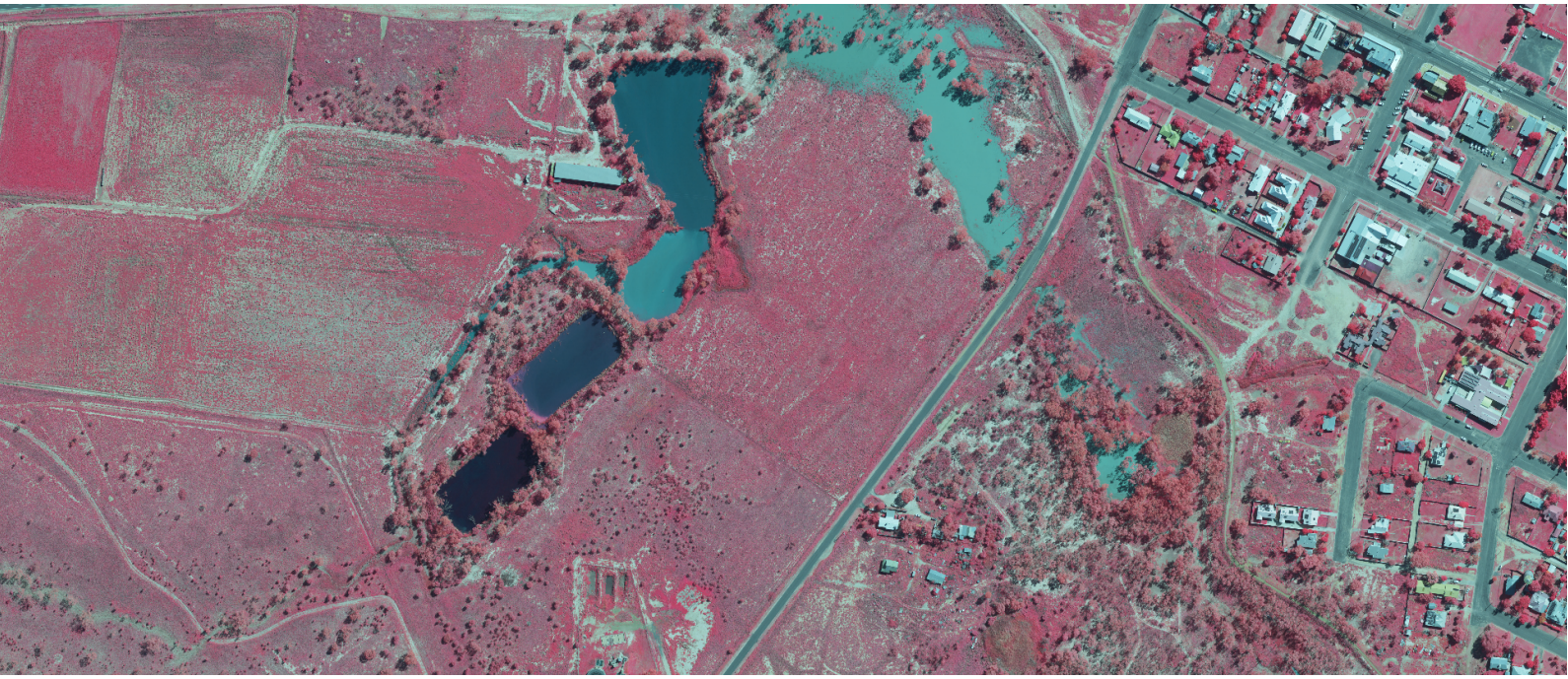
G5

PROJECT USER MANUAL



Semantic Segmentation Hybrid Pipeline (XGB-GMM)

Aerial Imagery Initiative

Contents

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 Extreme Gradient Boosted Trees (XGBoost) and Gaussian Mixture Model (GMM) algorithms to generate a hybrid semantic segmentation model to generate flood extent maps for use in a GIS software environment. 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:

- Load third party modules
- Setup environment variables
- Import libraries and initialise S3 connections
- Download dataset
- Image preprocessing and feature extraction
- XGBoost Predictions
- Remove false positives (if necessary)
- GMM Predictions
- Merge binary images
- Extract contours
- 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.
Previous pipelines used a lifecycle configuration for installing GIS-related dependencies. Due to the new technique of combining binary images, so as to output a single shapefile object, this pipeline requires manual installation of these libraries along with a previous version of OpenCV. The code is shown in Figure 1.

```
#conda commands for environment setup
%conda install -c conda-forge gdal fiona rasterio

#pip commands in case conda cant get the required packages
!pip install 'opencv-python>=4.5.3.56' # required as older versions fail opening some jp2 images
```

Figure 1 - cell to install gdal and fiona

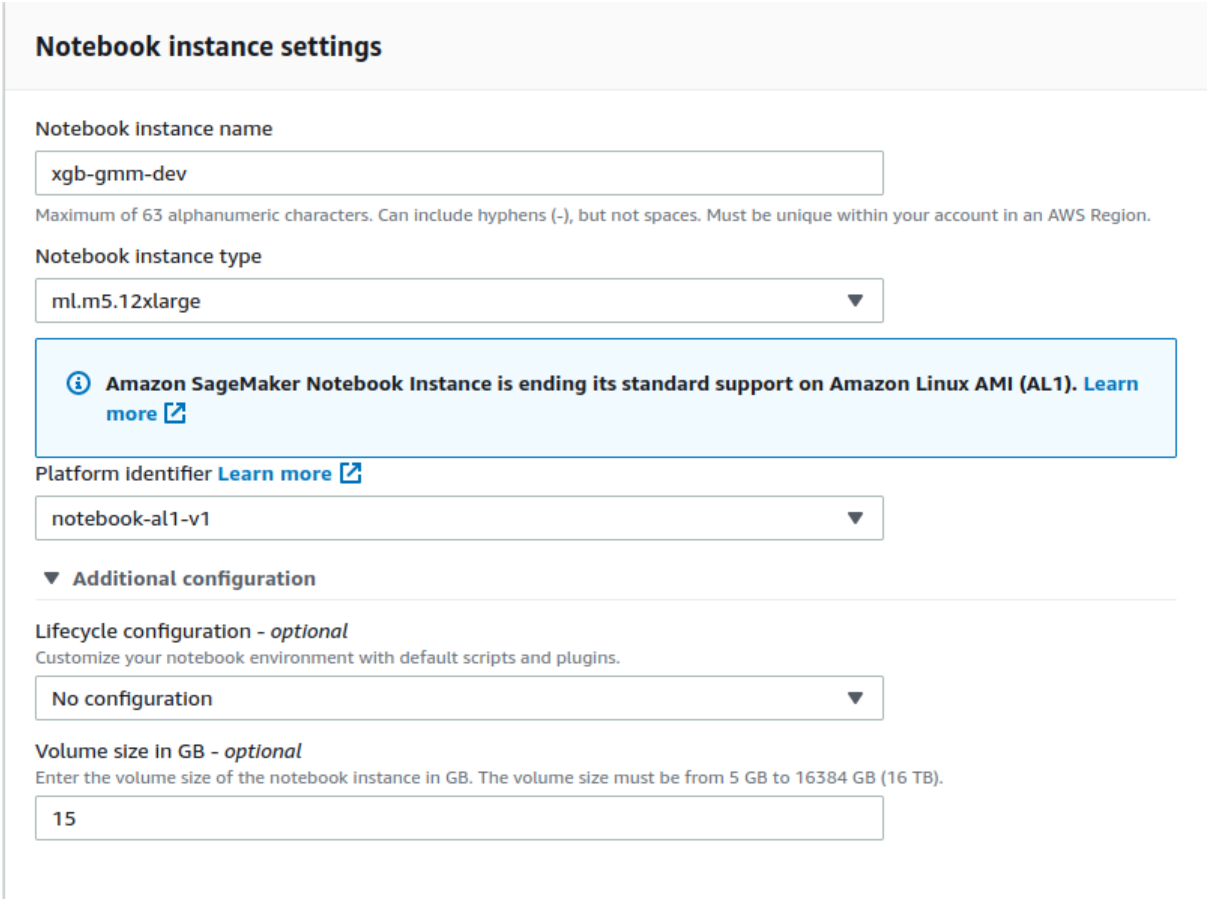


Figure 2 - Notebook configuration settings

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	XGB-GMM-dev
Notebook instance type	ml.r5.12xlarge
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/gmm-xgb-pipeline/. This repository must be cloned into an AWS SageMaker environment.

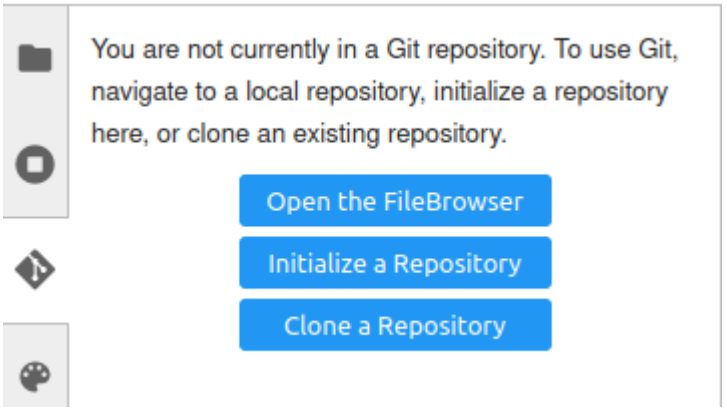


Figure 3 - SageMaker Git Directory

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

```
flood-extent-extraction/
├── App/
│   ├── XGB-GMM Extraction Method/
├── Documentation/
├── Images/
│   ├──lower_clarence
├── 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 XGB-GMM Semantic Segmentation pipeline can be found at

```
flood-extent-extraction/
├─ App/
│  ├─ XGB-GMM Extraction Method/
│  │    ├─XGB_GMM_Semantic_Segmentation.ipynb
```

The first few cells download modules and import libraries. The forth cell labelled *Notebook Settings* 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

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

XGB & GMM Model Settings

- **xgb_model_path** - String: path of trained XGB model
- **gmm_model_path** - String: path of trained GMM 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 this cell is configured properly, all subsequent cells can be executed without further user input except for *6.1-Remove False Positives*. The user must first determine if this step is required and uncomment this cell. All cells can be executed automatically by selecting *Run > Run All Cells*. The large-scale steps are as follows.

1. Setup

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.

```
from osgeo import gdal
from osgeo import osr
import fiona
import rasterio
from rasterio.io import MemoryFile
import rasterio.merge

from keras.models import Model
from keras.applications.vgg16 import VGG16

import tensorflow as tf
import keras
import lightgbm as lgb

from IPython.display import clear_output

role = get_execution_role()
region = boto3.Session().region_name
sess = sagemaker.Session()

region = boto3.Session().region_name
smclient = boto3.Session().client('sagemaker')

print(region)
print(smclient)
```

```
ap-southeast-2
<botocore.client.SageMaker object at 0x7f35b737be80>
CPU times: user 148 ms, sys: 27.4 ms, total: 176 ms
Wall time: 227 ms
```

Figure 4 - Output of initialising S3 connections

2. 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 5 - Download dataset output

3. Image Preprocessing

The input images must be separately processed for the GMM and XGB models. The GMM model expects the input images in their native format. The XGB model however requires the images to be converted from RGB to BGR. This is a constraint of the training process for the XGB classifier. The images are also resized for memory constraints within the notebook.

3. Image Preprocessing

[9]:
```
%%time
cell_error = False
try:

    xgb_img_list = []
    gmm_img_list = []
    img_path = "../../Images/lower_clarence"
    counter = 1
    for file in os.listdir(img_path):
        print(f"opening {file} {counter}/{len(os.listdir(img_path))}")
        img = cv2.imread(os.path.join(img_path, file))
        #img = make_square(img) # not needed for lower clarence dataset
        scale = width/img.shape[0]
        print(f"image width: {img.shape[0]}\nrescale width: {width}\nscale: {scale}\nimage shape: {img.shape}")
        gmm_img = rescale(img.copy(), scale, anti_aliasing=False, multichannel=True)
        gmm_img_list.append(gmm_img)
        aspect = img.shape[1]/img.shape[0]
        xgb_img = cv2.resize(img, (width,int(width/aspect)), interpolation=cv2.INTER_NEAREST)
        xgb_img = cv2.cvtColor(xgb_img, cv2.COLOR_RGB2BGR)
        xgb_img_list.append(xgb_img)
        print(f"resize image shape: {xgb_img.shape}")

        counter += 1
        clear_output(wait=True)

    xgb_img_list = np.array(xgb_img_list)
    gmm_img_list = np.array(gmm_img_list)
except Exception as e:
    logger.error(f"Unable to open images: {e}")
    cell_error = True

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

```
CPU times: user 17min 17s, sys: 2min 13s, total: 19min 31s
Wall time: 18min 45s
```

Figure 6 - Image preprocessing

Next, the XGB pipeline requires a feature extraction step. This changes an input image of dimension n × n × 3, where there are 3 pixel channels, to an n × n × 64 image. This is now an image with a feature vector of dimension 64 for each pixel. This allows for greater accuracy in the classification process for the XGB classifier model.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 1000, 1000, 3)]   0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 1000, 1000, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 1000, 1000, 64)    36928
=================================================================
Total params: 38,720
Trainable params: 0
Non-trainable params: 38,720
_________________________________________________________________
```

[11]:
```
%%time
cell_error = False

try:
    feature_vector = new_model.predict(xgb_img_list)
    print(f"feature vector: {feature_vector.shape}")
    feature_vector = feature_vector.reshape(-1, feature_vector.shape[3])
    print(f"input shape: {feature_vector.shape}")
except Exception as e:
    logger.error(f"Unable to extract feature vector: {e}")
    cell_error = True

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

```
feature vector: (176, 1000, 1000, 64)
input shape: (176000000, 64)
CPU times: user 7min 7s, sys: 2min 18s, total: 9min 25s
Wall time: 32.3 s
```

Figure 7 - feature extraction

Feature extraction is performed by using a pre-trained convolutional neural network, VGG16, with the weights generated from the open source ImageNet dataset. The top three layers of this network are used, as shown in Figure 8, with the third layer outputting an image of shape n × n × 64. This feature vector is then flattened before being passed to the XGB model. The XGB model will later return a flattened array which will be reshaped into the shape of the original input. The numpy reshape operations maintain the order of the pixels so there is no corruption of the final image.

4. XGB Predictions

The output of the XGB model, once reshaped, is an array of all the tiled images with 5 values for each pixel. For the Lower Clarence dataset this results in 176 images of size 1000 × 1000 (the input images were resized as part of the image preprocessing stage) with 5 values for each pixel representing the probability of that pixel being classified as either flood, flood + cloud, trees, buildings, or land respectively as shown in Figure 9.

```
%%time
cell_error = False

xgb_flood = []
xgb_flood_clusters = (0,1)

try:
    num_classes = 5
    xgb_proba = xgb_model.predict_proba(feature_vector)
    xgb_proba = xgb_proba.reshape((xgb_img_list.shape[0], xgb_img_list.shape[1], xgb_img_list.shape[2], num_classes))
    print(f"output shape: {xgb_proba.shape}")

    for output in xgb_proba:
        hp_joined_img = np.greater(output[:,:,xgb_flood_clusters[0]],0.15)
        hp_joined_img = np.logical_or(hp_joined_img, np.greater(output[:,:,xgb_flood_clusters[1]],0.15))
        xgb_flood.append(hp_joined_img)

except Exception as e:
    logger.error(f"Unable to complete xgb predictions: {e}")
    cell_error = True

if cell_error:
    raise SystemExit("Execution Stopped")

for i in range(3):
    fig, ax = plt.subplots(1, 2, sharey=True, figsize=(10,20))
    ax[0].imshow(xgb_img_list[i])
    ax[1].imshow(xgb_flood[i])
```

```
output shape: (176, 1000, 1000, 5)
CPU times: user 18min 11s, sys: 2.61 s, total: 18min 14s
Wall time: 28.5 s
```

Figure 8 - XGB output

5. GMM Predictions

The output of the GMM model is similar to the XGB model, an array of images with 4 values for each pixel correlating to the gaussian distributions that model floods from the GMM model.

6. Combine XGB and GMM output

This step performs a crude averaging six probabilities across the XGB and GMM models, two from XGB and four from GMM. By performing a pixel-wise averaging of flood probabilities, the XGB and GMM models are combined into a single output that obtains higher IoU scores than either model. For results refer to XGBoost and Hybrid Approaches.

```
%%time
cell_error = False

hybrid_img_list = []

try:

    for i in range(len(xgb_proba)):
        xgb_proba_01 = xgb_proba[i][:,:,xgb_flood_clusters[0]]
        xgb_proba_02 = xgb_proba[i][:,:,xgb_flood_clusters[1]]
        gmm_proba_01 = gmm_proba_list[i][:,:,gmm_flood_clusters[0]]
        gmm_proba_02 = gmm_proba_list[i][:,:,gmm_flood_clusters[1]]
        gmm_proba_03 = gmm_proba_list[i][:,:,gmm_flood_clusters[2]]
        gmm_proba_04 = gmm_proba_list[i][:,:,gmm_flood_clusters[3]]
        avg_img = (xgb_proba_01 + xgb_proba_02 + gmm_proba_01 + gmm_proba_02 + gmm_proba_03 + gmm_proba_04) / 6
        flood_img = np.greater(avg_img, 0.15)
        hybrid_img_list.append(flood_img)

    hybrid_img_list = np.array(hybrid_img_list)
    print(f"output shape: {hybrid_img_list.shape}")

except Exception as e:
    logger.error(f"Unable to combine XGB and GMM output: {e}")
    cell_error = True

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

```
output shape: (176, 1000, 1000)
CPU times: user 6 s, sys: 143 ms, total: 6.15 s
Wall time: 6.15 s
```

Figure 9 - combining models

a. 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.

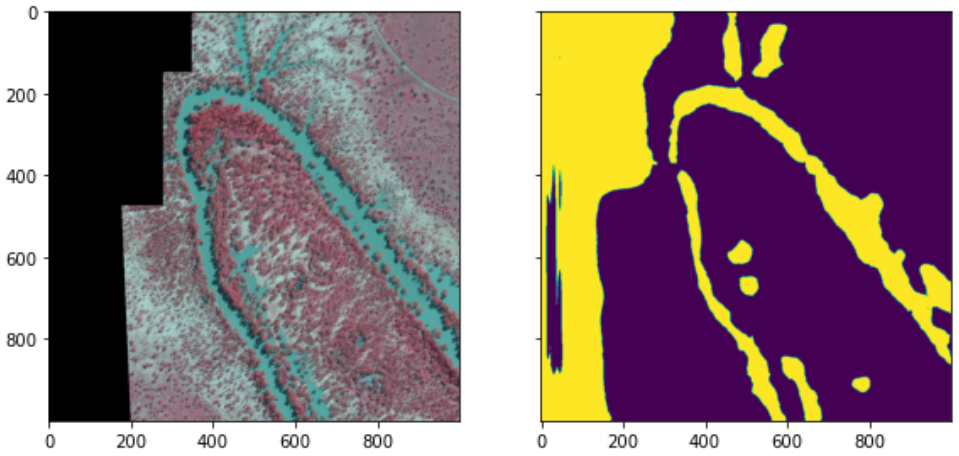


Figure 10 - False positives

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.

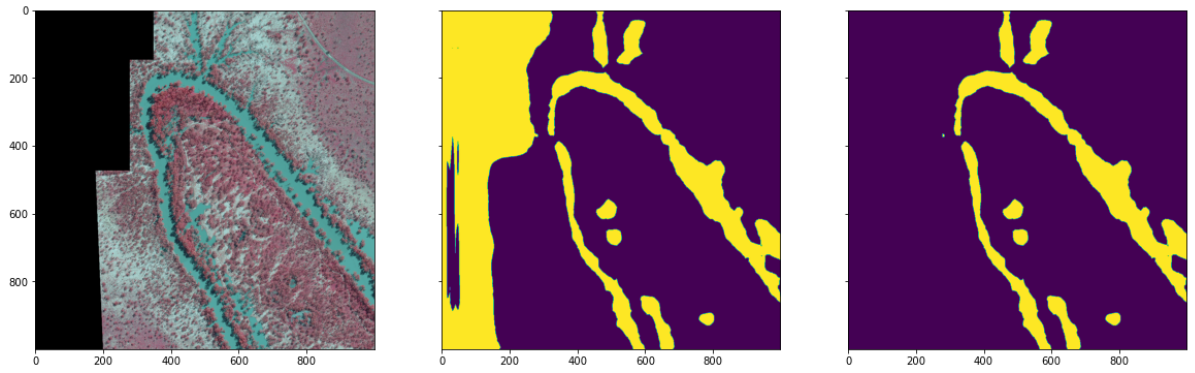


Figure 11 - Removal of false positives

Note that this step will not be needed for all datasets, only those that contain input images as shown in the above figure. The cell for performing this step will be commented out, it is up to users to examine the input before notebook execution to determine if this step is necessary.

7. Combine Binary Images and Extract Contours

This step is performed in both the original GMM and U-Net pipelines. The code here has been taken from those approaches and applied here in the same manner. For a detailed explanation refer to Combining Images / Shapefiles.

8. Generate Shapefiles

This step is again replicated across the other pipelines. It extracts the contours from the large binary image to create a shapefile ready for use in GIS software systems. Processing times may be long. For the Lower Clarence dataset of 176 images resized to 1000 pixels by 1000 pixels, the processing time was approximately 40 minutes.

```
# write to shape file
shp_file_output = os.path.join(output_directory, f"{raster_shp_output_prefix}_{timestamp}.shp")
logger.debug(f"Writing {shp_file_output}")
try:
    with fiona.open(shp_file_output, 'w', 'ESRI Shapefile', schema=schema, crs=merged_dataset.crs) as shp_file:
        logger.debug(f"Shape file espg: {shp_file.crs}")
        shp_file.write({
            'geometry' : {'type':'MultiPolygon',
            'coordinates': [multi_polygon]}, # Here the xyList is in brackets
            'properties': {
                'tag': ' flood extent'
            }
        })

except Exception as e:
    logger.error(f"Unable to write shape file {shp_file_output}: {e}", exc_info=True)
    cell_error = True
else:
     logger.debug(f"No polygons. No shape file will be generated")

if cell_error:
    logger.warn(f"Finished converting polygons to shape file with errors")
else:
    logger.info(f"Finished converting polygons to shape file")
```

```
[2021-10-17 03:28:31,151] NOTEBOOK_LOG INFO - Started converting contours / polygons to shape file
[2021-10-17 04:08:16,828] NOTEBOOK_LOG INFO - Finished converting polygons to shape file
```

Figure 12 - Shapefile generation

The remaining step is to compress the output to zip and cleanup resources within the notebook environment. The zip folder can be downloaded from the directory within the notebook environment and directly dropped into any GIS software environment.

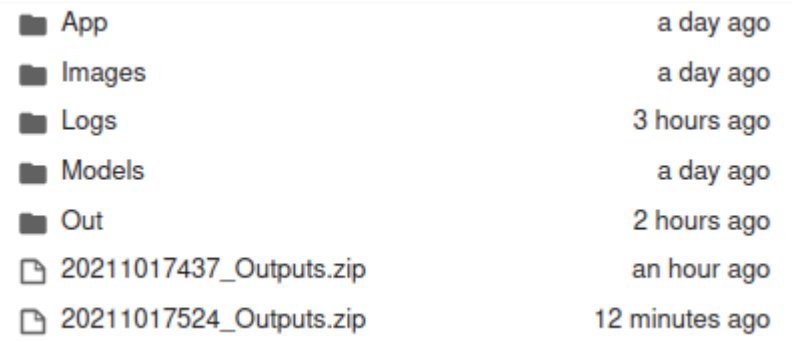


Figure 13 - zip output