# 3.4-C 模式生成：cGAN——模式转化和未知区域地理信息修复

## 3.4.1 条件对抗网络（Conditional Adversarial Networks，cGAN）与地理空间模式潜在应用方式

在地理空间数据分析中，最重要的信息来源之一是遥感（航拍）影像数据，且包含有多个波段，根据不同波段的特性分析相关内容，这在Landsat遥感影像、Sentinel-2和NAIP航拍影像等章节中均有阐释。遥感影像（RGB波段）合成的自然真彩色图像应用为卫星地图被广泛使用，但是为了进行城市空间模式的分析，需要从影像地图中提取信息，例如反演地表温度、计算NDVI（归一化植被指数）、NDWI（归一化水体指数）和NDBI（归一化建筑指数）等各类指数等，其中影像解译为土地覆盖类型（land cover，LC）是研究城市空间模式的基础，能够分析地物分布结构等特点，例如*标记距离*一章对 LC 的模式搜索、监测和分割等。根据波段特征，应用影像解译等技术，例如[eCognition](https://geospatial.trimble.com/what-is-ecognition)①面向对象影像分割分类方法等已经生产大量覆盖覆盖全球和多个连续时间点的 LC 数据，且仍然在进行中。因为具有了大量已经分类的 LC 等地理信息数据，由条件对抗网络（cGAN）可以构建遥感影像到 LC 或 LC 到遥感影像的转化模型，由此会为模式的分析带来潜在新的研究途径，这包括：

1. 将网络（例如使用 U-Net）的编码器（encoder）结果用于模式标记特征，分析基于已知 LC 分类的样方模式特征和分布及样方间模式结构的比较等；
2. 用网络解码器（decoder）的不同层（逆卷积层）分析对应尺度空间深度分布特征；
3. 构建具有关联的各类对应地理空间数据间的映射网络模型，例如生境质量、碳存储和固持、小气候环境数据等，作1、2条的分析；
4. 设计层面，将绘制的 LC 实时生成影像图片，协助规划设计[1]；
5. 设计层面，根据已知地理空间数据预测未知区域信息，用于规划设计参考，并分析被修复区域潜在模式的数据分布;
6. 设计层面，迁移特定类型的城市空间模式于给定区域，观察模式变化形态。

对 cGAN 的探讨主要基于Isola, P., Zhu, J.-Y., Zhou, T. 等人[2] 对 cGAN 大量已有研究实现的一个通用框架，该框架主要包括Pix2pix GAN和CycleGAN两个图像（输入）到图像（输出）的网络模型（而 DCGAN、WGAN和StyleGAN等输入为噪声（隐藏）向量），其中Pix2pix GAN 为成对的图像互相匹配，例如图像或遥感影像的语义分割将影像和分类图块对应起来，或者将轮廓线与对象对应起来等；CycleGAN 成对图像间没有匹配关系，可以把一个图像的特征迁移至另一个图像，为域迁移（Domain Adaptation）。 Isola, P. 等作者对应论文开发了 [CycleGAN and pix2pix](https://phillipi.github.io/pix2pix/)②工具，包括Torch、Tensorflow和PyTorch等版本，这里使用[PyTorch版本](https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix)[3]，并将其迁移至USDA库进行使用，位于migrated\_project.pix2pix子包。

在迁移时，需要注意模块相对路径调入的方式，当在本地调试时，调入模块方式通常为from options import cfg\_train（位于同一文件夹下）；但是对于Python包一般要在前面加入点（同样位于同一文件夹下），为from .options import cfg\_train，因此为了在本地调试，一般可以用下述方式调入，同时满足本地调试调用和作为安装包调用。

if \_\_package\_\_:   
 from .options import cfg\_train   
 from .data import create\_dataset  
 from .models import create\_model  
 from .util.\_visualizer import Visualizer  
   
else:  
 from options import cfg\_train   
 from data import create\_dataset  
 from models import create\_model  
 from util.\_visualizer import Visualizer

## 3.4.1.1 cGAN 方法

GANs 生成网络是从随机噪声向量输入到图像输出的映射，  
G*:*z*→*y*xzy*G*:{*x*,*z*}→*y*GD*

*L*cGAN*​(*G*,*D*)=E*x*,*y*​[log*D*(*x*,*y*)]+E*x*,*z*​[log(1−*D*(*x*,*G*(*x*,*z*))]* GD

L1L2距离混合在一

起使用会优化网络。

对于损失函数，也称之为最小化绝对误差（Least Abosulote Error，LAE），为最小化真实值和预测值之间差值绝对值的和，公式为：。实际为平均绝对误差（Mean absolute error，MAE）。使用损失函数即要求 ；损失函数，也称为最小化平方误差（Least Square Error，LSE），为最小化真实值和预测值之间差值平方和，公式为：。：实际为均方误差（mean-square error，MSE）。使用损失函数即要求 。

在 cGANs 中，使用距离，公式表述为：，因此更新 cGANs 的损失函数为：。

如果没有噪声向量，网络仍然可以学习从到的映射，但会产生确定性输出，因此无法匹配除狄拉克函数（delta function）以外的任何分布，因此在网络中应用dropout的形式提供噪声。

## 3.4.1.2 网络体系结构

使用迁移到USDA库中的 cGAN 通用框架，以pix2pix网络为例，配置参数调入网络。训练数据集使用了cGAN 通用框架作者提供的[maps](http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/)③，为遥感影像和地图对应的数据集，对应配置参数p2p.opt.basic.dataroot，默认文件夹名为train；且加载了预训练模型[map2sat或sat2map](http://efrosgans.eecs.berkeley.edu/pix2pix/models-pytorch/)④，对应参数p2p.opt.basic.checkpoints\_dir，默认预训练模型文件名为latest\_net\_G.pth（）和latest\_net\_D.pth()。数据集的表现形式使用了aligned方式，将遥感影像和对应的地图图像并在一个文件，即左右拼接了对应的两幅图像，有参数p2p.opt.dataset.dataset\_mode配置。如果由遥感影像生成地图，则配置p2p.opt.dataset.direction为A2B，反之，用地图生成对应的遥感影像配置为B2A。

# IPython extension to reload modules before executing user code.  
%load\_ext autoreload   
# Reload all modules (except those excluded by %aimport) every time before executing the Python code typed.  
%autoreload 2   
  
from usda.migrated\_project.pix2pix import train  
from usda.migrated\_project.pix2pix import test  
from usda.migrated\_project.stylegan import adjust\_dynamic\_range  
  
import matplotlib.pyplot as plt  
import numpy as np  
import torch  
from torchshape import tensorshape  
from fastai.vision.gan import basic\_critic,partial

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

配置基本参数，并用p2p.create\_dataset()创建数据加载器。

p2p=train.Pix2pix\_train()  
p2p.opt.basic.dataroot=r'I:\data\pix2pix\_dataset\maps'  
p2p.opt.dataset.dataset\_mode='aligned'   
p2p.opt.dataset.direction='BtoA'  
p2p.opt.basic.isTrain=True  
p2p.opt.model.model='pix2pix'  
  
p2p.create\_dataset()

dataset [AlignedDataset] was created  
The number of training images = 1096

用p2p.create\_model()方法构建模型。

p2p.opt.basic.checkpoints\_dir=r'I:\model\_ckpts\pix2pix\_02'  
p2p.create\_model()  
p2p\_net=p2p.model

initialize network with normal  
initialize network with normal  
model [Pix2PixModel] was created  
---------- Networks initialized -------------  
[Network G] Total number of parameters : 54.414 M  
[Network D] Total number of parameters : 2.769 M  
-----------------------------------------------

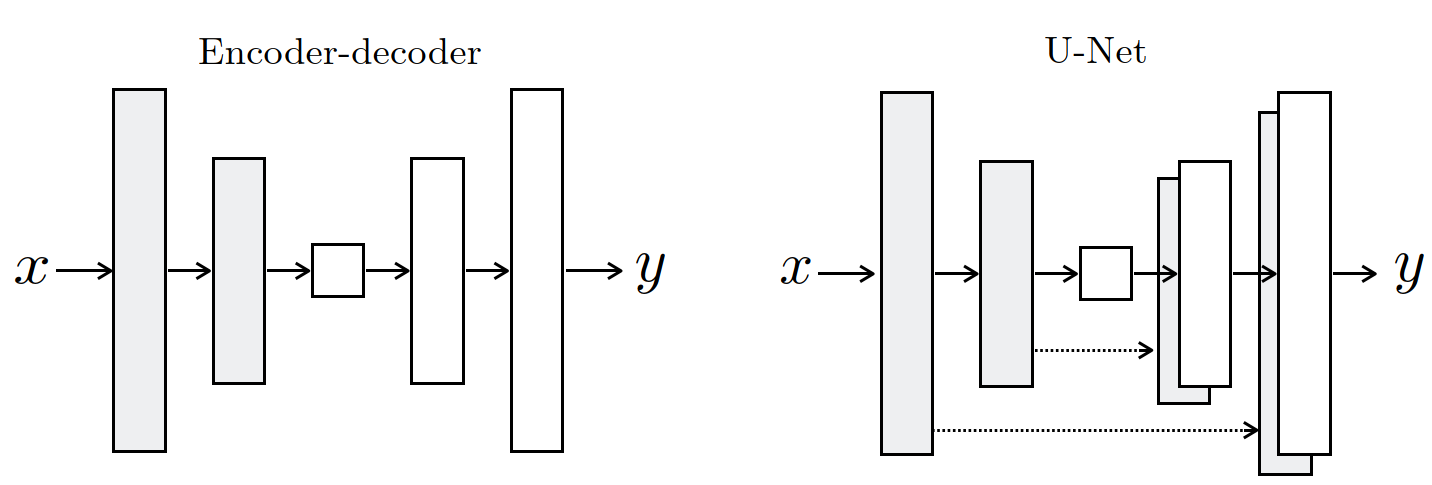
查看网络。

p2p\_net.netG

DataParallel(  
 (module): UnetGenerator(  
 (model): UnetSkipConnectionBlock(  
 (model): Sequential(  
 (0): Conv2d(3, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (1): UnetSkipConnectionBlock(  
 (model): Sequential(  
 (0): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (1): Conv2d(64, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (3): UnetSkipConnectionBlock(  
 (model): Sequential(  
 (0): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (1): Conv2d(128, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (3): UnetSkipConnectionBlock(  
 (model): Sequential(  
 (0): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (1): Conv2d(256, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (3): UnetSkipConnectionBlock(  
 (model): Sequential(  
 (0): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (1): Conv2d(512, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (3): UnetSkipConnectionBlock(  
 (model): Sequential(  
 (0): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (1): Conv2d(512, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (3): UnetSkipConnectionBlock(  
 (model): Sequential(  
 (0): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (1): Conv2d(512, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (3): UnetSkipConnectionBlock(  
 (model): Sequential(  
 (0): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (1): Conv2d(512, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (2): ReLU(inplace=True)  
 (3): ConvTranspose2d(512, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (4): ReLU(inplace=True)  
 (5): ConvTranspose2d(1024, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (6): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (4): ReLU(inplace=True)  
 (5): ConvTranspose2d(1024, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (6): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (4): ReLU(inplace=True)  
 (5): ConvTranspose2d(1024, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (6): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (4): ReLU(inplace=True)  
 (5): ConvTranspose2d(1024, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (4): ReLU(inplace=True)  
 (5): ConvTranspose2d(512, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (4): ReLU(inplace=True)  
 (5): ConvTranspose2d(256, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (6): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 )  
 )  
 (2): ReLU(inplace=True)  
 (3): ConvTranspose2d(128, 3, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1))  
 (4): Tanh()  
 )  
 )  
 )  
)

#### 1) 带跳跃（skips）的网络

cGAN 网络的设计与 StyleGAN 网络设计具有相似一致的逻辑内核，StyleGAN 通过连续不同深度分别对接由噪声向量经过多层感知机非线性映射网络得到的控制向量，解缠连续深度特征影响；而 cGAN 将编码器和解码器对应的卷积层和逆卷积层连接起来，同一“深度”互相对应。如图[]：



上图左，一般编码器和解码器网络是通过一系列逐步下采样的层后，将该过程对应反转为上采样的连续过程，那么不同深度之间的分布特征发生了传递共享，这类似于 PGGAN 网络连续上采样后再继续下采样的过程。上图右，cGAN 跳跃连接的方法将第层和第层对应连接起来，其中为层的总数 ，那么就实现了对应深度卷积层特征的对接，避免了其它深度特征的影响。核心代码如下：

class UnetGenerator(nn.Module):  
 """Create a Unet-based generator"""  
  
 def \_\_init\_\_(self, input\_nc, output\_nc, num\_downs, ngf=64, norm\_layer=nn.BatchNorm2d, use\_dropout=False):  
 """Construct a Unet generator  
 Parameters:  
 input\_nc (int) -- the number of channels in input images  
 output\_nc (int) -- the number of channels in output images  
 num\_downs (int) -- the number of downsamplings in UNet. For example, # if |num\_downs| == 7,  
 image of size 128x128 will become of size 1x1 # at the bottleneck  
 ngf (int) -- the number of filters in the last conv layer  
 norm\_layer -- normalization layer  
  
 We construct the U-Net from the innermost layer to the outermost layer.  
 It is a recursive process.  
 """  
 super(UnetGenerator, self).\_\_init\_\_()  
 # construct unet structure  
 unet\_block = UnetSkipConnectionBlock(ngf \* 8, ngf \* 8, input\_nc=None, submodule=None, norm\_layer=norm\_layer, innermost=True) # add the innermost layer  
 for i in range(num\_downs - 5): # add intermediate layers with ngf \* 8 filters  
 unet\_block = UnetSkipConnectionBlock(ngf \* 8, ngf \* 8, input\_nc=None, submodule=unet\_block, norm\_layer=norm\_layer, use\_dropout=use\_dropout)  
 # gradually reduce the number of filters from ngf \* 8 to ngf  
 unet\_block = UnetSkipConnectionBlock(ngf \* 4, ngf \* 8, input\_nc=None, submodule=unet\_block, norm\_layer=norm\_layer)  
 unet\_block = UnetSkipConnectionBlock(ngf \* 2, ngf \* 4, input\_nc=None, submodule=unet\_block, norm\_layer=norm\_layer)  
 unet\_block = UnetSkipConnectionBlock(ngf, ngf \* 2, input\_nc=None, submodule=unet\_block, norm\_layer=norm\_layer)  
 self.model = UnetSkipConnectionBlock(output\_nc, ngf, input\_nc=input\_nc, submodule=unet\_block, outermost=True, norm\_layer=norm\_layer) # add the outermost layer  
  
 def forward(self, input):  
 """Standard forward"""  
 return self.model(input)  
  
  
class UnetSkipConnectionBlock(nn.Module):  
 """Defines the Unet submodule with skip connection.  
 X -------------------identity----------------------  
 |-- downsampling -- |submodule| -- upsampling --|  
 """  
  
 def \_\_init\_\_(self, outer\_nc, inner\_nc, input\_nc=None,  
 submodule=None, outermost=False, innermost=False, norm\_layer=nn.BatchNorm2d, use\_dropout=False):  
 """Construct a Unet submodule with skip connections.  
  
 Parameters:  
 outer\_nc (int) -- the number of filters in the outer conv layer  
 inner\_nc (int) -- the number of filters in the inner conv layer  
 input\_nc (int) -- the number of channels in input images/features  
 submodule (UnetSkipConnectionBlock) -- previously defined submodules  
 outermost (bool) -- if this module is the outermost module  
 innermost (bool) -- if this module is the innermost module  
 norm\_layer -- normalization layer  
 use\_dropout (bool) -- if use dropout layers.  
 """  
 super(UnetSkipConnectionBlock, self).\_\_init\_\_()  
 self.outermost = outermost  
 if type(norm\_layer) == functools.partial:  
 use\_bias = norm\_layer.func == nn.InstanceNorm2d  
 else:  
 use\_bias = norm\_layer == nn.InstanceNorm2d  
 if input\_nc is None:  
 input\_nc = outer\_nc  
 downconv = nn.Conv2d(input\_nc, inner\_nc, kernel\_size=4,  
 stride=2, padding=1, bias=use\_bias)  
 downrelu = nn.LeakyReLU(0.2, True)  
 downnorm = norm\_layer(inner\_nc)  
 uprelu = nn.ReLU(True)  
 upnorm = norm\_layer(outer\_nc)  
  
 if outermost:  
 upconv = nn.ConvTranspose2d(inner\_nc \* 2, outer\_nc,  
 kernel\_size=4, stride=2,  
 padding=1)  
 down = [downconv]  
 up = [uprelu, upconv, nn.Tanh()]  
 model = down + [submodule] + up  
 elif innermost:  
 upconv = nn.ConvTranspose2d(inner\_nc, outer\_nc,  
 kernel\_size=4, stride=2,  
 padding=1, bias=use\_bias)  
 down = [downrelu, downconv]  
 up = [uprelu, upconv, upnorm]  
 model = down + up  
 else:  
 upconv = nn.ConvTranspose2d(inner\_nc \* 2, outer\_nc,  
 kernel\_size=4, stride=2,  
 padding=1, bias=use\_bias)  
 down = [downrelu, downconv, downnorm]  
 up = [uprelu, upconv, upnorm]  
  
 if use\_dropout:  
 model = down + [submodule] + up + [nn.Dropout(0.5)]  
 else:  
 model = down + [submodule] + up  
  
 self.model = nn.Sequential(\*model)  
  
 def forward(self, x):  
 if self.outermost:  
 return self.model(x)  
 else: # add skip connections  
 return torch.cat([x, self.model(x)], 1)

UnetGenerator类实现了基于 U-Net 网络的构建，构建的顺序由最内层开始，然后成对构建，直至最外层。使用torch.cat([x, self.model(x)], 1)实现对应层的连接。

#### 2) 马尔可夫判别器（PatchGAN）

如果用损失，生成的图像会产生模糊的效果（未捕获到高频信息（high-frequency）），但图像的整体结构是趋于吻合的（能够捕获到低频信息（low-frequency））。为了解决上述问题，限制网络仅对高频结构建模，依靠项强制低频的正确性，只需将作用于分割的图像单元上，称之为 PatchGAN。通过对生成图像中每个分割图像单元判断真假，平均所有响应，作为的输出。该种方法有效的将图像视为马尔科夫随机场（Markov random ﬁeld），生成图像的像素类型对应目标图像（）像素类型在图像分割单元大小内（邻域）的邻里关系得以推断，即像素只与邻域的像素点信息有关，而和邻域外的像素点无关。

实现 PatchGAN 的方法，通过对比 cGAN 和 WGAN 的输出可以观察到对于cGAN，输出的数据形状为(1, 1, 30, 30)，而对于 WGAN 输出形状为(1, 1, 1, 1)，因此可以判断作者仅仅通过卷积的方法实现了 PatchGAN 分割图像单元判断真假的目的。同时，在损失函数计算中，如果配置opt.train.gan\_mode = 'vanilla' ，则调用loss = nn.BCEWithLogitsLoss()，其中参数reduction='mean'；如果配置opt.train.gan\_mode = 'wgangp'，则通过如下方法计算损失：

elif self.gan\_mode == 'wgangp':  
 if target\_is\_real:  
 loss = -prediction.mean()  
 else:  
 loss = prediction.mean()

cGAN 网络打印如下：

p2p\_net.netD

DataParallel(  
 (module): NLayerDiscriminator(  
 (model): Sequential(  
 (0): Conv2d(6, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1))  
 (1): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (2): Conv2d(64, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (4): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (5): Conv2d(128, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
 (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (7): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (8): Conv2d(256, 512, kernel\_size=(4, 4), stride=(1, 1), padding=(1, 1), bias=False)  
 (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (10): LeakyReLU(negative\_slope=0.2, inplace=True)  
 (11): Conv2d(512, 1, kernel\_size=(4, 4), stride=(1, 1), padding=(1, 1))  
 )  
 )  
)

使用[torchshape](https://github.com/yuezuegu/torchshape)⑤库提供的方法计算输出形状，目前支持的操作有：

* nn.Conv1d
* nn.Conv2d
* nn.Linear
* nn.MaxPool1d
* nn.MaxPool2d
* nn.AvgPool1d
* nn.AvgPool2d
* nn.Flatten
* nn.BatchNorm1d
* nn.BatchNorm2d

计算 cGAN 的网络各卷积层输出形状。

shape=(1,6,256,256)  
for op in p2p\_net.netD.module.model:  
 if isinstance(op, torch.nn.modules.conv.Conv2d):  
 shape=tensorshape(op,shape)  
 print(f'{shape} \t{op}')

(1, 64, 128, 128) Conv2d(6, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1))  
(1, 128, 64, 64) Conv2d(64, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
(1, 256, 32, 32) Conv2d(128, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
(1, 512, 31, 31) Conv2d(256, 512, kernel\_size=(4, 4), stride=(1, 1), padding=(1, 1), bias=False)  
(1, 1, 30, 30) Conv2d(512, 1, kernel\_size=(4, 4), stride=(1, 1), padding=(1, 1))

计算 WGAN 的 网络各卷积层输出形状。

critic=basic\_critic(in\_size=256, n\_channels=3,n\_features=64,act\_cls=partial(torch.nn.LeakyReLU, negative\_slope=0.2))  
flattened\_critic=[module for module in critic.modules() if not isinstance(module, torch.nn.Sequential)]  
shape=(1,3,256,256)  
for op in flattened\_critic:  
 if isinstance(op, torch.nn.modules.conv.Conv2d):  
 shape=tensorshape(op,shape)  
 print(f'{shape} \t{op}')

(1, 64, 128, 128) Conv2d(3, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1))  
(1, 128, 64, 64) Conv2d(64, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
(1, 256, 32, 32) Conv2d(128, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
(1, 512, 16, 16) Conv2d(256, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
(1, 1024, 8, 8) Conv2d(512, 1024, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
(1, 2048, 4, 4) Conv2d(1024, 2048, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
(1, 1, 1, 1) Conv2d(2048, 1, kernel\_size=(4, 4), stride=(1, 1))

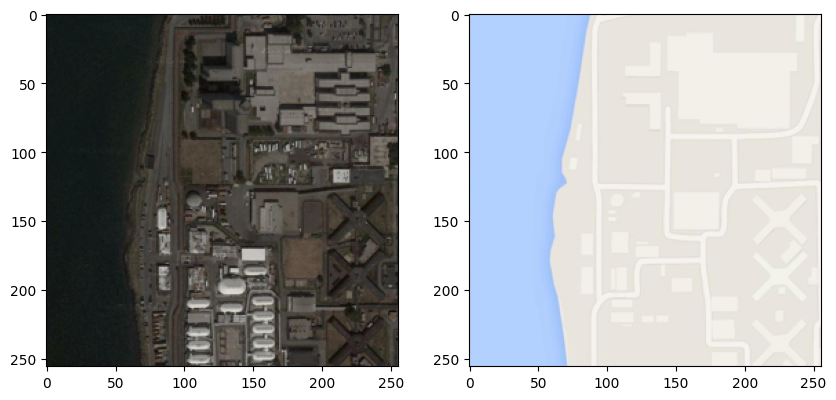
* 其它训练技巧

1. 优化的同时，将目标除以2，降低相对于的学习速度，对应代码为self.loss\_D = (self.loss\_D\_fake + self.loss\_D\_real) \* 0.5；
2. 使用 minibatch SGD，并应用 Adam 解释器，配置学习率为2e-4，动量参数（momentum parameters）为，$ *{1} =0.5；* {2} =0.999$；
3. 配置批量大小在1到10之间，如果配置为1，称为“实例归一化（instance normal-ization）”。

#### 3) 查看数据，训练和测试生成

cGAN 通用框架含有对拼接图像的拆分处理，构建完数据集和数据加载器后，图像实现拆分，以A和B为键，标识A和B数据。根据转化的方向，一个用于，一个用于。

images=next(iter(p2p.dataset))  
fig, axs = plt.subplots(1,2,figsize=(10, 10))  
axs[0].imshow(adjust\_dynamic\_range(images['A'].permute(0,2,3,1)[0]))  
axs[1].imshow(adjust\_dynamic\_range(images['B'].permute(0,2,3,1)[0]))  
plt.show()



visualizer方法用于实现训练过程中，计算阶段性训练模型预测（翻译）的结果图像，并存储至p2p.opt.basic.checkpoints\_dir配置存储路径下。

p2p.opt.train.visual.display\_id=-1  
p2p.visualizer()

create web directory I:\model\_ckpts\pix2pix\_02\web...

* 训练模型

p2p.train()

learning rate 0.0002000 -> 0.0002000  
(epoch: 1, iters: 100, time: 0.067, data: 7.584) G\_GAN: 1.612 G\_L1: 20.842 D\_real: 0.013 D\_fake: 0.471   
(epoch: 1, iters: 200, time: 0.068, data: 0.001) G\_GAN: 1.667 G\_L1: 27.958 D\_real: 0.003 D\_fake: 0.424   
(epoch: 1, iters: 300, time: 0.066, data: 0.001) G\_GAN: 1.009 G\_L1: 8.658 D\_real: 1.523 D\_fake: 0.310   
(epoch: 1, iters: 400, time: 0.173, data: 0.001) G\_GAN: 1.162 G\_L1: 26.299 D\_real: 0.463 D\_fake: 0.128   
(epoch: 1, iters: 500, time: 0.070, data: 0.001) G\_GAN: 2.325 G\_L1: 19.845 D\_real: 0.102 D\_fake: 0.118   
(epoch: 1, iters: 600, time: 0.072, data: 0.001) G\_GAN: 2.664 G\_L1: 14.551 D\_real: 0.007 D\_fake: 2.713   
(epoch: 1, iters: 700, time: 0.067, data: 0.001) G\_GAN: 1.895 G\_L1: 18.732 D\_real: 0.244 D\_fake: 0.405   
(epoch: 1, iters: 800, time: 0.068, data: 0.001) G\_GAN: 3.193 G\_L1: 22.693 D\_real: 0.039 D\_fake: 0.132   
(epoch: 1, iters: 900, time: 0.066, data: 0.001) G\_GAN: 2.161 G\_L1: 16.417 D\_real: 0.475 D\_fake: 0.319   
(epoch: 1, iters: 1000, time: 0.068, data: 0.001) G\_GAN: 1.669 G\_L1: 20.868 D\_real: 0.142 D\_fake: 0.216   
End of epoch 1 / 200 Time Taken: 73 sec  
learning rate 0.0002000 -> 0.0002000  
(epoch: 2, iters: 4, time: 0.059, data: 0.000) G\_GAN: 2.092 G\_L1: 20.703 D\_real: 0.050 D\_fake: 0.532   
(epoch: 2, iters: 104, time: 0.177, data: 0.000) G\_GAN: 2.218 G\_L1: 27.503 D\_real: 0.001 D\_fake: 0.537   
(epoch: 2, iters: 204, time: 0.070, data: 0.001) G\_GAN: 0.933 G\_L1: 28.514 D\_real: 1.289 D\_fake: 0.113   
(epoch: 2, iters: 304, time: 0.068, data: 0.001) G\_GAN: 1.923 G\_L1: 21.235 D\_real: 0.117 D\_fake: 0.171   
(epoch: 2, iters: 404, time: 0.066, data: 0.001) G\_GAN: 2.134 G\_L1: 21.554 D\_real: 0.600 D\_fake: 0.058   
(epoch: 2, iters: 504, time: 0.075, data: 0.001) G\_GAN: 1.511 G\_L1: 14.303 D\_real: 0.232 D\_fake: 0.148   
(epoch: 2, iters: 604, time: 0.076, data: 0.001) G\_GAN: 2.297 G\_L1: 23.186 D\_real: 0.609 D\_fake: 0.049   
(epoch: 2, iters: 704, time: 0.069, data: 0.001) G\_GAN: 1.207 G\_L1: 13.543 D\_real: 0.620 D\_fake: 0.989   
(epoch: 2, iters: 804, time: 0.069, data: 0.001) G\_GAN: 2.229 G\_L1: 23.092 D\_real: 0.144 D\_fake: 0.162   
(epoch: 2, iters: 904, time: 0.194, data: 0.000) G\_GAN: 1.966 G\_L1: 22.935 D\_real: 0.467 D\_fake: 0.087   
(epoch: 2, iters: 1004, time: 0.066, data: 0.001) G\_GAN: 2.268 G\_L1: 24.545 D\_real: 0.003 D\_fake: 0.245   
End of epoch 2 / 200 Time Taken: 66 sec  
learning rate 0.0002000 -> 0.0002000  
(epoch: 3, iters: 8, time: 0.084, data: 0.001) G\_GAN: 1.096 G\_L1: 15.798 D\_real: 0.325 D\_fake: 0.656

* 测试和生成数据

用预训练的模型生成图像，查看预测结果。图像存储于p2p.opt.test.results\_dir配置存储路径下，如下对应A和B的部分生成图像。

p2p=test.Pix2pix\_test()  
  
p2p.opt.basic.dataroot=r'I:\data\pix2pix\_dataset\maps'  
p2p.opt.test.results\_dir=r'I:\model\_ckpts\pix2pix\_02'  
p2p.opt.dataset.dataset\_mode='aligned'   
p2p.opt.dataset.direction='BtoA'  
p2p.opt.train.saveload.phase='val'  
p2p.opt.basic.isTrain=False  
p2p.opt.model.model='pix2pix'  
  
p2p.create\_dataset()  
  
p2p.opt.basic.checkpoints\_dir=r'I:\model\_ckpts\pix2pix\_02'  
p2p.create\_model()  
m=p2p.model   
  
p2p.test()

dataset [AlignedDataset] was created  
initialize network with normal  
model [Pix2PixModel] was created  
loading the model from I:\model\_ckpts\pix2pix\_02\latest\_net\_G.pth  
---------- Networks initialized -------------  
[Network G] Total number of parameters : 54.414 M  
-----------------------------------------------  
creating web directory I:\model\_ckpts\pix2pix\_02\val\_latest  
processing (0000)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\1.jpg']  
processing (0005)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\1002.jpg']  
processing (0010)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\1007.jpg']  
processing (0015)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\1011.jpg']  
processing (0020)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\1016.jpg']  
processing (0025)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\1020.jpg']  
processing (0030)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\1025.jpg']  
processing (0035)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\103.jpg']  
processing (0040)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\1034.jpg']  
processing (0045)-th image... ['I:\\data\\pix2pix\_dataset\\maps\\val\\1039.jpg']

| read-A | real-B | fake-b |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

## 3.4.2 NAIP 遥感影像和土地覆盖类型（land cover，LC）之间的翻译转化

### 3.4.2.1 构建训练样本数据

主要使用[TorchGeo](https://github.com/microsoft/torchgeo)⑥和[PIL（Pillow）](https://pillow.readthedocs.io/en/stable/)⑦库构建训练数据集。关于TorchGeo样本提取可以参考*NAIP航拍影像与分割模型库及Colaboratory和Planetary Computer Hub*一章的阐释。

# IPython extension to reload modules before executing user code.  
%load\_ext autoreload   
# Reload all modules (except those excluded by %aimport) every time before executing the Python code typed.  
%autoreload 2   
import usda.geodata\_process as usda\_geoprocess  
import usda.imgs\_process as usda\_imgs  
  
from yacs.config import CfgNode as CN  
import os  
from torchgeo.datasets import NAIP,ChesapeakeDE,stack\_samples   
from torchgeo.samplers import RandomGeoSampler  
from torch.utils.data import DataLoader  
import matplotlib.pyplot as plt  
import torchvision.transforms as T  
from torchgeo.datasets import unbind\_samples  
import numpy as np  
from PIL import Image  
from tqdm import tqdm

配置原始遥感影像和 LC 数据文件所在存储路径，及提取后样本数据存储位置，包括单独的影像文件夹cfg.sample\_imgs\_dir，LC 文件夹cfg.sample\_lc\_dir及影像和LC拼接图像的存储位置cfg.sample\_img\_lc\_dir。

cfg=CN()  
cfg.Chesapeake\_root=r'E:\data\Delaware'  
cfg.Chesapeake\_LC=os.path.join(cfg.Chesapeake\_root,'LC')  
cfg.Chesapeake\_imagery=os.path.join(cfg.Chesapeake\_root,'imagery')  
cfg.sample\_imgs\_dir=r'I:\data\naip\_lc4pix2pix\imgs'  
cfg.sample\_lc\_dir=r'I:\data\naip\_lc4pix2pix\lc'  
cfg.sample\_img\_lc\_dir=r'I:\data\naip\_lc4pix2pix\img\_lc'

构建数据集和数据加载器。

naip=NAIP(cfg.Chesapeake\_imagery)  
chesapeake=ChesapeakeDE(cfg.Chesapeake\_LC, crs=naip.crs, res=naip.res, download=False)  
dataset=naip & Chesapeake

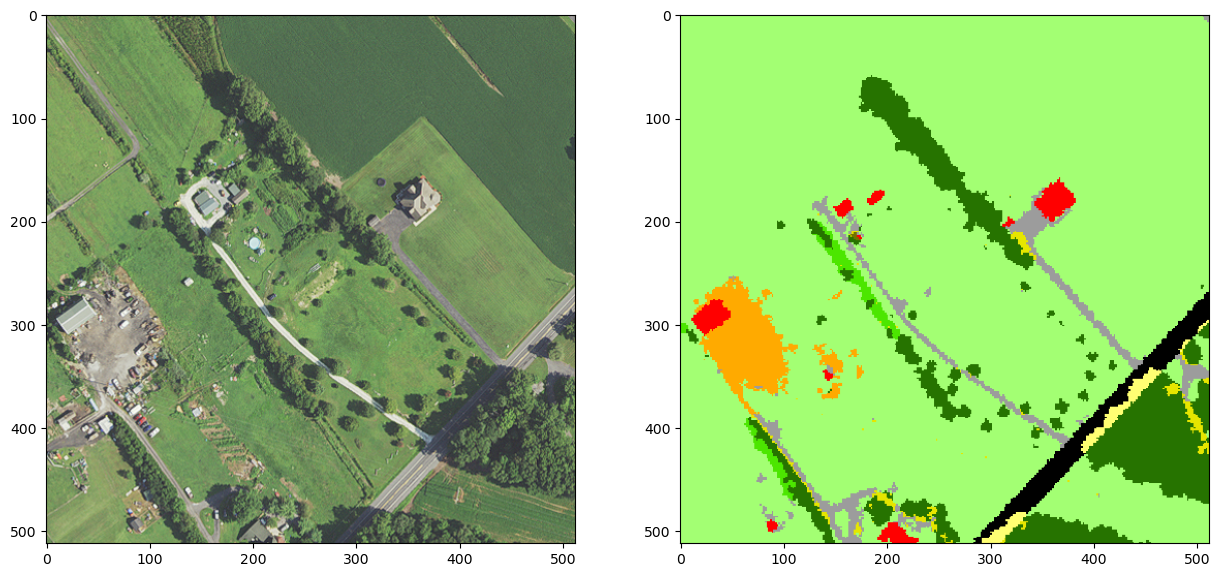
采样的图像大小配置为size=512，总共产生影像和LC对应的length=10000)个随机样本。

sampler=RandomGeoSampler(dataset, size=512, length=10000)  
dataloader=DataLoader(dataset, sampler=sampler, collate\_fn=stack\_samples)  
for batch in dataloader:  
 image=batch["image"]  
 target=batch["mask"]  
 break  
   
print(f'sample length={len(sampler)};\nimage shape:{image.shape};\ntarget shape:{target.shape}')

sample length=10000;  
image shape:torch.Size([1, 4, 512, 512]);  
target shape:torch.Size([1, 1, 512, 512])

配置 LC 的颜色，将 LC 分类数值转化为颜色值（3个通道）。

LC\_color\_dict={  
 0: (0, 0, 0),  
 1: (0, 197, 255),  
 2: (0, 168, 132),  
 3: (38, 115, 0),  
 4: (76, 230, 0),  
 5: (163, 255, 115),  
 6: (255, 170, 0),  
 7: (255, 0, 0,),  
 8: (156, 156, 156),  
 9: (0, 0, 0),  
 10: (115, 115, 0),  
 11: (230, 230, 0),  
 12: (255, 255, 115),  
 13: (197, 0, 255),  
 }  
  
target\_T=target[0].permute(1,2,0)  
target\_img=torch.tensor([LC\_color\_dict[i.item()] for i in torch.flatten(target\_T)]).reshape(target\_T.shape[0],target\_T.shape[1],3)  
img=np.transpose(image[0][:3],(1,2,0))  
  
fig, axs = plt.subplots(1,2,figsize=(15, 15))  
axs[0].imshow(img/255)  
axs[1].imshow(target\_img)  
plt.show()



示例影像和LC拼接后的图像。

img\_pil=Image.fromarray(img.numpy().astype('uint8'),'RGB')  
target\_pil=Image.fromarray(target\_img.numpy().astype('uint8'),'RGB')  
img\_target=usda\_imgs.imgs\_concat\_h(img\_pil,target\_pil)  
img\_target



批量生成样本数据。

如果要生成大数量的样本，可使用多线程。下述代码未使用。

suffix='.jpg'  
  
for i,batch in enumerate(dataloader):  
 image=batch["image"]  
 target=batch["mask"]  
   
 target\_T=target[0].permute(1,2,0)  
 target\_img=torch.tensor([LC\_color\_dict[i.item()] for i in torch.flatten(target\_T)]).reshape(target\_T.shape[0],target\_T.shape[1],3)  
 img=np.transpose(image[0][:3],(1,2,0))   
  
 img\_pil=Image.fromarray(img.numpy().astype('uint8'),'RGB')  
 target\_pil=Image.fromarray(target\_img.numpy().astype('uint8'),'RGB')  
 img\_target=usda\_imgs.imgs\_concat\_h(img\_pil,target\_pil)  
  
 img\_pil.save(os.path.join(cfg.sample\_imgs\_dir,f'{i}{suffix}'))  
 target\_pil.save(os.path.join(cfg.sample\_lc\_dir,f'{i}{suffix}'))  
 img\_target.save(os.path.join(cfg.sample\_img\_lc\_dir,f'{i}{suffix}'))  
   
 print(f'---{i}',end='\r')

---9999

### 3.4.2.2 训练影像到LC和LC到影像的 cGAN 网络模型

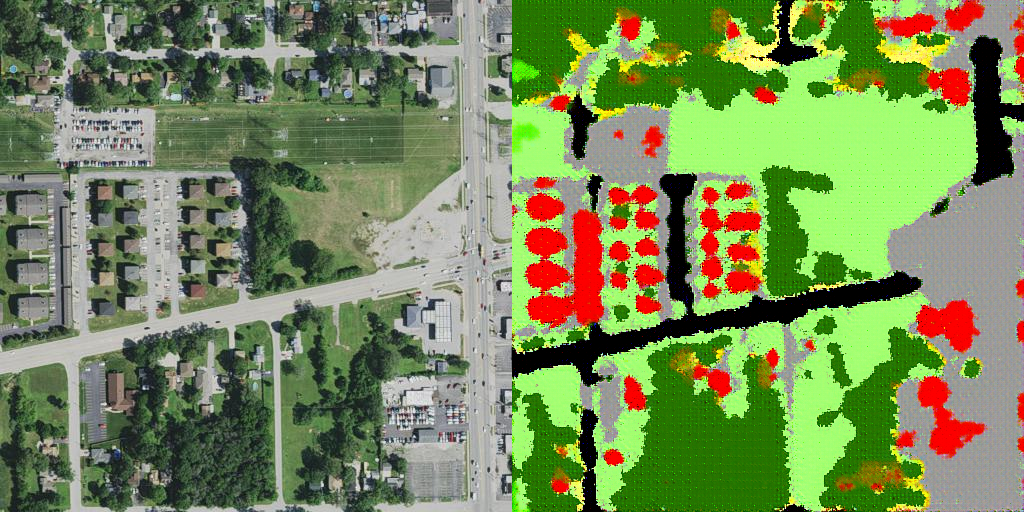
使用 cGAN 的 pix2pix 方法构建 NAIP 高分辨率（1m）航拍影像和 LC（13类）之间的映射关系，模型训练的方法同上。调用训练后的模型用于影像和LC之间的互相转化，下述使用了芝加哥区域的 NAIP 航拍影像，应用训练的模型实验。

# IPython extension to reload modules before executing user code.  
%load\_ext autoreload   
# Reload all modules (except those excluded by %aimport) every time before executing the Python code typed.  
%autoreload 2   
  
from usda.migrated\_project.pix2pix import A2B  
from usda.migrated\_project.pix2pix import sketch\_A2B  
import usda.imgs\_process as usda\_imgs  
from yacs.config import CfgNode as CN

转换芝加哥区域 NAIP 真实影像到 LC 分类，能够大概分类主要的地物关系，但是细节，例如建筑、不透水地面等边缘比较模糊。一方面由于训练的 LC 数据集本身边缘并不清晰；另一方面由于 pix2pix 本身网络结构的影响。在进一步的探索中，除了使用精度高的 LC 数据集外，可以借鉴 StyleGAN 的网络。

pretrained\_model\_Img2LC\_fn=r'I:\model\_ckpts\pix2pix\pix2pix4Img2LC\latest\_net\_G.pth'  
img\_fn=r'I:\data\NAIP4StyleGAN\naip\_512\0\_167.jpg'  
  
AorB,BorA=A2B.A2B\_generator(pretrained\_model\_Img2LC\_fn,img\_fn)  
AnB=usda\_imgs.imgs\_concat\_h(AorB,BorA)  
AnB

initialize network with normal



LC 分类到影像的转化，人工建筑部分可以大概“翻译”，通常模糊且边界不清晰。因为并不关注植被部分的细节，因此植被的“翻译”相对更理想。

pretrained\_mode\_LC2Img\_fn=r'I:\model\_ckpts\pix2pix\pix2pix4LC2IMG\latest\_net\_G.pth'  
img\_fn=r'I:\model\_ckpts\pix2pix\pix2pix4LC2IMG\web\images\epoch001\_real\_A.png'  
  
AorB,BorA=A2B.A2B\_generator(pretrained\_mode\_LC2Img\_fn,img\_fn,cfg)  
AnB=usda\_imgs.imgs\_concat\_h(AorB,BorA)  
AnB

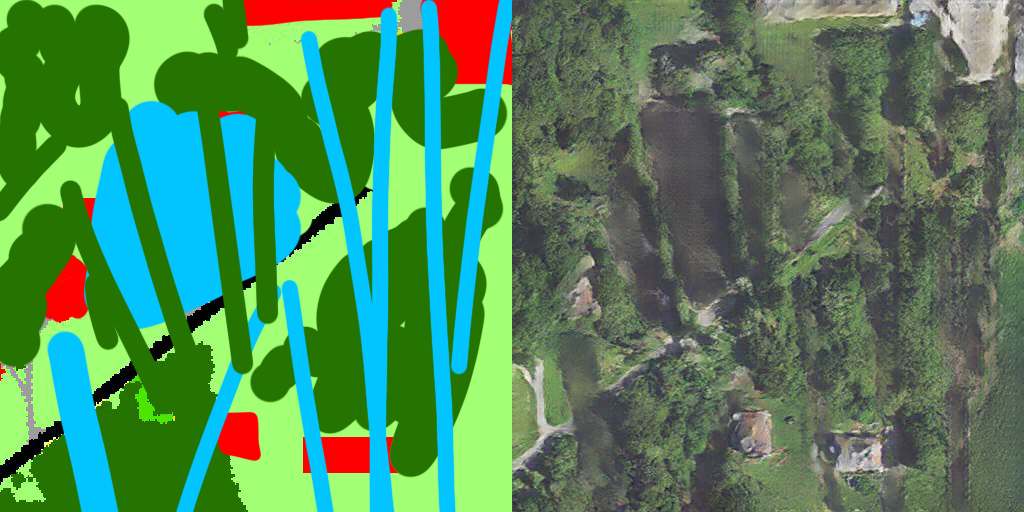
initialize network with normal



下面进行的实验是完全人为涂鸦一个 LC 分类（分类颜色同训练所用 LC），将其转换为影像，并用生成的影像再转化回 LC 分类。从结果可以判断，图像翻译的方式可以辅助规划设计，通过草图到“真实”影像的转化，理解分类布局潜在真实的形态，从而进一步调整分类布局。

img\_fn=r'C:\Users\richi\omen\_richiebao\omen\_github\USDA\_special\_study\imgs\3\_4\_c\3\_4\_c\_02.png'  
  
AorB,BorA=A2B.A2B\_generator(pretrained\_mode\_LC2Img\_fn,img\_fn)  
AnB=usda\_imgs.imgs\_concat\_h(AorB,BorA)  
AnB

initialize network with normal

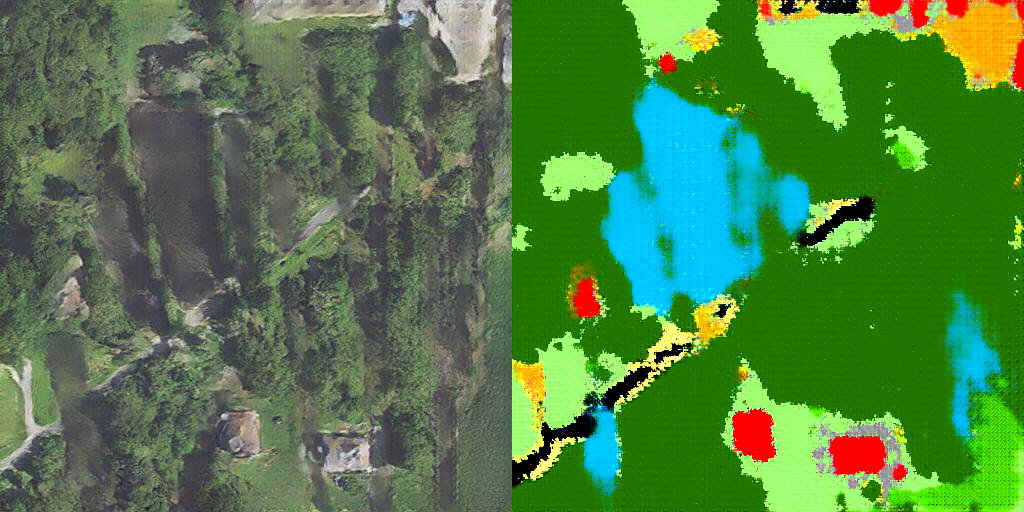


BorA\_save\_fn=r'C:\Users\richi\omen\_richiebao\omen\_github\USDA\_special\_study\imgs\3\_4\_c\3\_4\_c\_02.jpg'  
BorA.save(BorA\_save\_fn)

将涂鸦分类生成的影像再转化回 LC 分类。虽然目前训练的模型还未达到理想的“翻译”要求，但是转化后的 LC 表达显然要比人为涂鸦更为自然，那么从LC涂鸦到影像，再转化回LC的方式，可以辅助规划设计，推断布局的合理性，并优化表达提升工作的效率。

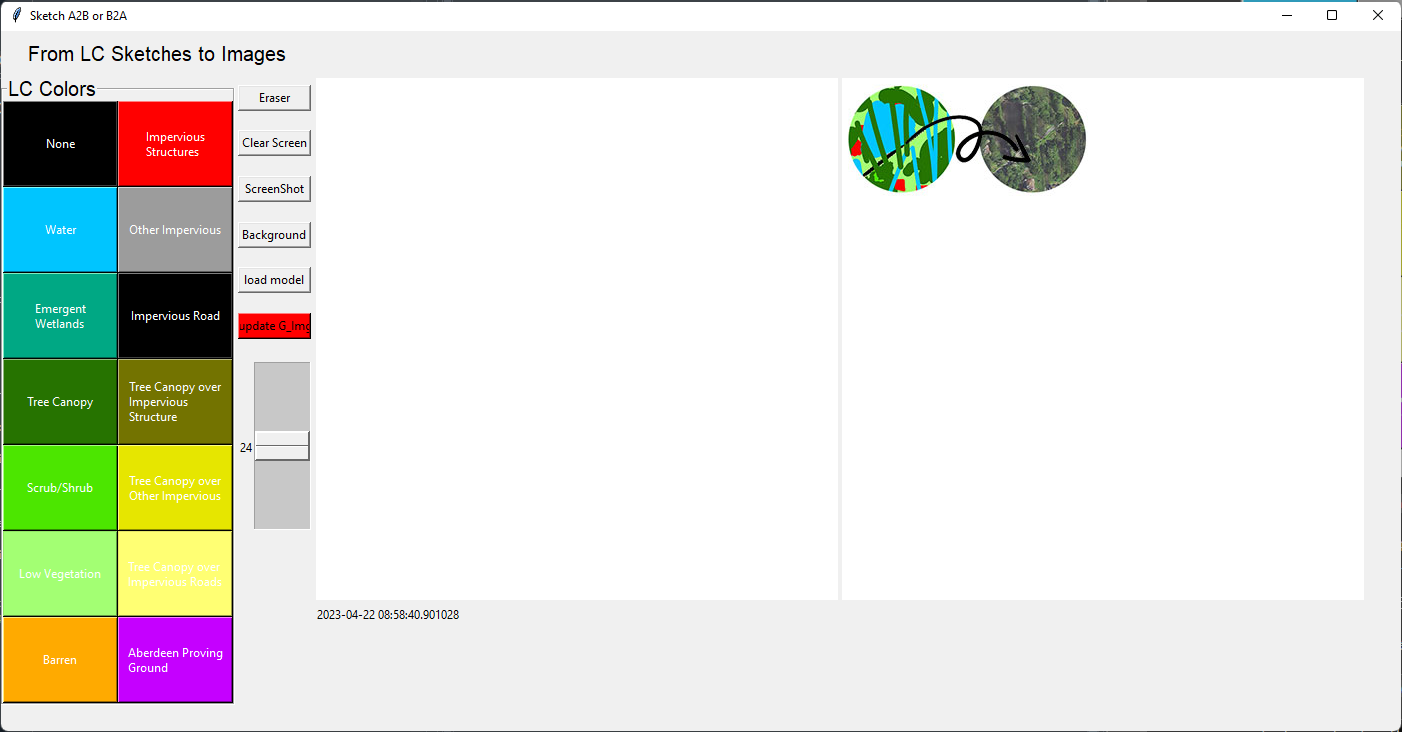
img\_fn=r'I:\data\NAIP4StyleGAN\naip\_512\0\_167.jpg'  
  
AorB,BorA=A2B.A2B\_generator(pretrained\_model\_Img2LC\_fn,BorA\_save\_fn)  
AnB=usda\_imgs.imgs\_concat\_h(AorB,BorA)  
AnB

initialize network with normal



## 3.4.3 辅助规划设计工具的构建

上述“翻译”实验需要在绘图软件中绘制 LC后，再调入到程序执行转化，操作模式脱节；并且需要吸取分类颜色后绘图，令工作繁琐。使用[Tkinter](https://www.pythonguis.com/tkinter/)⑧库构建辅助规划设计工具的雏形，实现 LC 涂鸦和翻译为影像。该部分实现位于usda.migrated\_project.pix2pix下的sketch\_A2B模块中，界面如下图：



在Tkinter中使用了canvas的postscript方法保存为EPS格式图像，并配合PIL库将EPS图像转化为常规的PNG或者JPG图像，因此使用到[Ghostscript](https://ghostscript.com/releases/gsdnld.html)⑨，需要下载安装，同时，通过下述语句调用，

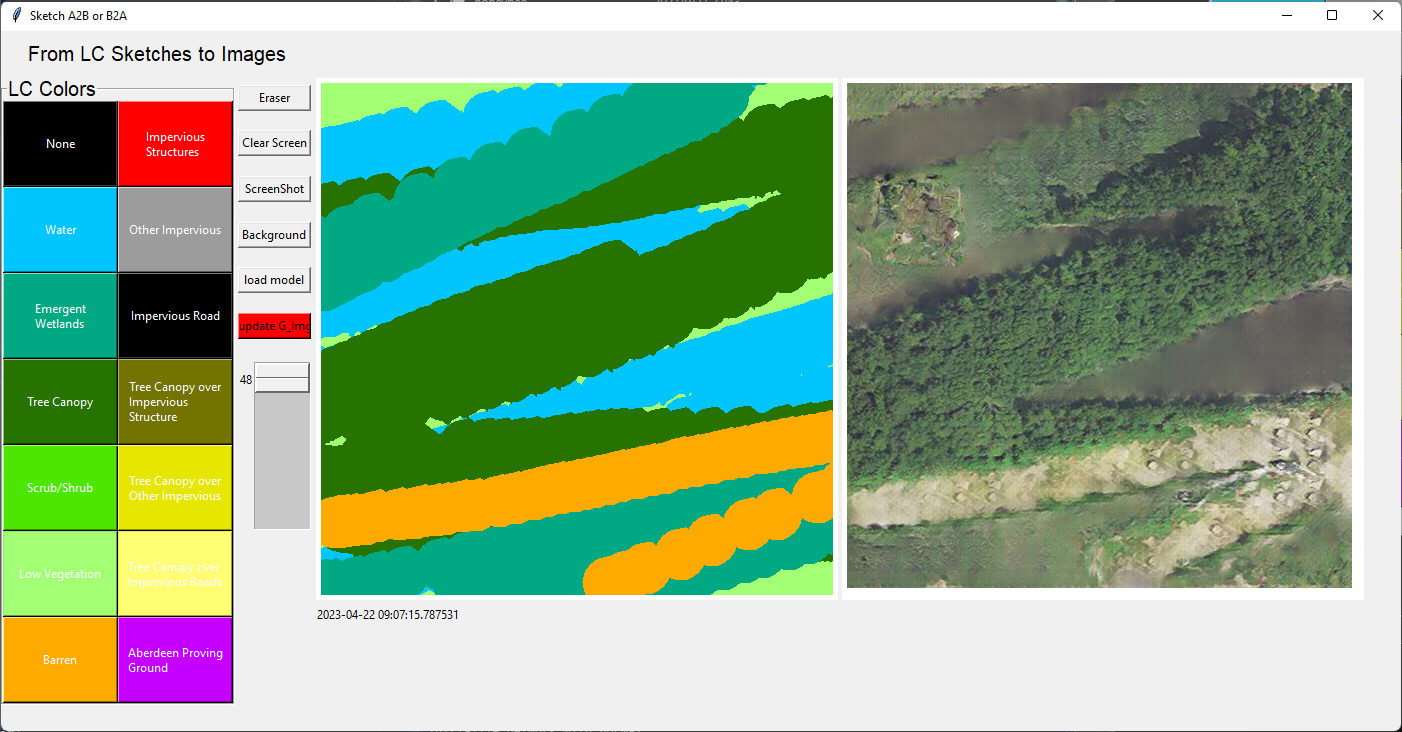
from PIL import EpsImagePlugin  
EpsImagePlugin.gs\_windows\_binary =r'C:\Program Files\gs\gs10.01.1\bin\gswin64c'

上述Ghostscript路径配置默认写在了Sketch\_A2B模块中，在实验Sketch\_A2B工具时，可以将Ghostscript默认安装到上述路径，或者修改USDA库Sketch\_A2B模块中对应的调入路径代码。

调用Sketch\_A2B工具，涂鸦 LC并翻译。其中，load model按钮用于加载训练好的网络模型；update G\_img按钮用于影像转化，实验结果如下。

app=sketch\_A2B.Sketch\_A2B()  
app.mainloop()

initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal  
initialize network with normal



## 3.4.4 缺失区域的修复实验

### 3.4.4.1 建立随机形状遮罩的影像数据集

使用[cv2（OpenCV-Python）](https://opencv24-python-tutorials.readthedocs.io/en/latest/index.html)⑩库协助处理具有随机形状覆盖（遮罩）的影像，该方法已写入USDA库，于usda\_imgs模块中。原始影像使用前文已经处理的大小为 512 的影像数据集。

# IPython extension to reload modules before executing user code.  
%load\_ext autoreload   
# Reload all modules (except those excluded by %aimport) every time before executing the Python code typed.  
%autoreload 2   
import usda.imgs\_process as usda\_imgs  
import glob, os  
import cv2  
from PIL import Image  
from tqdm import tqdm

naip\_512\_path=r'I:\data\naip\_lc4pix2pix\imgs'  
img\_fns=glob.glob(naip\_512\_path+"/\*.jpg")  
img=cv2.imread(img\_fns[0])  
  
\_,masked\_img=usda\_imgs.random\_shape\_onImage(img,thresh1=130) # thresh2=255（默认值）  
  
img\_pil=Image.fromarray(img.astype('uint8'),'RGB')  
img\_maskedImg=usda\_imgs.imgs\_concat\_h(img\_pil,masked\_img)  
img\_maskedImg



img\_maskedImg\_path=r'I:\data\naip\_lc4pix2pix\img\_maskedImg'  
suffix='.jpg'  
i=0  
for fn in tqdm(img\_fns):  
 img=cv2.imread(fn)  
 \_,masked\_img=usda\_imgs.random\_shape\_onImage(img,thresh1=130)  
 img\_pil=Image.fromarray(img.astype('uint8'),'RGB')  
 img\_maskedImg=usda\_imgs.imgs\_concat\_h(img\_pil,masked\_img)   
 img\_maskedImg.save(os.path.join(img\_maskedImg\_path,f'{i}{suffix}'))  
 i+=1

100%|██████████████████████████████████████████████████████████████████| 10000/10000 [12:08<00:00, 13.72it/s]

### 3.4.4.2 训练与结果

训练方法同上，结果如下：

| read-A | real-B | fake-b |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

训练结果质量与随机形状遮罩大小，和训练的数据集影像大小和内容等有关。遮罩大小普遍高于建筑尺度，影像多以自然景观为主，从结果能够发现，自然景观部分的修复较好，但是人工建筑部分并不理想。

注释（Notes）：

① eCognition，（<https://geospatial.trimble.com/what-is-ecognition>）。

② CycleGAN and pix2pix，（<https://phillipi.github.io/pix2pix/>）。

③ cGAN 通用框架maps数据集，（<http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/>）。

④ cGAN 通用框架预训练模型map2sat或sat2map，（<http://efrosgans.eecs.berkeley.edu/pix2pix/models-pytorch/>）。

⑤ torchshape，（<https://github.com/yuezuegu/torchshape>）。

⑥ TorchGeo，（<https://github.com/microsoft/torchgeo>）。

⑦ PIL（Pillow），（<https://pillow.readthedocs.io/en/stable/>）。

⑧ Tkinter，（<https://www.pythonguis.com/tkinter/>）。

⑨ Ghostscript，（<https://ghostscript.com/releases/gsdnld.html>）。

⑩ cv2（OpenCV-Python），（<https://opencv24-python-tutorials.readthedocs.io/en/latest/index.html>）。

参考文献（References）:

[1] Raman, T. A., Kollar, J. & Penman, S. Chapter 17 - SASAKI: Filling the design gap—Urban impressions with AI. in Artificial Intelligence in Urban Planning and Design (eds. As, I., Basu, P. & Talwar, P.) 339–362 (Elsevier, 2022). doi:https://doi.org/10.1016/B978-0-12-823941-4.00002-0.

[2] Isola, P., Zhu, J.-Y., Zhou, T. & Efros, A. A. Image-to-Image Translation with Conditional Adversarial Networks. (2016).

[3] Pix2pix的PyTorch版本, <https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix>.