设置，可以选择不同的模型  
 # 不同的网络在这里进行设置，可以选择不同的模型  
 # 不同的网络在这里进行设置，可以选择不同的模型* if os.path.exists(cfg.CKPT\_PRE\_TRAINED):  
 param\_dict = load\_checkpoint(cfg.CKPT\_PRE\_TRAINED)  
 load\_param\_into\_net(train\_net, param\_dict)  
  
 *# optimizer* iters\_per\_epoch = train\_dataset.get\_dataset\_size()  
 total\_train\_steps = iters\_per\_epoch \* cfg.EPOCHS  
 if cfg.LR\_TYPE == 'cos':  
 lr\_iter = learning\_rates.cosine\_lr(cfg.BASE\_LR, total\_train\_steps,  
 total\_train\_steps)  
 elif cfg.LR\_TYPE == 'poly':  
 lr\_iter = learning\_rates.poly\_lr(cfg.BASE\_LR,  
 total\_train\_steps,  
 total\_train\_steps,  
 end\_lr=0.0,  
 power=0.9)  
 elif cfg.LR\_TYPE == 'exp':  
 lr\_iter = learning\_rates.exponential\_lr(cfg.BASE\_LR,  
 cfg.LR\_DECAY\_STEP,  
 cfg.LR\_DECAY\_RATE,  
 total\_train\_steps,  
 staircase=True)  
 else:  
 raise ValueError('unknown learning rate type')  
  
 opt = nn.SGD(params=train\_net.trainable\_params(),  
 learning\_rate=lr\_iter,  
 momentum=0.9,  
 weight\_decay=0.0001,  
 loss\_scale=cfg.LOSS\_SCALE)  
  
 *# loss scale* manager\_loss\_scale = FixedLossScaleManager(cfg.LOSS\_SCALE,  
 drop\_overflow\_update=False)  
 model = Model(train\_net,  
 optimizer=opt,  
 amp\_level="O3",  
 loss\_fn=loss,  
 loss\_scale\_manager=manager\_loss\_scale)  
 epoch\_per\_eval = {"epoch": [], "FWIou": []}  
 *# callback for saving ckpts* time\_cb = TimeMonitor(data\_size=iters\_per\_epoch)  
 loss\_cb = LossMonitor()  
  
 *# 保存模型  
 # save\_checkpoint\_steps表示每隔多少个step保存一次，keep\_checkpoint\_max表示最多保留checkpoint文件的数量* config\_ckpt = CheckpointConfig(  
 save\_checkpoint\_steps=cfg.SAVE\_CHECKPOINT\_STEPS,  
 keep\_checkpoint\_max=cfg.KEEP\_CHECKPOINT\_MAX)  
 *# prefix表示生成CheckPoint文件的前缀名；directory：表示存放模型的目录* cbs\_1 = ModelCheckpoint(prefix=cfg.PREFIX,  
 directory=cfg.OUTPUT\_DIR,  
 config=config\_ckpt)  
 *# eval\_cb = EvalCallBack(model, valid\_dataset, cfg.EVAL\_PER\_EPOCH,  
 # epoch\_per\_eval)  
   
 # 需要打印的参数列表* cbs = [time\_cb, loss\_cb, cbs\_1]  
 *# 训练模型* model.train(cfg.EPOCHS, train\_dataset, callbacks=cbs, dataset\_sink\_mode=False)  
  
 *# mox.file.copy\_parallel(src\_url=cfg.OUTPUT\_DIR,  
 # dst\_url='obs://lqy/img-segment-main/output\_train/unet++')  
  
 # mox.file.copy\_parallel(src\_url=cfg.SUMMARY\_DIR,  
 # dst\_url='obs://lqy/img-segment-main/summary\_log/unet++')*if \_\_name\_\_ == "\_\_main\_\_":  
 *# device\_id = int(os.getenv('DIVICE\_ID', 0))  
 # context.set\_context(mode=context.GRAPH\_MODE, device\_target="CPU", save\_graphs=False, device\_id=device\_id)  
  
 # 使用晟腾显卡  
 # context.set\_context(mode=context.PYNATIVE\_MODE, device\_target="Ascend")  
 # 使用CPU* context.set\_context(mode=context.PYNATIVE\_MODE, device\_target="Ascend")  
  
 train()

****

****

****

**实验3-2**

**实验要求：**

将推理结果在itk-snap中进行三维重建。

结果如下：





**选做1**

**实验要求：**

将推理结果做后处理操作后在进行三维重建。

提示：1. 使用最大连通域方法进行后处理

2. 使用numpy库函数构建长方体

3. 将后处理后的的图像保存

要求：用不同颜色来显示后处理前后的三维重建结果

**结果如下：**

****

