## 2.6.7 NAIP航拍影像与分割模型库及Colaboratory和Planetary Computer Hub

### 2.6.7.1 NAIP航拍影像和构建图像数据集

（美）国家农业图像项目（National Agriculture Imagery Program，NAIP），由美国农业部农业服务局（the U.S. Department of Agriculture’s Farm Service Agency，FSA）通过位于犹他州（Utah）盐湖城（Salt Lack City）航空摄影驻地办公室（Aerial Photography Field Office，APFO）管理。NAIP在农业种植季或植被生长季（leaf on），采集地面采样距离为1m分辨率的航空影像（结合具有地理参照的图像特征进行正射校正）。每个单独的图像瓦片（tile）都为3.75-分钟经度和3.75-分钟纬度的四分四边形，加上每边300m的缓冲距离。NAIP图像包含红、绿和蓝波段，且通常含有近红外波段。美国地质调查局（U.S. Geological Survey，USGS）地球资源观测与科学（ Earth Resources Observation and Science，EROS）中心发布格式为GeoTIFF和JPEG2000（内嵌地理坐标信息的压缩文件）的NAIP产品。10：1的有损压缩以减小图像大小，但图像质量也会有所下降。APFO要求图像符合制图标准，所有交付的图像均会用自动化视觉检验图像的质量，确保其准确性且符合规范①。数据至少每3年更新一次。

NAIP航拍影像获取图途径可以从USGS的[EarthExplorer](https://earthexplorer.usgs.gov/)②中检索下载；或从微软（Microsoft）的[行星计算机（Planetary Computer）](https://planetarycomputer.microsoft.com/)③上检索下载。Planetary Computer借助云的力量支持全球环境可持续发展决策，集合具有应用接口（APIs）千万亿字节的全球环境数据目录，弹性的科学环境允许用户回答有关数据的全球问题，并将其提交到相关保护利益者手中。例如官网应用中例举了全球规模的环境监测案例，使数据驱动决策成为可能，这包括全球土地利用和土地覆盖分类（Land use and Land cover，LULC），森林砍伐风险分析，生态系统监测（森林变化、栖息地连通性、土地使用等），保护规划，森林碳（排放）风险评估，基于AI的土地覆盖物分类等。

#### 1）NAIP航拍影像数据下载

此次实验所用数据为特拉华州（Delaware）的全部NAIP数据，约147个单独影像文件，而EarthExplorer不支持批量下载，因此使用[TorchGeo](https://torchgeo.readthedocs.io/en/stable/user/installation.html)④库提供的download\_url方法（基本类似[torchvision](https://pytorch.org/vision/stable/index.html)⑤库提供的download\_url方法）从Planetary Computer云平台下载。首先在EarthExplorer地图中绘制检索区域（Polygon方式），选择NAIP数据集（Data Sets标签），检索结果的文件名（Entity ID）和区域边界可以导出为Shapefile、CSV、Comma(,)Delimited和IKMZ等文件格式，然后Python（GeoPandas库）读取，根据影像获取年份，移除重叠影像，提取获得最终要下载的文件名。

参数管理使用AttrDict()方法（具体查看“Cityscapes数据集——参数管理”一节）。data子属性存储数据文件路径。文件名Shapefile数据存储于naip\_de\_entityID变量路径名文件夹下，NAIP影像数据存储于Chesapeake\_root变量路径名文件夹下，对应的土地覆盖数据存储于Chesapeake\_LC变量路径名文件夹下。

植被生长期（leaf-on），当树木或者灌木等生长有枝叶时采集航拍影像可以获得植被物种特有的光谱反射率（spectral reflection），可用于区分植被类型；也可以检测农作物的长势和健康状况等。植被落叶期（leaf-off），当树木或者灌木等落叶，枝头很少或者没有叶子时采集航拍影像可以更清楚的看到地面特征，有助于绘制建筑物和道路等对象。

from util\_misc import AttrDict  
import os  
\_\_C=AttrDict()   
args=\_\_C  
  
\_\_C.data=AttrDict()  
\_\_C.data.Chesapeake\_root=r'E:\data\Delaware'  
\_\_C.data.Chesapeake\_LC=os.path.join(args.data.Chesapeake\_root,'LC')  
\_\_C.data.Chesapeake\_imagery=os.path.join(args.data.Chesapeake\_root,'imagery')  
\_\_C.data.naip\_de\_entityID='./data/naip\_Delaware/naip\_63b945b372d740b7.shp'

读取查看从EarthExplorer检索获取的NAIP数据文件名字段，包括有NAIP Entit(y)，和几何对象geometry字段。

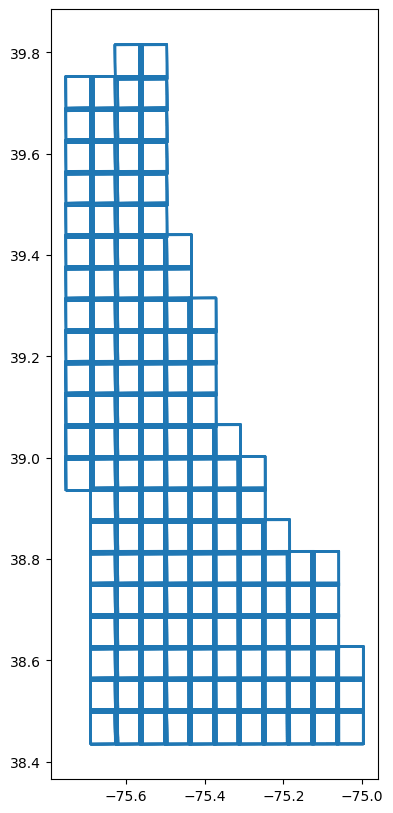
import geopandas as gpd  
naip\_de\_entityID\_gdf=gpd.read\_file(args.data.naip\_de\_entityID)  
naip\_de\_entityID\_gdf.sort\_values(by='Acquisitio',ascending=False,inplace=True)  
naip\_de\_entityID\_gdf.head(1).squeeze()

NAIP Entit M\_3907562\_SW\_18\_060\_20211019  
State DE  
Agency USDA  
Vendor USDA-FSA-APFO  
Map Projec UTM  
Projection 18N  
Datum NAD83  
Resolution 0.600000000000000  
Units METER  
Number of 4  
Sensor Typ CNIR  
Project Na 202112\_DELAWARE\_NAIP\_0X6000M\_UTM\_CNIR  
Acquisitio 2021-10-19  
Center Lat 39&deg;01'52.48"N  
Center Lon 75&deg;20'37.51"W  
NW Corner 39&deg;03'51.42"N  
NW Corne\_1 75&deg;22'39.55"W  
NE Corner 39&deg;03'52.13"N  
NE Corne\_1 75&deg;18'36.62"W  
SE Corner 38&deg;59'53.50"N  
SE Corne\_1 75&deg;18'35.58"W  
SW Corner 38&deg;59'52.79"N  
SW Corne\_1 75&deg;22'38.28"W  
Center L\_1 39.0312443  
Center L\_2 -75.3437527  
NW Corne\_2 39.0642833  
NW Corne\_3 -75.3776527  
NE Corne\_2 39.0644805  
NE Corne\_3 -75.3101722  
SE Corne\_2 38.9981944  
SE Corne\_3 -75.3098833  
SW Corne\_2 38.9979971  
SW Corne\_3 -75.3772999  
geometry POLYGON ((-75.3772999 38.9979971, -75.3776527 ...  
Name: 1343, dtype: object

打印NAIP影像瓦片分布可以发现，很多瓦片重叠，一方面是不同年份的影像，另一方面是同一年份不同时间段可能也有重叠，因此需要移除重叠的瓦片。定义IoU\_2Polygons()函数计算交并比（Intersection over Union，IoU）；定义drop\_overlapping\_polygons()函数，根据根据交并比移除过度重叠的瓦片。移除的方式是保留第一个出现的瓦片，而移除后续与之重叠的瓦片。

naip\_de\_entityID\_gdf.boundary.plot(figsize=(10,10))

<AxesSubplot: >



def IoU\_2Polygons(polygon1,polygon2):  
 '''  
 计算两个Poygon（Shapely）对象的交并比  
  
 Parameters  
 ----------  
 polygon1 : POLYGON（Shapely）  
 多边形对象1.  
 polygon2 : POLYGON（Shapely）  
 多边形对象2.  
  
 Returns  
 -------  
 iou : float  
 交并比 Intersection over Union，IoU.  
  
 '''   
 from shapely.geometry import Polygon  
   
 intersect\_area=polygon1.intersection(polygon2).area  
 union\_area=polygon1.union(polygon2).area  
 iou=intersect\_area/union\_area  
   
 return iou  
  
def drop\_overlapping\_polygons(gdf\_,iou=0.5):  
 from tqdm import tqdm   
 '''  
 移除GeoDataFrame格式文件Polygon对象重叠的行。保留第一个出现的对象，而移除后面与之重叠的对象  
  
 Parameters  
 ----------  
 gdf\_ : GeoDataFrame  
 为Polygon对象的地理信息数据.  
 iou : float, optional  
 交并比（Intersection over Union，IoU）. The default is 0.5.  
  
 Returns  
 -------  
 gdf\_non\_overlapping : GeoDataFrame  
 移除重叠的Polygon后的GeoDataFrame格式数据.  
  
 '''   
 gdf=gdf\_.copy(deep=True)  
 polygons\_dict=gdf['geometry'].to\_dict()   
 tabu\_idx=[]  
 for idx,row in tqdm(gdf.iterrows(),total=gdf.shape[0]):   
 tabu\_idx.append(idx)  
 polygons\_except4one\_dict={key:value for key, value in polygons\_dict.items() if key not in tabu\_idx}  
 pg\_gdf=row.geometry  
 for k,pg\_dict in polygons\_except4one\_dict.items():   
 iou\_2pgs=IoU\_2Polygons(pg\_dict,pg\_gdf)   
 if iou\_2pgs>iou:  
 polygons\_dict.pop(k)  
   
 gdf\_non\_overlapping=gdf.loc[list(polygons\_dict.keys())]   
 return gdf\_non\_overlapping

提取2018年的数据行，并执行移除重叠多边形的操作。

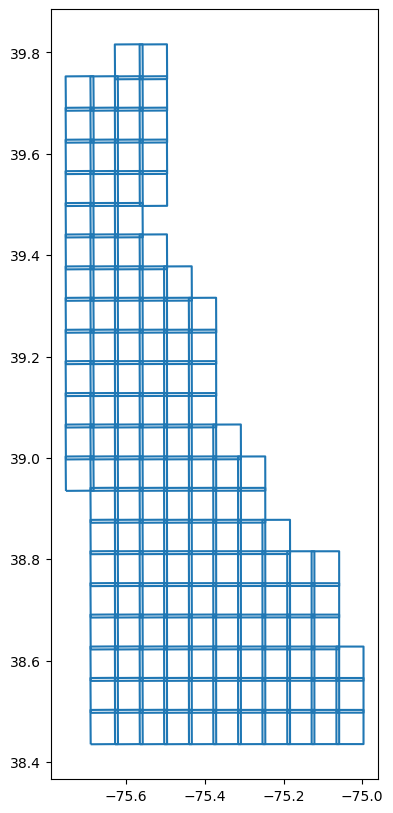
naip\_de\_entityIDd\_2018=[i for i in naip\_de\_entityID\_gdf['Acquisitio'].unique() if i.split('-')[0]=='2018']  
naip\_de\_entityID\_2018\_gdf=naip\_de\_entityID\_gdf.loc[naip\_de\_entityID\_gdf['Acquisitio'].isin(naip\_de\_entityIDd\_2018)]  
naip\_de\_entityID\_2018\_nonOverlapping\_gdf=drop\_overlapping\_polygons(naip\_de\_entityID\_2018\_gdf)

0%| | 0/162 [00:00<?, ?it/s]C:\Users\richi\anaconda3\envs\torchgeo3\lib\site-packages\shapely\set\_operations.py:133: RuntimeWarning: invalid value encountered in intersection  
 return lib.intersection(a, b, \*\*kwargs)  
100%|██████████| 162/162 [00:00<00:00, 199.94it/s]

打印处理后的NAIP影像瓦片分布，检测瓦片的重叠情况，确定已经移除了重叠的瓦片，并且基本能够覆盖整个特拉华州。

naip\_de\_entityID\_2018\_nonOverlapping\_gdf.boundary.plot(figsize=(10,10))

<AxesSubplot: >



搜索位于Planetary Computer上的NAIP数据集，[NAIP: National Agriculture Imagery Program](https://planetarycomputer.microsoft.com/dataset/naip)⑥条目提供了NAIP的信息，并在Example Notebook标签下给出了获取数据的Python代码。这里参考TorchGeo库提供的方法，通过文件名（Entity ID）构建文件下载地址，因为2018年的NAIP数据位于38075和39075两个目录下，因此分别构建下载地址下载对应的影像文件，核心使用download\_url方法，从而无需自行构建检索下载工具。

from torchgeo.datasets.utils import download\_url # from torchvision.datasets.utils import download\_url  
naip\_38075\_url=("https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/38075/")  
naip\_39075\_url=("https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/39075/")  
tiles=[i.lower()+'.tif' for i in naip\_de\_entityID\_2018\_nonOverlapping\_gdf['NAIP Entit'].tolist()]  
tiles\_38075=[i for i in tiles if '38075' in i.split('\_')[1]]  
tiles\_39075=[i for i in tiles if '39075' in i.split('\_')[1]]  
  
naip\_download\_rul=lambda url,tile,root:download\_url(url+tile,root)  
  
downloaded\_tiles=[]  
failed\_tiles=[]  
for tile in tiles\_38075:  
 try:  
 naip\_download\_rul(naip\_38075\_url,tile,args.data.Chesapeake\_imagery)  
 downloaded\_tiles.append(tile)  
 except:  
 failed\_tiles.append(tile)  
   
for tile in tiles\_39075:  
 try:  
 naip\_download\_rul(naip\_39075\_url,tile,args.data.Chesapeake\_imagery)  
 downloaded\_tiles.append(tile)  
 except:  
 failed\_tiles.append(tile)

Downloading https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/38075/m\_3807536\_ne\_18\_060\_20181104.tif to E:\_3807536\_ne\_18\_060\_20181104.tif 100%|██████████| 508699496/508699496 [14:35<00:00, 580974.44it/s] Downloading https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/38075/m\_3807535\_ne\_18\_060\_20181104.tif to E:\_3807535\_ne\_18\_060\_20181104.tif 100%|██████████| 515336170/515336170 [14:03<00:00, 610631.50it/s] Downloading https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/38075/m\_3807527\_se\_18\_060\_20181104.tif to E:\_3807527\_se\_18\_060\_20181104.tif 100%|██████████| 514118760/514118760 [14:06<00:00, 607084.33it/s] Downloading https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/38075/m\_3807527\_ne\_18\_060\_20181104.tif to E:\_3807527\_ne\_18\_060\_20181104.tif 100%|██████████| 530246098/530246098 [14:10<00:00, 623712.25it/s]

2）构建数据集并查看样本

几十年来，地球观测卫星、飞机及无人机平台一直在搜集地球表面图像，使得图像具有时空连续性，从而利于人们用遥感影像解决当今人类面临的诸多挑战，例如适应气候变化、自然灾害监测、水资源管理和全球人口日益增长下的粮食安全问题等。从计算机视觉领域则包括土地覆盖分类（语义分割）、森林砍伐和洪水监测（变化检测）、冰川流（像素跟踪）、飓风跟踪和强度估计（回归）及建筑和道路检测（物体检测、实例分割）等应用。PyTorch Ecosystem中的TorchGeo库试图同时处理深度学习模型和地理空间数据这两个截然不同领域的专业知识，为处理地理空间数据的PyTorch深度学习库，使得用深度学习处理地理空间数据工作变的简单[1]。

TorchGeo库提供了常用数据的下载工具，目前集成的[数据集](https://torchgeo.readthedocs.io/en/stable/api/datasets.html)⑦摘录如下：

| **Dataset** | **Type** | **Source** | **Size (px)** | **Resolution (m)** |
| --- | --- | --- | --- | --- |
| [**Aboveground Woody Biomass**](https://richiebao.github.io/USDA_CH_endup/#aboveground-woody-biomass) | Masks | Landsat, LiDAR | 40,000x40,000 | 30 |
| [**Aster Global DEM**](https://richiebao.github.io/USDA_CH_endup/#aster-global-dem) | Masks | Aster | 3,601x3,601 | 30 |
| [**Canadian Building Footprints**](https://richiebao.github.io/USDA_CH_endup/#canadian-building-footprints) | Geometries | Bing Imagery |  |  |
| [**Chesapeake Land Cover**](https://richiebao.github.io/USDA_CH_endup/#chesapeake-land-cover) | Imagery, Masks | NAIP |  | 1 |
| [**Global Mangrove Distribution**](https://richiebao.github.io/USDA_CH_endup/#global-mangrove-distribution) | Masks | Remote Sensing, In Situ Measurements |  | 3 |
| [**Cropland Data Layer**](https://richiebao.github.io/USDA_CH_endup/#cropland-data-layer) | Masks | Aerial |  | 30 |
| [**EDDMapS**](https://richiebao.github.io/USDA_CH_endup/#eddmaps) | Points | Citizen Scientists |  |  |
| [**EnviroAtlas**](https://richiebao.github.io/USDA_CH_endup/#enviroatlas) | Imagery, Masks | NAIP, NLCD, OpenStreetMap |  | 1 |
| [**Esri2020**](https://richiebao.github.io/USDA_CH_endup/#esri2020) | Masks | Sentinel-2 |  | 10 |
| [**EU-DEM**](https://richiebao.github.io/USDA_CH_endup/#eu-dem) | Masks | Aster, SRTM, Russian Topomaps |  | 25 |
| [**GBIF**](https://richiebao.github.io/USDA_CH_endup/#gbif) | Points | Citizen Scientists |  |  |
| [**GlobBiomass**](https://richiebao.github.io/USDA_CH_endup/#globbiomass) | Masks | Landsat | 45,000x45,000 | 100 |
| [**iNaturalist**](https://www.inaturalist.org/) | Points | Citizen Scientists |  |  |
| [**Landsat**](https://richiebao.github.io/USDA_CH_endup/#landsat) | Imagery | Landsat | 8,900x8,900 | 30 |
| [**NAIP**](https://richiebao.github.io/USDA_CH_endup/#naip) | Imagery | Aerial | 6,100x7,600 | 1 |
| [**Open Buildings**](https://richiebao.github.io/USDA_CH_endup/#open-buildings) | Geometries | Maxar, CNES/Airbus |  |  |
| [**Sentinel**](https://richiebao.github.io/USDA_CH_endup/#sentinel) | Imagery | Sentinel | 10,000x10,000 | 10 |

TorchGeo库参考PyTorch Ecosystem下的torchvision、PyTorch Lightning，同时提供了用于深度学习训练的数据集（Datasets）构建方法、样本采样（Sampler）、数据加载（DataLoader）及模型训练（Training）方式，大幅度简化了直接使用PyTorch构建深度学习模型从数据集（训练、测试、验证等）建立、模型构建（激活函数、模型和损失函数等）到训练的整个流程。NAIP方法可以直接构建NAIP航拍影像的数据集，ChesapeakeDE方法则可以下载来自于[切萨皮克保护协会（Chesapeake Conservancy Center，CIC）](https://www.chesapeakeconservancy.org/conservation-innovation-center/high-resolution-data/)⑧提供的对应NAIP数据土地覆盖类型数据集。CIC成立于2013年，旨在利用尖端技术赋予保护和修复环境以数据驱动力。通过dataset=naip & chesapeake方式可以直接对位叠加NAIP的航拍影像和对应的土地覆盖类型数据为单个的数据集，且可以用RandomGeoSampler等方法采样数据，返回给定大小的影像image及对应的分类mask。该土地覆盖类型包括13类[2]：

1. Water（水体）：所有开放水域，包括池塘、河流、湖泊（含不依附码头的船只）等，及小型人为的农场池塘和雨水滞留设施等。MMU=25m²；
2. Emergent Wetlands（湿地）：位于海洋或河口的低植被区域，视觉上可以观察到包含植被的饱和地面，且位于主要水道（河流、海洋等）。对于弗吉尼亚州（Virginia）的潮汐带，该类湿地通常具有低矮的植被、木本植被和土地贫瘠（裸地）的特征。MMU=225m²；
3. Tree Canopy（树冠/林冠）：自然演替或人工种植的落叶和常绿木本植被，高度通常超过3米。单株，离散的林木团和紧密连接的单株都计算在内。MMU=9²；
4. Scrub/Shrub（灌木丛）：落叶和/或常绿木本植被的异质区域，贯穿覆盖有低矮植被和草地，高低变化斑驳分布的灌木丛和幼树为特征，包括离散的灌木丛和紧密联系的单株植被。和由于环境条件发育不良矮小的灌木、幼树等，它们混长于植被较低的异质景观中。MMU=225²；
5. Low Vegetation（低矮植被）：高度小于约3米的植被，包括草坪、耕地、有或无防水布覆盖的苗圃种植区，近期砍伐的森林管理区，和自然地表覆盖物等。MMU=9m²；
6. Barren（裸地）：由天然土质构成的无植被区域，包括海滩、泥滩和建筑工地的裸露地面等。MMU=25m²；
7. Impervious Structures（不透水结构）：由不透水材料制成高度大于约2米的人造构筑，包括房屋、商城和电力塔等。MMU=9m²；
8. Other Impervious（其它不透水类）：水不能渗透、小于约2米的人造构筑。MMU=9m²；
9. Impervious Road（不透水路面）：为运输使用和需要维护的不透水表面。MMU=9m²；
10. Tree Canopy over Impervious Structure（覆盖于不透水结构上的树冠）：与不透水表面重叠的森林或树木覆盖物，使得不透水结构部分部分或完全不可见。注：不透水表面和树冠层是独立绘制的，通过重叠区域识别。MMU=9m²；
11. Tree Canopy over Other Impervious（覆盖于其它不透水结构上的树冠）：与不透水表面重叠的森林或树木覆盖物，使得不透水结构部分部分或完全不可见。MMU=9m²；
12. Tree Canopy over Impervious Roads（覆盖于不透水路面上的树冠）：与不透水路面重叠的森林或树木覆盖物，使得不透水结构部分部分或完全不可见。MMU=9m²；

254.Aberdeen Proving Ground（阿伯丁实验场）：该类区域无源图像或辅助数据可用，该类存在于马里兰郡的哈福德（Harford,County Maryland）。

PyTorch的Ecosystem由来自学术界、工业界、应用程序开发人员和ML（Machine Learning）工程师开发的项目、工具和库组成，其目的是支持、加速和帮助对PyTorch的探索。

from torchgeo.datasets import NAIP,ChesapeakeDE,stack\_samples   
  
naip=NAIP(args.data.Chesapeake\_imagery)  
chesapeake=ChesapeakeDE(args.data.Chesapeake\_LC, crs=naip.crs, res=naip.res, download=False)  
dataset=naip & Chesapeake

from torchgeo.samplers import RandomGeoSampler  
from torch.utils.data import DataLoader  
  
sampler=RandomGeoSampler(dataset, size=256, length=10)  
dataloader=DataLoader(dataset, sampler=sampler, collate\_fn=stack\_samples)  
for batch in dataloader:  
 image=batch["image"]  
 target=batch["mask"]  
 break  
print(image,'\n',target)

tensor([[[[102, 94, 91, ..., 87, 93, 91],  
 [ 96, 89, 92, ..., 98, 92, 99],  
 [ 96, 85, 88, ..., 96, 96, 99],  
 ...,  
 [110, 112, 144, ..., 117, 63, 51],  
 [102, 104, 119, ..., 137, 90, 83],  
 [113, 103, 104, ..., 129, 81, 74]],  
  
 [[121, 113, 115, ..., 103, 112, 109],  
 [114, 106, 107, ..., 121, 98, 117],  
 [117, 104, 108, ..., 112, 102, 123],  
 ...,  
 [134, 127, 171, ..., 111, 55, 63],  
 [129, 122, 144, ..., 118, 77, 92],  
 [135, 109, 113, ..., 115, 76, 73]],  
  
 [[ 81, 78, 78, ..., 78, 79, 80],  
 [ 78, 74, 75, ..., 85, 77, 81],  
 [ 79, 78, 76, ..., 87, 80, 81],  
 ...,  
 [ 94, 89, 129, ..., 110, 63, 72],  
 [ 94, 86, 97, ..., 111, 72, 85],  
 [ 95, 83, 86, ..., 103, 75, 81]],  
  
 [[153, 149, 150, ..., 156, 156, 155],  
 [147, 142, 148, ..., 158, 154, 155],  
 [146, 149, 144, ..., 159, 156, 150],  
 ...,  
 [163, 163, 155, ..., 60, 30, 30],  
 [158, 158, 156, ..., 66, 39, 39],  
 [159, 152, 152, ..., 57, 41, 35]]]], dtype=torch.uint8)   
 tensor([[[[5, 5, 5, ..., 5, 5, 5],  
 [5, 5, 5, ..., 5, 5, 5],  
 [5, 5, 5, ..., 5, 5, 5],  
 ...,  
 [3, 3, 3, ..., 8, 8, 8],  
 [3, 3, 3, ..., 8, 8, 8],  
 [3, 3, 3, ..., 8, 8, 8]]]], dtype=torch.uint8)

unbind\_samples方法与stack\_sample方法相逆，将小批量（mini-batch）样本转换成样本列表。

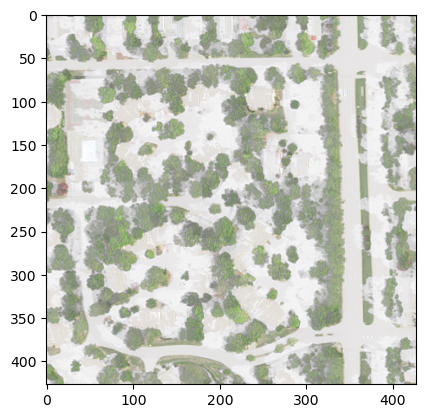
from torchgeo.datasets import unbind\_samples  
  
sample=unbind\_samples(batch)[0]  
sample

{'image': tensor([[[107, 118, 149, ..., 103, 132, 138],  
 [174, 159, 187, ..., 130, 134, 136],  
 [190, 202, 207, ..., 103, 100, 97],  
 ...,  
 [ 57, 55, 64, ..., 166, 162, 163],  
 [ 57, 62, 59, ..., 133, 146, 143],  
 [ 58, 56, 60, ..., 116, 125, 124]],  
   
 [[109, 127, 129, ..., 143, 175, 166],  
 [131, 148, 160, ..., 148, 159, 164],  
 [140, 176, 190, ..., 137, 117, 121],  
 ...,  
 [ 52, 45, 68, ..., 136, 143, 144],  
 [ 53, 53, 48, ..., 117, 126, 117],  
 [ 51, 54, 53, ..., 121, 125, 114]],  
   
 [[ 80, 86, 85, ..., 80, 93, 92],  
 [ 96, 110, 120, ..., 89, 81, 102],  
 [105, 140, 161, ..., 82, 83, 73],  
 ...,  
 [ 68, 56, 107, ..., 136, 125, 133],  
 [ 65, 69, 61, ..., 96, 116, 107],  
 [ 63, 69, 66, ..., 88, 92, 85]],  
   
 [[193, 197, 179, ..., 188, 206, 212],  
 [168, 177, 175, ..., 197, 212, 203],  
 [157, 163, 163, ..., 192, 180, 152],  
 ...,  
 [ 17, 16, 18, ..., 52, 52, 57],  
 [ 17, 18, 17, ..., 131, 115, 97],  
 [ 17, 16, 15, ..., 191, 180, 174]]], dtype=torch.uint8),  
 'crs': CRS.from\_epsg(26918),  
 'bbox': BoundingBox(minx=494628.6, maxx=494884.6, miny=4264922.399999999, maxy=4265178.399999999, mint=1534262399.999999, maxt=1534348799.999999),  
 'mask': tensor([[[ 3, 3, 3, ..., 10, 10, 10],  
 [ 3, 3, 3, ..., 10, 10, 10],  
 [11, 11, 3, ..., 10, 10, 10],  
 ...,  
 [ 8, 8, 8, ..., 10, 10, 10],  
 [ 8, 8, 8, ..., 10, 10, 10],  
 [ 8, 8, 8, ..., 10, 10, 10]]], dtype=torch.uint8)}

分别打印样本的影像和对应的分类（标签）。

import matplotlib.pyplot as plt  
import torchvision.transforms as T  
  
print(sample['image'].shape)  
plt.imshow(T.ToPILImage()(sample['image'])); # plt.imshow(sample['image'][:3].T)

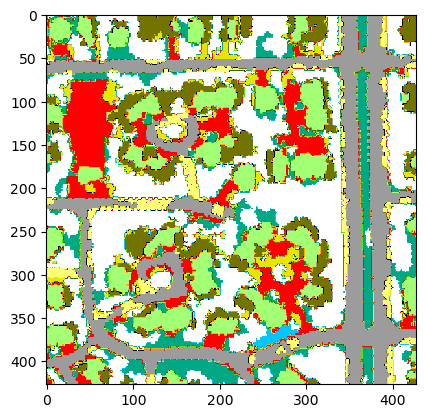
torch.Size([4, 427, 427])



配置土地覆盖类型的颜色字典LC\_color\_dict，并转换为matplotlib库的cmap颜色图实例，用于地图打印分类颜色。

import matplotlib  
import numpy as np  
  
LC\_color\_dict={  
 0: (0, 0, 0, 0),  
 1: (0, 197, 255, 255),  
 2: (0, 168, 132, 255),  
 3: (38, 115, 0, 255),  
 4: (76, 230, 0, 255),  
 5: (163, 255, 115, 255),  
 6: (255, 170, 0, 255),  
 7: (255, 0, 0, 255),  
 8: (156, 156, 156, 255),  
 9: (0, 0, 0, 255),  
 10: (115, 115, 0, 255),  
 11: (230, 230, 0, 255),  
 12: (255, 255, 115, 255),  
 13: (197, 0, 255, 255),  
 }  
  
cmap\_LC, norm=matplotlib.colors.from\_levels\_and\_colors(list(LC\_color\_dict.keys()),[[v/255 for v in i] for i in LC\_color\_dict.values()],extend='max')  
print(target.shape)  
plt.imshow(np.squeeze(target),cmap=cmap\_LC);

torch.Size([1, 1, 427, 427])



可以用naip.plot()方法直接打印unbind\_samples后的小批量样本。

sampler\_naip=RandomGeoSampler(naip, size=4096, length=3)  
dataloader\_naip=DataLoader(naip, sampler=sampler\_naip, collate\_fn=stack\_samples)  
  
i=0  
for batch in dataloader\_naip:  
 sample=unbind\_samples(batch)[0]  
 naip.plot(sample['image'])  
 if i==3:break  
 i+=1







#### 3）图像数据增强变换（ data augmentation transforms）

[TorchVison](https://pytorch.org/vision/stable/transforms.html)⑤和[Kornia](https://github.com/kornia/kornia)⑨均为计算机视觉库，且都提供了图像增强变换的诸多方法。TorchGeo的transforms模块则针对地理空间数据多个栅格波段追加了如下指数：

NBR（Normalized Burn Ratio ）：；

NDBI（Normalized Difference Built-up Index）：；

NDSI（Normalized Difference Snow Index）：；

NDVI（Normalized Difference Vegetation Index）：；

NDWI（Normalized Difference Water Index）：；

SWI（Standardized Water-Level Index）：；

GNDVI（Green Normalized Difference Vegetation Index）：；

BNDVI（Blue Normalized Difference Vegetation Index ）：；

NDRE（Normalized Difference Red Edge Vegetation Index）：；

GRNDVI（Green-Red Normalized Difference Vegetation Index）：；

GBNDVI（Green-Blue Normalized Difference Vegetation Index）：；

RBNDVI（Red-Blue Normalized Difference Vegetation Index）：。

下述代码调用了kornia库提供的图像增强变换方法，并追加了NDVI和NDWI两个指数变换图像。

dataloader=DataLoader(dataset, sampler=sampler, collate\_fn=stack\_samples)  
dataloader=iter(dataloader)  
batch=next(dataloader)  
x, y=batch["image"], batch["mask"]

from torchgeo.transforms import AugmentationSequential, indices  
import kornia.augmentation as K  
import torch.nn as nn  
  
augmentations=AugmentationSequential(  
 K.RandomHorizontalFlip(p=0.5),  
 K.RandomVerticalFlip(p=0.5),  
 # K.RandomAffine(degrees=(0, 90), p=0.25),  
 # K.RandomGaussianBlur(kernel\_size=(3, 3), sigma=(0.1, 2.0), p=0.25),  
 # K.RandomResizedCrop(size=(512, 512), scale=(0.8, 1.0), p=0.25),  
 data\_keys=["image",'mask'],  
)  
transforms=nn.Sequential(  
 indices.AppendNDVI(index\_nir=3, index\_red=0),  
 indices.AppendNDWI(index\_green=1, index\_nir=3),  
 augmentations,  
)  
  
batch=next(dataloader)  
print(batch["image"].shape)  
batch=transforms(batch)  
print(batch["image"].shape)

torch.Size([1, 4, 427, 427])  
torch.Size([1, 6, 427, 427])

nvidia-smi（NVSMI）为NVIDIA显卡（例如Tesla, Quadro, GRID和GeForce等）设备提供监控和管理功能。下述打印的信息显示了显卡驱动（driver）版本，CUDA（Compute Unified Device Architecture)）版本等相关信息。

!nvidia-smi

Wed Jan 11 17:54:04 2023   
+-----------------------------------------------------------------------------+  
| NVIDIA-SMI 512.78 Driver Version: 512.78 CUDA Version: 11.6 |  
|-------------------------------+----------------------+----------------------+  
| GPU Name TCC/WDDM | Bus-Id Disp.A | Volatile Uncorr. ECC |  
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |  
| | | MIG M. |  
|===============================+======================+======================|  
| 0 NVIDIA GeForce ... WDDM | 00000000:01:00.0 On | N/A |  
| N/A 64C P0 37W / N/A | 3320MiB / 8192MiB | 1% Default |  
| | | N/A |  
+-------------------------------+----------------------+----------------------+  
   
+-----------------------------------------------------------------------------+  
| Processes: |  
| GPU GI CI PID Type Process name GPU Memory |  
| ID ID Usage |  
|=============================================================================|  
| 0 N/A N/A 1144 C+G ...ekyb3d8bbwe\HxOutlook.exe N/A |  
| 0 N/A N/A 5804 C+G ...n1h2txyewy\SearchHost.exe N/A |  
| 0 N/A N/A 6956 C+G ...d\runtime\WeChatAppEx.exe N/A |  
| 0 N/A N/A 8908 C+G ...zilla Firefox\firefox.exe N/A |  
| 0 N/A N/A 9428 C+G ...zilla Firefox\firefox.exe N/A |  
| 0 N/A N/A 12076 C+G C:\Windows\explorer.exe N/A |  
| 0 N/A N/A 12320 C+G ...462.76\msedgewebview2.exe N/A |  
| 0 N/A N/A 13148 C+G ...artMenuExperienceHost.exe N/A |  
| 0 N/A N/A 13856 C+G ...lPanel\SystemSettings.exe N/A |  
| 0 N/A N/A 14928 C+G ...e\Current\LogiOverlay.exe N/A |  
| 0 N/A N/A 15388 C+G ...ystemEventUtilityHost.exe N/A |  
| 0 N/A N/A 15912 C+G ...2txyewy\TextInputHost.exe N/A |  
| 0 N/A N/A 16884 C+G ...e\PhoneExperienceHost.exe N/A |  
| 0 N/A N/A 17076 C+G ...y\ShellExperienceHost.exe N/A |  
| 0 N/A N/A 17812 C+G ...ge\Application\msedge.exe N/A |  
| 0 N/A N/A 18776 C+G ...ram Files\LGHUB\lghub.exe N/A |  
| 0 N/A N/A 19372 C+G ...ray\lghub\_system\_tray.exe N/A |  
| 0 N/A N/A 20280 C+G ...arkupHero\Markup Hero.exe N/A |  
| 0 N/A N/A 21124 C+G ...cw5n1h2txyewy\LockApp.exe N/A |  
| 0 N/A N/A 22632 C+G ...me\Application\chrome.exe N/A |  
| 0 N/A N/A 27516 C+G ...tracted\WechatBrowser.exe N/A |  
| 0 N/A N/A 27820 C+G ...wekyb3d8bbwe\Video.UI.exe N/A |  
| 0 N/A N/A 31488 C+G ...root\Office16\WINWORD.EXE N/A |  
| 0 N/A N/A 34900 C+G ...icrosoft VS Code\Code.exe N/A |  
+-----------------------------------------------------------------------------+

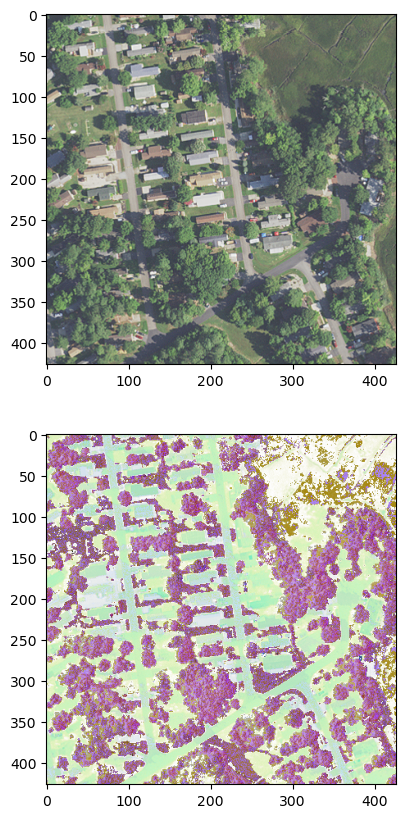
将图像增强变换nn.Sequential对象（按照构造函数中传递的对象顺序添加到模块容器中）通过sequential.to(device[cuda])转换到GPU中进行计算。打印原始图像和增强变换后的图像。

import torch  
from copy import deepcopy  
  
device="cuda" if torch.cuda.is\_available() else "cpu"  
print(device)  
transforms\_gpu=deepcopy(transforms).to(device)

cuda

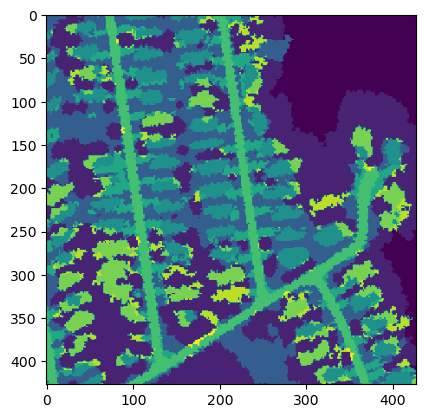
import copy  
sampler\_naip=RandomGeoSampler(naip, size=4096, length=1)  
dataloader=DataLoader(dataset, sampler=sampler, collate\_fn=stack\_samples)  
  
fig, axs=plt.subplots(2,figsize=(10,10))  
for n in dataloader:  
 batch=copy.deepcopy(n)  
  
 print(batch['image'].shape)  
 axs[0].imshow(T.ToPILImage()(batch['image'][0][:3]));  
 batch=transforms\_gpu(batch)  
 print(batch['image'].shape)  
 axs[1].imshow(T.ToPILImage()(batch['image'][0][2:6]));   
 break

torch.Size([1, 4, 427, 427])  
torch.Size([1, 6, 427, 427])



打印真实分类。

plt.imshow(T.ToPILImage()(batch['mask'][0]));



### 2.6.7.2 分割模型库及Colaboratory和Planetary Computer Hub

#### 1）Segmentation Models

深度学习的不断发展更新，新的网络不断涌现，大量被证实可用的深度学习网络广泛传播，因此集成已有成熟的网络（通常包含预训练参数权重）库为网络的实际应用带来便利，例如前文涉及到的[ONNX Model Zoo](https://github.com/onnx/models)⑩、[torchvision.models](https://pytorch.org/vision/stable/models.html)⑪等，及[PyTorch Image Models（TIMM）](https://github.com/rwightman/pytorch-image-models)⑫、[Transformers](https://github.com/huggingface/transformers)⑬和[Segmentation Models](https://smp.readthedocs.io/en/stable/index.html)⑭等，且不仅上述所限。以图像语义分割为主的Segmentation Models为例，可以仅用几行代码调用预训练的模型，涉及的分割网络包括Unet、Unet++、MAnet、Linknet、FPN、PSPNet、PAN、DeepLabV3和DeepLabV3+等；且包括高达124个可用的编码器（encoders）,所有编码器都有预训练的参数权重，以实现更快更好的模型训练收敛，例如ResNet（imagenet / ssl / swsl）、ResNeXt（imagenet / instagram / ssl / swsl）、ResNeSt（imagenet）、Res2Ne(X)t（imagenet）、RegNet(x/y)（imagenet）、GERNet（imagenet）、SE-Net（imagenet）、SK-ResNe(X)t（imagenet）、DenseNet（imagenet）、Inception（imagenet / imagenet+background）、EfficientNet（imagenet / advprop / noisy-student）、MobileNet（imagenet）、DPN（imagenet+5k）、VGG（imagenet）、Mix Vision Transformer（imagenet）和MobileOne（imagenet）等。其中预训练模型最常用的图像数据库为[ImageNet](https://www.image-net.org/index.php)⑮（仅包含名词），为按照[WordNet](https://en.wikipedia.org/wiki/WordNet)⑯层次结构组织的图像数据库，其中层次结构的每个节点由成百上千的图像描述。该项目在推进计算机视觉和深度学习研究方面发挥了重要作用。（ImageNet图像数据库开源用于非商业用途的研究）。

下述示例调用网络构建模型和预测（假值）的代码，并打印了该Unet网络模型，其中编码器使用了resnet34，预训练参数权重来自于图像数据库imagenet。预测的结果为mask掩码。

import segmentation\_models\_pytorch as smp  
import torch  
  
model=smp.Unet(  
 encoder\_name="resnet34", # choose encoder, e.g. mobilenet\_v2 or efficientnet-b7  
 encoder\_weights="imagenet", # use `imagenet` pre-trained weights for encoder initialization  
 in\_channels=3, # model input channels (1 for gray-scale images, 3 for RGB, etc.)  
 classes=13, # model output channels (number of classes in your dataset)  
)  
mask=model(torch.ones([1, 3,64, 64]))  
print(mask[:,0,:])  
print('-'\*50)  
print(model)

tensor([[[-0.0060, -0.1177, -0.1687, ..., 0.3846, 0.0027, 0.0376],  
 [ 0.4949, -0.3818, 0.0144, ..., 0.3487, 0.0475, -0.0637],  
 [ 0.1829, -0.9314, -0.4307, ..., 1.1400, 0.3028, -0.0593],  
 ...,  
 [ 0.0888, -0.0988, -0.2369, ..., -0.4098, -0.2458, -0.5457],  
 [-0.6411, -0.4427, -0.9519, ..., -0.9773, -0.4209, -0.2852],  
 [-0.1010, -0.4232, -0.8313, ..., -0.2045, 0.1289, 0.0031]]],  
 grad\_fn=<SliceBackward0>)  
--------------------------------------------------  
Unet(  
 (encoder): ResNetEncoder(  
 (conv1): Conv2d(3, 64, kernel\_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)  
 (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (maxpool): MaxPool2d(kernel\_size=3, stride=2, padding=1, dilation=1, ceil\_mode=False)  
 (layer1): Sequential(  
 (0): BasicBlock(  
 (conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 (1): BasicBlock(  
 (conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 (2): BasicBlock(  
 (conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (layer2): Sequential(  
 (0): BasicBlock(  
 (conv1): Conv2d(64, 128, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (downsample): Sequential(  
 (0): Conv2d(64, 128, kernel\_size=(1, 1), stride=(2, 2), bias=False)  
 (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (1): BasicBlock(  
 (conv1): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 (2): BasicBlock(  
 (conv1): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 (3): BasicBlock(  
 (conv1): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (layer3): Sequential(  
 (0): BasicBlock(  
 (conv1): Conv2d(128, 256, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (downsample): Sequential(  
 (0): Conv2d(128, 256, kernel\_size=(1, 1), stride=(2, 2), bias=False)  
 (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (1): BasicBlock(  
 (conv1): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 (2): BasicBlock(  
 (conv1): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 (3): BasicBlock(  
 (conv1): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 (4): BasicBlock(  
 (conv1): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 (5): BasicBlock(  
 (conv1): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (layer4): Sequential(  
 (0): BasicBlock(  
 (conv1): Conv2d(256, 512, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (downsample): Sequential(  
 (0): Conv2d(256, 512, kernel\_size=(1, 1), stride=(2, 2), bias=False)  
 (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (1): BasicBlock(  
 (conv1): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 (2): BasicBlock(  
 (conv1): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (relu): ReLU(inplace=True)  
 (conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 )  
 (decoder): UnetDecoder(  
 (center): Identity()  
 (blocks): ModuleList(  
 (0): DecoderBlock(  
 (conv1): Conv2dReLU(  
 (0): Conv2d(768, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention1): Attention(  
 (attention): Identity()  
 )  
 (conv2): Conv2dReLU(  
 (0): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention2): Attention(  
 (attention): Identity()  
 )  
 )  
 (1): DecoderBlock(  
 (conv1): Conv2dReLU(  
 (0): Conv2d(384, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention1): Attention(  
 (attention): Identity()  
 )  
 (conv2): Conv2dReLU(  
 (0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention2): Attention(  
 (attention): Identity()  
 )  
 )  
 (2): DecoderBlock(  
 (conv1): Conv2dReLU(  
 (0): Conv2d(192, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention1): Attention(  
 (attention): Identity()  
 )  
 (conv2): Conv2dReLU(  
 (0): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention2): Attention(  
 (attention): Identity()  
 )  
 )  
 (3): DecoderBlock(  
 (conv1): Conv2dReLU(  
 (0): Conv2d(128, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention1): Attention(  
 (attention): Identity()  
 )  
 (conv2): Conv2dReLU(  
 (0): Conv2d(32, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention2): Attention(  
 (attention): Identity()  
 )  
 )  
 (4): DecoderBlock(  
 (conv1): Conv2dReLU(  
 (0): Conv2d(32, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention1): Attention(  
 (attention): Identity()  
 )  
 (conv2): Conv2dReLU(  
 (0): Conv2d(16, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
 (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU(inplace=True)  
 )  
 (attention2): Attention(  
 (attention): Identity()  
 )  
 )  
 )  
 )  
 (segmentation\_head): SegmentationHead(  
 (0): Conv2d(16, 13, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): Identity()  
 (2): Activation(  
 (activation): Identity()  
 )  
 )  
)

所有编码器都有预训练的权重。以与权重训练相同的方式准备数据可能使得训练更快的收敛且获得更好的预测准确度（各类测量指标，例如precision（查准率）、recall（召回率）、F1-Score（（F1-分数）和Intersection over Uion（IOU）等）。但是如果训练整个模型，而不仅是解码器则没有必要。

from segmentation\_models\_pytorch.encoders import get\_preprocessing\_fn  
preprocess\_input=get\_preprocessing\_fn('resnet18', pretrained='imagenet')  
print(preprocess\_input)

functools.partial(<function preprocess\_input at 0x000001B11D4C4AF0>, input\_space='RGB', input\_range=[0, 1], mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

输入通道in\_channels数量根据训练图像数据确定，如果预训练权重来自于ImageNet，第一个卷积层的权重将被重复使用。对于单通道（in\_channels=1）的情况，为第一个卷积层的权重之和，否则通道权重类似为new\_weight[:, i] = pretrained\_weight[:, i % 3]，然后用new\_weight \* 3 / new\_in\_channels缩放。

model=smp.FPN('resnet34', in\_channels=1)  
mask=model(torch.ones([1, 1, 64, 64]))  
print(mask,mask.shape)

tensor([[[[0.6180, 0.5185, 0.4191, ..., 0.6451, 0.4130, 0.1810],  
 [0.6129, 0.5560, 0.4990, ..., 1.0183, 0.7343, 0.4504],  
 [0.6078, 0.5934, 0.5790, ..., 1.3915, 1.0556, 0.7197],  
 ...,  
 [1.6890, 1.6174, 1.5457, ..., 2.0490, 1.9673, 1.8856],  
 [1.8117, 1.7673, 1.7229, ..., 2.1160, 2.0245, 1.9329],  
 [1.9345, 1.9173, 1.9000, ..., 2.1830, 2.0816, 1.9802]]]],  
 grad\_fn=<UpsampleBilinear2DBackward1>) torch.Size([1, 1, 64, 64])

所有模型都支持aux\_params辅助分类输出参数，默认值为None，不创建分类辅助输出；如果配置了该参数，模型不仅产生掩码（mask），且产生分类标签（联合预测概率），分类头（Classification head）由GlobalPooling->Dropout(optional)->Linear->Activation(optional)等层组成。

aux\_params=dict(  
 pooling='avg', # one of 'avg', 'max'  
 dropout=0.5, # dropout ratio, default is None  
 activation='sigmoid', # activation function, default is None  
 classes=3, # define number of output labels  
)  
model=smp.Unet('resnet34', classes=3, aux\_params=aux\_params)  
mask,label=model(torch.ones([1, 3, 64, 64]))  
print(mask.shape)  
print(label)

torch.Size([1, 3, 64, 64])  
tensor([[0.0278, 0.3745, 0.2524]], grad\_fn=<SigmoidBackward0>)

encoder\_depth深度参数可以配置下采样大小，较小的值可以减轻模型的计算量。

model=smp.DeepLabV3Plus('resnet34', encoder\_depth=3)

#### 2）Colaboratory和Planetary Computer Hub

深度学习神经网络的参数量通常在百万级之上（例如包括卷积层参数、全连接层参数、输出层参数及训练和测试的参数等），因此对硬件通常有必要的需求，而神经网络是高度并行的，那么适合于并行计算的GPU（Graphics processing unit）图形处理器成为深度学习的重要依赖，在CPU上需要花费几个小时完成的训练任务，在GPU上只需要10几分钟；那么对于GPU需要几天甚至几周完成的训练任务，很难想象用CPU计算完成。目前个人电脑根据需要通常配置有GPU，但是其显存大小、位宽、带宽和计算能力影响到GPU的效能，高的配置可以减小深度学习网络训练的时间，尤其无法避免反复调参的情况下，高效率的GPU配置将大幅度减少等待时间。高配置的GPU往往价格偏高，对于个人使用可能会受到限制；如果科研团队配置有高配的GPU，但是数量可能有限，如果有经常训练模型的需求，团队公用的GPU也限制了使用的自由，因此[Colaborator，CoLab](https://colab.research.google.com/)⑰（由Google推出）和[Planetary Computer Hub，PcHub](https://pccompute.westeurope.cloudapp.azure.com/compute/hub)⑱（由Microsoft推出）等云端提供的算力则解决了这一问题，同时以社区方式的平台建立，也大大增加了深度学习相关研究的推进和开源分享。CoLab和PcHub均集成了Jupyter笔记服务，且预先安装了常用的Python库，这包括PyTorch和TenserFlow等深度学习库。

CoLab通常配合Google Drive使用，可以实现数据集和训练模型的无缝连接，避免数据丢失（重新下载或上传），尤其训练好的模型丢失造成的损失。CoLab的另一优势在于可以访问计算机系统层，这对于需要修改Pytho库代码的用户而言是必要的。相比CoLab，Hub无法访问系统层，但Hub集成了很多全球环境数据目录，例如naip、sentinel、modis、landsat、us-census、noaa、eclipse等几十种，可以方便快捷的调用数据用于相关分析。

以NAIP航拍影像语义分割为例，使用CoLab书写代码、训练模型，配合Google Drive存储数据和训练结果。分割模型调用Segmentation Models库提供的Unet网络，使用TorchGeo和Pytorcch Lighting库处理。因为TorchGeo库目前处于开发上升期，因此有些不方便应用的方法则根据本次实验进行调整。Pyton库无以计数，不同的库处于不同的状态，例如Pytorch、Pandas等稳定型，有大量使用者和大量开发者维护；有些库则处于开发上升期，代码变更可能会比较频繁，有些方法在前一版本存在，在后一版本可能就会取消，又或者变更到其它模块，或者修改了名称，及调整了参数和返回值等；还有些库已经停止了维护及更新，但有些功能在特定的分析中却非常有用。因此，面对不同库的状态，进行数据分析时，往往也会对调用的库进行修改，修改的方式主要包括两种，一种是，直接在安装环境下修改库代码；另一种是迁移库代码，结合自身分析调整代码等。

下述代码为CoLab中执行代码

* 库的安装

因为CoLab没有预先安装TorchGeo库，因此需要安装；当前版本的Matplotlib库的plt.imshow方法提示错误，因此更新到3.6.2版本。

%pip install torchgeo tensorboard pytorch\_lightning

如果要修改TorchGeo库的部分代码，满足分析需要，可以通过!pip show torchgeo获取该库位置。（本次试验未修改安装库的代码，但是通过代码迁移修改的方式实现）

!pip show torchgeo

Name: torchgeo  
Version: 0.4.1  
Summary: TorchGeo: datasets, samplers, transforms, and pre-trained models for geospatial data  
Home-page: https://github.com/microsoft/torchgeo  
Author: Adam J. Stewart  
Author-email: ajstewart426@gmail.com  
License:   
Location: /usr/local/lib/python3.10/dist-packages  
Requires: einops, fiona, kornia, lightning, matplotlib, numpy, pillow, pyproj, rasterio, rtree, scikit-learn, segmentation-models-pytorch, shapely, timm, torch, torchmetrics, torchvision  
Required-by:

%matplotlib inline  
%load\_ext tensorboard  
  
import os  
from torchvision.transforms import Compose  
from torchgeo.datasets import RasterDataset,stack\_samples,unbind\_samples  
from torchgeo.datasets.utils import download\_url  
  
from torchgeo.transforms import indices,AugmentationSequential  
import kornia.augmentation as K  
  
from torchgeo.samplers import RandomGeoSampler  
from torch.utils.data import DataLoader  
  
import matplotlib  
import numpy as np  
import matplotlib.pyplot as plt  
  
from torchgeo.datamodules import NAIPChesapeakeDataModule  
import torch  
from lightning.pytorch import Trainer  
from lightning.pytorch.callbacks import EarlyStopping, ModelCheckpoint  
from lightning.pytorch.loggers import TensorBoardLogger  
from torchgeo.trainers import SemanticSegmentationTask  
from os import path  
import torch.nn as nn  
import torchvision.transforms as T  
from pytorch\_lightning.loggers import CSVLogger  
  
import torchgeo  
from lightning.pytorch import LightningDataModule  
from torchgeo.datasets import BoundingBox  
  
import csv  
  
from torchgeo.samplers.batch import RandomBatchGeoSampler  
from torchgeo.samplers.single import GridGeoSampler

The tensorboard extension is already loaded. To reload it, use:  
 %reload\_ext tensorboard

* 连接Google Drive

为避免数据丢失，方便数据调用，连接Google Drive到CoLab，文件夹会加载到CoLab本地显示，用法同本地文件。

from google.colab import drive  
drive.mount('/content/gdrive')

Mounted at /content/gdrive

* 配置数据存储路径

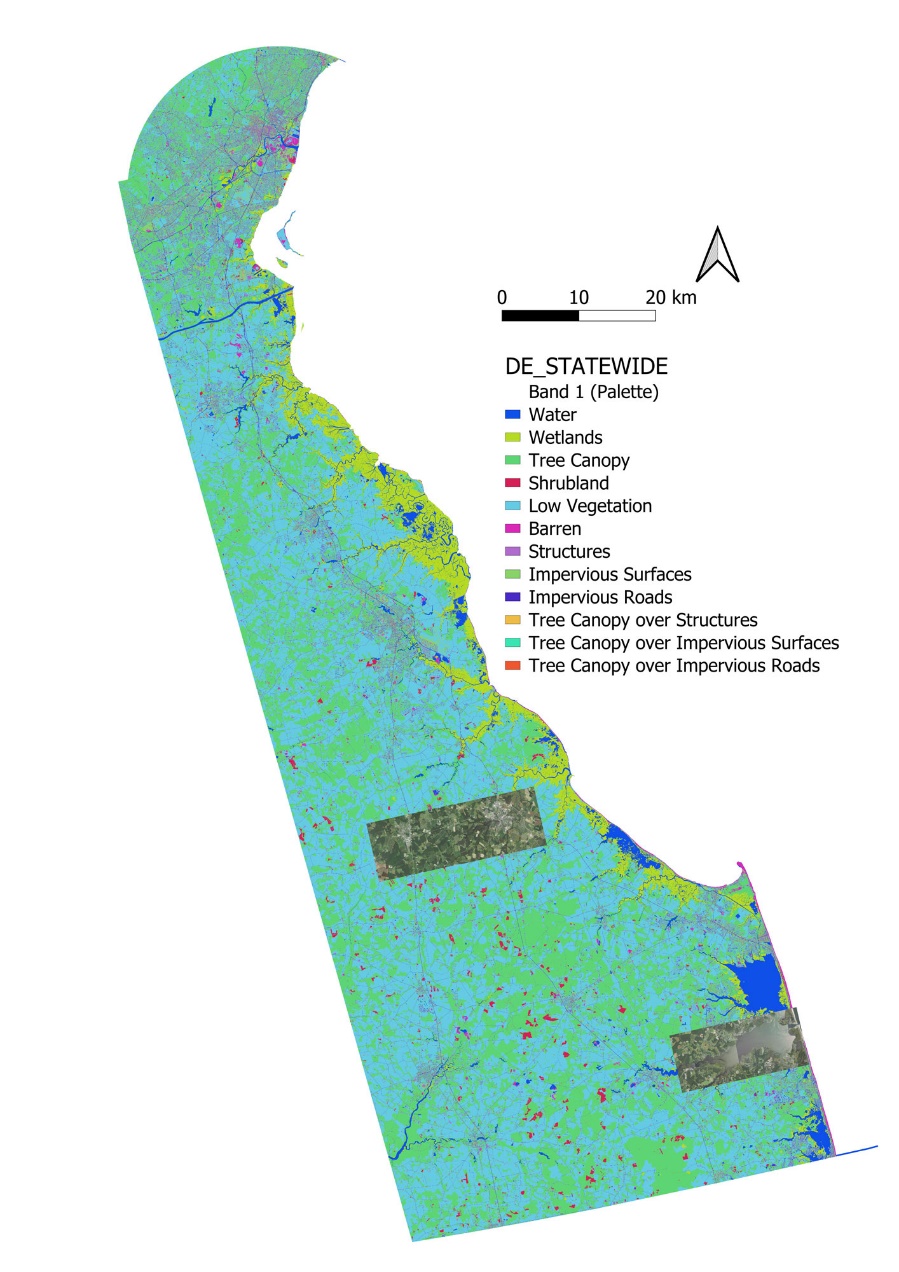
数据文件、训练过程和结果文件均存储于Google Drive中，文件夹可以在Googel Drive中建立，也可以直接在CoLab中建立。文件路径可以在对应文件上右键Copy path直接获取。

root='/content/gdrive/MyDrive/dataset/naipNdelaware4seg'  
imagery\_data=os.path.join(root, "imagery") # 存储NAIP航拍影像数据  
LC\_data=os.path.join(root, "LC") # 存储NAIP土地覆盖分类数据  
data\_dir=os.path.join(root, "training") # 存储模型训练过程和结果文件

* NAIP图像数据下载

训练时未使用全部Delaware区域影像，仅选取了4个瓦片。将瓦片编号录入文本文件naipEntityID\_selection.txt下，传至Google Drive用于读取并下载对应图像数据至Google Drive。下载时直接调用torchvision库的download\_url方法。

下图为 Delaware 区域 13 类土地覆盖类型，及所用的 NAIP 7个影像瓦片。



naipEntityID\_selection\_fn=os.path.join(root,'naipEntityID\_selection.txt')  
with open(naipEntityID\_selection\_fn,'r') as f:  
 naipEntityID\_selection=f.readlines()  
naipEntityID\_selection=[line.rstrip() for line in naipEntityID\_selection]  
naipEntityID\_selection

['m\_3807505\_se\_18\_060\_20180827.tif',  
 'm\_3807505\_sw\_18\_060\_20180815.tif',  
 'm\_3807504\_se\_18\_060\_20180815.tif',  
 'm\_3807504\_sw\_18\_060\_20180815.tif',  
 'm\_3807532\_nw\_18\_060\_20180815.tif',  
 'm\_3807531\_nw\_18\_060\_20180815.tif',  
 'm\_3807531\_ne\_18\_060\_20180815.tif']

naip\_38075\_url=("https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/38075/")  
  
naip\_download\_rul=lambda url,tile,root:download\_url(url+tile,root)  
for tile in naipEntityID\_selection:  
 try:  
 naip\_download\_rul(naip\_38075\_url,tile,imagery\_data)  
 except:  
 print(f'Can not access to:{tile}')

Using downloaded and verified file: /content/gdrive/MyDrive/dataset/naipNdelaware4seg/imagery/m\_3807505\_se\_18\_060\_20180827.tif  
Using downloaded and verified file: /content/gdrive/MyDrive/dataset/naipNdelaware4seg/imagery/m\_3807505\_sw\_18\_060\_20180815.tif  
Using downloaded and verified file: /content/gdrive/MyDrive/dataset/naipNdelaware4seg/imagery/m\_3807504\_se\_18\_060\_20180815.tif  
Using downloaded and verified file: /content/gdrive/MyDrive/dataset/naipNdelaware4seg/imagery/m\_3807504\_sw\_18\_060\_20180815.tif  
Downloading https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/38075/m\_3807532\_nw\_18\_060\_20180815.tif to /content/gdrive/MyDrive/dataset/naipNdelaware4seg/imagery/m\_3807532\_nw\_18\_060\_20180815.tif  
  
  
100%|██████████| 489103340/489103340 [00:42<00:00, 11388534.38it/s]  
  
  
Downloading https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/38075/m\_3807531\_nw\_18\_060\_20180815.tif to /content/gdrive/MyDrive/dataset/naipNdelaware4seg/imagery/m\_3807531\_nw\_18\_060\_20180815.tif  
  
  
100%|██████████| 497914540/497914540 [00:44<00:00, 11290614.87it/s]  
  
  
Downloading https://naipeuwest.blob.core.windows.net/naip/v002/de/2018/de\_060cm\_2018/38075/m\_3807531\_ne\_18\_060\_20180815.tif to /content/gdrive/MyDrive/dataset/naipNdelaware4seg/imagery/m\_3807531\_ne\_18\_060\_20180815.tif  
  
  
100%|██████████| 467740726/467740726 [00:40<00:00, 11532384.15it/s]

* 对应NAIP的土地覆盖类型数据下载

TorchGeo库下载土地覆盖类型数据时，直接将其转换为训练数据集，这里单独下载为初始的ZIP压缩文件，解压后再单独建立训练数据集。

chesapeakebay\_landcover\_url="https://cicwebresources.blob.core.windows.net/chesapeakebaylandcover"  
base\_folder="DE"  
zipfile="\_DE\_STATEWIDE.zip"  
chesapeakebay\_landcover\_url += f"/{base\_folder}/{zipfile}"

download\_url(chesapeakebay\_landcover\_url, LC\_data, filename=zipfile)

Using downloaded and verified file: /content/gdrive/MyDrive/dataset/naipNdelaware4seg/LC/\_DE\_STATEWIDE.zip

* 迁移解压缩代码

因为下载的土地覆盖类型数据为ZIP格式压缩文件，因此迁移TorchGeo库已有解压缩代码，位于torchgeo.datasets.utils模块。

import tarfile  
from typing import (  
 Any,  
 Dict,  
 Iterable,  
 Iterator,  
 List,  
 Optional,  
 Sequence,  
 Tuple,  
 Union,  
 cast,  
 overload,  
)  
  
class \_rarfile:  
 class RarFile:  
 def \_\_init\_\_(self, \*args: Any, \*\*kwargs: Any) -> None:  
 self.args = args  
 self.kwargs = kwargs  
  
 def \_\_enter\_\_(self) -> Any:  
 try:  
 import rarfile  
 except ImportError:  
 raise ImportError(  
 "rarfile is not installed and is required to extract this dataset"  
 )  
  
 # TODO: catch exception for when rarfile is installed but not  
 # unrar/unar/bsdtar  
 return rarfile.RarFile(\*self.args, \*\*self.kwargs)  
  
 def \_\_exit\_\_(self, exc\_type: None, exc\_value: None, traceback: None) -> None:  
 pass  
  
  
class \_zipfile:  
 class ZipFile:  
 def \_\_init\_\_(self, \*args: Any, \*\*kwargs: Any) -> None:  
 self.args = args  
 self.kwargs = kwargs  
  
 def \_\_enter\_\_(self) -> Any:  
 try:  
 # Supports normal zip files, proprietary deflate64 compression algorithm  
 import zipfile\_deflate64 as zipfile  
 except ImportError:  
 # Only supports normal zip files  
 # https://github.com/python/mypy/issues/1153  
 import zipfile # type: ignore[no-redef]  
  
 return zipfile.ZipFile(\*self.args, \*\*self.kwargs)  
  
 def \_\_exit\_\_(self, exc\_type: None, exc\_value: None, traceback: None) -> None:  
 pass  
  
  
def extract\_archive(src: str, dst: Optional[str] = None) -> None:  
 """Extract an archive.  
  
 Args:  
 src: file to be extracted  
 dst: directory to extract to (defaults to dirname of ``src``)  
  
 Raises:  
 RuntimeError: if src file has unknown archival/compression scheme  
 """  
 if dst is None:  
 dst = os.path.dirname(src)  
  
 suffix\_and\_extractor: List[Tuple[Union[str, Tuple[str, ...]], Any]] = [  
 (".rar", \_rarfile.RarFile),  
 (  
 (".tar", ".tar.gz", ".tar.bz2", ".tar.xz", ".tgz", ".tbz2", ".tbz", ".txz"),  
 tarfile.open,  
 ),  
 (".zip", \_zipfile.ZipFile),  
 ]  
  
 for suffix, extractor in suffix\_and\_extractor:  
 if src.endswith(suffix):  
 with extractor(src, "r") as f:  
 f.extractall(dst)  
 return  
  
 suffix\_and\_decompressor: List[Tuple[str, Any]] = [  
 (".bz2", bz2.open),  
 (".gz", gzip.open),  
 (".xz", lzma.open),  
 ]  
  
 for suffix, decompressor in suffix\_and\_decompressor:  
 if src.endswith(suffix):  
 dst = os.path.join(dst, os.path.basename(src).replace(suffix, ""))  
 with decompressor(src, "rb") as sf, open(dst, "wb") as df:  
 df.write(sf.read())  
 return  
  
 raise RuntimeError("src file has unknown archival/compression scheme")

extract\_archive(os.path.join(LC\_data, zipfile))

* 构建包含图像增强变换的训练数据集

TorchGeo库设计时对图像增强变换的操作有两个途径，一个是构建训练数据集时，且融入到了具体的数据类型下，例如torchgeo.datasets.NAIP(root, crs=None, res=None, transforms=None, cache=True)集成了NAIP图像下载、图像增强变换和构建训练数据集等多个功能；另一个是用DataLoader加载小批量训练样本时，直接对小批量样本执行图像增强变换操作。在后续训练时调用pytorch\_lightning库提供的Trainer类，其fit(model, train\_dataloaders=None, val\_dataloaders=None, datamodule=None, ckpt\_path=None)方法提供datamodule，为LightningDataModule类的一个实例对象。TorchGeop构建的NAIPChesapeakeDataModule(pl.LightningDataModule)类继承了LightningDataModule类，NAIPChesapeakeDataModule构建时融合了土地覆盖类型数据的下载，并构建训练数据集，和NAIP训练数据集的构建，及一些图像增强变换操作，这些融合的方式虽然大幅度增加了数据处理的效率，但混杂也带来了很多数据处理分步的不清晰，和不便利，因此下述迁移更新了NAIPChesapeakeDataModule类，定义为为geo\_datamodule(pl.LightningDataModule)。新定义的geo\_datamodule类，拆离了训练数据集构建，从而把图像增强变换仅作用于训练数据集构建过程中，而不再混杂于datamodule的构建过程。这样也方便观察图像增强变换后的数据结果，及用于图像预测时输入数据的变换。

土地覆盖类型中，树冠类有很多子类很难通过航拍影像识别，如果仍然保留进行训练将会影响训练收敛和预测精度，因此定义labes\_merge()函数作为图像增强变换输入条件，合并部分分类。下述定义的变量label\_merge\_mapping合并了树冠类的各个子类，及灌丛和低矮植被，和不透水人造对象。同时，调整了标签数值，排序为1至7，缩减为7个土地覆盖类；另，如果读取区域为空值，对应值则为 0。因此分类数量由13类缩减为8类（含空值类），以便提高训练收敛的速度及预测的精度。

# 迁移于TorchGeo库：torchgeo.datamodules.naip模块  
  
def chesapeake\_transform(sample: Dict[str, Any]) -> Dict[str, Any]:  
 """Transform a single sample from the Chesapeake Dataset.  
  
 Args:  
 sample: Chesapeake mask dictionary  
  
 Returns:  
 preprocessed Chesapeake data  
 """  
 sample["mask"] = sample["mask"].long()[0]  
  
 return sample  
  
def remove\_bbox(sample: Dict[str, Any]) -> Dict[str, Any]:  
 """Removes the bounding box property from a sample.  
  
 Args:  
 sample: dictionary with geographic metadata  
  
 Returns  
 sample without the bbox property  
 """  
 del sample["bbox"]  
 return sample  
  
def labes\_merge(sample: Dict[str, Any]) -> Dict[str, Any]:  
 """Merge land cover classifications.  
  
 Args:  
 sample: dictionary with geographic metadata  
  
 Returns  
 sample without the bbox property  
 """  
 label\_merge\_mapping={  
 0:0, # Null  
 1:1, # Water ->Water  
 2:2, # Emergent Wetlands -> Emergent Wetlands  
 3:3, # Tree Canopy -> Tree Canopy  
 4:4, # Scrub/Shrub -> Scrub/Shrub  
 5:4, # Low Vegetation -> Scrub/Shrub  
 6:5, # Barren -> Barren(6->5)  
 7:6, # Impervious Structures -> Impervious Structures(7->6)  
 8:6, # Other Impervious -> Impervious Structures(7->6)  
 9:6, # Impervious Road -> Impervious Road(9->7)  
 10:3, # Tree Canopy over Impervious Structure -> Tree Canopy  
 11:3, # Tree Canopy over Other Impervious -> Tree Canopy  
 12:3, # Tree Canopy over Impervious Roads -> Tree Canopy  
 254:7, # Aberdeen Proving Ground -> Aberdeen Proving Ground(254->7)  
 }  
  
 old=sample["mask"]  
 indexer=np.array([label\_merge\_mapping.get(i, -1) for i in range(old.min(), old.max() + 1)])  
 new=torch.from\_numpy(indexer[(old - old.min())])  
 sample["mask"]=new  
 return sample

TorchGeo定义RasterDataset类时，输入的参数部分为全局变量（位于类下而于函数外），因此定义类的方式对变量进行赋值，例如对土地覆盖类型构建TorchGeo的RasterDataset栅格数据集，建立类delaware\_lc\_rd(RasterDataset)，继承RasterDataset类，并对父类的filename\_glob变量进行更新。同时增加迁移的上述两个变换chesapeake\_transform和remove\_bbox。

chesapeak\_transforms=Compose([chesapeake\_transform,remove\_bbox,labes\_merge])  
  
class delaware\_lc\_rd(RasterDataset):  
 filename\_glob="DE\_STATEWIDE.tif"  
 is\_image=False  
  
chesapeake=delaware\_lc\_rd(LC\_data,transforms=chesapeak\_transforms)

print(f"crs:{chesapeake.crs}\nres:{chesapeake.res}")

crs:ESRI:102039  
res:1.0

同样的方式操作NAIP图像，在图像增强变换时，增加了 NDVI 一个指数层（同时也备用给出了 NDWI）。因为TorchGeo的transforms模块中计算指数时，将其结果追加到 dim=0 维度上，从而构建数据加载器时，本应输入维度为 4 维，变成了 5 维，如(1,1,5,256,256)，而应为(1,5,256,256)，因此直接迁移TorchGeo库对应代码，修改AppendNormalizedDifferenceIndex类的返回值return input为return input[0]，并直接调用。

图像增强变换的内容和组合结构需要根据训练效果不断调整实验，使得训练收敛且提高预测精度。

from typing import Optional  
  
import torch  
from kornia.augmentation import IntensityAugmentationBase2D  
from torch import Tensor  
  
\_EPSILON = 1e-10  
  
  
class AppendNormalizedDifferenceIndex(IntensityAugmentationBase2D):  
 r"""Append normalized difference index as channel to image tensor.  
  
 Computes the following index:  
  
 .. math::  
  
 \text{NDI} = \frac{A - B}{A + B}  
  
 .. versionadded:: 0.2  
 """  
  
 def \_\_init\_\_(self, index\_a: int, index\_b: int) -> None:  
 """Initialize a new transform instance.  
  
 Args:  
 index\_a: reference band channel index  
 index\_b: difference band channel index  
 """  
 super().\_\_init\_\_(p=1)  
 self.flags = {"index\_a": index\_a, "index\_b": index\_b}  
  
 def apply\_transform(  
 self,  
 input: Tensor,  
 params: dict[str, Tensor],  
 flags: dict[str, int],  
 transform: Optional[Tensor] = None,  
 ) -> Tensor:  
 """Apply the transform.  
  
 Args:  
 input: the input tensor  
 params: generated parameters  
 flags: static parameters  
 transform: the geometric transformation tensor  
  
 Returns:  
 the augmented input  
 """  
 band\_a = input[..., flags["index\_a"], :, :]  
 band\_b = input[..., flags["index\_b"], :, :]  
 ndi = (band\_a - band\_b) / (band\_a + band\_b + \_EPSILON)  
 ndi = torch.unsqueeze(ndi, -3)  
 input = torch.cat((input, ndi), dim=-3)  
 return input[0]  
  
class AppendNDVI(AppendNormalizedDifferenceIndex):  
 r"""Normalized Difference Vegetation Index (NDVI).  
  
 Computes the following index:  
  
 .. math::  
  
 \text{NDVI} = \frac{\text{NIR} - \text{R}}{\text{NIR} + \text{R}}  
  
 If you use this index in your research, please cite the following paper:  
  
 \* https://doi.org/10.1016/0034-4257(79)90013-0  
 """  
  
 def \_\_init\_\_(self, index\_nir: int, index\_red: int) -> None:  
 """Initialize a new transform instance.  
  
 Args:  
 index\_nir: index of the Near Infrared (NIR) band in the image  
 index\_red: index of the Red band in the image  
 """  
 super().\_\_init\_\_(index\_a=index\_nir, index\_b=index\_red)  
  
class AppendNDWI(AppendNormalizedDifferenceIndex):  
 r"""Normalized Difference Water Index (NDWI).  
  
 Computes the following index:  
  
 .. math::  
  
 \text{NDWI} = \frac{\text{G} - \text{NIR}}{\text{G} + \text{NIR}}  
  
 If you use this index in your research, please cite the following paper:  
  
 \* https://doi.org/10.1080/01431169608948714  
 """  
  
 def \_\_init\_\_(self, index\_green: int, index\_nir: int) -> None:  
 """Initialize a new transform instance.  
  
 Args:  
 index\_green: index of the Green band in the image  
 index\_nir: index of the Near Infrared (NIR) band in the image  
 """  
 super().\_\_init\_\_(index\_a=index\_green, index\_b=index\_nir)

定义继承父类RasterDataset的naip\_rd类实现 NAIP 影像数据集定义。图像增强变换除了应用 NDVI 等指数试验外，增加了自定义的naip\_preprocess()变换，将图像 4 个波段值归一化。

def naip\_preprocess(sample: Dict[str, Any]) -> Dict[str, Any]:  
 """Transform a single sample from the NAIP Dataset.  
  
 Args:  
 sample: NAIP image dictionary  
  
 Returns:  
 preprocessed NAIP data  
 """  
 sample["image"] = sample["image"].float()  
 sample["image"] /= 255.0  
  
 return sample  
  
class naip\_rd(RasterDataset):  
 filename\_glob = "m\_\*.\*"  
 filename\_regex = r"""  
 ^m  
 \_(?P<quadrangle>\d+)  
 \_(?P<quarter\_quad>[a-z]+)  
 \_(?P<utm\_zone>\d+)  
 \_(?P<resolution>\d+)  
 \_(?P<date>\d+)  
 (?:\_(?P<processing\_date>\d+))?  
 \..\*$  
 """  
 is\_image=True  
 all\_bands = ["R", "G", "B", "NIR"]  
 rgb\_bands = ["R", "G", "B"]  
  
ndvi=AppendNDVI(index\_nir=3, index\_red=0)  
ndwi=AppendNDWI(index\_green=1, index\_nir=3)  
  
naip\_transforms=Compose([  
 naip\_preprocess,  
 remove\_bbox,  
 AugmentationSequential(ndvi,data\_keys=["image"])  
 ])  
  
naip=naip\_rd(imagery\_data,chesapeake.crs,chesapeake.res,transforms=naip\_transforms)

print(f"crs:{naip.crs}\nres:{naip.res}")

crs:ESRI:102039  
res:1.0

构建包含图像和类标的训练数据集。TorchGeo库重写了\_\_and\_\_、\_\_or\_\_等方法，可以直接对地理空间数据集执行交集和并集的操作。

dataset=chesapeake & naip

* 查看图像增强变换后的数据集样本

因为增加了 NDVI 一个指数层，因此数组的维度为[1, 5, 256, 256]，即包括R、G、B、Nir及NDVI等5个波段。

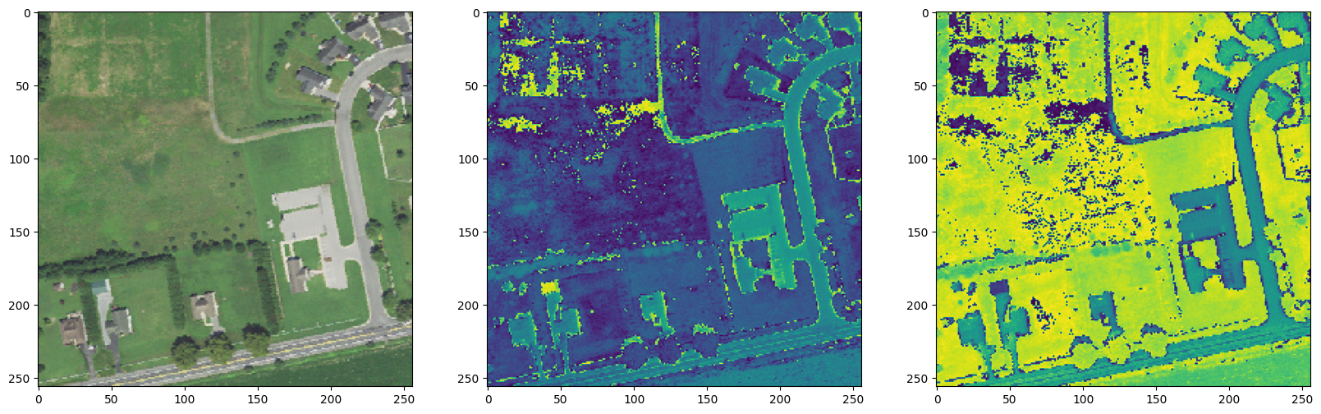
sampler=RandomGeoSampler(dataset,size=256, length=10)  
dataloader=DataLoader(dataset, sampler=sampler, collate\_fn=stack\_samples)  
  
for batch in dataloader:  
 sample = unbind\_samples(batch)[0]  
 print(sample.keys())  
 print(sample['image'].shape)  
 break

dict\_keys(['crs', 'mask', 'image'])  
torch.Size([5, 256, 256])

注意下述所用生成的数据集含有 NDWI 指数，即含 6 个通道。

sampler=RandomGeoSampler(dataset,size=256, length=10)  
dataloader=DataLoader(dataset, sampler=sampler, collate\_fn=stack\_samples)  
  
for batch in dataloader:  
 sample = unbind\_samples(batch)[0]  
 print(sample.keys())  
 print(sample['image'].shape)  
  
 fig, axes= plt.subplots(1, 3,figsize=(20,10))  
 axes[0].imshow(T.ToPILImage()(sample['image'][:3]))  
 axes[1].imshow(T.ToPILImage()(sample['image'][4]))  
 axes[2].imshow(T.ToPILImage()(sample['image'][5]))  
 plt.show()  
  
 break

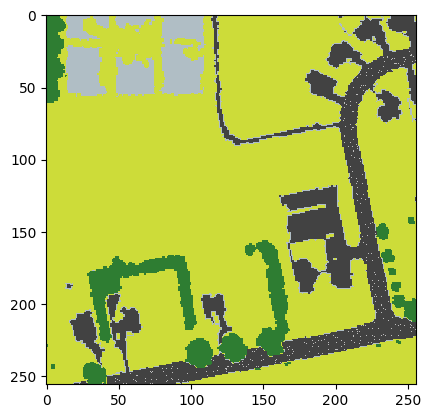
dict\_keys(['crs', 'mask', 'image'])  
torch.Size([6, 256, 256])



np.unique(sample['mask'])

array([3, 4, 5, 6, 7])

LC\_color\_dict={  
 # 0: (0, 0, 0, 0),  
 1: (30, 136, 229,255),  
 2: (230, 238, 156, 255),  
 3: (46, 125, 50,255),  
 4: (205, 220, 57, 255),  
 5: (176, 190, 197, 255),  
 6: (66, 66, 66, 255),  
 7: (189, 189, 189, 255),  
 }  
  
LC\_color\_dict\_selection={k:LC\_color\_dict[k] for k,v in LC\_color\_dict.items()}  
cmap\_LC, norm=matplotlib.colors.from\_levels\_and\_colors(list(LC\_color\_dict\_selection.keys()),[[v/255 for v in i] for i in LC\_color\_dict\_selection.values()],extend='max')  
plt.imshow(np.squeeze(sample['mask']),cmap=cmap\_LC);



* 更新datamodule类

新定义的geo\_datamodule类，移除了原NAIPChesapeakeDataModule类中的数据集前期处理部分内容（包括数据下载、图像增强变换等），而集中于训练数据切分为训练数据集（train\_dataloader）、测试数据集（test\_dataloader）和验证数据集（val\_dataloader）的Dataloder对象的构建上。

class Geo\_datamodule(LightningDataModule):  
 """LightningDataModule implementation for the TorchGeo datasets.  
  
 Uses the train/val/test splits from the dataset.  
 """  
 # TODO: tune these hyperparams  
 length=1000  
 stride=128  
  
 def \_\_init\_\_(  
 self,  
 ds\_image: torchgeo.datasets,  
 ds\_label: torchgeo.datasets,  
 batch\_size: int=64,  
 num\_workers: int=0,  
 patch\_size: int=256,  
 \*\*kwargs: Any,  
 ) -> None:  
 """Initialize a LightningDataModule for TorchGeo dataset based DataLoaders.  
  
 Args:  
 ds\_image: Image dataset in TorchGeo RasterDataset format  
 ds\_label: Target(label) dataset in TorchGeo RasterDataset format  
 batch\_size: The batch size to use in all created DataLoaders  
 num\_workers: The number of workers to use in all created DataLoaders  
 patch\_size: size of patches to sample  
 """  
 super().\_\_init\_\_()  
 self.ds\_image=ds\_image  
 self.ds\_label=ds\_label  
 self.batch\_size=batch\_size  
 self.num\_workers=num\_workers  
 self.patch\_size=patch\_size  
  
 def setup(self, stage: Optional[str] = None) -> None:  
 """Initialize the main ``Dataset`` objects.  
  
 This method is called once per GPU per run.  
  
 Args:  
 stage: state to set up  
 """  
 # TODO: these transforms will be applied independently, this won't work if we  
 # add things like random horizontal flip  
  
 # TODO: figure out better train/val/test split  
 self.dataset = self.ds\_label & self.ds\_image  
 roi=self.dataset.bounds  
 midx=roi.minx + (roi.maxx - roi.minx) / 2  
 midy=roi.miny + (roi.maxy - roi.miny) / 2  
 train\_roi=BoundingBox(roi.minx, midx, roi.miny, roi.maxy, roi.mint, roi.maxt)  
 val\_roi=BoundingBox(midx, roi.maxx, roi.miny, midy, roi.mint, roi.maxt)  
 test\_roi=BoundingBox(roi.minx, roi.maxx, midy, roi.maxy, roi.mint, roi.maxt)  
  
 self.train\_sampler=RandomBatchGeoSampler(  
 self.ds\_image, self.patch\_size, self.batch\_size, self.length, train\_roi  
 )  
 self.val\_sampler = GridGeoSampler(self.ds\_image, self.patch\_size, self.stride, val\_roi)  
 self.test\_sampler = GridGeoSampler(self.ds\_image, self.patch\_size, self.stride, test\_roi)  
  
 def train\_dataloader(self) -> DataLoader[Any]:  
 """Return a DataLoader for training.  
  
 Returns:  
 training data loader  
 """  
  
 return DataLoader(  
 self.dataset,  
 batch\_sampler=self.train\_sampler,  
 num\_workers=self.num\_workers,  
 collate\_fn=stack\_samples,  
 )  
  
 def val\_dataloader(self) -> DataLoader[Any]:  
 """Return a DataLoader for validation.  
  
 Returns:  
 validation data loader  
 """  
  
 return DataLoader(  
 self.dataset,  
 batch\_size=self.batch\_size,  
 sampler=self.val\_sampler,  
 num\_workers=self.num\_workers,  
 collate\_fn=stack\_samples,  
 )  
  
 def test\_dataloader(self) -> DataLoader[Any]:  
 """Return a DataLoader for testing.  
  
 Returns:  
 testing data loader  
 """  
  
 return DataLoader(  
 self.dataset,  
 batch\_size=self.batch\_size,  
 sampler=self.test\_sampler,  
 num\_workers=self.num\_workers,  
 collate\_fn=stack\_samples,  
 )  
  
 def plot(  
 self,  
 sample: dict[str, Any],  
 show\_titles: bool = True,  
 suptitle: Optional[str] = None,  
 ) -> plt.Figure:  
 """Plot a sample from the dataset.  
  
 Args:  
 sample: a sample returned by :meth:`RasterDataset.\_\_getitem\_\_`  
 show\_titles: flag indicating whether to show titles above each panel  
 suptitle: optional string to use as a suptitle  
  
 Returns:  
 a matplotlib Figure with the rendered sample  
  
 .. versionchanged:: 0.3  
 Method now takes a sample dict, not a Tensor. Additionally, possible to  
 show subplot titles and/or use a custom suptitle.  
 """  
 image = sample["image"][0:3, :, :].permute(1, 2, 0)  
  
 fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(4, 4))  
  
 ax.imshow(image)  
 ax.axis("off")  
 if show\_titles:  
 ax.set\_title("Image")  
  
 if suptitle is not None:  
 plt.suptitle(suptitle)  
  
 return fig

datamodule=Geo\_datamodule(  
 ds\_image=naip,  
 ds\_label=chesapeake,  
 batch\_size=64,  
 patch\_size=256  
 )

* 构建深度学习网络模型

TorchGeo的深度学习网络模型构建SemanticSegmentationTask类继承了pytorch\_lightning的LightningModule类，并调用了segmentation\_models\_pytorch库的unet模型。配置了aux\_params参数，返回掩码和分类标签。因为增加了 NDVI 一个指数层，因此总共5个通道，配置in\_channels=5。调整后的土地覆盖类型共计8类，因此配置num\_classes=8。

aux\_params=dict(  
 pooling='avg', # one of 'avg', 'max'  
 dropout=0.5, # dropout ratio, default is None  
 activation='sigmoid', # activation function, default is None  
 )  
  
task=SemanticSegmentationTask(  
 model='unet',  
 backbone='resnet34',  
 weights='imagenet',  
 pretrained=True,  
 in\_channels=5,   
 num\_classes=8,  
 ignore\_index=0,  
 loss='ce', # 'jaccard'  
 learning\_rate=0.1,  
 learning\_rate\_schedule\_patience=5,  
 aux\_params=aux\_params,  
 )

* 配置训练参数

PyTorch Lightning库提供的Trainer方法大幅度简化了训练参数的配置，如果不希望部分参数自动化处理，也可以自行配置。下述配置了ModelCheckpoint，通过配置moniter参数监测给定参数的值，定期保存模型。在LightningModule中使用log()或log\_dict()记录的每个指标都可以为监控的对象；配置EarlyStopping，监控给定的指标，当该指标停止改进时则停止训练，这非常有利于了解到模型训练的程度，避免无效的训练，利于反复调参实验；同时配置了训练文件、过程网络模型等保存的本地位置（本次存储到 Google Drive 中）default\_root\_dir；用min\_epochs和max\_epochs配置最小和最大的迭代次数；fast\_dev\_run为一种单元测试；配置accelerator="gpu"，支持传递不同的加速器类型，例如“cpu”, “gpu”, “tpu”, “ipu”, “hpu”, “mps, “auto”，及自定义的加速器实例等。Trainer方法提供的参数高达50多个，可以从官网查看应用。

在配置日志时，给出了两种途径，一种是使用CSVLogger方法，以 YAML 或 CSV 格式存储测量指标，用于后续读取分析；另一种是使用TensorBoardLogger方法，后续可以直接用于训练报告查看。

accelerator = "gpu" if torch.cuda.is\_available() else "cpu"  
  
checkpoint\_callback=ModelCheckpoint(monitor="val\_loss", dirpath=data\_dir, save\_top\_k=1, save\_last=True) #,save\_on\_train\_epoch\_end=True  
early\_stopping\_callback=EarlyStopping(monitor="val\_loss", min\_delta=0.00, patience=10)  
# logger=CSVLogger(save\_dir=data\_dir, name="segmentation\_unet")  
logger=TensorBoardLogger(save\_dir=data\_dir, name="tensorboard\_logs",version=1,)  
in\_tests="PYTEST\_CURRENT\_TEST" in os.environ  
  
trainer=Trainer(  
 callbacks=[checkpoint\_callback,early\_stopping\_callback],  
 logger=logger,  
 default\_root\_dir=data\_dir,  
 min\_epochs=1,  
 max\_epochs=300,  
 fast\_dev\_run=in\_tests,  
 accelerator=accelerator,  
 #limit\_val\_batches=500,  
 )

INFO:pytorch\_lightning.utilities.rank\_zero:GPU available: True (cuda), used: True  
INFO:pytorch\_lightning.utilities.rank\_zero:TPU available: False, using: 0 TPU cores  
INFO:pytorch\_lightning.utilities.rank\_zero:IPU available: False, using: 0 IPUs  
INFO:pytorch\_lightning.utilities.rank\_zero:HPU available: False, using: 0 HPUs

* 网络模型训练

fit方法传入的参数ckpt\_path，只对已经训练且保存的模型权重值有效，因此需要检查该文件是否已经存在。利用CoLab提供的GPU算力，模型达到给定指标不再有效提升时所需要的时间约为1个小时42分钟（CoLab Pro版）。

ckpt\_path=os.path.join(data\_dir,'last.ckpt')  
if path.exists(ckpt\_path):  
 print('ckpt file exists.')  
 trainer.fit(model=task,datamodule=datamodule,ckpt\_path=ckpt\_path)  
else:  
 trainer.fit(model=task,datamodule=datamodule)

/usr/local/lib/python3.10/dist-packages/lightning/pytorch/callbacks/model\_checkpoint.py:613: UserWarning: Checkpoint directory /content/gdrive/MyDrive/dataset/naipNdelaware4seg/training exists and is not empty.  
 rank\_zero\_warn(f"Checkpoint directory {dirpath} exists and is not empty.")  
INFO: LOCAL\_RANK: 0 - CUDA\_VISIBLE\_DEVICES: [0]  
INFO:lightning.pytorch.accelerators.cuda:LOCAL\_RANK: 0 - CUDA\_VISIBLE\_DEVICES: [0]  
INFO:   
 | Name | Type | Params  
---------------------------------------------------  
0 | model | Unet | 24.4 M  
1 | loss | CrossEntropyLoss | 0   
2 | train\_metrics | MetricCollection | 0   
3 | val\_metrics | MetricCollection | 0   
4 | test\_metrics | MetricCollection | 0   
---------------------------------------------------  
24.4 M Trainable params  
0 Non-trainable params  
24.4 M Total params  
97.775 Total estimated model params size (MB)  
INFO:lightning.pytorch.callbacks.model\_summary:  
 | Name | Type | Params  
---------------------------------------------------  
0 | model | Unet | 24.4 M  
1 | loss | CrossEntropyLoss | 0   
2 | train\_metrics | MetricCollection | 0   
3 | val\_metrics | MetricCollection | 0   
4 | test\_metrics | MetricCollection | 0   
---------------------------------------------------  
24.4 M Trainable params  
0 Non-trainable params  
24.4 M Total params  
97.775 Total estimated model params size (MB)  
Sanity Checking: 0it [00:00, ?it/s]

* 从日志（Log）中提取训练测量指标

1）读取CSVLogger方法的日志

测量指标已经存入Google Drive文件夹下，读取metrics.csv文件数据，并打印Train loss和Validation Accuracy指标，可以初步判断深度学习网络模型有效收敛，且验证数据集上的预测精度能够达到约0.8以上。

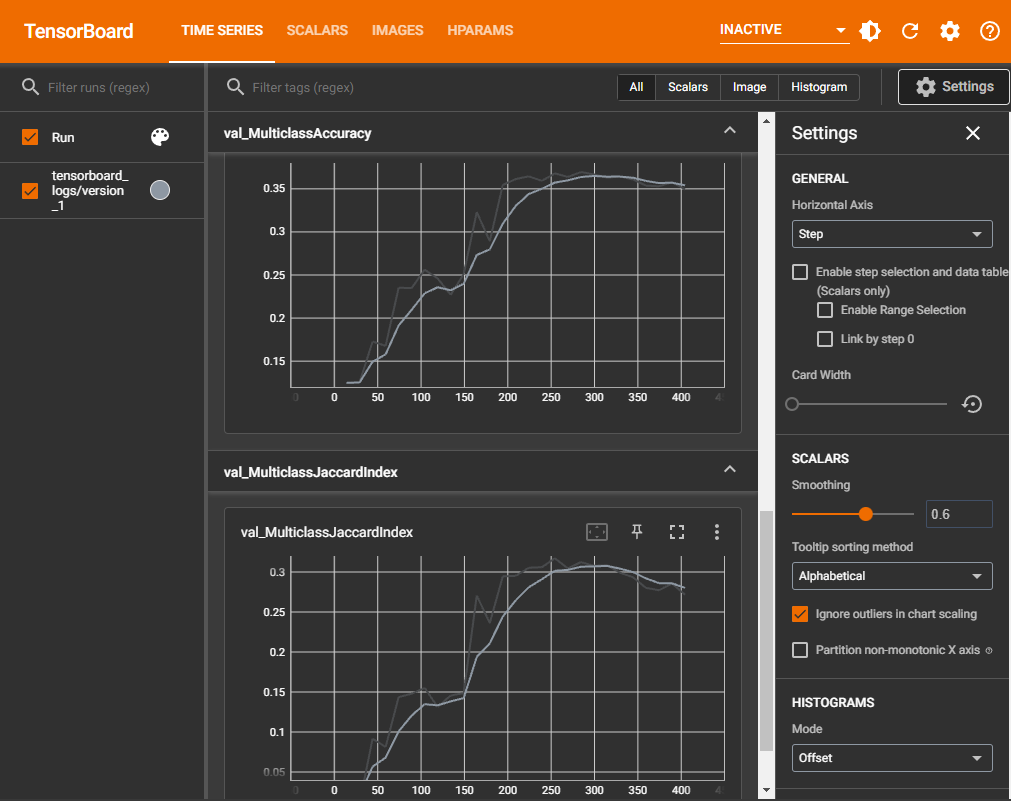
if not in\_tests:  
 train\_steps = []  
 train\_rmse = []  
  
 val\_steps = []  
 val\_rmse = []  
 with open(  
 os.path.join(data\_dir, "segmentation\_unet", "version\_0", "metrics.csv"), "r"   
 ) as f:  
 csv\_reader = csv.DictReader(f, delimiter=",")  
 for i, row in enumerate(csv\_reader):  
 try:  
 train\_rmse.append(float(row["train\_loss"]))  
 train\_steps.append(i)  
 except ValueError: # Ignore rows where train RMSE is empty  
 pass  
  
 try:  
 val\_rmse.append(float(row["val\_Accuracy"]))  
 val\_steps.append(i)  
 except ValueError: # Ignore rows where val RMSE is empty  
 pass

if not in\_tests:  
 plt.figure()  
 plt.plot(train\_steps, train\_rmse, label="Train loss")  
 plt.plot(val\_steps, val\_rmse, label="Validation Accuracy")  
 plt.legend(fontsize=15)  
 plt.xlabel("Batches", fontsize=15)  
 plt.ylabel("Loss", fontsize=15)  
 plt.show()  
 plt.close()



2）读取TensorBoardLogger方法的日志

%tensorboard --logdir "$data\_dir"



* TorchMetrics 指标度量库

[TorchMetrics](https://torchmetrics.readthedocs.io/en/stable/)⑲ 集合了 100+ 个指标实现，且提供了一个易于使用的 API（Application Programming Interface）用于自定义指标。该 API 提供了可以增加再现性（reproducibility）的一个标准化接口；并兼容分布式训练；可以在批次（batch）间自动积累；在多个设备间自动同步；并经过了严格的测试。

上述影像分割实验中，TorchGeo库在SemanticSegmentationTask类中引入了TorchMetrics提供的MulticlassAccuracy和MulticlassJaccardIndex两个指标。

1）MulticlassAccuracy 为多分类精度计算，公式为，式中，为真实标签（目标值），为预测值。如上图val\_MulticlassAccuracy标签部分；

2）MulticlassJaccardIndex为多分类 Jaccard 指数，即交并集，也称为 Jaccard 相似系数，公式为。如有，

| 真实值 | A | B | C |
| --- | --- | --- | --- |
| **A** | AA | AB | Ac |
| **B** | BA | BB | BC |
| **C** | CA | CB | CC |

则多分类 Jaccard 指数计算可表述为；Jaccard 平均得分计算可表述为。

* 测试数据集测试

用测试数据集测试，测试结果显示测试精度test\_Accuracy为0.820。

trainer.test(model=task, datamodule=datamodule)

INFO: LOCAL\_RANK: 0 - CUDA\_VISIBLE\_DEVICES: [0]  
INFO:lightning.pytorch.accelerators.cuda:LOCAL\_RANK: 0 - CUDA\_VISIBLE\_DEVICES: [0]  
/usr/local/lib/python3.10/dist-packages/lightning/pytorch/trainer/connectors/data\_connector.py:224: PossibleUserWarning: The dataloader, test\_dataloader 0, does not have many workers which may be a bottleneck. Consider increasing the value of the `num\_workers` argument` (try 4 which is the number of cpus on this machine) in the `DataLoader` init to improve performance.  
 rank\_zero\_warn(  
  
  
  
Testing: 0it [00:00, ?it/s]

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓

┃ **Test metric** ┃ **DataLoader 0** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩

│ test\_MulticlassAccuracy │ 0.362870454788208 │

│ test\_MulticlassJaccardIndex │ 0.35294270515441895 │

│ test\_loss │ 0.45782798528671265 │

└─────────────────────────────┴─────────────────────────────┘

[{'test\_loss': 0.45782798528671265,  
 'test\_MulticlassAccuracy': 0.362870454788208,  
 'test\_MulticlassJaccardIndex': 0.35294270515441895}]

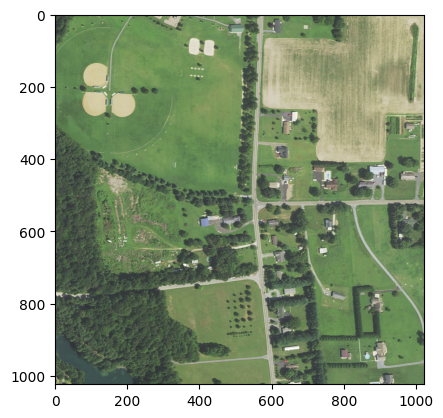
* 用训练的模型预测

这里并未使用用于训练下载的4个NAIP图像瓦片，而是下载了新的16个图像瓦片，新建立数据集，随机采样用已经训练好的模型进行预测。从预测结果来看，所提取的样本包括1、3、4、6三个分类，对应水体、林冠、灌木丛和不透水区域，目视分类图结果，能够较好的对原始的NAIP图像进行语义分割（解译）。

from torchgeo.samplers import GridGeoSampler,RandomGeoSampler  
from torchgeo.datasets import NAIP,stack\_samples  
from torch.utils.data import DataLoader  
import torch  
  
import matplotlib.pyplot as plt  
import torchvision.transforms as T  
import matplotlib  
import numpy as np

X\_pre=NAIP(r'E:\data\Delaware\imagery\_test',transforms=naip\_transforms)  
X\_sample=RandomGeoSampler(X\_pre, size=1024,length=100)   
X\_dataloader=DataLoader(X\_pre, sampler=X\_sample, collate\_fn=stack\_samples)  
X\_dataloader\_=iter(X\_dataloader)

X\_batch=next(X\_dataloader\_)  
X=X\_batch["image"] #.float()  
  
plt.imshow(T.ToPILImage()(X[0][:3]));



unet\_model=task.load\_from\_checkpoint(os.path.join(data\_dir,'last.ckpt'))  
unet\_model.freeze()

y\_probs=unet\_model(X)  
print(y\_probs.shape)

torch.Size([1, 8, 1024, 1024])

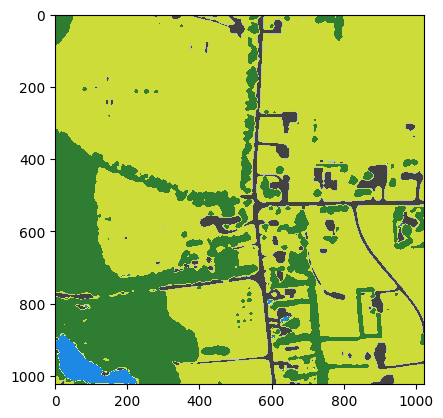
将联合预测概率转换为对应的土地覆盖类型标签（即概率最大对应的列）。

import numpy as np  
y\_pred=np.argmax(y\_probs,axis=1)#.reshape(-1,1)  
print(y\_pred.shape,'\n',y\_pred,'\n',y\_pred.unique())

torch.Size([1, 1024, 1024])   
 tensor([[[4, 4, 4, ..., 4, 4, 4],  
 [4, 4, 4, ..., 4, 4, 4],  
 [4, 4, 3, ..., 4, 4, 4],  
 ...,  
 [4, 4, 4, ..., 4, 4, 4],  
 [4, 4, 4, ..., 4, 4, 4],  
 [4, 4, 4, ..., 4, 4, 4]]])   
 tensor([1, 3, 4, 6])

LC\_color\_dict\_selection={k:LC\_color\_dict[k] for k,v in LC\_color\_dict.items()}  
cmap\_LC, norm=matplotlib.colors.from\_levels\_and\_colors(list(LC\_color\_dict\_selection.keys()),[[v/255 for v in i] for i in LC\_color\_dict\_selection.values()],extend='max')

plt.imshow(np.squeeze(y\_pred),cmap=cmap\_LC);



注释（Notes）：

① USGS EROS Archive - Aerial Photography - National Agriculture Imagery Program (NAIP)，（<https://www.usgs.gov/centers/eros/science/usgs-eros-archive-aerial-photography-national-agriculture-imagery-program-naip>）。

② EarthExplorer，（<https://earthexplorer.usgs.gov/>）。

③ 行星计算机（Planetary Computer），（<https://planetarycomputer.microsoft.com/>）。

④ TorchGeo，（<https://torchgeo.readthedocs.io/en/stable/user/installation.html>）。

⑤ torchvision，（<https://pytorch.org/vision/stable/index.htm>）。

⑥ NAIP: National Agriculture Imagery Program，（<https://planetarycomputer.microsoft.com/dataset/naip>）。

⑦ Geospatial Datasets（TorchGeo），（<https://torchgeo.readthedocs.io/en/stable/api/datasets.html>）。

⑧ Chesapeake Conservancy，（<https://www.chesapeakeconservancy.org/conservation-innovation-center/high-resolution-data/>）。

⑨ Kornia，（<https://github.com/kornia/kornia>）。

⑩ ONNX Model Zoo，（<https://github.com/onnx/models>）。

⑪ torchvision.models，（<https://pytorch.org/vision/stable/models.html>）。

⑫ PyTorch Image Models，（<https://github.com/rwightman/pytorch-image-models>）。

⑬ Transformers，（<https://github.com/huggingface/transformers>）。

⑭ Segmentation Models，（<https://smp.readthedocs.io/en/stable/index.html>）。

⑮ ImageNet，（<https://www.image-net.org/index.php>）。

⑯ WordNet（Wikipedia），为200多种语言中单词之间语义关系的词汇数据库。（<https://en.wikipedia.org/wiki/WordNet>）。

⑰ Colaborator，CoLab，（<https://colab.research.google.com/>）。

⑱ Planetary Computer Hub，PcHub，（<https://pccompute.westeurope.cloudapp.azure.com/compute/hub>）。

⑲ TorchMetrics，(<https://torchmetrics.readthedocs.io/en/stable/>)。

参考文献（References）:

[1] Geospatial deep learning with TorchGeo，<https://pytorch.org/blog/geospatial-deep-learning-with-torchgeo/>。

[2] Conservation Innovation Center (CIC). LC Class Descriptions, <https://www.chesapeakeconservancy.org/conservation-innovation-center/high-resolution-data/lulc-data-project-2022/>.