# 2.6.6 高分辨率遥感影像解译

### 2.6.6.1 无监督土地分类（聚类方法）

遥感影像的各个波段记录了地物的相关信息，那么以波段的数据作为机器学习的训练数据集，喂入相关模型，可以对应解决相关问题。其中之一为使用聚类的方法初步实现无监督土地分类（K-Menas算法）。sentinel-2影像有多个波段，可以尝试使用单个波段，或者多个波段作为特征向量，对比波段的合成显示，估计不同输入数据聚类结果的效果。

sentinel-2影像的信息均记录于“MTD\_MSIL2A.xml”中（需要查看sentinel-2部分内容），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。

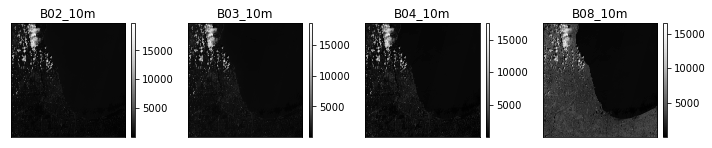
def Sentinel2\_bandFNs(MTD\_MSIL2A\_fn):   
 '''  
 funciton - 获取sentinel-2波段文件路径，和打印主要信息  
   
 Paras:  
 MTD\_MSIL2A\_fn - MTD\_MSIL2A 文件路径  
   
 Returns:  
 band\_fns\_list - 波段相对路径列表  
 band\_fns\_dict - 波段路径为值，反应波段信息的字段为键的字典  
 '''  
 import xml.etree.ElementTree as ET  
  
 Sentinel2\_tree=ET.parse(MTD\_MSIL2A\_fn)  
 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'}

根据返回字典的键，可以提取对应的波段路径名。EarthPy库的stack方法可以融合多个波段，同时会返回波段的元数据，包括：driver驱动，dtype数据类型，nodata空值，width影像宽度，height影像高度，count波段数量，crs坐标系统（投影），transform变换，blockxsizex向单元数量（每个单元的精度为10m，即一个像素代表10m的实际地理空间，小于10m的地物则无法分辨），blockysizey向单元数量。

import os  
import matplotlib.pyplot as plt  
import earthpy.spatial as es  
import earthpy.plot as ep  
import geopandas as gpd  
  
imgs\_root=r"G:\data\S2B\_MSIL2A\_20200709T163839\_N0214\_R126\_T16TDM\_20200709T211044.SAFE"  
bands\_selection=["B02\_10m","B03\_10m","B04\_10m","B08\_10m"]  
stack\_bands=[os.path.join(imgs\_root,band\_fns\_dict[b]) for b in bands\_selection]  
array\_stack, meta\_data=es.stack(stack\_bands)  
print("meta\_data:\n",meta\_data)  
  
ep.plot\_bands(array\_stack,title=bands\_selection,cols=array\_stack.shape[0],cbar=True,figsize=(10,10))  
plt.show()

meta\_data:  
 {'driver': 'JP2OpenJPEG', 'dtype': 'uint16', 'nodata': None, 'width': 10980, 'height': 10980, 'count': 4, 'crs': CRS.from\_epsg(32616), 'transform': Affine(10.0, 0.0, 399960.0,  
 0.0, -10.0, 4700040.0), 'blockxsize': 1024, 'blockysize': 1024, 'tiled': True}



在QGIS中读取一个波段，或多个波段的组合显示，绘制裁切边界（设置坐标为WGS84，不配置投影，读取后根据影像的投影再进行定义），用于影像的裁切。裁切文件保存于指定的文件夹下。

crop\_output\_dir=r'G:\data\data\_processed\sentinel-2\_crop'  
imgs\_crs=meta\_data['crs']  
  
shape\_polygon\_fp='./data/sentinel2Chicago\_boundary/sentinel2Chicago\_boundary.shp'  
crop\_bound=gpd.read\_file(shape\_polygon\_fp)  
crop\_bound\_proj=crop\_bound.to\_crs(imgs\_crs)  
  
crop\_imgs=es.crop\_all([os.path.join(imgs\_root,f+'.jp2') for f in band\_fns\_list], crop\_output\_dir, crop\_bound\_proj, overwrite=True) #对所有波段band执行裁切  
print("finished cropping...")

finished cropping...

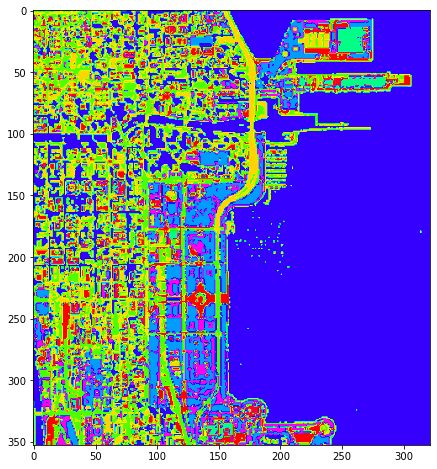
显示裁切后的影像。

import glob  
croppedImgs\_fns=glob.glob(crop\_output\_dir+"/\*.jp2")  
croppedBands\_fnsDict={f.split('\_')[-3]+'\_'+f.split('\_')[-2]:f for f in croppedImgs\_fns}  
  
bands\_selection\_=["B02\_10m","B03\_10m","B04\_10m","B08\_10m"] #,"AOT\_10m","WVP\_10m"  
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=(10,10))  
plt.show()

可以尝试调整不同的聚类数量n\_cluster参数，分类越多划分的地物类别也就越细。基于聚类无监督分类的结果并没有明确分类的名称，需要结合已经聚类的结果，根据实际地物情况判别。注意喂入模型数据的形状为（样本数，特征数）。如果理解为矩阵，则每一列为一个特征向量，每一行为一个样本的多个特征值。通常输入的特征数越多，波段数越多，分类的精度可能越好。

from sklearn import cluster  
import matplotlib.pyplot as plt  
  
X=cropped\_array\_stack.reshape(cropped\_array\_stack.shape[0],-1).transpose(1,0)  
print(X)  
  
k\_means=cluster.KMeans(n\_clusters=8)  
k\_means.fit(X)  
X\_cluster = k\_means.labels\_  
X\_cluster = X\_cluster.reshape(cropped\_array\_stack[0,:,:].shape)  
  
plt.figure(figsize=(8,8))  
plt.imshow(X\_cluster, cmap="hsv")  
  
plt.show()

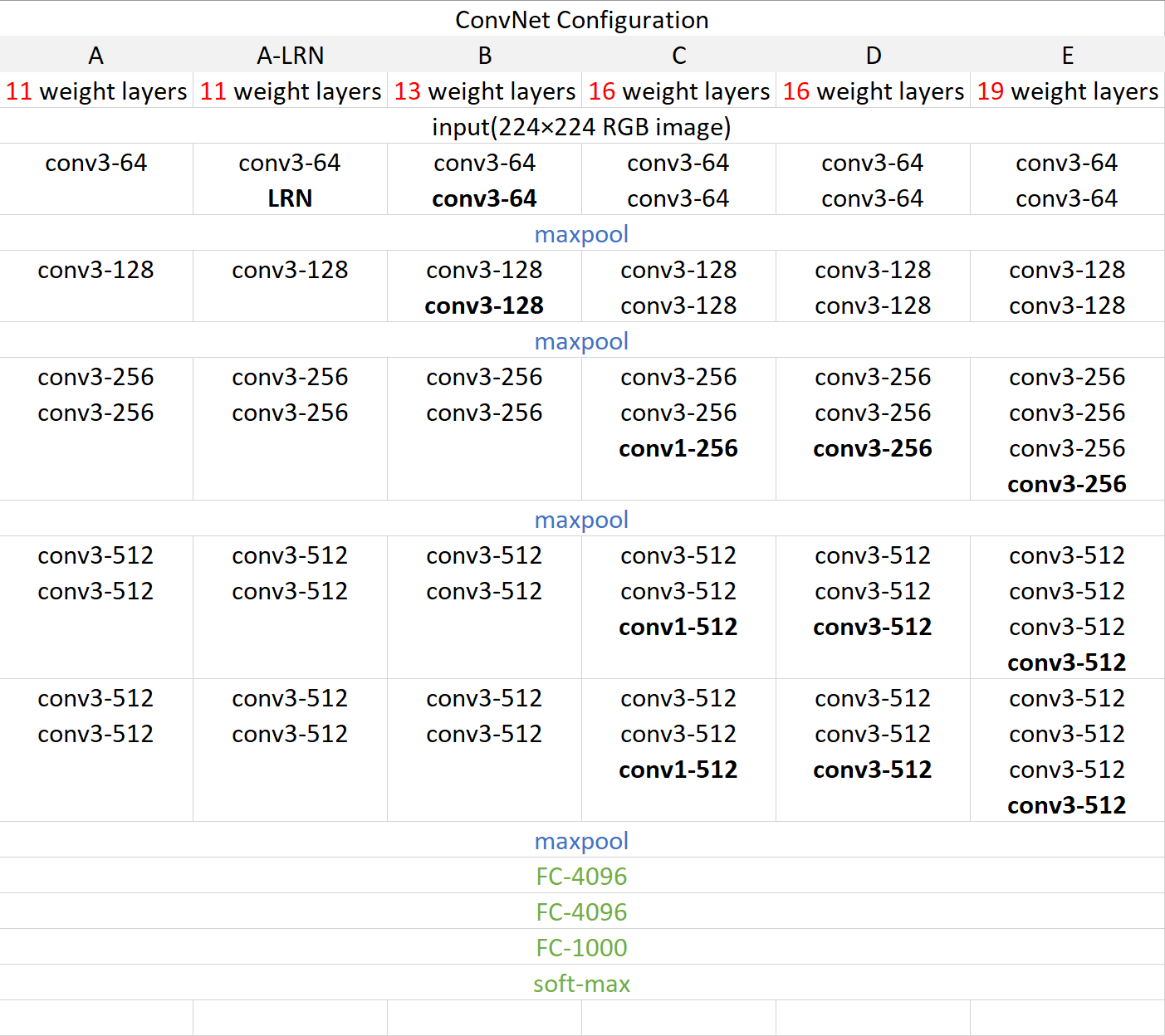
[[2972 3248 3344 3550]  
 [2574 2548 2602 2768]  
 [1406 1406 1824 1920]  
 ...  
 [ 825 749 519 484]  
 [ 811 738 516 498]  
 [ 776 760 538 488]]



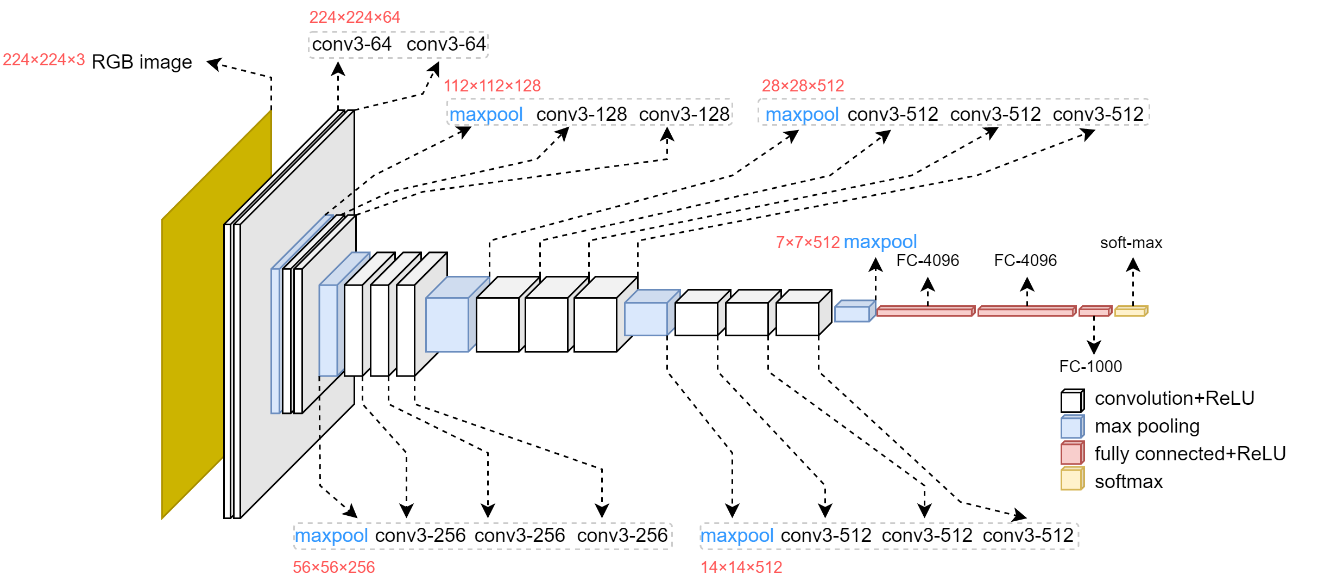
### 2.6.6.2 VGG16卷积神经网络

[VGGNet](https://arxiv.org/abs/1409.1556)[1]研究了在大规模图像识别环境下，卷积网络深度对识别精度的影响。主要贡献是使用非常小的()卷积滤波器（卷积核）和()的最大池化层反复堆叠，在深度不断增加的网络下的表现评估。当将网络深度推进到16-19个全支持层时（图表的第C、D列为16层，第E列为19层），可以发现识别精度得以显著提升。该项研究最初用于[ImageNet](http://www.image-net.org/)①数据集，并同时能够很好的泛化到其它数据集。

对VGGNet网络的理解同样可以对应到图像特征提取——尺度不变特征转换下尺度空间（scale space）的概念上。因为不同的地物尺寸不同，因此对于同一地理范围下的影像，分辨率越高，例如0.3~0.5m，则可以识别出行人轮廓。但是10m的高空分辨率则无法识别，对于通常大于10m的对象，例如建筑、绿地则可以识别。这个变换的分辨率就是尺度空间的纵向深度，由降采样实现。对应到VGG网络上，就是网络深度的不断增加，是由maxpool最大池化层实现。因为不同地物的尺寸多样，但是通常可以形成一个连续的尺寸变化，例如从室外摆放的餐具，过往或静坐的行人，到车辆、建筑，再到农田和成片的林地。因此为了检测到每一地物对应的尺度空间，采用的最大池化能够很好的捕捉到不同的地物。即低分辨率的图像可以忽略掉较小的对象，而专注于该尺度及之上的对象，以此类推。在尺度空间中还有一个水平向，使用不同的卷积核检测同一尺度即深度下地物即图像的特征。不同的卷积核会识别出不同的特征内容，例如对象间的边界形状，颜色的差异变化，及很多一般常识很难判定但却可以区分对象的特征。因此在每一深度进行卷积操作时，通常要使用多个不同的卷积核，并随机初始化卷积核数值，以捕捉到对象的特征。这对应到深度网络结构中的输出通道数。VGGNet在深度增加过程中，所使用的卷积核大小不变，均为。因为深度的逐层增加，不同尺度的地物会被捕捉到，同一大小的卷积核可以检测到不同地物的特征。同时，使用的卷积数量在增加，以适应深度增加，尺度增大，即图像越加模糊时的特征提取。图像的特征并不仅表现在一次卷积的结果上，例如，如果应用一次卷积提取了对象的轮廓边界，那么仍然可以再应用卷积提取对象轮廓边界的特征，以此类推。这可以用于解释每一层深度/尺度下使用多层卷积的原因。



将上述表格的第D列，即VGG16，通过方块序列图的形式可以更好的表述，观察层级间的变化。



* ImageNet数据集

ImageNet数据集于2007年开始建设，已有超过1500万张图像，2万多个类别，是一个庞大的数据集。是根据[WordNet](https://wordnet.princeton.edu/)②层次结构（目前只有名词nouns）组织的图像数据库。其中层次结构的每一个节点都由成百上千张图像描述。ImageNet数据集1000个类别文件可以从[imagenet\_classes.txt](https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt)③处下载，其分类涉及动植物，各类人造物。

VGGNet预训练模型已经置于[torchvision.models](https://pytorch.org/vision/0.8/models.html)④模型库中，通过下载该模型，来尝试识别对象。参考[VGG-NETS](https://pytorch.org/hub/pytorch_vision_vgg/)[2]。

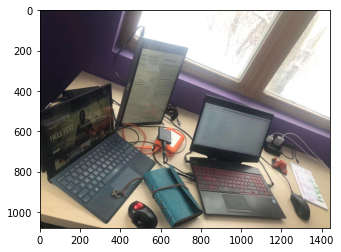
* 01-下载预训练的VGG16模型

import torch  
model=torch.hub.load('pytorch/vision:v0.6.0', 'vgg16', pretrained=True)  
model.eval()

Downloading: "https://github.com/pytorch/vision/archive/v0.6.0.zip" to C:\Users\richi/.cache\torch\hub\v0.6.0.zip  
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to C:\Users\richi/.cache\torch\hub\checkpoints\vgg16-397923af.pth  
  
  
  
 0%| | 0.00/528M [00:00<?, ?B/s]  
  
  
  
  
  
VGG(  
 (features): Sequential(  
 (0): Conv2d(3, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU(inplace=True)  
 (2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (3): ReLU(inplace=True)  
 (4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (5): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (6): ReLU(inplace=True)  
 (7): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (8): ReLU(inplace=True)  
 (9): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (10): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (11): ReLU(inplace=True)  
 (12): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (13): ReLU(inplace=True)  
 (14): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (15): ReLU(inplace=True)  
 (16): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (17): Conv2d(256, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (18): ReLU(inplace=True)  
 (19): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (20): ReLU(inplace=True)  
 (21): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (22): ReLU(inplace=True)  
 (23): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (24): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (25): ReLU(inplace=True)  
 (26): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (27): ReLU(inplace=True)  
 (28): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (29): ReLU(inplace=True)  
 (30): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (avgpool): AdaptiveAvgPool2d(output\_size=(7, 7))  
 (classifier): Sequential(  
 (0): Linear(in\_features=25088, out\_features=4096, bias=True)  
 (1): ReLU(inplace=True)  
 (2): Dropout(p=0.5, inplace=False)  
 (3): Linear(in\_features=4096, out\_features=4096, bias=True)  
 (4): ReLU(inplace=True)  
 (5): Dropout(p=0.5, inplace=False)  
 (6): Linear(in\_features=4096, out\_features=1000, bias=True)  
 )  
)

* 02-读取一幅图像。执行调整图像大小Resize、裁切CenterCrop、转换为张量ToTensor和标准化Normalize等操作，使其满足网络结构的数据输入需求。

from PIL import Image  
from torchvision import transforms  
import matplotlib.pyplot as plt  
import numpy as np  
  
cat\_01=r'./data/stuff\_01.jpg' #cat\_01.png;stuff\_01.jpg  
cat\_img=Image.open(cat\_01).convert('RGB')  
plt.imshow(cat\_img)  
plt.show()



input\_image=Image.open(cat\_01)  
preprocess=transforms.Compose([  
 transforms.Resize(256),  
 transforms.CenterCrop(224),  
 transforms.ToTensor(),  
 transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),  
])  
input\_tensor=preprocess(input\_image)  
input\_batch=input\_tensor.unsqueeze(0) # create a mini-batch as expected by the model  
print("VGG16输入数据的形状（batchsize, nChannels, Height, Width）：",input\_batch.shape)

VGG16输入数据的形状（batchsize, nChannels, Height, Width）： torch.Size([1, 3, 224, 224])

* 03 - 图像中的内容预测概率

# 输入数据和模型传入GPU执行运输 move the input and model to GPU for speed if available  
if torch.cuda.is\_available():  
 input\_batch=input\_batch.to('cuda')  
 model.to('cuda')  
  
with torch.no\_grad():  
 output=model(input\_batch)  
# 全连接最后一层的线性输出通道数为1000，对应ImageNet数据集的1000个分类 Tensor of shape 1000, with confidence scores over Imagenet's 1000 classes  
# The output has unnormalized scores. To get probabilities, you can run a softmax on it.  
probabilities= torch.nn.functional.softmax(output[0], dim=0)  
print("预测的1000个分类联合概率分布数组的形状：",probabilities.shape)

预测的1000个分类联合概率分布数组的形状： torch.Size([1000])

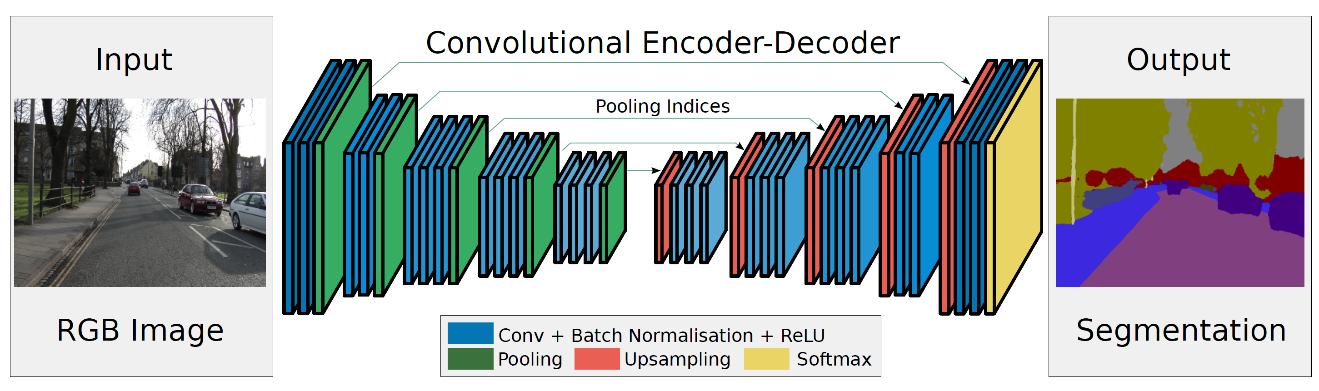
* 04 - 打印预测概率分布中最大的前几个对象，可以观察到预测的对象，desktop computer（及monitor、laptop、screen、computer keyboard）、notebook、desk都出现在该图像中。而printer和modem则没有，但是modem和插座的形状比较近似。

# Read the categories ImageNet数据集1000个类别文件可以从[imagenet\_classes.txt](https://raw.githubusercontent.com/pytorch/hub/master/imagenet\_classes.txt)处下载  
with open("./data/imagenet\_classes.txt", "r") as f:  
 categories=[s.strip() for s in f.readlines()]  
# 显示所预测图像，前几个最大概率对应的分类名 Show top categories per image  
top5\_prob,top5\_catid=torch.topk(probabilities, 10)  
for i in range(top5\_prob.size(0)):  
 print(categories[top5\_catid[i]], top5\_prob[i].item())

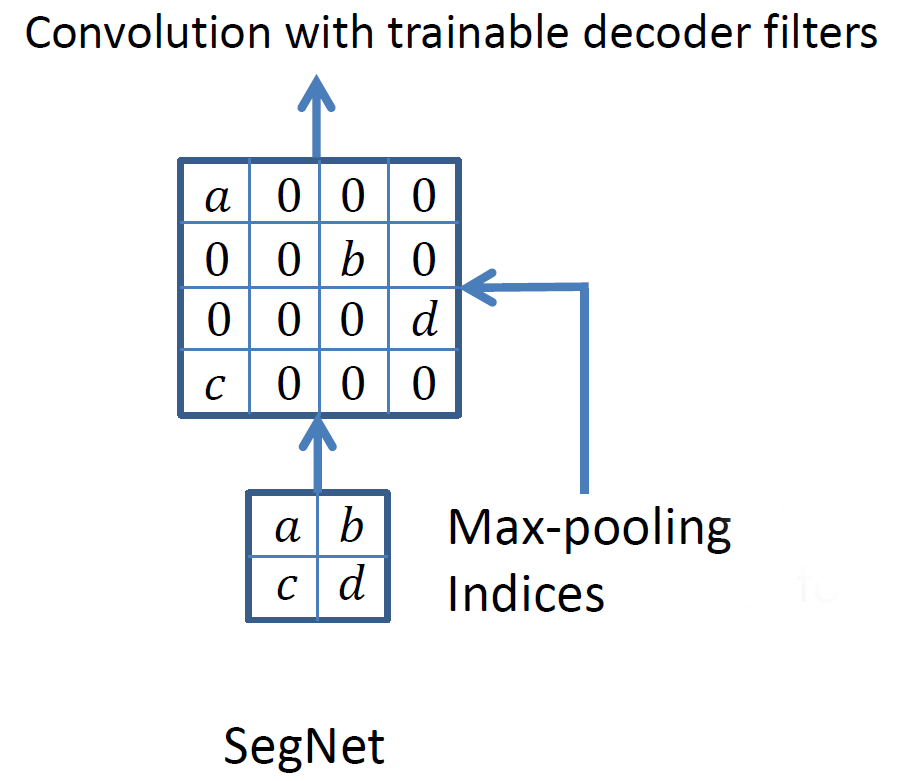
desktop computer 0.17022140324115753  
notebook 0.15083454549312592  
desk 0.14025698602199554  
monitor 0.1317901611328125  
laptop 0.10974671691656113  
mouse 0.10374213010072708  
screen 0.07918370515108109  
computer keyboard 0.033995289355516434  
printer 0.021042412146925926  
modem 0.008885478600859642

### 2.6.6.3 SegNet遥感影像语义分割/解译

SegNet于2016年提出，核心的概念是将网络划分为encoder编码器网络，decoder解码器网络，和一个像素级的分类层SoftMax。编码器网络结构与VGG16的13个特征提取卷积层结构相同。而解码器网络的结构与编码器网络刚好相逆，可以理解为反卷积的过程，每个编码器层都对应一个解码器层，将编码结果的低分辨率特征重新映射到输入时的分辨率，以便进行像素级分类，为每个像素生成类概率，输出不同分类的最大值，得到图像分割图。编码过程是池化层（nn.MaxPool2d(2, return\_indices=True)）下采样的过程，而解码过程是提取的特征值上采样(nn.MaxUnpool2d(2))的过程，如下图[2]。



下采样（pooling）就是池化层的作用，增加网络的深度。对于最大池化层，是提取区域内最大值作为输出，那么就可以得到最大值所在位置的索引。因此在上采样（upsampling）的过程中，对于池化下采样结果执行上采样时，已经丢失3个权重值，在将特征图放大2倍后，原来特征图的数据会根据下采样时获取的位置索引归位放入。对于池化最大值位置索引，PyTorch的nn.MaxPool2d()下return\_indices=True参数配置可以返回索引值。



代码迁移于[Deep learning for Earth Observation](https://github.com/nshaud/DeepNetsForEO)[3]。

* nn.MaxPool2d(2, return\_indices=True)

下述代码片段展示了编码器最大池化，及解码器应用索引值上采样的过程。

import torch.nn as nn  
pool=nn.MaxPool2d(2, stride=2, return\_indices=True)  
unpool = nn.MaxUnpool2d(2, stride=2)  
input = torch.tensor([[[[ 0., 1, 2, 3],  
 [ 4, 5, 6, 7],  
 [ 8, 9, 10, 11],  
 [12, 13, 14, 15]]]])  
  
output, indices=pool(input)  
print("最大池化索引：\n",indices)  
print("最大池化结果：\n",output)  
  
upsampling=unpool(output, indices)  
print("根据池化索引上采样结果：\n",upsampling)

最大池化索引：  
 tensor([[[[ 5, 7],  
 [13, 15]]]])  
最大池化结果：  
 tensor([[[[ 5., 7.],  
 [13., 15.]]]])  
根据池化索引上采样结果：  
 tensor([[[[ 0., 0., 0., 0.],  
 [ 0., 5., 0., 7.],  
 [ 0., 0., 0., 0.],  
 [ 0., 13., 0., 15.]]]])

* 01-调入所用的库

# imports and stuff  
import numpy as np  
from skimage import io  
from glob import glob  
from tqdm import tqdm\_notebook as tqdm  
from sklearn.metrics import confusion\_matrix  
import random  
import itertools  
import os  
  
# Matplotlib  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
# Torch imports  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.utils.data as data  
import torch.optim as optim  
import torch.optim.lr\_scheduler  
import torch.nn.init  
from torch.autograd import Variable

* 02 - ISPRS数据集，及数据查看，预处理，和小批量可迭代数据加载

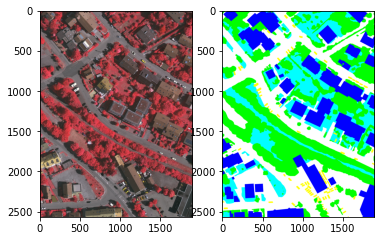
[ISPRS](https://www.isprs.org/data/)⑥是遥感图像数据集。对于遥感图像数据集，因为大量影像的开源和图像解译工具的存在（例如e-Cognition），可以用传统的解译工具建立影像的分割标签，从而建立数据集。因此目前有大量的遥感影像数据集可以使用，避免自行从新建立。下载的ISPRS数据集，包括三个地方分别为Potsdam、Toronto和Vaihingen。每个对应区域的所有数据放置于以地名命名的文件夹下，包含.tif格式（GTiff驱动），投影为CRS–>EPSG:32633 - WGS 84 / UTM zone ?N - Projected的原始影像，及影像标签，标签类别包括”roads”、 “buildings”、“low veg.”、“trees”、 “cars”和”clutter”等，可以分辨出主要的地物内容。如果该数据集的标签不能满足解译后使用上的需求，可以用其它满足要求的影像数据集替换，或者用传统工具自行解译部分影像用作训练数据集。数据有blue、green、red和NIR四个波段，不过波段已经合成为RGB、IRRG和RGBIR等形式，通常放置于各自单独的文件夹下。下述训练的数据使用的为IRRG合成的波段，即NIR+red+green。Vaihingen区域数据是由德国摄影测量和遥感协会（German Association of Photogrammetry and Remote Sensing，DGPF）用于测试数字航拍数据的子集。图像为8cm地面分辨率。

建立数据存放的字符串格式化模式，在后续调用class ISPRS\_dataset(torch.utils.data.Dataset)时使用。将训练集的数据对应到DATA\_FOLDE文件夹下，训练集的标签对应到LABEL\_FOLDER文件夹下，测试集的数据对应到ERODED\_FOLDER文件夹下。

# Parameters  
WINDOW\_SIZE = (256, 256) # Patch size  
STRIDE = 32 # Stride for testing  
IN\_CHANNELS = 3 # Number of input channels (e.g. RGB)  
FOLDER = r"G:/data/ISPRS/" # Replace with your "/path/to/the/ISPRS/dataset/folder/"  
BATCH\_SIZE = 10 # Number of samples in a mini-batch 10  
  
LABELS = ["roads", "buildings", "low veg.", "trees", "cars", "clutter"] # Label names  
N\_CLASSES = len(LABELS) # Number of classes  
WEIGHTS = torch.ones(N\_CLASSES) # Weights for class balancing  
CACHE = True # Store the dataset in-memory  
  
DATASET = 'Vaihingen'  
  
if DATASET == 'Potsdam':  
 MAIN\_FOLDER = FOLDER + 'Potsdam/'  
 # Uncomment the next line for IRRG data  
 # DATA\_FOLDER = MAIN\_FOLDER + '3\_Ortho\_IRRG/top\_potsdam\_{}\_IRRG.tif'  
 # For RGB data  
 DATA\_FOLDER = MAIN\_FOLDER + '2\_Ortho\_RGB/top\_potsdam\_{}\_RGB.tif'  
 LABEL\_FOLDER = MAIN\_FOLDER + '5\_Labels\_for\_participants/top\_potsdam\_{}\_label.tif'  
 ERODED\_FOLDER = MAIN\_FOLDER + '5\_Labels\_for\_participants\_no\_Boundary/top\_potsdam\_{}\_label\_noBoundary.tif'   
elif DATASET == 'Vaihingen':  
 MAIN\_FOLDER = FOLDER + 'Vaihingen/'  
 DATA\_FOLDER = MAIN\_FOLDER + 'top/top\_mosaic\_09cm\_area{}.tif'  
 LABEL\_FOLDER = MAIN\_FOLDER + 'gts\_for\_participants/top\_mosaic\_09cm\_area{}.tif'  
 ERODED\_FOLDER = MAIN\_FOLDER + 'gts\_eroded\_for\_participants/top\_mosaic\_09cm\_area{}\_noBoundary.tif'

数据查看，包括影像和对应标签。定义的函数convert\_to\_color(arr\_2d, palette=palette)和convert\_from\_color(arr\_3d, palette=invert\_palette)给定数值和对应RGB颜色值映射字典，实现数值和颜色之间的互相转换。

# ISPRS color palette  
# Let's define the standard ISPRS color palette  
palette = {0 : (255, 255, 255), # Impervious surfaces (white)  
 1 : (0, 0, 255), # Buildings (blue)  
 2 : (0, 255, 255), # Low vegetation (cyan)  
 3 : (0, 255, 0), # Trees (green)  
 4 : (255, 255, 0), # Cars (yellow)  
 5 : (255, 0, 0), # Clutter (red)  
 6 : (0, 0, 0)} # Undefined (black)  
  
invert\_palette = {v: k for k, v in palette.items()}  
  
def convert\_to\_color(arr\_2d, palette=palette):  
 """数值标签转换为RGB颜色标签 Numeric labels to RGB-color encoding """  
 arr\_3d = np.zeros((arr\_2d.shape[0], arr\_2d.shape[1], 3), dtype=np.uint8)  
  
 for c, i in palette.items():  
 m = arr\_2d == c  
 arr\_3d[m] = i  
  
 return arr\_3d  
  
def convert\_from\_color(arr\_3d, palette=invert\_palette):  
 """RGB颜色标签转换为数值标签（灰度图） RGB-color encoding to grayscale labels """  
 arr\_2d = np.zeros((arr\_3d.shape[0], arr\_3d.shape[1]), dtype=np.uint8)  
  
 for c, i in palette.items():  
 m = np.all(arr\_3d == np.array(c).reshape(1, 1, 3), axis=2)  
 arr\_2d[m] = i  
  
 return arr\_2d  
  
# We load one tile from the dataset and we display it  
img = io.imread(r'G:\data\ISPRS\Vaihingen\top/top\_mosaic\_09cm\_area11.tif')  
fig = plt.figure()  
fig.add\_subplot(121)  
plt.imshow(img)  
  
# We load the ground truth  
gt = io.imread(r'G:\data\ISPRS\Vaihingen\gts\_for\_participants/top\_mosaic\_09cm\_area11.tif')  
fig.add\_subplot(122)  
plt.imshow(gt)  
plt.show()  
  
# We also check that we can convert the ground truth into an array format  
array\_gt = convert\_from\_color(gt)  
print("Ground truth in numerical format has shape ({},{}) : \n".format(\*array\_gt.shape[:2]), array\_gt)



Ground truth in numerical format has shape (2566,1893) :   
 [[3 3 3 ... 3 3 3]  
 [3 3 3 ... 3 3 3]  
 [3 3 3 ... 3 3 3]  
 ...  
 [2 2 2 ... 1 1 1]  
 [2 2 2 ... 1 1 1]  
 [2 2 2 ... 1 1 1]]

定义小批量可迭代数据加载类，同时执行图像增广（image augmentation），由定义的data\_augmentation函数执行随机的翻转和镜像；并标准化数据到[0,1]。同时标识data\_augmentation函数有@classmethod装饰器，即标记该方法为类方法的装饰器。除了由实例对象调用外，可以直接由该类调用。如果作为父类，其子类也可以直接调用父类的类方法。

class C:  
 @classmethod  
 def f(cls,arg\_str):  
 print(cls,arg\_str)  
class C\_child(C):  
 pass  
print(C.f("类对象调用类方法..."))  
c=C()  
print(c.f("类实例调用类方法..."))  
print(C\_child.f("子类调用父类的类方法..."))

<class '\_\_main\_\_.C'> 类对象调用类方法...  
None  
<class '\_\_main\_\_.C'> 类实例调用类方法...  
None  
<class '\_\_main\_\_.C\_child'> 子类调用父类的类方法...  
None

def get\_random\_pos(img, window\_shape):  
 """给定窗口大小，随机提取部分图像 Extract of 2D random patch of shape window\_shape in the image """  
 w, h = window\_shape  
 W, H = img.shape[-2:]  
 x1 = random.randint(0, W - w - 1)  
 x2 = x1 + w  
 y1 = random.randint(0, H - h - 1)  
 y2 = y1 + h  
 return x1, x2, y1, y2  
  
# Dataset class  
class ISPRS\_dataset(torch.utils.data.Dataset):  
 def \_\_init\_\_(self, ids, data\_files=DATA\_FOLDER, label\_files=LABEL\_FOLDER,  
 cache=False, augmentation=True):  
 super(ISPRS\_dataset, self).\_\_init\_\_()  
   
 self.augmentation = augmentation  
 self.cache = cache  
   
 # List of files  
 self.data\_files = [DATA\_FOLDER.format(id) for id in ids]  
 self.label\_files = [LABEL\_FOLDER.format(id) for id in ids]  
  
 # Sanity check : raise an error if some files do not exist  
 for f in self.data\_files + self.label\_files:  
 if not os.path.isfile(f):  
 raise KeyError('{} is not a file !'.format(f))  
   
 # Initialize cache dicts  
 self.data\_cache\_ = {}  
 self.label\_cache\_ = {}  
   
   
 def \_\_len\_\_(self):  
 # Default epoch size is 10 000 samples  
 return 10000  
   
 @classmethod  
 def data\_augmentation(cls, \*arrays, flip=True, mirror=True):  
 will\_flip, will\_mirror = False, False  
 if flip and random.random() < 0.5:  
 will\_flip = True  
 if mirror and random.random() < 0.5:  
 will\_mirror = True  
   
 results = []  
 for array in arrays:  
 if will\_flip:  
 if len(array.shape) == 2:  
 array = array[::-1, :]  
 else:  
 array = array[:, ::-1, :]  
 if will\_mirror:  
 if len(array.shape) == 2:  
 array = array[:, ::-1]  
 else:  
 array = array[:, :, ::-1]  
 results.append(np.copy(array))  
   
 return tuple(results)  
   
 def \_\_getitem\_\_(self, i):  
 # Pick a random image  
 random\_idx = random.randint(0, len(self.data\_files) - 1)  
   
 # If the tile hasn't been loaded yet, put in cache  
 if random\_idx in self.data\_cache\_.keys():  
 data = self.data\_cache\_[random\_idx]  
 else:  
 # Data is normalized in [0, 1]  
 data = 1/255 \* np.asarray(io.imread(self.data\_files[random\_idx]).transpose((2,0,1)), dtype='float32')  
 if self.cache:  
 self.data\_cache\_[random\_idx] = data  
   
 if random\_idx in self.label\_cache\_.keys():  
 label = self.label\_cache\_[random\_idx]  
 else:   
 # Labels are converted from RGB to their numeric values  
 label = np.asarray(convert\_from\_color(io.imread(self.label\_files[random\_idx])), dtype='int64')  
 if self.cache:  
 self.label\_cache\_[random\_idx] = label  
  
 # Get a random patch  
 x1, x2, y1, y2 = get\_random\_pos(data, WINDOW\_SIZE)  
 data\_p = data[:, x1:x2,y1:y2]  
 label\_p = label[x1:x2,y1:y2]  
   
 # Data augmentation  
 data\_p, label\_p = self.data\_augmentation(data\_p, label\_p)  
  
 # Return the torch.Tensor values  
 return (torch.from\_numpy(data\_p),  
 torch.from\_numpy(label\_p))

加载数据。并切分数据集为训练和测试数据集。

# Load the datasets  
if DATASET == 'Potsdam':  
 all\_files = sorted(glob(LABEL\_FOLDER.replace('{}', '\*')))  
 all\_ids = ["".join(f.split('')[5:7]) for f in all\_files]  
elif DATASET == 'Vaihingen':   
 all\_files = sorted(glob(LABEL\_FOLDER.replace('{}', '\*')))  
 all\_ids = [f.split('area')[-1].split('.')[0] for f in all\_files]  
# Random tile numbers for train/test split  
train\_ids = random.sample(all\_ids, 2 \* len(all\_ids) // 3 + 1)  
test\_ids = list(set(all\_ids) - set(train\_ids))  
print("Tiles for training : ", train\_ids)  
print("Tiles for testing : ", test\_ids)  
  
train\_set = ISPRS\_dataset(train\_ids, cache=CACHE)  
train\_loader = torch.utils.data.DataLoader(train\_set,batch\_size=BATCH\_SIZE)

Tiles for training : ['17', '28', '32', '34', '26', '15', '7', '21', '30', '23', '3']  
Tiles for testing : ['13', '37', '5', '1', '11']

* 03 - 定义网络

class SegNet(nn.Module):  
 # SegNet network  
 @staticmethod  
 def weight\_init(m):  
 if isinstance(m, nn.Linear):  
 torch.nn.init.kaiming\_normal(m.weight.data)  
   
 def \_\_init\_\_(self, in\_channels=IN\_CHANNELS, out\_channels=N\_CLASSES):  
 super(SegNet, self).\_\_init\_\_()  
 self.pool = nn.MaxPool2d(2, return\_indices=True)  
 self.unpool = nn.MaxUnpool2d(2)  
   
 self.conv1\_1 = nn.Conv2d(in\_channels, 64, 3, padding=1)  
 self.conv1\_1\_bn = nn.BatchNorm2d(64)  
 self.conv1\_2 = nn.Conv2d(64, 64, 3, padding=1)  
 self.conv1\_2\_bn = nn.BatchNorm2d(64)  
   
 self.conv2\_1 = nn.Conv2d(64, 128, 3, padding=1)  
 self.conv2\_1\_bn = nn.BatchNorm2d(128)  
 self.conv2\_2 = nn.Conv2d(128, 128, 3, padding=1)  
 self.conv2\_2\_bn = nn.BatchNorm2d(128)  
   
 self.conv3\_1 = nn.Conv2d(128, 256, 3, padding=1)  
 self.conv3\_1\_bn = nn.BatchNorm2d(256)  
 self.conv3\_2 = nn.Conv2d(256, 256, 3, padding=1)  
 self.conv3\_2\_bn = nn.BatchNorm2d(256)  
 self.conv3\_3 = nn.Conv2d(256, 256, 3, padding=1)  
 self.conv3\_3\_bn = nn.BatchNorm2d(256)  
   
 self.conv4\_1 = nn.Conv2d(256, 512, 3, padding=1)  
 self.conv4\_1\_bn = nn.BatchNorm2d(512)  
 self.conv4\_2 = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv4\_2\_bn = nn.BatchNorm2d(512)  
 self.conv4\_3 = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv4\_3\_bn = nn.BatchNorm2d(512)  
   
 self.conv5\_1 = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv5\_1\_bn = nn.BatchNorm2d(512)  
 self.conv5\_2 = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv5\_2\_bn = nn.BatchNorm2d(512)  
 self.conv5\_3 = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv5\_3\_bn = nn.BatchNorm2d(512)  
   
 #-------------------------------------------------------------  
   
 self.conv5\_3\_D = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv5\_3\_D\_bn = nn.BatchNorm2d(512)  
 self.conv5\_2\_D = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv5\_2\_D\_bn = nn.BatchNorm2d(512)  
 self.conv5\_1\_D = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv5\_1\_D\_bn = nn.BatchNorm2d(512)  
   
 self.conv4\_3\_D = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv4\_3\_D\_bn = nn.BatchNorm2d(512)  
 self.conv4\_2\_D = nn.Conv2d(512, 512, 3, padding=1)  
 self.conv4\_2\_D\_bn = nn.BatchNorm2d(512)  
 self.conv4\_1\_D = nn.Conv2d(512, 256, 3, padding=1)  
 self.conv4\_1\_D\_bn = nn.BatchNorm2d(256)  
   
 self.conv3\_3\_D = nn.Conv2d(256, 256, 3, padding=1)  
 self.conv3\_3\_D\_bn = nn.BatchNorm2d(256)  
 self.conv3\_2\_D = nn.Conv2d(256, 256, 3, padding=1)  
 self.conv3\_2\_D\_bn = nn.BatchNorm2d(256)  
 self.conv3\_1\_D = nn.Conv2d(256, 128, 3, padding=1)  
 self.conv3\_1\_D\_bn = nn.BatchNorm2d(128)  
   
 self.conv2\_2\_D = nn.Conv2d(128, 128, 3, padding=1)  
 self.conv2\_2\_D\_bn = nn.BatchNorm2d(128)  
 self.conv2\_1\_D = nn.Conv2d(128, 64, 3, padding=1)  
 self.conv2\_1\_D\_bn = nn.BatchNorm2d(64)  
   
 self.conv1\_2\_D = nn.Conv2d(64, 64, 3, padding=1)  
 self.conv1\_2\_D\_bn = nn.BatchNorm2d(64)  
 self.conv1\_1\_D = nn.Conv2d(64, out\_channels, 3, padding=1)  
   
 self.apply(self.weight\_init)  
   
 def forward(self, x):  
 # Encoder block 1  
 x = self.conv1\_1\_bn(F.relu(self.conv1\_1(x)))  
 x = self.conv1\_2\_bn(F.relu(self.conv1\_2(x)))  
 x, mask1 = self.pool(x)  
   
 # Encoder block 2  
 x = self.conv2\_1\_bn(F.relu(self.conv2\_1(x)))  
 x = self.conv2\_2\_bn(F.relu(self.conv2\_2(x)))  
 x, mask2 = self.pool(x)  
   
 # Encoder block 3  
 x = self.conv3\_1\_bn(F.relu(self.conv3\_1(x)))  
 x = self.conv3\_2\_bn(F.relu(self.conv3\_2(x)))  
 x = self.conv3\_3\_bn(F.relu(self.conv3\_3(x)))  
 x, mask3 = self.pool(x)  
   
 # Encoder block 4  
 x = self.conv4\_1\_bn(F.relu(self.conv4\_1(x)))  
 x = self.conv4\_2\_bn(F.relu(self.conv4\_2(x)))  
 x = self.conv4\_3\_bn(F.relu(self.conv4\_3(x)))  
 x, mask4 = self.pool(x)  
   
 # Encoder block 5  
 x = self.conv5\_1\_bn(F.relu(self.conv5\_1(x)))  
 x = self.conv5\_2\_bn(F.relu(self.conv5\_2(x)))  
 x = self.conv5\_3\_bn(F.relu(self.conv5\_3(x)))  
 x, mask5 = self.pool(x)  
   
 #-------------------------------------------------------------  
   
 # Decoder block 5  
 x = self.unpool(x, mask5)  
 x = self.conv5\_3\_D\_bn(F.relu(self.conv5\_3\_D(x)))  
 x = self.conv5\_2\_D\_bn(F.relu(self.conv5\_2\_D(x)))  
 x = self.conv5\_1\_D\_bn(F.relu(self.conv5\_1\_D(x)))  
   
 # Decoder block 4  
 x = self.unpool(x, mask4)  
 x = self.conv4\_3\_D\_bn(F.relu(self.conv4\_3\_D(x)))  
 x = self.conv4\_2\_D\_bn(F.relu(self.conv4\_2\_D(x)))  
 x = self.conv4\_1\_D\_bn(F.relu(self.conv4\_1\_D(x)))  
   
 # Decoder block 3  
 x = self.unpool(x, mask3)  
 x = self.conv3\_3\_D\_bn(F.relu(self.conv3\_3\_D(x)))  
 x = self.conv3\_2\_D\_bn(F.relu(self.conv3\_2\_D(x)))  
 x = self.conv3\_1\_D\_bn(F.relu(self.conv3\_1\_D(x)))  
   
 # Decoder block 2  
 x = self.unpool(x, mask2)  
 x = self.conv2\_2\_D\_bn(F.relu(self.conv2\_2\_D(x)))  
 x = self.conv2\_1\_D\_bn(F.relu(self.conv2\_1\_D(x)))  
   
 # Decoder block 1  
 x = self.unpool(x, mask1)  
 x = self.conv1\_2\_D\_bn(F.relu(self.conv1\_2\_D(x)))  
 x = F.log\_softmax(self.conv1\_1\_D(x))  
 return x

从地址 <https://download.pytorch.org/models/vgg16_bn-6c64b313.pth> 下载VGG16网络模型预训练参数。因为下载的预训练参数对应的层名与上述模型定义的不同，因此需要一一对位，将权值映射到新的层名上来。然后应用net.state\_dict().update(mapped\_weights)方法更新权值。

# instantiate the network  
net = SegNet()

import os  
try:  
 from urllib.request import URLopener  
except ImportError:  
 from urllib import URLopener  
  
# Download VGG-16 weights from PyTorch  
vgg\_url = 'https://download.pytorch.org/models/vgg16\_bn-6c64b313.pth'  
if not os.path.isfile(r'G:\data\model\vgg16\_bn-6c64b313.pth'):  
 weights = URLopener().retrieve(vgg\_url, r'G:\data\model\vgg16\_bn-6c64b313.pth')  
  
vgg16\_weights = torch.load(r'G:\data\model\vgg16\_bn-6c64b313.pth')  
mapped\_weights = {}  
for k\_vgg, k\_segnet in zip(vgg16\_weights.keys(), net.state\_dict().keys()):  
 if "features" in k\_vgg:  
 mapped\_weights[k\_segnet] = vgg16\_weights[k\_vgg]  
 print("Mapping {} to {}".format(k\_vgg, k\_segnet))  
   
try:  
 net.state\_dict().update(mapped\_weights)  
 print("\_"\*50)  
 print("Loaded VGG-16 weights in SegNet !")  
except:  
 print("Ignore missing keys")  
 pass

Mapping features.0.weight to conv1\_1.weight  
Mapping features.0.bias to conv1\_1.bias  
Mapping features.1.weight to conv1\_1\_bn.weight  
Mapping features.1.bias to conv1\_1\_bn.bias  
Mapping features.1.running\_mean to conv1\_1\_bn.running\_mean  
Mapping features.1.running\_var to conv1\_1\_bn.running\_var  
Mapping features.3.weight to conv1\_1\_bn.num\_batches\_tracked  
Mapping features.3.bias to conv1\_2.weight  
Mapping features.4.weight to conv1\_2.bias  
Mapping features.4.bias to conv1\_2\_bn.weight  
Mapping features.4.running\_mean to conv1\_2\_bn.bias  
Mapping features.4.running\_var to conv1\_2\_bn.running\_mean  
Mapping features.7.weight to conv1\_2\_bn.running\_var  
Mapping features.7.bias to conv1\_2\_bn.num\_batches\_tracked  
Mapping features.8.weight to conv2\_1.weight  
Mapping features.8.bias to conv2\_1.bias  
Mapping features.8.running\_mean to conv2\_1\_bn.weight  
Mapping features.8.running\_var to conv2\_1\_bn.bias  
Mapping features.10.weight to conv2\_1\_bn.running\_mean  
Mapping features.10.bias to conv2\_1\_bn.running\_var  
Mapping features.11.weight to conv2\_1\_bn.num\_batches\_tracked  
Mapping features.11.bias to conv2\_2.weight  
Mapping features.11.running\_mean to conv2\_2.bias  
Mapping features.11.running\_var to conv2\_2\_bn.weight  
Mapping features.14.weight to conv2\_2\_bn.bias  
Mapping features.14.bias to conv2\_2\_bn.running\_mean  
Mapping features.15.weight to conv2\_2\_bn.running\_var  
Mapping features.15.bias to conv2\_2\_bn.num\_batches\_tracked  
Mapping features.15.running\_mean to conv3\_1.weight  
Mapping features.15.running\_var to conv3\_1.bias  
Mapping features.17.weight to conv3\_1\_bn.weight  
Mapping features.17.bias to conv3\_1\_bn.bias  
Mapping features.18.weight to conv3\_1\_bn.running\_mean  
Mapping features.18.bias to conv3\_1\_bn.running\_var  
Mapping features.18.running\_mean to conv3\_1\_bn.num\_batches\_tracked  
Mapping features.18.running\_var to conv3\_2.weight  
Mapping features.20.weight to conv3\_2.bias  
Mapping features.20.bias to conv3\_2\_bn.weight  
Mapping features.21.weight to conv3\_2\_bn.bias  
Mapping features.21.bias to conv3\_2\_bn.running\_mean  
Mapping features.21.running\_mean to conv3\_2\_bn.running\_var  
Mapping features.21.running\_var to conv3\_2\_bn.num\_batches\_tracked  
Mapping features.24.weight to conv3\_3.weight  
Mapping features.24.bias to conv3\_3.bias  
Mapping features.25.weight to conv3\_3\_bn.weight  
Mapping features.25.bias to conv3\_3\_bn.bias  
Mapping features.25.running\_mean to conv3\_3\_bn.running\_mean  
Mapping features.25.running\_var to conv3\_3\_bn.running\_var  
Mapping features.27.weight to conv3\_3\_bn.num\_batches\_tracked  
Mapping features.27.bias to conv4\_1.weight  
Mapping features.28.weight to conv4\_1.bias  
Mapping features.28.bias to conv4\_1\_bn.weight  
Mapping features.28.running\_mean to conv4\_1\_bn.bias  
Mapping features.28.running\_var to conv4\_1\_bn.running\_mean  
Mapping features.30.weight to conv4\_1\_bn.running\_var  
Mapping features.30.bias to conv4\_1\_bn.num\_batches\_tracked  
Mapping features.31.weight to conv4\_2.weight  
Mapping features.31.bias to conv4\_2.bias  
Mapping features.31.running\_mean to conv4\_2\_bn.weight  
Mapping features.31.running\_var to conv4\_2\_bn.bias  
Mapping features.34.weight to conv4\_2\_bn.running\_mean  
Mapping features.34.bias to conv4\_2\_bn.running\_var  
Mapping features.35.weight to conv4\_2\_bn.num\_batches\_tracked  
Mapping features.35.bias to conv4\_3.weight  
Mapping features.35.running\_mean to conv4\_3.bias  
Mapping features.35.running\_var to conv4\_3\_bn.weight  
Mapping features.37.weight to conv4\_3\_bn.bias  
Mapping features.37.bias to conv4\_3\_bn.running\_mean  
Mapping features.38.weight to conv4\_3\_bn.running\_var  
Mapping features.38.bias to conv4\_3\_bn.num\_batches\_tracked  
Mapping features.38.running\_mean to conv5\_1.weight  
Mapping features.38.running\_var to conv5\_1.bias  
Mapping features.40.weight to conv5\_1\_bn.weight  
Mapping features.40.bias to conv5\_1\_bn.bias  
Mapping features.41.weight to conv5\_1\_bn.running\_mean  
Mapping features.41.bias to conv5\_1\_bn.running\_var  
Mapping features.41.running\_mean to conv5\_1\_bn.num\_batches\_tracked  
Mapping features.41.running\_var to conv5\_2.weight  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Loaded VGG-16 weights in SegNet !

if torch.cuda.is\_available():  
 net.to('cuda')

* 04 - 定义训练模型相关函数、损失函数、预测精度及相关度量值（全局精度、F1分数和kappa系数）等内容。

def CrossEntropy2d(input, target, weight=None, size\_average=True):  
 """定义损失函数——2D版交叉熵损失 2D version of the cross entropy loss """  
 dim = input.dim()  
 if dim == 2:  
 return F.cross\_entropy(input, target, weight, size\_average)  
 elif dim == 4:  
 output = input.view(input.size(0),input.size(1), -1)  
 output = torch.transpose(output,1,2).contiguous()  
 output = output.view(-1,output.size(2))  
 target = target.view(-1)  
 return F.cross\_entropy(output, target,weight, size\_average)  
 else:  
 raise ValueError('Expected 2 or 4 dimensions (got {})'.format(dim))  
  
def accuracy(input, target):  
 '''定义预测精度'''  
 return 100 \* float(np.count\_nonzero(input == target)) / target.size  
  
def sliding\_window(top, step=10, window\_size=(20,20)):  
 """给定步幅，窗口形状，滑动过整幅图像，迭代计算窗口所在图像x,y位置值，返回每一切分图像(patch)的x,y坐标值和高宽大小，即yield返回值。参数step可以控制切分窗口叠合的程度 Slide a window\_shape window across the image with a stride of step """  
 for x in range(0, top.shape[0], step):  
 if x + window\_size[0] > top.shape[0]:  
 x = top.shape[0] - window\_size[0]  
 for y in range(0, top.shape[1], step):  
 if y + window\_size[1] > top.shape[1]:  
 y = top.shape[1] - window\_size[1]  
 yield x, y, window\_size[0], window\_size[1]  
   
def count\_sliding\_window(top, step=10, window\_size=(20,20)):  
 """计算图像滑动给定窗口大小的数量 Count the number of windows in an image """  
 c = 0  
 for x in range(0, top.shape[0], step):  
 if x + window\_size[0] > top.shape[0]:  
 x = top.shape[0] - window\_size[0]  
 for y in range(0, top.shape[1], step):  
 if y + window\_size[1] > top.shape[1]:  
 y = top.shape[1] - window\_size[1]  
 c += 1  
 return c  
  
def grouper(n, iterable):  
 """ Browse an iterator by chunk of n elements """  
 it = iter(iterable)  
 while True:  
 chunk = tuple(itertools.islice(it, n))  
 if not chunk:  
 return  
 yield chunk  
  
def metrics(predictions, gts, label\_values=LABELS):  
 '''预测值度量'''  
 cm = confusion\_matrix(  
 gts,  
 predictions,  
 labels=range(len(label\_values)))   
   
 print("Confusion matrix :")  
 print(cm)  
   
 print("---")  
   
 # 全局精度 Compute global accuracy  
 total = sum(sum(cm))  
 accuracy = sum([cm[x][x] for x in range(len(cm))])  
 accuracy \*= 100 / float(total)  
 print("{} pixels processed".format(total))  
 print("Total accuracy : {}%".format(accuracy))  
   
 print("---")  
   
 # F1分数 Compute F1 score  
 F1Score = np.zeros(len(label\_values))  
 for i in range(len(label\_values)):  
 try:  
 F1Score[i] = 2. \* cm[i,i] / (np.sum(cm[i,:]) + np.sum(cm[:,i]))  
 except:  
 # Ignore exception if there is no element in class i for test set  
 pass  
 print("F1Score :")  
 for l\_id, score in enumerate(F1Score):  
 print("{}: {}".format(label\_values[l\_id], score))  
  
 print("---")  
   
 # 计算kappa系数 Compute kappa coefficient  
 total = np.sum(cm)  
 pa = np.trace(cm) / float(total)  
 pe = np.sum(np.sum(cm, axis=0) \* np.sum(cm, axis=1)) / float(total\*total)  
 kappa = (pa - pe) / (1 - pe);  
 print("Kappa: " + str(kappa))  
 return accuracy

使用标准的随机梯度下降算法优化网络的权值。如果调入了预先训练的VGG16模型参数，则可调整学习率。即encoder编码部分（VGG16卷积，特征提取部分）的训练速度为decoder解码器的一半。

base\_lr = 0.01  
params\_dict = dict(net.named\_parameters())  
params = []  
for key, value in params\_dict.items():  
 if '\_D' in key:  
 # Decoder weights are trained at the nominal learning rate  
 params += [{'params':[value],'lr': base\_lr}]  
 else:  
 # Encoder weights are trained at lr / 2 (we have VGG-16 weights as initialization)  
 params += [{'params':[value],'lr': base\_lr / 2}]  
  
optimizer = optim.SGD(net.parameters(), lr=base\_lr, momentum=0.9, weight\_decay=0.0005)  
# We define the scheduler  
scheduler = optim.lr\_scheduler.MultiStepLR(optimizer, [25, 35, 45], gamma=0.1)

* 05 - 定义测试函数，显示RGB影像，及对应的真实值和预测值图像。计算metrics函数定义的相关预测度量值。

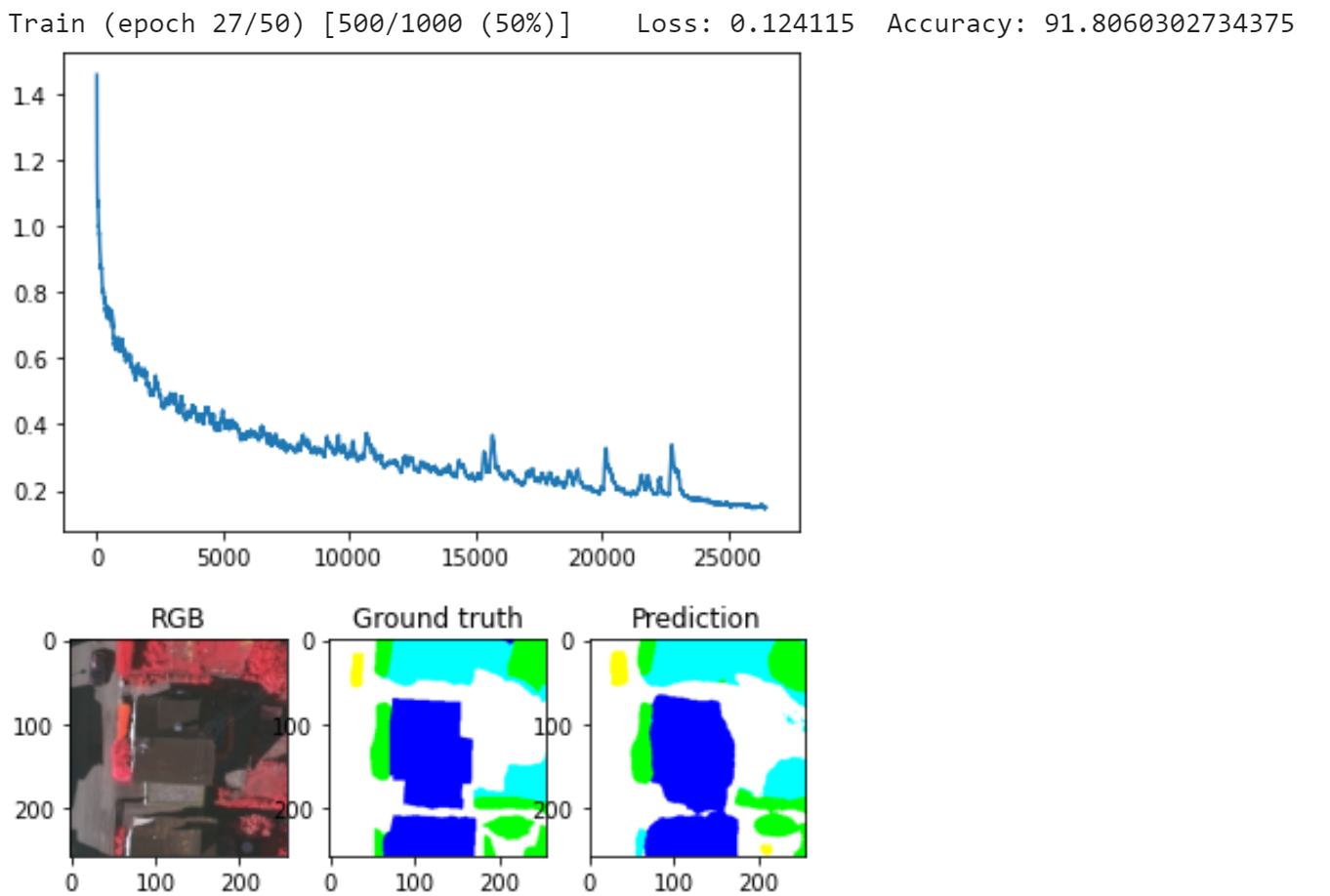
def test(net, test\_ids, all=False, stride=WINDOW\_SIZE[0], batch\_size=BATCH\_SIZE, window\_size=WINDOW\_SIZE):  
 # Use the network on the test set  
 test\_images = (1 / 255 \* np.asarray(io.imread(DATA\_FOLDER.format(id)), dtype='float32') for id in test\_ids)  
 test\_labels = (np.asarray(io.imread(LABEL\_FOLDER.format(id)), dtype='uint8') for id in test\_ids)  
 eroded\_labels = (convert\_from\_color(io.imread(ERODED\_FOLDER.format(id))) for id in test\_ids)  
 all\_preds = []  
 all\_gts = []  
   
 # Switch the network to inference mode  
 net.eval()  
  
 for img, gt, gt\_e in tqdm(zip(test\_images, test\_labels, eroded\_labels), total=len(test\_ids), leave=False):  
 pred = np.zeros(img.shape[:2] + (N\_CLASSES,))  
  
 total = count\_sliding\_window(img, step=stride, window\_size=window\_size) // batch\_size  
 for i, coords in enumerate(tqdm(grouper(batch\_size, sliding\_window(img, step=stride, window\_size=window\_size)), total=total, leave=False)):  
 # Display in progress results  
 if i > 0 and total > 10 and i % int(10 \* total / 100) == 0:  
 \_pred = np.argmax(pred, axis=-1)  
 fig = plt.figure()  
 fig.add\_subplot(1,3,1)  
 plt.imshow(np.asarray(255 \* img, dtype='uint8'))  
 fig.add\_subplot(1,3,2)  
 plt.imshow(convert\_to\_color(\_pred))  
 fig.add\_subplot(1,3,3)  
 plt.imshow(gt)  
 clear\_output()  
 plt.show()  
   
 # Build the tensor  
 image\_patches = [np.copy(img[x:x+w, y:y+h]).transpose((2,0,1)) for x,y,w,h in coords]  
 image\_patches = np.asarray(image\_patches)  
 image\_patches = Variable(torch.from\_numpy(image\_patches).cuda(), volatile=True)  
   
 # Do the inference  
 outs = net(image\_patches)  
 outs = outs.data.cpu().numpy()  
   
 # Fill in the results array  
 for out, (x, y, w, h) in zip(outs, coords):  
 out = out.transpose((1,2,0))  
 pred[x:x+w, y:y+h] += out  
 del(outs)  
   
 pred = np.argmax(pred, axis=-1)  
  
 # Display the result  
 clear\_output()  
 fig = plt.figure()  
 fig.add\_subplot(1,3,1)  
 plt.imshow(np.asarray(255 \* img, dtype='uint8'))  
 fig.add\_subplot(1,3,2)  
 plt.imshow(convert\_to\_color(pred))  
 fig.add\_subplot(1,3,3)  
 plt.imshow(gt)  
 plt.show()  
  
 all\_preds.append(pred)  
 all\_gts.append(gt\_e)  
  
 clear\_output()  
 # Compute some metrics  
 metrics(pred.ravel(), gt\_e.ravel())  
 accuracy = metrics(np.concatenate([p.ravel() for p in all\_preds]), np.concatenate([p.ravel() for p in all\_gts]).ravel())  
 if all:  
 return accuracy, all\_preds, all\_gts  
 else:  
 return accuracy

* 06 - 定义训练函数。输出损失曲线，显示RGB影像，及对应的真实值和预测值图像。打印损失值和精度值，观察模型训练情况。同时指定文件夹，保存模型参数。

from IPython.display import clear\_output  
  
def train(net, optimizer, epochs, scheduler=None, weights=WEIGHTS, save\_epoch = 5):   
 losses = np.zeros(1000000)  
 mean\_losses = np.zeros(100000000)  
 weights = weights.cuda()  
  
 criterion = nn.NLLLoss2d(weight=weights)  
 iter\_ = 0  
   
 for e in range(1, epochs + 1):  
 if scheduler is not None:  
 scheduler.step()  
 net.train()  
 for batch\_idx, (data, target) in enumerate(train\_loader):  
 data, target = Variable(data.cuda()), Variable(target.cuda())  
 optimizer.zero\_grad()  
 output = net(data)  
 loss = CrossEntropy2d(output, target, weight=weights)  
 loss.backward()  
 optimizer.step()  
   
 #print("\_"\*50)  
 #print(iter\_)  
 #print(loss.data.item())  
 losses[iter\_] = loss.data.item()#losses[iter\_] = loss.data[0]  
 mean\_losses[iter\_] = np.mean(losses[max(0,iter\_-100):iter\_])  
   
 if iter\_ % 100 == 0:  
 clear\_output()  
 rgb = np.asarray(255 \* np.transpose(data.data.cpu().numpy()[0],(1,2,0)), dtype='uint8')  
 pred = np.argmax(output.data.cpu().numpy()[0], axis=0)  
 gt = target.data.cpu().numpy()[0]  
 print('Train (epoch {}/{}) [{}/{} ({:.0f}%)]\tLoss: {:.6f}\tAccuracy: {}'.format(  
 e, epochs, batch\_idx, len(train\_loader),  
 100. \* batch\_idx / len(train\_loader), loss.data.item(), accuracy(pred, gt))) #100. \* batch\_idx / len(train\_loader), loss.data[0], accuracy(pred, gt)))  
 plt.plot(mean\_losses[:iter\_]) and plt.show()  
 fig = plt.figure()  
 fig.add\_subplot(131)  
 plt.imshow(rgb)  
 plt.title('RGB')  
 fig.add\_subplot(132)  
 plt.imshow(convert\_to\_color(gt))  
 plt.title('Ground truth')  
 fig.add\_subplot(133)  
 plt.title('Prediction')  
 plt.imshow(convert\_to\_color(pred))  
 plt.show()  
 iter\_ += 1  
   
 del(data, target, loss)  
 if e % save\_epoch == 0:  
 # We validate with the largest possible stride for faster computing  
 acc = test(net, test\_ids, all=False, stride=min(WINDOW\_SIZE))  
 torch.save(net.state\_dict(), './model/segnet256\_epoch{}\_{}'.format(e, acc))  
 torch.save(net.state\_dict(), './model/segnet\_final')

* 07 - 训练模型

train(net, optimizer, 10, scheduler) #50



Confusion matrix :  
[[1353141 32015 65375 19629 4922 1733]  
 [ 34820 796320 9073 3273 670 3]  
 [ 45405 13111 541334 98071 62 0]  
 [ 9127 2476 85643 1235041 45 0]  
 [ 9981 742 776 581 32381 214]  
 [ 0 0 0 0 0 0]]  
---  
4395964 pixels processed  
Total accuracy : 90.0420704082199%  
---  
F1Score :  
roads: 0.9238699220186195  
buildings: 0.9430473175696921  
low veg.: 0.7732326608502883  
trees: 0.9186125171862234  
cars: 0.7825750709926893  
clutter: 0.0  
---  
Kappa: 0.8641704967438216  
Confusion matrix :  
[[5943500 251376 192249 59281 23200 3986]  
 [ 257402 6629996 54712 9507 3639 72]  
 [ 370167 111835 4206096 564403 529 3257]  
 [ 59915 12051 533183 4003598 151 0]  
 [ 45805 6880 1743 828 110866 883]  
 [ 0 0 0 0 0 0]]  
---  
23461110 pixels processed  
Total accuracy : 89.05825853934446%  
---  
F1Score :  
roads: 0.9039281827651989  
buildings: 0.9493484358580146  
low veg.: 0.8211607074003321  
trees: 0.8659690705092675  
cars: 0.7260617570974819  
clutter: 0.0  
---  
Kappa: 0.8533983408961401

* 08 - 加载保存的SegNet模型参数，应用测试数据集测试模型。通过配置stride参数，设置图像被切分为多个小块之间的重叠程度。重叠程度由stride参数和WINDOW\_SIZE=(256, 256)参数（即patch大小）确定。

注意，在应用所训练的模型解译新的图像时，新图像的形式应该与训练数据的形式保持一致或近似，这样才能够保证正确的预测结果。例如图像的高空分辨率，及波段合成信息等能够基本相同。

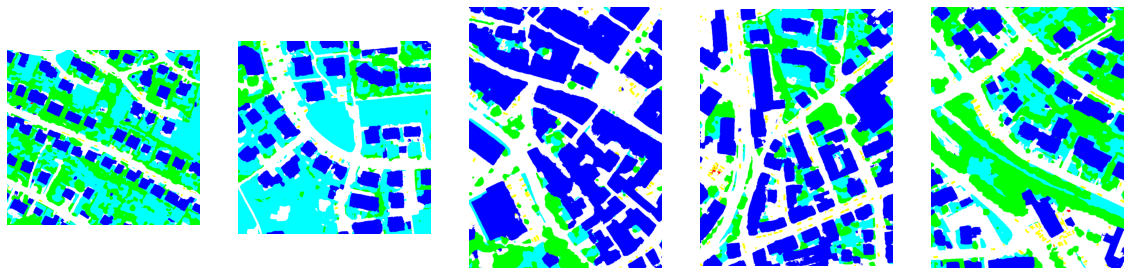
net.load\_state\_dict(torch.load('./model/segnet\_final'))  
if torch.cuda.is\_available():  
 net.to('cuda')

\_, all\_preds, all\_gts = test(net, test\_ids, all=True, stride=32)

Confusion matrix :  
[[1370081 25031 57259 19254 4022 1168]  
 [ 29015 804242 7865 2584 453 0]  
 [ 47582 10867 542282 97228 24 0]  
 [ 7684 2311 72972 1249350 15 0]  
 [ 10697 260 726 528 32321 143]  
 [ 0 0 0 0 0 0]]  
---  
4395964 pixels processed  
Total accuracy : 90.95333810740944%  
---  
F1Score :  
roads: 0.9314341810696175  
buildings: 0.9535316888675476  
low veg.: 0.7864362436887593  
trees: 0.9250072928497495  
cars: 0.7930560667402773  
clutter: 0.0  
---  
Kappa: 0.8764381914078332  
Confusion matrix :  
[[6007256 209282 180520 56386 18409 1739]  
 [ 201880 6694014 48209 8352 2772 101]  
 [ 366170 98613 4254461 535807 385 851]  
 [ 53371 10865 501938 4042672 52 0]  
 [ 49247 4373 1509 899 110263 714]  
 [ 0 0 0 0 0 0]]  
---  
23461110 pixels processed  
Total accuracy : 89.97300639228067%  
---  
F1Score :  
roads: 0.9135457843795346  
buildings: 0.9581715479898872  
low veg.: 0.8307122067878274  
trees: 0.8738065240147697  
cars: 0.737826462263204  
clutter: 0.0  
---  
Kappa: 0.8656335585255769

* 09 - 显示与保存预测的图像分割/解译

import matplotlib.pyplot as plt  
from tqdm import tqdm  
plt.figure(figsize=(20,5))  
  
i=0  
for p, id\_ in tqdm(zip(all\_preds,test\_ids),total=len(all\_preds),leave=False):  
 img = convert\_to\_color(p)  
 plt.subplot(1,len(all\_preds),i+1)  
 plt.imshow(img)  
 plt.axis('off')  
   
 io.imsave('./results/segment\_pred/inference\_tile\_{}.png'.format(id\_), img)  
 i+=1  
plt.show()



可以尝试在DUC(Dense Upsampling Convolution)图像分割部分，用SegNet模型替换DUC实现。

注释（Notes）：

① ImageNet 数据集，是按 WordNet 层次结构（目前只有名词）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。 这些数据免费提供给研究人员用于非商业用途，（<https://image-net.org/>）。

② WordNet，是一个大型的英语词汇数据库。 名词、动词、形容词和副词被分组到认知同义词集（同义词集），每个都表达一个不同的概念。 同义词集通过概念——语义和词汇关系相互关联。 WordNet 可以免费公开下载。 WordNet 的结构使其成为计算语言学和自然语言处理的有用工具，（<https://wordnet.princeton.edu/>）。

③ ImageNet数据集1000个类别文件——imagenet\_classes.txt，（<https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt>）。

④ torchvision.models，预定义深度学习模型，（<https://pytorch.org/vision/0.8/models.html>）。

⑤ VGG-NETS，2014年 Imagenet ILSVRC 挑战赛获奖的 ConvNets，（<https://pytorch.org/hub/pytorch_vision_vgg/>）。

⑥ ISPRS数据集，遥感影像数据集，（<https://www.isprs.org/data>）。

参考文献（References）:

[1] Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. (2014). <https://arxiv.org/abs/1409.1556>

[2] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, Senior Member, IEEE. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation[J].arXiv:1511.00561v3 [cs.CV] 10 Oct 2016

[3] Deep learning for Earth Observation, <https://github.com/nshaud/DeepNetsForEO>.