NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

IMAGE PROCESSING LAB FILE

DIVISION OF INFORMATION TECHNOLOGY

SHIV KUMAR

2016UIT2563
6TH SEMESTER
IT-2

INDEX

Grey and 3 channel separation

Grey and 3 channel separation

```
In [4]: import cv2
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [5]: image = cv2.imread("flower.jpeg", cv2.IMREAD_COLOR)
```

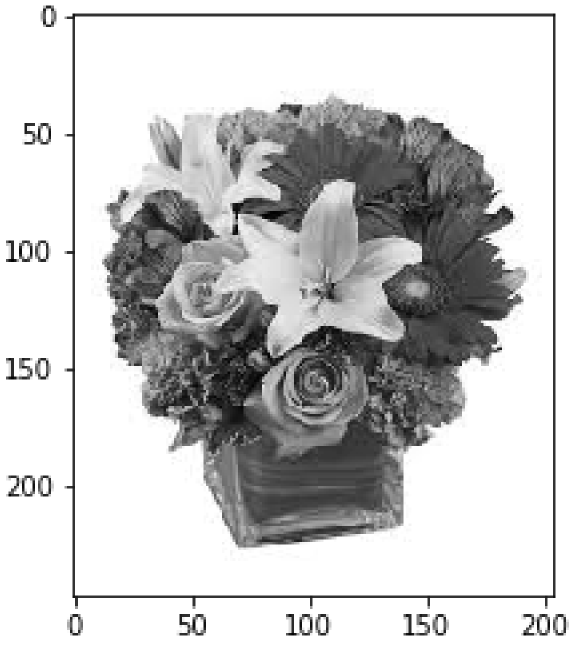
```
In [6]: image[:, :, 0], image[:, :, 2] = np.array(image[:, :, 2]), np.array(image[:, :, 0])
```

```
In [8]: shape = image.shape
        gray_image = np.zeros(shape[:2])
```

```
In [10]: for i in range(shape[0]):
             for j in range(shape[1]):
                     red   = image[i][j][0]
                     green = image[i][j][1]
                     blue  = image[i][j][2]
                     gray_image[i][j] = 0.3 * red + 0.59 * green + 0.11 * blue
```

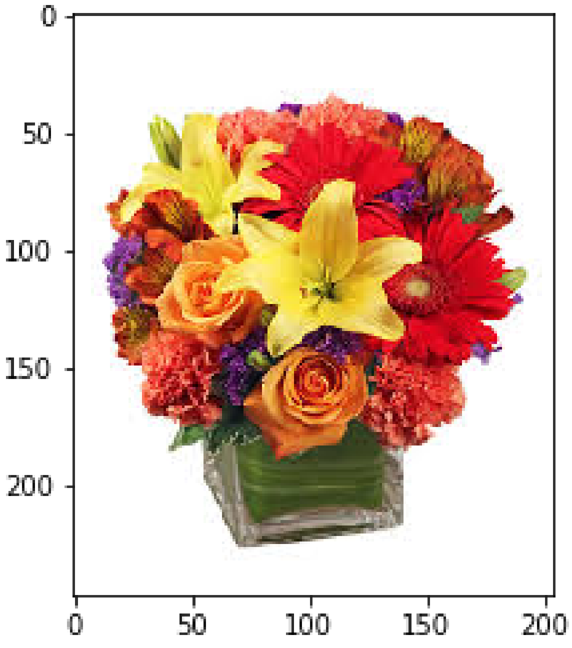
```
In [11]: plt.imshow(gray_image, cmap='gray')
```

```
Out[11]: <matplotlib.image.AxesImage at 0x12a8e3e48>
```

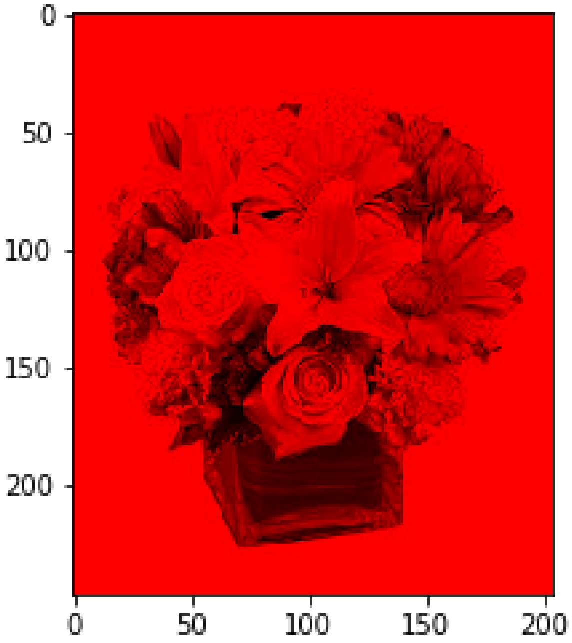


```
In [12]: plt.imshow(image)
```

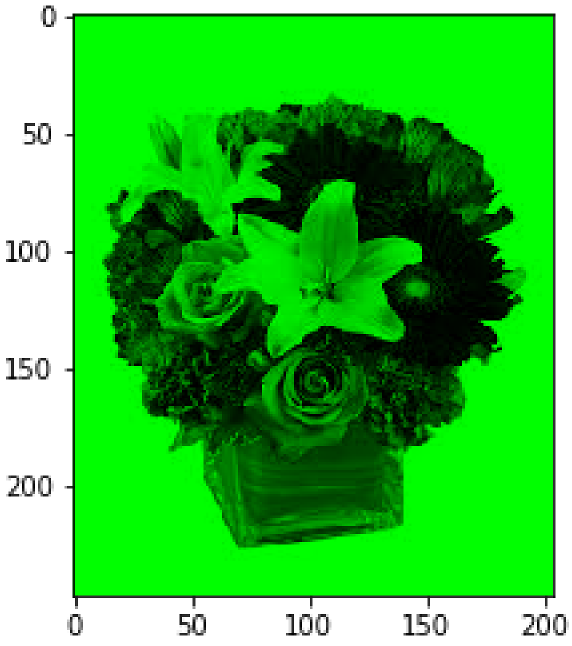
```
Out[12]: <matplotlib.image.AxesImage at 0x12ab10588>
```



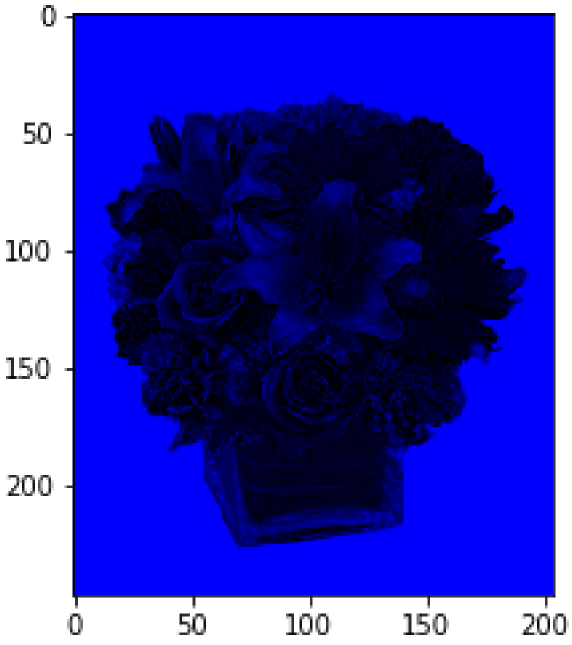
```
In [14]: red_channel = np.array(image)
         red_channel[:, :, 1:3] = 0
         plt.imshow(red_channel)

Out[14]: <matplotlib.image.AxesImage at 0x12aae2b70>
```



```
In [16]: green_channel = np.array(image)
         green_channel[:, :, 0] = 0
         green_channel[:, :, 2] = 0
         plt.imshow(green_channel)

Out[16]: <matplotlib.image.AxesImage at 0x12a9992e8>
```



```
In [17]: blue_channel = np.array(image)
         blue_channel[:, :, 0:2] = 0
         plt.imshow(blue_channel)

Out[17]: <matplotlib.image.AxesImage at 0x12ad253c8>
```



```
In [ ]:
```

Distance Measure

```
In [1]: import numpy as np
        import cv2
        import matplotlib.pyplot as plt

In [2]: img = cv2.imread('Echoes1.jpg')
        img = np.array(img)
        img[:,:,0],img[:,:,2] = img[:,:,2],img[:,:,0].copy()
        plt.imshow(img)

Out[2]: <matplotlib.image.AxesImage at 0x1298e2c88>
```

```
In [6]: print(img[0][0])

[88 77 57]
```

```
In [1]: class Pixel():

            def __init__(self,x,y):
                self.x = x
                self.y = y
```

```
In [3]: np.power(2,3)
```

```
Out[3]: 8
```

```
In [4]: def euclid(p1,p2):
            dis = np.sqrt(np.power((p1.x-p2.x),2)+np.power((p1.y-p2.y),2))
            return dis
```

```
In [5]: def city_block(p1,p2):
            return np.absolute(p1.x-p2.x)+np.absolute(p1.y-p2.y)
```

```
In [6]: def chessboard(p1,p2):
            return max(np.absolute(p1.x-p2.x),np.absolute(p1.y-p2.y))
```

```
In [8]: p1 = Pixel(2,10)
        p2 = Pixel(8,6)
```

```
In [9]: print(euclid(p1,p2))
```

```
7.211102550927978
```

```
In [10]: print(city_block(p1,p2))
```

```
10
```

```
In [11]: print(chessboard(p1,p2))
```

```
6
```

```
In [ ]:
```

steganography

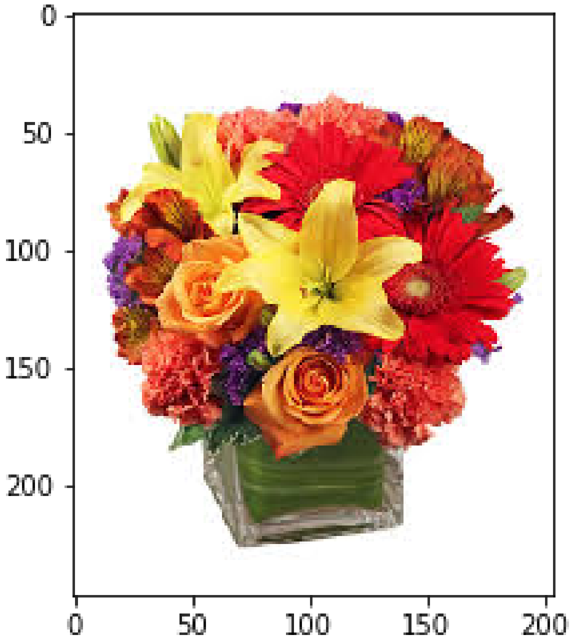
```
In [1]: # STEGANOGRAPHY.py
        import numpy as np
        import cv2
        import matplotlib.pyplot as plt
```

```
In [3]: img = cv2.imread('Echoes1.jpg')
        img = np.array(img)
        img[:,:,0],img[:,:,2] = img[:,:,2],img[:,:,0].copy()
        plt.imshow(img)
```

```
Out[3]: <matplotlib.image.AxesImage at 0x11eff2048>
```

```
In [4]: img2 = cv2.imread('flower.jpeg')
        img2 = np.array(img2)
        img2[:,:,0],img2[:,:,2] = img2[:,:,2],img2[:,:,0].copy()
        plt.imshow(img2)
```

```
Out[4]: <matplotlib.image.AxesImage at 0x129c99ef0>
```



```
In [5]: print(img.shape)
        print(img2.shape)
```

```
(3000, 4000, 3)
(247, 204, 3)
```

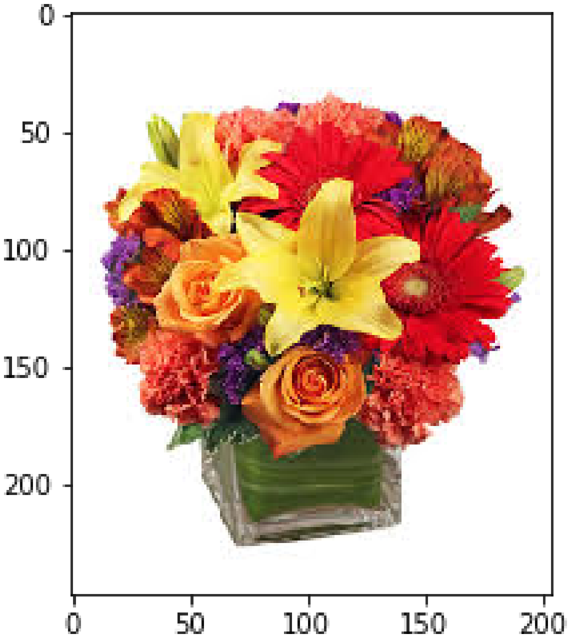
```
In [6]: # function for steganography
        def steganography(img, img2):
            for k in range(3):
                for i in range(img2.shape[0]):
                    col = 0
                    for j in range(img2.shape[1]):
                        c = 0
                        while c != 8:
                            if img[i][col][k] % 2:
                                img[i][col][k] -= 1
                            img[i][col][k] += img2[i][j][k] % 2
                            img2[i][j][k] //= 2
                            col = col + 1
                            c = c + 1
            return img
```

```
In [7]: steg = img.copy()
        to_hide = img2.copy()
        steg = steganography(steg,  to_hide)
```

```
In [8]: plt.imshow(steg)

        steg2 = steg.copy()
        steg2[:, :, 0], steg2[:, :, 2] = steg2[:, :, 2], steg2[:, :, 0].copy()
        cv2.imwrite('STEGANOGRAPHY.png', steg2, [0]) # for preserving quality

Out[8]: True
```

```
In [9]: # function for extracting image
        def extraction(steg, shape):
            ext = np.zeros(shape, dtype = int)
            for k in range(3):
                for i in range(shape[0]):
                    for j in range(shape[1] * 8):
                        if j % 8 == 0:
                            pro = 1
                        else:
                            pro *= 2
                        ext[i][j // 8][k] += steg[i][j][k] % 2 * pro
            return ext

In [10]: steg_img = cv2.imread('STEGANOGRAPHY.png', cv2.IMREAD_COLOR)
         steg_img = np.array(steg_img)
         steg_img[:, :, 0], steg_img[:, :, 2] = steg_img[:, :, 2], steg_img[:, :, 0].copy()
         plt.imshow(steg_img)

Out[10]: <matplotlib.image.AxesImage at 0x12edb4c88>
```

```
In [11]: # Extracting from steganographic image
         extract = extraction(steg_img, img2.shape)

In [12]: plt.imshow(extract)

Out[12]: <matplotlib.image.AxesImage at 0x13614ba90>
```



```
In [ ]:
```

Connectivity

```
In [18]: class pixel:

             def __init__(self,x,y):
                 self.x = x
                 self.y = y
```

```
In [23]: def four_way_check(p1,p2):

             x1,y1 = p1.x,p1.y
             x2,y2 = p2.x,p2.y

             if x2==x1:
                 if y2-1==y1:
                     return True
                 elif y2+1==y2:
                     return True
             elif y2==y1:
                 if x2-1==x1:
                     return True
                 elif x2+1==x1:
                     return True
             else:
                 return False
```

```
In [29]: def four_way(p1):

             x1,y1 = p1.x,p1.y
             neig_four = [(x1-1,y1),(x1+1,y1),(x1,y1-1),(x1,y1+1)]

             return neig_four
```

```
In [31]: def d_way(p1):
             x1,y1 = p1.x,p1.y
             neig_d = [(x1-1,y1-1),(x1-1,y1+1),(x1+1,y1-1),(x1+1,y1+1)]
             return neig_d
```

```
In [33]: def eight_way(p1):
             x1,y1 = p1.x,p1.y
```

```
            eight = [(x1-1,y1-1),(x1-1,y1),(x1-1,y1+1),(x1,y1-1),(x1,y1+1),(x1+1,y1-1),(x1+1,y1
            return eight
```

```
In [38]: print(four_way(p2))
```

```
[(0, 0), (2, 0), (1, -1), (1, 1)]
```

```
In [39]: print(d_way(p2))
```

```
[(0, -1), (0, 1), (2, -1), (2, 1)]
```

```
In [40]: print(eight_way(p2))
```

```
[(0, -1), (0, 0), (0, 1), (1, -1), (1, 1), (2, -1), (2, 0), (2, 1)]
```

```
In [21]: p1 = pixel(0,0)
         p2 = pixel(1,0)

         print(four_way(p1))
```

```
True
```

```
In [7]: p3 = pixel(1,1)
        print(four_way(p1,p3))
```

```
False
```

```
In [24]: def d_way_check(p1,p2):

             x1,y1 = p1.x,p1.y
             x2,y2 = p2.x,p2.y

             if x2-1==x1 and y2-1==y1:
                 return True
             elif x2-1==x1 and y2+1==y1:
                 return True
             elif x2+1==x1 and y2-1==y1:
                 return True
             elif x2+1==x1 and y2+1==y1:
                 return True
             else:
                 return False
```

```
In [9]: d_way(p1,p2)
```

```
Out[9]: False
```

```
In [10]: d_way(p1,p3)
```

```
Out[10]: True
```

```
In [25]: def eight_way_check(p1,p2):

             x1,y1 = p1.x,p1.y
             x2,y2 = p2.x,p2.y

             eight = [(x1-1,y1-1),(x1-1,y1),(x1-1,y1+1),(x1,y1-1),(x1,y1+1),(x1+1,y1-1),(x1+1,y1
             if (x2,y2) in eight:
                 return True
             else:
                 return False
```

```
In [12]: eight_way(p1,p2)
```

```
Out[12]: True
```

```
In [13]: p4 = pixel(2,3)
```

```
In [14]: eight_way(p1,p4)
```

```
Out[14]: False
```

```
In [26]: def m_way_check(p1,p2):
             x1,y1 = p1.x,p1.y
             x2,y2 = p2.x,p2.y

             eight = [(x1-1,y1-1),(x1-1,y1),(x1-1,y1+1),(x1,y1-1),(x1,y1+1),(x1+1,y1-1),(x1+1,y1
             if (x2,y2) in eight:
                 return True
             else:
                 return False
```

```
In [16]: m_way(p1,p2)
```

```
Out[16]: True
```

```
In [17]: m_way(p1,p4)
```

```
Out[17]: False
```

```
In [ ]:
```

Connected Components

```
In [2]: import cv2
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [3]: shp = (25,25)
        img = np.floor(np.random.random(shp) + 0.5)
```

```
In [4]: def four_way(out, i, j, color, label):

            if i < 0 or i>= shp[0]:
                return
            if j < 0 or j >= shp[1]:
                return
            if vis[i][j] or img[i][j]==1:
                return

            vis[i][j] = True
            out[i][j] = color
            labels[i][j] = label
            four_way(out, i - 1, j, color,label)
            four_way(out, i + 1, j, color,label)
            four_way(out, i, j - 1, color,label)
            four_way(out, i, j + 1, color,label)
```

```
In [5]: def eight_way(out, i, j, color,label):
            if i < 0 or i >= shp[0]:
                return
            if j < 0 or j >= shp[1]:
                return
            if vis[i][j] or img[i][j] == 1:
                return

            vis[i][j] = True
            out[i][j] = color
            labels[i][j] = label

            eight_way(out, i - 1, j, color,label)
            eight_way(out, i + 1, j, color,label)
            eight_way(out, i, j - 1, color,label)
            eight_way(out, i, j + 1, color,label)
            eight_way(out, i - 1, j - 1, color,label)
            eight_way(out, i - 1, j + 1, color,label)
            eight_way(out, i + 1, j - 1, color,label)
            eight_way(out, i + 1, j + 1, color,label)
```

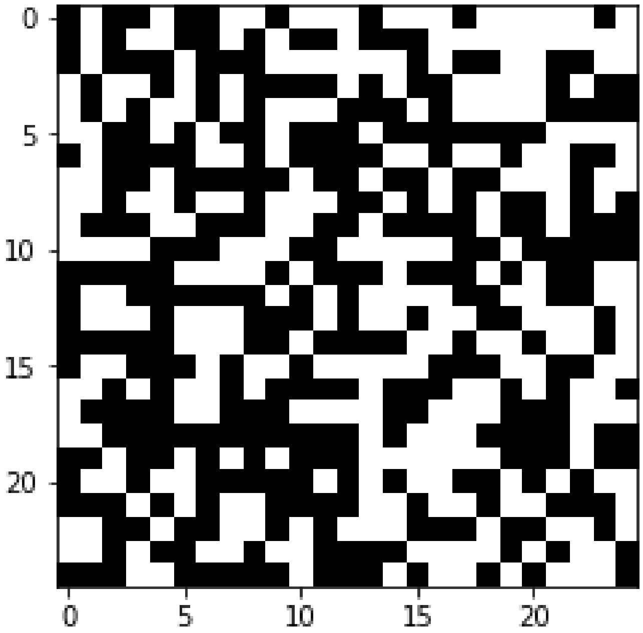
```
In [6]: def m_conn(x1,y1,x2,y2):

            four_way_n_p1 = {}
            four_way_n_p1[(x1-1,y1-1)] = img[x1-1][y1-1]
            four_way_n_p1[(x1-1,y1)] = img[x1-1][y1]
            four_way_n_p1[(x1,y1-1)] = img[x1][y1-1]
            four_way_n_p1[(x1,y1)] = img[x1][y1]

            n_p2 = [(x2-1,y2-1),(x2-1,y2),(x2,y2-1),(x2,y2)]

            for cord in n_p2:
                if cord in four_way_n_p1.keys():
                    if img[cord[0]][cord[1]] == 1:
                        return False

            return True
```

```
In [7]: def m_way(out, i, j, color,label):
            if i<0 or i>= shp[0]:
                return
            if j<0 or j>= shp[1]:
                return
            if vis[i][j] or img[i][j] == 1:
                return

            vis[i][j] = True
            out[i][j] = color
            labels[i][j] = label

            m_way(out, i - 1, j, color,label)
            m_way(out, i + 1, j, color,label)
            m_way(out, i, j - 1, color,label)
            m_way(out, i, j + 1, color,label)

            if m_conn(i,j,i-1,j-1):
                m_way(out, i - 1, j - 1, color,label)
            elif m_conn(i,j,i-1,j+1):
                m_way(out, i - 1, j + 1, color,label)
            elif m_conn(i,j,i+1,j-1):
                m_way(out, i + 1, j - 1, color,label)
            elif m_conn(i,j,i+1,j+1):
                m_way(out, i + 1, j + 1, color,label)
```

```
In [8]: def plot(out):
            fig, ax = plt.subplots(figsize=(20,10))

            ax.imshow(out)
            for i in range(shp[0]):
                for j in range(shp[1]):
                    c = labels[j][i]
                    ax.text(i, j, str(c), va='center', ha='center')
```

```
In [9]: fig = plt.figure(100)
        fig.canvas.set_window_title('Main')
        plt.imshow(img, cmap="Greys")
```

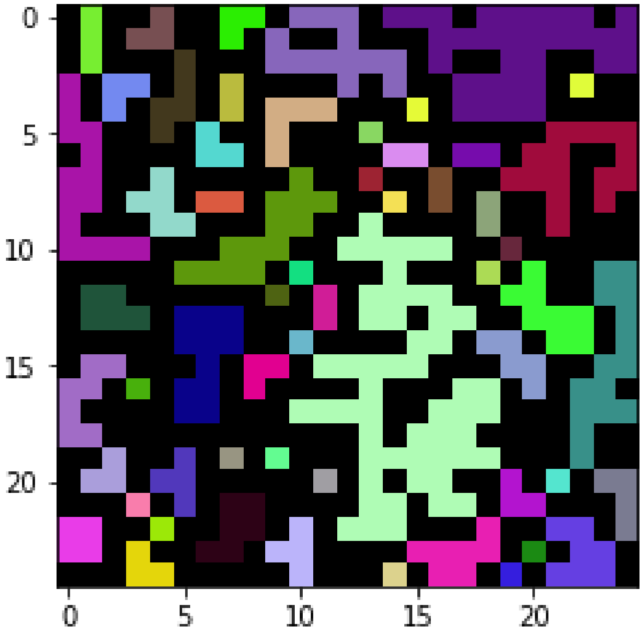
```
Out[9]: <matplotlib.image.AxesImage at 0x1254fed30>
```



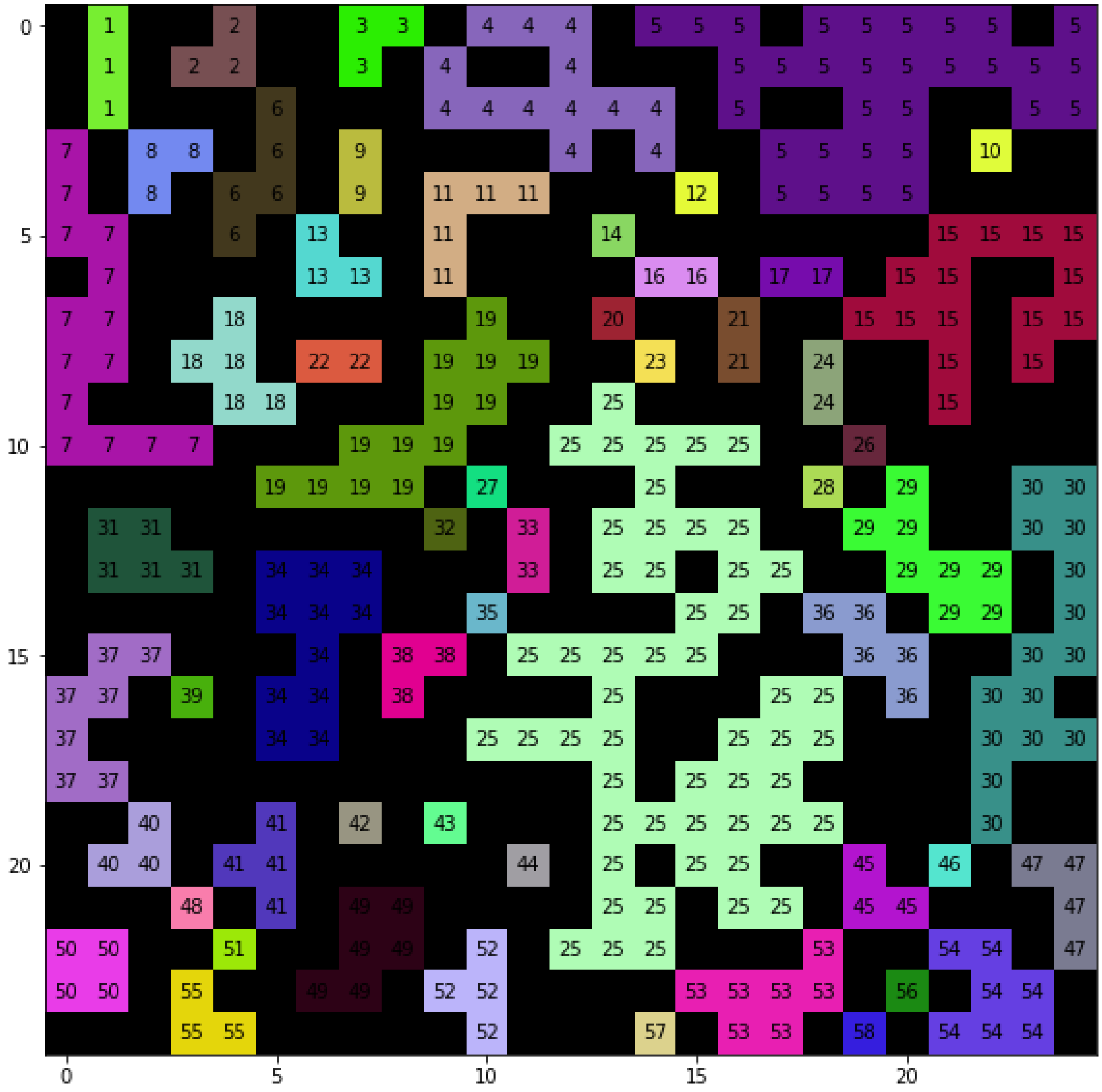
```
In [10]: vis = np.zeros(shp,dtype=bool)
         out = np.zeros(shp + (3, ), dtype=int)
         labels = np.zeros(shp,dtype=int)
         label=1
         for i in range(shp[0]):
             for j in range(shp[1]):
                 if vis[i][j] or img[i][j]==1:
                     continue

                 color = np.random.randint(0, 255, 3)
                 four_way(out, i, j, color,label)
```

```
            label+=1

    fig = plt.figure(200)
    fig.canvas.set_window_title('4-Way')
    plt.imshow(out)
```

```
Out[10]: <matplotlib.image.AxesImage at 0x12566d748>
```



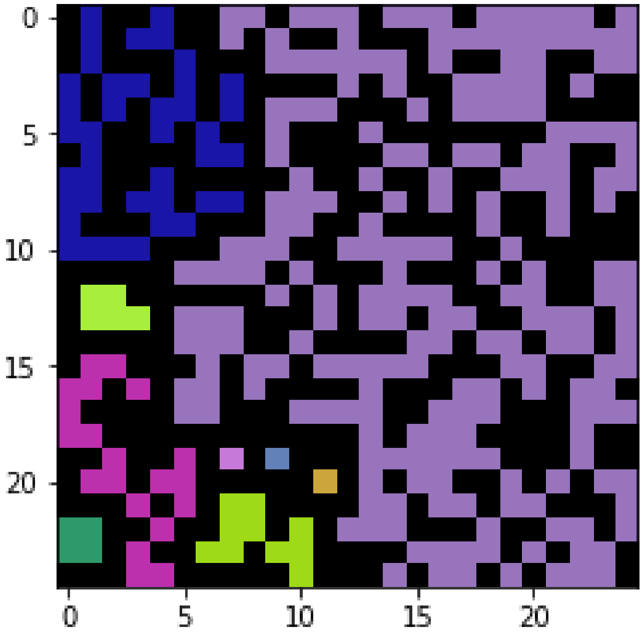
```
In [11]: plot(out)
```



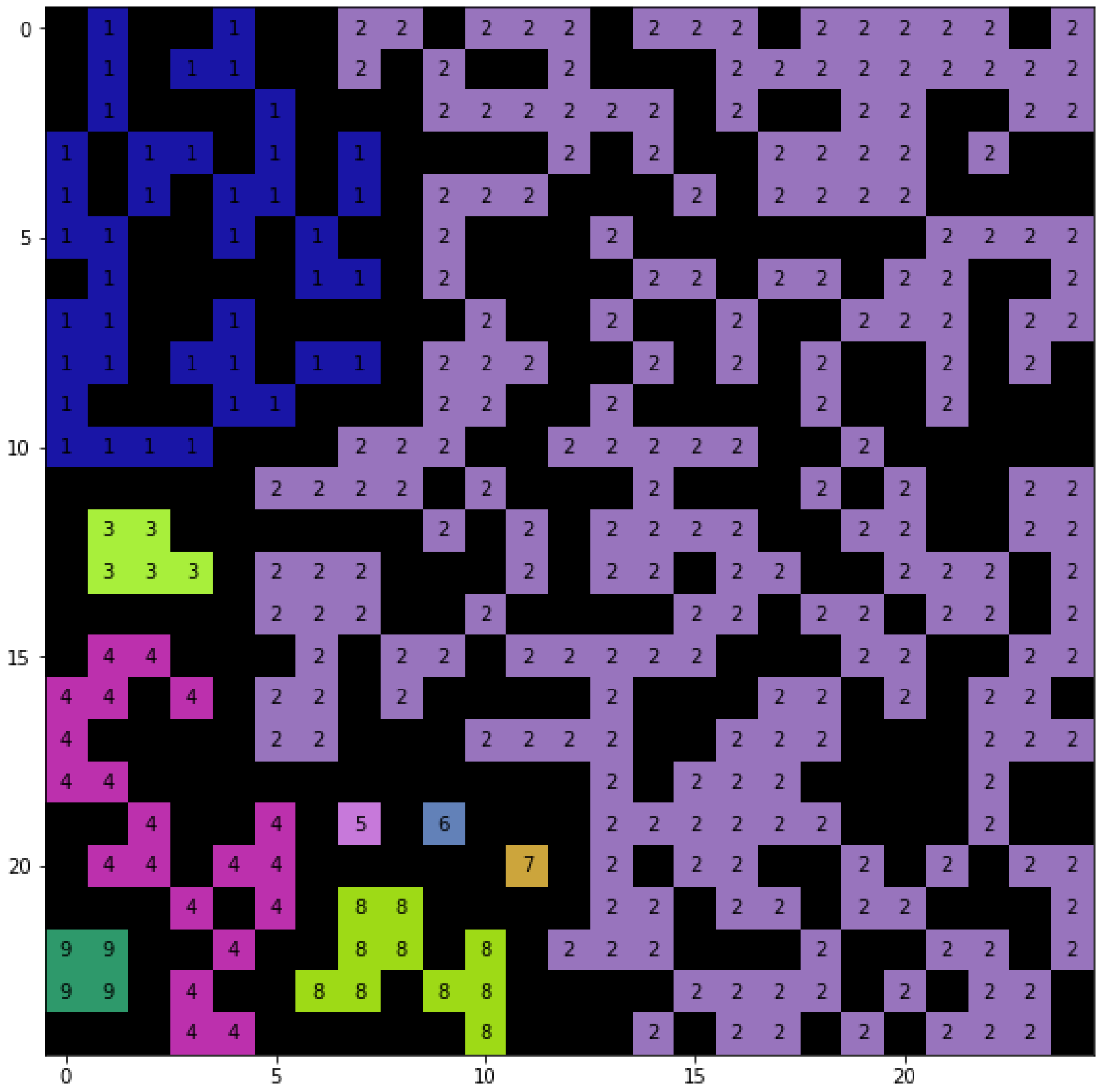
```
In [12]: vis = np.zeros(shp, dtype=bool)
         out = np.zeros(shp + (3, ), dtype=int)
         label=1
         for i in range(shp[0]):
             for j in range(shp[1]):

                 if vis[i][j] or img[i][j]==1:
                     continue
                 color = np.random.randint(0,255,3)
                 eight_way(out, i, j, color,label)
                 label+=1

         fig = plt.figure(300)
```

```
        fig.canvas.set_window_title('8-Way')
        plt.imshow(out)
```

```
Out[12]: <matplotlib.image.AxesImage at 0x125f7dd30>
```

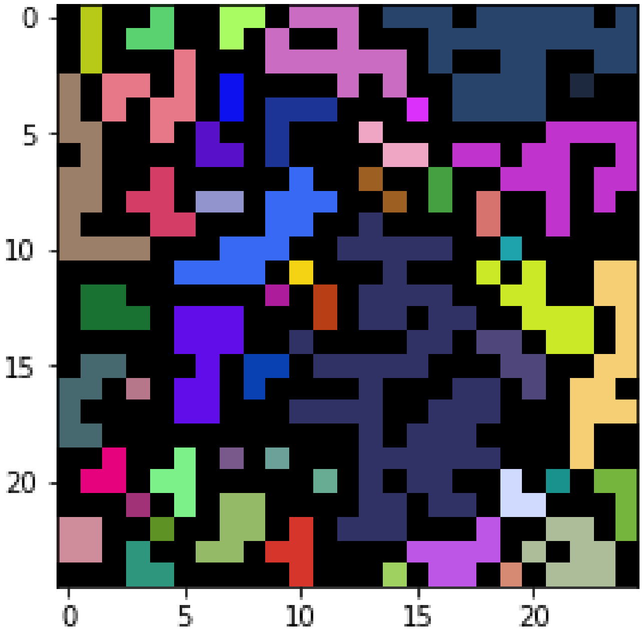


```
In [13]: plot(out)
```

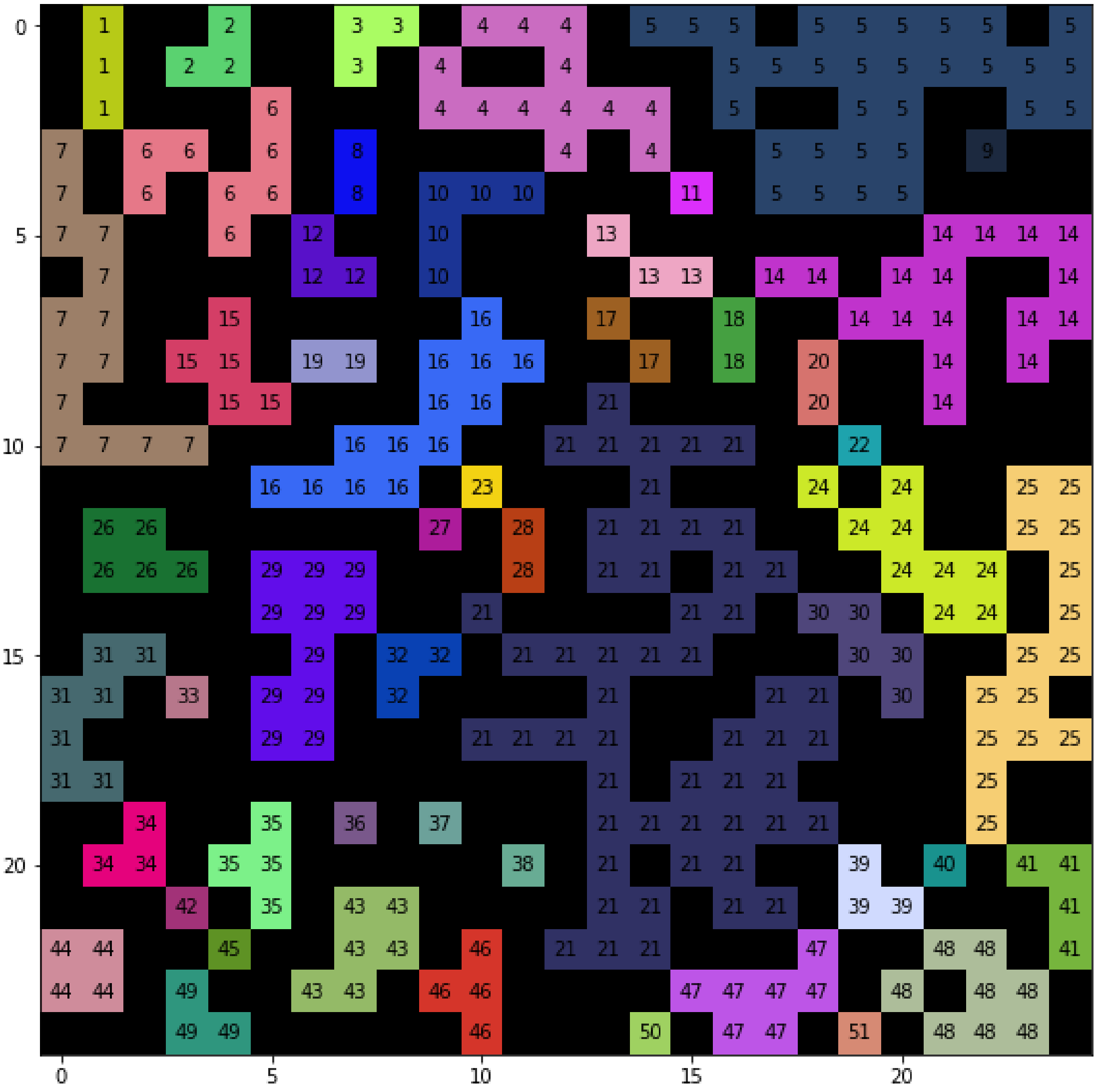


```
In [14]: vis = np.zeros(shp, dtype=bool)
         out = np.zeros(shp + (3, ), dtype=int)
         label=1
         for i in range(shp[0]):
             for j in range(shp[1]):
                 if vis[i][j] or img[i][j]==1:
                     continue

                 color = np.random.randint(0, 255, 3)
                 m_way(out, i, j, color,label)
                 label+=1

         fig = plt.figure(400)
```

```
fig.canvas.set_window_title('m-Way')
plt.imshow(out)
plt.show()
```



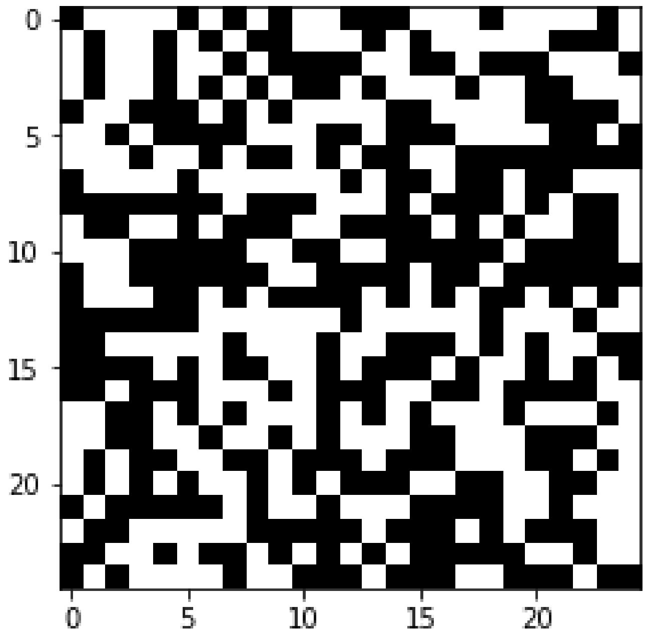
```
In [15]: plot(out)
```


0
5
10
15
20
1 2 3 3 4 4 4 5 5 5 5 5 5 5 5 5
1 2 2 3 4 4 5 5 5 5 5 5 5 5 5
1 6 4 4 4 4 4 4 5 5 5 5 5
7 6 6 6 8 4 4 5 5 5 5 9
7 6 6 6 8 10 10 10 11 5 5 5 5
7 7 6 12 10 13 14 14 14 14
7 12 12 10 13 13 14 14 14 14 14
7 7 15 16 17 18 14 14 14 14 14
7 7 15 15 19 19 16 16 16 17 18 20 14 14
7 15 15 16 16 21 20 14
7 7 7 7 16 16 16 21 21 21 21 21 22
16 16 16 16 23 21 24 24 25 25
26 26 27 28 21 21 21 21 24 24 25 25
26 26 26 29 29 29 28 21 21 21 21 24 24 24 25
29 29 29 21 21 21 30 30 24 24 25
31 31 29 32 32 21 21 21 21 21 30 30 25 25
31 31 33 29 29 32 21 21 21 30 25 25
31 29 29 21 21 21 21 21 21 21 25 25 25
31 31 21 21 21 21 25
34 35 36 37 21 21 21 21 21 21 25
34 34 35 35 38 21 21 21 39 40 41 41
42 35 43 43 21 21 21 21 39 39 41
44 44 45 43 43 46 21 21 21 47 48 48 41
44 44 49 43 43 46 46 47 47 47 47 48 48 48
49 49 46 50 47 47 51 48 48 48
0
5
10
15
20

```
In [ ]:
```

Component Labelling

```
In [1]: import cv2
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: shp = (25,25)
        img = np.floor(np.random.random(shp) + 0.5)

In [3]: def plot(out):
            fig, ax = plt.subplots(figsize=(20,10))

            ax.imshow(out)
            for i in range(shp[0]):
                for j in range(shp[1]):
                    c = out[j,i]
                    ax.text(i, j, str(c), va='center', ha='center')

In [4]: fig = plt.figure(100)
        fig.canvas.set_window_title('Main')
        plt.imshow(img, cmap="Greys")

Out[4]: <matplotlib.image.AxesImage at 0x1195c2828>
```



```
In [5]: def first_pass(img):


            label=1


            for i in range(shp[0]):
                for j in range(shp[1]):
                    if img[i][j]==1:
                        continue
                    if i==0 and j==0:
                        out[i][j] = label
                        if label not in parents:
                            parents[label]=label
                        label+=1
                    elif i==0:
                        if out[i][j-1]==0:
                            out[i][j] = label
                            if label not in parents:
                                parents[label]=label
                            label+=1
                        else:
                            out[i][j] = out[i][j-1]
                    elif j==0:
                        if out[i-1][j-1]==0:
                            out[i][j] = label
                            if label not in parents:
                                parents[label]=label
                            label+=1
                        else:
                            out[i][j] = out[i-1][j]
                    else:
                        if out[i-1][j]==0 and out[i][j-1]==0:
                            out[i][j] = label
                            if label not in parents:
                                parents[label]=label
                            label+=1
                        elif out[i-1][j]==0:
                            out[i][j] = out[i][j-1]
                        elif out[i][j-1]==0:
                            out[i][j] = out[i-1][j]
                        else:
                            out[i][j] = min(out[i-1][j],out[i][j-1])
                            parent1 = parents[out[i-1][j]]
                            current1 = out[i-1][j]
                            while parent1!=current1:
                                current1 = parent1
                                parent1=parents[current1]

                            parent2 = parents[out[i][j-1]]
                            current2 = out[i][j-1]
                            while parent2!=current2:
                                current2 = parent2
                                parent2=parents[current2]

                            parents[out[i-1][j]] = min(parent1,parent2)
                            parents[out[i][j-1]] = min(parent1,parent2)

In [6]: def second_pass(out):

            for i in range(shp[0]):
                for j in range(shp[1]):
                    if out[i][j]==0:
                        continue
                    else:
                        if parents[out[i][j]]==out[i][j]:
                            continue
                        else:
                            parent = parents[out[i][j]]
                            current = out[i][j]
                            while parent!=current:
                                current = parent
                                parent=parents[current]
                            parents[out[i][j]] = parent
                            out[i][j] = parent
```

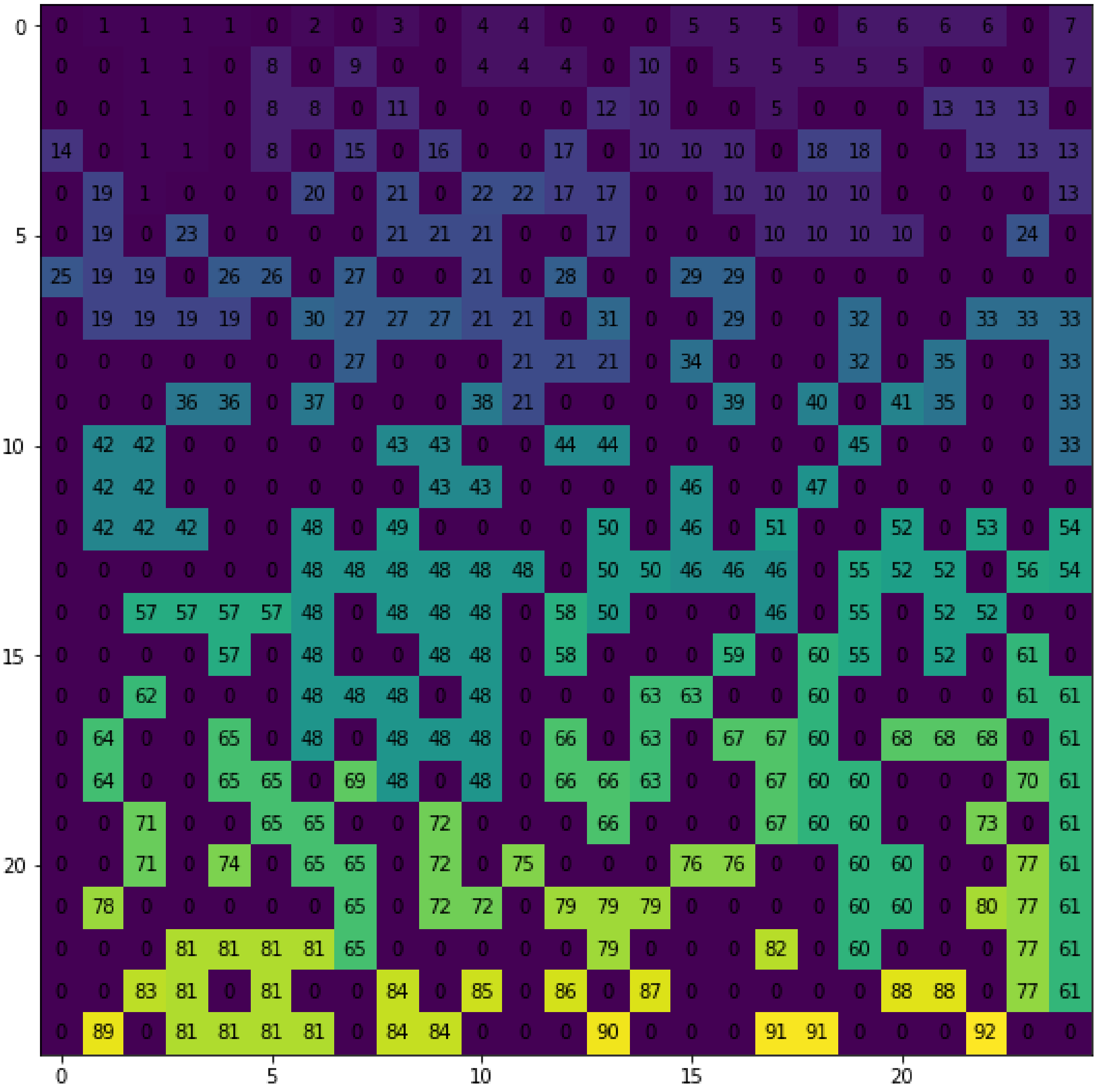
```
In [7]: def replace(out):

            for i in range(shp[0]):
                for j in range(shp[1]):
                    if out[i][j]==0:
                        continue
                    out[i][j] = final_components[out[i][j]]
```

```
In [8]: def final_list_gen(parents):
            for key,value in parents.items():
                if key==value:
                    final_list.append(key)
                else:
                    continue
```

```
In [9]: def final_components_gen(final_list):
            for i in range(1,len(final_list)+1):
                final_components[final_list[i-1]] = i
```

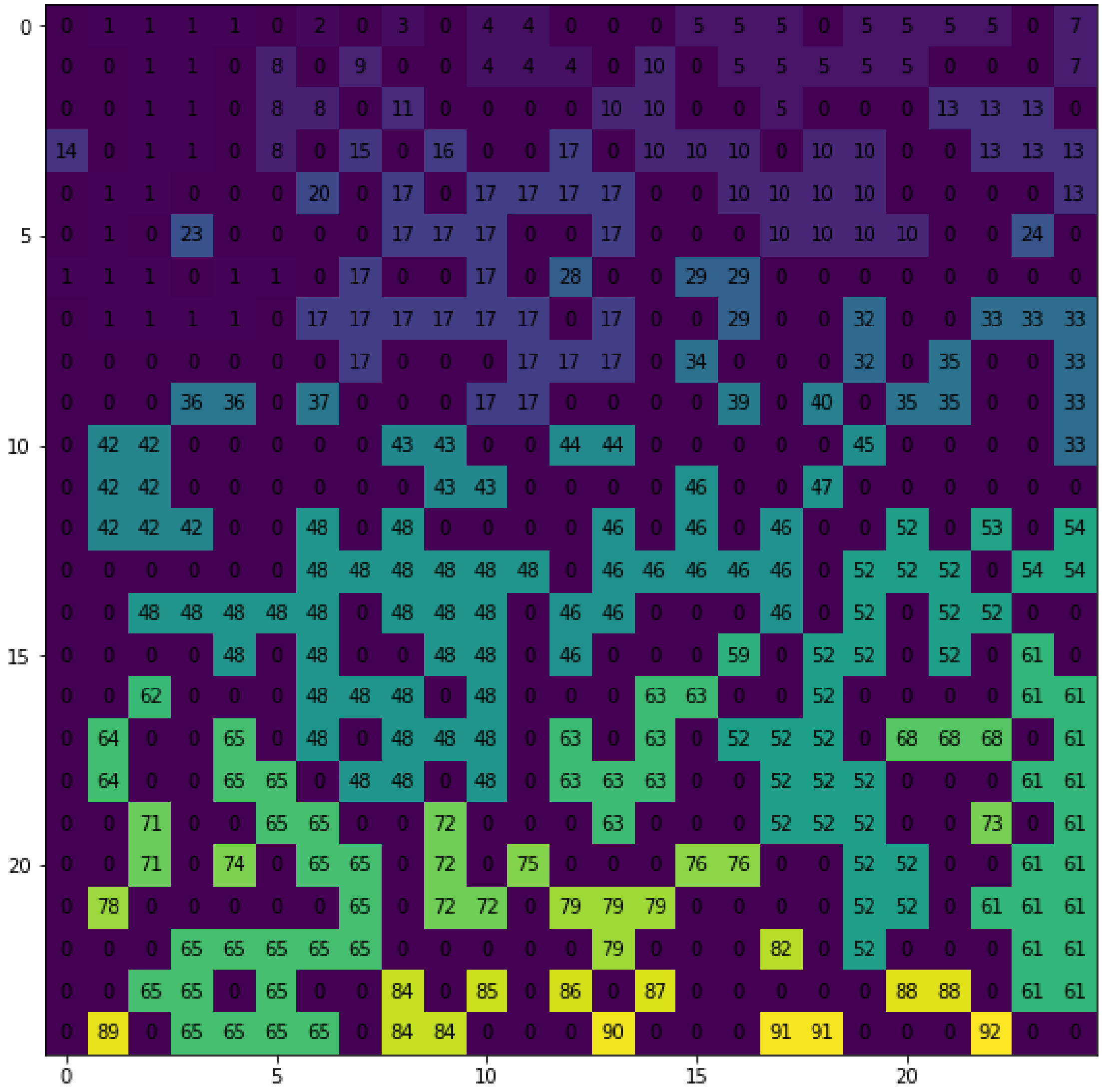
```
In [10]: parents = {
                 1:1
             }
         out = np.zeros(shp,dtype=int)
```

```
In [11]: first_pass(img)
         plot(out)
```



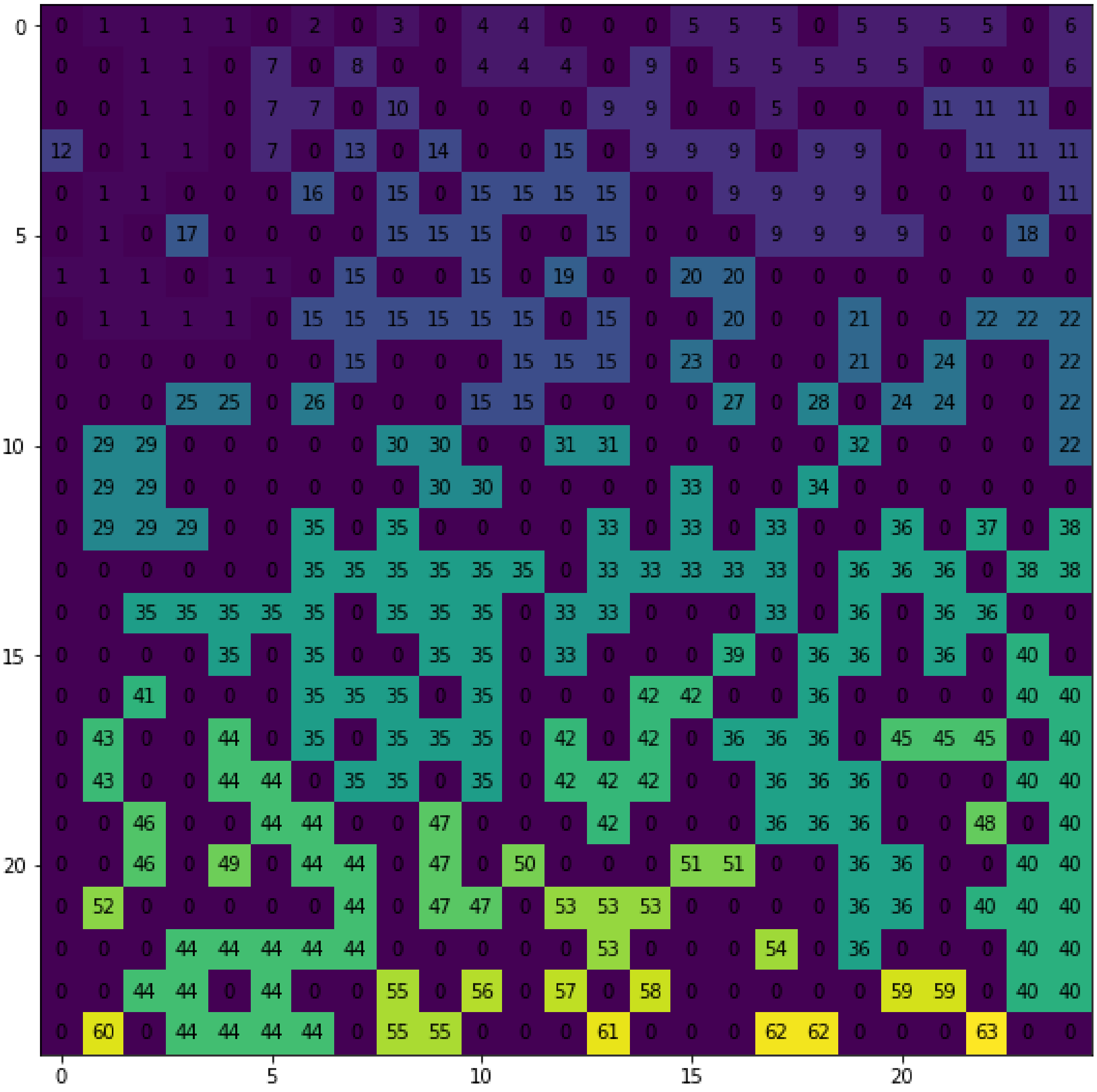
0	1	1	1	1	0	2	0	3	0	4	4	0	0	0	5	5	5	0	6	6	6	6	0	7
0	0	1	1	0	8	0	9	0	0	4	4	4	0	10	0	5	5	5	5	5	0	0	0	7
0	0	1	1	0	8	8	0	11	0	0	0	0	12	10	0	0	5	0	0	0	13	13	13	0
14	0	1	1	0	8	0	15	0	16	0	0	17	0	10	10	10	0	18	18	0	0	13	13	13
0	19	1	0	0	0	20	0	21	0	22	22	17	17	0	0	10	10	10	10	0	0	0	0	13
0	19	0	23	0	0	0	0	21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

```
In [12]: second_pass(out)
         plot(out)
```



```
 0  1  1  1  1  0  2  0  3  0  4  4  0  0  0  5  5  5  0  5  5  5  5  0  7
 0  0  1  1  0  8  0  9  0  0  4  4  4  0 10  0  5  5  5  5  5  0  0  0  7
 0  0  1  1  0  8  8  0 11  0  0  0  0 10 10  0  0  5  0  0  0 13 13 13  0
14  0  1  1  0  8  0 15  0 16  0  0 17  0 10 10 10  0 10 10  0  0 13 13 13
 0  1  1  0  0  0 20  0 17  0 17 17 17 17  0  0 10 10 10 10  0  0  0  0 13
 0  1  0 23  0  0  0  0 17 17 17  0  0 17  0  0  0 10 10 10 10  0  0 24  0
 1  1  1  0  1  1  0 17  0  0 17  0 28  0  0 29 29  0  0  0  0  0  0  0  0
 0  1  1  1  1  0 17 17 17 17 17 17  0 17  0  0 29  0  0 32  0  0 33 33 33
 0  0  0  0  0  0  0 17  0  0  0 17 17 17  0 34  0  0  0 32  0 35  0  0 33
 0  0  0 36 36  0 37  0  0  0 17 17  0  0  0  0 39  0 40  0 35 35  0  0 33
 0 42 42  0  0  0  0  0 43 43  0  0 44 44  0  0  0  0  0 45  0  0  0  0 33
 0 42 42  0  0  0  0  0  0 43 43  0  0  0  0 46  0  0 47  0  0  0  0  0  0
 0 42 42 42  0  0 48  0 48  0  0  0  0 46  0 46  0 46  0  0 52  0 53  0 54
 0  0  0  0  0  0 48 48 48 48 48 48  0 46 46 46 46 46  0 52 52 52  0 54 54
 0  0 48 48 48 48 48  0 48 48 48  0 46 46  0  0  0 46  0 52  0 52 52  0  0
 0  0  0  0 48  0 48  0  0 48 48  0 46  0  0  0 59  0 52 52  0 52  0 61  0
 0  0 62  0  0  0 48 48 48  0 48  0  0  0 63 63  0  0 52  0  0  0  0 61 61
 0 64  0  0 65  0 48  0 48 48 48  0 63  0 63  0 52 52 52  0 68 68 68  0 61
 0 64  0  0 65 65  0 48 48  0 48  0 63 63 63  0  0 52 52 52  0  0  0 61 61
 0  0 71  0  0 65 65  0  0 72  0  0  0 63  0  0  0 52 52 52  0  0 73  0 61
 0  0 71  0 74  0 65 65  0 72  0 75  0  0  0 76 76  0  0 52 52  0  0 61 61
 0 78  0  0  0  0  0 65  0 72 72  0 79 79 79  0  0  0  0 52 52  0 61 61 61
 0  0  0 65 65 65 65 65  0  0  0  0  0 79  0  0  0 82  0 52  0  0  0 61 61
 0  0 65 65  0 65  0  0 84  0 85  0 86  0 87  0  0  0  0  0 88 88  0 61 61
 0 89  0 65 65 65 65  0 84 84  0  0  0 90  0  0  0 91 91  0  0  0 92  0  0
```

```
In [13]: final_list = []
         final_list_gen(parents)
         final_components = {}
         final_components_gen(final_list)
         replace(out)
         plot(out)
```



	0					5					10					15					20				
0	0	1	1	1	1	0	2	0	3	0	4	4	0	0	0	5	5	5	0	5	5	5	5	0	6
	0	0	1	1	0	7	0	8	0	0	4	4	4	0	9	0	5	5	5	5	5	0	0	0	6
	0	0	1	1	0	7	7	0	10	0	0	0	0	9	9	0	0	5	0	0	0	11	11	11	0
	12	0	1	1	0	7	0	13	0	14	0	0	15	0	9	9	9	0	9	9	0	0	11	11	11
	0	1	1	0	0	0	16	0	15	0	15	15	15	15	0	0	9	9	9	9	0	0	0	0	11
5	0	1	0	17	0	0	0	0	15	15	15	0	0	15	0	0	0	9	9	9	9	0	0	18	0
	1	1	1	0	1	1	0	15	0	0	15	0	19	0	0	20	20	0	0	0	0	0	0	0	0
	0	1	1	1	1	0	15	15	15	15	15	15	0	15	0	0	20	0	0	21	0	0	22	22	22
	0	0	0	0	0	0	0	15	0	0	0	15	15	15	0	23	0	0	0	21	0	24	0	0	22
	0	0	0	25	25	0	26	0	0	0	15	15	0	0	0	0	27	0	28	0	24	24	0	0	22
10	0	29	29	0	0	0	0	0	30	30	0	0	31	31	0	0	0	0	0	32	0	0	0	0	22
	0	29	29	0	0	0	0	0	0	30	30	0	0	0	0	33	0	0	34	0	0	0	0	0	0
	0	29	29	29	0	0	35	0	35	0	0	0	0	33	0	33	0	33	0	0	36	0	37	0	38
	0	0	0	0	0	0	35	35	35	35	35	35	0	33	33	33	33	33	0	36	36	36	0	38	38
	0	0	35	35	35	35	35	0	35	35	35	0	33	33	0	0	0	33	0	36	0	36	36	0	0
15	0	0	0	0	35	0	35	0	0	35	35	0	33	0	0	0	39	0	36	36	0	36	0	40	0
	0	0	41	0	0	0	35	35	35	0	35	0	0	0	42	42	0	0	36	0	0	0	0	40	40
	0	43	0	0	44	0	35	0	35	35	35	0	42	0	42	0	36	36	36	0	45	45	45	0	40
	0	43	0	0	44	44	0	35	35	0	35	0	42	42	42	0	0	36	36	36	0	0	0	40	40
	0	0	46	0	0	44	44	0	0	47	0	0	0	42	0	0	0	36	36	36	0	0	48	0	40
20	0	0	46	0	49	0	44	44	0	47	0	50	0	0	0	51	51	0	0	36	36	0	0	40	40
	0	52	0	0	0	0	0	44	0	47	47	0	53	53	53	0	0	0	0	36	36	0	40	40	40
	0	0	0	44	44	44	44	44	0	0	0	0	0	53	0	0	0	54	0	36	0	0	0	40	40
	0	0	44	44	0	44	0	0	55	0	56	0	57	0	58	0	0	0	0	0	59	59	0	40	40
	0	60	0	44	44	44	44	0	55	55	0	0	0	61	0	0	0	62	62	0	0	0	63	0	0

```
In [14]: parents
```

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

```
In [15]: final_components
```

```
Out[15]: {1: 1,
           2: 2,
           3: 3,
           4: 4,
           5: 5,
           7: 6,
           8: 7,
           9: 8,
           10: 9,
           11: 10,
           13: 11,
           14: 12,
15: 13,
16: 14,
17: 15,
20: 16,
23: 17,
24: 18,
28: 19,
29: 20,
32: 21,
33: 22,
34: 23,
35: 24,
36: 25,
37: 26,
39: 27,
40: 28,
42: 29,
43: 30,
44: 31,
45: 32,
46: 33,
47: 34,
48: 35,
52: 36,
53: 37,
54: 38,
59: 39,
61: 40,
62: 41,
63: 42,
64: 43,
65: 44,
68: 45,
71: 46,
72: 47,
73: 48,
74: 49,
75: 50,
76: 51,
78: 52,
79: 53,
82: 54,
84: 55,
85: 56,
86: 57,
87: 58,
88: 59,
89: 60,
          90: 61,
          91: 62,
          92: 63}
```

```
In [16]: print("Number of components:", len(final_components.keys()))
```

```
Number of components: 63
```

```
In [17]: size = {}

         def find_size(out):
             for i in range(shp[0]):
                 for j in range(shp[1]):
                     if out[i][j]==0:
                         continue
                     value = out[i][j]
                     if value not in size.keys():
                         size[value] = 1
                     else:
                         size[value]+=1
```

```
In [18]: find_size(out)
         size
```

```
Out[18]: {1: 22,
          2: 1,
          3: 1,
          4: 5,
          5: 13,
          6: 2,
          7: 4,
          8: 1,
          9: 16,
          10: 1,
          11: 7,
          12: 1,
          13: 1,
          14: 1,
          15: 25,
          16: 1,
          17: 1,
          18: 1,
          19: 1,
          20: 3,
          21: 2,
          22: 6,
          23: 1,
          24: 3,
 25: 2,
 26: 1,
 27: 1,
 28: 1,
 29: 7,
 30: 4,
 31: 2,
 32: 1,
 33: 13,
 34: 1,
 35: 31,
 36: 25,
 37: 1,
 38: 3,
 39: 1,
 40: 16,
 41: 1,
 42: 8,
 43: 2,
 44: 20,
 45: 3,
 46: 2,
 47: 4,
 48: 1,
 49: 1,
 50: 1,
 51: 2,
 52: 1,
 53: 4,
 54: 1,
 55: 3,
 56: 1,
 57: 1,
 58: 1,
 59: 2,
 60: 1,
 61: 1,
 62: 2,
 63: 1}
```

```
In [ ]:
```

Zoom

```
In [1]: import cv2
        import numpy as np

In [2]: img=cv2.imread('landscape.jpg',1)
        out=np.zeros((img.shape[0]*2,img.shape[1]*2,img.shape[2]))
        out2=np.zeros((img.shape[0]*2,img.shape[1]*2,img.shape[2]))

In [4]: shp=img.shape
        shp

Out[4]: (290, 590, 3)

In [8]: img[4]

Out[8]: 200

In [9]: for i in range(shp[0]):
            out[2*i,:shp[1]]=np.copy(img[i,:])

In [ ]:

In [ ]: for i in range(shp[0]):
                out[2*i,:shp[1]]=np.copy(img[i,:])

        for j in range(shp[1]):
                out2[:,2*j]=np.copy(out[:,j])

        out=np.copy(out2)

        for i in range(img.shape[0]-1):
                out[2*i+1,:]=out[2*i,:]/2+out[2*i+2,:]/2

        for j in range(img.shape[1]-1):
                out[:,2*j+1]=out[:,2*j]/2+out[:,2*j+2]/2

        out[-1,:]=out[-2,:]
        out[:,-1]=out[:,-2]

        out=np.array(out, dtype = np.uint8)
```

```
cv2.imshow('image', img)
cv2.imshow('zoomed', out)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Shrink

```
In [ ]: import cv2
        import numpy as np

        img=cv2.imread('cat.jpeg',0)
        out=np.copy(img)

        i=1

        while i<out.shape[0]:
                out=np.delete(out,(i),axis=0)
                i=i+1
        j=1

        while j<out.shape[1]:
                out=np.delete(out,(j),axis=1)
                j=j+1

        out=np.array(out, dtype = np.uint8)
        print(img.shape)
        print(out.shape)
        cv2.imshow('image', img)
        cv2.imshow('shrinked', out)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

In [ ]:
```

Contrast

```
In [ ]: import cv2
        import numpy as np

        img=cv2.imread('flower1.jpg',0)
        int_min=np.amin(img)
        int_max=np.amax(img)
        x1=int_min
        y1=0
        x2=int_max
        y2=255
        m=float(y2-y1)/float(x2-x1)
        out=img*m
        out=out-(x1*m)

        out=np.array(out, dtype = np.uint8)
        print(img)
        print(out)
        cv2.imshow('image', img)
        cv2.imshow('contrast expanded', out)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

```
[[126 126 126 ... 128 128 128]
 [126 126 126 ... 128 128 128]
 [126 126 127 ... 128 129 129]
 ...
 [135 135 134 ... 114 114 113]
 [133 134 133 ... 114 113 113]
 [134 136 134 ... 114 113 113]]
[[122 122 122 ... 124 124 124]
 [122 122 122 ... 124 124 124]
 [122 122 123 ... 124 125 125]
 ...
 [131 131 130 ... 110 110 108]
 [129 130 129 ... 110 108 108]
 [130 132 130 ... 110 108 108]]
```

```
In [ ]:
```

Invert

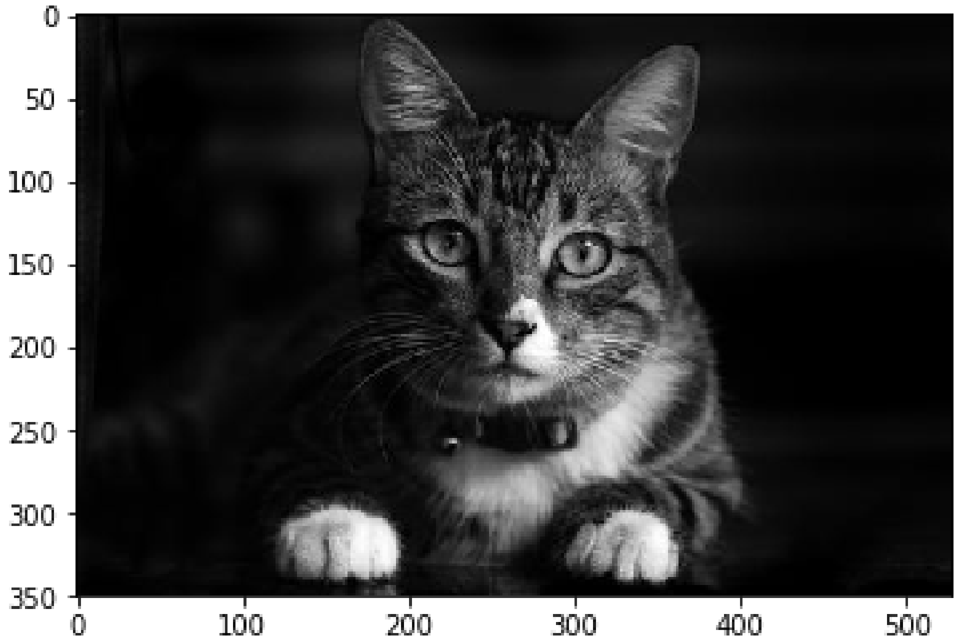
```
In [2]: import cv2
        import numpy as np

        def invertor(value):
                return 255-value

        img=cv2.imread('cat.jpeg',0)
        out=np.zeros(img.shape)
        for i in range(img.shape[0]):
                for j in range(img.shape[1]):
                        out[i,j]=invertor(img[i,j])

        out=np.array(out, dtype = np.uint8)

        cv2.imshow('image', img)
        cv2.imshow('inverted', out)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

In [ ]:
```

Low pass filter

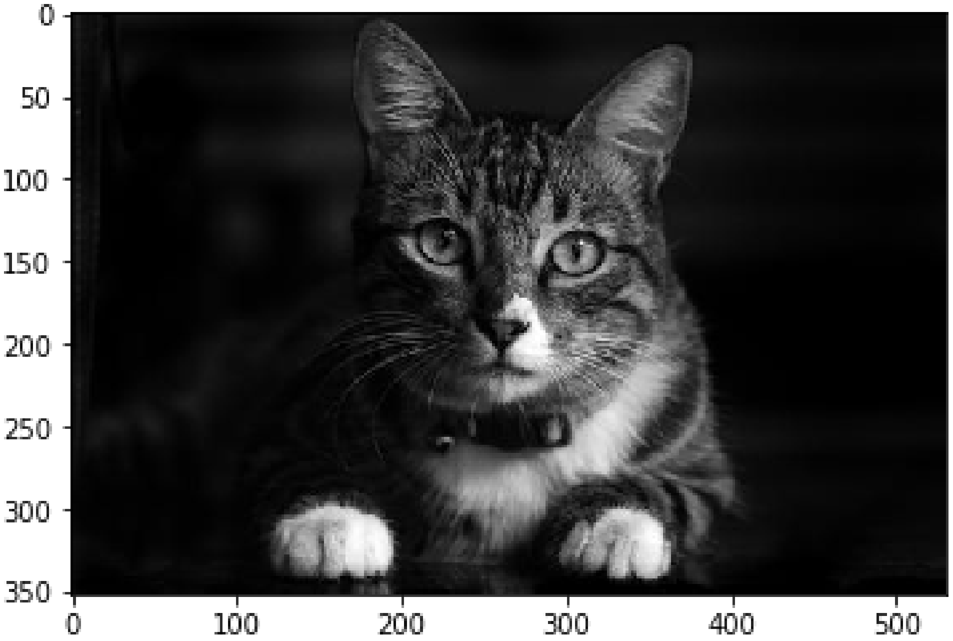
```
In [1]: import cv2
        import numpy as np
        import matplotlib.pyplot as plt
```

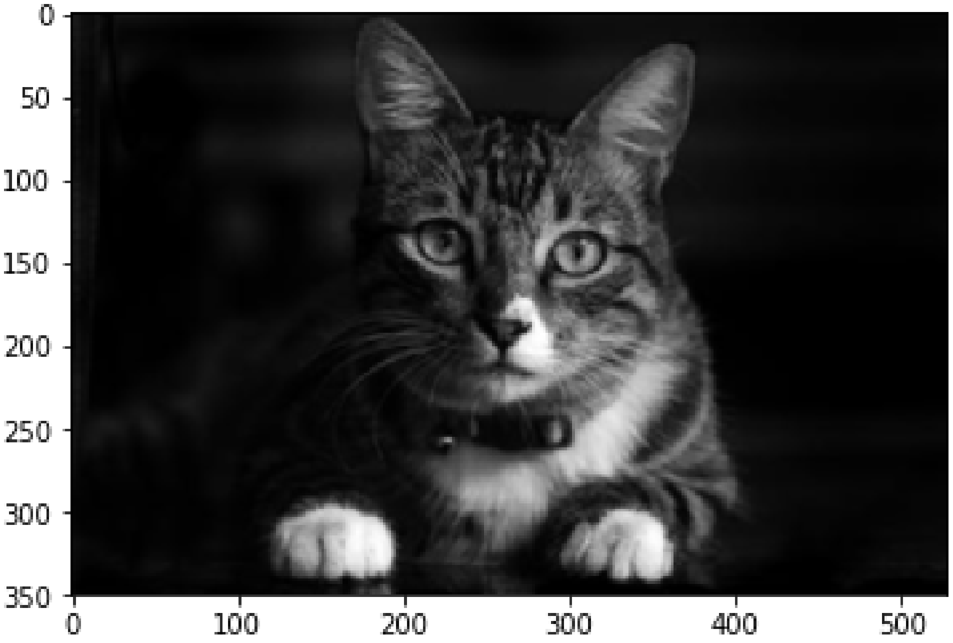
```
In [2]: def pad(img,shp):
            p=np.zeros((shp[0]+2,shp[1]+2))
            p[1:-1,1:-1]=np.copy(img)
            p[0,1:-1],p[-1,1:-1]=img[0],img[-1]
            p[1:-1,0],p[1:-1,-1]=img[:,0],img[:,-1]
            p[0,0],p[0,-1]=img[0,0],img[0,-1]
            p[-1,0],p[-1,-1]=img[-1,0],img[-1,-1]
            return p
```

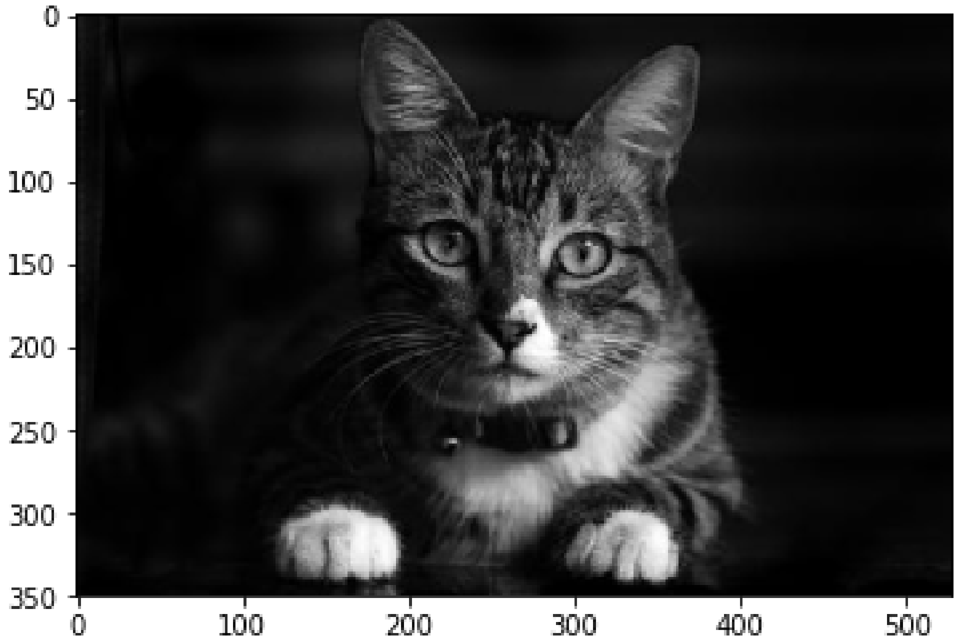
```
In [22]: img=cv2.imread('cat.jpeg',0)
         shp=img.shape
         shpm=(3,3)
         mask=np.full(shpm,1/9)
         mask2=np.array([[0,1/8,0],[1/8,1/2,1/8],[0,1/8,0]])
         p=pad(img,shp)
         out=np.zeros((shp))
         out2 = np.zeros((shp))
         plt.imshow(img,cmap='gray')
         plt.show()
```



```
In [26]: for i in range(shp[0]):
             for j in range(shp[1]):
                 out[i,j]=np.floor(np.multiply(p[i:i+shpm[0],j:j+shpm[1]],mask).sum())
                 out2[i,j]=np.floor(np.multiply(p[i:i+shpm[0],j:j+shpm[1]],mask2).sum())
         out=out.astype(int)
         out=np.array(out, dtype = np.uint8)
         out2=out2.astype(int)
         out2=np.array(out2,dtype = np.uint8)
         plt.imshow(p,cmap='gray')
         plt.show()
         plt.imshow(out,'gray')
         plt.show()
         plt.imshow(out2,'gray')
         plt.show()
```


0
50
100
150
200
250
300
350
0
100
200
300
400
500


0
50
100
150
200
250
300
350
0
100
200
300
400
500

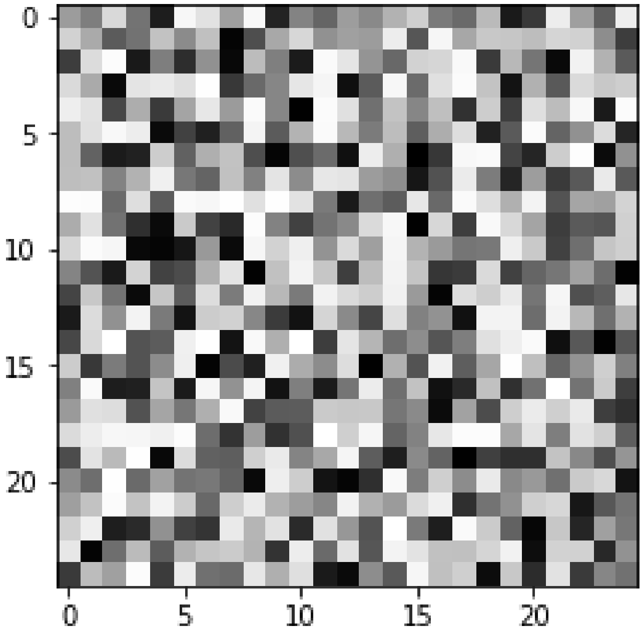


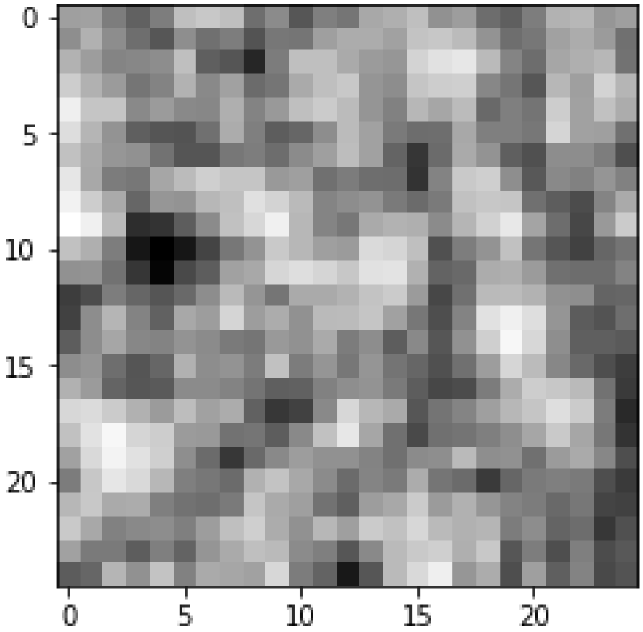
```
In [ ]:
```

Masking

```
In [2]: import cv2
        import numpy as np
        import matplotlib.pyplot as plt

        def pad(img,shp):
                p=np.zeros((shp[0]+2,shp[1]+2))
                p[1:-1,1:-1]=np.copy(img)
                p[0,1:-1],p[-1,1:-1]=img[0],img[-1]
                p[1:-1,0],p[1:-1,-1]=img[:,0],img[:,-1]
                p[0,0],p[0,-1]=img[0,0],img[0,-1]
                p[-1,0],p[-1,-1]=img[-1,0],img[-1,-1]
                return p


        shp=(25,25)
        img = np.floor(np.random.random(shp)*255)


        shpm=(3,3)
        mask=np.full(shpm,1)
        p=pad(img,shp)
        out=np.zeros((shp))

        for i in range(shp[0]):
                for j in range(shp[1]):
                        temp=np.multiply(p[i:i+shpm[0],j:j+shpm[1]],mask)
                        temp2=temp.sum()
                        out[i,j]=temp2

        out=out/9
        out=out.astype(int)
        fig = plt.figure(100)
        fig.canvas.set_window_title('Original image')
        plt.imshow(img, cmap="Greys")

        fig = plt.figure(200)
        fig.canvas.set_window_title('Masked')
```

```
plt.imshow(out, cmap="Greys")

plt.show()
```





```
In [3]:

Out[3]: '3.4.4'

In [ ]:
```

Gaussian Filter

```
In [ ]: import numpy as np
        import cv2
        from matplotlib import pyplot as plt
```

```
In [ ]: def pad(img,shp,l):
                p=np.zeros((shp[0]+2*l,shp[1]+2*l))
                p[l:-l,l:-l]=np.copy(img)
                for j in range(l):
                        p[l:-l,j]=p[l:-l,l]
                        p[l:-l,-j-1]=p[l:-l,-l-1]

                for i in range(l):
                        p[i]=p[l]
                        p[-i-1]=p[-l-1]

                return p
```

```
In [ ]: def gauss(Z,var):
                N=int(Z/2)
                fil=np.zeros((Z,Z))
                for x in range(1,N+1):   #for non zero
                        for y in range(x,N+1):
                                ex=np.exp(-float(x*x+y*y)/(2*var))
                                print(ex)
                                fil[N-x,N-y]=ex
                                fil[N+x,N-y]=ex
                                fil[N-x,N+y]=ex
                                fil[N+x,N+y]=ex
                                if x!=y:
                                        fil[N-y,N-x]=ex
                                        fil[N+y,N-x]=ex
                                        fil[N-y,N+x]=ex
                                        fil[N+y,N+x]=ex
                for x in range(1,N+1):  #for zero elements
                        ex=np.exp(-float(x*x)/(2*var))
                        fil[N-x,N]=ex
                        fil[N+x,N]=ex
                        fil[N,N-x]=ex
                fil[N,N+x]=ex
        fil[N,N]=1
        print(fil)
        c=float(1)/float(fil[Z-1,Z-1])
        print(c)
        fil=np.round(fil*c).astype(int)
        return fil
```

In [ ]:

```
N=int(input("Enter size of Gaussian Filter (odd number only): "))
var=int(input("Enter the variance of Gaussian Filter: "))
fil=gauss(N,var)
print(fil)
coeff=np.sum(fil)
print(coeff)
img=cv2.imread('cat.jpeg',0)
shp=img.shape
shpm=(N,N)
mask=fil
p=pad(img,shp,shpm[1])

out=np.zeros((shp))

for i in range(shp[0]):
        for j in range(shp[1]):
                temp=np.multiply(p[i:i+shpm[0],j:j+shpm[1]],mask)
                temp2=temp.sum()
                out[i,j]=np.floor(temp2)
out=out/coeff

out=np.array(out, dtype = np.uint8)

cv2.imshow('image', img)
cv2.imshow('masked', out)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

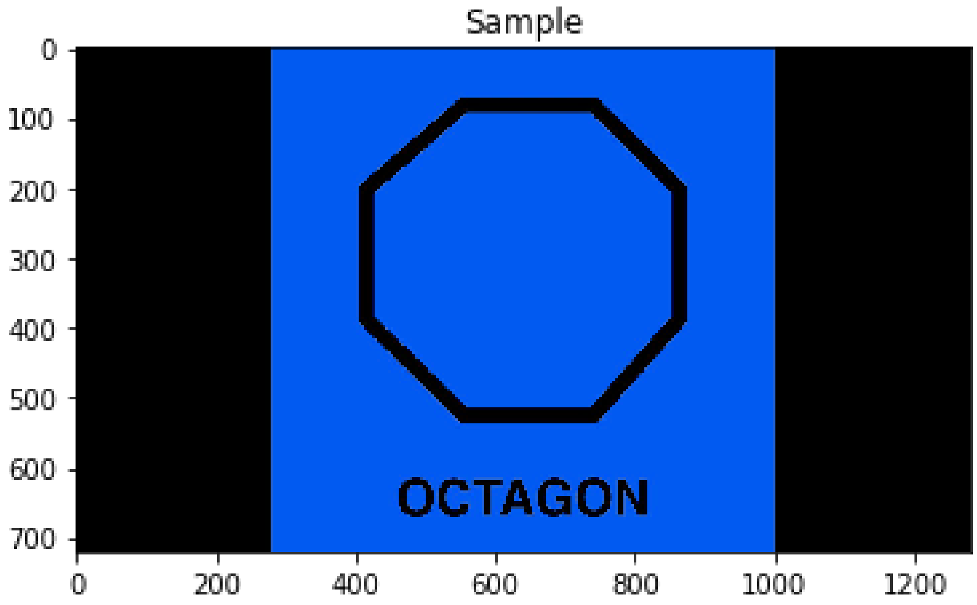
In [ ]:

Edge Detection

0.0.1 Edge Detection

```
In [1]: import cv2
        import numpy as np
        from matplotlib import pyplot as plt
```

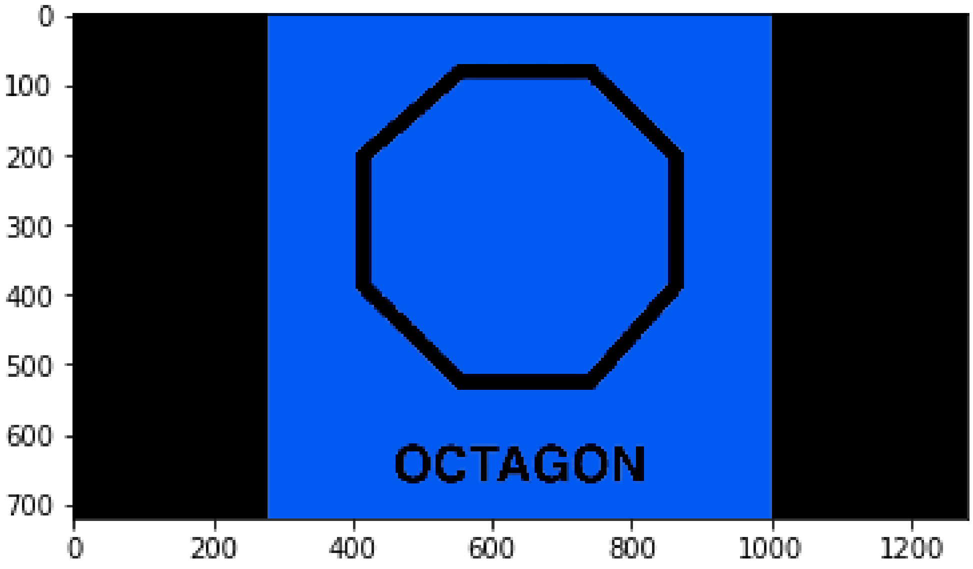
```
In [2]: img = cv2.imread('sample5.jpg')
        plt.imshow(img)
        plt.title('Sample')
        plt.show()
```



```
In [3]: blur = cv2.fastNlMeansDenoisingColored(img,None,10,10,7,21)
```

```
In [4]: plt.imshow(blur,cmap='gray')
```

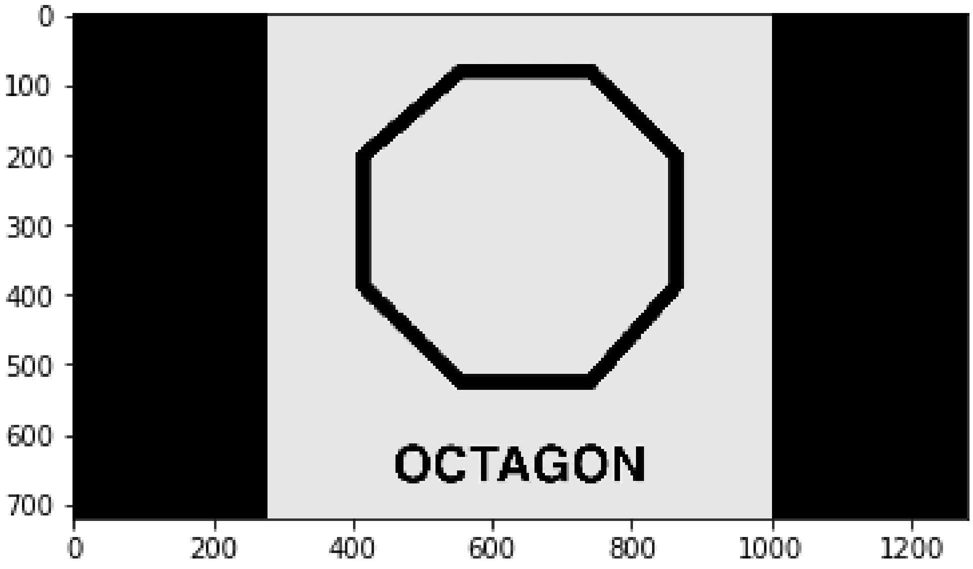
```
Out[4]: <matplotlib.image.AxesImage at 0x11a1daf28>
```



```
In [5]: gray = cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)
```

```
In [6]: plt.imshow(gray,cmap='gray')
```

```
Out[6]: <matplotlib.image.AxesImage at 0x11bf20e80>
```



```
In [7]: print(gray.shape)
```

```
(720, 1280)
```

```
In [8]: def pad(img,shp):
            p=np.zeros((shp[0]+2,shp[1]+2))
            p[1:-1,1:-1]=np.copy(img)
            p[0,1:-1],p[-1,1:-1]=img[0],img[-1]
            p[1:-1,0],p[1:-1,-1]=img[:,0],img[:,-1]
            p[0,0],p[0,-1]=img[0,0],img[0,-1]
            p[-1,0],p[-1,-1]=img[-1,0],img[-1,-1]
            return p
```

```
In [9]: def sobel_filter(img):
            sabel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
            sabel_y = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
            shp = img.shape
            shpm=(3,3)
            padded_img=pad(img,shp)
            grad_matrix=np.zeros(shp)
            out=np.zeros(shp)
            exp = np.zeros(shp)
            for i in range(shp[0]):
                for j in range(shp[1]):
                    g_x=np.multiply(padded_img[i:i+shpm[0],j:j+shpm[1]],sabel_x).sum()
                    g_y=np.multiply(padded_img[i:i+shpm[0],j:j+shpm[1]],sabel_y).sum()
                    if g_y!=0 or g_x!=0:
                        if g_x==0:
                            rad=np.arctan2(g_y,g_x)
                        else:
                            rad=np.arctan2(g_y,g_x)
                        deg=rad*(180/np.pi)
                        rad_rev = deg*(np.pi/180)
                        #print(rad*(180/np.pi),end=" ")
                        #print(rad_rev)
                        grad_matrix[i][j]=deg
                        if grad_matrix[i][j]<0:
                            exp[i][j] = grad_matrix[i][j]
                        out[i][j] = np.sqrt(np.square(g_x)+np.square(g_y))
                    else:
                        out[i,j]=255
                        grad_matrix[i][j]=255
            out=np.array(out, dtype = np.uint8)
            grad_matrix=np.array(grad_matrix,dtype= np.uint8)

            return out,grad_matrix,exp
```

```
In [10]: def prewitt_filter(img):
             prewitt_x = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
```

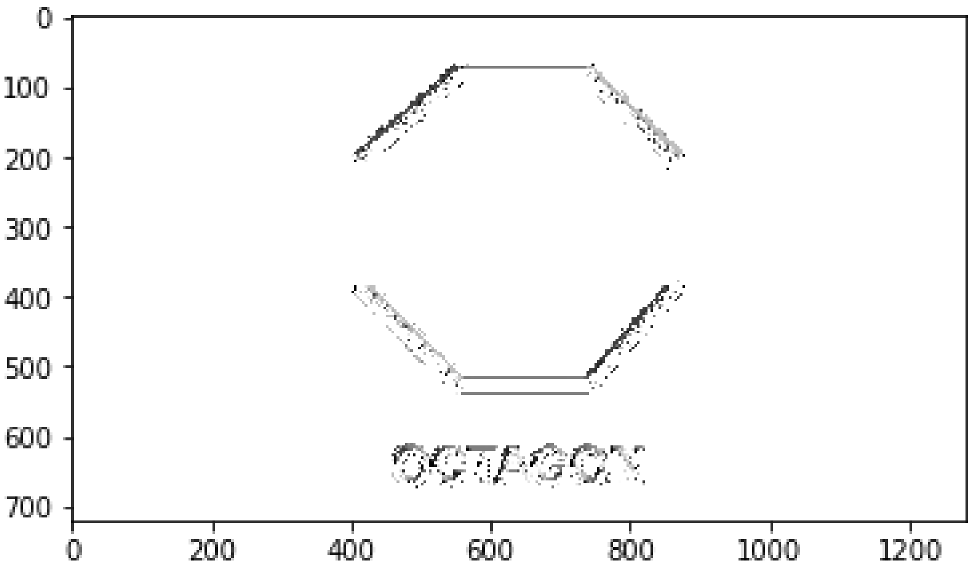
```
    prewitt_y = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
    shp = img.shape
    shpm=(3,3)
    padded_img=pad(img,shp)
    grad_matrix=np.zeros(shp)
    out=np.zeros(shp)
    exp = np.zeros(shp)
    for i in range(shp[0]):
        for j in range(shp[1]):
            g_x=np.multiply(padded_img[i:i+shpm[0],j:j+shpm[1]],prewitt_x).sum()
            g_y=np.multiply(padded_img[i:i+shpm[0],j:j+shpm[1]],prewitt_y).sum()
            if g_y!=0 or g_x!=0:
                if g_x==0:
                    rad=np.arctan2(g_y,g_x)
                else:
                    rad=np.arctan2(g_y,g_x)
                deg=rad*(180/np.pi)
                rad_rev = deg*(np.pi/180)
                #print(rad*(180/np.pi),end=" ")
                #print(rad_rev)
                grad_matrix[i][j]=deg
                if grad_matrix[i][j]<0:
                    exp[i][j] = grad_matrix[i][j]
                    print(exp[i][j])
                out[i][j] = np.sqrt(np.square(g_x)+np.square(g_y))
            else:
                out[i,j]=255
                grad_matrix[i][j]=255
    out=np.array(out, dtype = np.uint8)

    return out,grad_matrix,exp
```

```
In [11]: output, grad_matrix,exp = sobel_filter(gray)
```

```
In [ ]:
```

```
In [12]: plt.imshow(exp,cmap='gray')
```

```
Out[12]: <matplotlib.image.AxesImage at 0x11c227dd8>
```



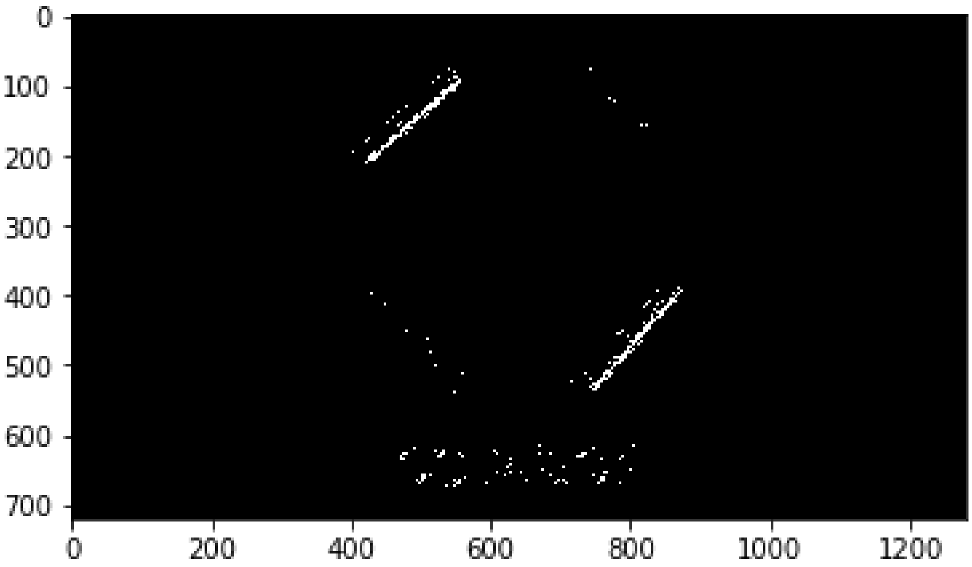
```
In [13]: output2, grad_matrix2,exp2 = prewitt_filter(gray)
```

```
-45.0
-63.43494882292201
-116.56505117707799
-135.0
-45.0
-63.43494882292201
-90.0
-116.56505117707799
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
-90.0
```

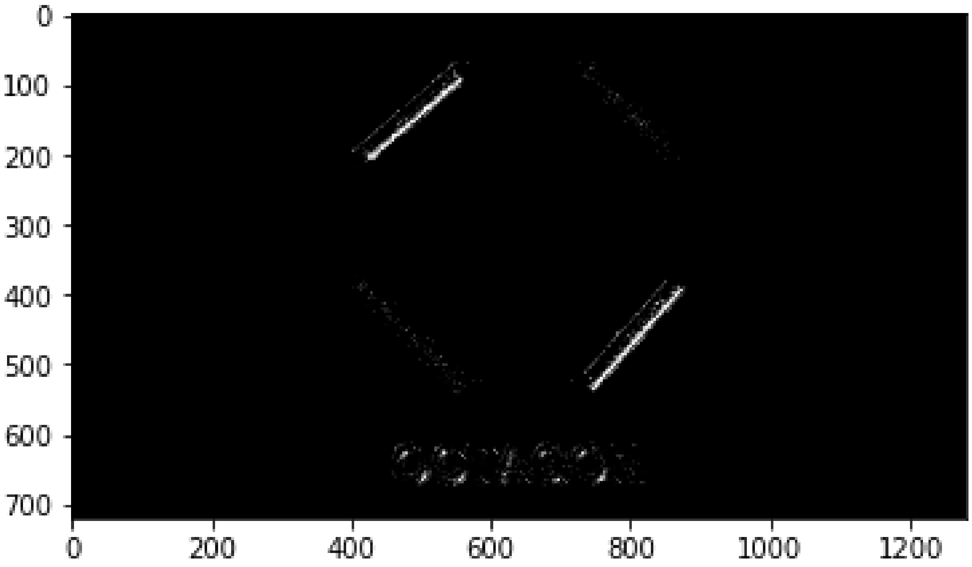
```
-48.03403964694501
-34.99202019855866
-47.48955292199916
-40.91438322002513
-57.10767025676035
-40.539151741483444
-45.0
-57.885169399703265
-38.736509385665464
-43.97696981133217
-41.18592516570965
-45.0
-49.899092453787766
-45.0
-45.0
-57.52880770915151
-45.0
-45.0
-39.472459848343824
-44.13194855025446
-45.0
-49.899092453787766
-47.48955292199916
-45.0
-58.10920819815429
-32.9052429229879
-45.0
-45.0
-45.0
-45.0
-50.19442890773481
-33.690067525979785
-45.0
-45.0
-45.0
-59.03624346792648
-45.0
-45.0
-53.13010235415598
-55.00797980144134
-47.12109639666146
-45.0
-30.96375653207352
-56.309932474020215
-45.0
-45.0
-56.309932474020215
-56.309932474020215
```

```
-33.690067525979785
-45.0
-36.86989764584402
-45.0
-45.0
-45.0
-45.0
-56.309932474020215
-45.0
-45.0
-45.0
-45.0
-59.03624346792648
-45.0
-45.0
-53.13010235415598
-45.0
```

```
In [25]: plt.imshow(filtered2,cmap="gray")
         cv2.imshow('image',filtered2)
         cv2.waitKey()
         cv2.destroyAllWindows()
```

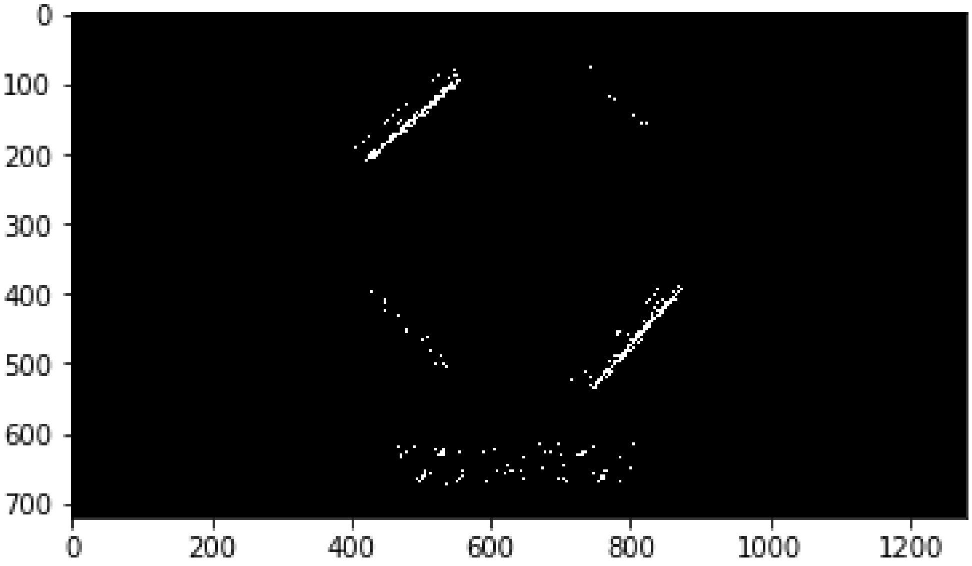


```
In [26]: smo_fil = cv2.GaussianBlur(filtered2,(5,5),0)

In [27]: plt.imshow(smo_fil,cmap='gray')
         plt.imsave('output_edge.png',smo_fil,cmap='gray')
```



```
In [ ]:
```

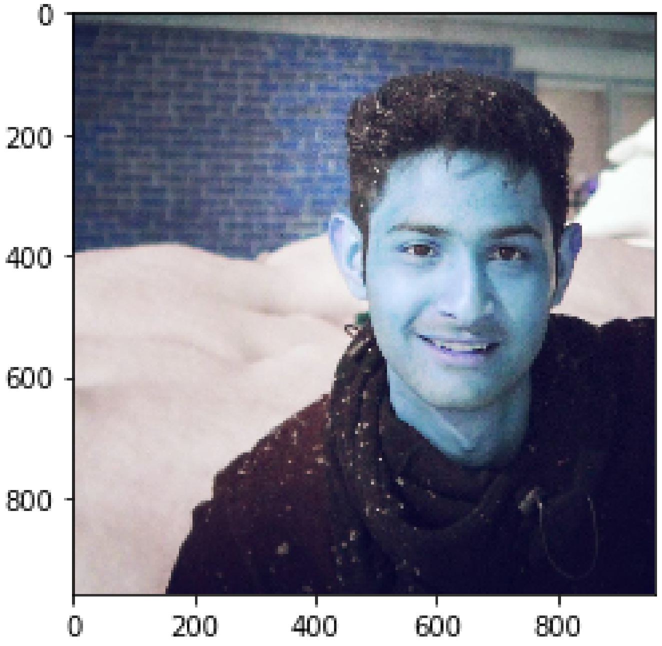
```
In [28]: plt.imshow(filtered,cmap="gray")
         cv2.imshow('image',filtered)
         cv2.waitKey()
         cv2.destroyAllWindows()
```



```
In [29]: x = np.array([-1, +1, +1, -1])
         y = np.array([-1, -1, +1, +1])

         np.arctan2(y,x)*180/np.pi

Out[29]: array([-135.,  -45.,   45.,  135.])

In [ ]:

In [ ]:
```

Histogram equalisation

```
In [2]: import cv2
        import numpy as np
        import matplotlib.pyplot as plt

In [3]: image = cv2.imread('sample7.jpg')

In [4]: plt.imshow(image)

Out[4]: <matplotlib.image.AxesImage at 0x123bbe5f8>
```

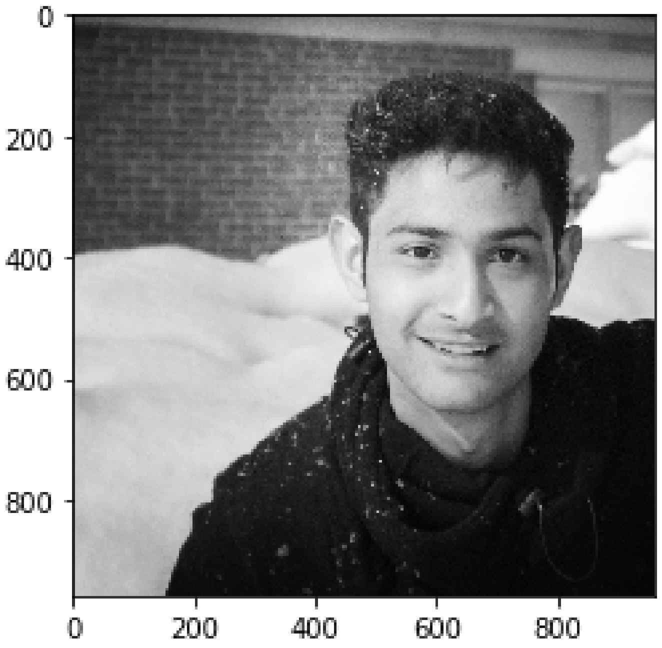


```
In [5]: image.shape

Out[5]: (959, 960, 3)

In [6]: gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
In [7]: gray.shape
```

```
Out[7]: (959, 960)
```

```
In [8]: plt.imshow(gray,cmap='gray')
```

```
Out[8]: <matplotlib.image.AxesImage at 0x12499f7f0>
```



```
In [9]: gray.shape
```

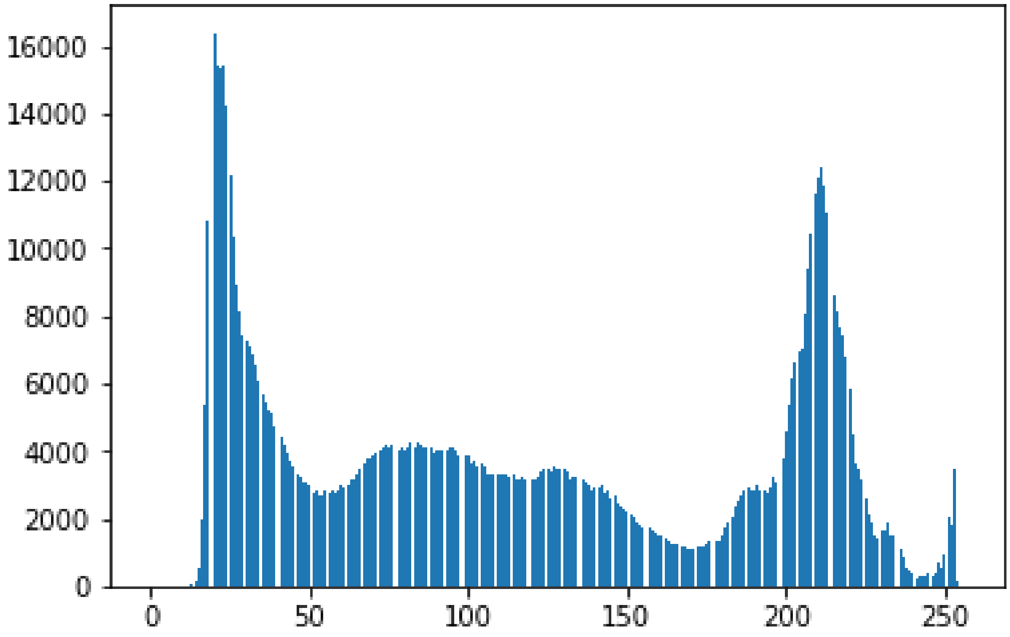
```
Out[9]: (959, 960)
```

```
In [10]: hist = np.zeros(256)

         for i in range(gray.shape[0]):
             for j in range(gray.shape[1]):
                 hist[gray[i][j]] = hist[gray[i][j]]+1
```

```
In [11]: hist
```

```
Out[11]: array([0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                0.0000e+00, 4.0000e+00, 1.4000e+01, 4.1000e+01, 1.3900e+02,
                5.3000e+02, 1.9360e+03, 5.4010e+03, 1.0821e+04, 1.4391e+04,
                1.6384e+04, 1.5396e+04, 1.5370e+04, 1.5431e+04, 1.4197e+04,
                1.2152e+04, 1.0335e+04, 8.9400e+03, 8.0950e+03, 7.4570e+03,
                7.2320e+03, 7.0980e+03, 6.8880e+03, 6.5340e+03, 6.1070e+03,
       5.6790e+03, 5.4360e+03, 5.1730e+03, 5.0900e+03, 4.7480e+03,
       4.4260e+03, 4.3900e+03, 4.1620e+03, 3.9660e+03, 3.6680e+03,
       3.5500e+03, 3.2820e+03, 3.2280e+03, 3.0660e+03, 3.0590e+03,
       2.9810e+03, 2.7740e+03, 2.8010e+03, 2.7070e+03, 2.6940e+03,
       2.8070e+03, 2.7780e+03, 2.7930e+03, 2.7750e+03, 2.8080e+03,
       2.9620e+03, 2.9490e+03, 2.9780e+03, 3.1340e+03, 3.1530e+03,
       3.3280e+03, 3.4790e+03, 3.6340e+03, 3.8070e+03, 3.8160e+03,
       3.8480e+03, 3.9350e+03, 3.9990e+03, 4.0850e+03, 4.1750e+03,
       4.0950e+03, 4.1620e+03, 4.0940e+03, 3.9930e+03, 4.0690e+03,
       3.9880e+03, 4.1040e+03, 4.2280e+03, 4.1020e+03, 4.2210e+03,
       4.1830e+03, 4.0910e+03, 4.1350e+03, 4.0990e+03, 3.9790e+03,
       4.0110e+03, 4.0540e+03, 4.0350e+03, 4.0300e+03, 4.1130e+03,
       4.1030e+03, 3.9810e+03, 3.8950e+03, 3.8410e+03, 3.8920e+03,
       3.8270e+03, 3.6620e+03, 3.7050e+03, 3.5640e+03, 3.6170e+03,
       3.5370e+03, 3.3260e+03, 3.3210e+03, 3.3270e+03, 3.3310e+03,
       3.2930e+03, 3.3090e+03, 3.2780e+03, 3.2510e+03, 3.2690e+03,
       3.1190e+03, 3.1670e+03, 3.2130e+03, 3.1790e+03, 3.2070e+03,
       3.1680e+03, 3.1680e+03, 3.2270e+03, 3.3870e+03, 3.4450e+03,
       3.4750e+03, 3.3830e+03, 3.5060e+03, 3.4310e+03, 3.4800e+03,
       3.4670e+03, 3.4230e+03, 3.1820e+03, 3.2260e+03, 3.2200e+03,
       3.1090e+03, 3.1170e+03, 3.0980e+03, 2.9830e+03, 2.8660e+03,
       2.8860e+03, 2.9300e+03, 2.9830e+03, 2.7810e+03, 2.8090e+03,
       2.6030e+03, 2.6710e+03, 2.4640e+03, 2.3790e+03, 2.3130e+03,
       2.1720e+03, 2.1240e+03, 2.0690e+03, 1.8860e+03, 1.8370e+03,
       1.7160e+03, 1.7500e+03, 1.7540e+03, 1.6800e+03, 1.5720e+03,
       1.4550e+03, 1.4500e+03, 1.4180e+03, 1.3210e+03, 1.2840e+03,
       1.2180e+03, 1.2120e+03, 1.1840e+03, 1.1950e+03, 1.0960e+03,
       1.0740e+03, 1.0900e+03, 1.1620e+03, 1.1720e+03, 1.1790e+03,
       1.2720e+03, 1.2930e+03, 1.2640e+03, 1.3320e+03, 1.3540e+03,
       1.5240e+03, 1.6960e+03, 1.8470e+03, 2.0300e+03, 2.3390e+03,
       2.4810e+03, 2.7110e+03, 2.7930e+03, 2.8720e+03, 2.8440e+03,
       2.8310e+03, 3.0060e+03, 2.8070e+03, 2.8200e+03, 2.7230e+03,
       2.9290e+03, 3.1890e+03, 3.0830e+03, 3.3730e+03, 3.7540e+03,
       4.5340e+03, 5.3890e+03, 6.1430e+03, 6.5910e+03, 6.9450e+03,
       7.0540e+03, 8.0210e+03, 9.3930e+03, 1.0456e+04, 1.1643e+04,
       1.2068e+04, 1.2370e+04, 1.1875e+04, 1.1074e+04, 1.0362e+04,
       8.5860e+03, 8.1610e+03, 7.6920e+03, 7.4170e+03, 6.8190e+03,
       5.8080e+03, 4.5270e+03, 3.6060e+03, 3.4900e+03, 3.1600e+03,
       2.6170e+03, 2.1490e+03, 1.8970e+03, 1.4790e+03, 1.4330e+03,
       1.6260e+03, 1.6690e+03, 1.9050e+03, 1.4740e+03, 1.4610e+03,
       1.1510e+03, 1.0960e+03, 8.4300e+02, 5.2100e+02, 4.3000e+02,
       3.5700e+02, 2.4500e+02, 2.7500e+02, 2.6800e+02, 3.3800e+02,
       3.5000e+02, 3.4000e+02, 4.0700e+02, 7.1200e+02, 5.7700e+02,
       9.2300e+02, 2.0710e+03, 1.8130e+03, 3.4500e+03, 1.3800e+02,
       8.0000e+00])
```

```
In [12]: index = np.arange(len(hist))
         plot1 = plt.bar(index,hist)
```



```
In [13]: pdf = [np.around(i/(gray.shape[0]*gray.shape[1]),decimals=5) for i in hist];pdf
```

```
Out[13]: [0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          2e-05,
          4e-05,
          0.00015,
          0.00058,
          0.0021,
          0.00587,
          0.01175,
          0.01563,
          0.0178,
          0.01672,
          0.01669,
          0.01676,
0.01542,
0.0132,
0.01123,
0.00971,
0.00879,
0.0081,
0.00786,
0.00771,
0.00748,
0.0071,
0.00663,
0.00617,
0.0059,
0.00562,
0.00553,
0.00516,
0.00481,
0.00477,
0.00452,
0.00431,
0.00398,
0.00386,
0.00356,
0.00351,
0.00333,
0.00332,
0.00324,
0.00301,
0.00304,
0.00294,
0.00293,
0.00305,
0.00302,
0.00303,
0.00301,
0.00305,
0.00322,
0.0032,
0.00323,
0.0034,
0.00342,
0.00361,
0.00378,
0.00395,
0.00414,
0.00414,
0.00418,
0.00427,
0.00434,
0.00444,
0.00453,
0.00445,
0.00452,
0.00445,
0.00434,
0.00442,
0.00433,
0.00446,
0.00459,
0.00446,
0.00458,
0.00454,
0.00444,
0.00449,
0.00445,
0.00432,
0.00436,
0.0044,
0.00438,
0.00438,
0.00447,
0.00446,
0.00432,
0.00423,
0.00417,
0.00423,
0.00416,
0.00398,
0.00402,
0.00387,
0.00393,
0.00384,
0.00361,
0.00361,
0.00361,
0.00362,
0.00358,
0.00359,
0.00356,
0.00353,
0.00355,
0.00339,
0.00344,
0.00349,
0.00345,
0.00348,
0.00344,
0.00344,
0.00351,
0.00368,
0.00374,
0.00377,
0.00367,
0.00381,
0.00373,
0.00378,
0.00377,
0.00372,
0.00346,
0.0035,
0.0035,
0.00338,
0.00339,
0.00337,
0.00324,
0.00311,
0.00313,
0.00318,
0.00324,
0.00302,
0.00305,
0.00283,
0.0029,
0.00268,
0.00258,
0.00251,
0.00236,
0.00231,
0.00225,
0.00205,
0.002,
0.00186,
0.0019,
0.00191,
0.00182,
0.00171,
0.00158,
0.00157,
0.00154,
0.00143,
0.00139,
0.00132,
0.00132,
0.00129,
0.0013,
0.00119,
0.00117,
0.00118,
0.00126,
0.00127,
0.00128,
0.00138,
0.0014,
0.00137,
0.00145,
0.00147,
0.00166,
0.00184,
0.00201,
0.0022,
0.00254,
0.00269,
0.00294,
0.00303,
0.00312,
0.00309,
0.00308,
0.00327,
0.00305,
0.00306,
0.00296,
0.00318,
0.00346,
0.00335,
0.00366,
0.00408,
0.00492,
0.00585,
0.00667,
0.00716,
0.00754,
0.00766,
0.00871,
0.0102,
0.01136,
0.01265,
0.01311,
0.01344,
0.0129,
0.01203,
0.01126,
0.00933,
 0.00886,
 0.00836,
 0.00806,
 0.00741,
 0.00631,
 0.00492,
 0.00392,
 0.00379,
 0.00343,
 0.00284,
 0.00233,
 0.00206,
 0.00161,
 0.00156,
 0.00177,
 0.00181,
 0.00207,
 0.0016,
 0.00159,
 0.00125,
 0.00119,
 0.00092,
 0.00057,
 0.00047,
 0.00039,
 0.00027,
 0.0003,
 0.00029,
 0.00037,
 0.00038,
 0.00037,
 0.00044,
 0.00077,
 0.00063,
 0.001,
 0.00225,
 0.00197,
 0.00375,
 0.00015,
 1e-05]
```

```
In [14]: cdf = []
         cdf.append(pdf[0])

         for i in range(1,len(pdf)):
             cdf.append(np.around(cdf[i-1]+pdf[i],decimals=6))

         cdf
```

```
Out[14]: [0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          0.0,
          2e-05,
          6e-05,
          0.00021,
          0.00079,
          0.00289,
          0.00876,
          0.02051,
          0.03614,
          0.05394,
          0.07066,
          0.08735,
          0.10411,
          0.11953,
          0.13273,
          0.14396,
          0.15367,
          0.16246,
          0.17056,
          0.17842,
          0.18613,
          0.19361,
          0.20071,
          0.20734,
          0.21351,
          0.21941,
          0.22503,
          0.23056,
          0.23572,
          0.24053,
          0.2453,
          0.24982,
          0.25413,
          0.25811,
          0.26197,
          0.26553,
          0.26904,
0.27237,
0.27569,
0.27893,
0.28194,
0.28498,
0.28792,
0.29085,
0.2939,
0.29692,
0.29995,
0.30296,
0.30601,
0.30923,
0.31243,
0.31566,
0.31906,
0.32248,
0.32609,
0.32987,
0.33382,
0.33796,
0.3421,
0.34628,
0.35055,
0.35489,
0.35933,
0.36386,
0.36831,
0.37283,
0.37728,
0.38162,
0.38604,
0.39037,
0.39483,
0.39942,
0.40388,
0.40846,
0.413,
0.41744,
0.42193,
0.42638,
0.4307,
0.43506,
0.43946,
0.44384,
0.44822,
0.45269,
0.45715,
0.46147,
0.4657,
0.46987,
0.4741,
0.47826,
0.48224,
0.48626,
0.49013,
0.49406,
0.4979,
0.50151,
0.50512,
0.50873,
0.51235,
0.51593,
0.51952,
0.52308,
0.52661,
0.53016,
0.53355,
0.53699,
0.54048,
0.54393,
0.54741,
0.55085,
0.55429,
0.5578,
0.56148,
0.56522,
0.56899,
0.57266,
0.57647,
0.5802,
0.58398,
0.58775,
0.59147,
0.59493,
0.59843,
0.60193,
0.60531,
0.6087,
0.61207,
0.61531,
0.61842,
0.62155,
0.62473,
0.62797,
0.63099,
0.63404,
0.63687,
0.63977,
0.64245,
0.64503,
0.64754,
0.6499,
0.65221,
0.65446,
0.65651,
0.65851,
0.66037,
0.66227,
0.66418,
0.666,
0.66771,
0.66929,
0.67086,
0.6724,
0.67383,
0.67522,
0.67654,
0.67786,
0.67915,
0.68045,
0.68164,
0.68281,
0.68399,
0.68525,
0.68652,
0.6878,
0.68918,
0.69058,
0.69195,
0.6934,
0.69487,
0.69653,
0.69837,
0.70038,
0.70258,
0.70512,
0.70781,
0.71075,
0.71378,
0.7169,
0.71999,
0.72307,
0.72634,
0.72939,
0.73245,
0.73541,
0.73859,
0.74205,
0.7454,
0.74906,
0.75314,
0.75806,
0.76391,
0.77058,
0.77774,
0.78528,
0.79294,
0.80165,
0.81185,
0.82321,
0.83586,
0.84897,
0.86241,
0.87531,
0.88734,
0.8986,
0.90793,
0.91679,
0.92515,
0.93321,
0.94062,
0.94693,
0.95185,
0.95577,
0.95956,
0.96299,
0.96583,
0.96816,
0.97022,
0.97183,
0.97339,
0.97516,
0.97697,
0.97904,
0.98064,
0.98223,
0.98348,
0.98467,
0.98559,
0.98616,
0.98663,
0.98702,
0.98729,
0.98759,
0.98788,
0.98825,
0.98863,
0.989,
0.98944,
0.99021,
0.99084,
0.99184,
0.99409,
0.99606,
0.99981,
0.99996,
0.99997]
```

```
In [15]: eq_levels = np.around(np.multiply(cdf,255))
```

```
In [16]: eq_levels
```

```
Out[16]: array([  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
                  0.,   0.,   0.,   0.,   0.,   1.,   2.,   5.,   9.,  14.,  18.,
                 22.,  27.,  30.,  34.,  37.,  39.,  41.,  43.,  45.,  47.,  49.,
                 51.,  53.,  54.,  56.,  57.,  59.,  60.,  61.,  63.,  64.,  65.,
                 66.,  67.,  68.,  69.,  69.,  70.,  71.,  72.,  73.,  73.,  74.,
                 75.,  76.,  76.,  77.,  78.,  79.,  80.,  80.,  81.,  82.,  83.,
                 84.,  85.,  86.,  87.,  88.,  89.,  90.,  92.,  93.,  94.,  95.,
                 96.,  97.,  98., 100., 101., 102., 103., 104., 105., 106., 108.,
                109., 110., 111., 112., 113., 114., 115., 117., 118., 119., 120.,
                121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
                132., 132., 133., 134., 135., 136., 137., 138., 139., 140., 140.,
                141., 142., 143., 144., 145., 146., 147., 148., 149., 150., 151.,
                152., 153., 153., 154., 155., 156., 157., 158., 158., 159., 160.,
                161., 162., 162., 163., 164., 164., 165., 166., 166., 167., 167.,
                168., 168., 169., 169., 170., 170., 171., 171., 171., 172., 172.,
                173., 173., 173., 174., 174., 174., 174., 175., 175., 175., 176.,
                176., 176., 177., 177., 178., 178., 179., 179., 180., 180., 181.,
                182., 183., 184., 184., 185., 186., 187., 188., 188., 189., 190.,
                191., 192., 193., 195., 196., 198., 200., 202., 204., 207., 210.,
                213., 216., 220., 223., 226., 229., 232., 234., 236., 238., 240.,
                241., 243., 244., 245., 246., 246., 247., 247., 248., 248., 249.,
                249., 250., 250., 250., 251., 251., 251., 251., 252., 252., 252.,
                252., 252., 252., 252., 252., 252., 253., 253., 253., 253., 254.,
                255., 255., 255.])
```

```
In [27]: eq_hist = np.zeros(hist.shape)

         for i in range(len(eq_hist)):
            eq_hist[i] = eq_levels[i]

        eq_hist = np.array(eq_hist,dtype=np.uint8)
```

```
In [28]: eq_hist
```

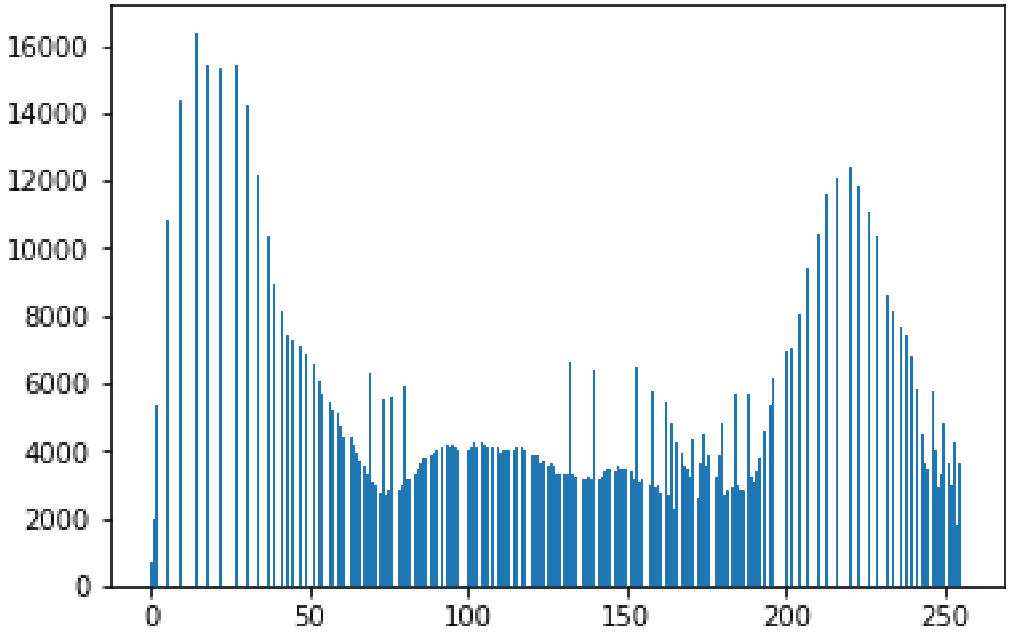
```
Out[28]: array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   1,   2,   5,   9,  14,  18,  22,  27,  30,  34,
                 37,  39,  41,  43,  45,  47,  49,  51,  53,  54,  56,  57,  59,
                 60,  61,  63,  64,  65,  66,  67,  68,  69,  69,  70,  71,  72,
                 73,  73,  74,  75,  76,  76,  77,  78,  79,  80,  80,  81,  82,
                 83,  84,  85,  86,  87,  88,  89,  90,  92,  93,  94,  95,  96,
                 97,  98, 100, 101, 102, 103, 104, 105, 106, 108, 109, 110, 111,
                112, 113, 114, 115, 117, 118, 119, 120, 121, 122, 123, 124, 125,
                126, 127, 128, 129, 130, 131, 132, 132, 133, 134, 135, 136, 137,
                138, 139, 140, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149,
                150, 151, 152, 153, 153, 154, 155, 156, 157, 158, 158, 159, 160,
                161, 162, 162, 163, 164, 164, 165, 166, 166, 167, 167, 168, 168,
                169, 169, 170, 170, 171, 171, 171, 172, 172, 173, 173, 173, 174,
                174, 174, 174, 175, 175, 175, 176, 176, 176, 177, 177, 178, 178,
                179, 179, 180, 180, 181, 182, 183, 184, 184, 185, 186, 187, 188,
                188, 189, 190, 191, 192, 193, 195, 196, 198, 200, 202, 204, 207,
                210, 213, 216, 220, 223, 226, 229, 232, 234, 236, 238, 240, 241,
                243, 244, 245, 246, 246, 247, 247, 248, 248, 249, 249, 250, 250,
                250, 251, 251, 251, 251, 252, 252, 252, 252, 252, 252, 252, 252,
                252, 253, 253, 253, 253, 254, 255, 255, 255], dtype=uint8)
```

```
In [29]: final_hist = np.zeros(256)

         for i in range(len(final_hist)):
             final_hist[eq_hist[i]]+=hist[i]

         final_hist
```

```
Out[29]: array([  728.,   1936.,   5401.,      0.,      0.,  10821.,      0.,      0.,
                    0.,  14391.,      0.,      0.,      0.,      0.,  16384.,      0.,
                    0.,      0.,  15396.,      0.,      0.,      0.,  15370.,      0.,
                    0.,      0.,      0.,  15431.,      0.,      0.,  14197.,      0.,
                    0.,      0.,  12152.,      0.,      0.,  10335.,      0.,   8940.,
                    0.,   8095.,      0.,   7457.,      0.,   7232.,      0.,   7098.,
                    0.,   6888.,      0.,   6534.,      0.,   6107.,   5679.,      0.,
                 5436.,   5173.,      0.,   5090.,   4748.,   4426.,      0.,   4390.,
                 4162.,   3966.,   3668.,   3550.,   3282.,   6294.,   3059.,   2981.,
                 2774.,   5508.,   2694.,   2807.,   5571.,   2775.,   2808.,   2962.,
                 5927.,   3134.,   3153.,   3328.,   3479.,   3634.,   3807.,   3816.,
                 3848.,   3935.,   3999.,      0.,   4085.,   4175.,   4095.,   4162.,
                 4094.,   3993.,   4069.,      0.,   3988.,   4104.,   4228.,   4102.,
                 4221.,   4183.,   4091.,      0.,   4135.,   4099.,   3979.,   4011.,
                 4054.,   4035.,   4030.,   4113.,      0.,   4103.,   3981.,   3895.,
        3841.,  3892.,  3827.,  3662.,  3705.,  3564.,  3617.,  3537.,
        3326.,  3321.,  3327.,  3331.,  6602.,  3278.,  3251.,  3269.,
        3119.,  3167.,  3213.,  3179.,  6375.,  3168.,  3227.,  3387.,
        3445.,  3475.,  3383.,  3506.,  3431.,  3480.,  3467.,  3423.,
        3182.,  6446.,  3109.,  3117.,  3098.,  2983.,  5752.,  2930.,
        2983.,  2781.,  5412.,  2671.,  4843.,  2313.,  4296.,  3955.,
        3553.,  3504.,  3252.,  4323.,  2605.,  3614.,  4455.,  3513.,
        3829.,  2686.,  3220.,  3877.,  4820.,  2711.,  2793.,  2872.,
        5675.,  3006.,  2807.,  2820.,  5652.,  3189.,  3083.,  3373.,
        3754.,  4534.,     0.,  5389.,  6143.,     0.,  6591.,     0.,
        6945.,     0.,  7054.,     0.,  8021.,     0.,     0.,  9393.,
           0.,     0., 10456.,     0.,     0., 11643.,     0.,     0.,
       12068.,     0.,     0.,     0., 12370.,     0.,     0., 11875.,
           0.,     0., 11074.,     0.,     0., 10362.,     0.,     0.,
        8586.,     0.,  8161.,     0.,  7692.,     0.,  7417.,     0.,
        6819.,  5808.,     0.,  4527.,  3606.,  3490.,  5777.,  4046.,
        2912.,  3295.,  4840.,  3611.,  3010.,  4283.,  1813.,  3596.])
```

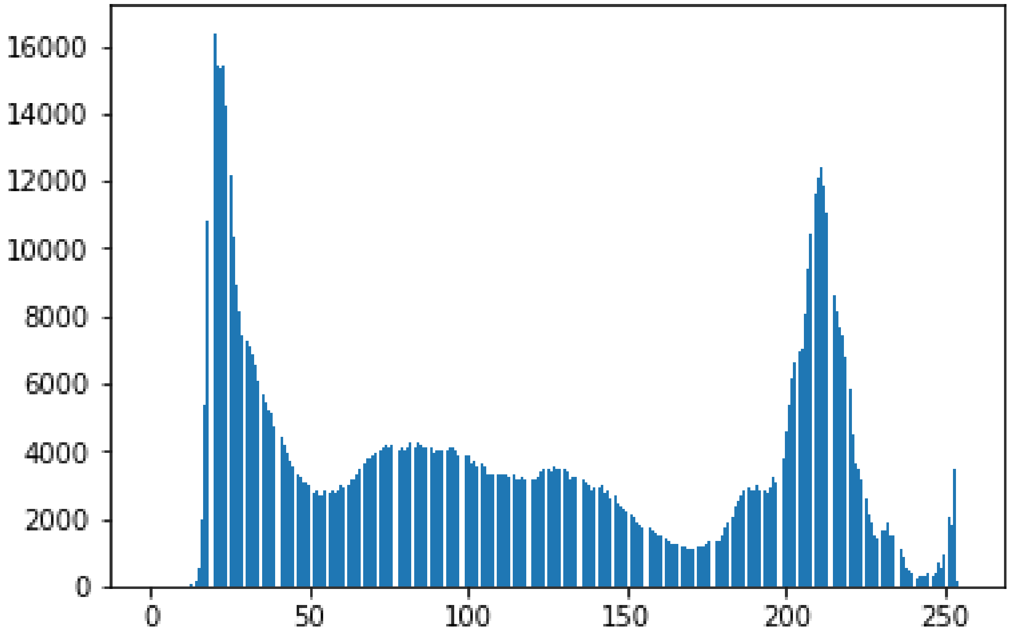
```
In [30]: plt.bar(index,final_hist)
```

```
Out[30]: <BarContainer object of 256 artists>
```

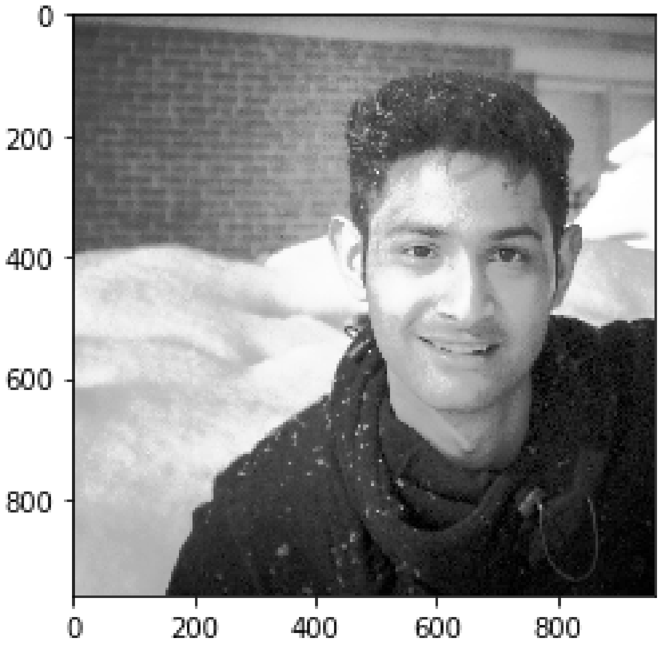


```
In [22]: plt.bar(index,hist)
```

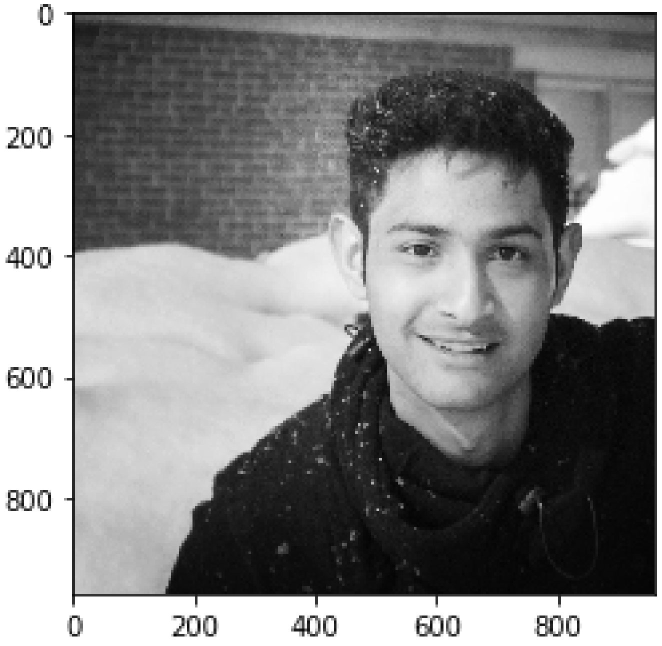
```
Out[22]: <BarContainer object of 256 artists>
```



```
In [46]: out=np.zeros(gray.shape)
         for i in range(gray.shape[0]):
             for j in range(gray.shape[1]):
                 out[i][j]=eq_levels[gray[i][j]]
         out = np.array(out,dtype=np.uint8)

In [47]: out

Out[47]: array([[ 92, 100, 105, ...,  80,  86,  96],
               [ 98, 102, 102, ...,  80,  83,  90],
               [106, 105, 100, ...,  80,  81,  85],
               ...,
               [193, 193, 193, ...,   1,   2,   5],
               [195, 195, 195, ...,   1,   2,   5],
               [195, 195, 195, ...,   1,   2,   5]], dtype=uint8)

In [48]: plt.imshow(out,cmap='gray')

Out[48]: <matplotlib.image.AxesImage at 0x1286f5358>
```



```
In [49]: plt.imshow(gray,cmap='gray')

Out[49]: <matplotlib.image.AxesImage at 0x1287c15f8>
```



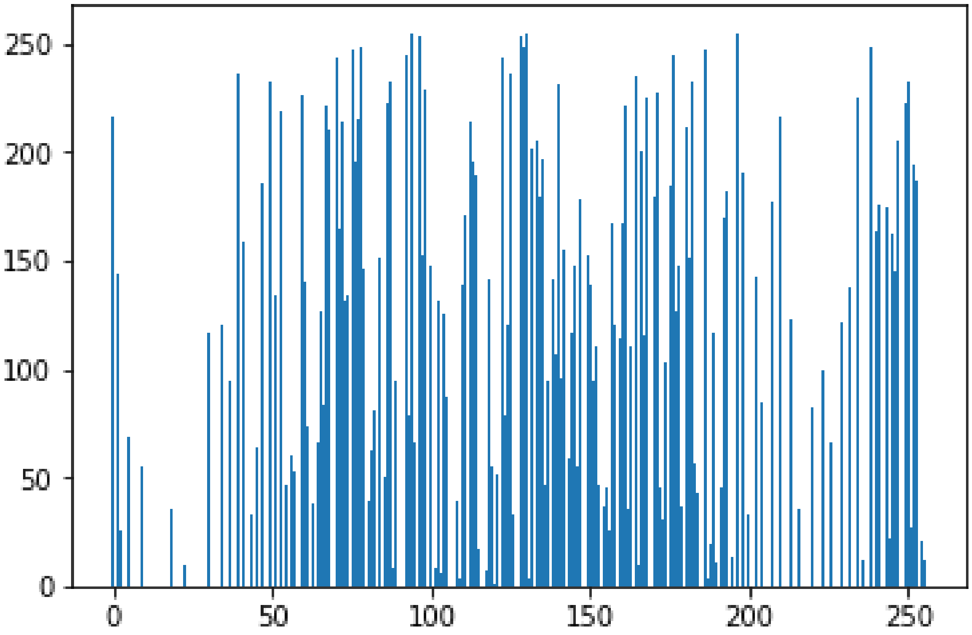
```
In [50]: out_hist = np.zeros(256)
         for i in range(out.shape[0]):
             for j in range(out.shape[1]):
                 out_hist[out[i,j]] = out_hist[out[i,j]]+1
```

```
In [51]: out_hist = np.array(out_hist,dtype=np.uint8)
         out_hist
```

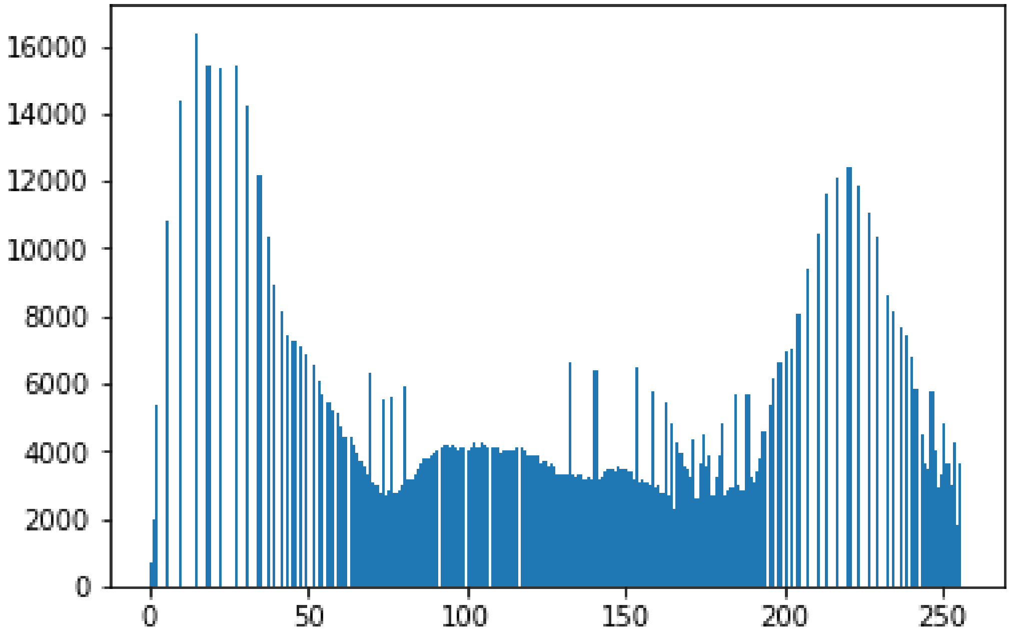
```
Out[51]: array([216, 144,  25,   0,   0,  69,   0,   0,   0,  55,   0,   0,   0,
                  0,   0,   0,   0,   0,  36,   0,   0,   0,  10,   0,   0,   0,
                  0,  71,   0,   0, 117,   0,   0,   0, 120,   0,   0,  95,   0,
                236,   0, 159,   0,  33,   0,  64,   0, 186,   0, 232,   0, 134,
                  0, 219,  47,   0,  60,  53,   0, 226, 140,  74,   0,  38,  66,
                126,  84, 222, 210, 150, 243, 165, 214, 132, 134, 247, 195, 215,
                248, 146,  39,  62,  81,   0, 151,  50, 223, 232,   8,  95, 159,
                  0, 245,  79, 255,  66, 254, 153, 229,   0, 148,   8, 132,   6,
                125,  87, 251,   0,  39,   3, 139, 171, 214, 195, 190,  17,   0,
                  7, 141,  55,   1,  52, 243,  78, 121, 236,  33, 209, 254, 249,
                255,   3, 202, 206, 179, 197,  47,  95, 141, 107, 231,  96, 155,
                 59, 117, 147,  55, 178, 103, 152, 139,  95, 110,  46,  37,  45,
                 26, 167, 120, 114, 167, 221,  36, 111, 235,   9, 200, 115, 225,
                176, 180, 227,  45,  30, 103, 185, 245, 126, 148,  37, 212, 151,
                233,  56,  43, 190, 247,   4,  20, 117,  11,  45, 170, 182,   0,
                 13, 255,   0, 191,   0,  33,   0, 142,   0,  85,   0,   0, 177,
                  0,   0, 216,   0,   0, 123,   0,   0,  36,   0,   0,   0,  82,
                  0,   0,  99,   0,   0,  66,   0,   0, 122,   0,   0, 138,   0,
                225,   0,  12,   0, 249,   0, 163, 176,   0, 175,  22, 162, 145,
                206,  96, 223, 232,  27, 194, 187,  21,  12], dtype=uint8)
```

```
In [52]: plt.bar(index,out_hist)
```

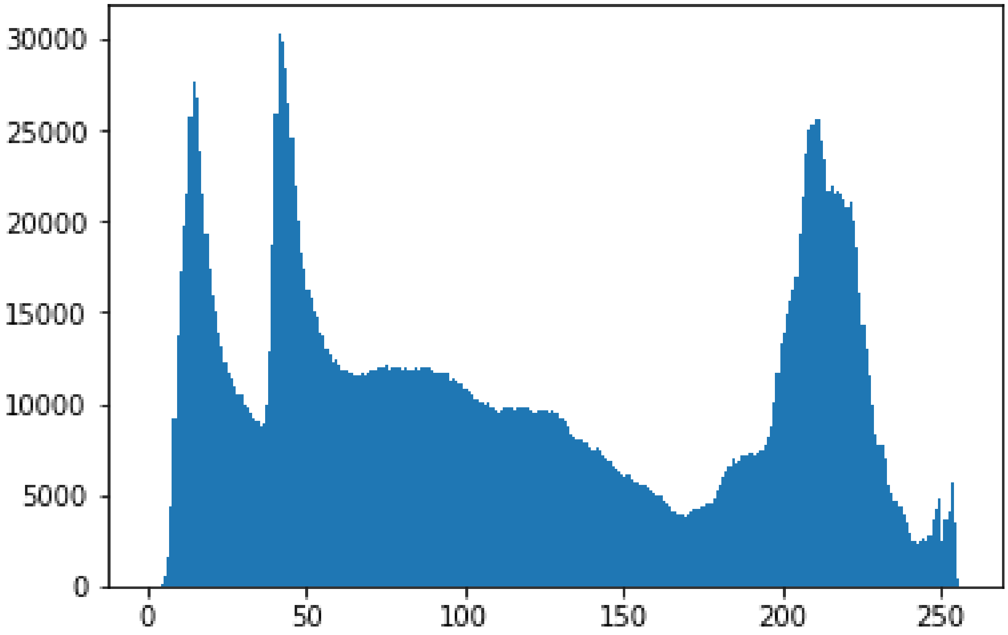
```
Out[52]: <BarContainer object of 256 artists>
```



```
In [53]: plt.hist(out.ravel(),256,[0,256])

Out[53]: (array([  728.,  1936.,  5401.,     0.,     0., 10821.,     0.,     0.,
                     0., 14391.,     0.,     0.,     0.,     0., 16384.,     0.,
                     0.,     0., 15396.,     0.,     0.,     0., 15370.,     0.,
                     0.,     0.,     0., 15431.,     0.,     0., 14197.,     0.,
                     0.,     0., 12152.,     0.,     0., 10335.,     0.,  8940.,
                     0.,  8095.,     0.,  7457.,     0.,  7232.,     0.,  7098.,
                     0.,  6888.,     0.,  6534.,     0.,  6107.,  5679.,     0.,
                  5436.,  5173.,     0.,  5090.,  4748.,  4426.,     0.,  4390.,
                  4162.,  3966.,  3668.,  3550.,  3282.,  6294.,  3059.,  2981.,
                  2774.,  5508.,  2694.,  2807.,  5571.,  2775.,  2808.,  2962.,
                  5927.,  3134.,  3153.,  3328.,  3479.,  3634.,  3807.,  3816.,
                  3848.,  3935.,  3999.,     0.,  4085.,  4175.,  4095.,  4162.,
                  4094.,  3993.,  4069.,     0.,  3988.,  4104.,  4228.,  4102.,
                  4221.,  4183.,  4091.,     0.,  4135.,  4099.,  3979.,  4011.,
                  4054.,  4035.,  4030.,  4113.,     0.,  4103.,  3981.,  3895.,
                  3841.,  3892.,  3827.,  3662.,  3705.,  3564.,  3617.,  3537.,
                  3326.,  3321.,  3327.,  3331.,  6602.,  3278.,  3251.,  3269.,
                  3119.,  3167.,  3213.,  3179.,  6375.,  3168.,  3227.,  3387.,
                  3445.,  3475.,  3383.,  3506.,  3431.,  3480.,  3467.,  3423.,
                  3182.,  6446.,  3109.,  3117.,  3098.,  2983.,  5752.,  2930.,
                  2983.,  2781.,  5412.,  2671.,  4843.,  2313.,  4296.,  3955.,
                  3553.,  3504.,  3252.,  4323.,  2605.,  3614.,  4455.,  3513.,
                  3829.,  2686.,  3220.,  3877.,  4820.,  2711.,  2793.,  2872.,
                  5675.,  3006.,  2807.,  2820.,  5652.,  3189.,  3083.,  3373.,
        3754.,  4534.,     0.,  5389.,  6143.,     0.,  6591.,     0.,
        6945.,     0.,  7054.,     0.,  8021.,     0.,     0.,  9393.,
           0.,     0., 10456.,     0.,     0., 11643.,     0.,     0.,
       12068.,     0.,     0.,     0., 12370.,     0.,     0., 11875.,
           0.,     0., 11074.,     0.,     0., 10362.,     0.,     0.,
        8586.,     0.,  8161.,     0.,  7692.,     0.,  7417.,     0.,
        6819.,  5808.,     0.,  4527.,  3606.,  3490.,  5777.,  4046.,
        2912.,  3295.,  4840.,  3611.,  3010.,  4283.,  1813.,  3596.]),
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
        11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
        22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
        33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
        44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
        55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
        66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
        77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
        88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
        99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
       110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
       121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
       132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
       143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
       154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
       165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
       176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
       187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
       198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
       209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
       220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
       231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
       242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
       253., 254., 255., 256.]),
<a list of 256 Patch objects>)
```



```
In [54]: plt.hist(image.ravel(),256,[0,256])

Out[54]: (array([1.0000e+00, 3.0000e+00, 1.1000e+01, 3.6000e+01, 1.3600e+02,
                5.6400e+02, 1.6230e+03, 4.3590e+03, 9.2610e+03, 1.3725e+04,
                1.7203e+04, 1.9697e+04, 2.1563e+04, 2.5752e+04, 2.7741e+04,
                2.6778e+04, 2.3856e+04, 2.1448e+04, 1.9279e+04, 1.7449e+04,
                1.6002e+04, 1.5037e+04, 1.3903e+04, 1.3218e+04, 1.2298e+04,
                1.1694e+04, 1.1451e+04, 1.0985e+04, 1.0499e+04, 1.0584e+04,
                9.9930e+03, 9.7670e+03, 9.4280e+03, 9.2190e+03, 9.0150e+03,
                8.7210e+03, 8.9560e+03, 9.9120e+03, 1.2814e+04, 1.8721e+04,
                2.5932e+04, 3.0355e+04, 2.9881e+04, 2.8446e+04, 2.6528e+04,
                2.4577e+04, 2.1939e+04, 2.0000e+04, 1.8323e+04, 1.7394e+04,
                1.6267e+04, 1.5848e+04, 1.5051e+04, 1.4769e+04, 1.3840e+04,
                1.3717e+04, 1.3040e+04, 1.2698e+04, 1.2313e+04, 1.2416e+04,
                1.2076e+04, 1.1903e+04, 1.1871e+04, 1.1764e+04, 1.1654e+04,
                1.1616e+04, 1.1490e+04, 1.1664e+04, 1.1596e+04, 1.1751e+04,
                1.1830e+04, 1.1907e+04, 1.1918e+04, 1.1971e+04, 1.2051e+04,
                1.2086e+04, 1.1887e+04, 1.2054e+04, 1.2045e+04, 1.1950e+04,
                1.1877e+04, 1.1963e+04, 1.1916e+04, 1.1820e+04, 1.2002e+04,
                1.1802e+04, 1.1939e+04, 1.2020e+04, 1.2028e+04, 1.1790e+04,
                1.1703e+04, 1.1687e+04, 1.1759e+04, 1.1672e+04, 1.1714e+04,
                1.1325e+04, 1.1396e+04, 1.1267e+04, 1.1069e+04, 1.0861e+04,
                1.0786e+04, 1.0725e+04, 1.0469e+04, 1.0190e+04, 1.0117e+04,
                1.0044e+04, 9.8820e+03, 1.0126e+04, 9.8470e+03, 9.6230e+03,
                9.4930e+03, 9.6140e+03, 9.8550e+03, 9.7420e+03, 9.7550e+03,
                9.7080e+03, 9.7800e+03, 9.7520e+03, 9.8470e+03, 9.7300e+03,
       9.7080e+03, 9.5240e+03, 9.5490e+03, 9.6560e+03, 9.6250e+03,
       9.5850e+03, 9.4810e+03, 9.6010e+03, 9.5480e+03, 9.4350e+03,
       9.2060e+03, 9.0970e+03, 8.7810e+03, 8.3940e+03, 8.2500e+03,
       7.9640e+03, 7.9960e+03, 7.8940e+03, 7.8460e+03, 7.6050e+03,
       7.5130e+03, 7.5610e+03, 7.3850e+03, 7.1510e+03, 6.9760e+03,
       6.8790e+03, 6.5170e+03, 6.4170e+03, 6.2190e+03, 6.1560e+03,
       6.0380e+03, 6.0940e+03, 5.8900e+03, 5.6710e+03, 5.7190e+03,
       5.5430e+03, 5.6000e+03, 5.4570e+03, 5.2950e+03, 5.1310e+03,
       4.9750e+03, 4.8830e+03, 4.7140e+03, 4.4980e+03, 4.3420e+03,
       4.0420e+03, 4.0760e+03, 3.9840e+03, 3.9940e+03, 3.8130e+03,
       3.9550e+03, 4.0100e+03, 4.1760e+03, 4.1590e+03, 4.3050e+03,
       4.4060e+03, 4.4710e+03, 4.5170e+03, 4.8100e+03, 5.2020e+03,
       5.5220e+03, 5.9230e+03, 6.2150e+03, 6.5960e+03, 6.9390e+03,
       6.7770e+03, 6.8780e+03, 7.2110e+03, 7.1220e+03, 7.3160e+03,
       7.2600e+03, 7.1840e+03, 7.3630e+03, 7.4980e+03, 7.7910e+03,
       8.2060e+03, 8.8030e+03, 1.0022e+04, 1.1720e+04, 1.3329e+04,
       1.3956e+04, 1.4900e+04, 1.5681e+04, 1.6289e+04, 1.7004e+04,
       1.9289e+04, 2.1375e+04, 2.3741e+04, 2.4990e+04, 2.5335e+04,
       2.5610e+04, 2.5646e+04, 2.4449e+04, 2.3451e+04, 2.1688e+04,
       2.1891e+04, 2.1551e+04, 2.1633e+04, 2.1568e+04, 2.1183e+04,
       2.0715e+04, 2.1077e+04, 2.0071e+04, 1.8596e+04, 1.6092e+04,
       1.4393e+04, 1.3026e+04, 1.1503e+04, 9.9660e+03, 8.2790e+03,
       7.7190e+03, 7.7200e+03, 6.9430e+03, 5.5660e+03, 5.0940e+03,
       4.7120e+03, 4.3880e+03, 4.3420e+03, 3.8980e+03, 3.5120e+03,
       2.8970e+03, 2.5100e+03, 2.3760e+03, 2.4630e+03, 2.5780e+03,
       2.4980e+03, 2.6890e+03, 3.6340e+03, 4.2020e+03, 4.7830e+03,
       2.4340e+03, 3.6320e+03, 4.1280e+03, 5.6280e+03, 3.4350e+03,
       4.1200e+02]),
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
        11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
        22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
        33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
        44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
        55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
        66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
        77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
        88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
        99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
       110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
       121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
       132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
       143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
       154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
       165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
       176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
       187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
       198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
       209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
        220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
        231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
        242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
        253., 254., 255., 256.]),
 <a list of 256 Patch objects>)
```



```
In [ ]:
```

Histogram specification

```
In [1]: import cv2
        import numpy as np
        import matplotlib.pyplot as plt
```

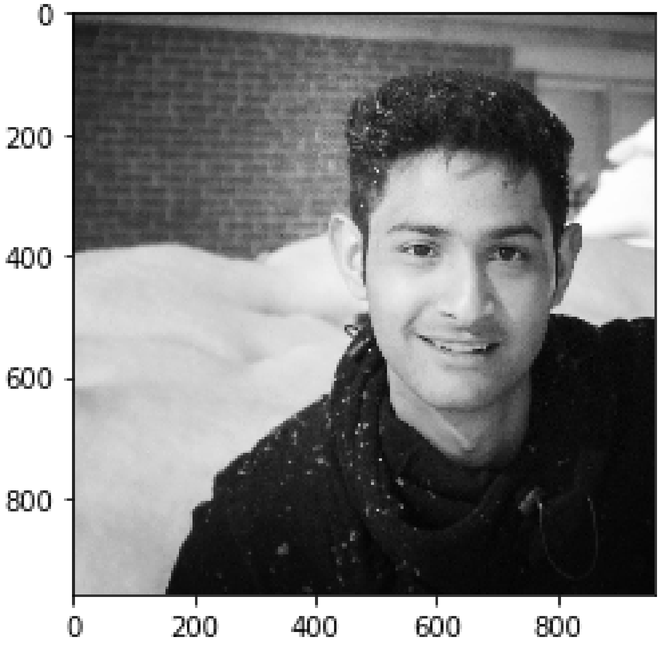
```
In [12]: image1 = cv2.imread('sample7.jpg')
         image1 = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
```

```
In [16]: image1.shape
```

```
Out[16]: (959, 960)
```

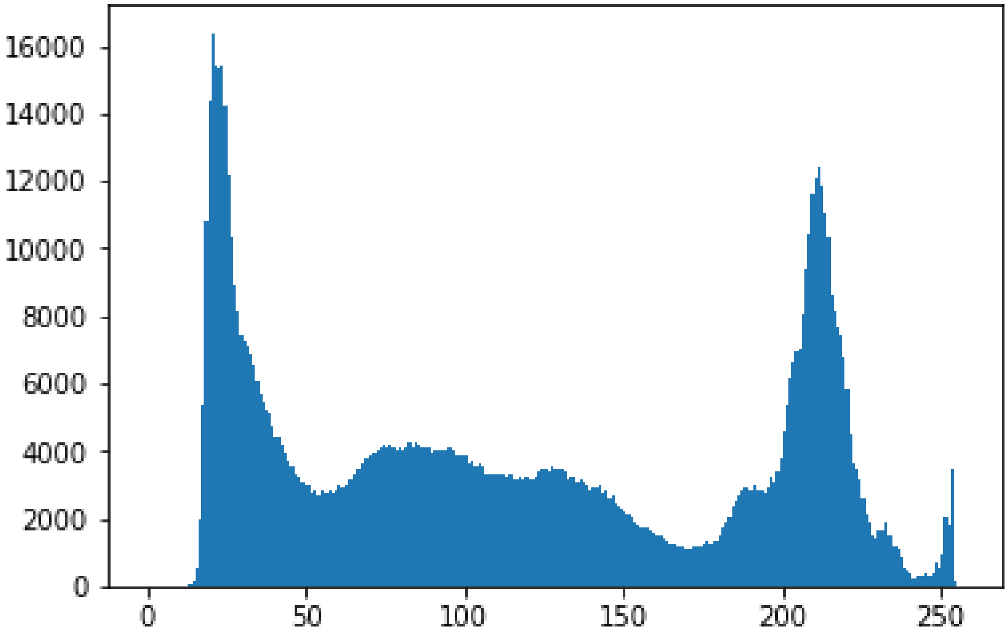
```
In [13]: plt.imshow(image1,cmap='gray')
```

```
Out[13]: <matplotlib.image.AxesImage at 0x13195c780>
```

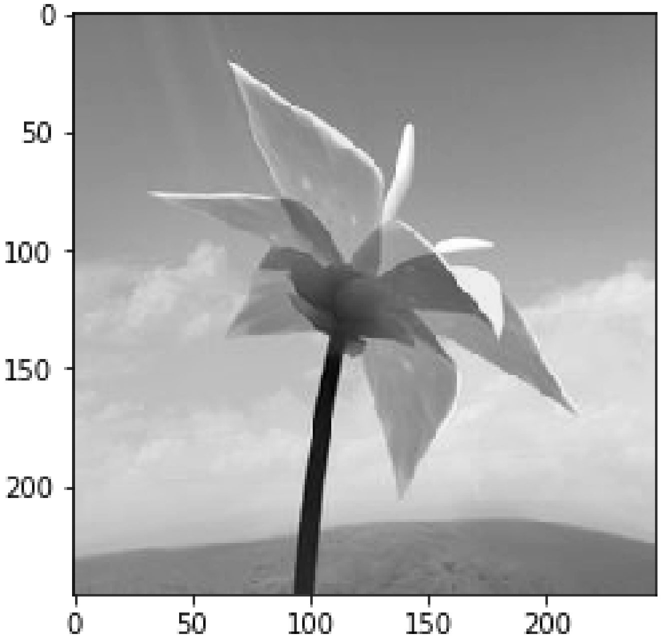


```
In [18]: plt.hist(image1.ravel(),256,[0,256])
```

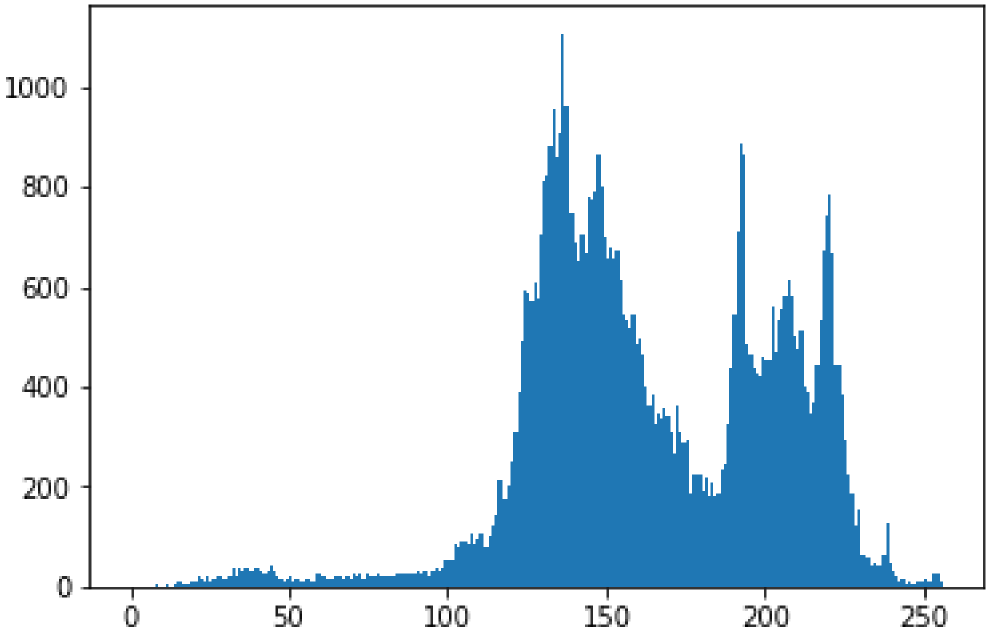
```
Out[18]: (array([0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 4.0000e+00, 1.4000e+01, 4.1000e+01, 1.3900e+02,
                 5.3000e+02, 1.9360e+03, 5.4010e+03, 1.0821e+04, 1.4391e+04,
                 1.6384e+04, 1.5396e+04, 1.5370e+04, 1.5431e+04, 1.4197e+04,
                 1.2152e+04, 1.0335e+04, 8.9400e+03, 8.0950e+03, 7.4570e+03,
                 7.2320e+03, 7.0980e+03, 6.8880e+03, 6.5340e+03, 6.1070e+03,
                 5.6790e+03, 5.4360e+03, 5.1730e+03, 5.0900e+03, 4.7480e+03,
                 4.4260e+03, 4.3900e+03, 4.1620e+03, 3.9660e+03, 3.6680e+03,
                 3.5500e+03, 3.2820e+03, 3.2280e+03, 3.0660e+03, 3.0590e+03,
                 2.9810e+03, 2.7740e+03, 2.8010e+03, 2.7070e+03, 2.6940e+03,
                 2.8070e+03, 2.7780e+03, 2.7930e+03, 2.7750e+03, 2.8080e+03,
                 2.9620e+03, 2.9490e+03, 2.9780e+03, 3.1340e+03, 3.1530e+03,
                 3.3280e+03, 3.4790e+03, 3.6340e+03, 3.8070e+03, 3.8160e+03,
                 3.8480e+03, 3.9350e+03, 3.9990e+03, 4.0850e+03, 4.1750e+03,
                 4.0950e+03, 4.1620e+03, 4.0940e+03, 3.9930e+03, 4.0690e+03,
                 3.9880e+03, 4.1040e+03, 4.2280e+03, 4.1020e+03, 4.2210e+03,
                 4.1830e+03, 4.0910e+03, 4.1350e+03, 4.0990e+03, 3.9790e+03,
                 4.0110e+03, 4.0540e+03, 4.0350e+03, 4.0300e+03, 4.1130e+03,
                 4.1030e+03, 3.9810e+03, 3.8950e+03, 3.8410e+03, 3.8920e+03,
                 3.8270e+03, 3.6620e+03, 3.7050e+03, 3.5640e+03, 3.6170e+03,
                 3.5370e+03, 3.3260e+03, 3.3210e+03, 3.3270e+03, 3.3310e+03,
                 3.2930e+03, 3.3090e+03, 3.2780e+03, 3.2510e+03, 3.2690e+03,
                 3.1190e+03, 3.1670e+03, 3.2130e+03, 3.1790e+03, 3.2070e+03,
                 3.1680e+03, 3.1680e+03, 3.2270e+03, 3.3870e+03, 3.4450e+03,
                 3.4750e+03, 3.3830e+03, 3.5060e+03, 3.4310e+03, 3.4800e+03,
                 3.4670e+03, 3.4230e+03, 3.1820e+03, 3.2260e+03, 3.2200e+03,
                 3.1090e+03, 3.1170e+03, 3.0980e+03, 2.9830e+03, 2.8660e+03,
                 2.8860e+03, 2.9300e+03, 2.9830e+03, 2.7810e+03, 2.8090e+03,
                 2.6030e+03, 2.6710e+03, 2.4640e+03, 2.3790e+03, 2.3130e+03,
                 2.1720e+03, 2.1240e+03, 2.0690e+03, 1.8860e+03, 1.8370e+03,
                 1.7160e+03, 1.7500e+03, 1.7540e+03, 1.6800e+03, 1.5720e+03,
                 1.4550e+03, 1.4500e+03, 1.4180e+03, 1.3210e+03, 1.2840e+03,
                 1.2180e+03, 1.2120e+03, 1.1840e+03, 1.1950e+03, 1.0960e+03,
                 1.0740e+03, 1.0900e+03, 1.1620e+03, 1.1720e+03, 1.1790e+03,
                 1.2720e+03, 1.2930e+03, 1.2640e+03, 1.3320e+03, 1.3540e+03,
                 1.5240e+03, 1.6960e+03, 1.8470e+03, 2.0300e+03, 2.3390e+03,
                 2.4810e+03, 2.7110e+03, 2.7930e+03, 2.8720e+03, 2.8440e+03,
                 2.8310e+03, 3.0060e+03, 2.8070e+03, 2.8200e+03, 2.7230e+03,
                 2.9290e+03, 3.1890e+03, 3.0830e+03, 3.3730e+03, 3.7540e+03,
                 4.5340e+03, 5.3890e+03, 6.1430e+03, 6.5910e+03, 6.9450e+03,
                 7.0540e+03, 8.0210e+03, 9.3930e+03, 1.0456e+04, 1.1643e+04,
                 1.2068e+04, 1.2370e+04, 1.1875e+04, 1.1074e+04, 1.0362e+04,
                 8.5860e+03, 8.1610e+03, 7.6920e+03, 7.4170e+03, 6.8190e+03,
                 5.8080e+03, 4.5270e+03, 3.6060e+03, 3.4900e+03, 3.1600e+03,
                 2.6170e+03, 2.1490e+03, 1.8970e+03, 1.4790e+03, 1.4330e+03,
                 1.6260e+03, 1.6690e+03, 1.9050e+03, 1.4740e+03, 1.4610e+03,
                 1.1510e+03, 1.0960e+03, 8.4300e+02, 5.2100e+02, 4.3000e+02,
       3.5700e+02, 2.4500e+02, 2.7500e+02, 2.6800e+02, 3.3800e+02,
       3.5000e+02, 3.4000e+02, 4.0700e+02, 7.1200e+02, 5.7700e+02,
       9.2300e+02, 2.0710e+03, 1.8130e+03, 3.4500e+03, 1.3800e+02,
       8.0000e+00]),
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
        11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
        22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
        33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
        44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
        55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
        66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
        77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
        88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
        99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
       110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
       121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
       132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
       143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
       154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
       165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
       176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
       187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
       198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
       209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
       220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
       231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
       242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
       253., 254., 255., 256.]),
<a list of 256 Patch objects>)
```



```
In [14]: image2 = cv2.imread('flower1.jpg')
         image2 = cv2.cvtColor(image2,cv2.COLOR_BGR2GRAY)

In [15]: plt.imshow(image2,cmap='gray')

Out[15]: <matplotlib.image.AxesImage at 0x131d64a58>
```



```
In [19]: plt.hist(image2.ravel(),256,[0,256])

Out[19]: (array([0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
                 0.000e+00, 0.000e+00, 2.000e+00, 0.000e+00, 1.000e+00, 3.000e+00,
                 1.000e+00, 1.000e+00, 6.000e+00, 7.000e+00, 3.000e+00, 4.000e+00,
                 5.000e+00, 8.000e+00, 9.000e+00, 1.800e+01, 1.300e+01, 1.000e+01,
                 2.000e+01, 1.100e+01, 1.400e+01, 2.100e+01, 2.100e+01, 1.700e+01,
                 1.700e+01, 2.200e+01, 3.400e+01, 1.800e+01, 3.400e+01, 3.200e+01,
                 3.700e+01, 2.900e+01, 3.300e+01, 3.800e+01, 3.500e+01, 3.100e+01,
                 2.600e+01, 3.200e+01, 4.200e+01, 2.900e+01, 1.900e+01, 1.600e+01,
                 1.100e+01, 1.700e+01, 1.900e+01, 9.000e+00, 1.700e+01, 1.200e+01,
                 1.200e+01, 1.500e+01, 1.600e+01, 7.000e+00, 2.300e+01, 2.600e+01,
                 1.900e+01, 2.100e+01, 1.500e+01, 1.700e+01, 2.200e+01, 2.000e+01,
                 2.000e+01, 1.600e+01, 1.900e+01, 1.300e+01, 2.400e+01, 1.800e+01,
                 2.500e+01, 1.600e+01, 2.300e+01, 1.800e+01, 2.200e+01, 2.100e+01,
                 2.400e+01, 2.100e+01, 2.000e+01, 2.100e+01, 1.800e+01, 1.900e+01,
                 2.400e+01, 2.600e+01, 2.500e+01, 2.300e+01, 2.800e+01, 2.500e+01,
                 3.300e+01, 2.300e+01, 3.300e+01, 3.000e+01, 2.000e+01, 3.100e+01,
                 3.400e+01, 3.000e+01, 3.900e+01, 5.200e+01, 5.100e+01, 5.200e+01,
                 8.200e+01, 8.000e+01, 9.000e+01, 9.100e+01, 8.500e+01, 1.040e+02,
                 8.500e+01, 9.400e+01, 1.040e+02, 8.000e+01, 7.900e+01, 9.800e+01,
                 1.240e+02, 1.420e+02, 2.140e+02, 1.770e+02, 1.750e+02, 2.030e+02,
                 2.490e+02, 3.080e+02, 3.900e+02, 4.910e+02, 5.900e+02, 5.850e+02,
                 5.710e+02, 6.090e+02, 5.760e+02, 7.050e+02, 8.130e+02, 8.210e+02,
                 8.790e+02, 9.540e+02, 8.570e+02, 9.090e+02, 1.107e+03, 9.590e+02,
                 7.480e+02, 7.460e+02, 6.900e+02, 6.500e+02, 7.040e+02, 6.690e+02,
                 7.820e+02, 7.760e+02, 7.920e+02, 8.630e+02, 7.990e+02, 7.000e+02,
                 6.560e+02, 6.790e+02, 6.590e+02, 6.720e+02, 6.160e+02, 5.430e+02,
                 5.350e+02, 5.150e+02, 5.440e+02, 4.860e+02, 4.940e+02, 4.660e+02,
                 4.000e+02, 3.600e+02, 3.820e+02, 3.250e+02, 3.490e+02, 3.350e+02,
                 3.590e+02, 3.410e+02, 3.080e+02, 2.660e+02, 3.600e+02, 3.090e+02,
                 2.850e+02, 2.950e+02, 1.880e+02, 2.260e+02, 2.220e+02, 2.240e+02,
                 1.920e+02, 2.160e+02, 1.790e+02, 2.080e+02, 1.800e+02, 1.870e+02,
                 2.320e+02, 2.460e+02, 3.260e+02, 4.380e+02, 5.450e+02, 7.080e+02,
                 8.850e+02, 8.630e+02, 4.840e+02, 4.650e+02, 4.360e+02, 4.250e+02,
                 4.220e+02, 4.580e+02, 4.510e+02, 4.520e+02, 5.580e+02, 4.720e+02,
                 5.320e+02, 5.530e+02, 5.840e+02, 6.120e+02, 5.790e+02, 4.990e+02,
                 4.760e+02, 5.110e+02, 3.990e+02, 3.870e+02, 3.460e+02, 3.690e+02,
                 4.420e+02, 5.350e+02, 6.700e+02, 7.430e+02, 7.870e+02, 6.650e+02,
                 4.450e+02, 4.400e+02, 3.820e+02, 2.920e+02, 2.240e+02, 1.860e+02,
                 1.210e+02, 1.550e+02, 6.100e+01, 6.400e+01, 5.900e+01, 4.400e+01,
                 4.500e+01, 4.200e+01, 4.100e+01, 6.400e+01, 1.280e+02, 4.800e+01,
                 3.000e+01, 2.200e+01, 7.000e+00, 1.600e+01, 6.000e+00, 8.000e+00,
                 5.000e+00, 3.000e+00, 7.000e+00, 7.000e+00, 1.400e+01, 1.100e+01,
                 1.200e+01, 2.600e+01, 2.400e+01, 8.000e+00]),
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
        11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
        22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
        33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
        44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
        55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
        66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
        77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
        88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
        99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
       110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
       121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
       132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
       143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
       154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
       165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
       176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
       187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
       198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
       209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
       220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
       231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
       242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
       253., 254., 255., 256.]),
<a list of 256 Patch objects>)
```

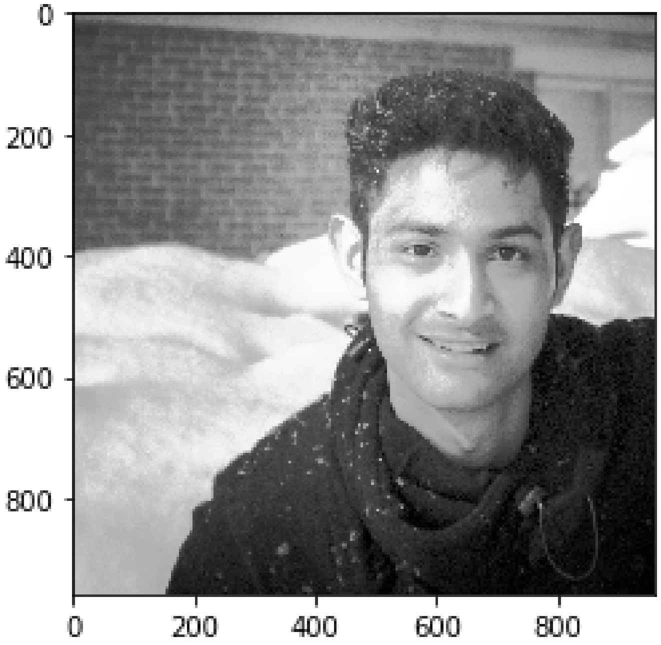


```
In [36]: def equalisation(image):
             hist = np.zeros(256)

             for i in range(image.shape[0]):
                 for j in range(image.shape[1]):
                     hist[image[i][j]] = hist[image[i][j]]+1

             pdf = [np.around(i/(image.shape[0]*image.shape[1]),decimals=5) for i in hist]
             cdf = []
             cdf.append(pdf[0])

             for i in range(1,len(pdf1)):
                 cdf.append(np.around(cdf[i-1]+pdf[i],decimals=6))

             eq_levels = np.around(np.multiply(cdf,255))

             out=np.zeros(image.shape)
             for i in range(image.shape[0]):
                 for j in range(image.shape[1]):
                     out[i][j]=eq_levels[image[i][j]]
             out = np.array(out,dtype=np.uint8)
             return out,eq_levels
```

```
In [37]: out1,levels1 = equalisation(image1)
```

```
In [38]: plt.imshow(out1,cmap='gray')
```

```
Out[38]: <matplotlib.image.AxesImage at 0x128f4ce48>
```



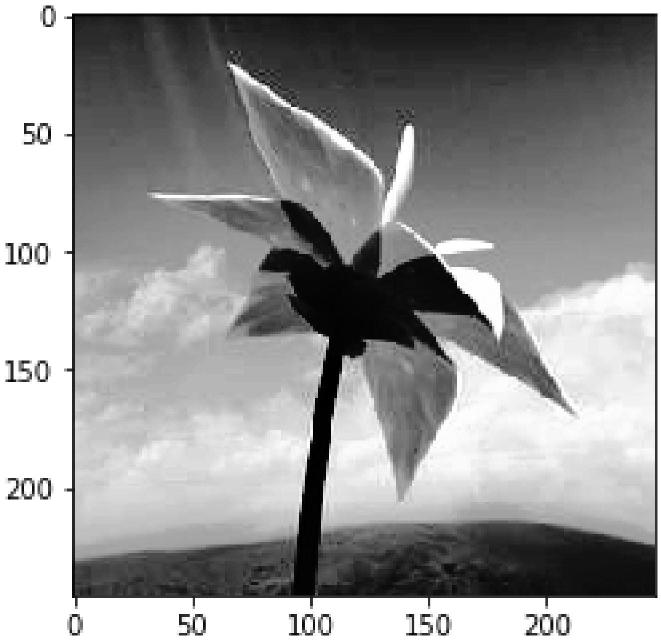
```
In [39]: levels1
```

```
Out[39]: array([  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
                  0.,   0.,   0.,   0.,   0.,   1.,   2.,   5.,   9.,  14.,  18.,
                 22.,  27.,  30.,  34.,  37.,  39.,  41.,  43.,  45.,  47.,  49.,
                 51.,  53.,  54.,  56.,  57.,  59.,  60.,  61.,  63.,  64.,  65.,
                 66.,  67.,  68.,  69.,  69.,  70.,  71.,  72.,  73.,  73.,  74.,
                 75.,  76.,  76.,  77.,  78.,  79.,  80.,  80.,  81.,  82.,  83.,
                 84.,  85.,  86.,  87.,  88.,  89.,  90.,  92.,  93.,  94.,  95.,
                 96.,  97.,  98., 100., 101., 102., 103., 104., 105., 106., 108.,
                109., 110., 111., 112., 113., 114., 115., 117., 118., 119., 120.,
                121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
                132., 132., 133., 134., 135., 136., 137., 138., 139., 140., 140.,
                141., 142., 143., 144., 145., 146., 147., 148., 149., 150., 151.,
                152., 153., 153., 154., 155., 156., 157., 158., 158., 159., 160.,
                161., 162., 162., 163., 164., 164., 165., 166., 166., 167., 167.,
                168., 168., 169., 169., 170., 170., 171., 171., 171., 172., 172.,
                173., 173., 173., 174., 174., 174., 174., 175., 175., 175., 176.,
                176., 176., 177., 177., 178., 178., 179., 179., 180., 180., 181.,
                182., 183., 184., 184., 185., 186., 187., 188., 188., 189., 190.,
                191., 192., 193., 195., 196., 198., 200., 202., 204., 207., 210.,
                213., 216., 220., 223., 226., 229., 232., 234., 236., 238., 240.,
                241., 243., 244., 245., 246., 246., 247., 247., 248., 248., 249.,
                249., 250., 250., 250., 251., 251., 251., 251., 252., 252., 252.,
                252., 252., 252., 252., 252., 252., 253., 253., 253., 253., 254.,
                255., 255., 255.])
```

```
In [40]: out2,levels2 = equalisation(image2)
```

```
In [41]: plt.imshow(out2,cmap='gray')
```

```
Out[41]: <matplotlib.image.AxesImage at 0x129004278>
```



```
In [42]: levels2

Out[42]: array([  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
                  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
                  0.,   0.,   0.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,
                  1.,   1.,   1.,   2.,   2.,   2.,   2.,   2.,   2.,   2.,   3.,
                  3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.,
                  4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.,
                  4.,   4.,   5.,   5.,   5.,   5.,   5.,   5.,   5.,   5.,   5.,
                  5.,   5.,   5.,   6.,   6.,   6.,   6.,   6.,   6.,   6.,   6.,
                  6.,   6.,   7.,   7.,   7.,   7.,   7.,   7.,   7.,   7.,   8.,
                  8.,   8.,   8.,   9.,   9.,   9.,  10.,  10.,  10.,  11.,  11.,
                 12.,  12.,  12.,  13.,  13.,  14.,  15.,  16.,  16.,  17.,  18.,
                 19.,  21.,  23.,  26.,  28.,  31.,  33.,  36.,  38.,  42.,  45.,
                 49.,  53.,  57.,  61.,  65.,  69.,  72.,  76.,  78.,  81.,  84.,
                 87.,  90.,  94.,  97., 101., 104., 107., 110., 112., 115., 118.,
                121., 123., 125., 127., 130., 132., 134., 136., 137., 139., 141.,
                142., 143., 145., 146., 148., 149., 150., 152., 153., 154., 155.,
                156., 157., 158., 159., 160., 161., 162., 162., 163., 164., 165.,
                166., 167., 169., 171., 174., 178., 182., 184., 186., 188., 189.,
                191., 193., 195., 197., 199., 201., 204., 206., 208., 211., 213.,
                215., 217., 220., 221., 223., 224., 226., 228., 230., 233., 236.,
                239., 242., 244., 246., 247., 249., 250., 250., 251., 252., 252.,
                252., 252., 253., 253., 253., 253., 253., 254., 254., 254., 254.,
                254., 254., 254., 254., 255., 255., 255., 255., 255., 255., 255.,
                255., 255., 255.])
```

```
In [51]: final_mapping = np.zeros(256)
         j = 0
         i = 0
         while i<256:
             if levels2[j]>=levels1[i]:
                 final_mapping[i]=j
                 i+=1
                 continue
             else:
                 j+=1
```

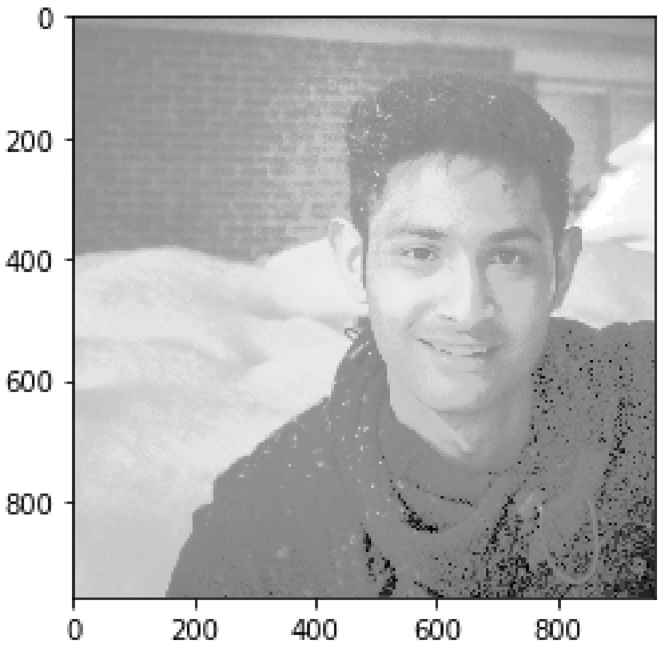
```
In [52]: final_mapping
```

```
Out[52]: array([  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
                  0.,   0.,   0.,   0.,   0.,  25.,  36.,  68., 102., 115., 120.,
                123., 125., 126., 128., 129., 130., 130., 131., 131., 132., 132.,
                133., 133., 134., 134., 134., 135., 135., 135., 136., 136., 136.,
                137., 137., 137., 137., 137., 138., 138., 138., 139., 139., 139.,
                139., 139., 139., 140., 140., 141., 141., 141., 141., 142., 142.,
                142., 143., 143., 143., 144., 144., 144., 145., 145., 145., 146.,
                146., 146., 147., 147., 147., 148., 148., 148., 149., 149., 150.,
                150., 150., 151., 151., 152., 152., 152., 153., 153., 154., 154.,
                154., 155., 155., 156., 156., 157., 157., 158., 158., 158., 159.,
                159., 159., 160., 160., 161., 161., 162., 163., 163., 164., 164.,
                164., 165., 166., 167., 167., 168., 169., 169., 170., 171., 172.,
                172., 173., 173., 174., 175., 176., 177., 178., 178., 179., 180.,
                181., 182., 182., 184., 185., 185., 186., 187., 187., 188., 188.,
                189., 189., 189., 189., 190., 190., 190., 190., 190., 191., 191.,
                191., 191., 191., 191., 191., 191., 191., 192., 192., 192., 192.,
                192., 192., 192., 192., 192., 192., 193., 193., 193., 193., 193.,
                193., 194., 194., 194., 195., 195., 196., 196., 196., 197., 198.,
                198., 199., 199., 200., 201., 202., 203., 204., 204., 206., 207.,
                208., 210., 211., 213., 215., 217., 218., 219., 219., 220., 221.,
                221., 222., 222., 223., 223., 223., 224., 224., 225., 225., 225.,
                225., 226., 226., 226., 228., 228., 228., 228., 229., 229., 229.,
                229., 229., 229., 229., 229., 229., 233., 233., 233., 233., 238.,
                246., 246., 246.])
```

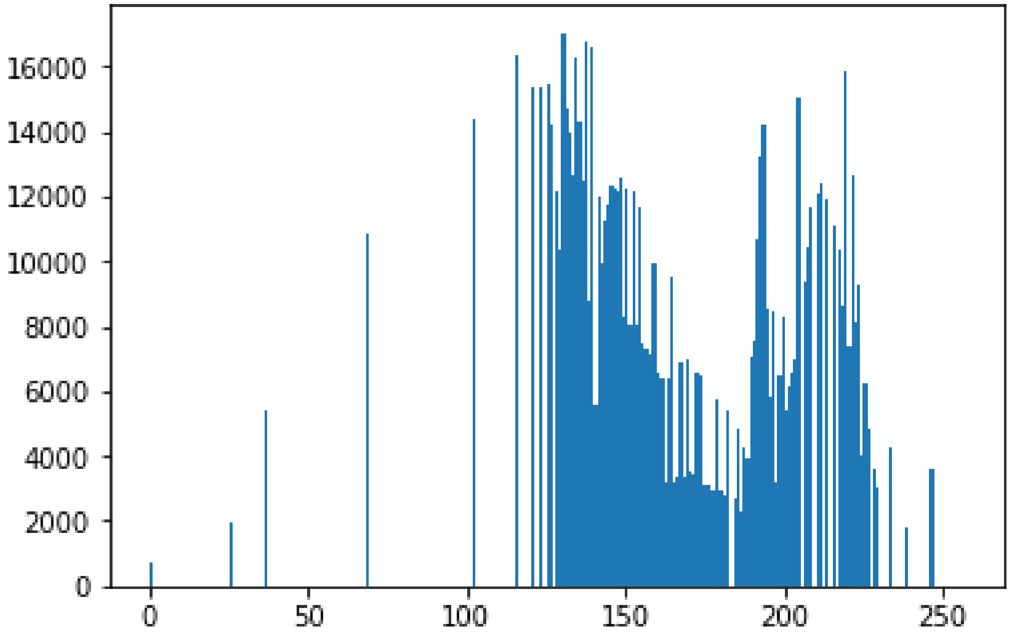
```
In [53]: final_out=np.zeros(image1.shape)
         for i in range(image1.shape[0]):
             for j in range(image1.shape[1]):
                 final_out[i][j]=final_mapping[image1[i][j]]
         final_out = np.array(final_out,dtype=np.uint8)
```

```
In [55]: plt.imshow(final_out,cmap='gray')
```

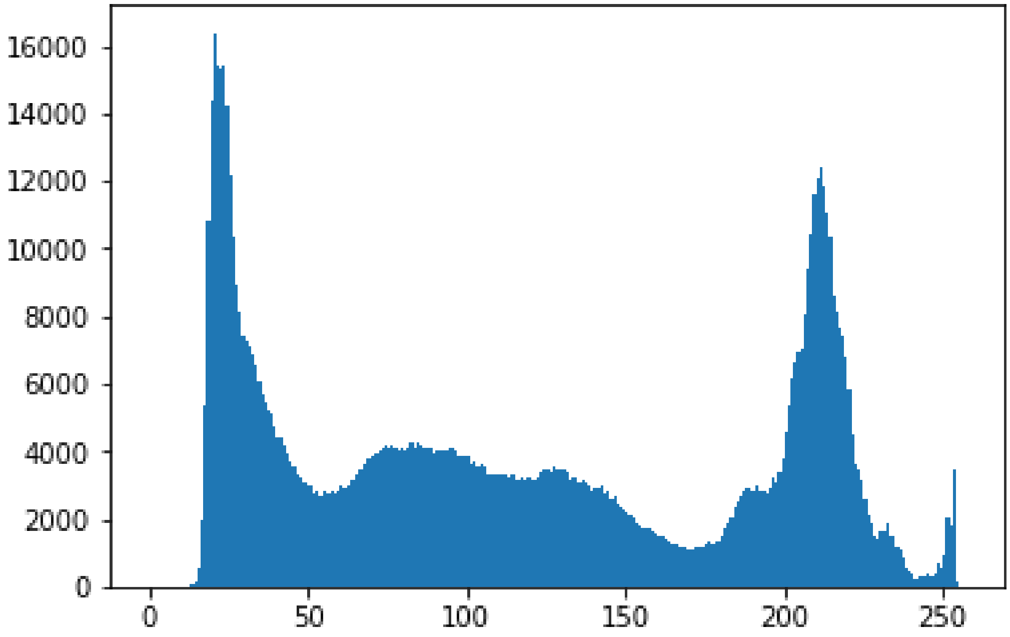
```
Out[55]: <matplotlib.image.AxesImage at 0x12945eeb8>
```



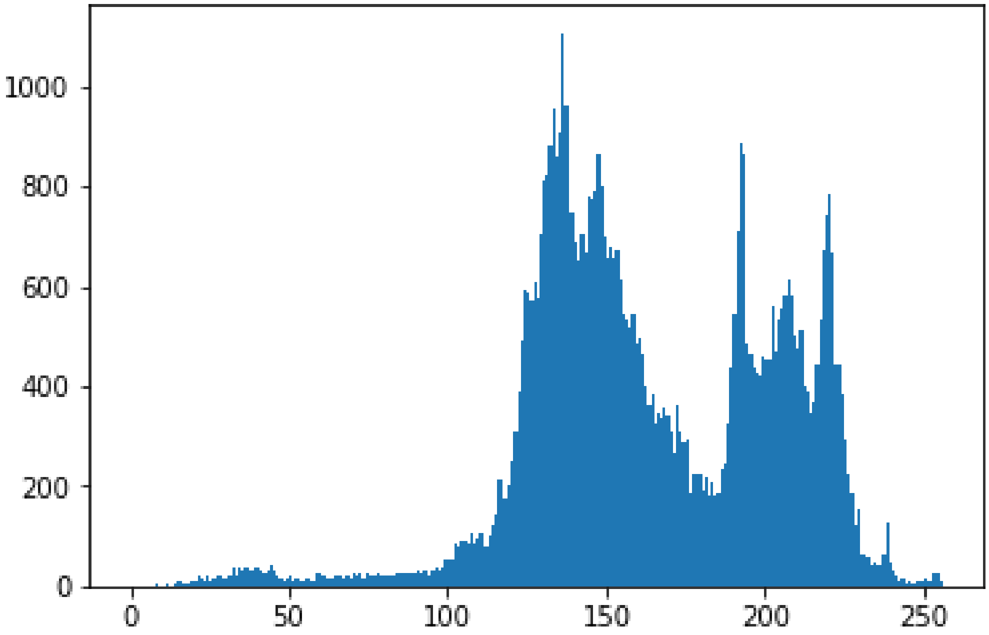
```
In [56]: plt.hist(final_out.ravel(),256,[0,256])

Out[56]: (array([  728.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,  1936.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,  5401.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0., 10821.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0., 14391.,     0.,
                     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,     0.,     0., 16384.,     0.,     0.,     0.,     0.,
                 15396.,     0.,     0., 15370.,     0., 15431., 14197.,     0.,
                 12152., 10335., 17035., 14689., 13986., 12641., 16288., 14264.,
                 12518., 16794.,  8814., 16580.,  5583., 12023.,  9960., 11257.,
                 11782., 12355., 12249., 12161., 12551.,  8274., 12213.,  8065.,
                 12178.,  8084., 11628.,  7489.,  7269.,  7154.,  9974.,  9933.,
                  6529.,  6388.,  3167.,  6392.,  9543.,  3227.,  3387.,  6920.,
                  3383.,  6937.,  3480.,  3467.,  6605.,  6446.,  3109.,  3117.,
                  3098.,  2983.,  5752.,  2930.,  2983.,  2781.,  5412.,     0.,
                  2671.,  4843.,  2313.,  4296.,  3955.,  7057.,  7575., 10674.,
       13248., 14201.,  8547.,  5813.,  8472.,  3189.,  6456.,  8288.,
        5389.,  6143.,  6591.,  6945., 15075.,     0.,  9393., 10456.,
       11643.,     0., 12068., 12370.,     0., 11875.,     0., 11074.,
           0., 10362.,  8586., 15853.,  7417., 12627.,  8133.,  9267.,
        4046.,  6207.,  4840.,     0.,  3611.,  3010.,     0.,     0.,
           0.,  4283.,     0.,     0.,     0.,     0.,  1813.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,  3596.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.]),
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
        11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
        22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
        33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
        44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
        55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
        66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
        77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
        88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
        99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
       110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
       121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
       132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
       143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
       154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
       165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
       176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
       187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
       198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
       209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
       220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
       231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
       242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
       253., 254., 255., 256.]),
<a list of 256 Patch objects>)
```



```
In [57]: plt.hist(image1.ravel(),256,[0,256])

Out[57]: (array([0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                0.0000e+00, 4.0000e+00, 1.4000e+01, 4.1000e+01, 1.3900e+02,
                5.3000e+02, 1.9360e+03, 5.4010e+03, 1.0821e+04, 1.4391e+04,
                1.6384e+04, 1.5396e+04, 1.5370e+04, 1.5431e+04, 1.4197e+04,
                1.2152e+04, 1.0335e+04, 8.9400e+03, 8.0950e+03, 7.4570e+03,
                7.2320e+03, 7.0980e+03, 6.8880e+03, 6.5340e+03, 6.1070e+03,
                5.6790e+03, 5.4360e+03, 5.1730e+03, 5.0900e+03, 4.7480e+03,
                4.4260e+03, 4.3900e+03, 4.1620e+03, 3.9660e+03, 3.6680e+03,
                3.5500e+03, 3.2820e+03, 3.2280e+03, 3.0660e+03, 3.0590e+03,
                2.9810e+03, 2.7740e+03, 2.8010e+03, 2.7070e+03, 2.6940e+03,
                2.8070e+03, 2.7780e+03, 2.7930e+03, 2.7750e+03, 2.8080e+03,
                2.9620e+03, 2.9490e+03, 2.9780e+03, 3.1340e+03, 3.1530e+03,
                3.3280e+03, 3.4790e+03, 3.6340e+03, 3.8070e+03, 3.8160e+03,
                3.8480e+03, 3.9350e+03, 3.9990e+03, 4.0850e+03, 4.1750e+03,
                4.0950e+03, 4.1620e+03, 4.0940e+03, 3.9930e+03, 4.0690e+03,
                3.9880e+03, 4.1040e+03, 4.2280e+03, 4.1020e+03, 4.2210e+03,
                4.1830e+03, 4.0910e+03, 4.1350e+03, 4.0990e+03, 3.9790e+03,
                4.0110e+03, 4.0540e+03, 4.0350e+03, 4.0300e+03, 4.1130e+03,
                4.1030e+03, 3.9810e+03, 3.8950e+03, 3.8410e+03, 3.8920e+03,
                3.8270e+03, 3.6620e+03, 3.7050e+03, 3.5640e+03, 3.6170e+03,
                3.5370e+03, 3.3260e+03, 3.3210e+03, 3.3270e+03, 3.3310e+03,
                3.2930e+03, 3.3090e+03, 3.2780e+03, 3.2510e+03, 3.2690e+03,
                3.1190e+03, 3.1670e+03, 3.2130e+03, 3.1790e+03, 3.2070e+03,
        3.1680e+03, 3.1680e+03, 3.2270e+03, 3.3870e+03, 3.4450e+03,
        3.4750e+03, 3.3830e+03, 3.5060e+03, 3.4310e+03, 3.4800e+03,
        3.4670e+03, 3.4230e+03, 3.1820e+03, 3.2260e+03, 3.2200e+03,
        3.1090e+03, 3.1170e+03, 3.0980e+03, 2.9830e+03, 2.8660e+03,
        2.8860e+03, 2.9300e+03, 2.9830e+03, 2.7810e+03, 2.8090e+03,
        2.6030e+03, 2.6710e+03, 2.4640e+03, 2.3790e+03, 2.3130e+03,
        2.1720e+03, 2.1240e+03, 2.0690e+03, 1.8860e+03, 1.8370e+03,
        1.7160e+03, 1.7500e+03, 1.7540e+03, 1.6800e+03, 1.5720e+03,
        1.4550e+03, 1.4500e+03, 1.4180e+03, 1.3210e+03, 1.2840e+03,
        1.2180e+03, 1.2120e+03, 1.1840e+03, 1.1950e+03, 1.0960e+03,
        1.0740e+03, 1.0900e+03, 1.1620e+03, 1.1720e+03, 1.1790e+03,
        1.2720e+03, 1.2930e+03, 1.2640e+03, 1.3320e+03, 1.3540e+03,
        1.5240e+03, 1.6960e+03, 1.8470e+03, 2.0300e+03, 2.3390e+03,
        2.4810e+03, 2.7110e+03, 2.7930e+03, 2.8720e+03, 2.8440e+03,
        2.8310e+03, 3.0060e+03, 2.8070e+03, 2.8200e+03, 2.7230e+03,
        2.9290e+03, 3.1890e+03, 3.0830e+03, 3.3730e+03, 3.7540e+03,
        4.5340e+03, 5.3890e+03, 6.1430e+03, 6.5910e+03, 6.9450e+03,
        7.0540e+03, 8.0210e+03, 9.3930e+03, 1.0456e+04, 1.1643e+04,
        1.2068e+04, 1.2370e+04, 1.1875e+04, 1.1074e+04, 1.0362e+04,
        8.5860e+03, 8.1610e+03, 7.6920e+03, 7.4170e+03, 6.8190e+03,
        5.8080e+03, 4.5270e+03, 3.6060e+03, 3.4900e+03, 3.1600e+03,
        2.6170e+03, 2.1490e+03, 1.8970e+03, 1.4790e+03, 1.4330e+03,
        1.6260e+03, 1.6690e+03, 1.9050e+03, 1.4740e+03, 1.4610e+03,
        1.1510e+03, 1.0960e+03, 8.4300e+02, 5.2100e+02, 4.3000e+02,
        3.5700e+02, 2.4500e+02, 2.7500e+02, 2.6800e+02, 3.3800e+02,
        3.5000e+02, 3.4000e+02, 4.0700e+02, 7.1200e+02, 5.7700e+02,
        9.2300e+02, 2.0710e+03, 1.8130e+03, 3.4500e+03, 1.3800e+02,
        8.0000e+00]),
 array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
         11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
         22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
         33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
         44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
         55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
         66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
         77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
         88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
         99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
        110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
        121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
        132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
        143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
        154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
        165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
        176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
        187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
        198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
        209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
                 220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
                 231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
                 242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
                 253., 254., 255., 256.]),
          <a list of 256 Patch objects>)
```



```
In [58]: plt.hist(image2.ravel(),256,[0,256])

Out[58]: (array([0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
                 0.000e+00, 0.000e+00, 2.000e+00, 0.000e+00, 1.000e+00, 3.000e+00,
                 1.000e+00, 1.000e+00, 6.000e+00, 7.000e+00, 3.000e+00, 4.000e+00,
                 5.000e+00, 8.000e+00, 9.000e+00, 1.800e+01, 1.300e+01, 1.000e+01,
                 2.000e+01, 1.100e+01, 1.400e+01, 2.100e+01, 2.100e+01, 1.700e+01,
                 1.700e+01, 2.200e+01, 3.400e+01, 1.800e+01, 3.400e+01, 3.200e+01,
                 3.700e+01, 2.900e+01, 3.300e+01, 3.800e+01, 3.500e+01, 3.100e+01,
                 2.600e+01, 3.200e+01, 4.200e+01, 2.900e+01, 1.900e+01, 1.600e+01,
                 1.100e+01, 1.700e+01, 1.900e+01, 9.000e+00, 1.700e+01, 1.200e+01,
                 1.200e+01, 1.500e+01, 1.600e+01, 7.000e+00, 2.300e+01, 2.600e+01,
                 1.900e+01, 2.100e+01, 1.500e+01, 1.700e+01, 2.200e+01, 2.000e+01,
                 2.000e+01, 1.600e+01, 1.900e+01, 1.300e+01, 2.400e+01, 1.800e+01,
                 2.500e+01, 1.600e+01, 2.300e+01, 1.800e+01, 2.200e+01, 2.100e+01,
                 2.400e+01, 2.100e+01, 2.000e+01, 2.100e+01, 1.800e+01, 1.900e+01,
                 2.400e+01, 2.600e+01, 2.500e+01, 2.300e+01, 2.800e+01, 2.500e+01,
                 3.300e+01, 2.300e+01, 3.300e+01, 3.000e+01, 2.000e+01, 3.100e+01,
                 3.400e+01, 3.000e+01, 3.900e+01, 5.200e+01, 5.100e+01, 5.200e+01,
                 8.200e+01, 8.000e+01, 9.000e+01, 9.100e+01, 8.500e+01, 1.040e+02,
       8.500e+01, 9.400e+01, 1.040e+02, 8.000e+01, 7.900e+01, 9.800e+01,
       1.240e+02, 1.420e+02, 2.140e+02, 1.770e+02, 1.750e+02, 2.030e+02,
       2.490e+02, 3.080e+02, 3.900e+02, 4.910e+02, 5.900e+02, 5.850e+02,
       5.710e+02, 6.090e+02, 5.760e+02, 7.050e+02, 8.130e+02, 8.210e+02,
       8.790e+02, 9.540e+02, 8.570e+02, 9.090e+02, 1.107e+03, 9.590e+02,
       7.480e+02, 7.460e+02, 6.900e+02, 6.500e+02, 7.040e+02, 6.690e+02,
       7.820e+02, 7.760e+02, 7.920e+02, 8.630e+02, 7.990e+02, 7.000e+02,
       6.560e+02, 6.790e+02, 6.590e+02, 6.720e+02, 6.160e+02, 5.430e+02,
       5.350e+02, 5.150e+02, 5.440e+02, 4.860e+02, 4.940e+02, 4.660e+02,
       4.000e+02, 3.600e+02, 3.820e+02, 3.250e+02, 3.490e+02, 3.350e+02,
       3.590e+02, 3.410e+02, 3.080e+02, 2.660e+02, 3.600e+02, 3.090e+02,
       2.850e+02, 2.950e+02, 1.880e+02, 2.260e+02, 2.220e+02, 2.240e+02,
       1.920e+02, 2.160e+02, 1.790e+02, 2.080e+02, 1.800e+02, 1.870e+02,
       2.320e+02, 2.460e+02, 3.260e+02, 4.380e+02, 5.450e+02, 7.080e+02,
       8.850e+02, 8.630e+02, 4.840e+02, 4.650e+02, 4.360e+02, 4.250e+02,
       4.220e+02, 4.580e+02, 4.510e+02, 4.520e+02, 5.580e+02, 4.720e+02,
       5.320e+02, 5.530e+02, 5.840e+02, 6.120e+02, 5.790e+02, 4.990e+02,
       4.760e+02, 5.110e+02, 3.990e+02, 3.870e+02, 3.460e+02, 3.690e+02,
       4.420e+02, 5.350e+02, 6.700e+02, 7.430e+02, 7.870e+02, 6.650e+02,
       4.450e+02, 4.400e+02, 3.820e+02, 2.920e+02, 2.240e+02, 1.860e+02,
       1.210e+02, 1.550e+02, 6.100e+01, 6.400e+01, 5.900e+01, 4.400e+01,
       4.500e+01, 4.200e+01, 4.100e+01, 6.400e+01, 1.280e+02, 4.800e+01,
       3.000e+01, 2.200e+01, 7.000e+00, 1.600e+01, 6.000e+00, 8.000e+00,
       5.000e+00, 3.000e+00, 7.000e+00, 7.000e+00, 1.400e+01, 1.100e+01,
       1.200e+01, 2.600e+01, 2.400e+01, 8.000e+00]),
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
        11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
        22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
        33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
        44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
        55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
        66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
        77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
        88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
        99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
       110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
       121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
       132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
       143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
       154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
       165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
       176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
       187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
       198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
       209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
       220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
       231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
       242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
        253., 254., 255., 256.]),
 <a list of 256 Patch objects>)
```



```
In [ ]:
```

Dilation

```
In [3]: import numpy as np
        import cv2
        def pad(img,shp):
                p=np.zeros((shp[0]+1,shp[1]+1))
                p[1:,1:]=np.copy(img)
                p[0,1:]=img[0]
                p[1:,0]=img[:,0]
                p[0,0]=img[0,0]
                p[-1,0]=img[-1,0]
                return p

        def comp(sample,metric):
                for i in range(2):
                        for j in range(2):
                                if sample[i,j]==metric[i,j]:
                                        return True
                return False

        def slice(img):
                temp=np.array(img)
                print(temp.shape)
                int_slice=np.zeros((temp.shape[0],temp.shape[1],8))
                for x in range(8):
                        int_slice[:,:,x]=temp%(2)
                        temp=(temp/2).astype(int)
                return int_slice

        def stitch(int_slice,shp):
                out=np.zeros(shp)
                for x in range(8):
                        out=out+int_slice[:,:,x]*(2**x)
                return out


        img=cv2.imread('edges_detected.png',0)
        shp=img.shape
        temp=pad(img,shp)
        int_slice=slice(temp)
```

```
struct_el=np.array([[1,1],[1,1]])
int_slice_new=np.zeros((shp[0],shp[1],8))
for x in range(8):
        for i in range(shp[0]):
                for j in range(shp[1]):
                        if comp(int_slice[i:i+2,j:j+2,x],struct_el):
                                int_slice_new[i,j,x]=1

out=stitch(int_slice_new,shp)
out=np.array(out, dtype = np.uint8)
print(img)
print(out)
cv2.imshow('image', img)
cv2.imshow('dilated', out)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
(289, 433)
[[255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 ...
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]]
[[255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 ...
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]]
```

In [ ]:

Erosion

```
In [2]: import numpy as np
        import cv2
        def pad(img,shp):
                p=np.zeros((shp[0]+1,shp[1]+1))
                p[1:,1:]=np.copy(img)
                p[0,1:]=img[0]
                p[1:,0]=img[:,0]
                p[0,0]=img[0,0]
                p[-1,0]=img[-1,0]
                return p

        def comp(sample,metric):
                for i in range(2):
                        for j in range(2):
                                if sample[i,j]!=metric[i,j]:
                                        return False
                return True

        def slice(img):
                temp=np.array(img)
                print(temp.shape)
                int_slice=np.zeros((temp.shape[0],temp.shape[1],8))
                for x in range(8):
                        int_slice[:,:,x]=temp%(2)
                        temp=(temp/2).astype(int)
                return int_slice

        def stitch(int_slice,shp):
                out=np.zeros(shp)
                for x in range(8):
                        out=out+int_slice[:,:,x]*(2**x)
                return out


        img=cv2.imread('cat.jpeg',0)
        shp=img.shape
        temp=pad(img,shp)
        int_slice=slice(temp)
        struct_el=np.array([[1,1],[1,1]])
        int_slice_new=np.zeros((shp[0],shp[1],8))
        for x in range(8):
                for i in range(shp[0]):
                        for j in range(shp[1]):
                                if comp(int_slice[i:i+2,j:j+2,x],struct_el):
                                        int_slice_new[i,j,x]=1

        out=stitch(int_slice_new,shp)
        out=np.array(out, dtype = np.uint8)
        print(img)
        print(out)
        cv2.imshow('image', img)
        cv2.imshow('eroded', out)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

```
(351, 529)
[[14 21 29 ...  7  7  7]
 [13 20 28 ...  6  6  6]
 [12 19 28 ...  6  6  6]
 ...
 [ 8  7  5 ...  0  0  0]
 [ 9  8  6 ...  0  0  0]
 [10  9  7 ...  0  0  0]]
[[14  4 21 ...  7  7  7]
 [12  4 20 ...  6  6  6]
 [12  0 16 ...  6  6  6]
 ...
 [ 0  0  4 ...  0  0  0]
 [ 8  0  0 ...  0  0  0]
 [ 8  8  0 ...  0  0  0]]
```

```
In [ ]:
```

Closing

```
In [ ]: import numpy as np
        import cv2
        def pad(img,shp):
                p=np.zeros((shp[0]+1,shp[1]+1))
                p[1:,1:]=np.copy(img)
                p[0,1:]=img[0]
                p[1:,0]=img[:,0]
                p[0,0]=img[0,0]
                p[-1,0]=img[-1,0]
                return p

        def comp_erosion(sample,metric):
                for i in range(2):
                        for j in range(2):
                                if sample[i,j]!=metric[i,j]:
                                        return False
                return True

        def comp_dilation(sample,metric):
                for i in range(2):
                        for j in range(2):
                                if sample[i,j]==metric[i,j]:
                                        return True
                return False

        def slice(img):
                temp=np.array(img)
                print(temp.shape)
                int_slice=np.zeros((temp.shape[0],temp.shape[1],8))
                for x in range(8):
                        int_slice[:,:,x]=temp%(2)
                        temp=(temp/2).astype(int)
                return int_slice

        def stitch(int_slice,shp):
                out=np.zeros(shp)
                for x in range(8):
```

```
                out=out+int_slice[:,:,x]*(2**x)
        return out


img=cv2.imread('cat.jpeg',0)
shp=img.shape
temp=pad(img,shp)
int_slice=slice(temp)
struct_el=np.array([[1,1],[1,1]])
int_slice_new=np.zeros((shp[0],shp[1],8))
for x in range(8):
        for i in range(shp[0]):
                for j in range(shp[1]):
                        if comp_dilation(int_slice[i:i+2,j:j+2,x],struct_el):
                                int_slice_new[i,j,x]=1

int_slice=np.array(int_slice_new)
int_slice_new=np.zeros((shp[0],shp[1],8))
for x in range(8):
        test=pad(int_slice[:,:,x],shp)
        for i in range(shp[0]):
                for j in range(shp[1]):
                        if comp_erosion(test[i:i+2,j:j+2],struct_el):
                                int_slice_new[i,j,x]=1

out=stitch(int_slice_new,shp)
out=np.array(out, dtype = np.uint8)
print(img)
print(out)
cv2.imshow('image', img)
cv2.imshow('closed', out)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
(351, 529)
[[14 21 29 ...  7  7  7]
 [13 20 28 ...  6  6  6]
 [12 19 28 ...  6  6  6]
 ...
 [ 8  7  5 ...  0  0  0]
 [ 9  8  6 ...  0  0  0]
 [10  9  7 ...  0  0  0]]
[[14 14 29 ...  7  7  7]
 [14 14 29 ...  7  7  7]
 [13 13 29 ...  6  6  6]
 ...
 [ 7  7  6 ...  0  0  0]
 [ 9  9  7 ...  0  0  0]
 [ 9  9 11 ...  0  0  0]]
```

```
In [ ]:
```

Opening

```
In [ ]: import numpy as np
        import cv2
        def pad(img,shp):
                p=np.zeros((shp[0]+1,shp[1]+1))
                p[1:,1:]=np.copy(img)
                p[0,1:]=img[0]
                p[1:,0]=img[:,0]
                p[0,0]=img[0,0]
                p[-1,0]=img[-1,0]
                return p

        def comp_erosion(sample,metric):
                for i in range(2):
                        for j in range(2):
                                if sample[i,j]!=metric[i,j]:
                                        return False
                return True

        def comp_dilation(sample,metric):
                for i in range(2):
                        for j in range(2):
                                if sample[i,j]==metric[i,j]:
                                        return True
                return False

        def slice(img):
                temp=np.array(img)
                print(temp.shape)
                int_slice=np.zeros((temp.shape[0],temp.shape[1],8))
                for x in range(8):
                        int_slice[:,:,x]=temp%(2)
                        temp=(temp/2).astype(int)
                return int_slice

        def stitch(int_slice,shp):
                out=np.zeros(shp)
                for x in range(8):
```

```
                out=out+int_slice[:,:,x]*(2**x)
        return out


img=cv2.imread('morph1.jpg',0)
shp=img.shape
temp=pad(img,shp)
int_slice=slice(temp)
struct_el=np.array([[1,1],[1,1]])
int_slice_new=np.zeros((shp[0],shp[1],8))
for x in range(8):
        for i in range(shp[0]):
                for j in range(shp[1]):
                        if comp_erosion(int_slice[i:i+2,j:j+2,x],struct_el):
                                int_slice_new[i,j,x]=1

int_slice=np.array(int_slice_new)
int_slice_new=np.zeros((shp[0],shp[1],8))
for x in range(8):
        test=pad(int_slice[:,:,x],shp)
        for i in range(shp[0]):
                for j in range(shp[1]):
                        if comp_dilation(test[i:i+2,j:j+2],struct_el):
                                int_slice_new[i,j,x]=1

out=stitch(int_slice_new,shp)
out=np.array(out, dtype = np.uint8)
print(img)
print(out)
cv2.imshow('image', img)
cv2.imshow('opened', out)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
(451, 451)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 4 0 0]
 ...
 [2 0 2 ... 1 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
In [ ]:
```