0.1 Flood Extent Extraction

This notebook is designed to be an end-to-end solution to extract flood extent maps from aerial imagery. All configurable settings are contained within the `Notebook Settings` cell. The configurable elements are...

- Image Directory Settings
  - `image_list_directory` - String: parent directory of all images
  - `jp2_directory` - String: directory containing jp2 images downloaded from s3
  - `jpg_directory` - String: directory containing jpg images after conversion from jp2
  - `resized_directory` - String: directory containing resized jpg images
  - `prediction_directory` - String: directory containing images representing predicted flood area
- Output Settings
  - `output_directory` - String: directory for output shapefile objects
- Log Settings
  - `log_file_prefix` - String: prefix of log files
  - `log_storage_directory` - String: directory for log files
- Semantic Segmentation Model Settings
  - `training_job` String: the name of the training job being used to generate predictions, different training jobs use different hyperparameter configurations and algorithmic implementations of semantic segmentation
- Image & Shapefile Settings
  - `min_contour_size` - Float: the minimum area of a polygon that will be included in the final output, polygons that do not meet this threshold value will be considered false positives (e.g. ponds)
  - `width` Float: the width to resize input images to before passing the image to the SageMaker Endpoint for flood area prediction

This notebook is an end-to-end solution to load a trained semantic segmentation model and perform inference on the Brewarrina dataset. The workflow can be broken down into the following high-level steps...

1. setup environment variables, logging, s3 connections, and dependencies
2. download Brewarrina dataset, jp2 images must first be converted to jpeg images before being passed to the AWS Sagemaker Endpoint
3. load a previously trained AWS Semantic Segmentation model
4. perform inference on jpeg images
5. remove false positives if necessary
6. extract contours from output png images
7. generate shapefiles

1. Setup

Notebook Settings

In [7]:

```python
import os

# URL of S3 bucket containing JP2 images
dataset = "s3://ss-csu-dataset/tiles/Brewarrina/CIR/"
with open("image_lists/dataset.txt", "w") as f:
    f.write("%s" %dataset)
# directory for text file image lists
os.makedirs("image_lists/", exist_ok=True)
# parent directort for all images
image_list_directory = "../../Images/SS"
os.makedirs(image_list_directory, exist_ok=True)
# NRG jpg and jp2 directories
jpg_directory = os.path.join(image_list_directory, "jpg")
jp2_directory = os.path.join(image_list_directory, "jp2")
os.makedirs(jpg_directory, exist_ok=True)
os.makedirs(jp2_directory, exist_ok=True)
# directory for images to perform test inference
resized_directory = os.path.join(image_list_directory, "jpg_resized")
os.makedirs(resized_directory, exist_ok=True)
# output directory to return PNG images from endpoint
prediction_directory = os.path.join(image_list_directory, "predictions")
os.makedirs(prediction_directory, exist_ok=True)
# output directory of shapefiles generated from inference images
output_directory = "../../Out"
os.makedirs(output_directory, exist_ok=True)
# log file constants
log_file_prefix = "NOTEBOOK_LOG"
log_storage_directory = "../../Logs"
os.makedirs(log_storage_directory, exist_ok=True)
# training job to load
training_job = "csu-flood-beta-resnet50-10-2021-08-25-10-37-24-882" # fcn resnet50 min 10 epochs
# ensure flood areas > 100 pixels
min_contour_size = 100
# png resize constant
width = 1000
```

Dependencies

In [ ]:

```
%%time
%conda update --all
%conda install -y -c conda-forge gdal
%conda install -y -c conda-forge fiona
```

Initiate s3 and sagemaker connections

In [5]:

```python
%%time
import sys
import logging
import shutil
import datetime
from PIL import Image
from matplotlib import pyplot as plt
import numpy as np
import cv2
from osgeo import gdal
from osgeo import osr
import fiona
import sagemaker
from sagemaker import get_execution_role

sess = sagemaker.Session()
role = get_execution_role()
bucket = sess.default_bucket()

print(role)
print(bucket)
```

```
arn:aws:iam::969964825157:role/SageMakerDeveloperAccess
sagemaker-ap-southeast-2-969964825157
CPU times: user 344 ms, sys: 8.11 ms, total: 352 ms
Wall time: 4.5 s
```

In [5]:

```python
cell_error = False

if not 'logger_initiated' in globals():
    logger = logging.getLogger(log_file_prefix)
    # use different log level for file and console
    timestamp = datetime.datetime.utcnow().strftime('%Y%m%d_%H-%M-%S')
    formatter = logging.Formatter('[%(asctime)s] %(name)s %(levelname)s - %(message)s')

    # file settings
    filename = os.path.join(log_storage_directory, f"{log_file_prefix}_{timestamp}.log")
    try:
        file_handler = logging.FileHandler(filename=filename)
        file_handler.setLevel(logging.DEBUG)
        file_handler.setFormatter(formatter)
        # stream stdout settings
        stream_handler = logging.StreamHandler(sys.stdout)
        stream_handler.setLevel(logging.INFO)
        stream_handler.setFormatter(formatter)
        # handlers have to be at a root level since they are the final output
        logger.addHandler(stream_handler)
        logger.addHandler(file_handler)

        # log global debug to a separate file, log file will include debug logs from import modules
        global_filename = os.path.join(log_storage_directory, f"{log_file_prefix}_EXTRA_{timestamp}.log")
        file_handler = logging.FileHandler(filename=global_filename)
        file_handler.setLevel(logging.DEBUG)
        file_handler.setFormatter(formatter)
        logging.basicConfig(
        level=logging.DEBUG,
        handlers=[
            file_handler
        ])
        logger.info(f"Finished configuring logging. Log file: {filename}, Global log file: {global_filename}")
        logger_initiated = True
    except Exception as e:
        cell_error = True
        logger.error(f"Unable to configure logging: {e}.")
else:
    logger.warning(f"Logger is already configured. Log file: {filename}, Global log file: {global_filename}")

if cell_error:
    raise SystemExit("Execution stopped...")
```

```
[2021-09-03 04:16:50,557] NOTEBOOK_LOG INFO - Finished configuring logging. Log file: Logs/NOTEBOOK_LOG_20210903_04-16-50.log, Global log file: Logs/NOTEBOOK_LOG_EXTRA_20210903_04-16-50.log
```

In [6]:

```python
# log settings to file
logger.debug(f"Notebook settings:\n"
            f"\t\t\t\t\t\t dataset {dataset}\n"
            f"\t\t\t\t\t\t training_job {training_job}\n"
            f"\t\t\t\t\t\t image_width {width}\n"
            f"\t\t\t\t\t\t minimum_contour_size {min_contour_size}\n"
            f"\t\t\t\t\t\t output_directory {output_directory}\n"
            f"\t\t\t\t\t\t log_file_prefix {log_file_prefix}\n"
            f"\t\t\t\t\t\t log_storage_directory {log_storage_directory}")
```

2. Download dataset

In [11]:

```
!aws s3 cp $(cat image_lists/dataset.txt) ../../Images/jp2/ --recursive
```

```
download: s3://live-demo-images/brewarrina/Brewarrina_Flood_2021_04_15cm_NRG_10.jp2 to nrg_images/brewarrina/jp2/Brewarrina_Flood_2021_04_15cm_NRG_10.jp2
download: s3://live-demo-images/brewarrina/Brewarrina_Flood_2021_04_15cm_NRG_16.jp2 to nrg_images/brewarrina/jp2/Brewarrina_Flood_2021_04_15cm_NRG_16.jp2
download: s3://live-demo-images/brewarrina/Brewarrina_Flood_2021_04_15cm_NRG_9.jp2 to nrg_images/brewarrina/jp2/Brewarrina_Flood_2021_04_15cm_NRG_9.jp2
download: s3://live-demo-images/brewarrina/Brewarrina_Flood_2021_04_15cm_NRG_2.jp2 to nrg_images/brewarrina/jp2/Brewarrina_Flood_2021_04_15cm_NRG_2.jp2
```

Convert jp2 to jpeg

In [29]:

```
!ls ../../Images/jp2/*.jp2 > image_lists/brewarrina_jp2_list.txt
```

In [30]:

```python
cell_error = False

with open("image_lists/brewarrina_jp2_list.txt") as f:
    jp2_img_list = f.read().splitlines()

try:
    filename_list = []
    if len(jp2_img_list) > 0:
        logger.info(f"Directory {nrg_path} contains {len(jp2_img_list)} images")
    else:
        raise Exception(f"No images in directory {ngr_path}")
        cell_error = True
    for i in jp2_img_list:
        stem = i.split(".jp2")[0]
        fn = stem.split("jp2/")[1]
        filename_list.append(fn)
except Exception as e:
    logger.error(f"Unable to open images: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution stopped")

with open("image_lists/brewarrina_stem_list.txt", "w") as f:
    for i in filename_list:
        f.write("%s\n" %i)
```

```
[2021-09-02 23:54:30,302] NOTEBOOK_LOG INFO - Directory nrg_images/brewarrina/ contains 5 images
```

In [31]:

```
!for file in $(cat image_lists/brewarrina_stem_list.txt); do gdal_translate -of JPEG ../../Images/jp2/$file.jp2 ../../Images/jpg/$file.jpeg; done
```

```
Input file size is 10000, 10000
0Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace
...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 10000, 10000
0Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace
...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 10000, 10000
0Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace
...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 10000, 10000
0Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace
...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 10000, 10000
0Warning 1: 4-band JPEGs will be interpreted on reading as in CMYK colorspace
...10...20...30...40...50...60...70...80...90...100 - done.
```

3. Load previously trained semantic segmentation model and deploy endpoint

In [8]:

```python
cell_error = False

try:
    ss_estimator = sagemaker.estimator.Estimator.attach(training_job)
    logger.info(f"Loading training job {training_job}")
except Exception as e:
    logger.error(f"Unable to load training job {training_job}: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution Stopped")
```

```
2021-08-25 12:37:18 Starting - Preparing the instances for training
2021-08-25 12:37:18 Downloading - Downloading input data
2021-08-25 12:37:18 Training - Training image download completed. Training in progress.
2021-08-25 12:37:18 Uploading - Uploading generated training model
2021-08-25 12:37:18 Completed - Training job completed
[2021-09-03 04:18:12,155] NOTEBOOK_LOG INFO - Loading training job csu-flood-beta-resnet50-10-2021-08-25-10-37-24-882
```

In [9]:

```python
%%time

cell_error = False

try:
    ss_predictor = ss_estimator.deploy(initial_instance_count=1, instance_type="ml.c5.xlarge")
    logger.info(f"Deploying endpoint {ss_predictor} for estimator {ss_estimator}")
except Exception as e:
    logger.error(f"Unable to deploy endpoint for estimator {ss_estimator}: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution Stopped")
```

```
-------------![2021-09-03 04:24:46,937] NOTEBOOK_LOG INFO - Deploying endpoint <sagemaker.predictor.Predictor object at 0x7fea3f65a3c8> for estimator <sagemaker.estimator.Estimator object at 0x7fea3f
6372b0>
CPU times: user 172 ms, sys: 10.1 ms, total: 182 ms
Wall time: 6min 32s
```

4. Perform inference on images

Define deserializer to interpret inference results as basic PNG ID mask

In [10]:

```python
class ImageDeserializer(sagemaker.deserializers.BaseDeserializer):
    """Deserialize a PIL-compatible stream of Image bytes into a numpy pixel array"""

    def __init__(self, accept="image/png"):
        self.accept = accept

    @property
    def ACCEPT(self):
        return (self.accept,)

    def deserialize(self, stream, content_type):
        try:
            return np.array(Image.open(stream))
        finally:
            stream.close()

cell_error = False

try:
    ss_predictor.deserializer = ImageDeserializer(accept="image/png")
    logger.info(f"Setting custom desierlizer for endpoint {ss_predictor}")
except Exception as e:
    logger.error(f"Unable to assign base deserializer to {ss_predictor}: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution Stopped")
```

```
[2021-09-03 04:30:13,377] NOTEBOOK_LOG INFO - Setting custom desierlizer for endpoint <sagemaker.predictor.Predictor object at 0x7fea3f65a3c8>
```

Get list of jpeg filenames

In [12]:

```python
with open("image_lists/brewarrina_stem_list.txt") as f:
    stem_list = f.read().splitlines()

# create list of jpeg image filenames
jpeg_img_list = []
for i in stem_list:
    fn = i + ".jpeg"
    jpeg_img_list.append(fn)
```

Resize the jpeg images before passing to endpoint

In [13]:

```python
%%time

cell_error = False

image_list = []

# create list of resized jpeg filenames
resize_list = []
for file in jpeg_img_list:
    file_resize = file.split(".jpeg")[0] + "_resized.jpeg"
    resize_list.append(file_resize)

try:
    logger.info(f"Resizing {len(jpeg_img_list)} jpeg images before passing to endpoint")
    for i, img in enumerate(jpeg_img_list):
        im = Image.open(f"nrg_images/brewarrina/jpg/{img}")
        aspect = im.size[0] / im.size[1]
        im.thumbnail([width, int(width/aspect)], Image.ANTIALIAS)
        im.save(f"predict_images/brewarrina/{resize_list[i]}", "JPEG")

        _im = cv2.imread(f"nrg_images/brewarrina/jpg/{img}", cv2.IMREAD_COLOR)
        _im = cv2.cvtColor(_im, cv2.COLOR_RGB2BGR)
        aspect = _im.shape[1] / _im.shape[0]
        _im = cv2.resize(_im, (width, int(width/aspect)))
        image_list.append(_im)

except Exception as e:
    logger.error(f"Error resizing jpeg images: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution Stopped")
```

```
[2021-09-03 04:30:25,692] NOTEBOOK_LOG INFO - Resizing 5 jpeg images before passing to endpoint
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/PIL/Image.py:2850: DecompressionBombWarning: Image size (100000000 pixels) exceeds limit of 89478485 pixels, could be decompression b
omb DOS attack.
  DecompressionBombWarning,
CPU times: user 13.4 s, sys: 1.59 s, total: 15 s
Wall time: 15.2 s
```

Convert images to stream of bytes to be sent to endpoint through serializer

In [15]:

```python
cell_error = False

# use default serializer
ss_predictor.serializer = sagemaker.serializers.IdentitySerializer("image/jpeg")

imbytes_list = []
try:
    logger.info(f"Converting {len(resize_list)} images to byte streams")
    for file in resize_list:
        imfile = open(f"predict_images/brewarrina/{file}", "rb")
        logger.debug(f"Converting {imfile} to byte stream")
        bytes_file = imfile.read()
        imbytes_list.append(bytes_file)
        imfile.close()
except Exception as e:
    logger.error(f"Unable to convert jpeg images into byte streams: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution Stopped")
```

```
[2021-09-03 05:18:15,918] NOTEBOOK_LOG INFO - Converting 5 images to byte streams
```

In [49]:

```python
print(len(imbytes_list))
```

```
5
```

perform inference

In [16]:

```python
%%time
cell_error = False

mask_list = []
try:

    logger.info(f"Using default serializer {ss_predictor.serializer} for endpoint")
    logger.info(f"Starting inference for {len(imbytes_list)} images using {ss_predictor}")
    for i, file in enumerate(imbytes_list):
        logger.debug(f"Predicting flood area for {resize_list[i]}")
        mask = ss_predictor.predict(file)
        mask_list.append(mask)
except Exception as e:
    logger.error(f"Unable to perform inference: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution Stopped")
```

```
[2021-09-03 05:18:20,690] NOTEBOOK_LOG INFO - Using default serializer <sagemaker.serializers.IdentitySerializer object at 0x7fea3d1f2748> for endpoint
[2021-09-03 05:18:20,691] NOTEBOOK_LOG INFO - Starting inference for 5 images using <sagemaker.predictor.Predictor object at 0x7fea3f65a3c8>
CPU times: user 89.7 ms, sys: 7.44 ms, total: 97.1 ms
Wall time: 1min 15s
```

In [18]:

```python
flood_list = []
flood_cloud_list = []

for mask in mask_list:
    new_mask = np.zeros((mask.shape))
    new_mask = mask == 0 # flood
    flood_list.append(new_mask)
    new_mask = np.zeros((mask.shape))
    new_mask = mask == 1 # flood + cloud
    flood_cloud_list.append(new_mask)
```

Combine flood and flood + cloud pixels

In [19]:

```python
comp_mask_list = []

for i, mask in enumerate(flood_list):
    comp_mask = np.zeros((mask.shape))
    comp_mask = np.logical_or(mask, flood_cloud_list[i])
    comp_mask_list.append(comp_mask)
```

Extract negative area from edge images

In [21]:

```python
neg_mask_list = []
for i in range(len(image_list)):
    temp_img = cv2.cvtColor(image_list[i], cv2.COLOR_BGR2GRAY)
    temp_mask = np.zeros(temp_img.shape)
    temp_mask = temp_img == 0
    neg_mask_list.append(temp_mask)
```

5. Remove false positives

In [22]:

```python
def remove_noise(comp_img, neg_img):
    for i in range(comp_img.shape[0]):
        for j in range(comp_img.shape[1]):
            if comp_img[i][j] == True:
                if neg_img[i][j] == True:
                    comp_img[i][j] = False
    return comp_img
```

In [23]:

```python
final_mask_list = []

for i in range(len(comp_mask_list)):
    comp_img = comp_mask_list[i].copy()
    neg_img = neg_mask_list[i]
    temp_img = remove_noise(comp_img, neg_img)
    final_mask_list.append(temp_img)
```

6. Extract contours from inference images

In [25]:

```python
%%time
cell_error = False
contour_results = []

try:
    logger.info(f"Extracting contours from {len(final_mask_list)} images")
    for i in range(len(final_mask_list)):
        contour_results.append([])
        logger.debug(f"Extracting contours from image {jpeg_img_list[i]}")
        contours, hierarchy = cv2.findContours(np.uint8(final_mask_list[i]), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        for c in range(len(contours)):
            if cv2.contourArea(contours[c]) > min_contour_size:
                contour_results[i].append(contours[c])
                # reverse orientation to create doughnuts
                if hierarchy[0][c][3] != -1:
                    logger.debug(f"Inverting island for image {jpeg_img_list[i]} contour {c}")
                    contour_results[i][-1] = np.flipud(contour_results[i][-1])
except Exception as e:
    logger.error(f"Unable to extract contours: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution Stopped")
```

```
[2021-09-03 05:20:19,912] NOTEBOOK_LOG INFO - Extracting contours from 5 images
CPU times: user 31.6 ms, sys: 4.51 ms, total: 36.1 ms
Wall time: 190 ms
```

7. generate shapefiles

In [ ]:

```python
%%time
cell_error = False

def pixel2location(dx, dy):
    """Convert pixel coordinates to spatial coordinates
    dx: x axis pixel
    dy: y axis pixel
    """
    x = dx * pixel_x_size + pixel_x_offset
    y = dy * pixel_y_size + pixel_y_offset
    return x,y

def getEPSG(ds):
    """Returns EPSG code for supplied dataset
    ds: gdal dataset
    """
    proj = osr.SpatialReference(wkt=ds.GetProjection())
    return proj.GetAttrValue('AUTHORITY', 1)

schema = {
    'geometry': 'MultiPolygon',
    'properties': [('tag', 'str')]
}

width = 1000

try:
    logger.info(f"Creating shapefiles for {len(jp2_img_list)} images")
    for i in range(len(jp2_img_list)):
        logger.debug(f"Reading spatial data from {jp2_img_list[i]}")
        ds = gdal.Open(jp2_img_list[i], gdal.GA_ReadOnly)
        pixel_x_offset, pixel_x_size, _, pixel_y_offset, _, pixel_y_size = ds.GetGeoTransform()
        epsg = getEPSG(ds)
        # scale pizel size
        scale = width / ds.RasterXSize
        pixel_x_size = pixel_x_size / scale
        pixel_y_size = pixel_y_size / scale
        logger.debug(f"{jp2_img_list[i]} spatial data: x pixel width = {pixel_x_size}m, y pixel width = {pixel_y_size}m, location is {pixel_x_offset}:{pixel_y_offset}")
        if not len(contour_results[i]) > 0:
            logger.debug(f"No polygons for {jp2_img_list[i]}. No shapefile will be generated.")
            continue
        else:
            logger.debug(f"{len(contour_results[i])} polygons for {jp2_img_list[i]} will be created.")
        multi_polygon = []
        for contour in range(len(contour_results[i])):
            polygon = []
            multi_polygon.append(polygon)
            # convert polygon pixel coordinates to spatial points
            for pixel in contour_results[i][contour]:
                x, y = pixel2location(pixel[0][0], pixel[0][1])
                polygon.append([x, y])

        shp_file_output = os.path.join(output_directory, f"{stem_list[i]}.shp")
        logger.debug(f"Writing {shp_file_output}")
        try:
            with fiona.open(shp_file_output, 'w', 'ESRI Shapefile', schema=schema, crs=ds.GetProjection()) as shp_file:
                logger.debug(f"Shape file epsg: {shp_file.crs}")
                shp_file.write({
                    'geometry': {'type': 'MultiPolygon',
                    'coordinates': [multi_polygon]},
                    'properties': {
                        'tag': 'flood extent'
                    }
                })
        except Exception as e:
            print(f"error creating shapefile...\n{e}")
except Exception as e:
    logger.error(f"Unable to convert contours to shapefiles: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution Stopped")
```

Export results to zip file

In [48]:

```python
zip_output = os.path.join(output_directory, "brewarrina")

try:
    logger.info(f"Starting compression of {output_directory}, writing to {zip_output}.zip")
    shutil.make_archive(zip_output, 'zip', output_directory)
    logger.info(f"Success compressing and writing outputs to {zip_output}.zip")
except Exception as e:
    print(f"Unable to compress and write outputs: {e}")
```

```
[2021-09-03 00:03:22,969] NOTEBOOK_LOG INFO - Starting compression of shapefiles/brewarrina/, writing to shapefiles/brewarrina_output/.zip
[2021-09-03 00:03:22,977] NOTEBOOK_LOG INFO - Success compressing and writing outputs to shapefiles/brewarrina_output/.zip
```

8. Clean up resources

In [ ]:

```python
try:
    for file in os.listdir(image_list_directory):
        if file.endswith(".zip") or os.path.isdir(file):
            pass
        else:
            os.remove(os.path.join(image_list_directory, file))
    logger.info(f"Removing files from outputs storage {output_directory}")
    for file in os.listdir(output_directory):
        if file.endswith(".zip") or os.path.isdir(file):
            pass
        else:
            os.remove(os.path.join(output_directory, file))
except Exception as e:
    logger.error(f"Unable to clean up directories: {e}")
```

In [50]:

```python
ss_predictor.delete_endpoint()
```