cldice.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from .soft_skeleton import soft_skel


class soft_cldice(nn.Module):
    def __init__(self, iter_=3, smooth = 1.):
        super(soft_cldice, self).__init__()
        self.iter = iter_
        self.smooth = smooth

    def forward(y_true, y_pred):
        skel_pred = soft_skel(y_pred, iters)
        skel_true = soft_skel(y_true, iters)
        tprec = (torch.sum(torch.multiply(skel_pred, y_true)
[:,1:,...])+smooth)/(torch.sum(skel_pred[:,1:,...])+smooth)
        tsens = (torch.sum(torch.multiply(skel_true, y_pred)
[:,1:,...])+smooth)/(torch.sum(skel_true[:,1:,...])+smooth)
        cl_dice = 1.- 2.0*(tprec*tsens)/(tprec+tsens)
        return cl_dice


def soft_dice(y_true, y_pred):
    """[function to compute dice loss]
    Args:
        y_true ([float32]): [ground truth image]
        y_pred ([float32]): [predicted image]
    Returns:
        [float32]: [loss value]
    """
    smooth = 1
    intersection = torch.sum((y_true * y_pred)[:,1:,...])
    coeff = (2. *  intersection + smooth) / (torch.sum(y_true[:,1:,...]) +
torch.sum(y_pred[:,1:,...]) + smooth)
    return (1. - coeff)


class soft_dice_cldice(nn.Module):
    def __init__(self, iter_=3, alpha=0.5, smooth = 1.):
        super(soft_cldice, self).__init__()
        self.iter = iter_
        self.smooth = smooth
        self.alpha = alpha

    def forward(y_true, y_pred):
        dice = soft_dice(y_true, y_pred)
        skel_pred = soft_skel(y_pred, self.iter)
        skel_true = soft_skel(y_true, self.iter)
        tprec = (torch.sum(torch.multiply(skel_pred, y_true)
[:,1:,...])+self.smooth)/(torch.sum(skel_pred[:,1:,...])+self.smooth)
        tsens = (torch.sum(torch.multiply(skel_true, y_pred)
[:,1:,...])+self.smooth)/(torch.sum(skel_true[:,1:,...])+self.smooth)
```

```
        cl_dice = 1.- 2.0*(tprec*tsens)/(tprec+tsens)
        return (1.0-self.alpha)*dice+self.alpha*cl_dice
```

2.soft_skeleton

```
import torch
import torch.nn as nn
import torch.nn.functional as F


def soft_erode(img):
    if len(img.shape)==4:
        p1 = -F.max_pool2d(-img, (3,1), (1,1), (1,0))
        p2 = -F.max_pool2d(-img, (1,3), (1,1), (0,1))
        return torch.min(p1,p2)
    elif len(img.shape)==5:
        p1 = -F.max_pool3d(-img,(3,1,1),(1,1,1),(1,0,0))
        p2 = -F.max_pool3d(-img,(1,3,1),(1,1,1),(0,1,0))
        p3 = -F.max_pool3d(-img,(1,1,3),(1,1,1),(0,0,1))
        return torch.min(torch.min(p1, p2), p3)


def soft_dilate(img):
    if len(img.shape)==4:
        return F.max_pool2d(img, (3,3), (1,1), (1,1))
    elif len(img.shape)==5:
        return F.max_pool3d(img,(3,3,3),(1,1,1),(1,1,1))


def soft_open(img):
    return soft_dilate(soft_erode(img))


def soft_skel(img, iter_):
    img1  =  soft_open(img)
    skel  =  F.relu(img-img1)
    for j in range(iter_):
        img  =  soft_erode(img)
        img1  =  soft_open(img)
        delta  =  F.relu(img-img1)
        skel  =  skel +  F.relu(delta-skel*delta)
    return skel
```