Name – Vivek Onkarnath Rathi

Student ID – 200256752

ECE 558 Project 01

Problem 1 (40 points): Visualizing and checking the smoothness prior. Given an image I, consider all valid pairs of neighboring pixels, compute the difference between their intensity or color values, and plot the histogram.

Solution –

Python Code –

```
# -*- coding: utf-8 -*-
"""
Created on Sun Sep 22 13:25:13 2019
@author: Vivek Rathi

Here Neighbours are
[ Dlu vu Dru]
[ hl  P  hr ]
[ Dld vd Drd]
"""
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Load image
imgrey = cv2.imread('wolves.png',0) #greyscale
ibgr = cv2.imread('wolves.png') # rgb
ihsv = cv2.cvtColor(ibgr, cv2.COLOR_BGR2HSV) #hsv
ilab = cv2.cvtColor(ibgr, cv2.COLOR_BGR2Lab) #lab

'''
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
'''
#case 1 - intensity
def diff_2d(img_mat,nb):
    r = img_mat.shape[0]
    c = img_mat.shape[1]
    a = np.array(imgrey, dtype = 'int32')
    d = np.zeros((r,c), dtype = 'int32')
    for i in range(r):
        for j in range(c):
            if nb == 'hr':
                if [i,j] == [i,c-1]:
                    pass
                else:
                    d[i,j] = (a[i,j] - a[i,j+1])**2
            elif nb == 'vd': # x,y  x+1,y
                if [i,j] == [r-1,j]:
```

```
                pass
            else:
                d[i,j] = (a[i,j] - a[i+1,j])**2
        elif nb == 'vu': # x,y  x-1,y
            if [i,j] == [0,j]:
                pass
            else:
                d[i,j] = (a[i,j] - a[i-1,j])**2
        elif nb == 'hl': # x,y  x,y-1
            if [i,j] == [i,0]:
                pass
            else:
                d[i,j] = (a[i,j] - a[i,j-1])**2
        elif nb == 'Dru': # x,y  x-1,y+1
            if [i,j] == [0,j] or [i,j] == [i,c-1]:
                pass
            else:
                d[i,j] = (a[i,j] - a[i-1,j+1])**2
        elif nb == 'Drd': # x,y  x+1,y+1
            if [i,j] == [r-1,j] or [i,j] == [i,c-1]:
                pass
            else:
                d[i,j] = (a[i,j] - a[i+1,j+1])**2
        elif nb == 'Dlu': # x,y  x-1,y-1
            if [i,j] == [0,j] or [i,j] == [i,0]:
                pass
            else:
                d[i,j] = (a[i,j] - a[i-1,j-1])**2
        elif nb == 'Dld': # x,y  x+1,y-1
            if [i,j] == [i,0] or [i,j] == [r-1,j]:
                pass
            else:
                d[i,j] = (a[i,j] - a[i+1,j-1])**2
if nb == 'hr':
    d = d[:,:c-1]
elif nb == 'vd':
    d = d[:r-1,:]
elif nb == 'vu':
    d = d[1:,:]
elif nb == 'hl':
    d = d[:,1:]
elif nb == 'Dru':
    d = d[1:,:c-1]
elif nb == 'Drd':
    d = d[:r-1,:c-1]
elif nb == 'Dlu':
    d = d[:r-1,1:]
elif nb == 'Dld':
```

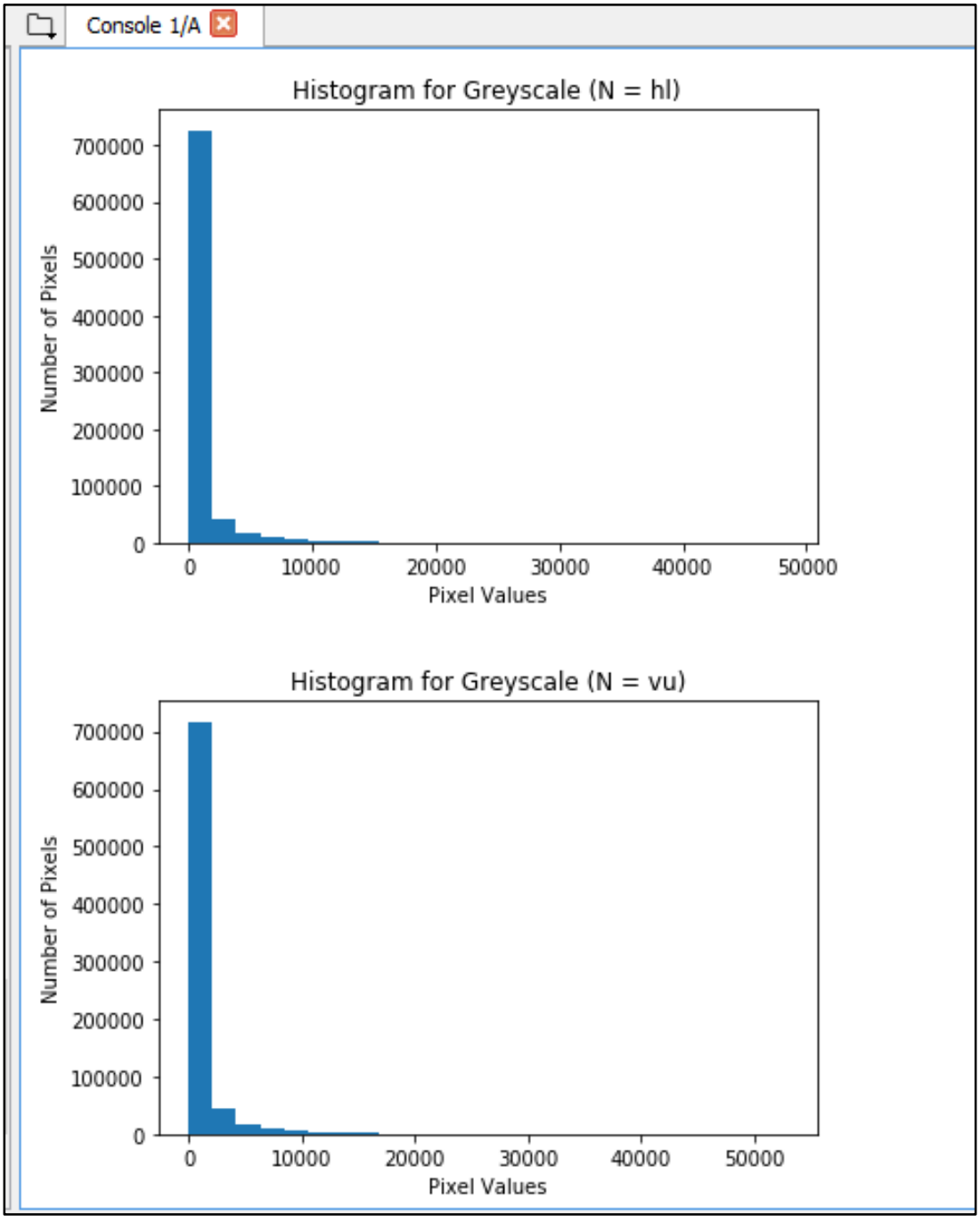
```
        d = d[1:,1:]

    return d

#case 2 3 channels
def diff_3d(img_mat,nb):
    r = img_mat.shape[0]
    c = img_mat.shape[1]
    ch1 = np.array((img_mat[:,:,0]), dtype = 'int32')
    ch2 = np.array((img_mat[:,:,1]), dtype = 'int32')
    ch3 = np.array((img_mat[:,:,2]), dtype = 'int32')
    d = np.zeros((r,c), dtype = 'int32')
    for i in range(r):
        for j in range(c):
            if nb == 'hr':
                if [i,j] == [i,c-1]:
                    pass
                else:
                    d[i,j] = (ch1[i,j] - ch1[i,j+1])**2 + (ch2[i,j] - ch2[i,j+1])**2 + (ch3[i,j] - ch3[i,j+1])**2
            elif nb == 'vd': # x,y  x+1,y
                if [i,j] == [r-1,j]:
                    pass
                else:
                    d[i,j] = (ch1[i,j] - ch1[i+1,j])**2 + (ch2[i,j] - ch2[i+1,j])**2 + (ch3[i,j] - ch3[i+1,j])**2
            elif nb == 'vu': # x,y  x-1,y
                if [i,j] == [0,j]:
                    pass
                else:
                    d[i,j] = (ch1[i,j] - ch1[i-1,j])**2 + (ch2[i,j] - ch2[i-1,j])**2 + (ch3[i,j] - ch3[i-1,j])**2
            elif nb == 'hl': # x,y  x,y-1
                if [i,j] == [i,0]:
                    pass
                else:
                    d[i,j] = (ch1[i,j] - ch1[i,j-1])**2 + (ch2[i,j] - ch2[i,j-1])**2 + (ch3[i,j] - ch3[i,j-1])**2
            elif nb == 'Dru': # x,y  x-1,y+1
                if [i,j] == [0,j] or [i,j] == [i,c-1]:
                    pass
                else:
                    d[i,j] = (ch1[i,j] - ch1[i-1,j+1])**2 + (ch2[i,j] - ch2[i-1,j+1])**2 + (ch3[i,j] - ch3[i-1,j+1])**2
            elif nb == 'Drd': # x,y  x+1,y+1
                if [i,j] == [r-1,j] or [i,j] == [i,c-1]:
                    pass
                else:
                    d[i,j] = (ch1[i,j] - ch1[i+1,j+1])**2 + (ch2[i,j] - ch2[i+1,j+1])**2 + (ch3[i,j] - ch3[i+1,j+1])**2
            elif nb == 'Dlu': # x,y  x-1,y-1
                if [i,j] == [0,j] or [i,j] == [i,0]:
                    pass
                else:
```

```
                d[i,j] = (ch1[i,j] - ch1[i-1,j-1])**2 + (ch2[i,j] - ch2[i-1,j-1])**2 + (ch3[i,j] - ch3[i-1,j-1])**2
            elif nb == 'Dld': # x,y  x+1,y-1
                if [i,j] == [i,0] or [i,j] == [r-1,j]:
                    pass
                else:
                    d[i,j] = (ch1[i,j] - ch1[i+1,j-1])**2 + (ch2[i,j] - ch2[i+1,j-1])**2 + (ch3[i,j] - ch3[i+1,j-1])**2
    if nb == 'hr':
        d = d[:,:c-1]
    elif nb == 'vd':
        d = d[:r-1,:]
    elif nb == 'vu':
        d = d[1:,:]
    elif nb == 'hl':
        d = d[:,1:]
    elif nb == 'Dru':
        d = d[1:,:c-1]
    elif nb == 'Drd':
        d = d[:r-1,:c-1]
    elif nb == 'Dlu':
        d = d[:r-1,1:]
    elif nb == 'Dld':
        d = d[1:,1:]

    return d
#histogram
def histogram(img,bins,title):
    plt.hist(img.ravel(),bins);
    plt.title(title)
    plt.xlabel('Pixel Values')
    plt.ylabel('Number of Pixels')
    plt.show()
#case 1 - intensity
g1 = diff_2d(imgrey,'hl')
g2 = diff_2d(imgrey,'vu')
#case 2 - RGB
bgr1 = diff_3d(ibgr,'hr')
bgr2 = diff_3d(ibgr,'vd')
#case 3- hsv
hsv1 = diff_3d(ihsv,'Dlu')
hsv2 = diff_3d(ihsv,'Dru')
#case 4 - Lab
lab1 = diff_3d(ilab,'Dld')
lab2 = diff_3d(ilab,'Drd')
# calculate histogram

histogram(g1,25,'Histogram for Greyscale (N = hl)')
histogram(g2,25,'Histogram for Greyscale (N = vu)')
```

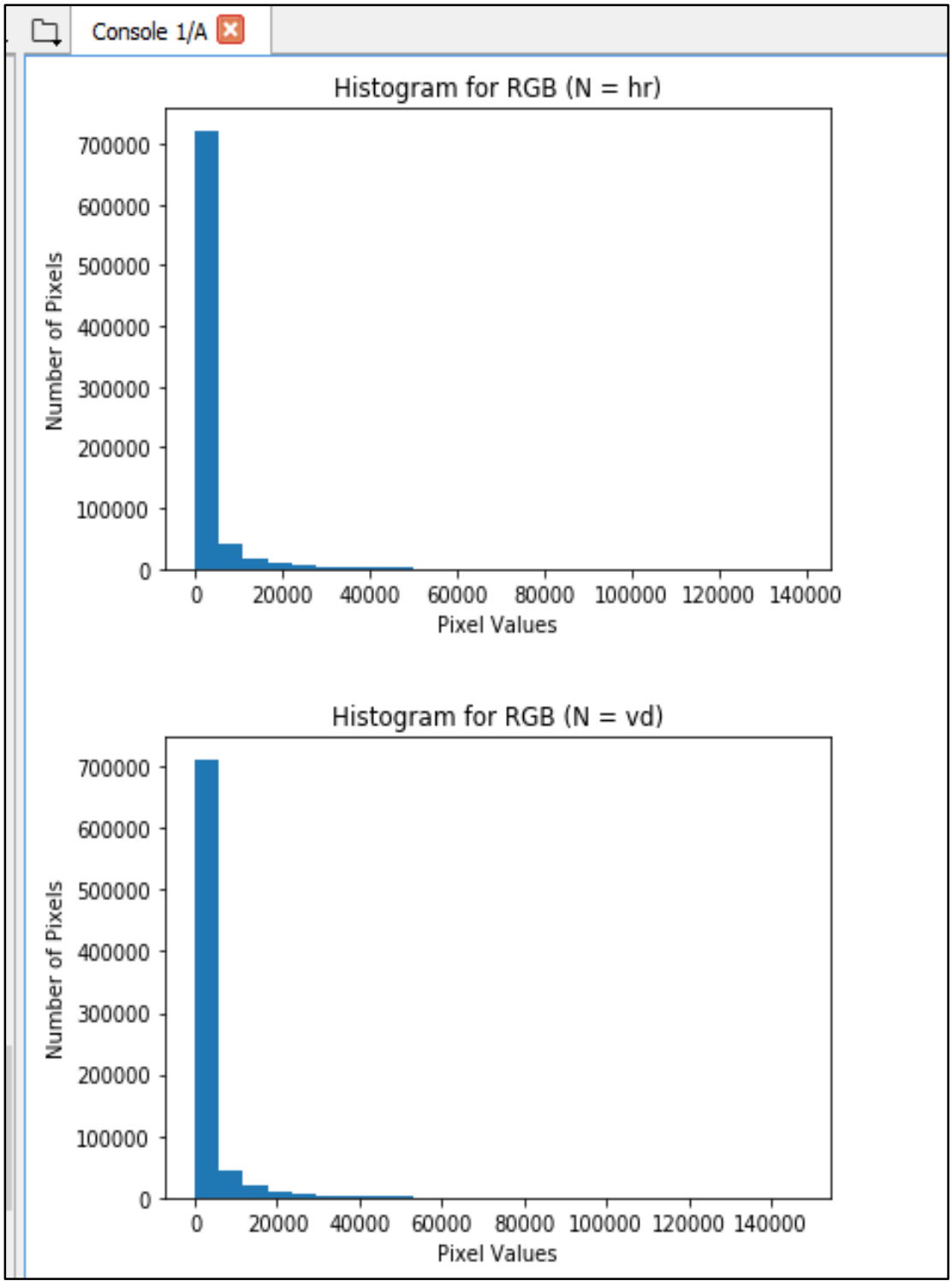
```
histogram(bgr1,25,'Histogram for RGB (N = hr)')
histogram(bgr2,25,'Histogram for RGB (N = vd)')

histogram(hsv1,25,'Histogram for HSV (N = Dlu)')
histogram(hsv2,25,'Histogram for HSV (N = Dru)')

histogram(lab1,25,'Histogram for Lab (N = Dld)')
histogram(lab2,25,'Histogram for Lab (N = Drd)')
```

OUTPUT –

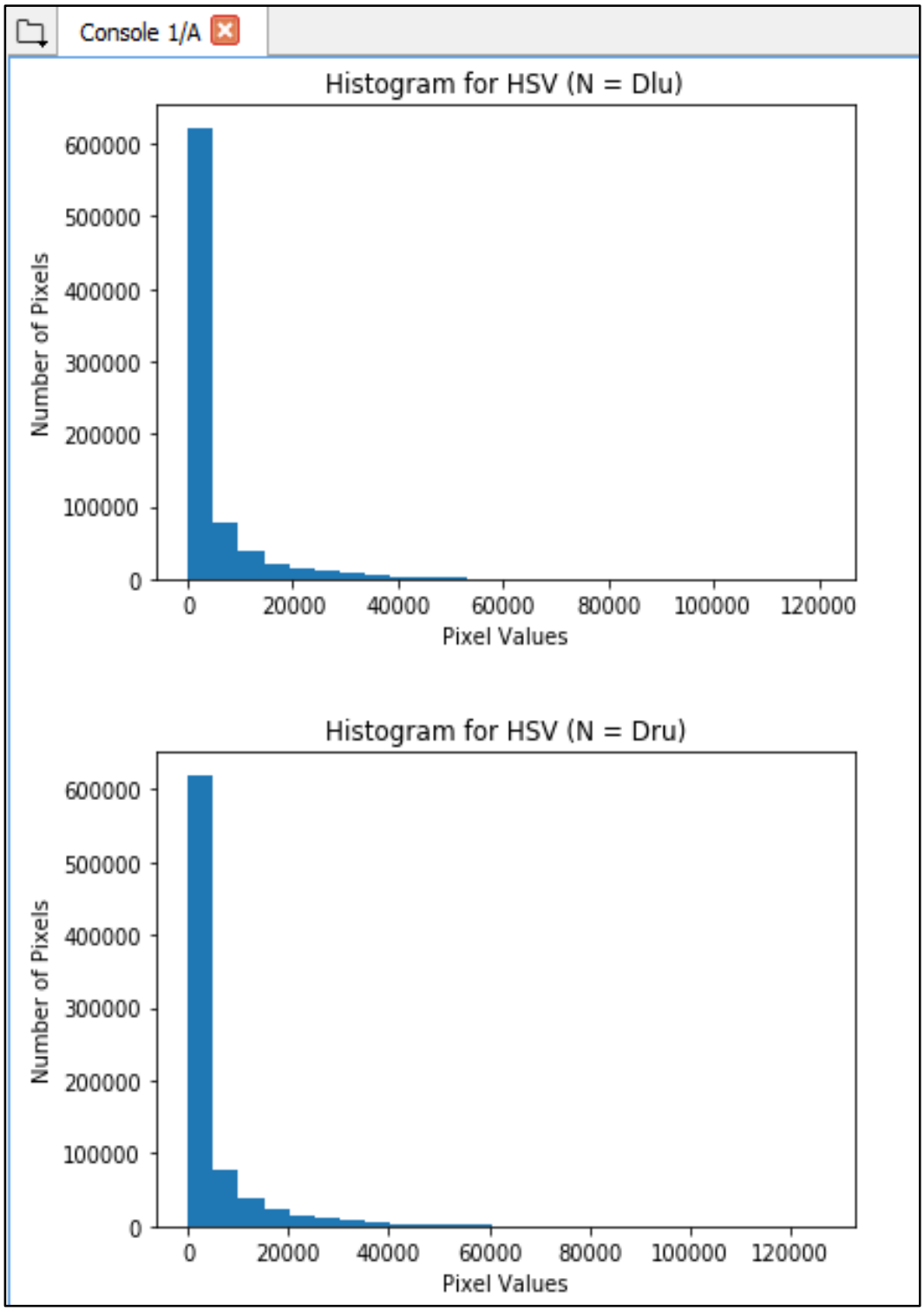
1. Greyscale Image



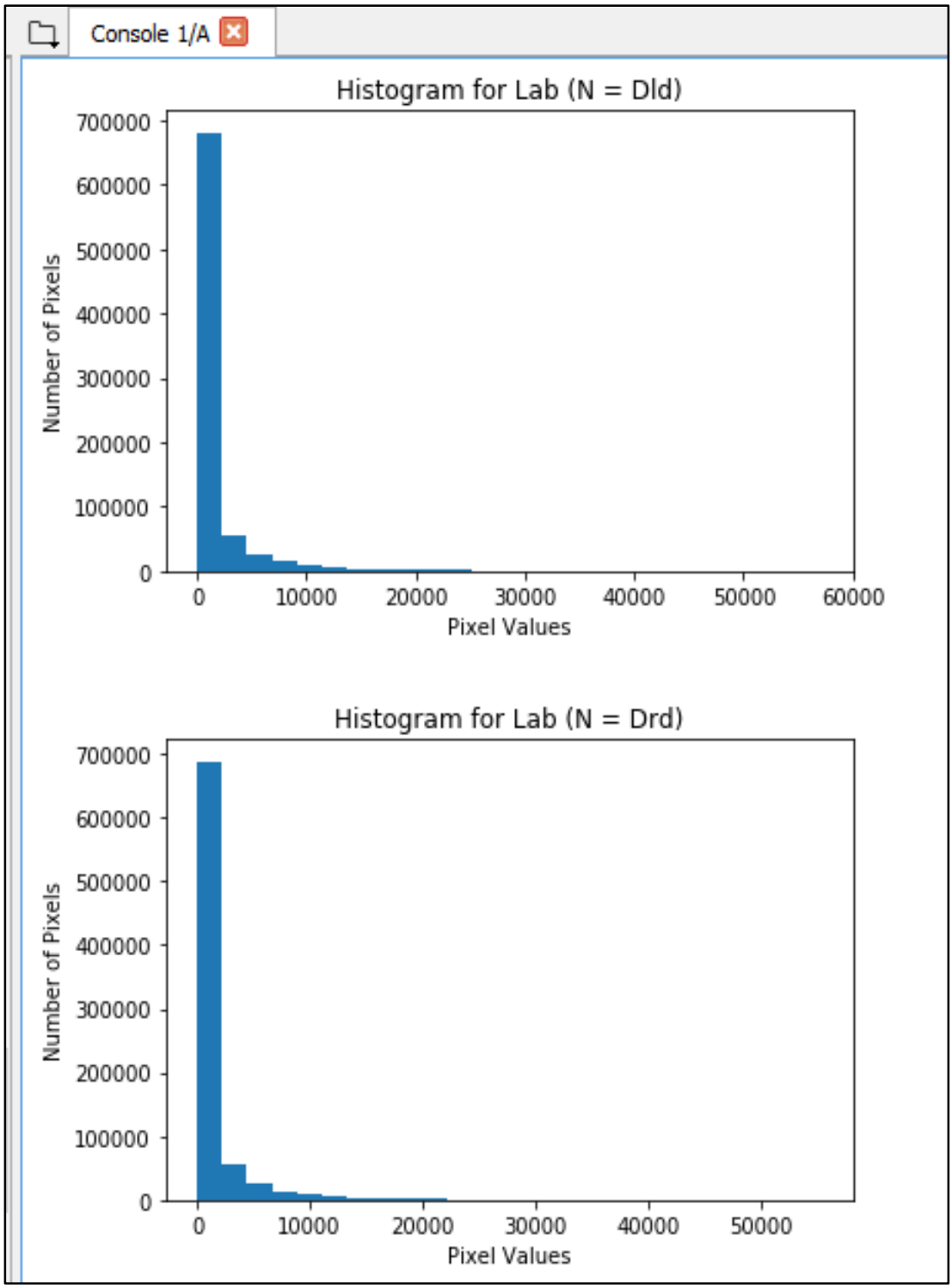
2. RGB Image


Console 1/A
Histogram for RGB (N = hr)
Number of Pixels
700000
600000
500000
400000
300000
200000
100000
0
0
20000
40000
60000
80000
100000
120000
140000
Pixel Values
Histogram for RGB (N = vd)
Number of Pixels
700000
600000
500000
400000
300000
200000
100000
0
0
20000
40000
60000
80000
100000
120000
140000
Pixel Values

3. HSV Image


Console 1/A
Histogram for HSV (N = Dlu)
Number of Pixels
600000
500000
400000
300000
200000
100000
0
0
20000
40000
60000
80000
100000
120000
Pixel Values
Histogram for HSV (N = Dru)
Number of Pixels
600000
500000
400000
300000
200000
100000
0
0
20000
40000
60000
80000
100000
120000
Pixel Values

4. Lab Image



The run time of the code is 1:23 minutes

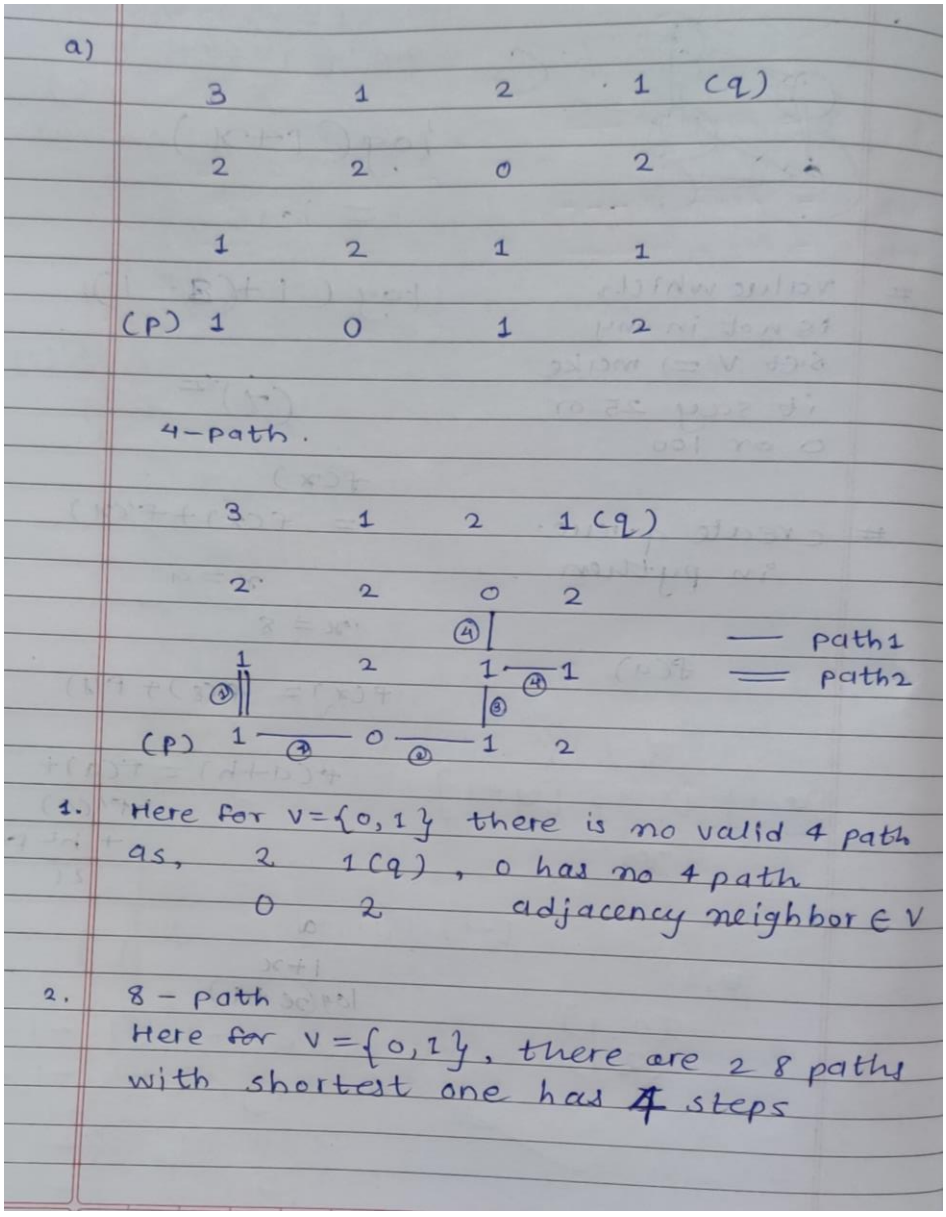
Problem 2 (60 points): Finding the shortest path.

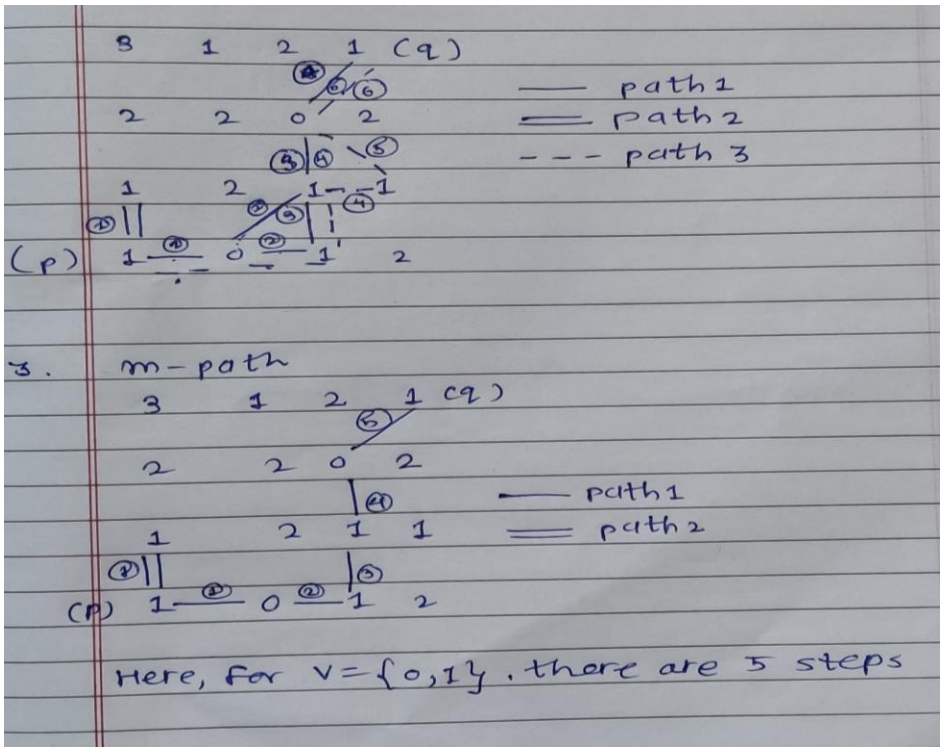
Consider the image segment shown.

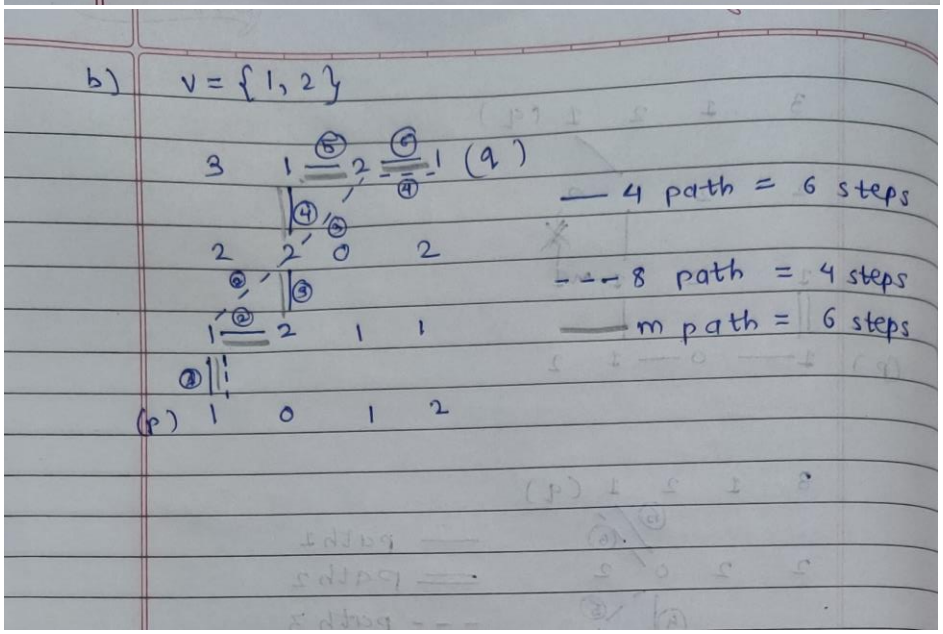
3	1	2	1 (q)
2	2	0	2
1	2	1	1
(p) 1	0	1	2

(a) (8 points) Let V = {0, 1} and compute the lengths of the shortest 4-, 8-, and m-path between p and q. Show the corresponding paths. If a particular path does not exist between these two points, explain why.
(b) (2 points) Repeat for V = {1,2}.
(c) (50 points) Write python / matlab code to implement the function.

Solution-







a) 4-path : No path
   8-path: 4 Steps
   m-path: 5 Steps
b) 4-path: 6 Steps
   8-path: 4 Steps
   m-path: 6 Steps

c)
Python Code-

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 23 22:37:51 2019

@author: Vivek Rathi

Shortest Path Finding using BFS
If there is no path, the function will return False
"""



import numpy as np
import cv2

# functions to traverse neighbours according to adjacency
def explore_4_neigh(r,c,A,V):
    dr = [-1,1,0,0] # direction for rows, i.e. up & down
    dc = [0,0,1,-1] # direction for columns, i.e. left & right
    for i in range(4):
        rr = r + dr[i] # update the [rr,cc] for validating the pixel location
        cc = c + dc[i]
        # use constraints to avoid considering illegal pixel locations
        if rr < 0 or cc < 0:
            continue
        if rr >= row or cc >= col:
            continue
        if visited_pixel[rr,cc]:
            continue
        if not(A[rr,cc] in V):
            continue
        # iff valid pixel locations, queue the locations [rr,cc]
        que_r.append(rr)
        que_c.append(cc)
        # keep track of visited pixel locations
        visited_pixel[rr,cc] = True
        # use global keyword to use the variable locally & globally
        global pixels_in_next_branch
        # Update pixels in next branch
        pixels_in_next_branch = pixels_in_next_branch + 1

def explore_8_neigh(r,c,A,V):
    dr = [-1,1,0,0,-1,+1,-1,1]
    dc = [0,0,1,-1,-1,-1,1,1]
    for i in range(8):
        rr = r + dr[i]
```

```
        cc = c + dc[i]
        if rr < 0 or cc < 0:
            continue
        if rr >= row or cc >= col:
            continue
        if visited_pixel [rr,cc]:
            continue
        if not(A[rr,cc] in V):
            continue
        que_r.append(rr)
        que_c.append(cc)
        visited_pixel[rr,cc] = True
        global pixels_in_next_branch
        pixels_in_next_branch = pixels_in_next_branch + 1

def explore_m_neigh(r,c,A,V):
    dr = [-1,1,0,0,-1,+1,-1,1]
    dc = [0,0,1,-1,-1,-1,1,1]
    for i in range(8):
        rr = r + dr[i]
        cc = c + dc[i]
        if rr < 0 or cc < 0:
            continue
        if rr >= row or cc >= col:
            continue
        if visited_pixel [rr,cc]:
            continue
        if not(A[rr,cc] in V):
            continue
        if rr == r - 1 and cc == c - 1 and ((A[r-1,c] in V) or (A[r,c-1] in V)):
            continue
        if rr == r + 1 and cc == c - 1 and ((A[r+1,c] in V) or (A[r,c-1] in V)):
            continue
        if rr == r - 1 and cc == c + 1 and ((A[r,c+1] in V) or (A[r-1,c] in V)):
            continue
        if rr == r + 1 and cc == c + 1 and ((A[r,c+1] in V) or (A[r+1,c] in V)):
            continue
        que_r.append(rr)
        que_c.append(cc)
        visited_pixel[rr,cc] = True
        global pixels_in_next_branch
        pixels_in_next_branch = pixels_in_next_branch + 1


def getpath(atype,mat,V):
    # Enqueque the queue with the starting pixel's row and column position
    que_r.append(sr)
    que_c.append(sc)
```

```
    # Update visited_pixel's array
    visited_pixel[sr,sc] = True
    #global declaration
    global at_end
    # do this untill my queue is empty
    while len(que_r) > 0:
        # Dequeque the queue to get values at the head of the queue
        rd = que_r.pop(0)
        cd = que_c.pop(0)
        # if we reached the end, exit the loop
        if [rd,cd] == [er,ec]:
            at_end = True
            break
        # depending on the type of adjacency, explore the neighbours
        if atype == '4':
            explore_4_neigh(rd,cd,mat,V)
        elif atype == '8':
            explore_8_neigh(rd,cd,mat,V)
        elif atype == 'm':
            explore_m_neigh(rd,cd,mat,V)
        # break/exit condition when there is no valid/legal path available
        if len(que_r) == 0:
            break
        global pixels_in_curr_branch, pixels_in_next_branch, path_steps
        # update the variable
        pixels_in_curr_branch = pixels_in_curr_branch - 1
        # if i have 0 pixels in the current branch, add the path steps and trajectory
        if pixels_in_curr_branch == 0:
            pixels_in_curr_branch = pixels_in_next_branch
            pixels_in_next_branch = 0
            path_steps = path_steps + 1
            path_pixels.append([rd,cd])
    # if we get to the end, mission accomplished, return path steps and trajectory
    if at_end:
        path_pixels.append([er,ec])
        return path_steps, path_pixels
    else:
        return False


# Main code

# Example 1
A = np.array(([3,1,2,1],[2,2,0,2],[1,2,1,1],[1,0,1,2]))
# get rows and columns of example matrix
row = A.shape[0]
col = A.shape[1]
V2 = [1,2] #Set 2 for neighbours adjacency
```

```
V1 = [0,1] #Set 1 for neighboiurs adjacency

# pixel locations for start pixel
sr = 3
sc = 0
# pixel locations for end pixel
er = 0
ec = 3

# define queues for rows and columns of the array i.e. pixel locations (x,y)
que_r = []
que_c = []
#path 4
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path4 = getpath('4',A,V1)


#path 8
que_r = [] #clear the queues
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path8 = getpath('8',A,V1)


#path m
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
```

```
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
pathm = getpath('m',A,V1)
print("Image1 = \n",A)
print("Set =",V1)
print("Path4 = ",path4)
print("Path8 = ",path8)
print("Pathm = ",pathm)
print('\n')

#path 4
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path4 = getpath('4',A,V2)


#path 8
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path8 = getpath('8',A,V2)


#path m
```

```
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
pathm = getpath('m',A,V2)
print("Image1 = \n",A)
print("Set =",V2)
print("Path4 = ",path4)
print("Path8 = ",path8)
print("Pathm = ",pathm)
print('\n')


##Example 2
imgrey = cv2.imread('wolves.png',0)
B = imgrey[300:305,400:405]
# get rows and columns of example matrix
row = B.shape[0]
col = B.shape[1]
V = [34,43,46]
sr = 0
sc = 0
er = 4
ec = 4
# define queues for rows and columns of the array i.e. pixel locations (x,y)
que_r = []
que_c = []

#path4
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path4 = getpath('4',B,V)
```

```
#path 8
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path8 = getpath('8',B,V)

#path m
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
pathm = getpath('m',B,V)
print("Image2 = \n",B)
print("Set =",V)
print("Path4 = ",path4)
print("Path8 = ",path8)
print("Pathm = ",pathm)
print('\n')

#Example 3
D = imgrey[4:9,5:10]
row = D.shape[0]
col = D.shape[1]
V = [0]
sr = 0
sc = 0
er = 4
ec = 0
# define queues for rows and columns of the array i.e. pixel locations (x,y)
que_r = []
```

```
que_c = []
#path4
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path4 = getpath('4',D,V)

#path 8
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path8 = getpath('8',D,V)

#path m
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
pathm = getpath('m',D,V)
print("Image3 = \n",D)
print("Set =",V)
print("Path4 = ",path4)
print("Path8 = ",path8)
print("Pathm = ",pathm)
```

```
print('\n')

#Example 4
E = imgrey[50:60,200:205]
row = E.shape[0]
col = E.shape[1]
V = [66,21,50]
sr = 9
sc = 4
er = 3
ec = 0
# define queues for rows and columns of the array i.e. pixel locations (x,y)
que_r = []
que_c = []
#path4
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path4 = getpath('4',E,V)

#path 8
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
path8 = getpath('8',E,V)

#path m
que_r = []
que_c = []
# create boolean array to track visited pixels
visited_pixel = np.zeros((row,col)).astype(bool)
# define list for getting path trajectory
```

```
path_pixels = []
# counter variable for path steps
path_steps = 0
pixels_in_curr_branch = 1
pixels_in_next_branch = 0
# flag for end location
at_end = False
pathm = getpath('m',E,V)
print("Image4 = \n",E)
print("Set =",V)
print("Path4 = ",path4)
print("Path8 = ",path8)
print("Pathm = ",pathm)
print('\n')
```

Output –

1. Image 1

```
Image1 =
 [[3 1 2 1]
 [2 2 0 2]
 [1 2 1 1]
 [1 0 1 2]]
Set = [0, 1]
Path4 =  False
Path8 =  (4, [[3, 0], [3, 1], [2, 2], [1, 2], [0, 3]])
Pathm =  (5, [[3, 0], [3, 1], [3, 2], [2, 2], [2, 3], [0, 3]])


Image1 =
 [[3 1 2 1]
 [2 2 0 2]
 [1 2 1 1]
 [1 0 1 2]]
Set = [1, 2]
Path4 =  (6, [[3, 0], [2, 0], [2, 1], [2, 2], [2, 3], [1, 3], [0, 3]])
Path8 =  (4, [[3, 0], [2, 1], [3, 2], [3, 3], [0, 3]])
Pathm =  (6, [[3, 0], [2, 0], [2, 1], [2, 2], [2, 3], [1, 3], [0, 3]])
```

2. Image 2

```
Image2 =
 [[ 34  70  61 103 118]
 [ 68  46  66  80  90]
 [ 56  43  53  46  61]
 [ 87  53  43  59  28]
 [ 43  39  46  43  34]]
Set = [34, 43, 46]
Path4 =  False
Path8 =  (5, [[0, 0], [1, 1], [2, 1], [3, 2], [4, 3], [4, 4]])
Pathm =  (6, [[0, 0], [1, 1], [2, 1], [3, 2], [2, 3], [4, 3], [4, 4]])
```

3. Image 3

```
Image3 =
 [[ 0  9  0  0  0]
 [ 0  0  0  0  0]
 [ 0  0  0  0 21]
 [21  0  0  0 34]
 [ 0  0  0  0 21]]
Set = [0]
Path4 =  (6, [[0, 0], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4], [4, 0]])
Path8 =  (4, [[0, 0], [1, 1], [2, 2], [3, 3], [4, 0]])
Pathm =  (6, [[0, 0], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4], [4, 0]])
```

4. Image 4

```
Image4 =
 [[111  92 136 175  95]
 [ 80  98  87  78  66]
 [114  95  68  64  80]
 [ 50  64  90  68  59]
 [ 34  21  43  43  21]
 [ 34  43  21  21  21]
 [ 50  21  21  28  21]
 [ 61  39  34  21  21]
 [ 59  43  43  21  28]
 [ 53  56  59  61  66]]
Set = [66, 21, 50]
Path4 =  False
Path8 =  (6, [[9, 4], [8, 3], [7, 4], [6, 4], [5, 4], [4, 4], [3, 0]])
Pathm =  (6, [[9, 4], [8, 3], [7, 3], [6, 2], [6, 1], [6, 0], [3, 0]])
```

The runtime of the code is 1.66 seconds