


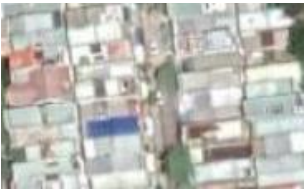



Technical report


1. Image labelling technology report

1.1 Classification System Development

As the goal of this project is to detect which areas in Da Nang become industrial buildings and residential buildings with the change of time, this project divides the study area into industrial buildings, residential buildings, and other area, which are three categories. As shown in *Table 1*.

Table 1. Classification system developed

Serial Number	Classification Name	Sample Image
1	Industrial building	
2	Residential building	
3	Other area: Forest	
4	Other area: Water	
5	Other area: Bare land	

6	Other area: Road	
---	------------------	--

1.2 Image Labelling

To get accurate change detection results, and if the samples are labelled completely manually, there is a great demand for time and energy., in this project, the image labelling technology adopted is a semi-automatic labelling technology based on the combination of manual interpretation and computer automatic classification based on spectrum. Using this technology, we need to find and use a convenient sample labelling tool to ensure the label quality. There are three requirements for tagging tools: brief introduction of software interface, simple operation and allowing users to carry out some secondary development. According to our own needs, we finally choose ENVI to label samples. Labeling software is shown in *Fig. 1*.

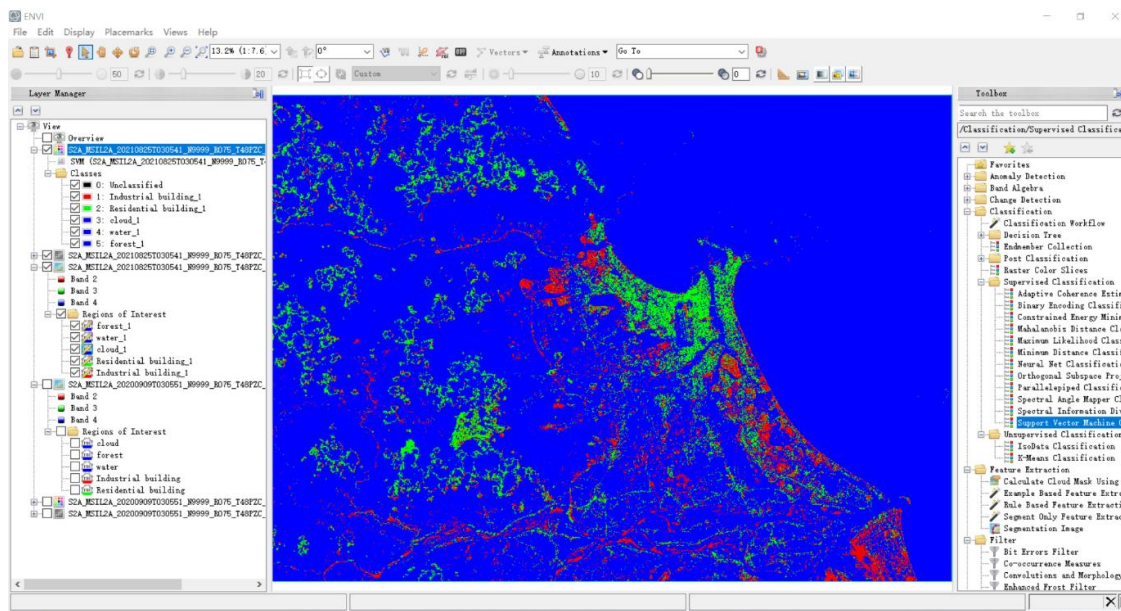


Fig. 1 Labelling software

Because we have acquired the low-resolution satellite images in TIF file format and the high-resolution images in non-TIF file format, we need to deal with the marking part according to the situation.

For low-resolution satellite images in TIF file format, there are three main steps in the image annotation preprocessing process:

Step 1: manually label sample.

Step 2: Calculate the separability of the sample.

Step 3: Select a classifier to carry out automatic computer classification based on spectrum.

After the step of calculating the separability of samples, the separability among various sample types can be observed, which is represented by the parameter Jeffries-Mastusita, Transformed Divergence", and the value of this parameter is between 0 and 2.0. The larger the parameter value, the better the separability between

samples. When the value of the parameter is less than 1, it is necessary to consider combining two kinds of samples into one kind of samples. *Fig.2* shows the separability calculation results of industrial building sample types and residential building sample types. We can find that the numerical value is about equal to 1.7, which is greater than 1.5, indicating that the selected samples of industrial buildings and residential buildings have a good separation effect.

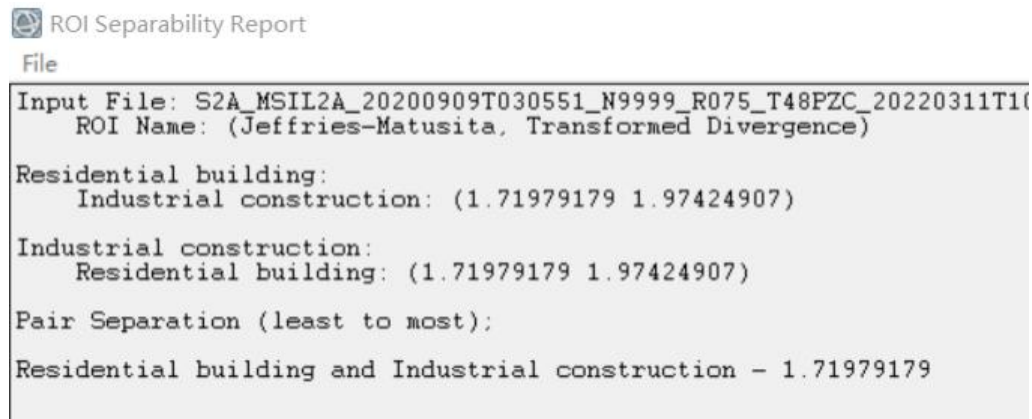


Fig.2 ROI separability report for low-resolution satellite images in TIF file format

To better detect the change, the classifier chose the support vector machine. SVM belongs to a method of combining statistical theory and machine learning theory knowledge. It can maximize the distance between classes, and can automatically generate support vectors, which are based on the ability to make a great difference in classification.

The main change process of remote sensing satellite images when using semi-automatic labelling technology for image labelling is shown in *Fig. 3*. Mark industrial buildings with red (255,0,0), residential buildings with green (0,255,0) and other areas with blue (0,0,255).

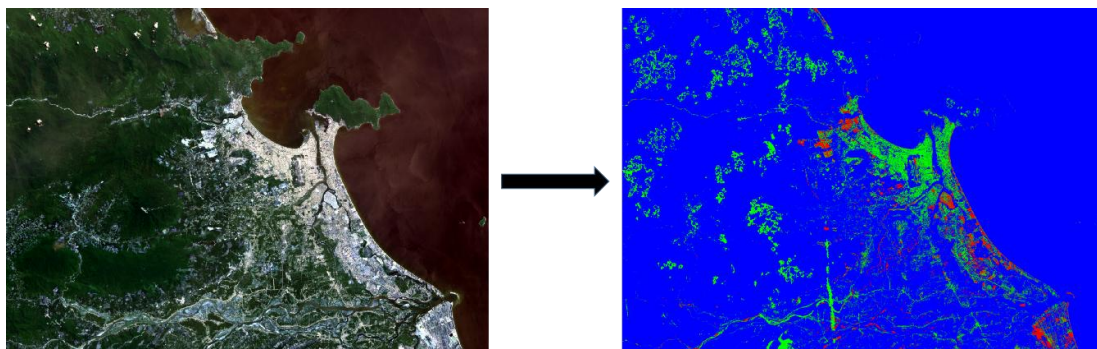


Fig. 3 The main change process of low-resolution satellite images in TIF file format

For high-resolution images in non-TIF file format, there are two main steps in the image annotation preprocessing process:

Step 1: manually label sample.

Step 2: Calculate the separability of the sample.

Fig.4 shows the separability calculation results of industrial building sample types and residential building sample types. We can find that the numerical value is about equal to 2.0, which is much greater than 1.5, indicating that the selected samples of

industrial buildings and residential buildings have a perfect separation effect. It can be observed that the numerical value of Jeffries-Mastusita of high-resolution satellite images is larger than that of low-resolution satellite images, indicating that the higher the resolution, the better the marking effect.

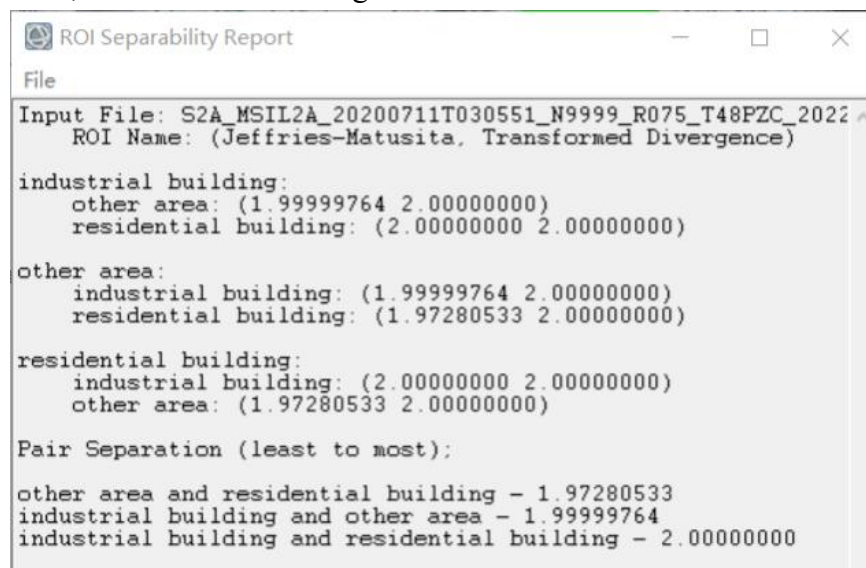


Fig.4 ROI separability report for high-resolution images in non-TIF file format

The main change process of remote sensing satellite images is shown in Fig. 5 when manually marking images. Industrial buildings are marked with red (255,0,0), residential buildings with green (0,255,0) and other areas with black (0,0,0).

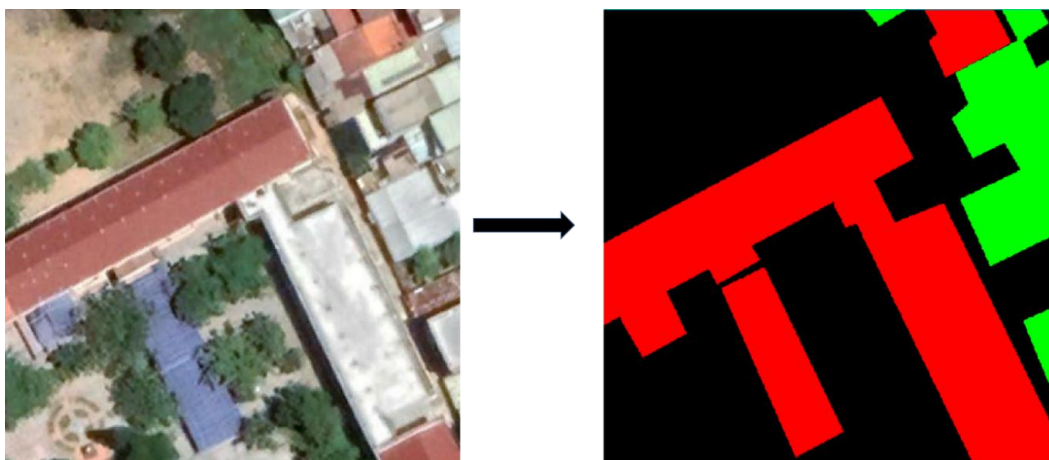


Fig. 5 The main change process of high-resolution images in non-TIF file format

1.3 Labeling software download address and labeling technology hand-in-hand instruction video

Labeling software download address: <https://www.envi-met.com/>

Labeling technology hand-in-hand instruction video:

https://www.bilibili.com/video/BV12x411j7q5?p=11&vd_source=dc20f8b467cb0b409602bde6c4452223

2. Project code technical report

The main purpose of this technical report is to enable the user to use this change detection smoothly.

Firstly for the first part of the code, the main function is to connect to the resources on google cloud drive and empty the various folders that need to be used for change detection.

Final change detection

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

Empty the folder

```
[ ] import shutil
    import os
    shutil.rmtree('/content/drive/MyDrive/capstone/P_png')
    os.mkdir('/content/drive/MyDrive/capstone/P_png')
    shutil.rmtree('/content/drive/MyDrive/capstone/P')
    os.mkdir('/content/drive/MyDrive/capstone/P')
    shutil.rmtree('/content/drive/MyDrive/capstone/P_cut')
    os.mkdir('/content/drive/MyDrive/capstone/P_cut')
    shutil.rmtree('/content/drive/MyDrive/capstone/diff')
    os.mkdir('/content/drive/MyDrive/capstone/diff')
    shutil.rmtree('/content/drive/MyDrive/capstone/changemap')
    os.mkdir('/content/drive/MyDrive/capstone/changemap')
    shutil.rmtree('/content/drive/MyDrive/capstone/cleanchangemap')
    os.mkdir('/content/drive/MyDrive/capstone/cleanchangemap')
```

The second part of the code is the collection of Sentinel II satellite remote sensing images. If there is already an image class that needs to be detected skip this step, if it is like the collection of Sentinel II satellite remote sensing images, you need to follow the following three steps.

The first step is the download of the Sentinel data, which consists of the following five parts, just modify the path according to the actual storage location.

- 1) ESA website: <https://scihub.copernicus.eu/dhus/#/home>
- 2) Register an account or log in
- 3) Select target region and data type (S2A_MSIL1C)
- 4) Add data to the cart
- 5) Download the .meta4 file and save it to the local PC

```
[ ] from xml.dom.minidom import parse
    from data_downloader import downloader
```

```
[ ] folder_out = r'D:\sentinal2_data' # A file that store output data
    url_file = r'D:\sentinal2_data\meta\products.meta4' # Downloaded the products.meta4

    data = parse(url_file).documentElement
    urls = [i.childNodes[0].nodeValue for i in data.getElementsByTagName('url')]

    downloader.download_datas(urls, folder_out)
```

The second step is the data Processing: atmospheric correction. This part requires

additional software for the operation, the operation consists of

1) Download Sen2Cor software :

<http://step.esa.int/main/third-party-plugins-2/sen2cor/>

2) Use command prompt to switch the path to the S2A_MSIL1C data

3) Enter L2A_Process file name --resolution=10 --refresh command to process S2A_MSIL1C data and convert it to S2A_MSIL2A data

The third step is the data format conversion, which is used to save the terminal data in Geotif format for subsequent processing. Care needs to be taken in the operation to modify to the actual file path.

```
if __name__ == "__main__":
    from osgeo import gdal
    SAFE_Path = (r'D:\sentinal2_data\L2A') # Tif file storage path
    data_list = glob.glob(SAFE_Path + "\\*.SAFE")

    #filename = ('E:\\RSDATA\\Sentinel2\\L2A\\S2A_MSIL2A_20210220T024731_N9999_R132_T51STA_20210306T024402.SAFE\\WTD_MSIL2A.xml')
    for i in range(len(data_list)):
        data_path = data_list[i]
        filename = data_path + "\\WTD_MSIL2A.xml"
        S2tif(filename)
        print(data_path + "-----Turn tif successfully ")
    print("-----Conversion end-----")
```

The third part of the code is the cropping and formatting of the images, which needs to be modified according to the actual file path. We will put the two images that need to detect changes into the folder P, and then run the code to convert the images in tiff format into png format and store them in the P_png folder, and then we can modify the w-width and h-height to crop them into small images of corresponding size and store them in the P cut file according to the requirements. The returned image names, the number of cuts and the number of rows and columns of the cropped images will be used in unsupervised learning change detection and image stitching.

```
import cv2 as cv
import os
from PIL import Image

def Convert_To_Png_AndCut(dir):
    files = os.listdir(dir)
    ResultPath1 = "/content/drive/MyDrive/capstone/P.png" # Define path to save format after conversion
    ResultPath2 = "/content/drive/MyDrive/capstone/P_cut" # Define save path after cropping
    file_name = []

    for file in files:
        a, b = os.path.splitext(file) # File name of split image map
        file_name.append(a)
        this_dir = os.path.join(dir + file) # Build Save Path + Filename

        img = cv.imread(this_dir, 1) # Read tif images

        cv.imwrite(ResultPath1 + a + ".png", img) # Save as png

        height = img.shape[0] #Get width and height
        width = img.shape[1]
        #Defining cut size
        w = 650 # Width
        h = 650 # Height
        _id = 1 # Crop result save file name: 0 ~ N in ascending order
        i = 0
        row = 0
        column = 0
        while (i + h < height): # Control height and leave out excess fixed size sum of image
            j = 0
            row = row + 1
            while (j + w < width): # Control width and leave out excess fixed size sum of image along
                column = column + 1
                cropped = img[i + h, j:j + w] # Cropped coordinates are [y0:y1, x0:x1]
                cv.imwrite(ResultPath2 + a + "_" + str(_id) + h, cropped)
                _id += 1
                j += w
                i += h + h
            column = column / row
            return file_name[0], file_name[1], _id - 1, row, int(column)
```

```
if __name__ == '__main__':
    _path = "/content/drive/MyDrive/capstone/P/" # Remote sensing tiff image location path
    # Crop image map
    after, before, number, row, column = Convert_To_Png_AndCut(_path)
```

The fourth part of the code is unsupervised learning methods for change detection.

For the first three methods `find_vector_set`, `find_FVS` and clustering can be defined and run directly, in defining `find_PCAKmeans` we need to modify the actual path where the file is stored. The `diff` folder contains the absolute value of the pixel difference between the two images, the `changemap` folder contains the clustering of the images in the `diff` folder, setting the more variable areas to white and the rest to black. the `cleanchangemap` folder contains the eroding of the images in the `changemap` folder. `erode` operation to remove areas of small variation.

```
def find_PCAKmeans(imagepath1, imagepath2, after, i):

    print('Operating')

    image1 = cv2.imread(imagepath1)
    image2 = cv2.imread(imagepath2)
    image1=image1[:, :,0]
    image2=image2[:, :,0]
    print(image1.shape, image2.shape)
    new_size = np.asarray(image1.shape) / 5
    new_size = new_size.astype(int) * 5
    image1 = resize(image1, (new_size))
    image2 = resize(image2, (new_size))
    diff_image = abs(image1 - image2)
    imageio.imwrite('/content/drive/MyDrive/capstone/diff/' + after + '_' + str(i) + '.png', diff_image)
    print('\nBoth images resized to ', new_size)

    vector_set, mean_vec = find_vector_set(diff_image, new_size)
    pca = PCA()
    pca.fit(vector_set)
    EVS = pca.components_
    FVS = find_FVS(EVS, diff_image, mean_vec, new_size)
    print('\ncomputing k means')

    components = 3
    least_index, change_map = clustering(FVS, components, new_size)
    change_map[change_map == least_index] = 255 ## change map formula
    change_map[change_map != 255] = 0
    change_map = change_map.astype(np.uint8)
    kernel = np.asarray(((0, 0, 1, 0, 0),
                          (0, 1, 1, 1, 0),
                          (1, 1, 1, 1, 1),
                          (0, 1, 1, 1, 0),
                          (0, 0, 1, 0, 0)), dtype=np.uint8)

    cleanChangeMap = cv2.erode(change_map, kernel)
    imageio.imwrite('/content/drive/MyDrive/capstone/changemap/' + after + '_' + str(i) + '.png', change_map)
    imageio.imwrite('/content/drive/MyDrive/capstone/cleanchangemap/' + after + '_' + str(i) + '.png', cleanChangeMap)
```

Finally automated batch unsupervised learning of change detection by modifying the actual storage path of the cropped images.

```
path = "/content/drive/MyDrive/capstone/P cut/"
for i in range(1, number + 1):
    a= path + before + '_' + str(i) + '.png'
    b= path + after + '_' + str(i) + '.png'
    find_PCAKmeans(a, b, after, i)
```

The fifth part of the code is Image stitching and multiply, the main purpose of which is to display the results by virtue of bringing the detected small image to the original size.

As shown in the diagram, where `IMAGES_PATH` is the path where the small images are stored before stitching, and `IMAGE_SAVE_PATH` is the path where the large images are stored after stitching, these need to be modified according to the actual path. The annotated row and column are used to determine the number of rows and columns by virtue of the large image, here as there is a return in the crop step can be commented, but also separately to modify the stitching. And in the stitching before the

need to sort the pictures of the folder, here is based on the naming of the cropped pictures to develop the sorting rules, if the stitching of other pictures can develop their own sorting rules.

The stitching code for this project stitches the images in the diff, changemap, cleanchangmap and label image folders separately, and can be skipped if the images in the label image are not available.

```
Stitching the images in the diff folder

import PIL.Image as Image
import os

IMAGES_PATH = '/content/drive/MyDrive/capstone/diff/'
IMAGES_FORMAT = ['.jpg', '.JPG', '.jpeg', '.png']

#row = 7
#column = 10

IMAGE_SIZE = 650# Size of each small image

IMAGE_ROW = row # Image spacing, i.e. how many rows there are when combined into one image
IMAGE_COLUMN = column # Image spacing, i.e. how many columns there are when combined into one image
IMAGE_SAVE_PATH = '/content/drive/MyDrive/capstone/stich result/' # Address after image conversion

# Get the names of all the images under the address of the image set
image_names = [name for name in os.listdir(IMAGES_PATH) for item in IMAGES_FORMAT if
                os.path.splitext(name)[1] == item]

x=image_names
a='.'
b='_'

image_names.sort(key = lambda x:int(x.split(a)[0].split(b)[-2])) # Image sorting
print(len(image_names))

def image_compose():
    to_image = Image.new('RGB', (IMAGE_COLUMN * IMAGE_SIZE, IMAGE_ROW * IMAGE_SIZE)) # Create a new image
    # Iterate through the loop, pasting each image into the corresponding position in order
    for y in range(1, IMAGE_ROW + 1):
        for x in range(1, IMAGE_COLUMN + 1):
            from_image = Image.open(IMAGES_PATH + image_names[IMAGE_COLUMN * (y - 1) + x - 1]).resize(
                (IMAGE_SIZE, IMAGE_SIZE),Image.ANTIALIAS)
            to_image.paste(from_image, ((x - 1) * IMAGE_SIZE, (y - 1) * IMAGE_SIZE))
    to_image.save(IMAGE_SAVE_PATH + 'diff_merged.png') # Save new image
image_compose() # Calling functions
```

Finally there is the multiplying of the images, if there are labeled images we can run this step, otherwise we can skip it. Label1 and mask are the paths of the two stacked images respectively, and im3 is the path to save the images after stacking, which needs to be modified according to the actual path.

```
Multiplying diff images and labelled images

# Importing Image and ImageChops module from PIL package
from PIL import Image, ImageChops
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
Image.MAX_IMAGE_PIXELS = None
# create label
label = Image.open(r"/content/drive/MyDrive/capstone/stich result/diff_merged.png")

# ValueError: images do not match, need to match image mode
label = label.convert("RGB")

# creat mask
mask = Image.open(r"/content/drive/MyDrive/capstone/stich result/labelled_merged.png")
mask = mask.convert("RGB")

# applying multiply method
im3 = ImageChops.multiply(label,mask)
im3.save('/content/drive/MyDrive/capstone/stich result/multiply_diff.png')
```


The last part of the code is the change detection of the Unet++ method. Run train.py to generate an evaluation image to update the model, if you want to use it directly you can ignore this step and use the trained model for change detection.

Note that to use the Unet++ method you need to modify the corresponding paths in the Lamboise-Master files eval.py, image.py, train.py and predict.py according to the actual paths of the files.

```
[ ] !python /content/drive/MyDrive/Lamboise-Master/train.py
```

Next for the Unet++ method change detection run, the statement is !python path0 -i path1 -o path2, where path0 refers to the path to predict.py, path1 refers to the path to the input file and path2 refers to the path to the output file. py, path1 refers to the path of the input file, path2 refers to the path of the output file, and the files need to be arranged as follows (below the file)

instance_1

before.png

after.png

The change detection result predicted.png will then be generated in the corresponding folder.

```
Run predict.py to do the architecture change detection (UNet++)
!python path0 -i path1 -o path2
(whole path is needed on colab)

• path0 refers to the path of predict.py
• path1 refers to the path of the input file
• path2 refers to the path of the output file
• files needed to be arranged as follow (under the file):
  ◦ instance_1
    • before.png
    • after.png
    • predicted.png (result 1)
  ◦ instance_2
    • before.png
    • after.png
    • predicted.png (result 2)

[ ] !python /content/drive/MyDrive/capstone/Lamboise-Master/predict.py -i /content/drive/MyDrive/capstone/Unet++/change_detection -o /content/drive/MyDrive/capstone/Unet++/change_detection
```

Finally, after the appropriate stitching operations, we will process the predicted image to make the result more legible, including enhancing contrast and brightness. Just follow the actual file path and make the changes.

```
If first stitching then process the images (recommended):

#read the image
im = Image.open("/content/drive/MyDrive/capstone/Unet++/report/mask/compose_image_mask.png") # input path

#image contrast enhancer
enhancer = ImageEnhance.Contrast(im)

factor = 3 #increase contrast
im_output = enhancer.enhance(factor)
im_output.save('/content/drive/MyDrive/capstone/Unet++/report/mask/more-contrast-image.png')

im1 = Image.open("/content/drive/MyDrive/capstone/Unet++/report/mask/more-contrast-image.png")

#image brightness enhancer
enhancer = ImageEnhance.Brightness(im1)

factor = 3 #brightens the image
im_output = enhancer.enhance(factor)
im_output.save('/content/drive/MyDrive/capstone/Unet++/report/mask/brightened-image.png')
```