C档模型 TransUNet

--by 2213409 石彬辰

准备工作

在项目根目录下创建data目录存放数据集文件，在TransUnet目录下创建config目录存放其他python文件，由于TransUNet项目使用npz模式的数据进行训练，在config目录下新建一个图像预处理文件，由于BUSI-256没有测试集，将15%的训练集数据划出去作为测试集，然后使用numpy将mask图像中的非0的灰度值转为1，接着将image和转换后对应的label作为两个numpy数据压缩到一起存储为npz文件，isic2018数据集有测试集，故不作划分处理

image_trans_npz.py

```python
import glob
import os
import random

import cv2
import numpy as np
from PIL import Image


def npz(path, path2, split=False):
    test_num = 0
    train_num = 0
    if not os.path.exists(path2):
        os.makedirs(path2)

    for i, img_path in enumerate(glob.glob(path)):

        # 读入图像
        image = cv2.imread(img_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        # image=np.array(image)
        # image[image>0]=1
        # 读入标签
        label_path = img_path.replace('images', 'masks')

        # print(img_path)
        # print(label_path)
        if not os.path.exists(img_path):
            print(111)
        if not os.path.exists(label_path):
            print(222)
        label = cv2.imread(label_path, flags=0)
        label=np.array(label)
        label[label>0]=1
        # for i in label:
        #     print(i)
        # break
        path_save = path2
        if split:
            train2test = random.random()
            if train2test >= 0.85:
                path_save = path_save.replace('train_npz', 'test_vol_h5')
                if not os.path.exists(path_save):
                    os.makedirs(path_save)
                test_num += 1
            else:
                train_num += 1
        else:train_num+=1

        # 保存npz
        np.savez(path_save + str(i + 1), image=image, label=label)
        print('------------', i + 1)

    print('ok')
    print('train_num:', train_num)
    print('test_num', test_num)


if __name__ == "__main__":
    aiot = {
        'BUSI-256': {
            'train': {
                "path": r'D:\Visual studio\code\pycharm\TransUNet\data\BUSI-256\images\*.png',
                "path2": r'D:\Visual studio\code\pycharm\TransUNet\data\BUSI-256\train_npz\\'
            },
            'test': None
        },
        'isic2018': {
            'train': {
                "path": r'D:\Visual studio\code\pycharm\TransUNet\data\isic2018\train\images\*.png',
                "path2": r'D:\Visual studio\code\pycharm\TransUNet\data\isic2018\train\train_npz\\'

            },
            'test': {
                "path": r'D:\Visual studio\code\pycharm\TransUNet\data\isic2018\test\images\*.png',
                "path2": r'D:\Visual studio\code\pycharm\TransUNet\data\isic2018\test\test_vol_h5\\'
            }
        }
    }

    npz(aiot['BUSI-256']['train']['path'], aiot['BUSI-256']['train']['path2'], split=True)
```

数据预处理之后在config目录下创建一个py文件进行训练集和测试集的预加载,将训练集和测试集文件名分别保存在train.txt和test_vol.txt中

data_load.py

```
import os


def load_dataset(dir_data, path_txt):
    npzs = os.listdir(dir_data)
    if not os.path.exists(os.path.dirname(path_txt)):
        os.makedirs(os.path.dirname(path_txt))
    with open(path_txt, 'w', encoding='utf-8') as f:
        for npz in npzs:
            f.write(npz.split('.')[0]+'\n')


if __name__ == '__main__':
    aiot = {
        'BUSI-256': {
            'train': {
                'dir': r'D:\Visual studio\code\pycharm\TransUNet\data\BUSI-256\train_npz',
                'txt': r'D:\Visual studio\code\pycharm\TransUNet\TransUnet\lists\lists_BUSI-256\train.txt'
            },
            'test': {
                'dir': r'D:\Visual studio\code\pycharm\TransUNet\data\BUSI-256\test_vol_h5',
                'txt': r'D:\Visual studio\code\pycharm\TransUNet\TransUnet\lists\lists_BUSI-256\test_vol.txt'
            }
        },
        'isic2018': {
            'train': {
                'dir': r'D:\Visual studio\code\pycharm\TransUNet\data\isic2018\train\train_npz',
                'txt': r'D:\Visual studio\code\pycharm\TransUNet\TransUnet\lists\lists_isic2018\train.txt'
            },
            'test': {
                'dir': r'D:\Visual studio\code\pycharm\TransUNet\data\isic2018\test\test_vol_h5',
                'txt': r'D:\Visual studio\code\pycharm\TransUNet\TransUnet\lists\lists_isic2018\test_vol.txt'
            }
        }
    }
    load_dataset(aiot['BUSI-256']['train']['dir'],aiot['BUSI-256']['train']['txt'])
    load_dataset(aiot['BUSI-256']['test']['dir'], aiot['BUSI-256']['test']['txt'])
    load_dataset(aiot['isic2018']['train']['dir'], aiot['isic2018']['train']['txt'])
    load_dataset(aiot['isic2018']['test']['dir'], aiot['isic2018']['test']['txt'])
```

修改trainer.py文件，使训练模型每5轮保存一次，且损失率降低时更新最佳模型，同时将每一轮的损失率写入loss.txt文件中

trainer.py

```
import argparse
import json
import logging
import os
import random
import sys
import time
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from tensorboardX import SummaryWriter
from torch.nn.modules.loss import CrossEntropyLoss
from torch.utils.data import DataLoader
from tqdm import tqdm
from utils import DiceLoss
from torchvision import transforms

```

```python

def trainer_synapse(args, model, snapshot_path):
    from datasets.dataset_synapse import Synapse_dataset, RandomGenerator
    logging.basicConfig(filename=snapshot_path + "/log.txt", level=logging.INFO,
                        format='[%(asctime)s.%(msecs)03d] %(message)s', datefmt='%H:%M:%S')
    logging.getLogger().addHandler(logging.StreamHandler(sys.stdout))
    logging.info(str(args))
    base_lr = args.base_lr
    num_classes = args.num_classes
    batch_size = args.batch_size * args.n_gpu
    # max_iterations = args.max_iterations
    db_train = Synapse_dataset(base_dir=args.root_path, list_dir=args.list_dir, split="train",
                               transform=transforms.Compose(
                                   [RandomGenerator(output_size=[args.img_size, args.img_size])]))
    print("The length of train set is: {}".format(len(db_train)))

    def worker_init_fn(worker_id):
        random.seed(args.seed + worker_id)

    trainloader = DataLoader(db_train, batch_size=batch_size, shuffle=True, num_workers=0, pin_memory=True,
                             worker_init_fn=worker_init_fn)
    if args.n_gpu > 1:
        model = nn.DataParallel(model)
    model.train()
    ce_loss = CrossEntropyLoss()
    dice_loss = DiceLoss(num_classes)
    optimizer = optim.SGD(model.parameters(), lr=base_lr, momentum=0.9, weight_decay=0.0001)
    writer = SummaryWriter(snapshot_path + '/log')
    iter_num = 0
    max_epoch = args.max_epochs
    max_iterations = args.max_epochs * len(trainloader)  # max_epoch = max_iterations // len(trainloader) + 1
    logging.info("{} iterations per epoch. {} max iterations ".format(len(trainloader), max_iterations))
    best_performance = 0.0
    iterator = tqdm(range(max_epoch), ncols=70)
    loss_min = float('inf')
    loss_file = snapshot_path + "/loss.txt"
    for epoch_num in iterator:
        loss_current = []

        for i_batch, sampled_batch in enumerate(trainloader):
            image_batch, label_batch = sampled_batch['image'], sampled_batch['label']
            image_batch, label_batch = image_batch.cuda(), label_batch.cuda()
            outputs = model(image_batch)
            loss_ce = ce_loss(outputs, label_batch[:].long())
            loss_dice = dice_loss(outputs, label_batch, softmax=True)
            loss = 0.5 * loss_ce + 0.5 * loss_dice

            optimizer.zero_grad()
            loss.backward()

            optimizer.step()
            lr_ = base_lr * (1.0 - iter_num / max_iterations) ** 0.9
            for param_group in optimizer.param_groups:
                param_group['lr'] = lr_

            iter_num = iter_num + 1
            writer.add_scalar('info/lr', lr_, iter_num)
            writer.add_scalar('info/total_loss', loss, iter_num)
            writer.add_scalar('info/loss_ce', loss_ce, iter_num)

            logging.info('iteration %d : loss : %f, loss_ce: %f' % (iter_num,
```

由于生成的loss.txt包含每一个iteration的loss，创建log_loss.py，将每一轮的平均loss写入loss_avg.txt

log_loss.py

```python
import json


def read_log(file_path):
    loss_avg=[]
    with open(file_path, 'r') as f:
        loss_total = f.readlines()
        for loss_epoch in loss_total:
            loss_epoch=json.loads(loss_epoch.replace("'",'"'))
            loss_avg.append(sum(loss_epoch["loss"])/len(loss_epoch["loss"]))
    with open(file_path.replace("loss.txt","loss_avg.txt"),"w")as f:
        for i in loss_avg:
            f.write(i.__str__())
            f.write('\r')



#read_log(r'D:\Visual studio\code\pycharm\TransUNet\model\TU_BUSI-256256\TU_pretrain_R50-ViT-B_16_skip3_epo100_bs6_256\loss.txt')
```

```
read_log(r'D:\Visual studio\code\pycharm\TransUNet\model\TU_isic2018256\TU_pretrain_R50-ViT-B_16_skip3_epo100_bs6_256\loss.txt')
```

创建epoch_loss.py，通过loss_avg.txt生成轮次-损失率图像

epoch_loss.py

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
def plot_training_curve(train_log_path):
    train_file = os.path.join(train_log_path, 'loss_avg.txt')
    output_path=os.path.join(train_log_path,'epoch_loss.png')
    loss_values=[]
    with open(train_file,'r') as f:
        for line in f:
            loss=float(line.strip())
            loss_values.append(loss)
    if not loss_values:
        print("训练日志为空，无法生成图表")
        return
    epochs=np.arange(1,len(loss_values)+1)
    #print(epochs)
    #print(loss_values)
    log_df=pd.DataFrame({'epoch': epochs, 'loss': loss_values})
    #print(log_df)
    # 动态获取轮次和损失范围
    max_epoch = log_df['epoch'].max()
    min_loss = log_df['loss'].min()
    max_loss = log_df['loss'].max()

    # 自动调整纵轴范围（留10%缓冲空间）
    loss_buffer = (max_loss - min_loss) * 0.1
    y_min = max(0, min_loss - loss_buffer)  # 确保不低于0
    y_max = max_loss + loss_buffer

    # 自动调整横轴刻度步长
    epoch_step = max(1, int(max_epoch / 5))  # 至少显示5个刻度

    # 动态调整平滑参数（基于数据长度）
    sigma = max(1.0, 30 / len(log_df))

    # 生成平滑曲线
    smooth_loss = gaussian_filter1d(log_df['loss'], sigma=sigma)

    # 创建图表
    plt.figure(figsize=(10, 6))
    plt.plot(log_df['epoch'], log_df['loss'], label='train loss', linewidth=2.5, color='#d62728')
    plt.plot(log_df['epoch'], smooth_loss, label='smooth train loss', linewidth=2.5, linestyle='--',color='#007B00')

    # 设置动态坐标轴
    plt.xlim(0, max_epoch)
    plt.ylim(y_min, y_max)
    plt.xticks(range(0, max_epoch + 1, epoch_step))

    # 样式设置
    plt.xlabel("Epoch", fontsize=12, fontweight='bold')
    plt.ylabel("Loss", fontsize=12, fontweight='bold')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend(frameon=True, shadow=True)

    plt.tight_layout()
    plt.savefig(output_path, dpi=350, bbox_inches='tight')
    plt.close()

if __name__=='__main__':
    #功能测试
    #plot_training_curve("D:\Visual studio\code\pycharm\TransUNet\model\TU_BUSI-256256\TU_pretrain_R50-ViT-B_16_skip3_epo100_bs6_256")
    plot_training_curve("D:\Visual studio\code\pycharm\TransUNet\model\TU_isic2018256\TU_pretrain_R50-ViT-B_16_skip3_epo100_bs6_256")
```

修改train.py，添加需要训练数据集的参数，将默认的训练轮数max epochs设为100，图像尺寸image size设为256，步长batch size设为6，分类数classes num设为2，开始训练

train.py

```
import argparse
import logging
import os
import random
import numpy as np
import torch
import torch.backends.cudnn as cudnn
from networks.vit_seg_modeling import VisionTransformer as ViT_seg
from networks.vit_seg_modeling import CONFIGS as CONFIGS_ViT_seg
from trainer import trainer_synapse

parser = argparse.ArgumentParser()
parser.add_argument('--root_path', type=str,
                    default='../data/Synapse/train_npz', help='root dir for data')
parser.add_argument('--dataset', type=str,
                    default='Synapse', help='experiment_name')
parser.add_argument('--list_dir', type=str,
                    default='./lists/lists_Synapse', help='list dir')
parser.add_argument('--num_classes', type=int,
                    default=2, help='output channel of network')
parser.add_argument('--max_iterations', type=int,
                    default=30000, help='maximum epoch number to train')
parser.add_argument('--max_epochs', type=int,
                    default=100, help='maximum epoch number to train')
parser.add_argument('--batch_size', type=int,
                    default=6, help='batch_size per gpu')
parser.add_argument('--n_gpu', type=int, default=1, help='total gpu')
parser.add_argument('--deterministic', type=int,  default=1,
                    help='whether use deterministic training')
parser.add_argument('--base_lr', type=float,  default=0.01,
                    help='segmentation network learning rate')
parser.add_argument('--img_size', type=int,
                    default=256, help='input patch size of network input')
parser.add_argument('--seed', type=int,
                    default=1234, help='random seed')
parser.add_argument('--n_skip', type=int,
                    default=3, help='using number of skip-connect, default is num')
parser.add_argument('--vit_name', type=str,
                    default='R50-ViT-B_16', help='select one vit model')
parser.add_argument('--vit_patches_size', type=int,
                    default=16, help='vit_patches_size, default is 16')
args = parser.parse_args()


if __name__ == "__main__":
    if not args.deterministic:
        cudnn.benchmark = True
        cudnn.deterministic = False
    else:
        cudnn.benchmark = False
        cudnn.deterministic = True

    args.dataset='BUSI-256'
    random.seed(args.seed)
    np.random.seed(args.seed)
    torch.manual_seed(args.seed)
    torch.cuda.manual_seed(args.seed)
    dataset_name = args.dataset
    dataset_config = {
        'Synapse': {
            'root_path': '../data/Synapse/train_npz',
            'list_dir': './lists/lists_Synapse',
            'num_classes': 9,
        },
        'BUSI-256':{
            'root_path': '../data/BUSI-256/train_npz',
            'list_dir': './lists/lists_BUSI-256',
            'num_classes': 2,
            'img_size':256,
        },
        'isic2018':{
            'root_path': '../data/isic2018/train/train_npz',
            'list_dir': './lists/lists_isic2018',
            'num_classes': 2,
            'img_size': 256,
        }
    }
    args.num_classes = dataset_config[dataset_name]['num_classes']
    args.root_path = dataset_config[dataset_name]['root_path']
    args.list_dir = dataset_config[dataset_name]['list_dir']
    args.is_pretrain = True
    args.exp = 'TU_' + dataset_name + str(args.img_size)
    snapshot_path = "../model/{}/{}".format(args.exp, 'TU')
    snapshot_path = snapshot_path + '_pretrain' if args.is_pretrain else snapshot_path
    snapshot_path += '_' + args.vit_name
```

修改utils.py，将预测得到的图像非0灰度值转为255

utils.py

```
import copy
import os

import numpy as np
import torch
from medpy import metric
from scipy.ndimage import zoom
import torch.nn as nn
import SimpleITK as sitk
from PIL import Image
```

```

class DiceLoss(nn.Module):
    def __init__(self, n_classes):
        super(DiceLoss, self).__init__()
        self.n_classes = n_classes

    def _one_hot_encoder(self, input_tensor):
        tensor_list = []
        for i in range(self.n_classes):
            temp_prob = input_tensor == i  # * torch.ones_like(input_tensor)
            tensor_list.append(temp_prob.unsqueeze(1))
        output_tensor = torch.cat(tensor_list, dim=1)
        return output_tensor.float()

    def _dice_loss(self, score, target):
        target = target.float()
        smooth = 1e-5
        intersect = torch.sum(score * target)
        y_sum = torch.sum(target * target)
        z_sum = torch.sum(score * score)
        loss = (2 * intersect + smooth) / (z_sum + y_sum + smooth)
        loss = 1 - loss
        return loss

    def forward(self, inputs, target, weight=None, softmax=False):
        if softmax:
            inputs = torch.softmax(inputs, dim=1)
        target = self._one_hot_encoder(target)
        if weight is None:
            weight = [1] * self.n_classes
        assert inputs.size() == target.size(), 'predict {} & target {} shape do not match'.format(inputs.size(), target.size())
        class_wise_dice = []
        loss = 0.0
        for i in range(0, self.n_classes):
            dice = self._dice_loss(inputs[:, i], target[:, i])
            class_wise_dice.append(1.0 - dice.item())
            loss += dice * weight[i]
        return loss / self.n_classes


def calculate_metric_percase(pred, gt):
    pred[pred > 0] = 1
    gt[gt > 0] = 1
    if pred.sum() > 0 and gt.sum()>0:
        dice = metric.binary.dc(pred, gt)
        hd95 = metric.binary.hd95(pred, gt)
        return dice, hd95
    elif pred.sum() > 0 and gt.sum()==0:
        return 1, 0
    else:
        return 0, 0


def test_single_volume(image, label, net, classes, patch_size=[256, 256], test_save_path=None, case=None, z_spacing=1):
    image, label = image.squeeze(0).cpu().detach().numpy(), label.squeeze(0).cpu().detach().numpy()
    _,x,y=image.shape
    if x != patch_size[0] or y != patch_size[1]:
        image = zoom(image, (1, patch_size[0] / x, patch_size[1] / y), order=3)
    input = torch.from_numpy(image).unsqueeze(0).float().cuda()
    net.eval()
    with torch.no_grad():
        out = torch.argmax(torch.softmax(net(input), dim=1), dim=1).squeeze(0)
        out = out.cpu().detach().numpy()
        if x != patch_size[0] or y != patch_size[1]:
            # 缩放图像至原始大小
            prediction = zoom(out, (x / patch_size[0], y / patch_size[1]), order=0)
        else:
            prediction = out

    metric_list = []
    for i in range(1, classes):
        metric_list.append(calculate_metric_percase(prediction == i, label == i))
    if test_save_path is not None:
```

修改test.py，添加数据集的参数和数据集对应的模型路径

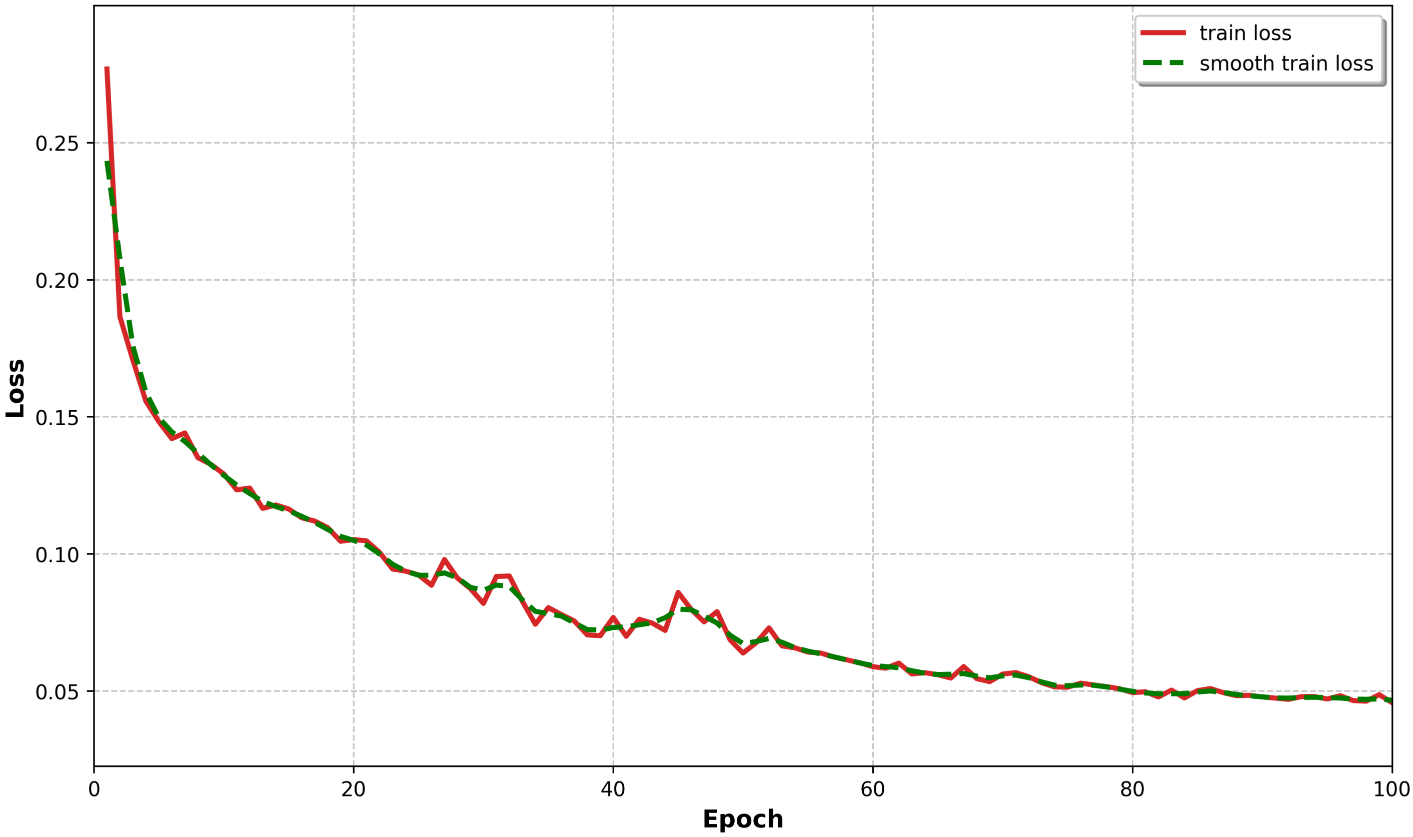
test.py

```
import argparse
import logging
import os
import random
import sys
import numpy as np
import torch
import torch.backends.cudnn as cudnn
import torch.nn as nn
from torch.utils.data import DataLoader
from tqdm import tqdm
from datasets.dataset_synapse import Synapse_dataset
```

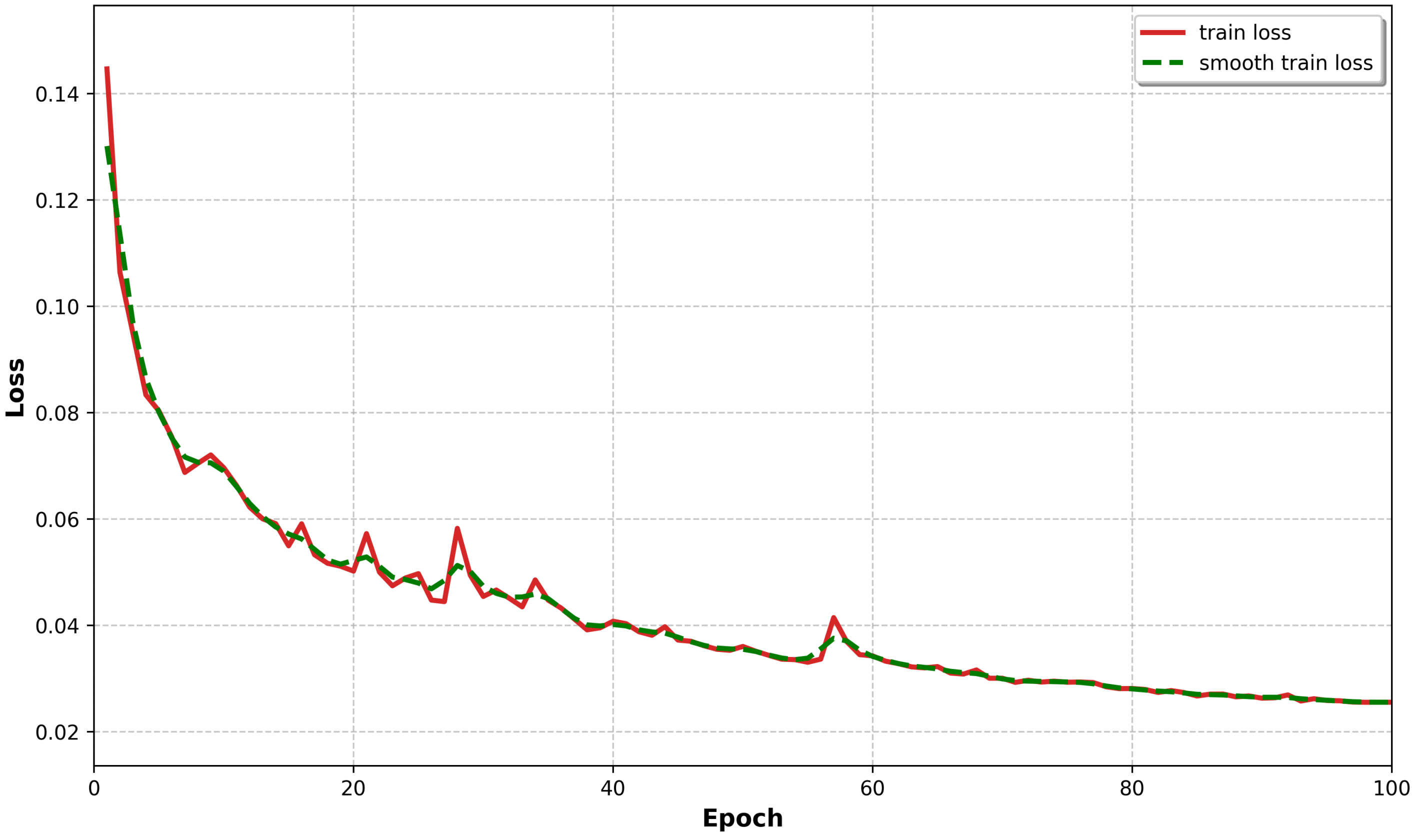
```python
from utils import test_single_volume
from networks.vit_seg_modeling import VisionTransformer as ViT_seg
from networks.vit_seg_modeling import CONFIGS as CONFIGS_ViT_seg

parser = argparse.ArgumentParser()
parser.add_argument('--volume_path', type=str,
                    default='../data/Synapse/test_vol_h5',
                    help='root dir for validation volume data')  # for acdc volume_path=root_dir
parser.add_argument('--dataset', type=str,
                    default='Synapse', help='experiment_name')
parser.add_argument('--num_classes', type=int,
                    default=2, help='output channel of network')
parser.add_argument('--list_dir', type=str,
                    default='./lists/lists_Synapse', help='list dir')

parser.add_argument('--max_iterations', type=int,
                    default=30000, help='maximum epoch number to train')
parser.add_argument('--max_epochs', type=int,
                    default=100, help='maximum epoch number to train')
parser.add_argument('--batch_size', type=int,
                    default=6, help='batch_size per gpu')
parser.add_argument('--img_size', type=int,
                    default=256, help='input patch size of network input')
parser.add_argument('--is_savenii', action="store_true", default=True,help='whether to save results during inference')

parser.add_argument('--n_skip', type=int, default=3, help='using number of skip-connect, default is num')
parser.add_argument('--vit_name', type=str, default='R50-ViT-B_16', help='select one vit model')

parser.add_argument('--test_save_dir', type=str, default='../predictions', help='saving prediction as nii!')
parser.add_argument('--deterministic', type=int, default=1, help='whether use deterministic training')
parser.add_argument('--base_lr', type=float, default=0.01, help='segmentation network learning rate')
parser.add_argument('--seed', type=int, default=1234, help='random seed')
parser.add_argument('--vit_patches_size', type=int, default=16, help='vit_patches_size, default is 16')
args = parser.parse_args()


def inference(args, model, test_save_path=None):
    db_test = args.Dataset(base_dir=args.volume_path, split="test_vol", list_dir=args.list_dir)
    testloader = DataLoader(db_test, batch_size=1, shuffle=False, num_workers=0)
    logging.info("{} test iterations per epoch".format(len(testloader)))
    model.eval()
    metric_list = 0.0
    for i_batch, sampled_batch in tqdm(enumerate(testloader)):
        h, w = sampled_batch["image"].size()[2:]
        image, label, case_name = sampled_batch["image"], sampled_batch["label"], sampled_batch['case_name'][0]
        metric_i = test_single_volume(image, label, model, classes=args.num_classes,
                                      patch_size=[args.img_size, args.img_size],
                                      test_save_path=test_save_path, case=case_name, z_spacing=args.z_spacing)
        metric_list += np.array(metric_i)
        logging.info('idx %d case %s mean_dice %f mean_hd95 %f' % (
        i_batch, case_name, np.mean(metric_i, axis=0)[0], np.mean(metric_i, axis=0)[1]))
    metric_list = metric_list / len(db_test)
    for i in range(1, args.num_classes):
        logging.info('Mean class %d mean_dice %f mean_hd95 %f' % (i, metric_list[i - 1][0], metric_list[i - 1][1]))
    performance = np.mean(metric_list, axis=0)[0]
    mean_hd95 = np.mean(metric_list, axis=0)[1]
    logging.info('Testing performance in best val model: mean_dice : %f mean_hd95 : %f' % (performance, mean_hd95))
    return "Testing Finished!"

```

训练模型

分别训练两个数据集，epoch-loss图像如下



BUSI-256数据集



isic2018数据集

预测图像

运行test.py测试两个数据集的最佳模型

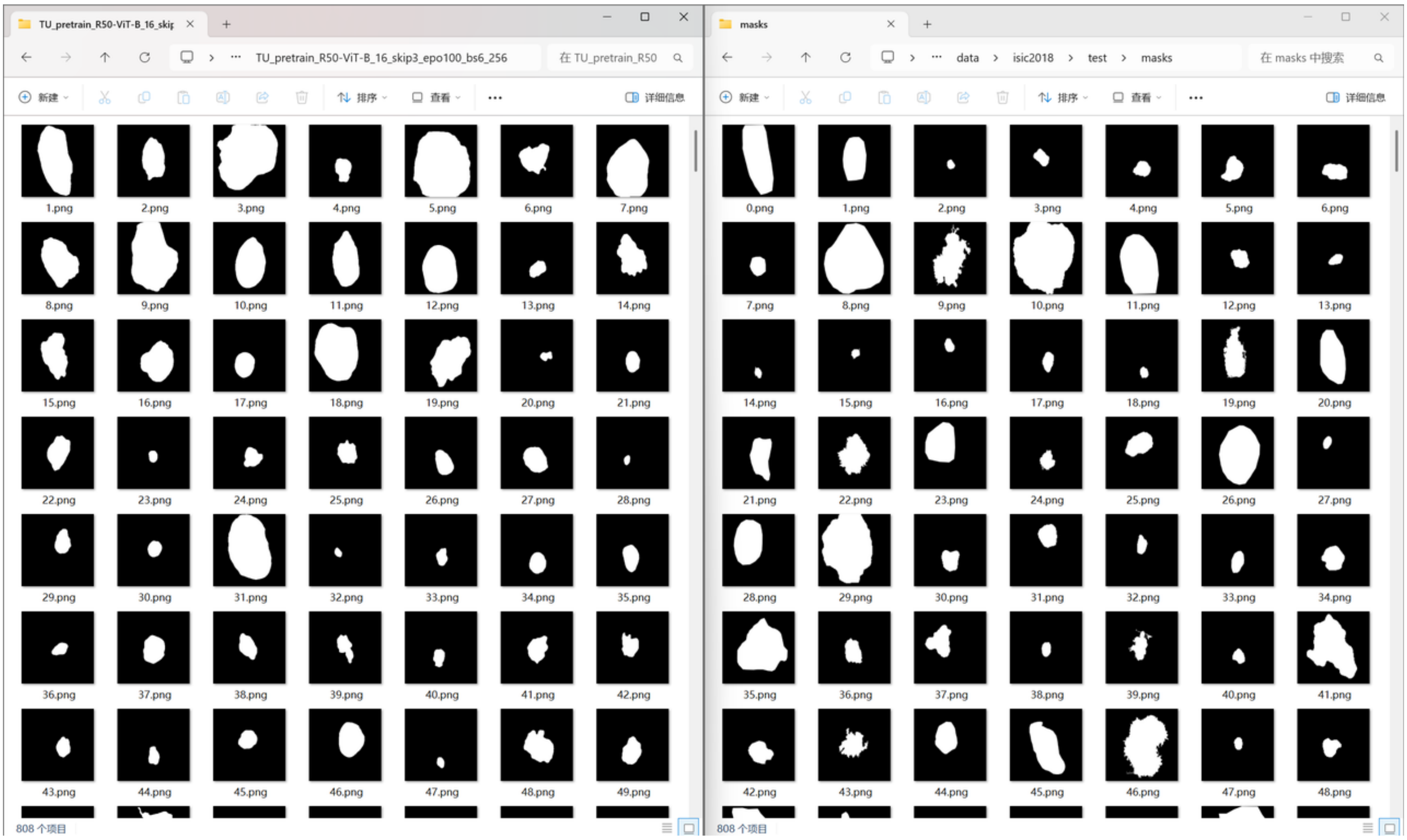
test_log_isic2018

```
Testing performance in best val model: mean_dice : 0.893347 mean_hd95 :
15.194364
```

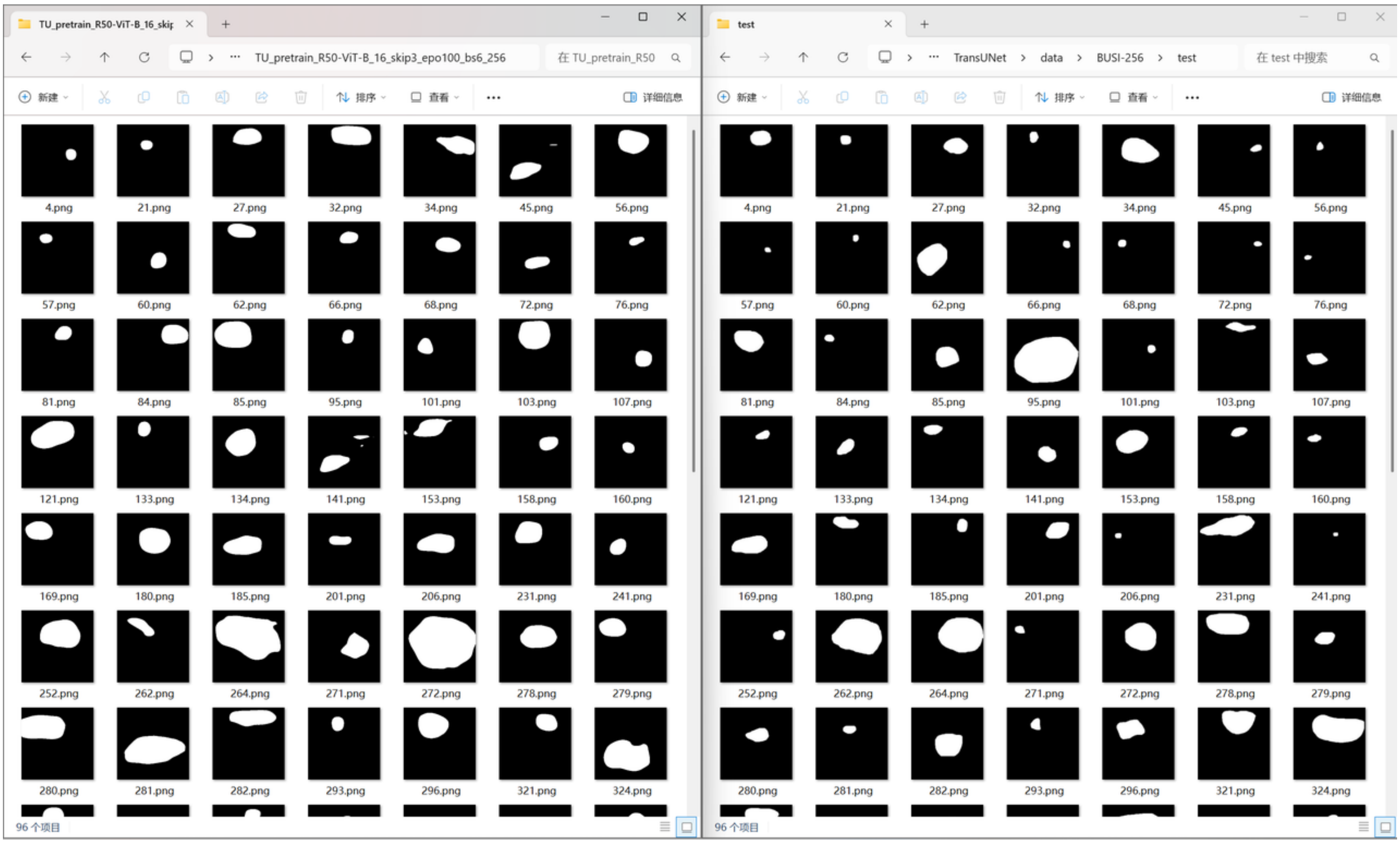
test_log_BUSI-256

```
Testing performance in best val model: mean_dice : 0.786324 mean_hd95 :
22.398065
```

可见isic2018数据集的最佳模型dice系数为0.89，模型性能良好，BUSI-256数据集最佳模型dice为0.786，性能不太良好


TU_pretrain_R50-ViT-B_16_skip
TU_pretrain_R50-ViT-B_16_skip3_epo100_bs6_256
在 TU_pretrain_R50
新建
排序
查看
详细信息
1.png 2.png 3.png 4.png 5.png 6.png 7.png
8.png 9.png 10.png 11.png 12.png 13.png 14.png
15.png 16.png 17.png 18.png 19.png 20.png 21.png
22.png 23.png 24.png 25.png 26.png 27.png 28.png
29.png 30.png 31.png 32.png 33.png 34.png 35.png
36.png 37.png 38.png 39.png 40.png 41.png 42.png
43.png 44.png 45.png 46.png 47.png 48.png 49.png
808 个项目
masks
data › isic2018 › test › masks
在 masks 中搜索
新建
排序
查看
详细信息
0.png 1.png 2.png 3.png 4.png 5.png 6.png
7.png 8.png 9.png 10.png 11.png 12.png 13.png
14.png 15.png 16.png 17.png 18.png 19.png 20.png
21.png 22.png 23.png 24.png 25.png 26.png 27.png
28.png 29.png 30.png 31.png 32.png 33.png 34.png
35.png 36.png 37.png 38.png 39.png 40.png 41.png
42.png 43.png 44.png 45.png 46.png 47.png 48.png
808 个项目

isic2018测试集对比，左为预测，右为mask


TU_pretrain_R50-ViT-B_16_skip
TU_pretrain_R50-ViT-B_16_skip3_epo100_bs6_256
在 TU_pretrain_R50
新建
排序
查看
详细信息
4.png 21.png 27.png 32.png 34.png 45.png 56.png
57.png 60.png 62.png 66.png 68.png 72.png 76.png
81.png 84.png 85.png 95.png 101.png 103.png 107.png
121.png 133.png 134.png 141.png 153.png 158.png 160.png
169.png 180.png 185.png 201.png 206.png 231.png 241.png
252.png 262.png 264.png 271.png 272.png 278.png 279.png
280.png 281.png 282.png 293.png 296.png 321.png 324.png
96 个项目
test
TransUNet › data › BUSI-256 › test
在 test 中搜索
新建
排序
查看
详细信息
4.png 21.png 27.png 32.png 34.png 45.png 56.png
57.png 60.png 62.png 66.png 68.png 72.png 76.png
81.png 84.png 85.png 95.png 101.png 103.png 107.png
121.png 133.png 134.png 141.png 153.png 158.png 160.png
169.png 180.png 185.png 201.png 206.png 231.png 241.png
252.png 262.png 264.png 271.png 272.png 278.png 279.png
280.png 281.png 282.png 293.png 296.png 321.png 324.png
96 个项目

BUSI-256测试集对比，左为预测，右为mask

备注

更新了需求文档requirement.txt