MappingSand_epub

April 27, 2023

1 Mapping sand in publicly available geospatial imagery using open-source image segmentation models

An epub for the CSDMS 2023 annual meeting. This notebook is part of the Doodleverse, an ecosystem of software, data, and trained models for geoscientific image segmentation.

Authors:

- Daniel Buscombe, Marda Science[1]
- Sharon Fitzpatrick[1]
- Venus Ku[1]
- Cameron Bodine[2]
- Evan Goldstein[3]

[1] Contracted to U.S. Geological Survey, Pacific Coastal and Marine Science Center, CA, United States [2] School of Informatics, Computing and Cyber Systems, Northern Arizona University Flagstaff, AZ, United States [3] Department of Geography, Environment, and Sustainability, University of North Carolina, Greensboro, NC, United States

1.1 Introduction

1.1.1 Open source data, code, and trained ML models for image segmentation

The Doodleverse is a growing ecosystem of python codes, tensorflow models, and datasets in support of generic deep-learning-based semantic segmentation of geoscientific imagery. This epub demonstrates the functionality of image segmentation models for generic geoscientific tasks, available through Segmentation Zoo, a collection of trained models for potentially broadly applicable image segmentation tasks. Many 'Zoo models (as they are affectionately known) are used in the Seg2Map application, which encodes a more sophisticated and broadly applicable workflow to image segmentation than this notebook represents. Zoo models are based on publicly available datasets and represent an entirely reproducible way to conduct image segmentation.

1.1.2 Mapping Sand

This notebook carries out the following tasks:

- It downloads a time-series of high-resolution orthomosaic imagery tiles, using Google Earth Engine (GEE), from a Region-of-Interest (ROI) provided by a geojson format file.
- It uses GDAL to create a seamless orthomosaic of the image tiles, then chops up that orthomosaic again into tiles small enough for model application. We make the tiles with 50% overlap in each direction, so we are able to create pixelwise averages of model outputs.

- It downloads a specified Zoo model for semantic segmentation of this imagery, from a public archive on Zenodo. The model architecture is provided via HuggingFace, and has been trained using the Segmentation Gym toolbox.
- It then uses this model to identify sand pixels in tiled imagery.
- It uses GDAL to orthomosaic image labels by mosaicing model outputs.
- Finally, we carry out time-series analysis on outputs from multiple years using xarray.

This notebook represents a very stripped-back and basic version of the Seg2Map toolbox. Using Seg2Map, it is possible to reproduce this analysis, i.e. using this model, and this location, as well as numerous additional image segmentation models, at any location inside the conterminuous United States. Here we use a specific model (Buscombe, 2023) that has been trained using the Coast Train dataset described by Buscombe et al. (2023), itself made using Doodler, a Human-in-the-Loop image labeling program (Buscombe et al., 2021). More details about the dataset can be found on the Coast Train website.

The model is called "orthoCT_8class_segformer_7641724", and is for the following classes: water, whitewater, sediment, other_bare_natural_terrain, marsh_vegetation, terrestrial_vegetation, agricultural, development. We'll be applying the model to imagery and using the outputs for the third class, "sediment". The model provides per-class probabilities for each pixel. The model is a Segformer (Xie et al., 2021) model pre-trained on ImageNet, fine-tuned using labels , within the *Segmentation Gym* (Buscombe and Goldstein, 2022) software package. The SegFormer model architecture uses a hierarchical Transformer architecture, called "Mix Transformer", as an encoder, and a lightweight decoder for segmentation. It can yield state-of-the-art performance on semantic segmentation while being more efficient than existing models.

1.1.3 Scientific applications

There are many potential reasons for mapping sand landforms and deposits from imagery. For example, mapping sand allows for monitoring and characterisation of the dynamics of natural sand landforms such as beaches and river mouths such as here, as well as deltas, estuaries, dunes, and perhaps any sandy deposit visible in ~1m spatial footprint imagery. According to a recent study by Bendixen et al. (2021), sand, gravel and crushed stone are the most mined materials on Earth, but these aggregates are also scarce resources associated with numerous issues in environmental sustainability.

Our example comes from the central California (CA) coast, at the beaches that form at the mouth of the San Lorenzo river in Santa Cruz, CA. The beaches in this area are potentially at an erosional tipping point due to development, storms, and rising seas. According to a recent study by Vitousek et al. (2023), "By 2100, the model estimates that 25 to 70% of California's beaches may become completely eroded due to sea-level rise scenarios of 0.5 to 3.0 m, respectively." Beaches that form at river mouths tend to have a constant or episodic sediment supply that is not well encapsulated into current forecasting models, and it is therefore not known whether such places will see such dramatic change. Monitoring sand at numerous locations should provide the requisite data to refine predictions at individual localities.

1.1.4 Data

Imagery is from the National Agriculture Imagery Program (NAIP). NAIP acquires aerial imagery at a resolution of 0.5 to 1-meter for the conterminous United States during the agricultural growing season. The program began with a 5-year cycle in 2003 then switched over to a 3-year cycle in

2009, with 2008 being a transition year. Tiles in the NAIP collection are natural color (red, green, and blue bands) or color near infra-red (red, green, blue, and near infrared bands) and may contain as much as 10 percent cloud cover per tile. See here for more details.

We use a time-series of NAIP imagery, all collected at approximately the same tide height, from summer 2005, 2010, 2014, 2016, 2018, and 2020.

1.1.5 Software dependencies

The main software dependencies are listed below:

- Keras and Tensorflow are used by the Doodleverse to implement UNets, Residual UNets (Buscombe and Goldstein, 2022), and Segformer model architectures.
- HuggingFace provides the specific Segformer architecture.
- The geemap package, which provides access to the GEE REST API.
- GDAL for orthomosaic creation and general geospatial imagery operations.
- scikit-image for carrying out morphological operations on model outputs, for rudimentary data cleaning.
- xarray for carrying out analyses on model outputs (and optionally Dask to help carry out computation tasks that don't fit in memory).

Note, this requires a Google account to access imagery (for free).

1. Sign up to use Google Earth Engine Python API. First, you need to request access to Google Earth Engine at https://signup.earthengine.google.com/. It takes about 1 day for Google to approve requests.

2. Authenticate with earthengine. Once your request has been approved, with the conda environment activated, run the following command on the terminal to link your environment to the GEE server:

   ```
   earthengine authenticate
   ```

A web browser will open, login with a gmail account and accept the terms and conditions. Then copy the authorization code into the Anaconda terminal. In the latest version of the earthengine-api, the authentication is done with `gcloud`. If an error is raised about `gcloud` missing, go to https://cloud.google.com/sdk/docs/install and install `gcloud`. After you have installed it, close the terminal and restart it, then activate the environment before running earthengine authenticate again.

1.1.6 A note on GPUs

GPUs are necessary to train ML models like those that we use here, but are not required for using those model, i.e. for inference

To make this notebook more accessible, it forces you to use your CPU, even if you have a GPU available. That's done by the line of code you'll see later `os.environ["CUDA_VISIBLE_DEVICES"] = "-1"`, but also in how we have specified the way tensorflow is installed using the provided `yml` conda environment file. If you want to use GPU for inference, you are better off using the provided conda environment in the Segmentation Gym repository, but with the additional dependencies used here (principally, `geopandas` and `xarray`)

Imports

[1]:
```
import geemap,ee
import os, json , shutil
import numpy as np
from glob import glob
from shapely.geometry import LineString, MultiPolygon, Polygon
from shapely.ops import split
from osgeo import gdal
import requests

import rioxarray
import xarray as xr
from datetime import datetime

from skimage.morphology import binary_erosion
from skimage.morphology import disk
```

ML imports: this is where we import the tools we need for image segmentation, from the Doodleverse, and from Huggingface (`transformers`)

[2]:
```
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
from doodleverse_utils.prediction_imports import *
```

```
Version:  2.12.0
Eager mode:  True
```

[3]:
```
from transformers import TFSegformerForSemanticSegmentation
from doodleverse_utils.model_imports import segformer
```

1.2 Dataset retrieval

Note that this is a bare-bones version of what the Seg2Map application does a lot better.

[4]:
```
###=================================================
############### USER INPUTS ##########################
###=================================================
#output raster spatial footprint in meters
# recent NAIP imagery comes at 0.5m. Older imagery is natively 1m and gets␣
 ↪upsampled
OUT_RES_M = 0.5

# number of columns and rows to split
# this is for situations where the ROI is larger than GEE will allow in a␣
 ↪single download
## so, we split the ROI into nx by ny regions
# in our case, we can use 2 x 2 regions
nx, ny = 2,2
```

```
#name of the site (name it anything you like)
site = 'beaches'

#name of file containing
roifile = 'ROI.geojson'

## years of data collection (list of strings)
years = ['2005','2010','2014','2016','2018','2020']

gee_collection = 'USDA/NAIP/DOQQ'
```

You only have to download and tile imagery once, so if you're running through this notebook for a second time and already have tiled imagery, you could set both of these flags to "False"

[5]:
```
download_data = retile_data = True
# download_data = retile_data = False

if download_data:
    # initialize Earth Engine
    # ee.Authenticate()
    ee.Initialize()
```

1.2.1 Download imagery

Part 1: use GEE to access image tiles

[6]:
```
if download_data:

    ## open geojson file and load into a variable
    with open(roifile) as f:
        json_data = json.load(f)

    features = json_data['features']
    ## cycle through every year
    for year in years:

        start_date = year+'-01-01' # get all imagery
        end_date = year+'-12-31' # get all imagery
        ## make the folder for the outputs
        try:
            os.mkdir(site)
        except:
            pass

        try:
            os.mkdir(site+os.sep+year)
        except:
```

```
            pass

        ## get the coordinates of the polygon, and its bounds
        coordinates = features[0]['geometry']['coordinates'][0]
        collection = ee.ImageCollection(gee_collection)
        polygon = Polygon([tuple(c) for c in coordinates])
        minx, miny, maxx, maxy = polygon.bounds

        ## split the polygon into nx x ny pieces
        dx = (maxx - minx) / nx  # width of a small part
        dy = (maxy - miny) / ny  # height of a small part
        horizontal_splitters = [LineString([(minx, miny + i*dy), (maxx, miny +␣
↪i*dy)]) for i in range(ny)]
        vertical_splitters = [LineString([(minx + i*dx, miny), (minx + i*dx,␣
↪maxy)]) for i in range(nx)]
        splitters = horizontal_splitters + vertical_splitters

        result = polygon
        for splitter in splitters:
            result = MultiPolygon(split(result, splitter))
        parts = [list(part.exterior.coords) for part in result.geoms]

        ## cycle through each piece, and download the imagery in that piece of␣
↪ROI
        counter = 1
        for part in parts:
            try:
                os.mkdir(site+os.sep+year+os.sep+'chunk'+str(counter))
            except:
                pass

            collection = ee.ImageCollection(gee_collection)

            polys = ee.Geometry.Polygon(part)

            centroid = polys.centroid()

            collection = collection.filterBounds(polys)
            collection = collection.filterDate(start_date, end_date).
↪sort('system:time_start', True)
            count = collection.size().getInfo()
            img_lst = collection.toList(1000)

            N = []
            for i in range(0, count):
                image = ee.Image(img_lst.get(i))
                name = image.get('system:index').getInfo()
```

```
                N.append(name)

            for n in N:
                image = ee.Image('USDA/NAIP/DOQQ/'+n)
                geemap.ee_export_image(image,
                                       os.getcwd()+os.sep+site+os.sep+year+os.
↪sep+'chunk'+str(counter)+os.sep+'chunk'+str(counter)+'_'+n+'_multiband.tif',
                                       scale=OUT_RES_M, region=polys,␣
↪file_per_band=False, crs="EPSG:4326")

                counter += 1
```

```
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/17fda7536056486724196175c668e57d-f3c17fe97403e35b90561c9a87be952a:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2005/chunk1/chunk1_n_3612208_ne_10_1_20050613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/6f6c6c07301d7932e631b6bc4d5429ae-e7cb9e5b81fbe3ca0c725596bce14669:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2005/chunk2/chunk2_n_3612101_nw_10_1_20050613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/b1b3c6e2434961baa19ffad772223c0e-12fc8b04641af49a5a9e8efbde1e7eb7:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2005/chunk2/chunk2_n_3612208_ne_10_1_20050613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/64513516baeb0b825755418e2a90a716-810fcd416743e19120416e5da135f82e:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2005/chunk3/chunk3_n_3612208_ne_10_1_20050613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/bd5c212fa5bafc67039e1db5a5846a86-a55f1e9b6da50991e5852f9f8e0d6595:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2005/chunk4/chunk4_n_3612101_nw_10_1_20050613_multiband.tif
```

```
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/42cf181014118e762fb910e47c88281c-64d634341590556361ed2769537699d1:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2005/chunk4/chunk4_n_3612208_ne_10_1_20050613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/481c41dbb3b24f65d35234d4b9c5822f-0a90cf538753fb77d6718d32cf44854e:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2010/chunk1/chunk1_m_3612208_ne_10_1_20100529_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/e2ce0754c9179a34b963e4164d0206b7-49bed1dd9800a855b5cd67ee64624803:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2010/chunk2/chunk2_m_3612101_nw_10_1_20100529_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/ae2ba4b690cc834962cc94d860364e9a-3890c3e4b290f1b4b08e8c7e16232a89:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2010/chunk2/chunk2_m_3612208_ne_10_1_20100529_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/9316e025e39364a5666f61c9da2e73d0-8a834fd59faf888080eb6a6e992506b7:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2010/chunk3/chunk3_m_3612208_ne_10_1_20100529_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/7d49a402be3b33a58f49f152d379fc98-1e047149a05f2433b8e4edd6a4108f1e:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2010/chunk4/chunk4_m_3612101_nw_10_1_20100529_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/b6d0b9a09fe583d7af868b70ecf69ebb-1da1c95cbbb2a0aae94d95f741fec469:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
```

```
023_epub/beaches/2010/chunk4/chunk4_m_3612208_ne_10_1_20100529_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/dbcc6c1a0e84a93e6aecb7739ae1d735-8e322ebc6ebb2d9a881dbc39964b9104:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2014/chunk1/chunk1_m_3612208_ne_10_1_20140613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/073adaf887b3d36f6ce8016e0d57dabe-092d1da1e93b71b6a9314d7281f62a75:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2014/chunk2/chunk2_m_3612101_nw_10_1_20140613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/d40c775f3ba5ffe2181b9f6a2ad53193-aec9c8af8da422d6788bfdf4dd2f63a3:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2014/chunk2/chunk2_m_3612208_ne_10_1_20140613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/e227da5bce3fa1fec2a48019001385b0-8cc7116b5eac1257df784ee492e628b6:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2014/chunk3/chunk3_m_3612208_ne_10_1_20140613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/92d9d98c487ff97efb81898a8feccd48-cc04911c951b97ddda6598aca44da366:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2014/chunk4/chunk4_m_3612101_nw_10_1_20140613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/10a2f8c63e8986b98a787e3d67a81787-8a26f125281015c4f943e4cd1cdebb35:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2014/chunk4/chunk4_m_3612208_ne_10_1_20140613_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/d087ba9be6cb47302aa95c9e5a16026f-8262f72fc4132cd570320071ae6e6953:getPixels
Please wait …
```

```
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2016/chunk1/chunk1_m_3612208_ne_10_h_20160619_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/aaaf7760f4e85b493b5c8c0c205d0d15-7e787b6b6d00f5dabf11952aa3a76f12:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2016/chunk2/chunk2_m_3612208_ne_10_h_20160619_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/e8d090994fa0278a37041005300bd841-93aab7719eaa4332d6ed2d165c2fd0a8:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2016/chunk3/chunk3_m_3612208_ne_10_h_20160619_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/b22905161f8bb4ff28d65be0d65db070-cc37d6e1f33c84193e8768c32d36f8c5:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2016/chunk4/chunk4_m_3612208_ne_10_h_20160619_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/9c895ff82aea54199ce8cc99f2c37e12-bd64f4564fb7866a5ccc7f374f66d466:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2018/chunk1/chunk1_m_3612208_ne_10_060_20180725_20190209_multib
and.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/84105bbb9c505d22de9267963fa5f840-6bd0d4019eed94b4d93fbd39ba56704e:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2018/chunk2/chunk2_m_3612208_ne_10_060_20180725_20190209_multib
and.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/0f5e0b4311ef0ca53cdb297958b3b3d8-9823b70ecdc7843134de5d9d97d3c7c2:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2018/chunk3/chunk3_m_3612208_ne_10_060_20180725_20190209_multib
and.tif
Generating URL …
```

```
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/a79ce3ac89325e62b1e61085f0571a02-8138590f6199024f0f19dd8772533bcc:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2018/chunk4/chunk4_m_3612208_ne_10_060_20180725_20190209_multib
and.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/6fa9e2167fe8395b7ed1d1d5557866ea-811e2f3fab52fc5f6fbe78a79b02c882:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2020/chunk1/chunk1_m_3612208_ne_10_060_20200527_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/5c56f7777cac7b97a38d2b4add7bf341-68d372ff05ff471ab3d5bbf69ed4869a:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2020/chunk2/chunk2_m_3612101_nw_10_060_20200527_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/1484b09836d0fdd8cca4e97d1ec8861d-4fa98cf9029e2fd6b018ce01847db8c9:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2020/chunk2/chunk2_m_3612208_ne_10_060_20200527_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/fee4eac7d3402266d64175c498f1ea22-d4e9394749e326b72e6dcb44b35869c9:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2020/chunk3/chunk3_m_3612208_ne_10_060_20200527_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/9839f506c85d5e0f2d126c44743ec2ac-f511b9a577bd8c5e411e725475e2bdeb:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
023_epub/beaches/2020/chunk4/chunk4_m_3612101_nw_10_060_20200527_multiband.tif
Generating URL …
Downloading data from
https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/thumbnail
s/7dfc07a8133060b10af7763e0b39cbdf-62394f2b477b657b5e0a8c5e928ff929:getPixels
Please wait …
Data downloaded to /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2
```

023_epub/beaches/2020/chunk4/chunk4_m_3612208_ne_10_060_20200527_multiband.tif

Part 2: Organize files copy files into a single directory per year

[7]:
```
if download_data:
    for year in years:
        # move multiband files into own directory
        outdirec = os.getcwd()+os.sep+site+os.sep+year+os.sep+'multiband'
        try:
            os.mkdir( outdirec)
        except:
            pass

        # copy each multiband image to the multiband folder for that year
        for folder in glob( os.getcwd()+os.sep+site+os.sep+year+os.
 ↪sep+'chunk*', recursive=True ):
            files = glob(folder+os.sep+'*multiband.tif')
            [shutil.copyfile(file,outdirec+os.sep+file.split(os.sep)[-1]) for␣
 ↪file in files]

        ## delete 'chunk' folders - they are no longer necessary
        for folder in glob( os.getcwd()+os.sep+site+os.sep+year+os.
 ↪sep+'chunk*', recursive=True ):
            shutil.rmtree(folder)
```

Part 3: Make merged orthomosaics Use gdal functions to mosaic the imagery back together, from 4 chunks per year, into one seamless image per year

[11]:
```
if download_data:
    for year in years:

        for folder in glob( os.getcwd()+os.sep+site+os.sep+year+os.
 ↪sep+'multiband', recursive=True ):
            print(folder)

            # ## run gdal to merge into big tiff
            vrtoptions = gdal.BuildVRTOptions(resampleAlg='average',␣
 ↪srcNodata=0, VRTNodata=0)
            files = glob(folder+os.sep+'*multiband.tif')
            print(len(files))
            outfile = os.getcwd()+os.sep+site+os.
 ↪sep+f'merged_multispectral{year}.vrt'
            ds = gdal.BuildVRT(outfile, files, options=vrtoptions)
            # ds.FlushCache()
            ds = None

            # create geotiff
```

```
            ds = gdal.Translate(outfile.replace('.vrt','.tif'),␣
↪creationOptions=["COMPRESS=LZW", "TILED=YES"], srcDS=outfile)
            ds = None

            # create jpeg and world file
            ds = gdal.Translate(outfile.replace('.vrt','.jpg'),␣
↪creationOptions=["WORLDFILE=YES", "QUALITY=100"], srcDS=outfile.replace('.
↪vrt','.tif'))
            ds = None
```

```
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/20
05/multiband
6
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/20
10/multiband
6
```

```
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk1_m_3612208_ne_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk4_m_3612101_nw_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk2_m_3612101_nw_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk3_m_3612208_ne_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
```

```
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk2_m_3612208_ne_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk4_m_3612208_ne_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk1_m_3612208_ne_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk4_m_3612101_nw_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk3_m_3612208_ne_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk4_m_3612208_ne_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk2_m_3612101_nw_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2010/multiband/chunk2_m_3612208_ne_10_1_20100529_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace
```

```
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/20
14/multiband
6
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk2_m_3612101_nw_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk4_m_3612208_ne_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk2_m_3612208_ne_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk1_m_3612208_ne_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk3_m_3612208_ne_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk4_m_3612101_nw_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
```

```
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk2_m_3612101_nw_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk4_m_3612208_ne_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk3_m_3612208_ne_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk4_m_3612101_nw_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk2_m_3612208_ne_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2014/multiband/chunk1_m_3612208_ne_10_1_20140613_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace

/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/20
16/multiband
4

Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2016/multiband/chunk1_m_3612208_ne_10_h_20160619_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
```

```
/beaches/2016/multiband/chunk3_m_3612208_ne_10_h_20160619_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2016/multiband/chunk2_m_3612208_ne_10_h_20160619_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2016/multiband/chunk4_m_3612208_ne_10_h_20160619_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2016/multiband/chunk1_m_3612208_ne_10_h_20160619_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2016/multiband/chunk3_m_3612208_ne_10_h_20160619_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2016/multiband/chunk4_m_3612208_ne_10_h_20160619_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2016/multiband/chunk2_m_3612208_ne_10_h_20160619_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace

/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/20
18/multiband
4
```

```
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2018/multiband/chunk1_m_3612208_ne_10_060_20180725_20190209_multiband.t
if: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2018/multiband/chunk4_m_3612208_ne_10_060_20180725_20190209_multiband.t
if: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2018/multiband/chunk3_m_3612208_ne_10_060_20180725_20190209_multiband.t
if: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2018/multiband/chunk2_m_3612208_ne_10_060_20180725_20190209_multiband.t
if: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2018/multiband/chunk1_m_3612208_ne_10_060_20180725_20190209_multiband.t
if: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2018/multiband/chunk4_m_3612208_ne_10_060_20180725_20190209_multiband.t
if: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2018/multiband/chunk3_m_3612208_ne_10_060_20180725_20190209_multiband.t
if: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
```

```
/beaches/2018/multiband/chunk2_m_3612208_ne_10_060_20180725_20190209_multiband.t
if: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace

/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/20
20/multiband
6

Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk2_m_3612208_ne_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk2_m_3612101_nw_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk1_m_3612208_ne_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk4_m_3612208_ne_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk3_m_3612208_ne_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
```

```
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk4_m_3612101_nw_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk2_m_3612208_ne_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk2_m_3612101_nw_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk1_m_3612208_ne_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk4_m_3612208_ne_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk3_m_3612208_ne_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: /media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub
/beaches/2020/multiband/chunk4_m_3612101_nw_10_060_20200527_multiband.tif:
TIFFReadDirectory:Sum of Photometric type-related color channels and
ExtraSamples doesn't match SamplesPerPixel. Defining non-color channels as
ExtraSamples.
Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace
```

1.3 Retile imagery

Next, we want to chop all of our images up into tiles, with overlap

The model will be pointed to each image tile, resulting in a label tile

The overlap is to oversample the imagery; by overlapping the imagery by 50% in two dimensions, the model provides predictions up to 4 times for every pixel

We mosaic the overlapping label tiles back together afterwards, to make a label map

Part 1: define some variables

[12]:
```
###### user variables
####========================
resampleAlg = 'mode' # alternatives = # 'nearest', 'max', 'min', 'average',␣
 ↪'gauss'
TARGET_SIZE = 768

OVERLAP_PX = TARGET_SIZE//2
print("Overlap size : {} px".format(OVERLAP_PX))
```

```
Overlap size : 384 px
```

Let's get a list of our image orthos and print them to screen

[13]:
```
image_orthos = glob(os.getcwd()+os.sep+site+os.sep+'*.tif')
print(image_orthos)
```

```
['/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/
merged_multispectral2005.tif', '/media/marda/TWOTB/USGS/Doodleverse/github/Mappi
ngSand_CSDMS2023_epub/beaches/merged_multispectral2010.tif', '/media/marda/TWOTB
/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/merged_multispectral
2016.tif', '/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epu
b/beaches/merged_multispectral2020.tif', '/media/marda/TWOTB/USGS/Doodleverse/gi
thub/MappingSand_CSDMS2023_epub/beaches/merged_multispectral2014.tif', '/media/m
arda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/merged_mul
tispectral2018.tif']
```

Part 2: tile imagery with overlap Below we define a function to convert a geotiff to a jpeg, with some input arguments (we want only the first 3 bands, and want "byte" output format)

[14]:
```
def gdal_translate_jpeg(f, bandList, kwargs):
    """converts a geotiff to a jpeg with world file

    Args:
        f (str): geotiff file name
        bandlist (list): list of bands to use
        kwargs (dict): other gdal input options

    Raises:
        FileNotFoundError: if model filename in 'BEST_MODEL.txt' does not
        exist online
    """
    ds = gdal.Translate(f.replace('.tif','.jpg'), f, bandList=bandList,␣
 ↪**kwargs)
    ds = None # close and save ds
```

```
### convert to jpegs for Zoo model
kwargs = {
    'format': 'JPEG',
    'outputType': gdal.GDT_Byte
}

bandList=[1,2,3]
```

[ ]:

[15]:
```
if retile_data:
    for year in years:
        image_ortho = [i for i in image_orthos if year in i][0]
        print(image_ortho)

        indir = os.path.dirname(image_ortho)
        outdir = indir+os.sep+'tiles'+str(year)

        try:
            os.mkdir(outdir)
        except:
            pass

        if os.name == "nt": ## true if windows

            try:
                cmd = 'python gdal_retile.py -r near -ot Byte -ps {} {}␣
↪-overlap {} -co "tiled=YES" -targetDir {} {}'.
↪format(TARGET_SIZE,TARGET_SIZE,OVERLAP_PX,outdir,image_ortho)
                os.system(cmd)
            except:

                from subprocess import Popen, PIPE

                process=Popen(["python","C:\\OSGeo4W64\\bin\\gdal_retile.
↪py","-r", "near", "-ot",␣
↪"Byte","-ps",str(TARGET_SIZE),str(TARGET_SIZE),"-overlap",str(OVERLAP_PX),"-co",␣
↪"tiled=YES","-targetDir",outdir, image_ortho], stdout=PIPE, stderr=PIPE)
                stdout, stderr = process.communicate()

        else: # true if linux/mac
            try:
                ## it would be cleaner if the gdal_retile.py script could be␣
↪wrapped in gdal/osgeo python, but it errored for me ...
                cmd = 'gdal_retile.py -r near -ot Byte -ps {} {} -overlap {}␣
↪-co "tiled=YES" -targetDir {} {}'.
↪format(TARGET_SIZE,TARGET_SIZE,OVERLAP_PX,outdir,image_ortho)
```

```
                os.system(cmd)
            except:
                cmd = 'python gdal_retile.py -r near -ot Byte -ps {} {}␣
↪-overlap {} -co "tiled=YES" -targetDir {} {}'.
↪format(TARGET_SIZE,TARGET_SIZE,OVERLAP_PX,outdir,image_ortho)
                os.system(cmd)

        ## get a list of the tif files
        files_to_convert = glob(outdir+os.sep+'*.tif')
        ## cycle through each tif image to convert to jpeg
        for f in files_to_convert:
            gdal_translate_jpeg(f, bandList, kwargs)

        ## delete tif files
        _ = [os.remove(k) for k in glob(outdir+os.sep+'*.tif')]
```

```
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2005.tif
0…10…20…30…40…50…60…70…80…90…100 - done.
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2010.tif
0…10…20…30…40…50…60…70…80…90…100 - done.
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2014.tif
0…10…20…30…40…50…60…70…80…90…100 - done.
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2016.tif
0…10…20…30…40…50…60…70…80…90…100 - done.
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2018.tif
0…10…20…30…40…50…60…70…80…90…100 - done.
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2020.tif
0…10…20…30…40…50…60…70…80…90…100 - done.
```

1.4 Download and prep model

Segmentation Zoo models live on Zenodo. Models are published as a set of files. Often, zenodo releases contain several sets of trained models, and are used in ensemble. Each set includes an 'h5' format file that contains model weights (trained), and a 'json' format file that contains metadata. Both are needed to make predictions

The following are a set of functions that download the 'best' model. For this simple demo, we are using just one model, but several models are also often used together (or 'ensembled') to make more stable predictions.

[17]:
```
def get_url_dict_to_download(models_json_dict: dict) -> dict:
    """Returns dictionary which contains
```

```
    paths to save downloaded files to matched with urls to download files

    each key in returned dictionary contains a full path to a file
    each value in returned dictionary contains url to download file
    ex.
    {'C:\Home\Project\file.json':"https://website/file.json"}

    Args:
        models_json_dict (dict): full path to files and links

    Returns:
        dict: full path to files and links
    """
    url_dict = {}
    for save_path, link in models_json_dict.items():
        if not os.path.isfile(save_path):
            url_dict[save_path] = link
        json_filepath = save_path.replace("_fullmodel.h5", ".json")
        if not os.path.isfile(json_filepath):
            json_link = link.replace("_fullmodel.h5", ".json")
            url_dict[json_filepath] = json_link

    return url_dict

def download_url(url, save_path, chunk_size=128):
    """downloads a file from url

    Args:
        url (str): a url to download
        save_path (str): path to file to save
        chunk_size (int): size of chunk to use when writing contents to file
    """
    r = requests.get(url, stream=True)
    with open(save_path, "wb") as fd:
        for chunk in r.iter_content(chunk_size=chunk_size):
            fd.write(chunk)

def download_BEST_model(files: list, model_direc: str) -> None:
    """downloads best model from zenodo.

    Args:
        files (list): list of available files for zenodo release
        model_direc (str): directory of model to download

    Raises:
        FileNotFoundError: if model filename in 'BEST_MODEL.txt' does not
        exist online
```

```
    """
    # dictionary to hold urls and full path to save models at
    models_json_dict = {}
    # retrieve best model text file
    best_model_json = [f for f in files if f["key"].strip() == "BEST_MODEL.
↪txt"][0]
    best_model_txt_path = os.path.join(model_direc, "BEST_MODEL.txt")
    # if BEST_MODEL.txt file not exist download it
    if not os.path.isfile(best_model_txt_path):
        download_url(
            best_model_json["links"]["self"],
            best_model_txt_path,
        )

    # read in BEST_MODEL.txt file
    with open(best_model_txt_path) as f:
        best_model_filename = f.read().strip()

    print(f"Best Model filename: {best_model_filename}")
    # check if json and h5 file in BEST_MODEL.txt exist
    model_json = [f for f in files if f["key"].strip() == best_model_filename]
    if model_json == []:
        FILE_NOT_ONLINE_ERROR = f"File {best_model_filename} not found online.␣
↪Raise an issue on Github"
        raise FileNotFoundError(FILE_NOT_ONLINE_ERROR)
    # path to save model
    outfile = os.path.join(model_direc, best_model_filename)
    # path to save file and json data associated with file saved to dict
    models_json_dict[outfile] = model_json[0]["links"]["self"]
    url_dict = get_url_dict_to_download(models_json_dict)
    # if any files are not found locally download them asynchronous
    if url_dict != {}:
        for save_path, url in url_dict.items():
            download_url(url, save_path)
```

[18]:
```
def get_weights_list(model_choice: str, weights_direc: str) -> list:
    """Returns of list of full paths to weights files(.h5) within weights_direc

    Args:
        model_choice (str): 'ENSEMBLE' or 'BEST'
        weights_direc (str): full path to directory containing model weights

    Returns:
        list: list of full paths to weights files(.h5) within weights_direc
    """
    if model_choice == "ENSEMBLE":
        return glob(weights_direc + os.sep + "*.h5")
```

```
    elif model_choice == "BEST":
        with open(weights_direc + os.sep + "BEST_MODEL.txt") as f:
            w = f.readline()
        return [weights_direc + os.sep + w[0]]

def get_config(weights_list: list) -> dict:
    """loads contents of config json files
    that have same name of h5 files in weights_list

    Args:
        weights_list (list): weight files(.h5) in weights_list

    Returns:
        dict: contents of config json files that have same name of h5 files in␣
 ↪weights_list
    """
    weights_file = weights_list[0]
    configfile = weights_file.replace(".h5", ".json").replace("weights",␣
 ↪"config").strip()
    if "fullmodel" in configfile:
        configfile = configfile.replace("_fullmodel", "").strip()
    with open(configfile.strip()) as f:
        config = json.load(f)
    return config

def request_available_files(zenodo_id: str) -> list:
    """returns list of available downloadable files for zenodo_id

    Args:
        zenodo_id (str): id of zenodo release

    Returns:
        list: list of available files downloadable for zenodo_id
    """
    # Send request to zenodo for selected model by zenodo_id
    root_url = "https://zenodo.org/api/records/" + zenodo_id
    r = requests.get(root_url)
    # get list of all files associated with zenodo id
    js = json.loads(r.text)
    files = js["files"]
    return files

def get_model_dir(parent_directory: str, dir_name: str) -> str:
    """returns full path to directory named dir_name and if it doesn't exist
    creates new directory dir_name within parent directory

    Args:
```

```
        parent_directory (str): directory to create new directory dir_name␣
↪within
        dir_name (str): name of directory to get full path to

    Returns:
        str: full path to dir_name directory
    """
    new_dir = os.path.join(parent_directory, dir_name)
    if not os.path.isdir(new_dir):
        print(f"Creating {new_dir}")
        os.mkdir(new_dir)
    return new_dir
```

Part 1: define some variables This is the name of the model we wish to use, based on the 8-class Coast Train dataset

[19]:
```
###### user variables
####========================
variable = "orthoCT_8class_segformer_7641724"

model_choice = 'BEST'
print("Model implementation choice : {}".format(model_choice))

zenodo_id = variable.split("_")[-1]
print("Zenodo ID : {}".format(zenodo_id))
```

```
Model implementation choice : BEST
Zenodo ID : 7641724
```

Part 2: download and prep the model Here is where we use all the above functions to find and download the correct files

[20]:
```
# segmentation zoo directory
parent_direc = os.path.dirname(os.getcwd())

# create downloaded models directory in segmentation_zoo/downloaded_models
downloaded_models_dir = get_models_dir = get_model_dir(parent_direc,␣
↪"downloaded_models")
print(f"Downloaded Models Located at: {downloaded_models_dir}")

# directory to hold specific downloaded model
model_direc = get_model_dir(downloaded_models_dir, variable)

# get list of available files to download for zenodo id
files = request_available_files(zenodo_id)

# download the model files
```

```
download_BEST_model(files, model_direc)
```

```
Downloaded Models Located at:
/media/marda/TWOTB/USGS/Doodleverse/github/downloaded_models
Best Model filename: ct_NAIP_8class_768_segformer_v3_fullmodel.h5
```

Here is the model weights file. It's a single file in a list, because these codes could be adapted (indeed, have been adapted elsewhere in the Doodleverse) to ingest several trained model weights and use them in ensemble prediction mode

[21]:
```
# get and show the location of the downloaded weights file
weights_files = get_weights_list(model_choice, model_direc)
print(weights_files)
```

```
['/media/marda/TWOTB/USGS/Doodleverse/github/downloaded_models/orthoCT_8class_se
gformer_7641724/ct_NAIP_8class_768_segformer_v3_fullmodel.h5']
```

Next, we define some functions that will help us apply the model and keep track of what we did (inside a dictionary called `metadatadict`)

[22]:
```
def get_metadatadict(weights_list: list, config_files: list, model_names: list)␣
 ↪-> dict:
    """returns dictionary of model weights,config_files, and model_names

    Args:
        weights_list (list): list of full paths to weights files(.h5)
        config_files (list): list of full paths to config files(.json)
        model_names (list): list of model names

    Returns:
        dict: dictionary of model weights,config_files, and model_names
    """
    metadatadict = {}
    metadatadict["model_weights"] = weights_list
    metadatadict["config_files"] = config_files
    metadatadict["model_names"] = model_names
    return metadatadict

def get_model(weights_list: list):
    """Loads models in from weights list and loads in corresponding config file
    for each model weights file(.h5) in weights_list

    Args:
        weights_list (list): full path to model weights files(.h5)

    Raises:
        Exception: raised if weights_list is empty
        Exception: An unknown model type was loaded from any of weights files in
```

```
        weights_list

    Returns:
        model, model_list, config_files, model_names
    """
    model_list = []
    config_files = []
    model_names = []
    if weights_list == []:
        raise Exception("No Model Info Passed")

    for weights in weights_list:
        # "fullmodel" is for serving on zoo they are smaller and more portable␣
↪between systems than traditional h5 files
        # gym makes a h5 file, then you use gym to make a "fullmodel" version␣
↪then zoo can read "fullmodel" version
        configfile = weights.replace(".h5", ".json").replace("weights",␣
↪"config").strip()
        if "fullmodel" in configfile:
            configfile = configfile.replace("_fullmodel", "").strip()
        with open(configfile) as f:
            config = json.load(f)
        TARGET_SIZE = config.get("TARGET_SIZE")
        MODEL = config.get("MODEL")
        NCLASSES = config.get("NCLASSES")
        N_DATA_BANDS = config.get("N_DATA_BANDS")

        try:
            # Get the selected model based on the weights file's MODEL key␣
↪provided
            # create the model with the data loaded in from the weights file
            # Load in the model from the weights which is the location of the␣
↪weights file
            model = tf.keras.models.load_model(weights)
        except BaseException:
            id2label = {}
            for k in range(NCLASSES):
                id2label[k]=str(k)
            model = segformer(id2label,num_classes=NCLASSES)

            weights=weights.strip()
            model.load_weights(weights)

        model_names.append(MODEL)
        model_list.append(model)
        config_files.append(configfile)
    return model, model_list, config_files, model_names
```

Part 3: reconstruct the model for use Now, we can make the model by reading in the config file, crerating the model architecture, then assigning our previously learned weights to the model

[23]:
```
weights = weights_files[0]

configfile = weights.replace("_fullmodel.h5", ".json")
with open(configfile) as file:
    config = json.load(file)

for k in config.keys():
    exec(k + '=config["' + k + '"]')

model, model_list, config_files, model_names = get_model(weights_files)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function
Conv._jit_compiled_convolution_op at 0x7f0895489ee0> triggered tf.function
retracing. Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has
reduce_retracing=True option that can avoid unnecessary retracing. For (3),
please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 6 calls to <function
Conv._jit_compiled_convolution_op at 0x7f089548bce0> triggered tf.function
retracing. Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has
reduce_retracing=True option that can avoid unnecessary retracing. For (3),
please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.

Some layers from the model checkpoint at nvidia/mit-b0 were not used when
initializing TFSegformerForSemanticSegmentation: ['classifier']
- This IS expected if you are initializing TFSegformerForSemanticSegmentation
from the checkpoint of a model trained on another task or with another
architecture (e.g. initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing
TFSegformerForSemanticSegmentation from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification
model from a BertForSequenceClassification model).
Some layers of TFSegformerForSemanticSegmentation were not initialized from the
model checkpoint at nvidia/mit-b0 and are newly initialized: ['decode_head']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[ ]:

1.5 Model application

Part 1: define some functions

[31]:
```
def est_label_multiclass(image,model):
    """Applies a multiclass segmentation Segformer model to an image
    Args:
        image (tensor):imagery will be resized to this size
        model (keras object): number of bands in imagery
    Returns:
        est_label: tensor of predicted model logits
    """
    est_label = model(tf.expand_dims(image, 0)).logits
    return est_label

# ============================================================
def do_seg(
    f, model, metadatadict, MODEL, sample_direc,
    NCLASSES, N_DATA_BANDS, TARGET_SIZE,
    out_dir_name='out'
):
    """Carries out image segmentation on provided file
    Args:
        f (str): file name
        model (keras object): trained and compiled keras model
        metadatadict (dict): dictionary of per-image model metadata
        MODEL (str): name of model type
        sample_direc (str): directory of samples
        NCLASSES (int): number of classes
        N_DATA_BANDS (int): number of data bands
        TARGET_SIZE (tuple): size of imagery inputs (x, y)
        out_dir_name (str): name of output folder
    """

    if f.endswith("jpg"):
        segfile = f.replace(".jpg", "_predseg.png")
    elif f.endswith("png"):
        segfile = f.replace(".png", "_predseg.png")
    elif f.endswith("npz"):
        segfile = f.replace(".npz", "_predseg.png")

    metadatadict["input_file"] = f

    # directory to hold the outputs of the models is named 'out' by default
    # create a directory to hold the outputs of the models, by default name it␣
 ↪'out' or the model name if it exists in metadatadict
```

```
    out_dir_path = os.path.normpath(sample_direc + os.sep + out_dir_name)
    if not os.path.exists(out_dir_path):
        os.mkdir(out_dir_path)

    segfile = os.path.normpath(segfile)
    segfile = segfile.replace(
        os.path.normpath(sample_direc), os.path.normpath(sample_direc + os.sep␣
↪+ out_dir_name)
    )

    metadatadict["nclasses"] = NCLASSES
    metadatadict["n_data_bands"] = N_DATA_BANDS

    ## read image from file
    image, w, h, bigimage = get_image(f,N_DATA_BANDS,TARGET_SIZE,MODEL)

    if np.std(image)==0:

        print("Image {} is empty".format(f))
        est_label = np.zeros((w,h))

    else:

        est_label = est_label_multiclass(image,model)
        # est_label cannot be float16 so convert to float32
        est_label = est_label.numpy().astype('float32')

        est_label = resize(est_label, (1, NCLASSES,␣
↪TARGET_SIZE[0],TARGET_SIZE[1]), preserve_range=True, clip=True).squeeze()
        est_label = np.transpose(est_label, (1,2,0))
        est_label = resize(est_label, (w, h))

    metadatadict["av_prob_stack"] = est_label

    softmax_scores = est_label.copy() #np.dstack((e0,e1))

    metadatadict["av_softmax_scores"] = softmax_scores

    if np.std(image)>0:
        est_label = np.argmax(softmax_scores, -1)
    else:
        est_label = est_label.astype('uint8')

    class_label_colormap = [
        "#3366CC",
        "#DC3912",
        "#FF9900",
```

```
        "#109618",
        "#990099",
        "#0099C6",
        "#DD4477",
        "#66AA00",
        "#B82E2E",
        "#316395",
        "#ffe4e1",
        "#ff7373",
        "#666666",
        "#c0c0c0",
        "#66cdaa",
        "#afeeee",
        "#0e2f44",
        "#420420",
        "#794044",
        "#3399ff",
    ]

    class_label_colormap = class_label_colormap[:NCLASSES]

    metadatadict["color_segmentation_output"] = segfile

    color_label = label_to_colors(
        est_label,
        bigimage[:, :, 0] == 0,
        alpha=128,
        colormap=class_label_colormap,
        color_class_offset=0,
        do_alpha=False,
    )

    imsave(segfile, (color_label).astype(np.uint8), check_contrast=False)

    metadatadict["color_segmentation_output"] = segfile

    segfile = segfile.replace("_predseg.png", "_res.npz")

    metadatadict["grey_label"] = est_label
    np.savez_compressed(segfile, **metadatadict)

    #### plot overlay
    segfile = segfile.replace("_res.npz", "_overlay.png")

    if N_DATA_BANDS <= 3:
        plt.imshow(bigimage, cmap='gray')
    else:
```

```
        plt.imshow(bigimage[:, :, :3])

    plt.imshow(color_label, alpha=0.5)
    plt.axis("off")
    plt.savefig(segfile, dpi=200, bbox_inches="tight")
    plt.close("all")

    #### image - overlay side by side
    segfile = segfile.replace("_res.npz", "_image_overlay.png")

    plt.subplot(121)
    if N_DATA_BANDS <= 3:
        plt.imshow(bigimage, cmap='gray')
    else:
        plt.imshow(bigimage[:, :, :3])
    plt.axis("off")

    plt.subplot(122)
    if N_DATA_BANDS <= 3:
        plt.imshow(bigimage, cmap='gray')
    else:
        plt.imshow(bigimage[:, :, :3])
    plt.imshow(color_label, alpha=0.5)
    plt.axis("off")
    plt.savefig(segfile, dpi=200, bbox_inches="tight")
    plt.close("all")


def compute_segmentation(
    TARGET_SIZE: tuple,
    N_DATA_BANDS: int,
    NCLASSES: int,
    MODEL,
    sample_direc: str,
    model_list: list,
    metadatadict: dict
) -> None:
    """applies models in model_list to directory of imagery in sample_direc.
    imagery will be resized to TARGET_SIZE and should contain number of bands␣
 ↪specified by
    N_DATA_BANDS. The outputted segmentation will contain number of classes␣
 ↪corresponding to NCLASSES.
    Outputted segmented images will be located in a new subdirectory named␣
 ↪'out' created within sample_direc.
    Args:
        TARGET_SIZE (tuple):imagery will be resized to this size
        N_DATA_BANDS (int): number of bands in imagery
```

```
        NCLASSES (int): number of classes used in segmentation model
        MODEL (str): name of mdoel
        sample_direc (str): full path to directory containing imagery to segment
        model_list (list): list of loaded models
        metadatadict (dict): config files, model weight files, and names of␣
↪each model in model_list
    """

    # Read in the image filenames as either .npz,.jpg, or .png
    files_to_segment = sort_files(sample_direc)
    sample_direc=os.path.abspath(sample_direc)
    # Compute the segmentation for each of the files

    for file_to_seg in files_to_segment:
        do_seg(
            file_to_seg,
            model_list,
            metadatadict,
            MODEL,
            sample_direc=sample_direc,
            NCLASSES=NCLASSES,
            N_DATA_BANDS=N_DATA_BANDS,
            TARGET_SIZE=TARGET_SIZE)
```

Utility functions that find and clean and move files:

[32]:
```
def sort_files(sample_direc: str) -> list:
    """returns list of sorted filenames in sample_direc

    Args:
        sample_direc (str): full path to directory of imagery/npz
        to run models on

    Returns:
        list: list of sorted filenames in sample_direc
    """
    # prepares data to be predicted
    sample_filenames = sorted(glob(sample_direc + os.sep + "*.*"))

    if sample_filenames[0].split(".")[-1] == "npz":
        sample_filenames = sorted(tf.io.gfile.glob(sample_direc + os.sep + "*.
↪npz"))
    else:
        sample_filenames = sorted(tf.io.gfile.glob(sample_direc + os.sep + "*.
↪jpg"))
        if len(sample_filenames) == 0:
            sample_filenames = sorted(glob(sample_direc + os.sep + "*.png"))
```

```
    return sample_filenames

def clean_pngs(outdir):
    """removes interim files no longer needed
    """
    # "prob.png" files ...
    _ = [os.remove(k) for k in glob(outdir+os.sep+'out'+os.sep+'*prob.png')]

    # "overlay.png" files ...
    _ = [os.remove(k) for k in glob(outdir+os.sep+'out'+os.sep+'*overlay.png')]


def label_ortho_prep(outdir):
    """prepares a set of geospatial metadata files for GDAL orthomosaic creation
    """
    ### the trick here is to make asure all the png files and xml files are
    ### in the same directoy and have the same filename root

    # Get imgs list
    imgsToMosaic = sorted(glob(os.path.join(outdir, 'out', '*.png')))

    ## copy the xml files into the 'out' folder
    xml_files = sorted(glob(os.path.join(outdir, '*.xml')))

    for k in xml_files:
        shutil.copyfile(k,k.replace(outdir,outdir+os.sep+'out'))

    ## rename pngs
    for k in imgsToMosaic:
        os.rename(k,k.replace('_predseg',''))

    xml_files = sorted(glob(os.path.join(outdir,'out', '*.xml')))
    ## rename xmls
    for k in xml_files:
        os.rename(k, k.replace('.jpg.aux.xml', '.png.aux.xml'))
```

Functions that call GDAL to carry out image mosaicing:

[33]:
```
def mosaic_ortho_label(indir, outdir, year, resampleAlg):
    """creates a color orthomosaic label raster in geotiff format
    Args:
        indir (str): input file directory
        outdir (str): output file directory
        year (str): year (to write into filename)
        resampleAlg (str): GDAL resampling type
    """
    # make some output paths
```

```
    outVRTrgb = os.path.join(indir, f'MosaicRGB{year}.vrt')
    outTIFrgb = os.path.join(indir, f'MosaicRGB{year}.tif')

    ## now we have pngs and png.xml files with the same names in the same folder
    imgsToMosaic = sorted(glob(os.path.join(outdir, 'out', '*.png')))

    # First build vrt for geotiff output
    vrt_options = gdal.BuildVRTOptions(resampleAlg=resampleAlg, srcNodata=0,␣
↪VRTNodata=0)
    ds = gdal.BuildVRT(outVRTrgb, imgsToMosaic, options=vrt_options)
    ds.FlushCache()
    ds = None

    # then build tiff
    ds = gdal.Translate(destName=outTIFrgb,␣
↪creationOptions=["NUM_THREADS=ALL_CPUS", "COMPRESS=LZW", "TILED=YES"],␣
↪srcDS=outVRTrgb)
    ds.FlushCache()
    ds = None

def mosaic_ortho_greylabel(outdir, year, resampleAlg):
    """creates a greyscale orthomosaic label raster in geotiff format
    Args:
        outdir (str): output file directory
        year (str): year (to write into filename)
        resampleAlg (str): GDAL resampling type
    """
    npzs = sorted(glob(os.path.join(outdir, 'out', '*.npz')))

    for k in npzs:
        write_greylabel_to_png(k)

    ## now we have pngs and png.xml files with the same names in the same folder
    imgsToMosaic = sorted(glob(os.path.join(outdir, 'out', '*res.png')))

    outVRT = os.path.join(indir, f'Mosaic{year}.vrt')
    outTIF = os.path.join(indir, f'Mosaic{year}.tif')

    xml_files = sorted(glob(os.path.join(outdir,'out', '*.xml')))
    ## copy and name xmls
    for k in xml_files:
        shutil.copyfile(k,k.replace('.png','_res.png'))

    # First build vrt for geotiff output
    vrt_options = gdal.BuildVRTOptions(resampleAlg=resampleAlg, srcNodata=0,␣
↪VRTNodata=0)
    ds = gdal.BuildVRT(outVRT, imgsToMosaic, options=vrt_options)
```

```
    ds.FlushCache()
    ds = None

    # then build tiff
    ds = gdal.Translate(destName=outTIF,␣
 ↪creationOptions=["NUM_THREADS=ALL_CPUS", "COMPRESS=LZW", "TILED=YES"],␣
 ↪srcDS=outVRT)
    ds.FlushCache()
    ds = None


def mosaic_ortho_greyprobs(outdir, indir, year):
    """creates a orthomosaic label probability raster in geotiff format for␣
 ↪each class
    Args:
        indir (str): input file directory
        outdir (str): output file directory
        year (str): year (to write into filename)
        resampleAlg (str): GDAL resampling type
    """
    npzs = sorted(glob(os.path.join(outdir,'out', '*res.npz')))
    for k in npzs:
        dat = write_greyprobs_to_tif(k)

    xml_files = sorted(glob(os.path.join(outdir,'out', '*res*.xml')))
    ## copy and name xmls
    for i in range(dat.shape[-1]):
        for k in xml_files:
            shutil.copyfile(k,k.replace('_res.png','_prob'+str(i)+'.tif'))

    if np.ndim(dat)==2:
        NCLASSES=1
    else:
        NCLASSES = dat.shape[-1]

    for i in range(NCLASSES):
        outVRT = os.path.join(indir, f'Mosaic{year}_Prob'+str(i)+'.vrt')
        outTIF = os.path.join(indir, f'Mosaic{year}_Prob'+str(i)+'.tif')

        ## now we have pngs and png.xml files with the same names in the same␣
 ↪folder
        imgsToMosaic = sorted(glob(os.path.join(outdir, 'out', '*prob'+str(i)+'.
 ↪tif')))

        # First build vrt for geotiff output
        vrt_options = gdal.BuildVRTOptions(resampleAlg="lanczos",srcNodata=0,␣
 ↪VRTNodata=0)
```

```
        ds = gdal.BuildVRT(outVRT, imgsToMosaic, options=vrt_options)
        ds.FlushCache()
        ds = None

        # then build tiff
        ds = gdal.Translate(destName=outTIF,␣
↪creationOptions=["NUM_THREADS=ALL_CPUS", "COMPRESS=LZW", "TILED=YES"],␣
↪srcDS=outVRT)
        ds.FlushCache()
        ds = None
```

[ ]:

[34]:
```
def write_greyprobs_to_tif(k):
    """reads contents of an npz (model output archive) and creates a greyscale␣
↪floating point tiff of the model softmax scores
    Args:
        k (str): npz filename
    """
    with np.load(k) as data:
        dat = tf.nn.softmax(data['av_softmax_scores']).numpy().astype('float32')
    if np.ndim(dat)==2:
        NCLASSES=1
    else:
        NCLASSES = dat.shape[-1]
    for i in range(NCLASSES):
        if NCLASSES>1:
            imsave(k.replace('res.npz','prob'+str(i)+'.tif'), dat[:,:,i],␣
↪check_contrast=False, compression=0)
        else:
            imsave(k.replace('res.npz','prob'+str(i)+'.tif'), dat,␣
↪check_contrast=False, compression=0)

    return dat

def write_greylabel_to_png(k):
    """reads contents of an npz (model output archive) and creates a greyscale␣
↪integer png file of the model label estimates
    Args:
        k (str): npz filename
    """
    with np.load(k) as data:
        dat = 1+np.round(data['grey_label'].astype('uint8'))
    imsave(k.replace('.npz','.png'), dat, check_contrast=False, compression=0)
```

Here we get a list of model metadata and parse some necessary inputs from the config file, in order to use the model

[35]:
```
# get dictionary containing all files needed to run models on data
metadatadict = get_metadatadict(weights_files, config_files, model_names)

# read contents of config file into dictionary
config = get_config(weights_files)
TARGET_SIZE = config.get("TARGET_SIZE")
NCLASSES = config.get("NCLASSES")
N_DATA_BANDS = config.get("N_DATA_BANDS")
```

Part 2: apply model to each image, and mosaic outputs

[36]:
```
for year in years:

    ######################################
    # STEP 1: define inputs and outputs
    image_ortho = [i for i in image_orthos if year in i][0]
    print(image_ortho)
    indir = os.path.dirname(image_ortho)
    outdir = indir+os.sep+'tiles'+str(year)
    sample_direc = outdir

    ######################################
    # STEP 2: read images
    sample_filenames = sort_files(sample_direc)
    print("Number of samples: %i" % (len(sample_filenames)))

    ######################################
    # STEP 3: apply model
    compute_segmentation(
        TARGET_SIZE,
        N_DATA_BANDS,
        NCLASSES,
        MODEL,
        sample_direc,
        model_list[0],
        metadatadict)

    ##################################
    #### STEP 4: STITCH ORTHO LABEL TILES

    ## now, you have the 'out' folder ...
    # the out folder may contain a bunch of crap we dont want. let's delete␣
↪those files
    # this should do nothing if mode='meta' or mode='minimal'
    clean_pngs(outdir)

    ##############################################
```

```
    ################ LABEL ORTHO CREATION PREP
    label_ortho_prep(outdir)

    ################################################
    ################ LABEL ORTHO CREATION
    ### let's stitch the label "predseg" pngs!
    mosaic_ortho_label(indir, outdir, year, resampleAlg)

    ###################################
    ##### STEP 5: MAKE AND STITCH ORTHO GREYSCALE LABEL TILES
    mosaic_ortho_greylabel(outdir, year, resampleAlg)

    ###################################
    ##### STEP 6: MAKE AND STITCH ORTHO GREYSCALE probability TILES
    mosaic_ortho_greyprobs(outdir, indir, year)
```

```
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2005.tif
Number of samples: 52
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2010.tif
Number of samples: 52
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2014.tif
Number of samples: 52
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2016.tif
Number of samples: 52
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2018.tif
Number of samples: 52
/media/marda/TWOTB/USGS/Doodleverse/github/MappingSand_CSDMS2023_epub/beaches/me
rged_multispectral2020.tif
Number of samples: 52
```

1.6 Mapping Sand

The commented code below may be uncommented to use a small Dask cluster to carry out these analyses. Dask and xarray work well together, allowing you to perform computations that you wouldn't ordinarily have computing resources to successfully perform.

Here, we're using a relatively small dataset that may easily fit in available memory, so this Dask cluster may not be required

[39]:
```
# from dask.distributed import Client

# n_workers = 8
```

```
# threads_per_worker = 2
# memory_limit='10GB'

# ## start client
# client = Client(n_workers=n_workers, threads_per_worker=threads_per_worker,␣
 ↪memory_limit=memory_limit)
```

Part 1: define some variables these are variables needed by xarray - we'll use floating point precision, and allow xarray to automatically create chunks of our data

[40]:
```
################################################################
#################### user inputs
dtype = 'float64'
chunksize = ("auto", "auto")
```

Make a list of output and input files, and a time array

[41]:
```
mosaic_files = [f'{site}/Mosaic{year}.tif' for year in years]
mosaic_files = sorted(mosaic_files)
print(mosaic_files)

image_files = [f'{site}/merged_multispectral{year}.tif' for year in years]
image_files = sorted(image_files)
print(image_files)

# Create variable used for time axis
time_var = xr.Variable('time',years)
print(time_var)
```

```
['beaches/Mosaic2005.tif', 'beaches/Mosaic2010.tif', 'beaches/Mosaic2014.tif',
'beaches/Mosaic2016.tif', 'beaches/Mosaic2018.tif', 'beaches/Mosaic2020.tif']
['beaches/merged_multispectral2005.tif', 'beaches/merged_multispectral2010.tif',
'beaches/merged_multispectral2014.tif', 'beaches/merged_multispectral2016.tif',
'beaches/merged_multispectral2018.tif', 'beaches/merged_multispectral2020.tif']
<xarray.Variable (time: 6)>
array(['2005', '2010', '2014', '2016', '2018', '2020'], dtype='<U4')
```

Part 2: load geotiffs into xarray

[65]:
```
# Load in and concatenate all individual GeoTIFFs
geotiffs_da = xr.concat([rioxarray.open_rasterio(i, chunks=chunksize,␣
 ↪dtype=dtype) for i in mosaic_files],
                        dim=time_var)
# Covert our xarray.DataArray into a xarray.Dataset
geotiffs_ds = geotiffs_da.to_dataset('band')

# Rename the variable to a more useful name
geotiffs_ds = geotiffs_ds.rename({1: 'label'})
```

```
# display dimensions and variables on screen
geotiffs_ds
```

[65]:
```
<xarray.Dataset>
Dimensions:      (time: 6, y: 1635, x: 5165)
Coordinates:
  * x            (x) float64 -122.0 -122.0 -122.0 … -122.0 -122.0 -122.0
  * y            (y) float64 36.97 36.97 36.97 36.97 … 36.96 36.96 36.96 36.96
    spatial_ref  int64 0
  * time         (time) <U4 '2005' '2010' '2014' '2016' '2018' '2020'
Data variables:
    label        (time, y, x) float32 dask.array<chunksize=(1, 1635, 5165),
meta=np.ndarray>
Attributes:
    AREA_OR_POINT:  Area
    _FillValue:     0.0
    scale_factor:   1.0
    add_offset:     0.0
```

Do the same for the image orthomosaics, i.e. model inputs

[43]:
```
# Load in and concatenate all individual GeoTIFFs
im_geotiffs_da = xr.concat([rioxarray.open_rasterio(i, chunks=chunksize,␣
 ↪dtype=dtype) for i in image_files],
                           dim=time_var)
# Covert our xarray.DataArray into a xarray.Dataset
im_geotiffs_ds = im_geotiffs_da.to_dataset('band')

# Rename the variable to a more useful name
im_geotiffs_ds = im_geotiffs_ds.rename({1: 'red'})
im_geotiffs_ds = im_geotiffs_ds.rename({2: 'green'})
im_geotiffs_ds = im_geotiffs_ds.rename({3: 'blue'})
im_geotiffs_ds = im_geotiffs_ds.drop_vars(4)

im_geotiffs_ds
```

[43]:
```
<xarray.Dataset>
Dimensions:      (time: 6, y: 1635, x: 5165)
Coordinates:
  * x            (x) float64 -122.0 -122.0 -122.0 … -122.0 -122.0 -122.0
  * y            (y) float64 36.97 36.97 36.97 36.97 … 36.96 36.96 36.96 36.96
    spatial_ref  int64 0
  * time         (time) <U4 '2005' '2010' '2014' '2016' '2018' '2020'
Data variables:
    red          (time, y, x) float64 dask.array<chunksize=(1, 1635, 5165),
meta=np.ndarray>
```

```
    green        (time, y, x) float64 dask.array<chunksize=(1, 1635, 5165),
meta=np.ndarray>
    blue         (time, y, x) float64 dask.array<chunksize=(1, 1635, 5165),
meta=np.ndarray>
Attributes:
    AREA_OR_POINT:  Area
    _FillValue:     0
    scale_factor:   1.0
    add_offset:     0.0
```

Part 3: show model outputs Create a figure that shows each year's image, with the mosaiced model output on top as a semi-transparent color overlay

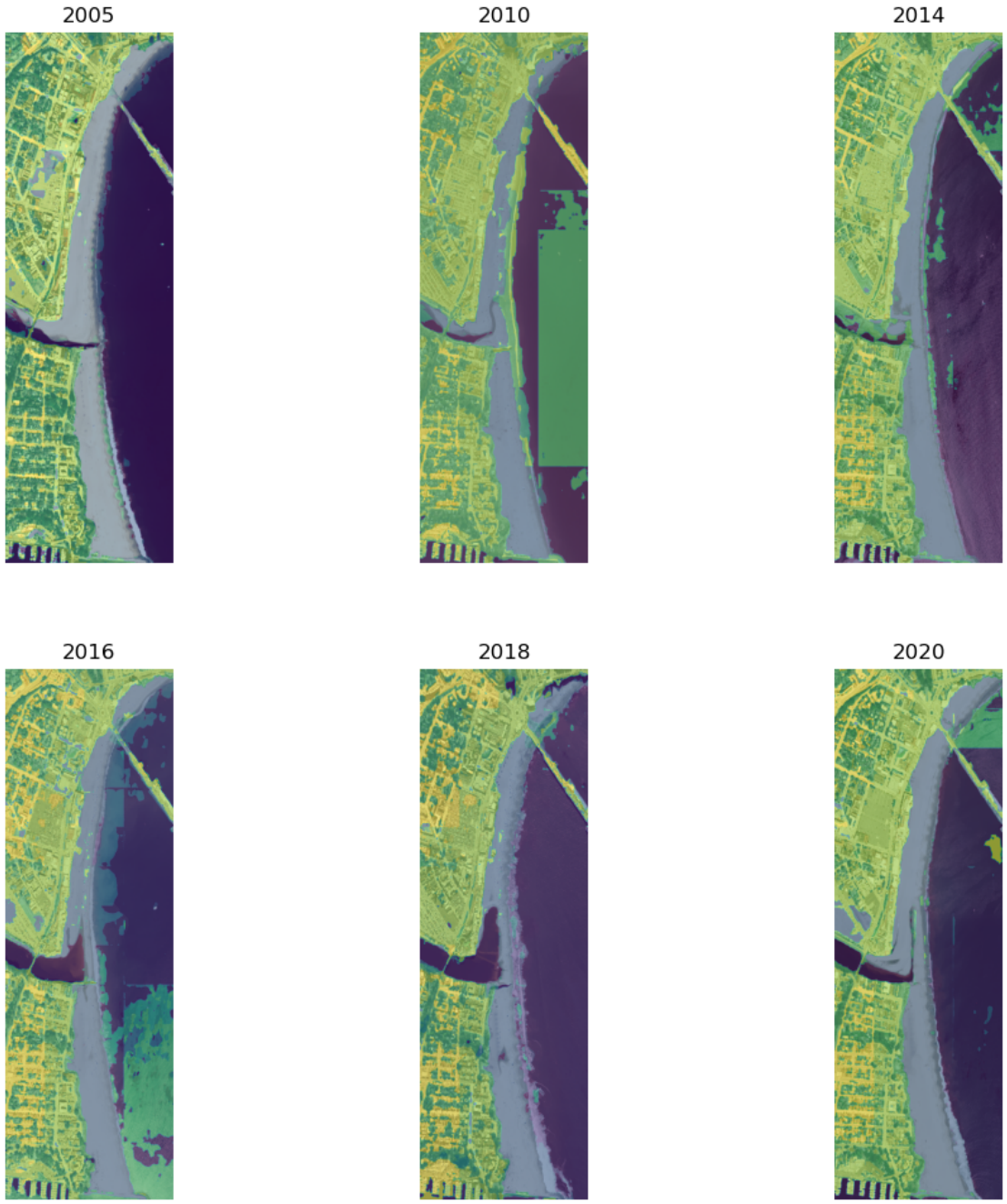
First, rearrange the image xarray so it can be plotted "channels last", like matplotlib expects

[44]:
```
im_show = xr.
 ↪concat([im_geotiffs_ds['red'],im_geotiffs_ds['green'],im_geotiffs_ds['blue']],dim=('x','x',
```

[45]:
```
%matplotlib inline
f, axs = plt.subplots(2,3, figsize=(12,12))

axs[0][0].imshow(im_show.sel(time='2005').squeeze().transpose()/255.)
axs[0][0].imshow(geotiffs_ds.sel(time='2005').to_array().squeeze().transpose(),␣
 ↪alpha=0.5)
axs[0][0].axis('off'); axs[0][0].set_title('2005')

axs[0][1].imshow(im_show.sel(time='2010').squeeze().transpose()/255.)
axs[0][1].imshow(geotiffs_ds.sel(time='2010').to_array().squeeze().transpose(),␣
 ↪alpha=0.5)
axs[0][1].axis('off'); axs[0][1].set_title('2010')

axs[0][2].imshow(im_show.sel(time='2014').squeeze().transpose()/255.)
axs[0][2].imshow(geotiffs_ds.sel(time='2014').to_array().squeeze().transpose(),␣
 ↪alpha=0.5)
axs[0][2].axis('off'); axs[0][2].set_title('2014')

axs[1][0].imshow(im_show.sel(time='2016').squeeze().transpose()/255.)
axs[1][0].imshow(geotiffs_ds.sel(time='2016').to_array().squeeze().transpose(),␣
 ↪alpha=0.5)
axs[1][0].axis('off'); axs[1][0].set_title('2016')

axs[1][1].imshow(im_show.sel(time='2018').squeeze().transpose()/255.)
axs[1][1].imshow(geotiffs_ds.sel(time='2018').to_array().squeeze().transpose(),␣
 ↪alpha=0.5)
axs[1][1].axis('off'); axs[1][1].set_title('2018')

axs[1][2].imshow(im_show.sel(time='2020').squeeze().transpose()/255.)
```

```
axs[1][2].imshow(geotiffs_ds.sel(time='2020').to_array().squeeze().transpose(),␣
↪alpha=0.5)
axs[1][2].axis('off'); axs[1][2].set_title('2020')
```

[45]: Text(0.5, 1.0, '2020')



As you can hopefully see, the model does a reasonably good job at identiying the sand, water,

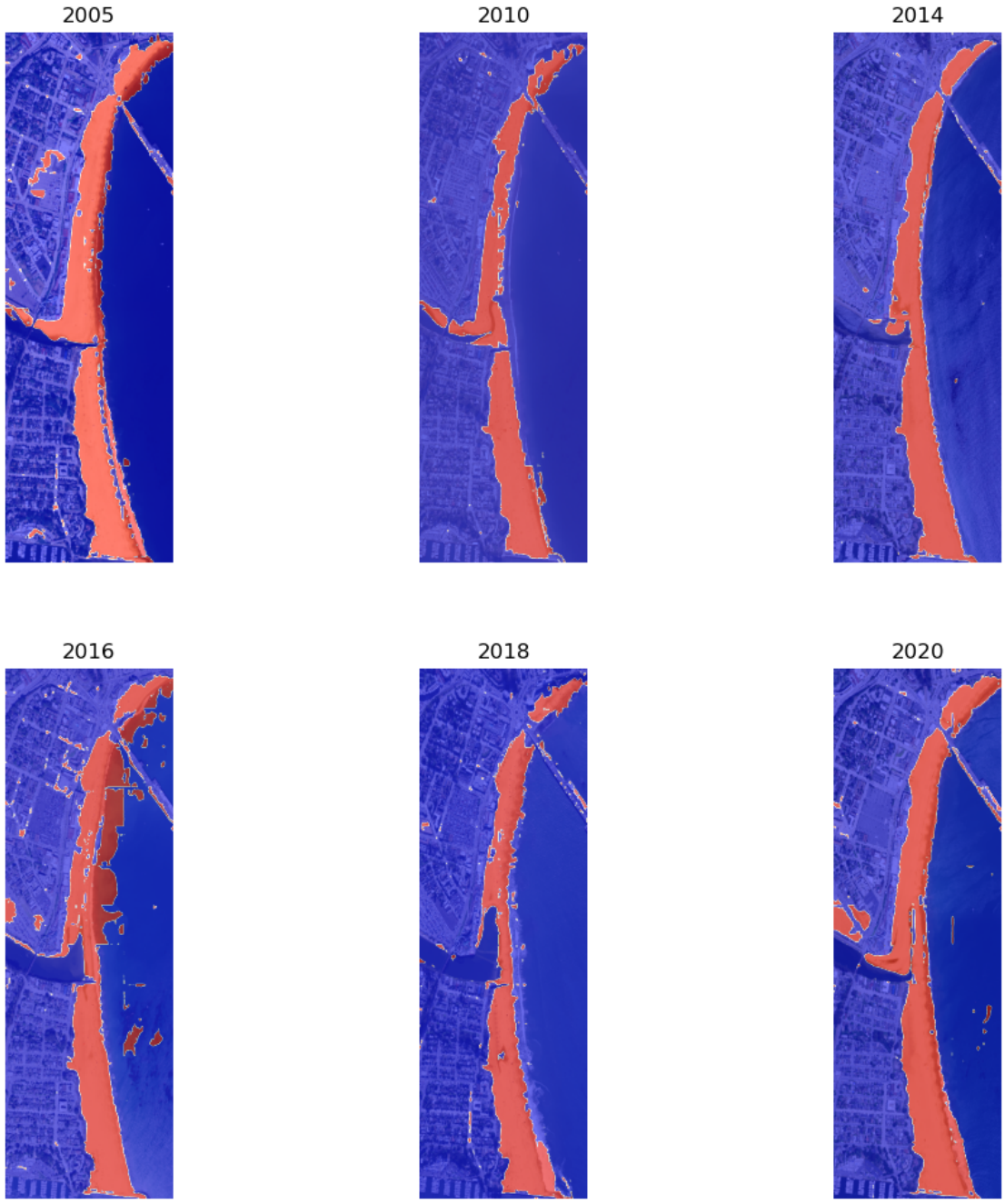
buildings, etc, but is not perfect

Part 4: show sand maps Create a figure that shows each year's image, with just the sand output on top as a semi-transparent color overlay

The model does quite well at detecting sand. 2016 is the noisiest prediction

[46]:
```
%matplotlib inline
f, axs = plt.subplots(2,3, figsize=(12,12))

axs[0][0].imshow(im_show.sel(time='2005').squeeze().transpose()/255.)
axs[0][0].imshow(geotiffs_ds.sel(time='2005').to_array().squeeze().
 ↪transpose()==3, alpha=0.5, cmap='bwr')
axs[0][0].axis('off'); axs[0][0].set_title('2005')

axs[0][1].imshow(im_show.sel(time='2010').squeeze().transpose()/255.)
axs[0][1].imshow(geotiffs_ds.sel(time='2010').to_array().squeeze().
 ↪transpose()==3, alpha=0.5, cmap='bwr')
axs[0][1].axis('off'); axs[0][1].set_title('2010')

axs[0][2].imshow(im_show.sel(time='2014').squeeze().transpose()/255.)
axs[0][2].imshow(geotiffs_ds.sel(time='2014').to_array().squeeze().
 ↪transpose()==3, alpha=0.5, cmap='bwr')
axs[0][2].axis('off'); axs[0][2].set_title('2014')

axs[1][0].imshow(im_show.sel(time='2016').squeeze().transpose()/255.)
axs[1][0].imshow(geotiffs_ds.sel(time='2016').to_array().squeeze().
 ↪transpose()==3, alpha=0.5, cmap='bwr')
axs[1][0].axis('off'); axs[1][0].set_title('2016')

axs[1][1].imshow(im_show.sel(time='2018').squeeze().transpose()/255.)
axs[1][1].imshow(geotiffs_ds.sel(time='2018').to_array().squeeze().
 ↪transpose()==3, alpha=0.5, cmap='bwr')
axs[1][1].axis('off'); axs[1][1].set_title('2018')

axs[1][2].imshow(im_show.sel(time='2020').squeeze().transpose()/255.)
axs[1][2].imshow(geotiffs_ds.sel(time='2020').to_array().squeeze().
 ↪transpose()==3, alpha=0.5, cmap='bwr')
axs[1][2].axis('off'); axs[1][2].set_title('2020')
```

[46]:
```
Text(0.5, 1.0, '2020')
```



Part 5: compute and show some time-averaged variables By way of some initial data exploration, next we use xarray to compute time-averaged imagery and time-averaged labels

We also create a raster of the standard deviation to look at the per-pixel variability in model outputs. Depending on location, this could encode signal or noise. For example, the locations of buildings should be unchanged and that variability should be low, whereas variability near the shoreline and in the rivermouth is natural. Our maps seem to capture this broad trend, except the

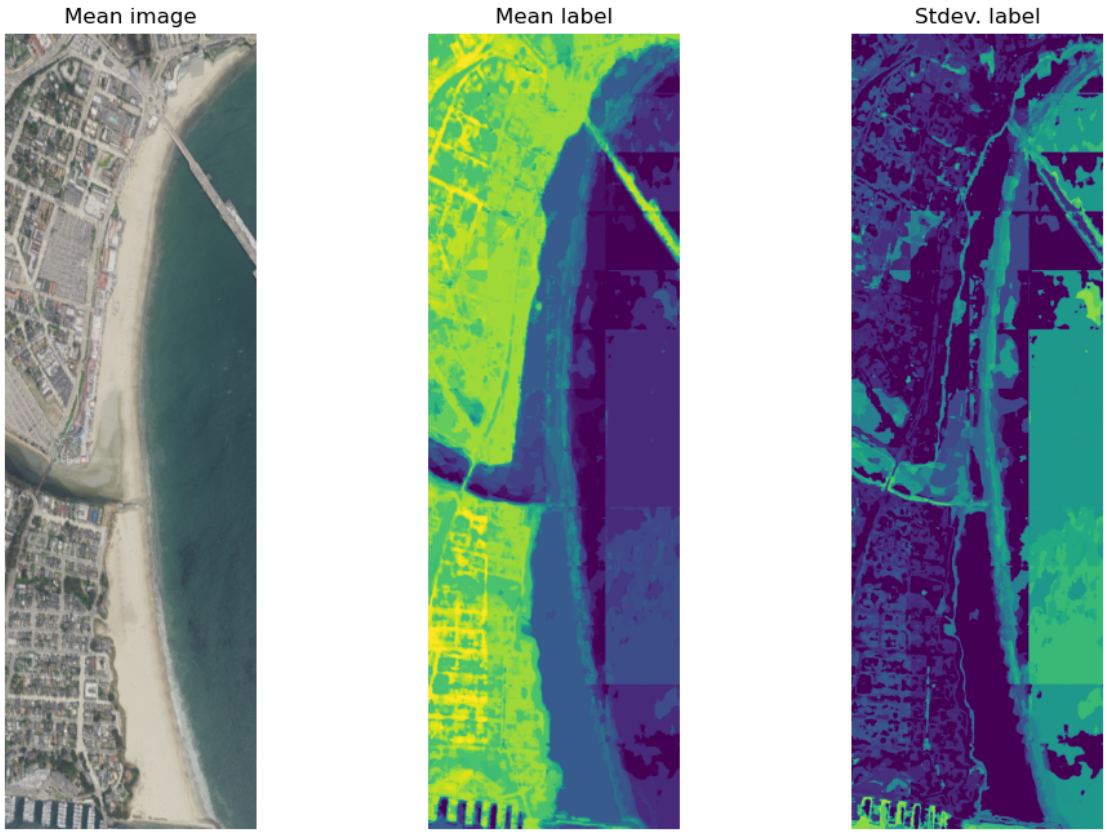
model sometimes struggles to detect open water (there are other models in the Doodleverse that may deal better with this class).

[47]:
```
mean_image = im_geotiffs_ds.mean("time", skipna=True)
```

[48]:
```
stdev_label = geotiffs_ds.std("time", skipna=True)
stdev_label.rio.to_raster(raster_path=f"{site}/label_time_stdev.tif",␣
 ↪dtype=dtype)

mean_label = geotiffs_ds.mean("time", skipna=True)
mean_label.rio.to_raster(raster_path=f"{site}/label_time_mean.tif", dtype=dtype)
```

[49]:
```
%matplotlib inline
f, axs = plt.subplots(1, 3, figsize=(12,8))

axs[0].imshow(mean_image.to_array().squeeze().transpose()/255.)
axs[0].axis('off'); axs[0].set_title('Mean image')

axs[1].imshow(mean_label.to_array().squeeze().transpose())
axs[1].axis('off'); axs[1].set_title('Mean label')

axs[2].imshow(stdev_label.to_array().squeeze().transpose())
axs[2].axis('off'); axs[2].set_title('Stdev. label')
```

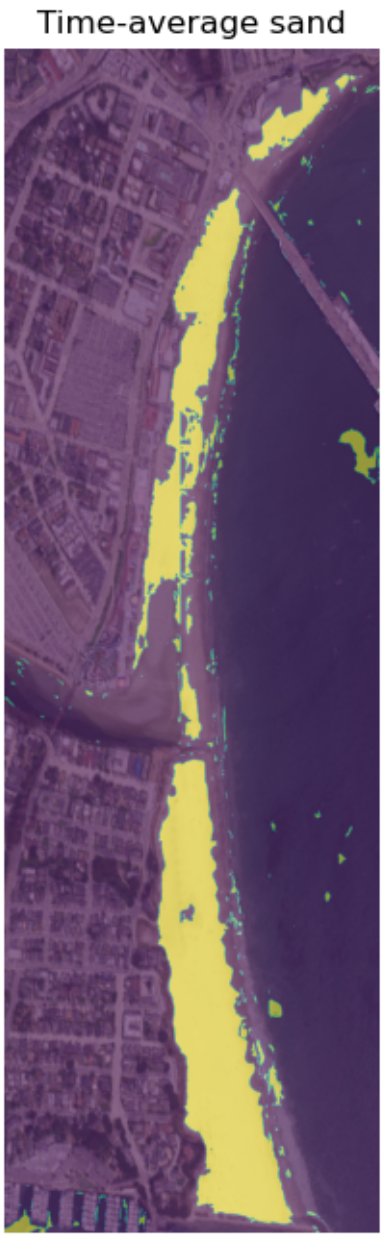
[49]:
```
Text(0.5, 1.0, 'Stdev. label')
```



Next we'll make a quick plot of the time-averaged sand

[50]:
```
%matplotlib inline
f, axs = plt.subplots(1, 1, figsize=(12,8))

axs.imshow(mean_image.to_array().squeeze().transpose()/255.)
axs.imshow(mean_label.to_array().squeeze().transpose()==3, alpha=0.5)
axs.axis('off'); axs.set_title('Time-average sand')
```

[50]:
```
Text(0.5, 1.0, 'Time-average sand')
```



Part 6: create sand maps from the per-class probabilities Next we'll carry out a similar workflow, this time using thresholding applied to the raw model output probabilities. Sediment (sand) is class 2 in the model (counting up from zero), so we read the "prob2" geotiffs into an xarray. These probability rasters were generated by applying the model - you'll see them in your folder of outputs for each year

[51]:
```
mosaic_files = [f'{site}/Mosaic{year}_Prob2.tif' for year in years]
mosaic_files = sorted(mosaic_files)
print(mosaic_files)
```

```
# Load in and concatenate all individual GeoTIFFs
geotiffs_da = xr.concat([rioxarray.open_rasterio(i, chunks=chunksize,␣
 ↪dtype=dtype) for i in mosaic_files],
                        dim=time_var)
# Covert our xarray.DataArray into a xarray.Dataset
geotiffs_ds = geotiffs_da.to_dataset('band')

# Rename the variable to a more useful name
geotiffs_ds = geotiffs_ds.rename({1: 'sand_probs'})

geotiffs_ds
```

```
['beaches/Mosaic2005_Prob2.tif', 'beaches/Mosaic2010_Prob2.tif',
'beaches/Mosaic2014_Prob2.tif', 'beaches/Mosaic2016_Prob2.tif',
'beaches/Mosaic2018_Prob2.tif', 'beaches/Mosaic2020_Prob2.tif']
```

[51]:
```
<xarray.Dataset>
Dimensions:      (time: 6, y: 1635, x: 5165)
Coordinates:
  * x            (x) float64 -122.0 -122.0 -122.0 … -122.0 -122.0 -122.0
  * y            (y) float64 36.97 36.97 36.97 36.97 … 36.96 36.96 36.96 36.96
    spatial_ref  int64 0
  * time         (time) <U4 '2005' '2010' '2014' '2016' '2018' '2020'
Data variables:
    sand_probs   (time, y, x) float32 dask.array<chunksize=(1, 1635, 5165),
meta=np.ndarray>
Attributes:
    AREA_OR_POINT:  Area
    _FillValue:     0.0
    scale_factor:   1.0
    add_offset:     0.0
```

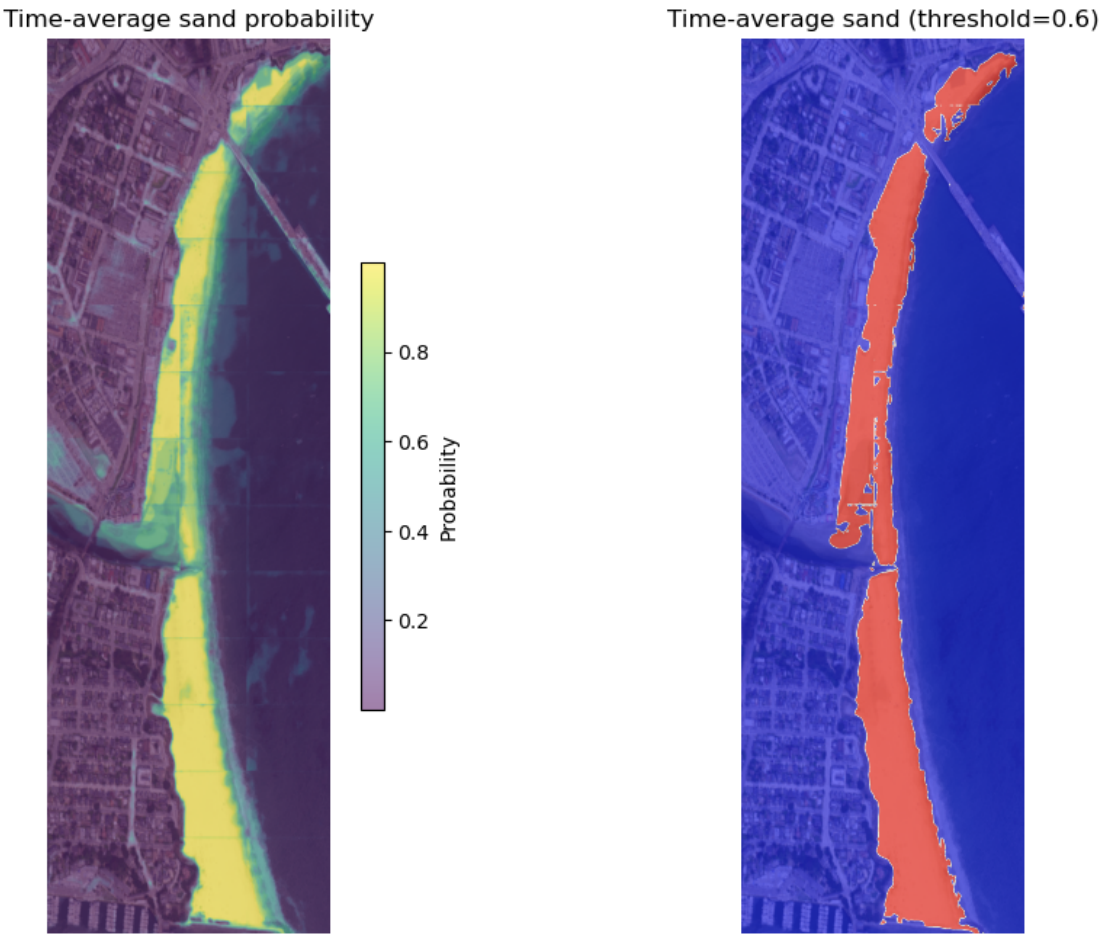
Next we'll compute and plot the time-averaged probabilities and play with a threshold to see what threshold creates the best looking outputs

[52]:
```
mean_sand_prob = geotiffs_ds.mean("time", skipna=True)
mean_sand_prob.rio.to_raster(raster_path=f"{site}/sand_prob_time_mean.tif",␣
 ↪dtype=dtype)
```

[53]:
```
%matplotlib inline
f, axs = plt.subplots(1, 2, figsize=(12,8))

axs[0].imshow(mean_image.to_array().squeeze().transpose()/255.)
im = axs[0].imshow(mean_sand_prob.to_array().squeeze().transpose(), alpha=0.5)
axs[0].axis('off'); axs[0].set_title('Time-average sand probability')
cb = plt.colorbar(im, shrink=0.5)
```

```
cb.set_label('Probability')

axs[1].imshow(mean_image.to_array().squeeze().transpose()/255.)
im = axs[1].imshow(mean_sand_prob.to_array().squeeze().transpose()>.6, alpha=0.
↪5, cmap='bwr')
axs[1].axis('off'); axs[1].set_title('Time-average sand (threshold=0.6)')
```

[53]:
```
Text(0.5, 1.0, 'Time-average sand (threshold=0.6)')
```

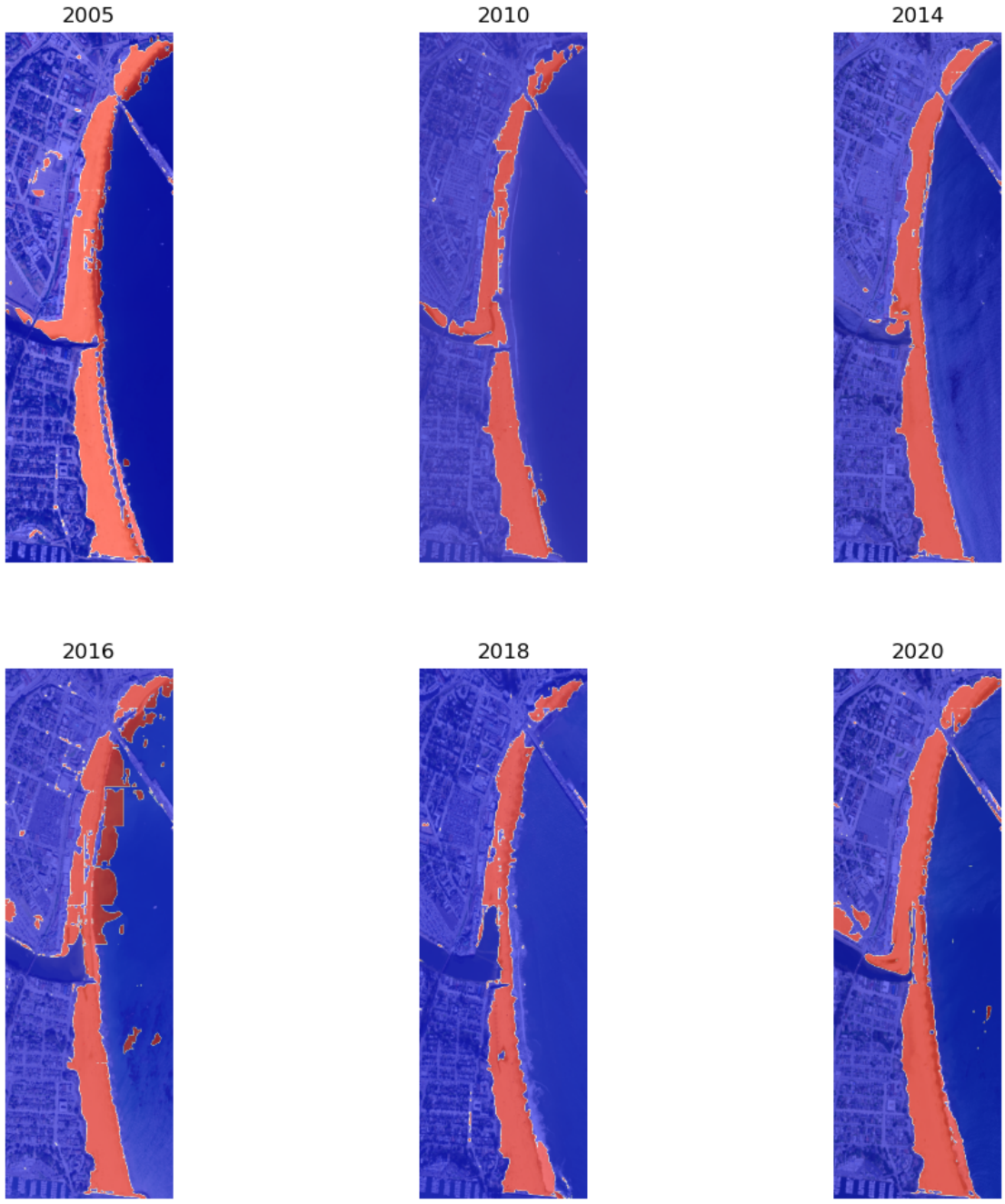


Now we have a threshold we like, let's apply it to all 6 images in our time-series and create a sand/no-sand plot

[54]:
```
%matplotlib inline
f, axs = plt.subplots(2,3, figsize=(12,12))

axs[0][0].imshow(im_show.sel(time='2005').squeeze().transpose()/255.)
axs[0][0].imshow(geotiffs_ds.sel(time='2005').to_array().squeeze().transpose()>.
↪6, alpha=0.5, cmap='bwr')
```

```
axs[0][0].axis('off'); axs[0][0].set_title('2005')

axs[0][1].imshow(im_show.sel(time='2010').squeeze().transpose()/255.)
axs[0][1].imshow(geotiffs_ds.sel(time='2010').to_array().squeeze().transpose()>.
 ↪6, alpha=0.5, cmap='bwr')
axs[0][1].axis('off'); axs[0][1].set_title('2010')

axs[0][2].imshow(im_show.sel(time='2014').squeeze().transpose()/255.)
axs[0][2].imshow(geotiffs_ds.sel(time='2014').to_array().squeeze().transpose()>.
 ↪6, alpha=0.5, cmap='bwr')
axs[0][2].axis('off'); axs[0][2].set_title('2014')

axs[1][0].imshow(im_show.sel(time='2016').squeeze().transpose()/255.)
axs[1][0].imshow(geotiffs_ds.sel(time='2016').to_array().squeeze().transpose()>.
 ↪6, alpha=0.5, cmap='bwr')
axs[1][0].axis('off'); axs[1][0].set_title('2016')

axs[1][1].imshow(im_show.sel(time='2018').squeeze().transpose()/255.)
axs[1][1].imshow(geotiffs_ds.sel(time='2018').to_array().squeeze().transpose()>.
 ↪6, alpha=0.5, cmap='bwr')
axs[1][1].axis('off'); axs[1][1].set_title('2018')

axs[1][2].imshow(im_show.sel(time='2020').squeeze().transpose()/255.)
axs[1][2].imshow(geotiffs_ds.sel(time='2020').to_array().squeeze().transpose()>.
 ↪6, alpha=0.5, cmap='bwr')
axs[1][2].axis('off'); axs[1][2].set_title('2020')
```

[54]: Text(0.5, 1.0, '2020')



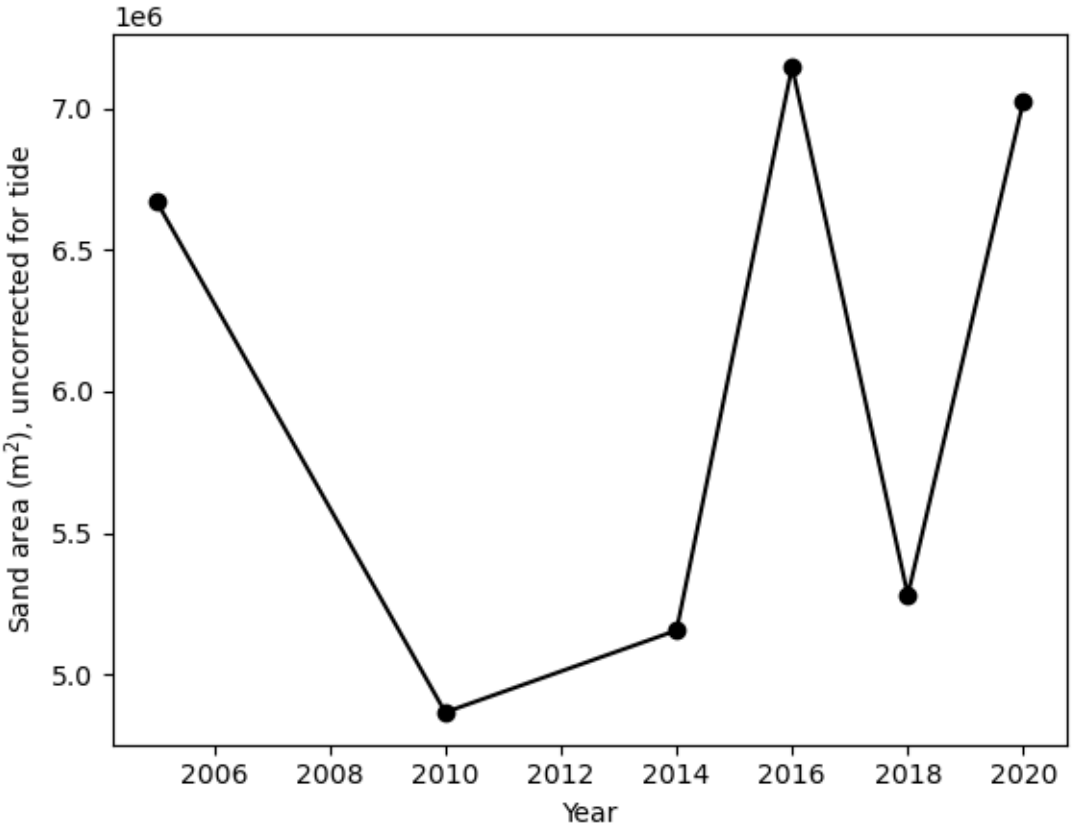
Part 6: compute and show some time-series of mapped sand Within error, the model seems capable of detecting sand with sufficient accuracy that time-series may be reliably obtained. These images have been captured close to the same state of the tide, however in reality some correction for tidal excursion might be necesaary on the imagery, along with other corrections when waves are large (Vitousek et al., 2023). We can compute sand area by summing the sand pixels and converting to square meters. The resulting data shows no longer-term trend, but individual peaks and troughs could indicate cycles of deposition and erosion, respectively, and might warrant

further analysis.

[55]:
```
px2sqm = (1/OUT_RES_M)**2
print(px2sqm)

sums = np.array([float(geotiffs_ds.sand_probs.sel(time=year).sum().compute())␣
 ↪for year in years])
dt = [datetime.strptime(year,'%Y') for year in years]

plt.plot(dt, sums*px2sqm, 'k-o')
plt.ylabel(r'Sand area (m$^2$), uncorrected for tide')
plt.xlabel('Year')
```

```
4.0
```

[55]:
```
Text(0.5, 0, 'Year')
```



Part 7: compute and show some time-series of shoreline locations Next we'll see if we can detect the shoreline and map the shoreline location, to see if that has changed over time. To do that we define a function that takes a binary sand mask, and attempts to compute the seaward edge of that mask

[56]:
```
def nan_indexer(y):
    """finds and indexes nans

    Args:
        y (array): an array that may contain nans

    Returns:
        array indicating location of nans
        array without nans
    """
    return np.isnan(y), lambda z: z.nonzero()[0]

def get_shoreline(im):
    """a rudimentary shoreline finding algorithm. The binary mask, im, is
    read in line by line, and the first nonzero location is returned as the␣
 ↪shoreline location
    No location is returned if there is no nonzero element in that row, or if
    there are additional potential shoreline locations nearby
    Finally, a rudimentary iterative cleaning algorithm is applied to the data,
    removing values outside 2 standard deviations of the mean
    Missing values are interpolated over
    Args:
        im (2d array): binary mask showing sand locations

    Returns:
        s: shoreline location in pixels
    """
    s=[]
    for k in range(im.shape[0]):
        tmp = np.where(im[k,:]>0)[0]
        if len(tmp)<1:
            tmp = np.nan
        else:
            tmp = tmp[-1]
            if np.any(np.gradient(tmp))>5:
                tmp = np.nan
        s.append(tmp)

    ## basic iterative outlier detection
    s = np.array(s)
    for k in range(5):
        s[s>(np.nanmean(s)+2*np.nanstd(s))] = np.nan
        s[s<(np.nanmean(s)-2*np.nanstd(s))] = np.nan

        nans, x= nan_indexer(s)
        s[nans]= np.interp(x(nans), x(~nans), s[~nans])
    return s
```

Next, we'll find the shoreline contour for each image. We start with a probability map, that is binarized (converted into 0s and 1s)

apply a morphological operation to each mask to remove the small "sand islands" (blobs of 1s in the mask)

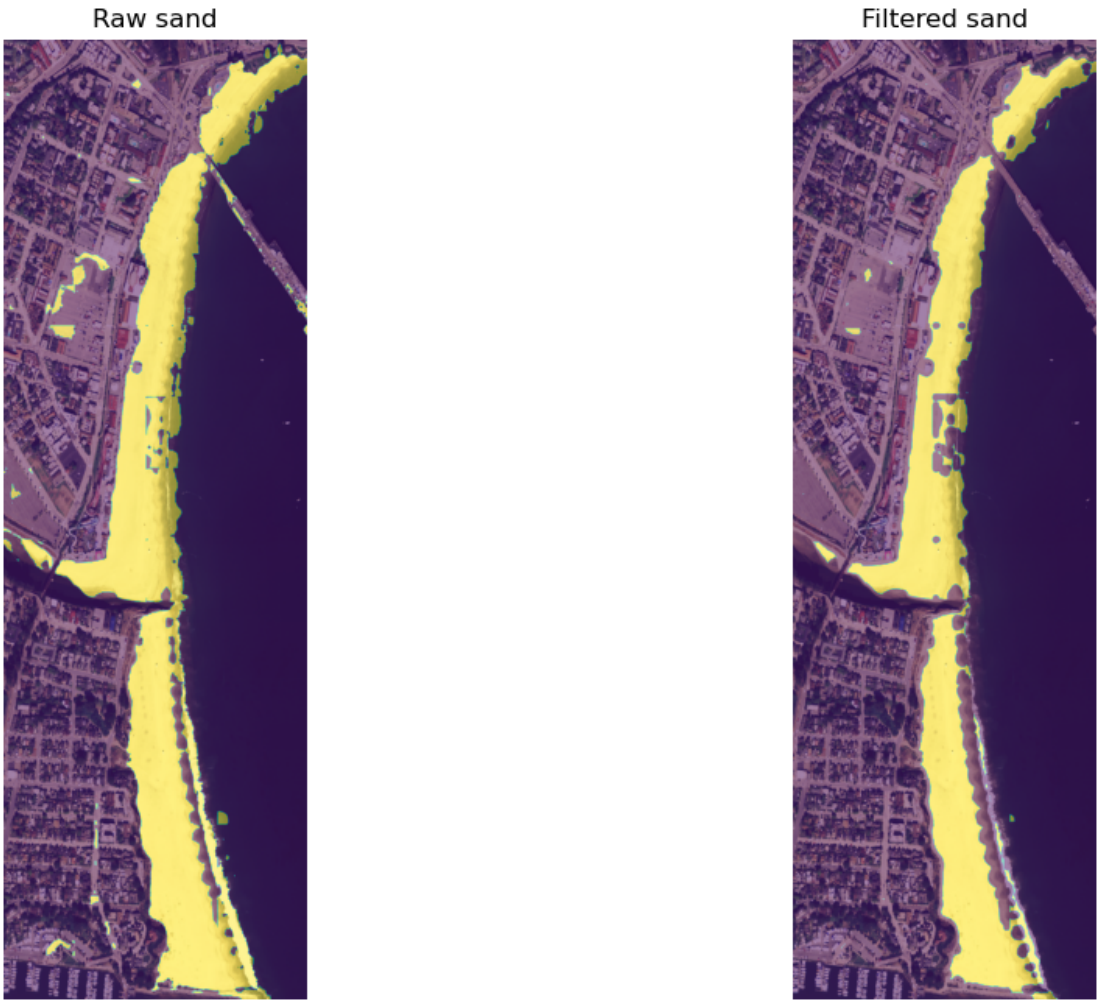
[57]:
```
se = disk(16)
thres_prob = 0.5
```

[58]:
```
im = (geotiffs_ds.sel(time='2005').to_array().squeeze().transpose()>thres_prob).
 ↪to_numpy().astype('int')
#gets rid of small mis-classified blobs of sand
ime = binary_erosion(im, se)
s2005 = get_shoreline(ime)
```

How well does the morpholocial operator work? Seems to work ok; it has removed some small islands of noise but not degraded the main sand areas

[59]:
```
%matplotlib inline
f, axs = plt.subplots(1, 2, figsize=(12,8))

axs[0].imshow(im_show.sel(time='2005').squeeze().transpose()/255.)
im = axs[0].imshow(im, alpha=0.5)
axs[0].axis('off'); axs[0].set_title('Raw sand')

axs[1].imshow(im_show.sel(time='2005').squeeze().transpose()/255.)
im = axs[1].imshow(ime, alpha=0.5)
axs[1].axis('off'); axs[1].set_title('Filtered sand')
```

[59]:
```
Text(0.5, 1.0, 'Filtered sand')
```



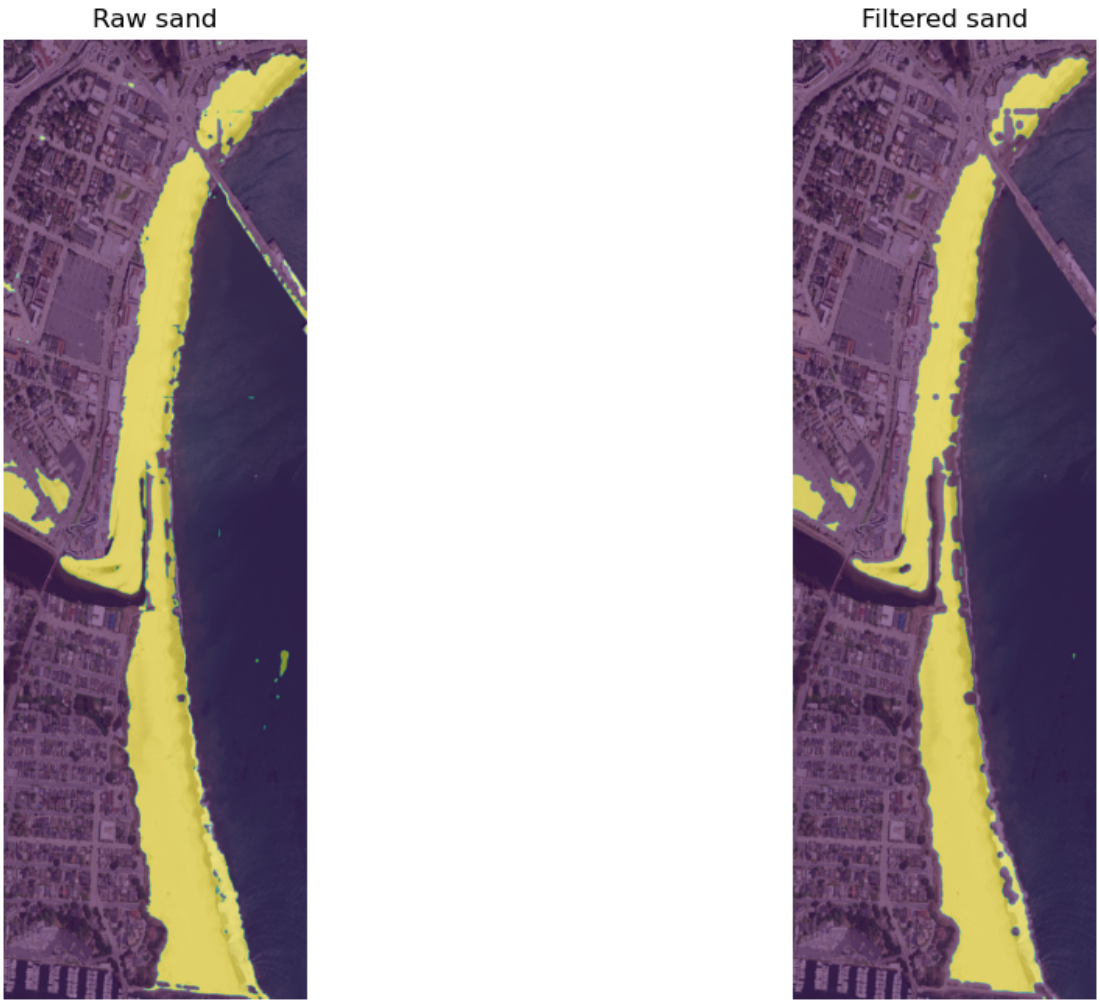
[60]:
```
im = (geotiffs_ds.sel(time='2010').to_array().squeeze().transpose()>.6).
 ↪to_numpy().astype('int')
ime = binary_erosion(im, se)
s2010 = get_shoreline(ime)

im = (geotiffs_ds.sel(time='2014').to_array().squeeze().transpose()>.6).
 ↪to_numpy().astype('int')
ime = binary_erosion(im, se)
s2014 = get_shoreline(ime)

im = (geotiffs_ds.sel(time='2016').to_array().squeeze().transpose()>.6).
 ↪to_numpy().astype('int')
ime = binary_erosion(im, se)
s2016 = get_shoreline(ime)
```

```
im = (geotiffs_ds.sel(time='2018').to_array().squeeze().transpose()>.6).
 ↪to_numpy().astype('int')
ime = binary_erosion(im, se)
s2018 = get_shoreline(ime)

im = (geotiffs_ds.sel(time='2020').to_array().squeeze().transpose()>.6).
 ↪to_numpy().astype('int')
ime = binary_erosion(im, se)
s2020 = get_shoreline(ime)
```

Let's also take a look at this last (2020) image, to make sure our algorithm is performing well

[61]:
```
%matplotlib inline
f, axs = plt.subplots(1, 2, figsize=(12,8))

axs[0].imshow(im_show.sel(time='2020').squeeze().transpose()/255.)
im = axs[0].imshow(im, alpha=0.5)
axs[0].axis('off'); axs[0].set_title('Raw sand')

axs[1].imshow(im_show.sel(time='2020').squeeze().transpose()/255.)
im = axs[1].imshow(ime, alpha=0.5)
axs[1].axis('off'); axs[1].set_title('Filtered sand')
```

[61]:
```
Text(0.5, 1.0, 'Filtered sand')
```

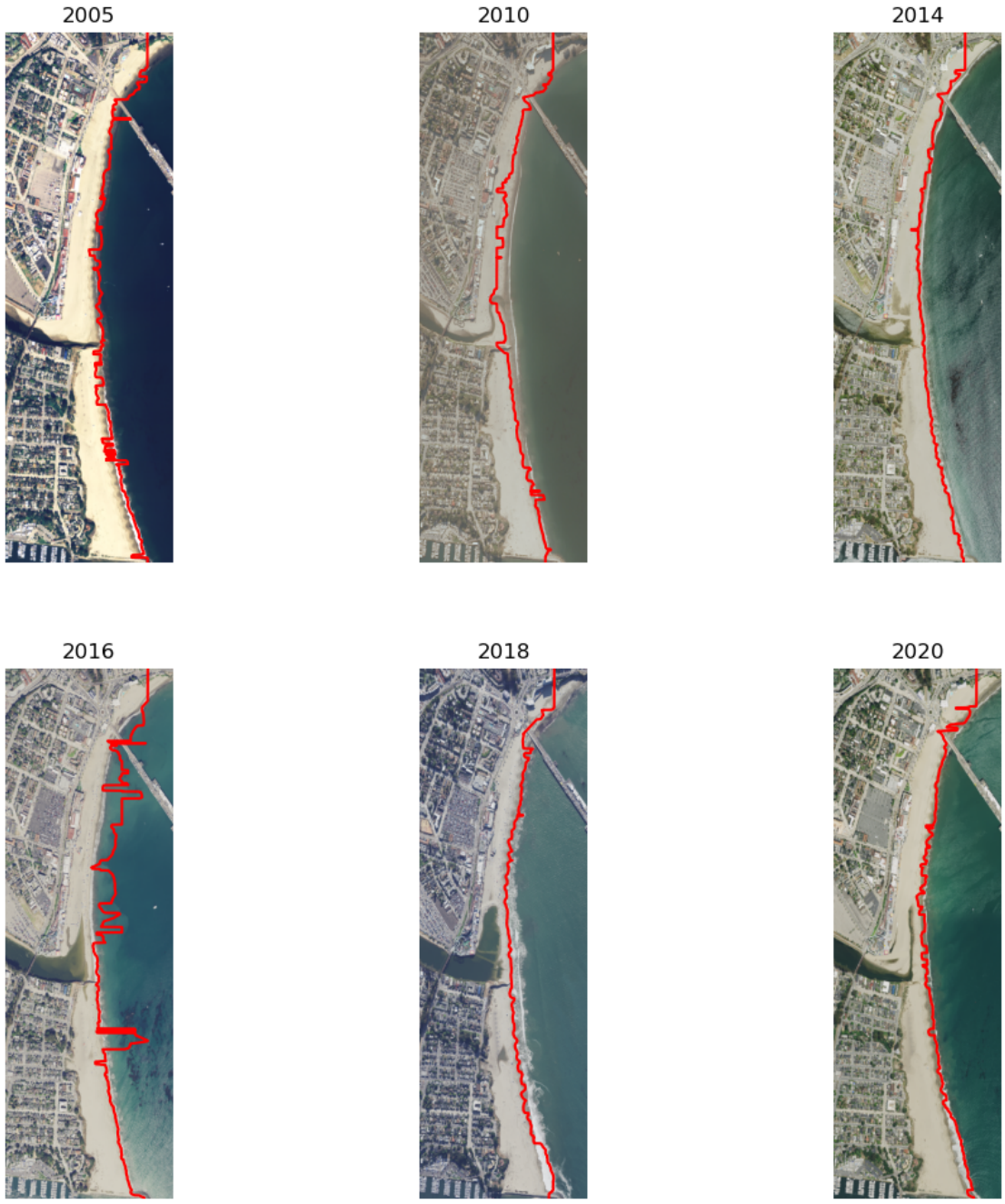


Now we'll make a plot of all images and extracted shorelines — overall, it works ok, but could probably be improved a little - the top end of the image is consistently error-prone, and the 2016 results are problematic

[63]:
```
%matplotlib inline
f, axs = plt.subplots(2,3, figsize=(12,12))

axs[0][0].imshow(im_show.sel(time='2005').squeeze().transpose()/255.)
axs[0][0].plot(s2005,np.arange(len(s2005)),'r')
axs[0][0].axis('off'); axs[0][0].set_title('2005')

axs[0][1].imshow(im_show.sel(time='2010').squeeze().transpose()/255.)
axs[0][1].plot(s2010,np.arange(len(s2010)),'r')
axs[0][1].axis('off'); axs[0][1].set_title('2010')

axs[0][2].imshow(im_show.sel(time='2014').squeeze().transpose()/255.)
```

```
axs[0][2].plot(s2014,np.arange(len(s2014)),'r')
axs[0][2].axis('off'); axs[0][2].set_title('2014')

axs[1][0].imshow(im_show.sel(time='2016').squeeze().transpose()/255.)
axs[1][0].plot(s2016,np.arange(len(s2016)),'r')
axs[1][0].axis('off'); axs[1][0].set_title('2016')

axs[1][1].imshow(im_show.sel(time='2018').squeeze().transpose()/255.)
axs[1][1].plot(s2018,np.arange(len(s2018)),'r')
axs[1][1].axis('off'); axs[1][1].set_title('2018')

axs[1][2].imshow(im_show.sel(time='2020').squeeze().transpose()/255.)
axs[1][2].plot(s2020,np.arange(len(s2020)),'r')
axs[1][2].axis('off'); axs[1][2].set_title('2020')
```

[63]: `Text(0.5, 1.0, '2020')`



According to East et al. (2018), "During record winter rainfall in 2016–2017, the San Lorenzo River, coastal California, had nine flow peaks representing 2–10-year flood magnitudes….these flows exported more than half as much sediment as had a 100-year flood 35 years earlier"

Finally, let's take a look at model outputs from two years bracketing that flood event in 2016/17, namely 2014 and 2020. The shoreline has advanced seaward significantly. We show that, within error, the beach widened slightly between 2014 and 2020. It is possible this is due to fresh sediment inputted during this time from the river.

[64]:
```
%matplotlib inline
f, axs = plt.subplots(1, 1, figsize=(12,8))

axs.imshow(mean_image.to_array().squeeze().transpose()/255.)
axs.plot(s2014,np.arange(len(s2014)),'r', label='2014')
axs.plot(s2020,np.arange(len(s2020)),'y', label='2020')
plt.legend()
plt.axis('off')
```

[64]:
```
(-0.5, 1634.5, 5164.5, -0.5)
```

1.7 Discussion

This is a *very* stripped-down version of the Seg2Map toolbox, through which is is possible to download an image time-series and apply numerous models for image segmentation. See the project wiki for more details. Similar functionality is also available for orthomosaic imagery, using this script in the Segmentation Zoo repository. Note that the Doodleverse is an ongoing community project - if you're interested in helping out, send us a note on Github!

Here we have used image segmentation to create a time-series of sand maps from the mouth of the San Lorenzo River. We show that even a simple application of the model can yield scientifically useful information, such as a time-series of beach area and shoreline location. While much emphasis in the literature is placed on ML model architectures, and the training of such models, as Earth scientists, we are in charge of using these models to extract the scientifically relevant information we need. So, implementation choices are important, too, and they are our purview. Here, we might have achieved better results by employing one or more of several strategies, including:

- Use an ensemble of models, rather than just one model. Typically, models have been trained for the same task on a different dataset, or using different hyperparameters on the same dataset. Ensemble models allow oversampling of the data, often leading to more stable pixelwise averages of per-class probabilities
- Use "test-time augmentation", which is a technique that augments the data at model inference time. Augmentation is the process of creating transformed versions of the imagery; when done well, augmentation of the image, and applying the reverse transformation on the resulting label, then averaging, can lead to significantly improved results. Specific applications will vary.

Similar options are available in the Doodleverse, which allows you to customize your trained models for generic scientific application.

Why is this workflow relevant to the numerical modeling community? Because many models are forced or evaluated by gridded datasets, and image segmentation allows us to isolate individual objects, landcover/use/form, and even similar patterns (i.e. image textures) in those data. We hope and expect the numerical process modeling community to make imaginative use of image segmentation to provide more refined model inputs, and to evaluate model outputs. We suggest that the workflows presented here are especially useful for modeling frameworks that can assimilate time-series data to refine predictions, such as Vitousek et al. (2023) that uses shoreline measurements from imagery (such as here) to improve coastal change forecasting.

1.7.1 Next steps?

- You could make a different ROI.geojson file for another region of interest. We recommend using https://geojson.io or QGIS for this task.
- You could try a different model. Take a look at all the models that are available in the Seg2Map project - you can find all the relevant details on the project's wiki. Note that the Doodleverse applications continue to grow and evolve - let us know if you're interested in being part of the endeavor!

1.8 Acknowledgements

Thanks to Jon Warrick, Chris Sherwood, and the rest of the USGS Remote Sensing Coastal Change (RSCC) project members, Huggingface, and CSDMS for the support!

1.9 References

- Bendixen, M., Iversen, L.L., Best, J., Franks, D.M., Hackney, C.R., Latrubesse, E.M. and Tusting, L.S., 2021. Sand, gravel, and UN Sustainable Development Goals: Conflicts, synergies, and pathways forward. *One Earth*, 4(8), pp.1095-1111.
- Buscombe, D., & Goldstein, E. B. (2022). A reproducible and reusable pipeline for segmentation of geoscientific imagery. *Earth and Space Science*, 9, e2022EA002332. https://doi.org/10.1029/2022EA002332
- Buscombe, D., Goldstein, E. B., Sherwood, C. R., Bodine, C., Brown, J. A., Favela, J., Fitzpatrick, S. (2022), et al. Human-in-the-loop segmentation of Earth surface imagery. *Earth and Space Science*, 9, e2021EA002085. https://doi.org/10.1029/2021EA002085
- Buscombe, D., Wernette, P., Fitzpatrick, S., Favela, J., Goldstein, E.B. and Enwright, N.M., 2023. A 1.2 Billion Pixel Human-Labeled Dataset for Data-Driven Classification of Coastal Environments. *Scientific Data*, 10(1), p.46.
- Buscombe, Daniel. (2023). Doodleverse/Segmentation Zoo/Seg2Map Res-UNet models for CoastTrain/8-class segmentation of RGB 768x768 NAIP images (v1.0) *Zenodo*. https://doi.org/10.5281/zenodo.7570583
- East, A.E., Stevens, A.W., Ritchie, A.C., Barnard, P.L., Campbell-Swarzenski, P., Collins, B.D. and Conaway, C.H., 2018. A regime shift in sediment export from a coastal watershed during a record wet winter, California: Implications for landscape response to hydroclimatic extremes. *Earth Surface Processes and Landforms*, 43(12), pp.2562-2577.
- Vitousek, S., Vos, K., Splinter, K.D., Erikson, L. and Barnard, P.L., 2023. A model integrating satellite-derived shoreline observations for predicting fine-scale shoreline response to waves and sea-level rise across large coastal regions. Authorea Preprints.
- Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J.M. and Luo, P., 2021. SegFormer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34, pp.12077-12090.

[ ]:

[ ]: