G5

PROJECT USER MANUAL

Semantic Segmentation Pipeline

Aerial Imagery Initiative

Index

Index **1**

Introduction **2**
Assumptions 2

Overview and Environment **2**
SageMaker Notebook 3

Workflow **5**
Clone the Repository 5
Notebook Setup and Execution 6
Initialise logging, S3 connections, and dependencies 6
Download dataset 7
Load Semantic Segmentation Mode and Deploy Endpoint 7
Predict Flood Extent 8
Remove False Positives 9
Extract Contours and Generate Shapefiles 10
Compress Output and Clean Up Resources 10

Introduction

This user manual provides a guide on how to use the Flood Extent Extraction (FEE) notebooks developed by the CSU Student Team for the NSW Department of Customer Service (DCS) and Spatial Services (SS). This manual will cover setting up the AWS SageMaker Environment, downloading a new dataset from AWS Simple Storage Service (S3), performing predictions on flood areas using a trained Semantic Segmentation model, generating shapefiles, and cleaning up resources.

Assumptions

It is assumed users will have some level of proficiency with the AWS services such as S3 and SageMaker, Bitbucket repositories, and coding in Python. For this reason some steps such as logging in to AWS services and standard input and output operations may be omitted.

List of Assumptions	
A1	User has an AWS account with valid credentials
A2	User has access to S3 bucket containing flood imagery in JP2 format
A3	User is familiar with working in AWS SageMaker environments
A4	User has access to Spatial Services Bitbucket repository with notebooks discussed in this user manual
A5	User is familiar with cloning repositories from Bitbucket
A6	User is proficient in Python and commonly used libraries in data science such as *numpy*
A7	JP2 images are in NRG color space (near infrared, red, green)

Overview & Environment

This document provides guidance on how to use the AWS SageMaker Semantic Segmentation algorithm to generate flood extent maps for use in a GIS software environment as well as common errors that may be encountered. This beta-model does not provide a fully automated end-to-end solution, instead this beta model is implemented as a Jupyter Notebook with blocks of cells sequentially executed. This format of work-flow is preferred for many data-intensive machine learning applications where the output of each step must be verified before proceeding to the next step. This staggered approach provides a degree of flexibility not otherwise available to an interpreted coding language. The SageMaker environment which hosts the Jupyter Notebook is a preconfigured Conda and Python environment. The execution stack is:

- AWS SageMaker
- AWS Instance
- Python / Conda Environment

- Executable Jupyter Notebook

The workflow of the flood extraction project consists of the following steps:

- Setup environment variables
- Load third party modules[1]
- Import libraries and initialise S3 connections
- Download dataset
- Convert images to JPEG format
- Load previously trained model and deploy endpoint
- Define deserializer
- Perform inference on input images
- Extract flood area from predictions
- Remove false positives
- Extract contours from images
- Generate shapefiles
- Export output to zip folder
- Cleanup resources

The prerequisites for executing the Semantic Segmentation Pipeline are outlined below.

List of Prerequisites	
P1	Access to the Bitbucket repository containing the required Notebook <link>
P2	Authorised access to AWS and AWS SageMaker
P3	JPEG2000 Images of a flood event hosted on an S3 bucket

SageMaker Notebook

The first step is to set up a SageMaker Notebook. There are multiple EC2 instance options on which to host the notebook. It is recommended to use the memory optimised ml.r5.12xlarge instance with a volume size of 15 GB. In addition to limporting the original JPEG2000 images from S3, this project involves translating and resizing the entire image dataset. Therefore this project requires a large amount of memory.
A lifecycle configuration has been created to automatically configure the Conda environment in the background each time the notebook starts. This script downloads the fiona and gdal packages that are required to import JPEG2000 images and generate shapefile objects. This may take 15 - 20 minutes to finish installing in the background. If this lifecycle is not available then there is a cell which will manually install the required packages within the notebook.

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

Figure 1 - cell to install gdal and fiona

[1] If the lifecycle script is not available. Otherwise the code described in Figure 1 will perform this step manually.

Figure 2 - Lifecycle script to automatically download gdal and fiona

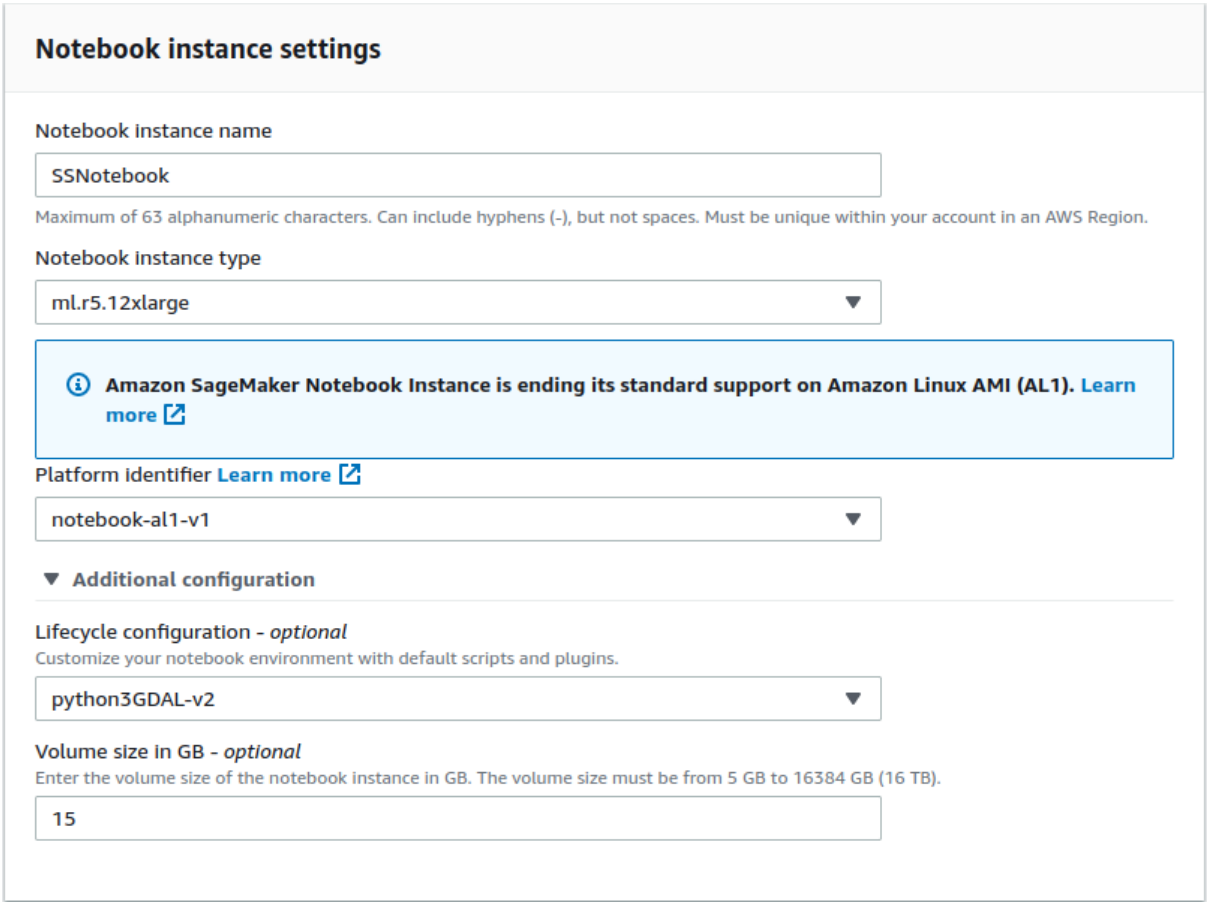

Figure 3 - Notebook configuration settings

The recommended kernel for the Jupyter Notebook is the conda_python3 kernel as it has the fewest number of inconsistencies with the gdal and fiona packages, thereby reducing the time required for Conda to resolve the environment.

Notebook instance name	SSNotebook
Notebook instance type	ml.r5.12xlarge
Lifecycle configuration	python3GDAL
Volume size	15GB

Workflow

Clone the Repository

All code for the Semantic Segmentation pipeline is available at the following repository URL: https://bitbucket.org/csu-spatialservices/flood-extent-extraction/src/ss-pipeline/. This repository must be cloned into an AWS SageMaker environment.

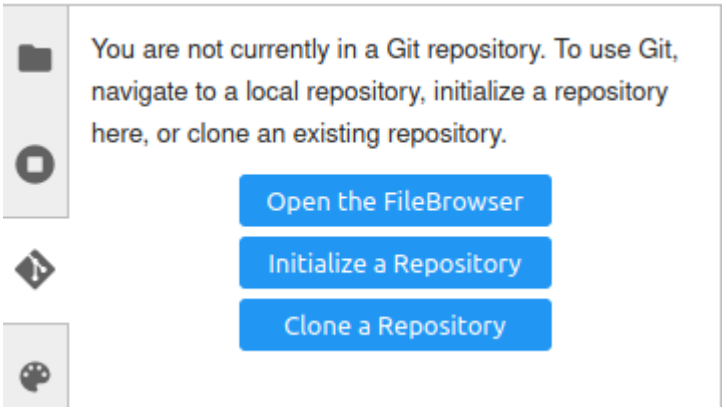

Figure 4 - SageMaker Git Directory

The directory structure of the repository is outlined below along with brief descriptions of each subfolder.

```
flood-extent-extraction/
├── App/
│   ├── SS Extraction Method/
│   ├── image_lists/
├── Documentation/
├── Images/
│   ├──SS
│       ├──jp2/
│       ├──jpg/
│       ├──jpg_resized/
│       ├──predictions/
├── Logs/
├── Out/
```

App - notebooks to be executed
Documentation - contains supporting documentation such as user manuals
Images - contains subdirectories for the different versions of the image dataset
Logs - files of the logging system output, error message can be examined in these files
Out - the shapefiles of the flood extent area

Notebook Setup and Execution

The notebook for the SSemantic Segmentation pipeline can be found at

```
flood-extent-extraction/
├─ App/
│  ├─ SS Extraction Method/
│  │   ├─AWS_Semantic_Segmentation_Pipeline.ipynb
```

The first cell in the notebook contains all configurable settings. Most settings are likely to remain constant except the *dataset* variable which contains the S3 bucket name of the dataset.

Image Directory Settings

- **dataset** - String: AWS S3 bucket containing JP2 input images
- **Image_list_directory** String: parent directory of all image folders
- **jp2_directory** - String: subdirectory containing dataset of JP2 images
- **jpg_directory** - String: subdirectory containing whole dataset in JPEG format
- **resized_directory** - String: resized JPEG images of dataset
- **prediction_directory** - String: directory containing images of predicted flood extent

Output Settings

- **output_directory** - String: directory containing shapefiles of predicted flood extent

Log Settings

- **log_file_prefix** - String: log file prefix
- **log_storage_directory** - String: directory of log files

Semantic Segmentation Model Settings

- **training_job** - String: name of training job for semantic segmentation machine learning model

Image & Shapefile Settings

- **min_contour_size** - Float: the minimum size of a polygon to be included in the final output
- **width -** Float: the width of all resized images

Once the first cell is configured properly, all subsequent cells can be executed without further user input. All cells can be executed automatically by selecting *Run* > *Run All Cells*. The large-scale steps are as follows.

Initialise logging, S3 connections, and dependencies

If the user has the permissions to access AWS SageMaker resources these cells should execute without error with a similar output to figure 5. If an error occurs, check the credentials before proceeding.

```
%%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/SC-969964825157-pp-7ldugvsvgee36-SageMakerRole-AIDFOLZYEOUC
sagemaker-ap-southeast-2-969964825157
CPU times: user 1.51 s, sys: 449 ms, total: 1.96 s
Wall time: 19.9 s
```

Figure 5 - Output of initialising S3 connections

Download dataset

A potential error that may occur during the execution of this cell is surpassing the resource limits of the SageMaker instance. If a download error failure is returned, attempt to create a new notebook instance with a higher volume size. Otherwise the output should be similar to figure 6.

```
!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
```

Figure 6 - Download dataset output

Conversion of the JPEG2000 images to JPEG format will proceed as long as gdal has successfully installed in the background.

```
!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.
```

Figure 7 - Translation of images from JPEG2000 to JPEG

Load Semantic Segmentation Model and Deploy Endpoint

The endpoint is the point of access for a previously trained model. Deploying the endpoint on its own EC2 instance takes approximately seven minutes.

```
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
```

```
%%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.es
timator.Estimator object at 0x7fea3f6372b0>
CPU times: user 172 ms, sys: 10.1 ms, total: 182 ms
Wall time: 6min 32s
```

Figure 8 - Deploy Endpoint

Images must be serialized before being sent to an endpoint and output from the endpoint must be serialized. For this project the default serializer for PNG images will be used, but a deserializer for the JPEG input images must be defined. The use of JPEG input images is a constraint of the AWS Semantic Segmentation training task not included in this notebook.

```
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>
```

Figure 9 - Deserializer

Predict Flood Extent

Predicted flood areas are returned from the endpoint as PNG mask images. The flood areas are pixels that have a label of either 1, representing the flood label, or 2, representing the flood with cloud label. Figure 10 shows some example output for the Brewarrina dataset. For

large datasets such as the Hawkesbury Central or Lower Clarence datasets this can take approximately one hour. If the cell times out without returning a specific error message it is likely due to insufficient resources on the EC2 instance hosting the endpoint. Either resize the input images to a smaller scale or redeploy the endpoint on a large EC2 instance. The one recommended for most datasets is the ml.c5.xlarge compute optimized instance for JPEG images with a width of 1000 pixels.

```
%%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
```

```
pltImages(mask_list)
```

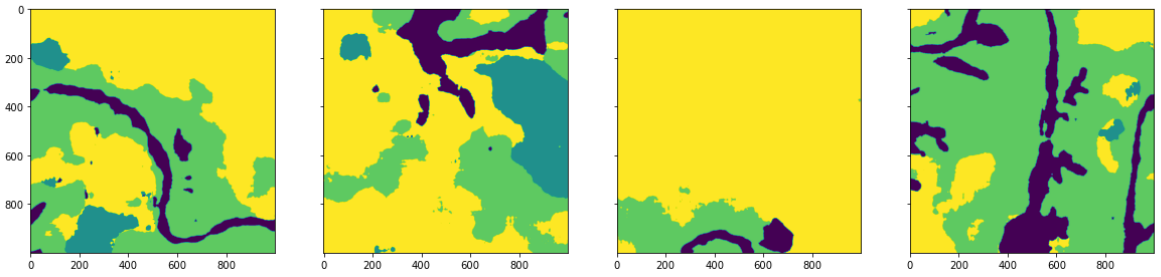

Figure 10 - Flood Predictions

Remove False Positives

For datasets that contain edge images such as the Brewarrina and Hawkesbury-Nepean datasets, there is likely to be a significant degree of false positives correlated with padded areas as outlined in the figure below.

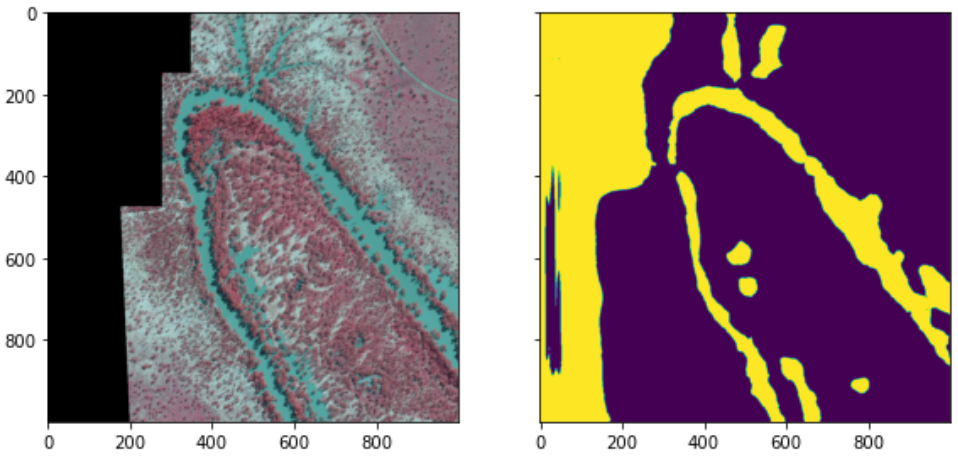

Figure 11 - False positives

These negative spaces in the dataset must be mapped out and removed from the output before the shapefiles can be generated. To this end, a simple method has been written that takes as input a composite image of the predicted flood area as well as the area of the padded image (this corresponds to the black part of the left image in figure 11). The method returns a new binary image without the false positives.

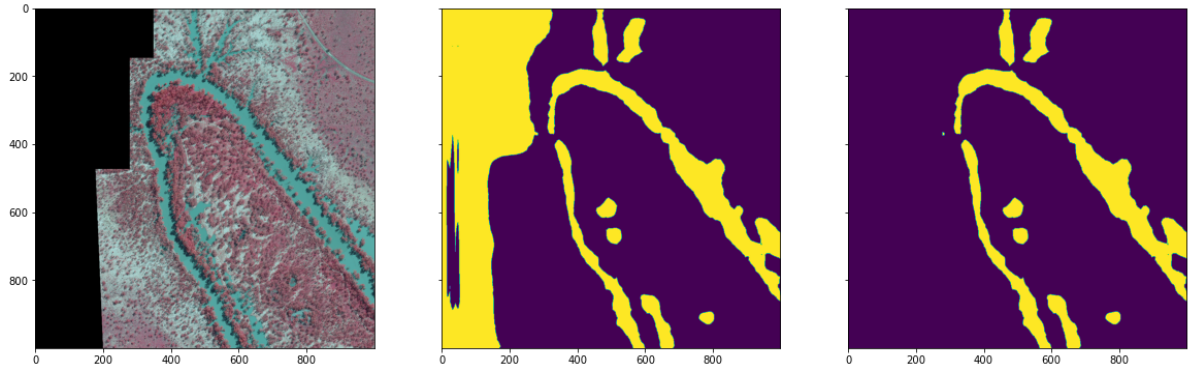

Figure 12 - Removal of false positives

Extract Contours and Generate Shapefiles

Contours of the flood area can now be extracted from the images. This should take less than a second. Shapefiles can also be generated using the metadata from the JP2 images to map the vector points to real world coordinates.

```
%%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
```

Figure 13 - Extract contours

Compress Output and Clean Up Resources

The final step is to export the shapefiles to zip and remove the images from the SageMaker Notebook instance. The JPEG, resized JPEG, and PNG inference images are no longer needed. It is also important to ensure that the EC2 instance hosting the endpoint is deleted after use. If it is left running it can incur significant charges.
Any endpoints left running can be checked in the SageMaker console environment under the *Inference* > *Endpoints* tab.

```
### Export results to zip file
```

```
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
```

```
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}")
```

DELETE ENDPOINT AT END OF EVERY SESSION

```
ss_predictor.delete_endpoint()
```

Figure 14 - Compress output and clean up resources