**QUANTUM IMAGE PROCESSING**

**Instruction Manual**

**Student Name: Sanjay Govindaraju**

**Student ID: 2073073**

**Student Name: Dhanush Vommi**

**Student ID: 2102904**

Image Segmentation for flood image 1:

Save the below image as floodorg.jpg



Reduce the size of the image with the below code:

Name: floodImageResize.py

import numpy as np

import matplotlib.pyplot as plt

import cv2

from PIL import Image

img = cv2.imread('floodorg.jpg')

type(img)

img.shape

fix\_img = cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)

plt.imshow(fix\_img)

new\_image = cv2.resize(fix\_img,(40,40))

plt.imshow(new\_image)

cv2.imwrite('flood1.jpg',new\_image)

Now the new image is stored as a 40x40 dimension image in the same file

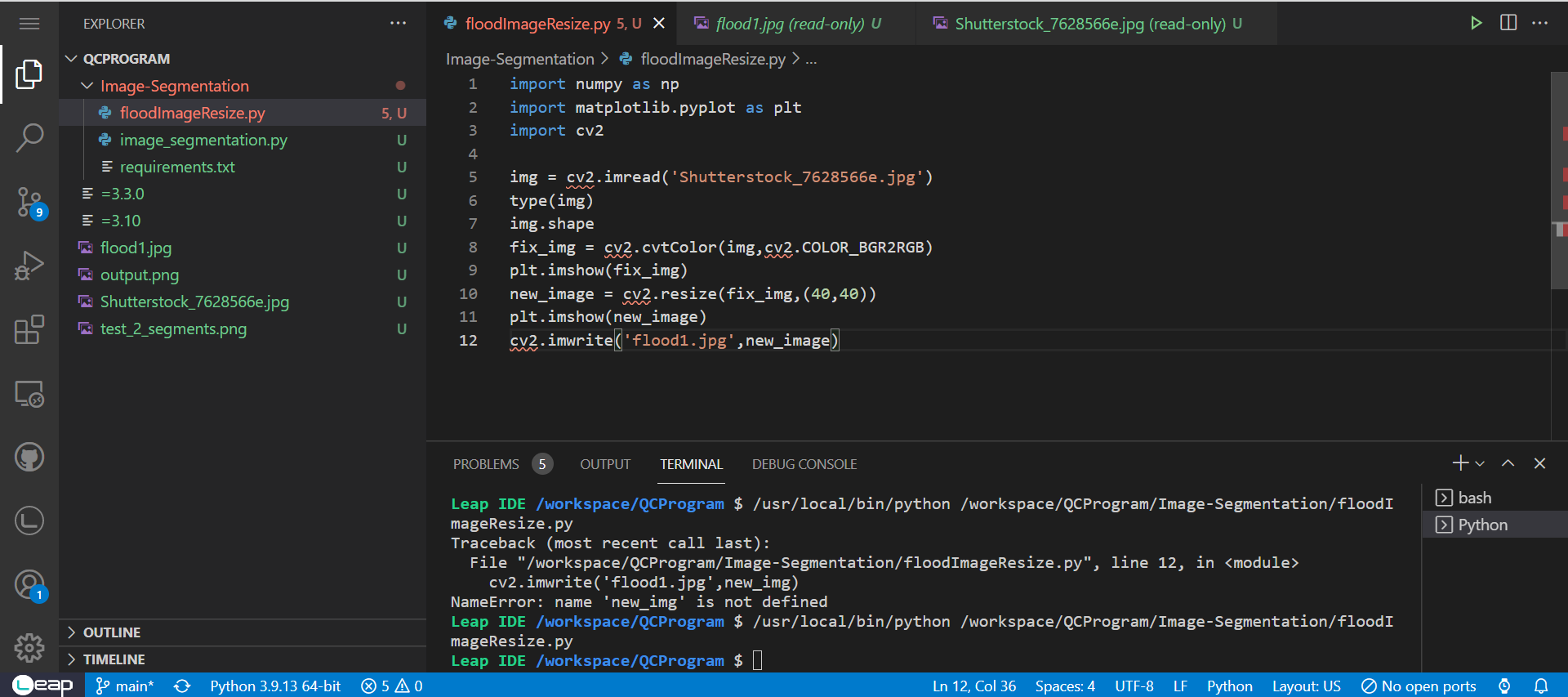
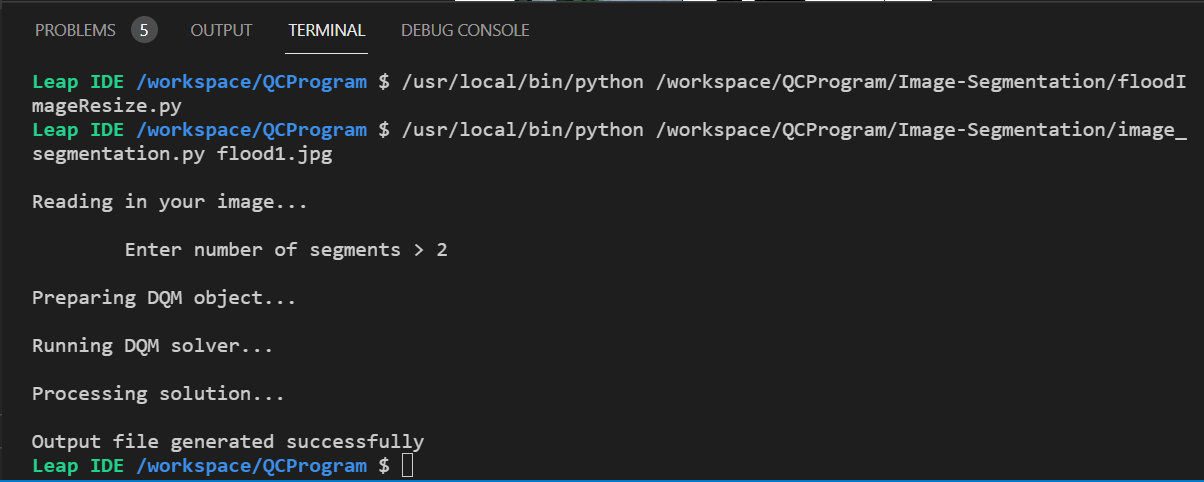


Image stored as 40x40 image

Now Pass this through the DQM sampler program for segmentation:



Program :

Name: image\_segmentation.py

import cv2

import sys

import numpy as np

import matplotlib

matplotlib.use("agg")    # must select backend before importing pyplot

import matplotlib.pyplot as plt

from dimod import DiscreteQuadraticModel

from dwave.system import LeapHybridDQMSampler

# Define our weight function

def weight(a, b, img):

    \_, cols, \_ = img.shape

    diff = img[int(a/cols), a%cols, :] - img[int(b/cols), b%cols, :]

    return np.sum(np.square(diff))

# Convert single index into tuple indices

def unindexing(a):

    rows, cols, \_ = img.shape

    y1 = a % cols

    x1 = int(a/cols)

    return (x1, y1)

if len(sys.argv) > 1:

    # Read in image file specified

    data\_file\_name = sys.argv[1]

    random = False

    print("\nReading in your image...")

    img = cv2.imread(data\_file\_name)

    response\_2 = input("\n\tEnter number of segments > ")

    try:

        num\_segments = int(response\_2)

    except ValueError:

        print("Must input an integer.")

        num\_segments = int(input("\n\tEnter number of segments > "))

else:

    # Generate a random image with segments

    print("\nCreating random image...")

    random = True

    # Collect user input on size of problem

    response\_1 = input("\n\tEnter image dimensions > ")

    try:

        dims = int(response\_1)

    except ValueError:

        print("Must input an integer.")

        dims = int(input("\n\tEnter image dimensions > "))

    response\_2 = input("\n\tEnter number of segments > ")

    try:

        num\_segments = int(response\_2)

    except ValueError:

        print("Must input an integer.")

        num\_segments = int(input("\n\tEnter number of segments > "))

    img = np.zeros((dims, dims, 3), np.uint8)

    img\_rows = np.sort(np.random.choice(dims, num\_segments, replace=False))

    img\_cols = np.sort(np.random.choice(dims, num\_segments, replace=False))

    for num in range(num\_segments-1):

        color = np.random.randint(0, 255, 3)

        img[img\_rows[num]:, img\_cols[num]:, :] = color

# Create a version of the image data that is signed, so that subtraction will

# not wrap around when computing differences.

img\_signed = img.astype(int)

# Build the DQM and set biases according to pixel similarity

print("\nPreparing DQM object...")

rows, cols, \_ = img.shape

linear\_biases = np.zeros(rows\*cols\*num\_segments)

case\_starts = np.arange(rows\*cols) \* num\_segments

num\_interactions = rows \* cols \* (rows\*cols - 1) \* num\_segments / 2

qb\_rows = []

qb\_cols = []

qb\_biases = []

for i in range(rows\*cols):

    for j in range(i+1, rows\*cols):

        for case in range(num\_segments):

            qb\_rows.append(i\*num\_segments + case)

            qb\_cols.append(j\*num\_segments + case)

            qb\_biases.append(weight(i, j, img\_signed))

quadratic\_biases = (np.asarray(qb\_rows), np.asarray(qb\_cols), np.asarray(qb\_biases))

dqm = DiscreteQuadraticModel.from\_numpy\_vectors(case\_starts, linear\_biases, quadratic\_biases)

# Initialize the DQM solver

print("\nRunning DQM solver...")

sampler = LeapHybridDQMSampler()

# Solve the problem using the DQM solver

sampleset = sampler.sample\_dqm(dqm, label='Image Segmentation')

# Get the first solution

sample = sampleset.first.sample

print("\nProcessing solution...")

im\_segmented = np.zeros((rows, cols))

for key, val in sample.items():

    x, y = unindexing(key)

    im\_segmented[x,y] = val

if random:

    row\_indices = [1+i for i in range(rows-1)]

    row\_indices.append(0)

    im\_segmented\_rowwrap = im\_segmented[row\_indices, :]

    col\_indices = [1+i for i in range(cols-1)]

    col\_indices.append(0)

    im\_segmented\_colwrap = im\_segmented[:, col\_indices]

    im\_seg\_rowdiff = im\_segmented - im\_segmented\_rowwrap

    im\_seg\_coldiff = im\_segmented - im\_segmented\_colwrap

    segmented\_image = np.ones((rows, cols, 3), np.uint8)\*255

    segmented\_image[im\_seg\_rowdiff != 0] = (255, 0, 0)

    segmented\_image[im\_seg\_coldiff != 0] = (255, 0, 0)

else:

    segmented\_image = im\_segmented

    img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

fig, (ax1, ax2) = plt.subplots(1, 2)

ax1.imshow(img)

ax1.axes.xaxis.set\_visible(False)

ax1.axes.yaxis.set\_visible(False)

ax2.imshow(segmented\_image, cmap='Greys')

ax2.axes.xaxis.set\_visible(False)

ax2.axes.yaxis.set\_visible(False)

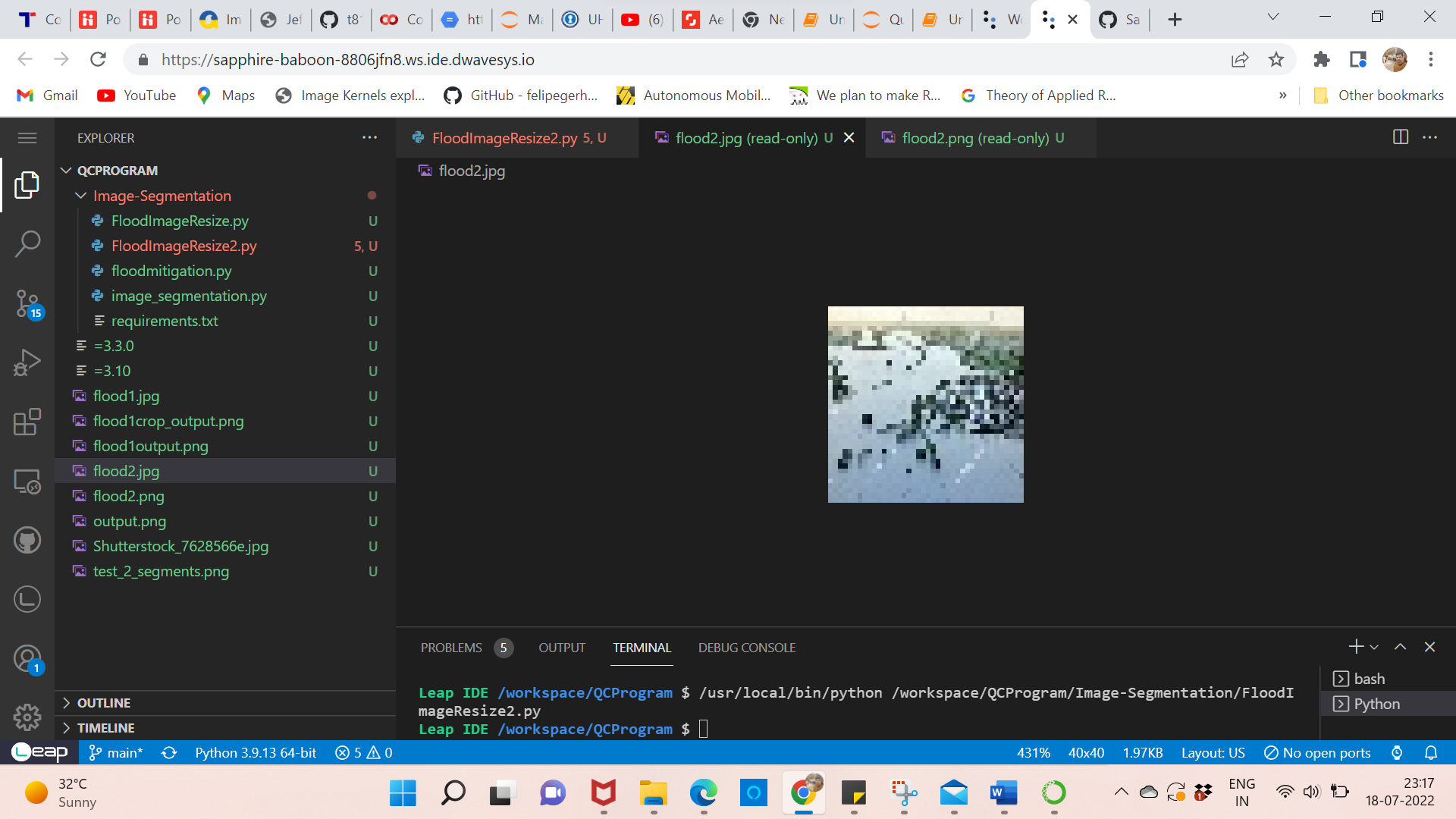
plt.savefig("output.png")

print("\nOutput file generated successfully")

Output:



Save the output as flood1output.png



Saved output image as flood1output.png



Crop the output and calculate the percentage of tree cover and percentage of land cover:

Name: floodmitigation.py

import numpy as np

import matplotlib.pyplot as plt

import cv2

from PIL import Image

#Crop the black and withe segmented output image to work on it

#Crop Dimensions :

left = 350

top  = 150

right = 575

bottom =355

imgcrop = Image.open('flood1output.png')

img2\_crop = imgcrop.crop((left, top, right, bottom))

plt.imshow(img2\_crop)

img2\_crop.save('flood1crop\_output.png')

#Read the image

img3\_gray = cv2.imread('flood1crop\_output.png',0)

plt.imshow(img3\_gray,cmap='gray')

print('gray scale image shape : ',img3\_gray.shape)

white = np.sum(img3\_gray != 0)

print('number of white pixels : ',white)

black = np.sum(img3\_gray == 0)

print('number of white pixels : ',black)

print('total number of pixels: ',img3\_gray.size)

Percentage\_green = (black/img3\_gray.size)\*100

Percentage\_water = (white/img3\_gray.size)\*100

print("Percentage of tree cover in the image is : ",Percentage\_green,"%")

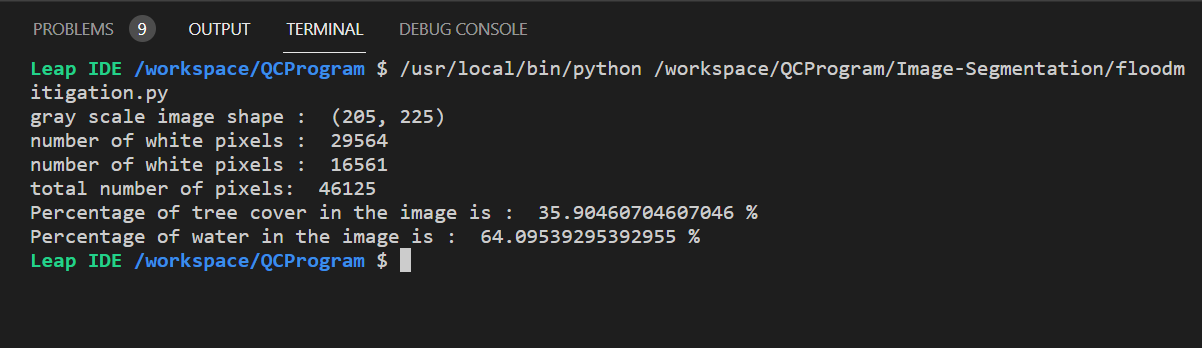
print("Percentage of water in the image is : ",Percentage\_water,"%")

Cropped output is saved as flood1crop\_output.png:

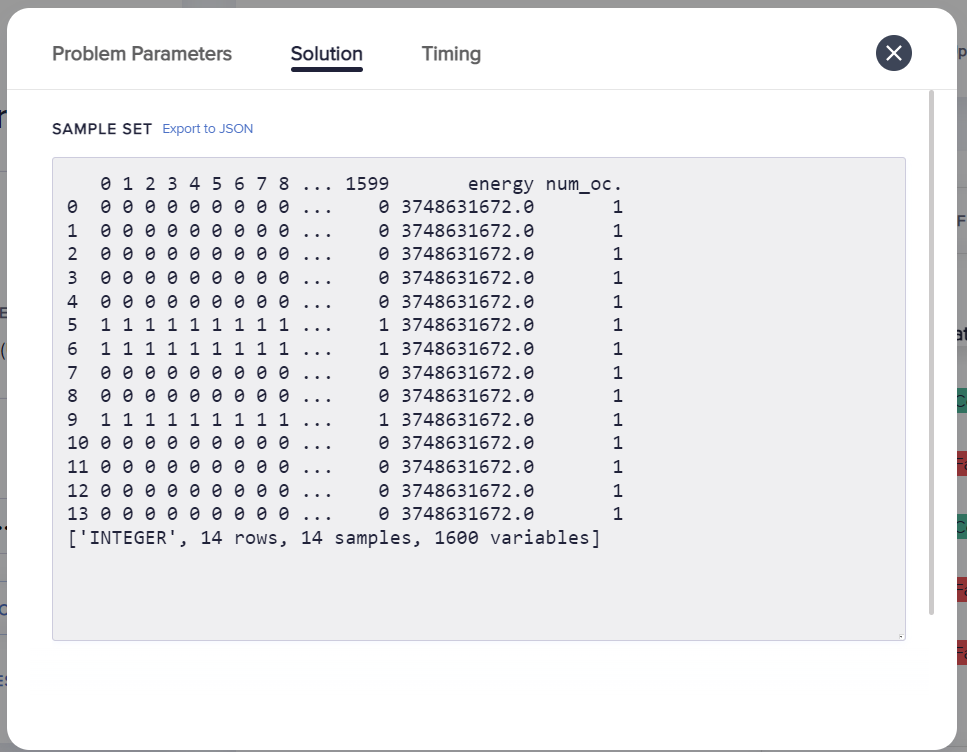
Note: The top sky part is cropped out here as it is irrelevant.



The percentage of tree(black) and water(white) cover is calculated:



Solution Parameters



Timing

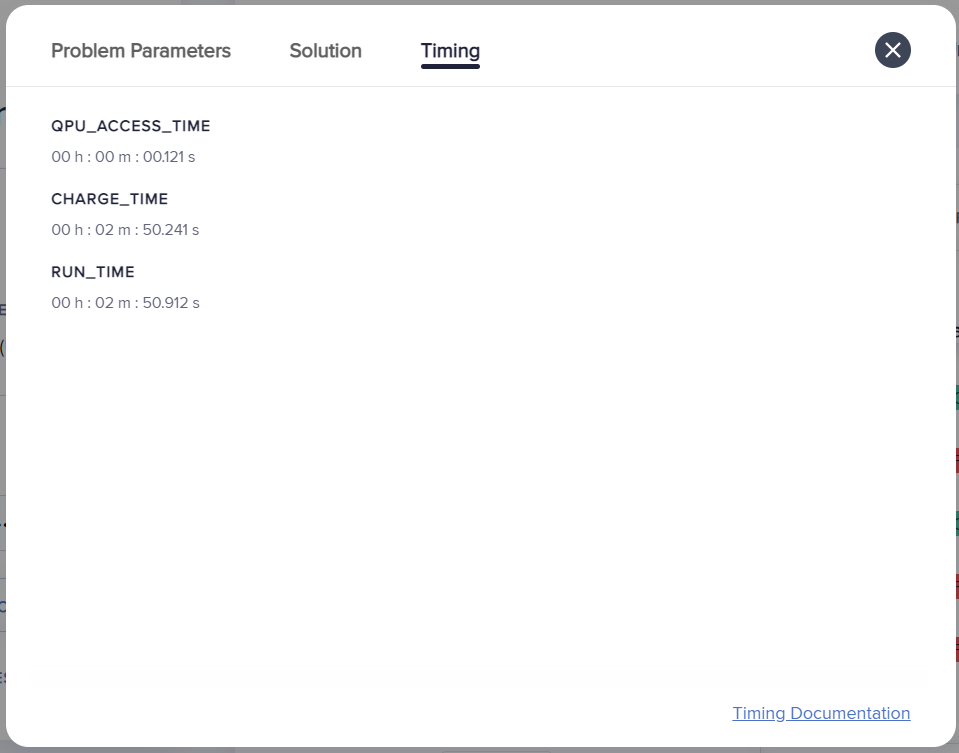


Image Segmentation for flood image 2:

Save the below image as floodorg.jpg



Reduce the size of the image with the below code:

Name: floodImageResize2.py

import numpy as np

import matplotlib.pyplot as plt

import cv2

from PIL import Image

img = cv2.imread('flood2.png')

type(img)

img.shape

fix\_img = cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)

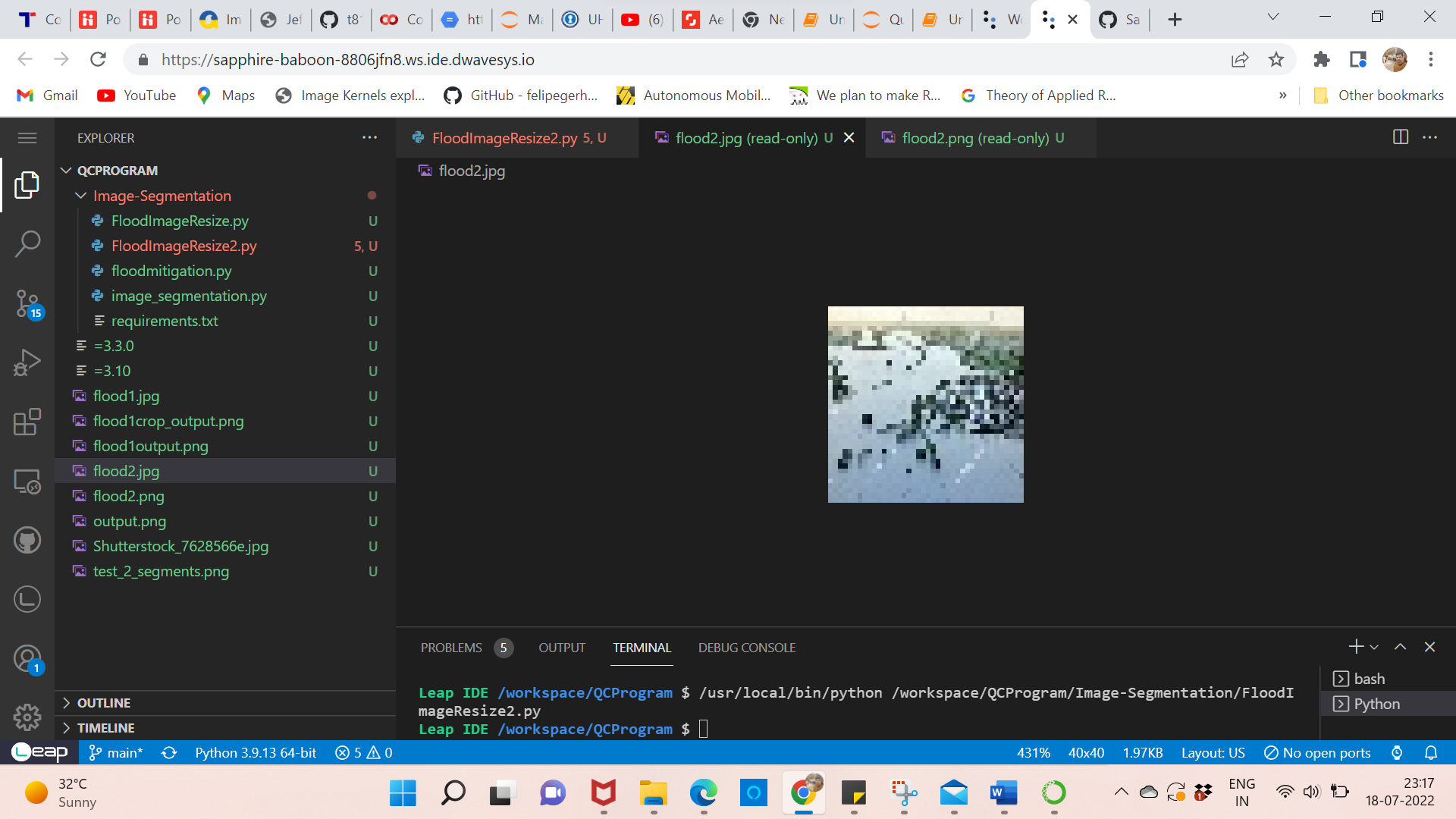
plt.imshow(fix\_img)

new\_image = cv2.resize(fix\_img,(40,40))

plt.imshow(new\_image)

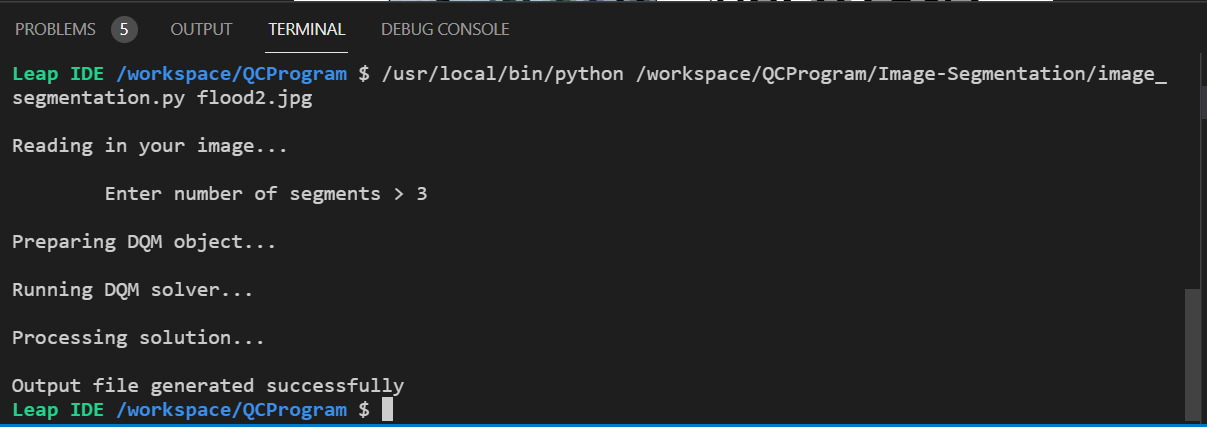
cv2.imwrite('flood2.jpg',new\_image)

Now the new image is stored as a 40x40 dimension image in the same file



Now Pass this through the DQM sampler program for segmentation:

Note: we need 3 segments here as there are three entities, buildings, flood water and trees.



Program :

Name: image\_segmentation.py

import cv2

import sys

import numpy as np

import matplotlib

matplotlib.use("agg")    # must select backend before importing pyplot

import matplotlib.pyplot as plt

from dimod import DiscreteQuadraticModel

from dwave.system import LeapHybridDQMSampler

# Define our weight function

def weight(a, b, img):

    \_, cols, \_ = img.shape

    diff = img[int(a/cols), a%cols, :] - img[int(b/cols), b%cols, :]

    return np.sum(np.square(diff))

# Convert single index into tuple indices

def unindexing(a):

    rows, cols, \_ = img.shape

    y1 = a % cols

    x1 = int(a/cols)

    return (x1, y1)

if len(sys.argv) > 1:

    # Read in image file specified

    data\_file\_name = sys.argv[1]

    random = False

    print("\nReading in your image...")

    img = cv2.imread(data\_file\_name)

    response\_2 = input("\n\tEnter number of segments > ")

    try:

        num\_segments = int(response\_2)

    except ValueError:

        print("Must input an integer.")

        num\_segments = int(input("\n\tEnter number of segments > "))

else:

    # Generate a random image with segments

    print("\nCreating random image...")

    random = True

    # Collect user input on size of problem

    response\_1 = input("\n\tEnter image dimensions > ")

    try:

        dims = int(response\_1)

    except ValueError:

        print("Must input an integer.")

        dims = int(input("\n\tEnter image dimensions > "))

    response\_2 = input("\n\tEnter number of segments > ")

    try:

        num\_segments = int(response\_2)

    except ValueError:

        print("Must input an integer.")

        num\_segments = int(input("\n\tEnter number of segments > "))

    img = np.zeros((dims, dims, 3), np.uint8)

    img\_rows = np.sort(np.random.choice(dims, num\_segments, replace=False))

    img\_cols = np.sort(np.random.choice(dims, num\_segments, replace=False))

    for num in range(num\_segments-1):

        color = np.random.randint(0, 255, 3)

        img[img\_rows[num]:, img\_cols[num]:, :] = color

# Create a version of the image data that is signed, so that subtraction will

# not wrap around when computing differences.

img\_signed = img.astype(int)

# Build the DQM and set biases according to pixel similarity

print("\nPreparing DQM object...")

rows, cols, \_ = img.shape

linear\_biases = np.zeros(rows\*cols\*num\_segments)

case\_starts = np.arange(rows\*cols) \* num\_segments

num\_interactions = rows \* cols \* (rows\*cols - 1) \* num\_segments / 2

qb\_rows = []

qb\_cols = []

qb\_biases = []

for i in range(rows\*cols):

    for j in range(i+1, rows\*cols):

        for case in range(num\_segments):

            qb\_rows.append(i\*num\_segments + case)

            qb\_cols.append(j\*num\_segments + case)

            qb\_biases.append(weight(i, j, img\_signed))

quadratic\_biases = (np.asarray(qb\_rows), np.asarray(qb\_cols), np.asarray(qb\_biases))

dqm = DiscreteQuadraticModel.from\_numpy\_vectors(case\_starts, linear\_biases, quadratic\_biases)

# Initialize the DQM solver

print("\nRunning DQM solver...")

sampler = LeapHybridDQMSampler()

# Solve the problem using the DQM solver

sampleset = sampler.sample\_dqm(dqm, label='Image Segmentation')

# Get the first solution

sample = sampleset.first.sample

print("\nProcessing solution...")

im\_segmented = np.zeros((rows, cols))

for key, val in sample.items():

    x, y = unindexing(key)

    im\_segmented[x,y] = val

if random:

    row\_indices = [1+i for i in range(rows-1)]

    row\_indices.append(0)

    im\_segmented\_rowwrap = im\_segmented[row\_indices, :]

    col\_indices = [1+i for i in range(cols-1)]

    col\_indices.append(0)

    im\_segmented\_colwrap = im\_segmented[:, col\_indices]

    im\_seg\_rowdiff = im\_segmented - im\_segmented\_rowwrap

    im\_seg\_coldiff = im\_segmented - im\_segmented\_colwrap

    segmented\_image = np.ones((rows, cols, 3), np.uint8)\*255

    segmented\_image[im\_seg\_rowdiff != 0] = (255, 0, 0)

    segmented\_image[im\_seg\_coldiff != 0] = (255, 0, 0)

else:

    segmented\_image = im\_segmented

    img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

fig, (ax1, ax2) = plt.subplots(1, 2)

ax1.imshow(img)

ax1.axes.xaxis.set\_visible(False)

ax1.axes.yaxis.set\_visible(False)

ax2.imshow(segmented\_image, cmap='Greys')

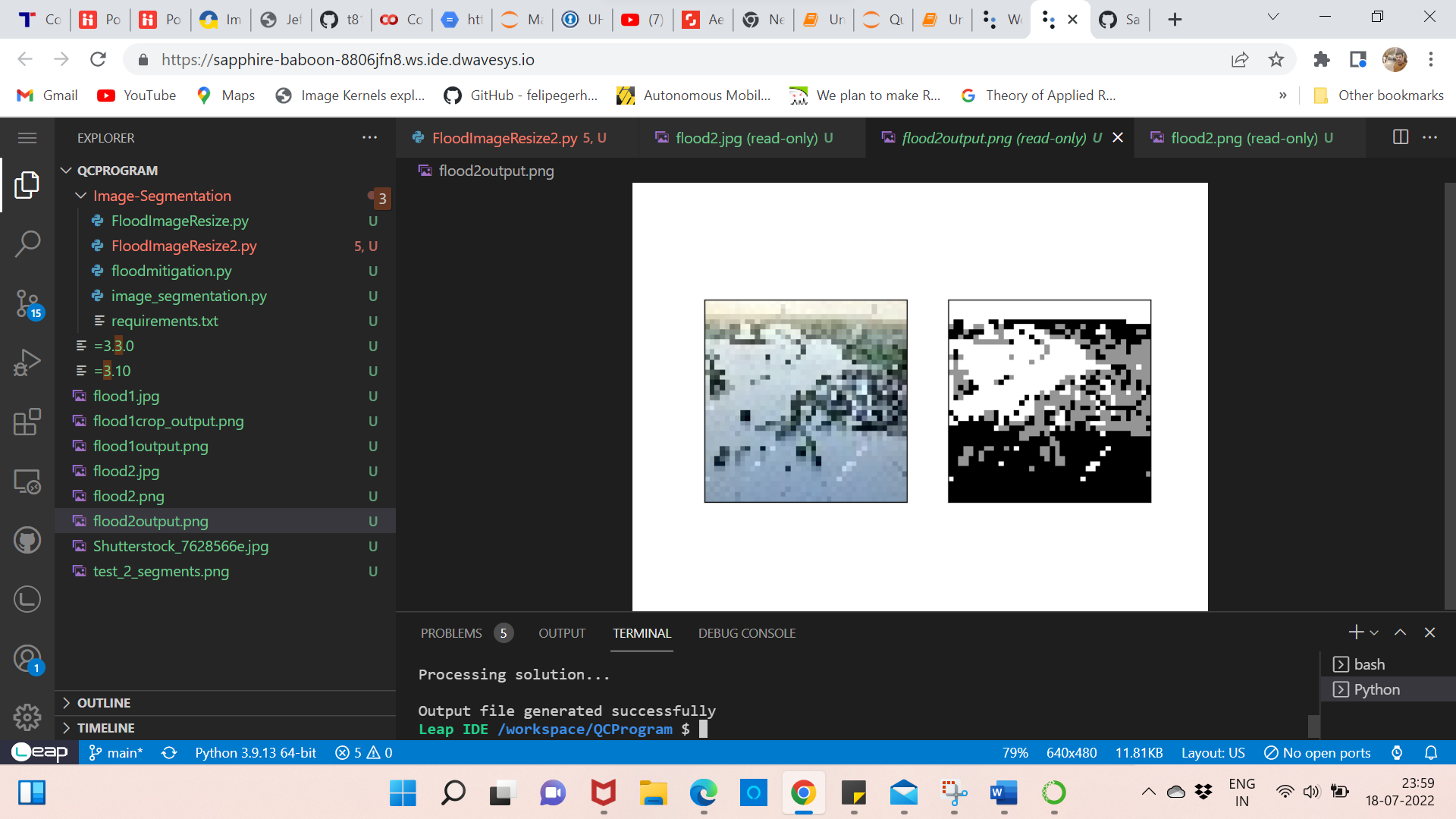
ax2.axes.xaxis.set\_visible(False)

ax2.axes.yaxis.set\_visible(False)

plt.savefig("output.png")

print("\nOutput file generated successfully")

Output (save it as flood2output.png):



Note: In this output, the segmentation has happened differently, for the upper half, gray is for land and buildings, black is for trees and white is for water. For the lower half, black and white is water, and gray is for trees. So we take this as two separate cropped images and calculate the tree, land, and water cover percentage separately.

Name: floodmitigation2.py

Program:

import numpy as np

import matplotlib.pyplot as plt

import cv2

from PIL import Image

left = 350

top  = 150

right = 575

bottom =355

imgcrop = Image.open('flood2output.png')

img2\_crop = imgcrop.crop((left, top, right, bottom))

plt.imshow(img2\_crop)

img2\_crop.save('flood2crop\_output.png')

img3\_gray = cv2.imread('flood2crop\_output.png',0)

plt.imshow(img3\_gray,cmap='gray')

img3\_gray.shape

lefta = 0

topa  = 0

righta = 225

bottoma = 120

img2\_cropa = img2\_crop.crop((lefta, topa, righta, bottoma))

plt.imshow(img2\_cropa)

img2\_cropa.save('flood2cropa.png')

img3\_graya = cv2.imread('flood2cropa.png',0)

plt.imshow(img3\_graya,cmap='gray')

print('shape of image flood2cropa: ', img3\_graya.shape)

blacka = np.sum(img3\_graya == 0)

whitea = np.sum(img3\_graya == 255)

graya =  (120\*225) - (whitea+blacka)

Percentage\_green\_a = (blacka/img3\_graya.size)\*100

Percentage\_water\_a = (whitea/img3\_graya.size)\*100

Percentage\_land\_a = (graya/img3\_graya.size)\*100

print("Percentage of tree cover in the image flood2cropa is : ",Percentage\_green\_a,"%")

print("Percentage of water in the image flood2cropa is : ",Percentage\_water\_a,"%")

print("Percentage of land and building in the image flood2cropa is : ",Percentage\_land\_a,"%")

leftb = 0

topb  = 121

rightb = 225

bottomb = 205

img2\_cropb = img2\_crop.crop((leftb, topb, rightb, bottomb))

plt.imshow(img2\_cropb)

img2\_cropb.save('flood2cropb.png')

img3\_grayb = cv2.imread('flood2cropb.png',0)

plt.imshow(img3\_grayb,cmap='gray')

print('shape of image flood2cropb: ',img3\_grayb.shape)

blackb = np.sum(img3\_grayb == 0)

whiteb = np.sum(img3\_grayb == 255)

grayb =  (84\*225) - (whiteb+blackb)

Percentage\_green\_b = (grayb/img3\_grayb.size)\*100

Percentage\_water\_b = ((whiteb+blackb)/img3\_grayb.size)\*100

print("Percentage of tree cover in the image flood2cropb is : ",Percentage\_green\_b,"%")

print("Percentage of water in the image flood2cropb is : ",Percentage\_water\_b,"%")

Percentage\_green\_total = ((blacka+grayb)/(img3\_graya.size+img3\_grayb.size))\*100

Percentage\_water\_total = ((whiteb+blackb+whitea)/(img3\_graya.size+img3\_grayb.size))\*100

Percentage\_land\_total = (graya/(img3\_graya.size+img3\_grayb.size)\*100)

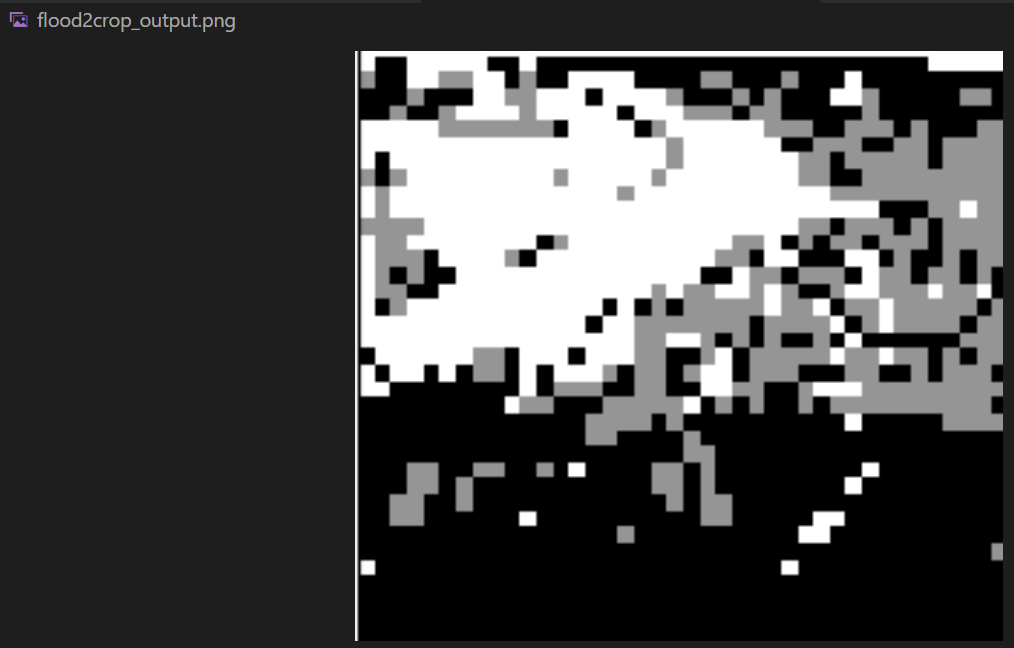
print("Percentage of tree cover in the image flood2crop is : ",Percentage\_green\_total,"%")

print("Percentage of water in the image flood2crop is : ",Percentage\_water\_total,"%")

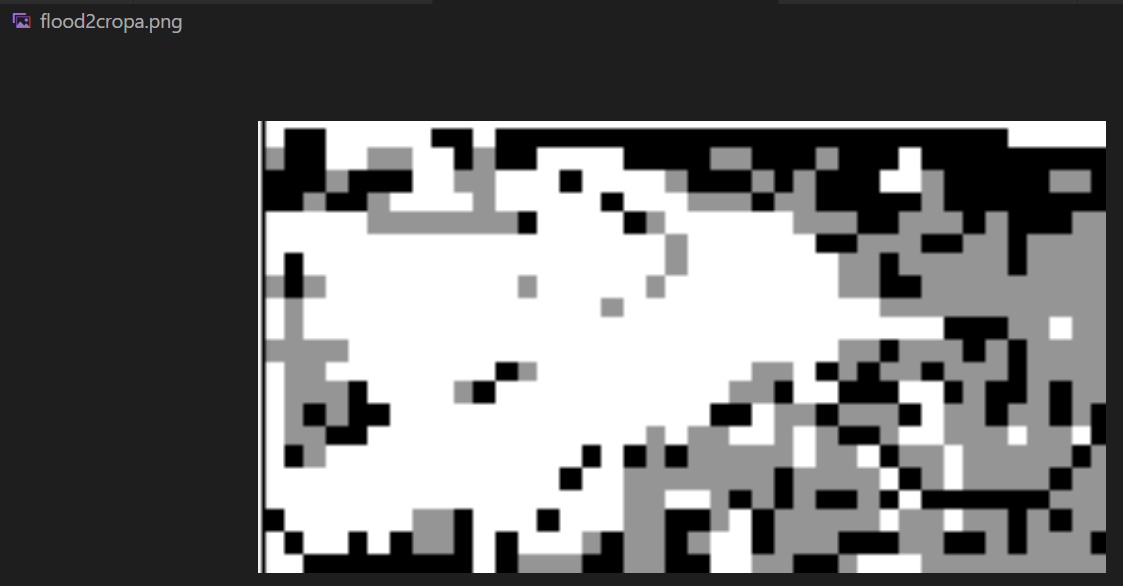
print("Percentage of land and building in the image flood2crop is : ",Percentage\_land\_total,"%")

Cropped output is saved as flood2crop\_output.png:

Note: The top sky part is cropped out here as it is irrelevant.

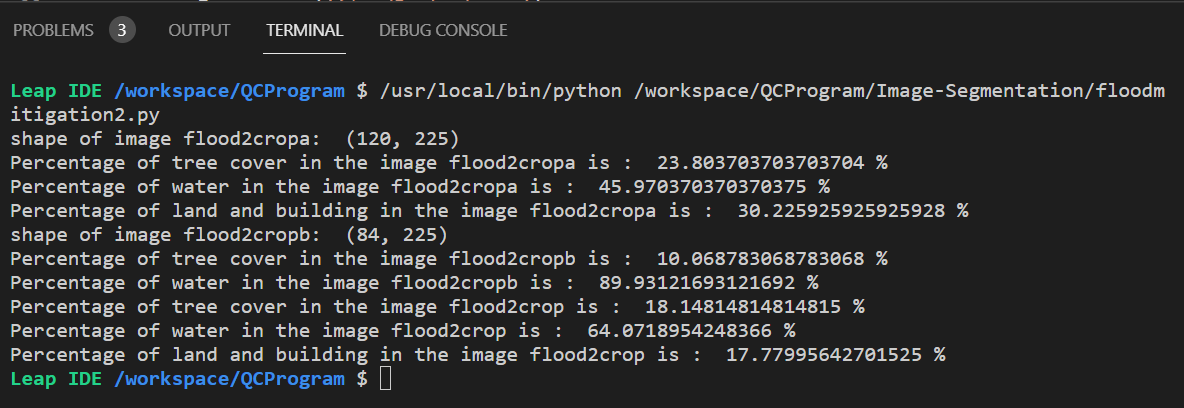


Now this output is cropped into top-half and bottom-half:

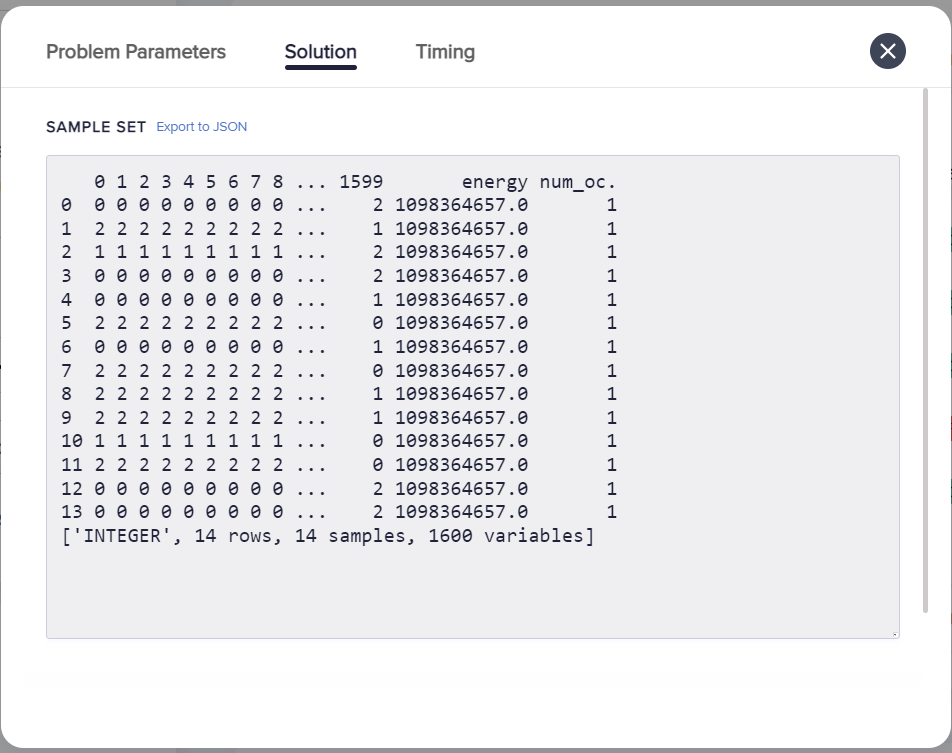




The percentage of tree, land and water cover is calculated:



Solution Parameters:



Timing:

