**Main.py**

from PIL import Image  
import numpy as np # 导入 numpy 库  
from skimage import metrics  
from skimage.transform import resize  
  
# 打开并调整图像大小  
image1 = Image.open(r"D:\Desktop\yyimage\image ql.1.png")  
image2 = Image.open(r"D:\Desktop\yyimage\image ql.1.png")  
image1 = image1.resize((500, 500)) # 调整图像1的大小为500x500  
image2 = image2.resize((500, 500)) # 调整图像2的大小为500x500  
  
# 将图像转换为灰度图像  
image1\_gray = image1.convert("L")  
image2\_gray = image2.convert("L")  
  
# 将图像转换为NumPy数组  
image1\_array = np.array(image1\_gray)  
image2\_array = np.array(image2\_gray)  
  
# 计算结构相似性指数（SSIM）  
similarity = metrics.structural\_similarity(image1\_array, image2\_array)  
  
# 将相似性指数转换为相似度（范围0到1，值越大表示相似度越高）  
similarity = (similarity + 1) / 2  
print(similarity)

**详细代码**

def gaussian(window\_size, sigma):  
 # 计算公式:e^(-x^2)/(2\*sigma^2)，其中x表示距离中心点的距离，sigma默认1.5  
 gauss = torch.Tensor([exp(-(x - window\_size // 2) \*\* 2 / float(2 \* sigma \*\* 2)) for x in range(window\_size)])  
 # 数据归一化  
 return gauss / gauss.sum()  
  
  
# 计算滑动窗口权重  
def create\_window(window\_size, channel):  
 # 利用滑动窗口尺寸先计算一个一维，并且服从正态分布的数据  
 # 注意这里利用unsqueeze函数扩了一下维度，从行向量变为了列向量  
 \_1D\_window = gaussian(window\_size, 1.5).unsqueeze(1)  
 # 列向量乘以行向量，变为n\*n的矩阵，正好对应窗口权重  
 \_2D\_window = \_1D\_window.mm(\_1D\_window.t()).float().unsqueeze(0).unsqueeze(0)  
 # 沿通道维度复制channel遍，每个通道对应一个权重(这里所有通道权重相同，均服从正态分布)，并且变为连续存储的数据  
 window = Variable(\_2D\_window.expand(channel, 1, window\_size, window\_size).contiguous())  
 # 返回窗口权重数据  
 return window  
  
  
def \_ssim(img1, img2, window, window\_size, channel, size\_average=True):  
 # 计算每个滑动窗口的均值  
 # 卷积运算正好是窗口数据按权重求和再取均值，因此可以利用二维卷积运算来计算窗口中数据的均值  
 mu1 = F.conv2d(img1, window, padding=window\_size // 2, groups=channel)  
 mu2 = F.conv2d(img2, window, padding=window\_size // 2, groups=channel)  
 # 均值取平方，即E^2(X)  
 mu1\_sq = mu1.pow(2)  
 mu2\_sq = mu2.pow(2)  
 # 计算E(X)E(Y)，用于后续计算协方差  
 mu1\_mu2 = mu1 \* mu2  
 # 依次计算img1与img2的方差  
 # 这里计算方差利用公式D(X)=E(X^2)-E^2(X)，其中E^2(X)表示均值的平方，即上述公式中的mu1\_sq、mu2\_sq、mu1\_mu2  
 sigma1\_sq = F.conv2d(img1 \* img1, window, padding=window\_size // 2, groups=channel) - mu1\_sq  
 sigma2\_sq = F.conv2d(img2 \* img2, window, padding=window\_size // 2, groups=channel) - mu2\_sq  
 # 计算img1、img2之间的协方差  
 # 利用公式Conv(X,Y)=E(XY)-E(X)E(Y)  
 sigma12 = F.conv2d(img1 \* img2, window, padding=window\_size // 2, groups=channel) - mu1\_mu2  
 C1 = 0.01 \*\* 2  
 C2 = 0.03 \*\* 2  
 # 利用上述得到的指标，传入公式计算ssim值，此时会得到一张图，最后再求均值即可  
 ssim\_map = ((2 \* mu1\_mu2 + C1) \* (2 \* sigma12 + C2)) / ((mu1\_sq + mu2\_sq + C1) \* (sigma1\_sq + sigma2\_sq + C2))  
  
 if size\_average:  
 return ssim\_map.mean()  
 else:  
 return ssim\_map.mean(1).mean(1).mean(1)  
  
  
def ssim(img1, img2, window\_size=11, size\_average=True):  
 # 将输入的图像数据限制为0-1之间(一般数据就是位于0-1之间，防止出现异常值)  
 img1 = torch.clamp(img1, min=0, max=1)  
 img2 = torch.clamp(img2, min=0, max=1)  
 # 得到图片通道数  
 (\_, channel, \_, \_) = img1.size()  
 # 得到窗口的权重数据，离窗口中心越远，权重越小。权重服从高斯分布(正态分布)  
 window = create\_window(window\_size, channel)  
 # 如果图片数据储存在cuda上(即利用显卡训练)，则将窗口权重数据也传入cuda中  
 if img1.is\_cuda:  
 window = window.cuda(img1.get\_device())  
 # 统一数据类型  
 window = window.type\_as(img1)  
 # 调用\_ssim，计算ssim值  
 return \_ssim(img1, img2, window, window\_size, channel, size\_average)  
  
from skimage.metrics import structural\_similarity  
  
  
# im1, im2分别表示参与计算的图像数据  
# data\_range表示图像数据的范围，一般设置为255或者1(如果对图像数据做了归一化操作，则为1)  
# channel\_axis表示颜色通道位于图像的第几维度，如果不指定的话，则默认输入灰度图像  
ssim = structural\_similarity(img1, img2, win\_size=None, gradient=False, data\_range=None,  
 channel\_axis=None, multichannel=False, gaussian\_weights=False, full=False)

第二段代码分析：

这段代码是用于计算两幅图像之间的结构相似性（Structural Similarity Index, SSIM）的Python代码。SSIM是一种常用的评价图像质量的指标，它通过比较图像的亮度、对比度和结构信息来衡量两幅图像的相似度。

代码分为几个部分：

1. `gaussian` 函数：用于生成一个高斯分布的权重数组。这个数组的中心点权重最高，向两边递减，符合高斯分布的特性。

2. `create\_window` 函数：用于创建一个二维的高斯权重窗口。这个窗口将用于后续的卷积操作，以计算图像的局部统计特性。

3. `\_ssim` 函数：这是计算SSIM的核心函数。它接受两个图像、高斯窗口、窗口大小、通道数和是否平均大小的标志。函数内部首先计算两个图像的局部均值、方差和协方差，然后根据SSIM的定义计算出SSIM值。

4. `ssim` 函数：这是一个封装函数，用于处理输入图像，创建高斯窗口，并调用 `\_ssim` 函数来计算SSIM值。

5. `structural\_similarity` 函数：这是 `skimage` 库中的一个函数，用于计算两幅图像之间的SSIM值。这个函数提供了更多的参数来控制SSIM的计算方式。

代码中有几个注意点：

- `torch.Tensor` 和 `torch.Variable`：在PyTorch中，`Tensor` 是用于存储数据的多维数组，而 `Variable` 是一个已经废弃的概念，用于自动微分。在最新版本的PyTorch中，`Tensor` 已经集成了 `Variable` 的功能。

- `F.conv2d`：这是一个二维卷积函数，用于计算图像和高斯窗口的卷积，从而得到图像的局部均值和方差。

- `torch.clamp`：这个函数用于将输入的图像数据限制在0到1之间，确保数据的有效性。

- `window\_size`：这是高斯窗口的大小，通常设置为11。

- `size\_average`：这个标志用于确定是否对SSIM值进行平均。

- `img1.is\_cuda`：这个条件判断用于检查图像数据是否存储在GPU上，如果是，则将高斯窗口也转移到GPU上。

- `window.type\_as(img1)`：这个操作确保高斯窗口的数据类型与图像数据一致。

最后，代码中的 `structural\_similarity` 函数调用了一些参数，但是没有正确地传递给函数。例如，`win\_size=None` 应该设置为一个具体的数值，如 `win\_size=11`，以匹配前面定义的高斯窗口大小。同样，`data\_range` 应该根据图像数据的范围进行设置，如果图像数据已经归一化到0-1之间，则 `data\_range=1`。

这段代码主要用于图像处理领域，特别是在图像质量评价和图像压缩等领域。通过计算两幅图像的SSIM值，可以评估图像之间的相似度，从而判断图像处理算法的效果。





