## 2.4.2 Sentinel-2及超像素级分割下高空分辨率特征尺度界定

### 2.4.2.1 [Sentinel-2 遥感影像](https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-2)①

Sentinel-2为高分辨率多光谱成像卫星，为2A和2B两颗卫星，分别于2015-06-23和2017-03-07日发射升空。每颗卫星重访周期为10天，两者为每5天完成一次对地球赤道地区的完整成像。卫星寿命为7.25年，其携带的多光谱器（ MultiSpectral Instrument，MSI），覆盖13个光谱波段，地面分辨率分别为10m、20m和60m。数据可以从[Copernicus Open Access Hub](https://scihub.copernicus.eu/dhus/#/home)②处下载。具体的波段解释与Landsat-8比较如下：

| Landset-8 |  |  | Sentinel-2 |  |  |
| --- | --- | --- | --- | --- | --- |
| Band | Wavelenght(nm) | Resolution(m) | Band | Wavelength(nm) | Resolution(m) |
| 1(Coastal) | 430-450 | 30 | 1(Coastal) | 433-453 | 60 |
| 2(Blue) | 450-515 | 30 | 2(Blue) | 458-523 | 10 |
| 3(Green) | 525-600 | 30 | 3(Green) | 543-578 | 10 |
| 4(Red) | 630-680 | 30 | 4(Red) | 650-680 | 10 |
|  |  |  | 5(red Edge) | 698-713 | 20 |
|  |  |  | 6(red Edge) | 733-748 | 20 |
|  |  |  | 7(red Edge) | 773-793 | 20 |
| 5(NIR) | 845-885 | 30 | 8(NIR) | 785-900 | 10 |
|  |  |  | 9(Water vapor) | 935-955 | 60 |
|  |  |  | 10(SWIR-Cirrus) | 1360-1390 | 60 |
| 6(SWIR-1) | 1560-1660 | 30 | 11(SWIR-1) | 1565-1655 | 20 |
| 7(SWIR-2) | 2100-2300 | 30 | 12(SWIR-2) | 2100-2280 | 20 |
| 8(PAN) | 503-676 | 15 |  |  |  |

Sentinel-2与Landset-8最大的区别除了各个波段的分辨率不同外，还有在近红外波段NIR与红色波段之间细分了Red Edge红边波段，这对检测植被健康信息非常有效。

Sentinel-2产品级别可以划分为：

Level-0，原始数据；

Level-1A，包含元信息的几何粗矫正产品；

Level-1B，辐射率产品，嵌入经GCP优化的几何模型，但未进行相应的几何校正；

Level-1C，经正射校正和亚像元级几何精校正后的大气表观反射率产品；

Level-2A，由Level-1C产品经过大气校正的大气底层反射率数据(Bottom Of Atmosphere (BOA) reflectance images derived from the associated Level-1C products)。

在由Level-1C生成Level-2A（即经辐射定标和大气校正），可以使用欧空局（European Space Agency，ESA）发布的[Sen2Cor](http://step.esa.int/main/snap-supported-plugins/sen2cor/)③工具，在Windows下的Command Prompt（CMD）终端下安装（执行L2A\_Process --help命令），并执行L2A\_Process +数据位置+参数（可选）。ESA发布的产品中混合有标识为“*MSIL1C*”的Level-1C产品，标识为“*MSIL2A*”的Level-2A产品，需要注意区分，Sen2Cor工具只对Level-1C产品执行大气校正产生Level-2A产品。

* [rio\_tiler库](https://cogeotiff.github.io/rio-tiler/) ④是[rasterio](https://rasterio.readthedocs.io/en/latest/)⑤的插件（plugin），用于从栅格数据集读取网页地图瓦片（web map tiles）。

#### 1） 以Web Mercator方式显示Sentinel-2的一个波段

import rio\_tiler  
help(rio\_tiler)

Help on package rio\_tiler:  
  
NAME  
 rio\_tiler - rio-tiler.  
  
PACKAGE CONTENTS  
 cmap\_data (package)  
 colormap  
 constants  
 errors  
 expression  
 io (package)  
 logger  
 models  
 mosaic (package)  
 profiles  
 reader  
 tasks  
 types  
 utils  
  
VERSION  
 3.1.6  
  
FILE  
 c:\users\richi\anaconda3\envs\usda\lib\site-packages\rio\_tiler\\_\_init\_\_.py

Web墨卡托投影（Web Mercator）是墨卡托投影的一种变体，是Web地图应用的事实标准。自2005年Google地图采用该投影之后，几乎所有的在线地图提供商都使用这一标准，包括[Google map](https://www.google.com/maps/@41.8305755,-87.6609536,14z)、 [Mapbox](https://www.mapbox.com/)、[Bing map](https://www.bing.com/maps)、 [OpenStreetMap](https://www.openstreetmap.org/#map=4/38.01/-95.84)、[MapQuest](https://www.mapquest.com/)和[Esri](https://www.esri.com/en-us/home)等⑥。其正式的EPSG标识符是EPSG:3857。

几个世纪以来，人们一直在使用坐标系统和地图投影将地球的形状转换成可用的平面地图。而世界地图很大，不能直接在电脑上显示，所以引出快速浏览和缩放地图的机制，地图瓦片（map tiles）。将世界划分为很多小方块，每个小方块都有固定的地理面积和规模。这样可以在不加载整个地图的情况下浏览其中的一小部分。这涉及到几种表示方法，大地坐标，投影系统，像素坐标和金子塔瓦片，及它们之间的相互转换。

1. 度（Degrees）， 用于Geodetic coordinates WGS84 (EPSG:4326)坐标系统。使用1984年定义的世界大地测量系统（World Geodetic System），GPS设备用于定义地球位置的经纬度坐标。
2. 米（Meters）， 用于Projected coordinates Spherical Mercator (EPSG:3857)坐标系统。全球投影坐标（Global projected coordinates ），用于GIS，A Web Map Tile Service （WM(T)S）服务的栅格瓦爿（raster tile）生成。
3. 像素（Pixels）， 用于Screen coordinates XY pixels at zoom坐标系统。影像金子塔每一层（each level of the pyramid）的特定缩放像素坐标。顶级（zoom=0）通常有像素，下一级为等。带有屏幕的设备（电脑，手机）等在定义的缩放级别计算像素坐标，并确定应该从服务器加载的区域用于可视屏幕。
4. 瓦片（Tiles）， 用于Tile coordinates Tile Map Service (ZXY)坐标系统。影像金子塔中指定缩放级别下（zoom level）瓦爿的索引，即x轴和y轴的位置/索引。每一级别下所有瓦片都有相同的尺寸，通常为像素。就是由粗到细不同分辨率的影像集合。其底部为图像的高分辨率表示，为原始图像，瓦片数应与原始图像的大小同；顶部为低分辨率的近似影像，最顶层只有1个瓦片，而后为4，16等。

**球面墨卡托投影金字塔的分辨率和比例列表**

| Zoom level | Resolution (meters / pixel) | Map Scale (at 96 dpi) | Width and Height of map (pixels) |
| --- | --- | --- | --- |
| 0 | 156,543.03 | 1 : 591,658,710.90 | 256 |
| 1 | 78,271.52 | 1 : 295,829,355.45 | 512 |
| 2 | 39,135.76 | 1 : 147,914,677.73 | 1,024 |
| 3 | 19,567.88 | 1 : 73,957,338.86 | 2,048 |
| 4 | 9,783.94 | 1 : 36,978,669.43 | 4,096 |
| 5 | 4,891.97 | 1 : 18,489,334.72 | 8,192 |
| 6 | 2,445.98 | 1 : 9,244,667.36 | 16,384 |
| 7 | 1,222.99 | 1 : 4,622,333.68 | 32,768 |
| 8 | 611.4962263 | 1 : 2,311,166.84 | 65,536 |
| 9 | 305.7481131 | 1 : 1,155,583.42 | 131,072 |
| 10 | 152.8740566 | 1 : 577,791.71 | 262,144 |
| 11 | 76.43702829 | 1 : 288,895.85 | 524,288 |
| 12 | 38.21851414 | 1 : 144,447.93 | 1,048,576 |
| 13 | 19.10925707 | 1 : 72,223.96 | 2,097,152 |
| 14 | 9.554728536 | 1 : 36,111.98 | 4,194,304 |
| 15 | 4.777314268 | 1 : 18,055.99 | 8,388,608 |
| 16 | 2.388657133 | 1 : 9,028.00 | 16,777,216 |
| 17 | 1.194328566 | 1 : 4,514.00 | 33,554,432 |
| 18 | 0.597164263 | 1 : 2,257.00 | 67,108,864 |
| 19 | 0.298582142 | 1 : 1,128.50 | 134,217,728 |
| 20 | 0.149291071 | 10:24.2 | 268,435,456 |
| 21 | 0.074645535 | 05:42.1 | 536,870,912 |
| 22 | 0.037322768 | 03:21.1 | 1,073,741,824 |
| 23 | 0.018661384 | 02:10.5 | 2,147,483,648 |

对于金字塔瓦片和坐标之间的转换可以查看[Tiles à la Google Maps](https://www.maptiler.com/google-maps-coordinates-tile-bounds-projection/)⑦，其中给出了转换的源代码。下述迁移的函数deg2num()可以将经纬度坐标转换为指定zoom level缩放级别下，金子塔中瓦片的坐标。其输入参数为经纬度坐标值和缩放级别。在使用rio\_tiler库COGReader方法读取栅格文件后，可以通过src.dataset、src.tms.identifier、src.minzoom、src.maxzoom、src.bounds、src.crs、src.geographic\_bounds和src.colormap等方式提取读取的栅格的属性值信息。

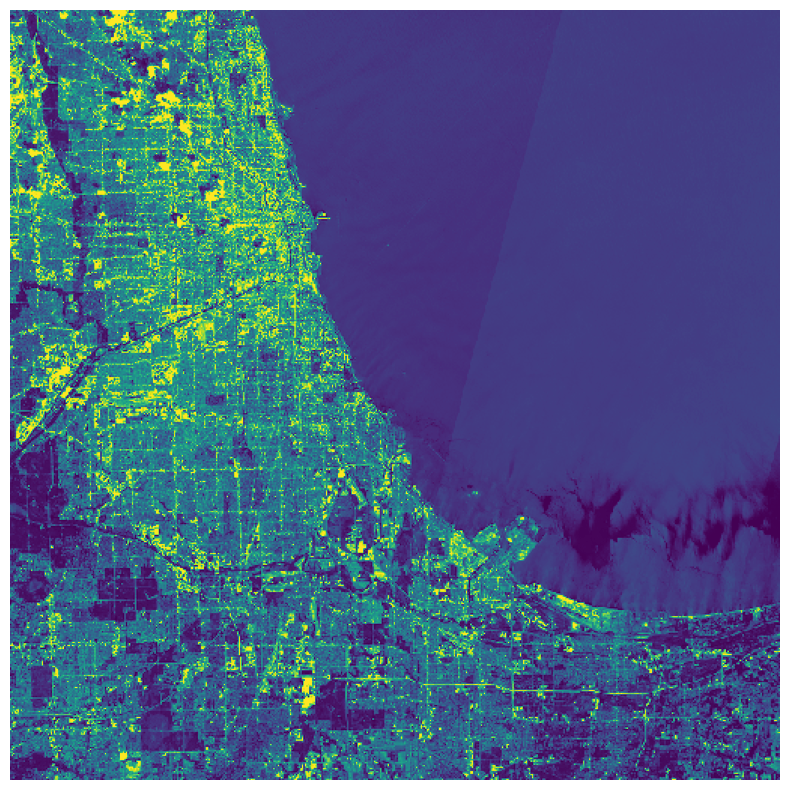
def deg2num(lat\_deg, lon\_deg, zoom):  
 '''  
 code migration  
 function - 将经纬度坐标转换为指定zoom level缩放级别下，金子塔中瓦片的坐标。  
   
 Params:  
 lat\_deg - 纬度；float  
 lon\_deg - 经度；float  
 zoom - 缩放级别；int  
   
 Returns:  
 xtile - 金子塔瓦片x坐标；int  
 ytile - 金子塔瓦片y坐标；int  
 '''  
 import math   
   
 lat\_rad = math.radians(lat\_deg)  
 n = 2.0 \*\* zoom  
 xtile = int((lon\_deg + 180.0) / 360.0 \* n)  
 ytile = int((1.0 - math.log(math.tan(lat\_rad) + (1 / math.cos(lat\_rad))) / math.pi) / 2.0 \* n)  
 return (xtile, ytile)  
  
def centroid(bounds):  
 '''  
 code migration  
 function - 根据获取的地图边界坐标[左下角经度，左下角纬度，右上角经度，右上角维度]计算中心点坐标  
   
 Params:  
 bounds - [左下角经度，左下角纬度，右上角经度，右上角维度]；numerical  
   
 Returns:  
 lat - 边界中心点维度；float  
 lng - 边界中心点经度；float  
 '''  
 lat=(bounds[1] + bounds[3]) / 2  
 lng=(bounds[0] + bounds[2]) / 2  
 return lat, lng

用src.bounds提取栅格边界（左下角经度，左下角纬度，右上角经度，右上角维度）时，为投影坐标；而用src.geographic\_bounds提取的边界为经纬度坐标。下述从COG中用src.tile()方法读取网络地图瓦片（Web Map tile），需要输入的参数tile\_x (int)，tile\_y (int)和tile\_z (int)为瓦片的水平、垂直和缩放级别索引，tilesize (int, optional)为输出图像的大小。

from rio\_tiler.io import COGReader  
from skimage import exposure  
import numpy as np  
  
scene\_id=r'G:\data\S2B\_MSIL2A\_20200709T163839\_N0214\_R126\_T16TDM\_20200709T211044.SAFE\GRANULE\L2A\_T16TDM\_A017455\_20200709T164859\IMG\_DATA\R10m\T16TDM\_20200709T163839\_B04\_10m.jp2'  
z=9 # 需要调整不同的缩放级别，查看显示结果，如果缩放级别过大，影像则会模糊，无法查看细节；如果缩放比例来过大，数据量增加，则会增加计算时长  
with COGReader(scene\_id) as src:  
 #print(help(image))  
 print('影像边界坐标：',src.geographic\_bounds)  
 x, y=deg2num(\*centroid(src.geographic\_bounds), z) # 指定缩放级别，转换影像中心点的经纬度坐标为金子塔瓦片坐标  
 print("影像中心点瓦片索引：",x\_,y\_)  
 img=src.tile(x,y,z, tilesize=512) # tilesize参数为瓦片大小，默认值为256  
tile=img.data   
# Move the colour dimension to the last axis  
tile=np.transpose(tile, (1, 2, 0))  
# Rescale the intensity to make a pretty picture for human eyes  
low, high=np.percentile(tile, (1, 97))  
tile=exposure.rescale\_intensity(tile, in\_range=(low, high)) / 65535  
# The tile only shows a subset of the whole image, which is several GB large  
print("瓦片的形状：",tile.shape)

影像边界坐标： (-88.21648331497754, 41.45751520585338, -86.88130582637692, 42.4526911672117)  
影像中心点瓦片索引： 647168 148  
瓦片的形状： (512, 512, 1)

import matplotlib.pyplot as plt  
  
plt.figure(figsize=(10,10))  
plt.imshow(tile)  
plt.axis("off")  
plt.show()

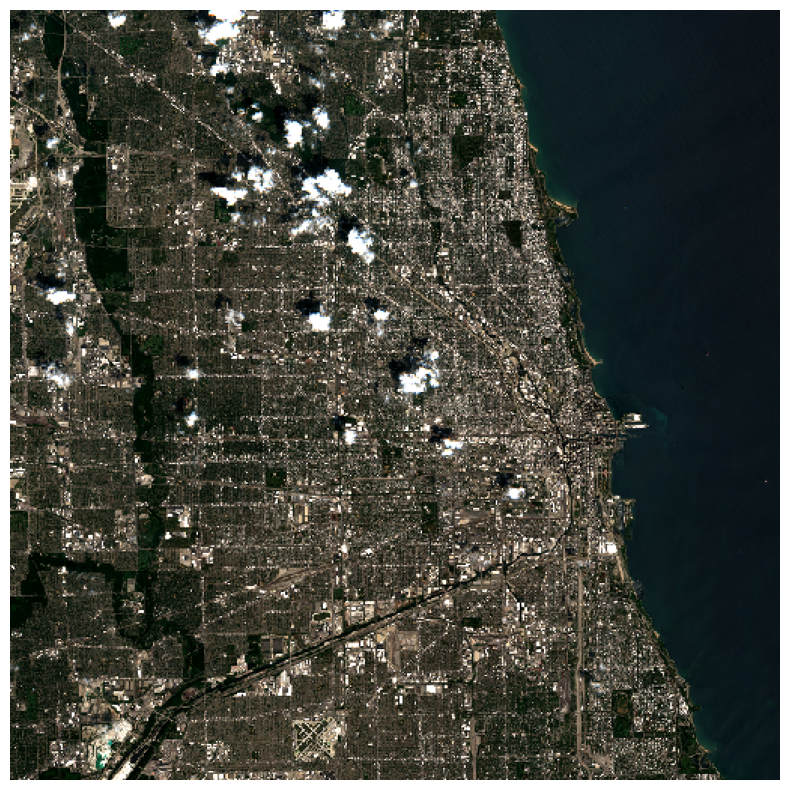


#### 2） Sentinel-2波段合成显示

波段合成显示的波段组合与Landsat部分阐述基本同，例如“4\_Red、3\_Green 、2\_Blue”波段组合为自然真彩色。

def Sentinel2\_bandsComposite\_show(RGB\_bands,zoom=10,tilesize=512,figsize=(10,10)):  
 '''  
 function - Sentinel-2波段合成显示。需要deg2num(lat\_deg, lon\_deg, zoom)和centroid(bounds)函数  
   
 Params:  
 RGB\_bands - 波段文件路径名字典，例如{"R":path\_R,"G":path\_G,"B":path\_B}；dict  
 zoom - zoom level缩放级别。The defalut is 10；int  
 tilesize - 瓦片大小。The default is 512；int  
 figsize- 打印图表大小。The default is (10,10)；tuple  
   
 Returns:  
 None  
 '''  
 %matplotlib inline  
 import matplotlib.pyplot as plt  
 import math,os  
 import numpy as np  
 from rio\_tiler.io import COGReader  
 from skimage import exposure  
 from rasterio.plot import show   
   
 B\_band=RGB\_bands["B"]  
 G\_band=RGB\_bands["G"]  
 R\_band=RGB\_bands["R"]  
   
 def band\_reader(band):  
 with COGReader(band) as image:  
 bounds=image.geographic\_bounds  
 print('影像边界坐标：',bounds)  
 x, y=deg2num(\*centroid(bounds), zoom)  
 print("影像中心点瓦片索引：",x,y)  
 img=image.tile(x, y, zoom, tilesize=tilesize)   
 return img.data  
   
 tile\_RGB\_list=[np.squeeze(band\_reader(band)) for band in RGB\_bands.values()]  
 tile\_RGB\_array=np.array(tile\_RGB\_list).transpose(1,2,0)  
 p2, p98=np.percentile(tile\_RGB\_array, (2,98))  
 image=exposure.rescale\_intensity(tile\_RGB\_array, in\_range=(p2, p98)) / 65535  
   
 plt.figure(figsize=(10,10))  
 plt.imshow(image)  
 plt.axis("off")  
 plt.show()  
   
import os  
sentinel2\_root=r"G:\data\S2B\_MSIL2A\_20200709T163839\_N0214\_R126\_T16TDM\_20200709T211044.SAFE\GRANULE\L2A\_T16TDM\_A017455\_20200709T164859\IMG\_DATA\R10m"   
RGB\_bands={   
 "R":os.path.join(sentinel2\_root,'T16TDM\_20200709T163839\_B04\_10m.jp2'),  
 "G":os.path.join(sentinel2\_root,'T16TDM\_20200709T163839\_B03\_10m.jp2'),  
 "B":os.path.join(sentinel2\_root,'T16TDM\_20200709T163839\_B02\_10m.jp2'),}  
  
Sentinel2\_bandsComposite\_show(RGB\_bands)

影像边界坐标： (-88.21648331497754, 41.45751520585338, -86.88130582637692, 42.4526911672117)  
影像中心点瓦片索引： 262 380  
影像边界坐标： (-88.21648331497754, 41.45751520585338, -86.88130582637692, 42.4526911672117)  
影像中心点瓦片索引： 262 380  
影像边界坐标： (-88.21648331497754, 41.45751520585338, -86.88130582637692, 42.4526911672117)  
影像中心点瓦片索引： 262 380



### 2.4.2.2 超像素级分割下高空分辨率特征尺度界定

在景观生态学中，斑块－廊道－基质模型是构成景观空间结构，描述景观空间异质性的一个基本模式。其中斑块是景观格局中的基本组成单元，是指不同于周围背景，相对均质的非线性区域。自然界各种等级系统都普遍存在时间和空间的斑块化。反应系统内部和系统间的相似性或相异性。不同斑块的大小、形状、边界性质及斑块的距离等空间分布特征构成了不同的生态带，形成了生态系统的差异，调节生态过程。廊道是不同于景观基质的现状或带状的景观要素，例如河流廊道，生态廊道等。其中生态廊道又称野生动物生态廊道或绿色廊道，是指用于连接因人类活动或构筑物而被隔开的野生动物种群生境的区域。生态廊道有利于野生动物的迁移扩散，提高生境间的连接，促进濒危物种不同群间的基因交流，降低种群灭绝风险。基质则是景观中面积最大，连接性最好的景观要素类型。斑廊基景观空间结构的提出为城市格局规划提供了依据，在宏观尺度上给出了保护自然生物的空间形式[1]。那么对于一个区域，如何自然界定斑块、廊道和基质的区域？或者即使是一个可以肉眼辨识的斑块，这个斑块自身也是呈现变化的，可以表现在地物的变化，例如可见的不同林地，不同物种的农田等，或者不可见的地表温度变化，物质流动等，那么又如何细分斑块的空间区域，挖掘斑块变化区域的流动方向或子区域？

一方面需要能够反映地物变化的信息数据，例如遥感影像的各个波段对不同地物的探测，Sentinel-2影像中新增加的5、6、7波段（red edge），可以有效监测植被健康信息；或衍生数据，如反演的地表温度，及NDVI等反应植被分布的指数，NDWI反应水体分布的指数，NDBI反应建城区分布的指数等。另一方面在分析这些数据时，可以介入超像素级分割的概念，探索由像素（或空间点数据）局部分组形成的区域，这类似于聚类的方法，将具有同一或近似属性的区域优先聚集即分割，分割区域的变化根据所提供反应不同内容的数据确定，例如探索植被分布的NDVI则优先聚集植被指数临近的区域。也可以组合波段，例如red、green和blue波段组合更倾向于优先聚集同一地物，例如建筑区域，林地区域等。

超像素级分割[2]是一种语义分割，是计算机视觉的基本方法，可以更加精准的执行地物分割、探测和分类等深度学习任务。这一方法也同样为景观、生态专业探索地物变化和地物之间的关系提供一种新的策略。[scikit-image](https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_segmentations.html#sphx-glr-auto-examples-segmentation-plot-segmentations-py)⑧提供了四种分割的算法。在下述实验中计算了Felzenszwalb和Quickshift两种方法。Felzenszwalb方法在分割图像时，虽然逐级增加scale参数大小，但是分割的图像并不是上级区域覆盖下级区域；而Quickshift方法，则基本是逐级覆盖的，因此选用Quickshift方法，指定逐级增加的kernel\_size参数，获取不同深度分割的结果。通过计算逐级覆盖的分割类别频数统计，及方差等指数，试图找到研究区域不同深度分割下区域间的关联，及区域的差异性程度。

* 01 - 读取所下载的Sentinel-2影像的元文件，获取影像波段路径

Sentinel-2影像的信息均记录于下载文件夹下的“MTD\_MSIL2A.xml”中，因此可以从该文件获取各个波段的路径。该文件给出的路径为相对于影像文件夹的相对路径。

def Sentinel2\_bandFNs(MTD\_MSIL2A\_fn):   
 '''  
 funciton - 获取sentinel-2波段文件路径，和打印主要信息  
   
 Params:  
 MTD\_MSIL2A\_fn - MTD\_MSIL2A 文件路径；string  
   
 Returns:  
 band\_fns\_list - 波段相对路径列表；list(string)  
 band\_fns\_dict - 波段路径为值，反应波段信息的字段为键的字典；dict  
 '''  
 import xml.etree.ElementTree as ET  
   
 Sentinel2\_tree=ET.parse(MTD\_MSIL2A\_fn)  
 Sentinel2\_root=Sentinel2\_tree.getroot()  
  
 print("GENERATION\_TIME:{}\nPRODUCT\_TYPE:{}\nPROCESSING\_LEVEL:{}".format(Sentinel2\_root[0][0].find('GENERATION\_TIME').text,  
 Sentinel2\_root[0][0].find('PRODUCT\_TYPE').text,   
 Sentinel2\_root[0][0].find('PROCESSING\_LEVEL').text  
 ))   
 print("MTD\_MSIL2A.xml 文件父结构:")  
 for child in Sentinel2\_root:  
 print(child.tag,"-",child.attrib)  
 print("\_"\*50)   
 band\_fns\_list=[elem.text for elem in Sentinel2\_root.iter('IMAGE\_FILE')] #[elem.text for elem in Sentinel2\_root[0][0][11][0][0].iter()]  
 band\_fns\_dict={f.split('\_')[-2]+'\_'+f.split('\_')[-1]:f+'.jp2' for f in band\_fns\_list}  
 print('获取sentinel-2波段文件路径:\n',band\_fns\_dict)  
   
 return band\_fns\_list,band\_fns\_dict  
   
MTD\_MSIL2A\_fn=r'G:\data\S2B\_MSIL2A\_20200709T163839\_N0214\_R126\_T16TDM\_20200709T211044.SAFE\MTD\_MSIL2A.xml'  
band\_fns\_list,band\_fns\_dict=Sentinel2\_bandFNs(MTD\_MSIL2A\_fn)

GENERATION\_TIME:2020-07-09T21:10:44.000000Z  
PRODUCT\_TYPE:S2MSI2A  
PROCESSING\_LEVEL:Level-2A  
MTD\_MSIL2A.xml 文件父结构:  
{https://psd-14.sentinel2.eo.esa.int/PSD/User\_Product\_Level-2A.xsd}General\_Info - {}  
{https://psd-14.sentinel2.eo.esa.int/PSD/User\_Product\_Level-2A.xsd}Geometric\_Info - {}  
{https://psd-14.sentinel2.eo.esa.int/PSD/User\_Product\_Level-2A.xsd}Auxiliary\_Data\_Info - {}  
{https://psd-14.sentinel2.eo.esa.int/PSD/User\_Product\_Level-2A.xsd}Quality\_Indicators\_Info - {}  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
获取sentinel-2波段文件路径:  
 {'B02\_10m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R10m/T16TDM\_20200709T163839\_B02\_10m.jp2', 'B03\_10m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R10m/T16TDM\_20200709T163839\_B03\_10m.jp2', 'B04\_10m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R10m/T16TDM\_20200709T163839\_B04\_10m.jp2', 'B08\_10m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R10m/T16TDM\_20200709T163839\_B08\_10m.jp2', 'TCI\_10m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R10m/T16TDM\_20200709T163839\_TCI\_10m.jp2', 'AOT\_10m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R10m/T16TDM\_20200709T163839\_AOT\_10m.jp2', 'WVP\_10m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R10m/T16TDM\_20200709T163839\_WVP\_10m.jp2', 'B02\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_B02\_20m.jp2', 'B03\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_B03\_20m.jp2', 'B04\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_B04\_20m.jp2', 'B05\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_B05\_20m.jp2', 'B06\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_B06\_20m.jp2', 'B07\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_B07\_20m.jp2', 'B8A\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_B8A\_20m.jp2', 'B11\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_B11\_20m.jp2', 'B12\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_B12\_20m.jp2', 'TCI\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_TCI\_20m.jp2', 'AOT\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_AOT\_20m.jp2', 'WVP\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_WVP\_20m.jp2', 'SCL\_20m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R20m/T16TDM\_20200709T163839\_SCL\_20m.jp2', 'B01\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B01\_60m.jp2', 'B02\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B02\_60m.jp2', 'B03\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B03\_60m.jp2', 'B04\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B04\_60m.jp2', 'B05\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B05\_60m.jp2', 'B06\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B06\_60m.jp2', 'B07\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B07\_60m.jp2', 'B8A\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B8A\_60m.jp2', 'B09\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B09\_60m.jp2', 'B11\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B11\_60m.jp2', 'B12\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_B12\_60m.jp2', 'TCI\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_TCI\_60m.jp2', 'AOT\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_AOT\_60m.jp2', 'WVP\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_WVP\_60m.jp2', 'SCL\_60m': 'GRANULE/L2A\_T16TDM\_A017455\_20200709T164859/IMG\_DATA/R60m/T16TDM\_20200709T163839\_SCL\_60m.jp2'}

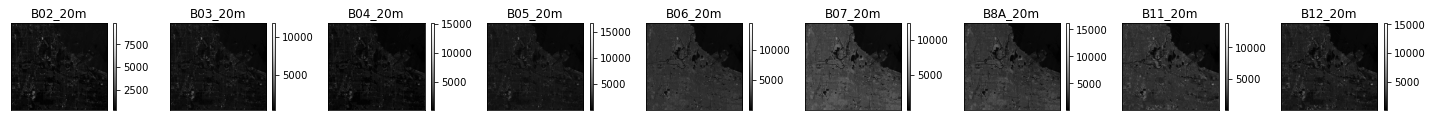
* 02 - 裁切到研究区域。裁切边界由QGIS绘制。

import util\_A  
import os  
raster\_fp=[os.path.join(r"G:\data\S2B\_MSIL2A\_20200709T163839\_N0214\_R126\_T16TDM\_20200709T211044.SAFE",band\_fns\_dict[k]) for k in band\_fns\_dict.keys() if k.split("\_")[-1]=="20m"]  
clip\_boundary\_fp=r'.\data\superPixel\_boundary\superPixel\_boundary.shp'  
save\_path=r'G:\data\_processed\RSi\crop\_20'  
util\_A.raster\_clip(raster\_fp,clip\_boundary\_fp,save\_path)

finished clipping.  
  
  
  
  
  
['G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_B02\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_B03\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_B04\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_B05\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_B06\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_B07\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_B8A\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_B11\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_B12\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_TCI\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_AOT\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_WVP\_20m\_crop.jp2',  
 'G:\\data\\RSi\\crop\_20\\T16TDM\_20200709T163839\_SCL\_20m\_crop.jp2']

* 03 - 读取裁切后的影像，显示查看。实验中仅分析了red、 blue和green波段组合。可以再深入分析red edge波段对植物的分割，及计算NDVI，NDWI，NDBI，或反演地表温度来进一步研究不同信息数据分割结果。

import glob  
import earthpy.spatial as es # conda install -c conda-forge geopandas ; pip install earthpy; conda install -c conda-forge earthpy   
import earthpy.plot as ep  
import matplotlib.pyplot as plt  
  
save\_path=r'G:\data\_processed\RSi\crop\_20'  
croppedImgs\_fns=glob.glob(save\_path+"/\*.jp2")  
croppedBands\_fnsDict={f.split('\_')[-3]+'\_'+f.split('\_')[-2]:f for f in croppedImgs\_fns}  
  
bands\_selection\_=['B02\_20m', 'B03\_20m', 'B04\_20m','B05\_20m', 'B06\_20m', 'B07\_20m', 'B8A\_20m', 'B11\_20m', 'B12\_20m'] # 'TCI\_20m', 'AOT\_20m', 'WVP\_20m', 'SCL\_20m'  
cropped\_stack\_bands=[croppedBands\_fnsDict[b] for b in bands\_selection\_]  
  
cropped\_array\_stack,\_=es.stack(cropped\_stack\_bands)  
ep.plot\_bands(cropped\_array\_stack,title=bands\_selection\_,cols=cropped\_array\_stack.shape[0],cbar=True,figsize=(20,10))  
plt.show()



import matplotlib.pyplot as plt  
import numpy as np  
  
from skimage.color import rgb2gray  
from skimage.filters import sobel  
from skimage.segmentation import felzenszwalb, slic, quickshift, watershed  
from skimage.segmentation import mark\_boundaries  
from skimage.util import img\_as\_float  
import pickle  
  
img=cropped\_array\_stack[[2,1,0]]  
img=img.transpose(1,2,0)

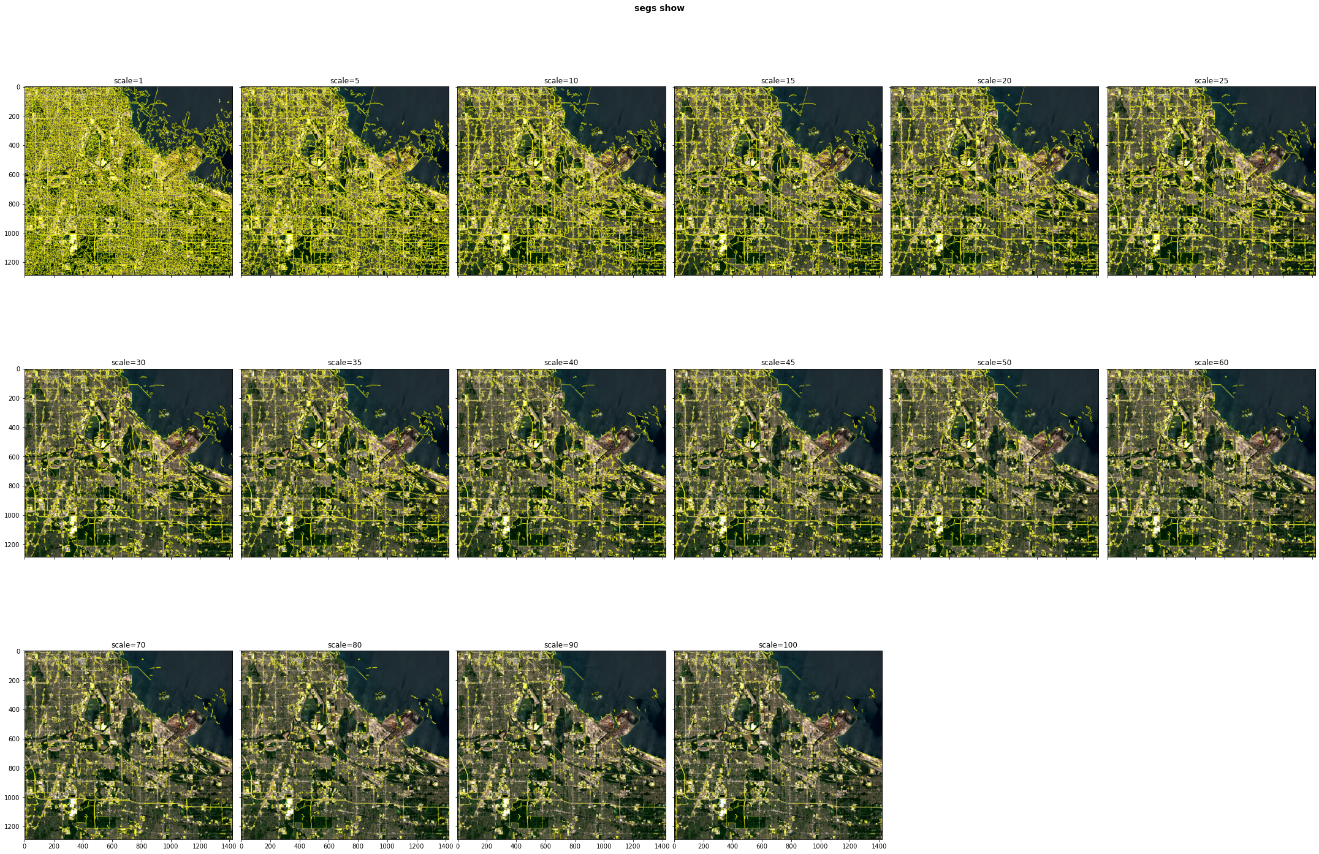
* 04 - Felzenszwalb 超像素级分割方法

def superpixel\_segmentation\_Felzenszwalb(img,scale\_list,sigma=0.5, min\_size=50):  
 '''  
 function - 超像素分割，skimage库felzenszwalb方法。给定scale参数列表，批量计算  
   
 Params:  
 img - 读取的遥感影像、图像；ndarray  
 scale\_list - 分割比例列表；list(float)  
 sigma - Width (standard deviation) of Gaussian kernel used in preprocessing.The default is 0.5； float  
 min\_size - Minimum component size. Enforced using postprocessing. The default is 50； int  
   
 Returns:  
 分割结果。Integer mask indicating segment labels；ndarray  
 '''  
 import numpy as np  
 from skimage.segmentation import felzenszwalb  
 from tqdm import tqdm # conda install -c conda-forge tqdm ;conda install -c conda-forge ipywidgets   
   
 segments=[felzenszwalb(img, scale=s, sigma=sigma, min\_size=min\_size) for s in tqdm(scale\_list)]  
 return np.stack(segments)  
   
scale\_list=[1,5,10,15,20,25,30,35,40,45,50,60,70,80,90,100]   
segs=superpixel\_segmentation\_Felzenszwalb(img,scale\_list)  
  
with open('./data\_processed/segs\_superpixel.pkl','wb') as f:  
 pickle.dump(segs,f)

100%|██████████| 16/16 [00:55<00:00, 3.49s/it]

* 05 - 显示分割图像，分割边界。

import math  
from skimage import exposure  
  
scale\_list=[1,5,10,15,20,25,30,35,40,45,50,60,70,80,90,100]  
with open('./data\_processed/segs\_superpixel.pkl','rb') as f:  
 segs=pickle.load(f)  
  
p2, p98=np.percentile(img, (2,98))  
img\_=exposure.rescale\_intensity(img, in\_range=(p2, p98)) / 65535  
  
def markBoundaries\_layoutShow(segs\_array,img,columns,titles,prefix,figsize=(15,10)):  
 '''  
 function - 给定包含多个图像分割的一个数组，排布显示分割图像边界。  
  
 Paras:  
 segs\_array - 多个图像分割数组；ndarray  
 img - 底图 ；ndarray  
 columns - 列数；int  
 titles - 子图标题；string  
 figsize - 图表大小。The default is (15,10)；tuple  
   
 Returns:  
 None  
 '''   
 import math,os  
 import matplotlib.pyplot as plt  
 from PIL import Image  
 from skimage.segmentation import mark\_boundaries   
   
 rows=math.ceil(segs\_array.shape[0]/columns)  
 fig,axes=plt.subplots(rows,columns,sharex=True,sharey=True,figsize=figsize) # 布局多个子图，每个子图显示一幅图像  
 ax=axes.flatten() # 降至1维，便于循环操作子图  
 for i in range(segs\_array.shape[0]):  
 ax[i].imshow(mark\_boundaries(img, segs\_array[i])) # 显示图像  
 ax[i].set\_title("{}={}".format(prefix,titles[i]))  
 invisible\_num=rows\*columns-len(segs\_array)  
 if invisible\_num>0:  
 for i in range(invisible\_num):  
 ax.flat[-(i+1)].set\_visible(False)  
 fig.tight\_layout() # 自动调整子图参数，使之填充整个图像区域   
 fig.suptitle("segs show",fontsize=14,fontweight='bold',y=1.02)  
 plt.show()  
   
columns=6   
markBoundaries\_layoutShow(segs,img\_,columns,scale\_list,'scale',figsize=(30,20))

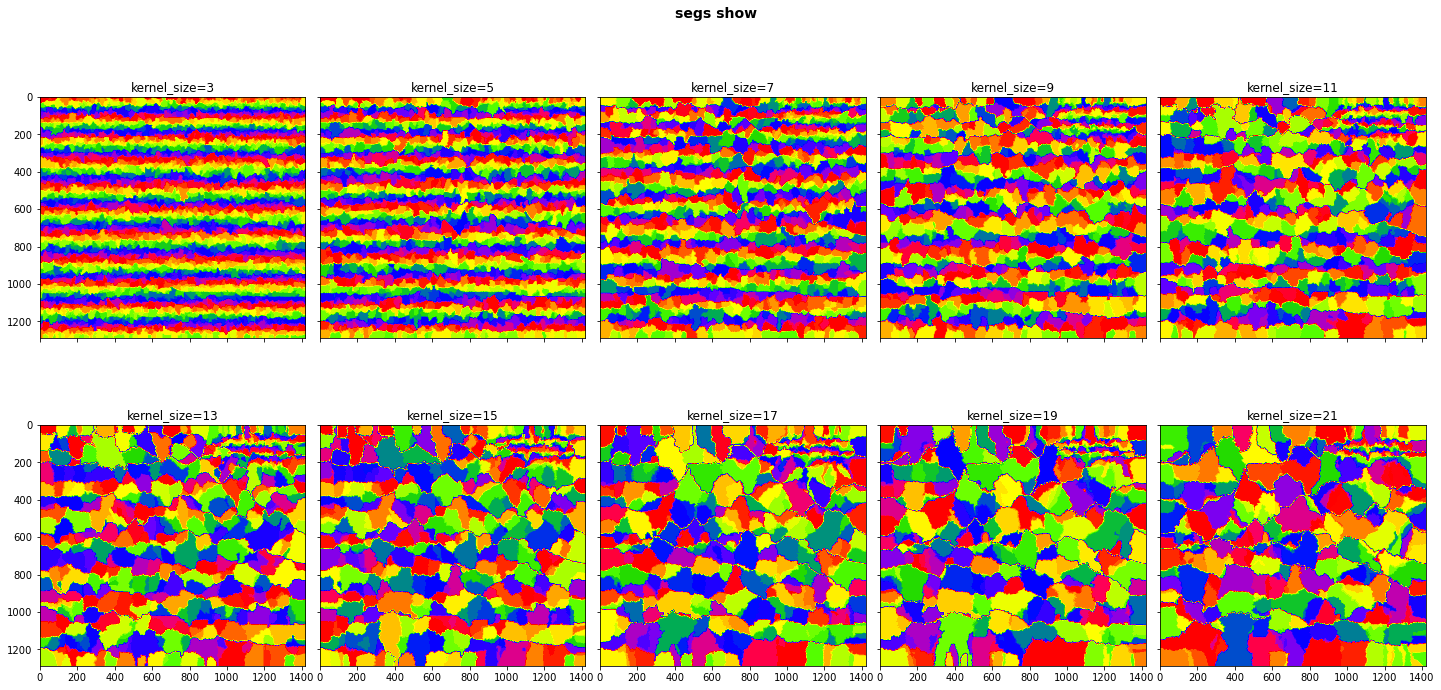


* 06 - Quickshift 超像素级分割方法，及显示。

def superpixel\_segmentation\_quickshift(img,kernel\_sizes, ratio=0.5):  
 '''  
 function - 超像素分割，skimage库quickshift方法。给定kernel\_size参数列表，批量计算  
   
 Params:  
 img - Input image. The axis corresponding to color channels can be specified via the channel\_axis argument；ndarray  
 kernel\_sizes - Width of Gaussian kernel used in smoothing the sample density. Higher means fewer clusters；float, optional  
 ratio - Balances color-space proximity and image-space proximity. Higher values give more weight to color-space. The default is 0.5；float, optional, between 0 and 1  
   
 Returns:  
 Integer mask indicating segment labels.  
 '''  
 import numpy as np  
 from skimage.segmentation import quickshift  
 from tqdm import tqdm # conda install -c conda-forge tqdm ;conda install -c conda-forge ipywidgets   
   
 segments=[quickshift(img, kernel\_size=k,ratio=ratio) for k in tqdm(kernel\_sizes)]  
 return np.stack(segments)  
   
kernel\_sizes=[3,5,7,9,11,13,15,17,19,21]   
segs\_quickshift=superpixel\_segmentation\_quickshift(img,kernel\_sizes)  
  
with open('./data\_processed/segs\_superpixel\_quickshift.pkl','wb') as f:  
 pickle.dump(segs\_quickshift,f)

100%|██████████| 10/10 [32:29<00:00, 194.94s/it]

def segMasks\_layoutShow(segs\_array,columns,titles,prefix,cmap='prism',figsize=(20,10)):  
 '''  
 function - 给定包含多个图像分割的一个数组，排布显示分割图像掩码。  
  
 Paras:  
 segs\_array - 多个图像分割数组；ndarray  
 columns - 列数；int  
 titles - 子图标题；string  
 figsize - 图表大小。The default is (20,10)；tuple(int)  
 '''   
 import math,os  
 import matplotlib.pyplot as plt  
 from PIL import Image  
   
 rows=math.ceil(segs\_array.shape[0]/columns)  
 fig,axes=plt.subplots(rows,columns,sharex=True,sharey=True,figsize=figsize) # 布局多个子图，每个子图显示一幅图像  
 ax=axes.flatten() # 降至1维，便于循环操作子图  
 for i in range(segs\_array.shape[0]):  
 ax[i].imshow(segs\_array[i],cmap=cmap) # 显示图像  
 ax[i].set\_title("{}={}".format(prefix,titles[i]))  
 invisible\_num=rows\*columns-len(segs\_array)  
 if invisible\_num>0:  
 for i in range(invisible\_num):  
 ax.flat[-(i+1)].set\_visible(False)  
 fig.tight\_layout() # 自动调整子图参数，使之填充整个图像区域   
 fig.suptitle("segs show",fontsize=14,fontweight='bold',y=1.02)  
 plt.show()  
   
columns=5  
segMasks\_layoutShow(segs\_quickshift,columns,kernel\_sizes,'kernel\_size')



* 08 - 多尺度超像素级分割结果叠合频数统计，包括各个层级与其之后所有层级间的计算。

def multiSegs\_stackStatistics(segs,save\_fp):  
 '''  
 function - 多尺度超像素级分割结果叠合频数统计  
   
 Params:  
 segs - 超级像素分割结果。Integer mask indicating segment labels；ndarray（int）  
 save\_fp - 保存路径名（pickle）；string  
   
 Returns:  
 stack\_statistics - 统计结果字典；dict  
 '''  
 from scipy.ndimage import label  
 from tqdm import tqdm   
   
 segs=list(reversed(segs))  
 stack\_statistics={}  
 for i in tqdm(range(len(segs)-1)):  
 labels=np.unique(segs[i])  
 coords=[np.column\_stack(np.where(segs[i]==k)) for k in labels]  
 i\_j={}  
 for j in range(i+1,len(segs)):  
 j\_k={}  
 for k in range(len(coords)):  
 covered\_elements=[segs[j][x,y] for x,y in zip(\*coords[k].T)]  
 freq=list(zip(np.unique(covered\_elements, return\_counts=True)))  
 j\_k[k]=freq  
 i\_j[j]=j\_k  
   
 stack\_statistics[i]=i\_j  
 with open(save\_fp,'wb') as f:  
 pickle.dump(stack\_statistics,f)  
   
 return stack\_statistics  
   
stack\_statistics=multiSegs\_stackStatistics(segs\_quickshift,'./data\_processed/multiSegs\_stackStatistics.pkl')

100%|██████████| 9/9 [01:09<00:00, 7.78s/it]

* 09 - 读取保存的分割层级叠合频数统计文件，提取卷积核即kernel\_size最大的层级与之后所有层级的频数统计，转换为DataFrame数据格式。

from tqdm import tqdm  
import pickle  
  
with open('./data\_processed/multiSegs\_stackStatistics.pkl','rb') as f:  
 stack\_statistics=pickle.load(f)  
segsOverlay\_0\_num={k:[stack\_statistics[0][k][i][0][0].shape[0] for i in stack\_statistics[0][k].keys()] for k in tqdm(stack\_statistics[0].keys())}

100%|██████████| 9/9 [00:00<00:00, 3008.11it/s]

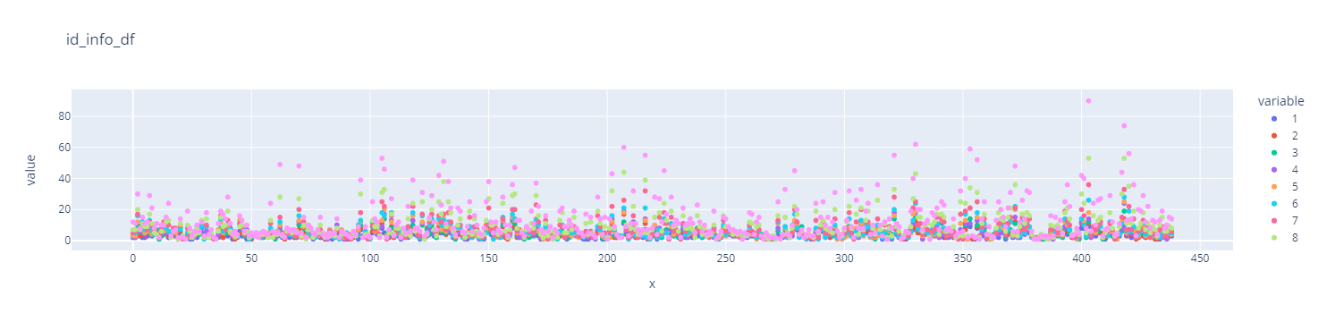
import pandas as pd  
segsOverlay\_0\_num\_df=pd.DataFrame.from\_dict(segsOverlay\_0\_num)  
segsOverlay\_0\_num\_df

|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2 | 3 | 5 | 3 | 3 | 5 | 5 | 7 | 12 |
| **1** | 2 | 2 | 2 | 2 | 3 | 5 | 6 | 7 | 11 |
| **2** | 6 | 8 | 10 | 12 | 12 | 16 | 17 | 20 | 30 |
| **3** | 3 | 3 | 6 | 6 | 7 | 7 | 8 | 7 | 8 |
| **4** | 2 | 4 | 5 | 7 | 8 | 12 | 10 | 11 | 15 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **434** | 3 | 3 | 3 | 4 | 5 | 4 | 9 | 15 | 19 |
| **435** | 4 | 2 | 3 | 2 | 3 | 3 | 3 | 6 | 8 |
| **436** | 2 | 2 | 3 | 1 | 3 | 4 | 3 | 6 | 10 |
| **437** | 3 | 3 | 4 | 5 | 6 | 9 | 8 | 9 | 15 |
| **438** | 3 | 4 | 4 | 4 | 4 | 5 | 6 | 8 | 14 |

439 rows × 9 columns

* 10 - 显示所有对应深度层级的频数变化。

import plotly.express as px  
x=list(segsOverlay\_0\_num\_df.index)  
y=list(segsOverlay\_0\_num\_df.columns)  
fig = px.scatter(segsOverlay\_0\_num\_df, x=x, y=y,  
 #hover\_data=[],  
 title='id\_info\_df'  
 )  
fig.show()



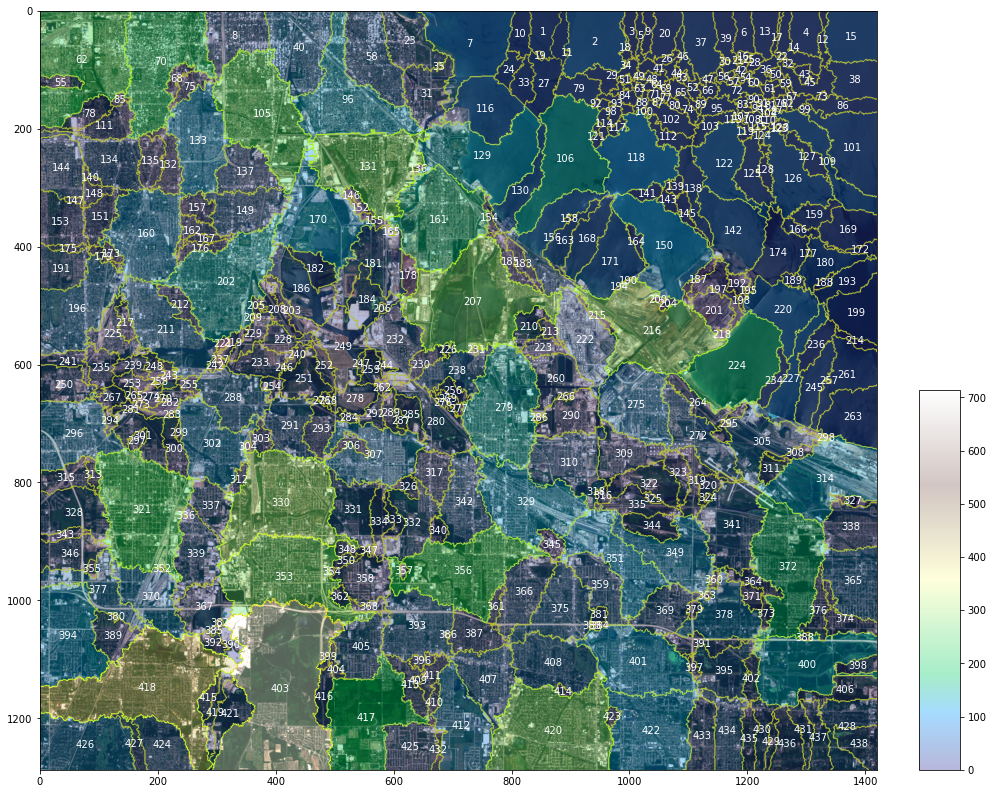
* 11 - 计算对应所有层级，父级分割（kernel\_size最大的层级）每一分割类，在各个深度层级上对应覆盖分割类数量的方差统计。可以分析父级每一个分割区域内深度层级下破碎（父级分割区域内子层分割的种类数量）的变化情况，如果值越大，往往父级分割区域内的“斑块”破碎化程度比较高，即区域内具有明显的异质性（差异性）；如果方差值越小，则说明父级分割区域内“斑块”属性基本近似，区域同质性。

将计算结果叠合到分割图上，以颜色显示方差变化，方便对应地理空间位置，并观察邻域间的情况。

import numpy as np  
  
var=segsOverlay\_0\_num\_df.var(axis=1)  
var\_dict=var.to\_dict()  
  
seg\_old=np.copy(segs\_quickshift[-1])  
seg\_new=np.copy(seg\_old).astype(float)  
for old,new in var\_dict.items():  
 seg\_new[seg\_old==old]=new

from skimage.measure import regionprops  
regions=regionprops(segs\_quickshift[-1])  
seg\_centroids={}  
for props in regions:  
 seg\_centroids[props.label]=props.centroid  
  
x,y=zip(\*seg\_centroids.values())  
labels=seg\_centroids.keys()

import matplotlib.pyplot as plt  
from mpl\_toolkits.axes\_grid1.inset\_locator import inset\_axes  
from skimage import exposure  
  
p2, p98=np.percentile(img, (2,98))  
img\_=exposure.rescale\_intensity(img, in\_range=(p2, p98)) / 65535  
  
fig,ax=plt.subplots(1,1,frameon=False,figsize=(15,15))   
im1=ax.imshow(mark\_boundaries(img\_, segs\_quickshift[-1]))  
im2=ax.imshow(seg\_new,cmap='terrain',alpha=.35)  
  
for k,coordi in seg\_centroids.items():  
 label=ax.text(x=coordi[1] ,y=coordi[0], s=k,ha='center', va='center',color='white')  
  
axins = inset\_axes(ax,  
 width="5%",   
 height="50%",   
 loc='lower left',  
 bbox\_to\_anchor=(1.05, 0., 1, 1),  
 bbox\_transform=ax.transAxes,  
 borderpad=0,  
 )  
fig.colorbar(im2, cax=axins)  
plt.show()



注释（Notes）：

① Sentinel-2 遥感影像，Sentinel-2由两颗极地轨道卫星组成，被置于同一个太阳同步轨道上，彼此相距180°。它的目的是监测陆地表面状况的变化，其宽幅扫描（290公里）和高重访时间（在赤道上用一颗卫星10天，在无云条件下用两颗卫星5天，在中纬度地区为2-3天）将支持监测地球表面的变化（<https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-2>）。

② Copernicus Open Access Hub，提供完整、免费和开放的Sentinel-1、Sentinel-2、 Sentinel-3 和 Sentinel-5P的用户产品（<https://scihub.copernicus.eu/dhus/#/home>）。

③ Sen2Cor，是一个用于Sentinel-2 2A级产品生成和格式化的处理器，对顶部大气层1C级输入数据进行大气、地形和卷云校正。Sen2Cor创建大气底层、可选择的地形和卷云校正的反射率图像，及气溶胶光学厚度、水汽、场景分类图以及云和雪概率的质量指标。它的输出产品格式等同于1C级用户产品：PEG 2000图像，三种不同的分辨率，60、20和10米（<http://step.esa.int/main/snap-supported-plugins/sen2cor/>）。

④ rio\_tiler库，最初被设计用来从大型栅格数据源中创建滑动地图瓦片（slippy map tiles），并在网络地图上动态渲染这些瓦片。从rio-tiler v2.0开始，则增加了许多辅助方法，从Rasterio/GDAL支持的任何栅格源中读取数据和元数据。这包括通过HTTP、AWS S3、谷歌云存储的本地和远程文件等。（<https://cogeotiff.github.io/rio-tiler/>）。

⑤ rasterio，地理信息系统（Geographic information systems，GIS）使用GeoTIFF和其它格式组织和存储栅格数据（raster datasets），如卫星影像和地形模型等。rasterio读写这些文件格式，并提供基于NumPy N-维数组和GeoJSON的Python API（<https://rasterio.readthedocs.io/en/latest/>）。

⑥ Google map（<https://www.google.com/maps/@41.8305755,-87.6609536,14z>）、 Mapbox（<https://www.mapbox.com/>）、Bing map（<https://www.bing.com/maps>）、 OpenStreetMap（<https://www.openstreetmap.org/#map=4/38.01/-95.84>）、MapQuest（<https://www.mapquest.com/>）和Esri（<https://www.esri.com/en-us/home>）。

⑦ Tiles à la Google Maps，坐标、瓦片界限和投影（Coordinates, Tile Bounds and Projection）。了解可缩放地图如何工作，什么是坐标系，及如何在它们之间转换（<https://www.maptiler.com/google-maps-coordinates-tile-bounds-projection>）。

⑧ scikit-image，分割和超级像素算法的比较（Comparison of segmentation and superpixel algorithms）（<https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_segmentations.html#sphx-glr-auto-examples-segmentation-plot-segmentations-py>）。

参考文献（References）:

[1] Robert E.Ricklefs著. 孙儒泳等译.生态学[M].高等教育出版社.北京.2004.7 第5版. （Robert E.Ricklefs. The economy of nature[M]. New York:W. H. Freeman and Company,2008, sixth edition.）

[2] Felzenszwalb, P. F. & Huttenlocher, D. P. Efficient Graph-Based Image Segmentation[J]. International Journal of Computer Vision vol. 59 (2004).