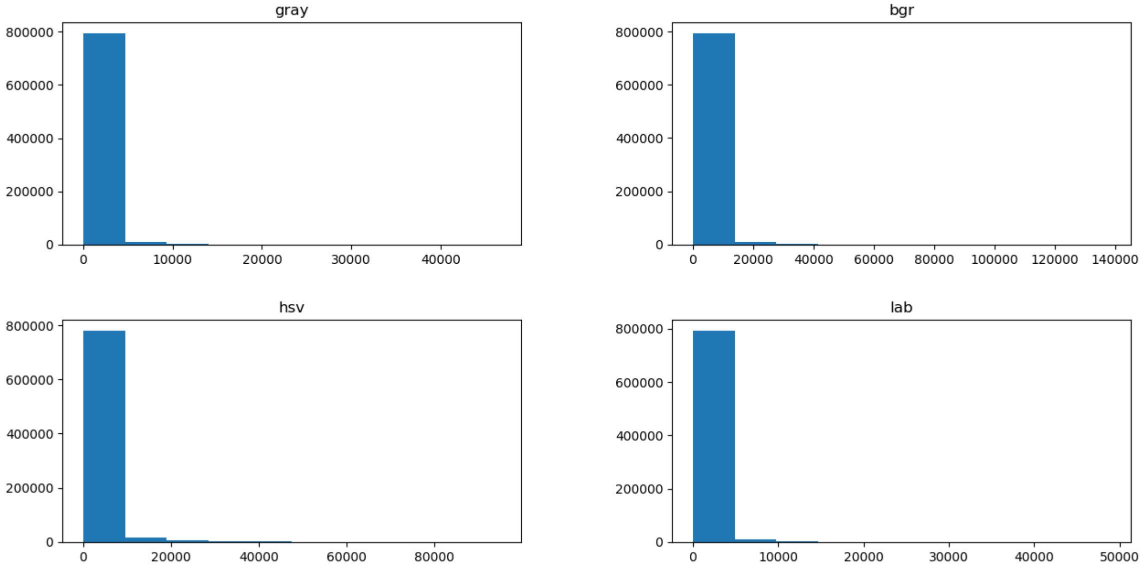
Question 1:

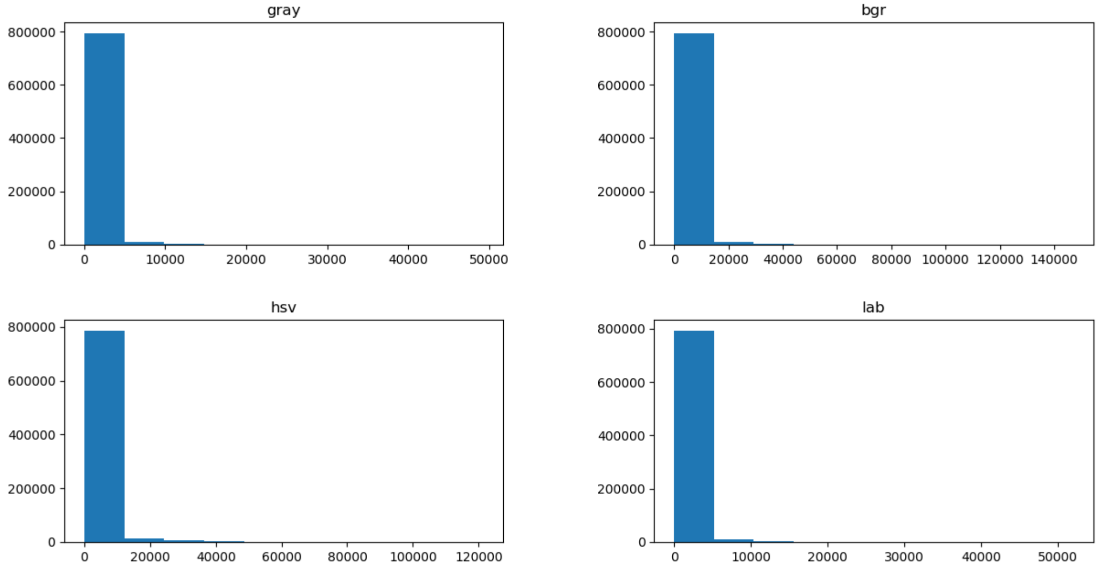
For neighbor (x,y+1):



Run time (one neighbor type and all four histograms):

--- 21.988512754440308 seconds ---

For neighbor (x+1,y):



Run time (one neighbor type and all four histograms):

```
--- 20.672368049621582 seconds ---
```

Question 2:

| | | | |
|---|---|---|---|
| 3 | 1 | 2 | 1 |
| 2 | 2 | 0 | 2 |
| 1 | 2 | 1 | 1 |
| 1 | 0 | 1 | 2 |

a) V = [0,1]
**4 path**: There is no 4-path between p and q, as none of the 4 neighbors of pixel q have values from V

| | | | |
|---|---|---|---|
| 3 | 1 | **2** | 1 |
| 2 | 2 | 0 | **2** |
| 1 | 2 | 1 | 1 |
| 1 | 0 | 1 | 2 |

**8 path**:

| | | | |
|---|---|---|---|
| 3 | 1 | 2 | 1 |
| 2 | 2 | 0 | 2 |
| 1 | 2 | 1 | 1 |
| 1 | 0 | 1 | 2 |

The length of the shortest 8-path is 4. The path is as highlighted.

(3,0) (3,1) (2,2) (1,2) (0,3)

**M path**:

| | | | |
|---|---|---|---|
| 3 | 1 | 2 | 1 |
| 2 | 2 | 0 | 2 |
| 1 | 2 | 1 | 1 |
| 1 | 0 | 1 | 2 |

The length of the path is 5. The path is as highlighted:

(3,0) (3,1) (3,2) (2,2) (1,2) (0,3)

b) V = [1,2]
**4 path**: Many possibilities for 4-path. The length of the shortest path is 6.

| | | | |
|---|---|---|---|
| 3 | 1 | 2 | 1 |
| 2 | 2 | 0 | 2 |
| 1 | 2 | 1 | 1 |
| 1 | 0 | 1 | 2 |

One such path is: (3,0) (2,0) (2,1) (1,1) (0,1) (0,2) (0,3)

**8 path**:

| | | | |
|---|---|---|---|
| 3 | 1 | 2 | 1 |
| 2 | 2 | 0 | 2 |
| 1 | 2 | 1 | 1 |
| 1 | 0 | 1 | 2 |

Many possibilities for 8 paths. The length of the shortest 8-path is 4. The path is as highlighted.

(3,0) (3,1) (2,2) (1,3) (0,3)

**M path**:

$$\begin{matrix} 3 & 1 & 2 & 1 \\ 2 & 2 & 0 & 2 \\ 1 & 2 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{matrix}$$

The length of the path is 6. The path is as highlighted:

(3,0) (2,0) (2,1) (1,1) (0,1) (0,2) (0,3)

c) Used the breadth first search algorithm for finding the shortest path

1)

Input parameters: Start position (3,0), End position (0,3), V = [0,1]

Input matrix:

$$\begin{matrix} 3 & 1 & 2 & 1 \\ 2 & 2 & 0 & 2 \\ 1 & 2 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{matrix}$$

Output:

```
4-path

None

8-path

shortest path: [(3, 0), (3, 1), (2, 2), (1, 2), (0, 3)] length of the shortest path:  4

m-path

shortest path: [(3, 0), (3, 1), (3, 2), (2, 2), (1, 2), (0, 3)] length of the shortest path:  5

--- Run time: 0.339019775390625 seconds ---
```

2)

Input matrix: same as before, Start position (3,0), End position (0,3), V = [1,2]

Output:

```
4-path

shortest path: [(3, 0), (2, 0), (2, 1), (2, 2), (2, 3), (1, 3), (0, 3)] length of the shortest path:  6

8-path

shortest path: [(3, 0), (2, 0), (1, 1), (0, 2), (0, 3)] length of the shortest path:  4

m-path

shortest path: [(3, 0), (2, 0), (2, 1), (2, 2), (2, 3), (1, 3), (0, 3)] length of the shortest path:  6

--- Run time: 0.3484315872192383 seconds ---
```

3)

Input matrix: same as before, Start position (3,0), End position (0,3), V = [0,1,2]

Output:

***4-path***

***shortest path: [(3, 0), (3, 1), (3, 2), (3, 3), (2, 3), (1, 3), (0, 3)] length of the shortest path: 6***

***8-path***

***shortest path: [(3, 0), (2, 1), (1, 2), (0, 3)] length of the shortest path: 3***

***m-path***

***shortest path: [(3, 0), (3, 1), (3, 2), (3, 3), (2, 3), (1, 3), (0, 3)] length of the shortest path: 6***

***--- Run time: 0.3281121253967285 seconds ---***

4)

Input matrix: image(wolves.png), Start position (3,0), End position (0,3), V = range (1,256) (values from 0,1,2….255)

Output:

***4-path***

***shortest path: [(3, 0), (3, 1), (3, 2), (3, 3), (2, 3), (1, 3), (0, 3)] length of the shortest path: 6***

***8-path***

***shortest path: [(3, 0), (2, 1), (1, 2), (0, 3)] length of the shortest path: 3***

***m-path***

***shortest path: [(3, 0), (3, 1), (3, 2), (3, 3), (2, 3), (1, 3), (0, 3)] length of the shortest path: 6***

***--- Run time: 0.37269139289855957 seconds ---***

5)

Input matrix: image(wolves.png), Start position (3,0), End position (0,100), V = range (1,256) (values from 0,1,2….255)

Output:

***4-path***

***shortest path: [(3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (3, 11), (3, 12), (3, 13), (3, 14), (3, 15), (3, 16), (3, 17), (3, 18), (3, 19), (3, 20), (3, 21), (3, 22), (3, 23), (3, 24), (3, 25), (3, 26), (3, 27), (3, 28), (3, 29), (3, 30), (3, 31), (3, 32), (3, 33), (3, 34), (3, 35), (3, 36), (3, 37), (3, 38), (3, 39), (3, 40), (3, 41), (3, 42), (3, 43), (3, 44), (3, 45), (3, 46), (3, 47), (3, 48), (3, 49), (3, 50), (3, 51), (3, 52), (3, 53), (3, 54), (3, 55), (3, 56), (3, 57), (3, 58), (3, 59), (3, 60), (3, 61), (3, 62), (3, 63), (3, 64), (3, 65), (3, 66), (3, 67), (3, 68), (3, 69), (3, 70), (3, 71), (3, 72), (3, 73), (3, 74), (3, 75), (3, 76), (3, 77), (3, 78), (3, 79), (3, 80), (3, 81), (3, 82), (3, 83), (3, 84), (3, 85), (3, 86), (3, 87), (3, 88), (3, 89), (3, 90), (3, 91), (3, 92), (3, 93), (3, 94), (3, 95), (3, 96), (3, 97), (3, 98), (3, 99), (3, 100), (2, 100), (1, 100), (0, 100)] length of the shortest path: 103***

***8-path***

```
shortest path:  [(3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (3, 11), (3, 12), (3, 13), (3,
14), (3, 15), (3, 16), (3, 17), (3, 18), (3, 19), (3, 20), (3, 21), (3, 22), (3, 23), (3, 24), (3, 25), (3, 26), (3, 27), (3, 28), (3,
29), (3, 30), (3, 31), (3, 32), (3, 33), (3, 34), (3, 35), (3, 36), (3, 37), (3, 38), (3, 39), (3, 40), (3, 41), (3, 42), (3, 43), (3,
44), (3, 45), (3, 46), (3, 47), (3, 48), (3, 49), (3, 50), (3, 51), (3, 52), (3, 53), (3, 54), (3, 55), (3, 56), (3, 57), (3, 58), (3,
59), (3, 60), (3, 61), (3, 62), (3, 63), (3, 64), (3, 65), (3, 66), (3, 67), (3, 68), (3, 69), (3, 70), (3, 71), (3, 72), (3, 73), (3,
74), (3, 75), (3, 76), (3, 77), (3, 78), (3, 79), (3, 80), (3, 81), (3, 82), (3, 83), (3, 84), (3, 85), (3, 86), (3, 87), (3, 88), (3,
89), (3, 90), (3, 91), (3, 92), (3, 93), (3, 94), (3, 95), (3, 96), (3, 97), (2, 98), (1, 99), (0, 100)] length of the shortest
path:  100

m-path

shortest path:  [(3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (3, 11), (3, 12), (3, 13), (3,
14), (3, 15), (3, 16), (3, 17), (3, 18), (3, 19), (3, 20), (3, 21), (3, 22), (3, 23), (3, 24), (3, 25), (3, 26), (3, 27), (3, 28), (3,
29), (3, 30), (3, 31), (3, 32), (3, 33), (3, 34), (3, 35), (3, 36), (3, 37), (3, 38), (3, 39), (3, 40), (3, 41), (3, 42), (3, 43), (3,
44), (3, 45), (3, 46), (3, 47), (3, 48), (3, 49), (3, 50), (3, 51), (3, 52), (3, 53), (3, 54), (3, 55), (3, 56), (3, 57), (3, 58), (3,
59), (3, 60), (3, 61), (3, 62), (3, 63), (3, 64), (3, 65), (3, 66), (3, 67), (3, 68), (3, 69), (3, 70), (3, 71), (3, 72), (3, 73), (3,
74), (3, 75), (3, 76), (3, 77), (3, 78), (3, 79), (3, 80), (3, 81), (3, 82), (3, 83), (3, 84), (3, 85), (3, 86), (3, 87), (3, 88), (3,
89), (3, 90), (3, 91), (3, 92), (3, 93), (3, 94), (3, 95), (3, 96), (3, 97), (3, 98), (3, 99), (3, 100), (2, 100), (1, 100), (0,
100)] length of the shortest path:  103

--- Run time: 13.111051321029663 seconds ---
```

**Question 1 Code:**

Code:

```
import time
start_time = time.time()
import cv2
import numpy as np
import matplotlib.pyplot as plt

type = 'x+1','y' #define neighbour type

img = cv2.imread('C:/Users/Shubham/Desktop/Fall19/imagin 558/ECE558-
HW01/ECE558-HW01/wolves.png')

b,g,r = cv2.split(img)
bgr = np.ndarray(shape=(img.shape[0],img.shape[1]))

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
graynew = np.ndarray(shape=(img.shape[0],img.shape[1]))

h,s,v = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2HSV))
hsv = np.ndarray(shape=(img.shape[0],img.shape[1]))

l,a,ba = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2Lab))
lab = np.ndarray(shape=(img.shape[0],img.shape[1]))


def error(a,b): #define error function
    v = (a-b)*(a-b)
    return v

def neighbour(a,b):
    if a == 'x+1' and b == 'y':
        return 1,0
    elif a == 'x+1' and b == 'y+1':
        return 1,1
    elif a == 'x' and b == 'y+1':
        return 0,1
    elif a == 'x-1' and b == 'y+1':
        return -1,1
    elif a == 'x-1' and b == 'y':
        return -1,0
    elif a == 'x-1' and b == 'y-1':
        return -1,-1
    elif a == 'x' and b == 'y-1':
        return 0,-1
    elif a == 'x+1' and b == 'y-1':
        return 1,-1



i,j = neighbour(type[0],type[1])
for c1 in range(0,539):
    for c2 in range(0,1500):
        if c1>=max(0,-i) and c2>=max(0,-j) and c1<=img.shape[1]-1-max(0,i)
and c2<=img.shape[0]-1-max(0,j):
```

```
            if c1+j>=max(0,-i) and c2+i>=max(0,-j) and c1+j<=img.shape[1]-1-
max(0,i) and c2+i<=img.shape[0]-1-max(0,j):

graynew[c2][c1]=error(int(gray[c2][c1]),int(gray[c2+i][c1+j]))

for c1 in range(0,539):
    for c2 in range(0,1500):
        if c1>=max(0,-i) and c2>=max(0,-j) and c1<=img.shape[1]-1-max(0,i)
and c2<=img.shape[0]-1-max(0,j):
            if c1+j>=max(0,-i) and c2+i>=max(0,-j) and c1+j<=img.shape[1]-1-
max(0,i) and c2+i<=img.shape[0]-1-max(0,j):

bgr[c2][c1]=error(int(b[c2][c1]),int(b[c2+i][c1+j]))+error(int(g[c2][c1]),int
(g[c2+i][c1+j]))+error(int(r[c2][c1]),int(r[c2+i][c1+j]))

for c1 in range(0,539):
    for c2 in range(0,1500):
        if c1>=max(0,-i) and c2>=max(0,-j) and c1<=img.shape[1]-1-max(0,i)
and c2<=img.shape[0]-1-max(0,j):
            if c1+j>=max(0,-i) and c2+i>=max(0,-j) and c1+j<=img.shape[1]-1-
max(0,i) and c2+i<=img.shape[0]-1-max(0,j):


hsv[c2][c1]=error(int(h[c2][c1]),int(h[c2+i][c1+j]))+error(int(s[c2][c1]),int
(s[c2+i][c1+j]))+error(int(v[c2][c1]),int(v[c2+i][c1+j]))

for c1 in range(0,539):
    for c2 in range(0,1500):
        if c1>=max(0,-i) and c2>=max(0,-j) and c1<=img.shape[1]-1-max(0,i)
and c2<=img.shape[0]-1-max(0,j):
            if c1+j>=max(0,-i) and c2+i>=max(0,-j) and c1+j<=img.shape[1]-1-
max(0,i) and c2+i<=img.shape[0]-1-max(0,j):


lab[c2][c1]=error(int(l[c2][c1]),int(l[c2+i][c1+j]))+error(int(a[c2][c1]),int
(a[c2+i][c1+j]))+error(int(ba[c2][c1]),int(ba[c2+i][c1+j]))


fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2)
ax1.hist(graynew.ravel())
ax1.set_title('gray')
ax2.hist(bgr.ravel())
ax2.set_title('bgr')
ax3.hist(hsv.ravel())
ax3.set_title('hsv')
ax4.hist(lab.ravel())
ax4.set_title('lab')
fig.tight_layout()
print("--- %s seconds ---" % (time.time() - start_time))
plt.show()
```

**Question 2 Code:**

```
import time
start_time = time.time()
import numpy as np
import collections
import cv2

img = cv2.imread('C:/Users/Shubham/Desktop/Fall19/imagin 558/ECE558-
HW01/ECE558-HW01/wolves.png')
V = [0,1] #set V for matrix, comment when using an image
#V = list(range(256)) #set V for image, uncomment when using an image
a = np.array([[3, 1, 2,1], [2, 2, 0,2], [1, 2, 1,1], [1,0,1,2]]) #comment
when using an image

p = (3,0) #first point
q = (0,3) #end point
#all paths will be printed together
#uncomment next two lines if required to use image
#gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#a = gray

endp = q
def mpath(grid, start,end):
    queue = collections.deque([[start]])
    seen = set([start])

    while queue:
        path = queue.popleft()
        x, y = path[-1]
        if y == end[0] and x == end[1]:
            return path
        listofpossible = []
        for (x1,y1) in ((x+1,y), (x-1,y), (x,y+1), (x,y-1)):
            if 0 <= x1 < width and 0 <= y1 < height:

                if x1==0:
                    listofpossible.append((0,y1))
                else:
                    listofpossible.append((x1,y1))
        for (x2, y2) in  ((x+1,y), (x-1,y), (x,y+1), (x,y-1),(x-1,y-
1),(x+1,y+1),(x-1,y+1),(x+1,y-1)):
            if (x2,y2) in ((x-1,y-1),(x+1,y+1),(x-1,y+1),(x+1,y-1)):
                listofpossible2 = []
                if 0 <= x2 < width and 0 <= y2 < height:
                    for(x3,y3) in  ((x2+1,y2), (x2-1,y2), (x2,y2+1), (x2,y2-
1)):

                        if 0 <= x3 < width and 0 <= y3 < height:

                            if x3==0:
                                listofpossible2.append((0,y3))
                            else:
                                listofpossible2.append((x3,y3))
                    inter = intersection(listofpossible,listofpossible2)
                    if inter==0 and (grid[y2][x2] in V or y2 == end[0] and x2
== end[1]) and (x2, y2) not in seen:
```

```
                        queue.append(path + [(x2, y2)])
                        seen.add((x2, y2))
            if (x2,y2) in  ((x+1,y), (x-1,y), (x,y+1), (x,y-1)):

                if 0 <= x2 < width and 0 <= y2 < height and (grid[y2][x2] in
V or y2 == end[0] and x2 == end[1]) and (x2, y2) not in seen:
                    queue.append(path + [(x2, y2)])
                    seen.add((x2, y2))
def eightpath(grid, start,end):
    queue = collections.deque([[start]])
    seen = set([start])

    while queue:
        path = queue.popleft()
        x, y = path[-1]
        if y == end[0] and x == end[1]:
            return path
        for x2, y2 in ((x+1,y), (x-1,y), (x,y+1), (x,y-1),(x-1,y-
1),(x+1,y+1),(x-1,y+1),(x+1,y-1)):
            if 0 <= x2 < width and 0 <= y2 < height and (grid[y2][x2] in V or
y2 == end[0] and x2 == end[1]) and (x2, y2) not in seen:
                queue.append(path + [(x2, y2)])
                seen.add((x2, y2))

def fourpath(grid, start,end):
    queue = collections.deque([[start]])
    seen = set([start])

    while queue:
        path = queue.popleft()
        x, y = path[-1]
        if y == end[0] and x == end[1]:
            return path
        for x2, y2 in ((x+1,y), (x-1,y), (x,y+1), (x,y-1)):
            if 0 <= x2 < width and 0 <= y2 < height and (grid[y2][x2] in V or
y2 == end[0] and x2 == end[1]) and (x2, y2) not in seen:
                queue.append(path + [(x2, y2)])
                seen.add((x2, y2))




width, height = a.shape[1], a.shape[0]

def Reverse(tuples):
    new_tup = ()
    for k in reversed(tuples):
        new_tup = new_tup + (k,)
    return new_tup

def Reversel(l):
    if l != None:
        for i in range(len(l)):
            l[i] = Reverse(l[i])
        print('shortest path: ',l, 'length of the shortest path: ', len(l)-1)
    else: print('None')
def intersection(r, s):
    a_set = set(r)
    b_set = set(s)
    intersection = a_set & b_set
    values = []
    for i in intersection:
        values.append(a[i[1]][i[0]])
    for v in V:
        if v in values:
            return 1
    return 0

if 0 <= endp[1] < width and 0 <= endp[0] < height and 0 <= Reverse(p)[0] <
height and 0 <= Reverse(p)[1] < width:

    fpath = fourpath(a, Reverse(p),endp)
    epath = eightpath(a,Reverse(p),endp)
    mpath = mpath(a, Reverse(p),endp)
    print('4-path')
    Reversel(fpath)
    print('8-path')
    Reversel(epath)
    print('m-path')
    Reversel(mpath)
else:
    print('invalid start or end position')
print("--- Run time: %s seconds ---" % (time.time() - start_time))
```