Project01_Report

1. Problem1

Given an image I, consider all valid pairs of neighboring pixels, compute the difference between their intensity or color values, and plot the histogram in order to visualizing and checking the smoothness prior.

1.1. Neighbors

In this project, I have considered five types of neighbors as follows.

1.1.1. The horizontal type

This type of neighbor is (x, y) and $(x, y+1)$.

1.1.2. The vertical type

This type of neighbor is (x, y) and $(x+1, y)$.

1.1.3. The 4-type

This type of neighbor is (x, y) whose neighbors are $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, and $(x, y+1)$. I will consider special cases on the border while writing Python codes.

1.1.4. The D-type

This type of neighbor is (x, y) whose neighbors are $(x-1, y-1)$, $(x-1, y+1)$, $(x+1, y-1)$, and $(x+1, y+1)$. I will consider special cases on the border while writing Python codes.

1.1.5. The 8-type

This type of neighbor is (x, y) whose neighbors are $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, $(x, y+1)$ $(x-1, y-1)$, $(x-1, y+1)$, $(x+1, y-1)$, and $(x+1, y+1)$. I will consider special cases on the border while writing Python codes.

1.2. Difference

1.2.1. The horizontal type

The Python code calculating the squared of differences of the horizontal type is **running for 13 seconds** and is shown as follows:

```
def hor_p(arr_):
    m,n,j=len(arr_),len(arr_.T),0
    data=[0]*(m*n)
    for i in range(0,m):
        for p in range(0,n-1):
            data[j]=int(arr_[i,p+1]-arr_[i,p])
            j+=1
    for i in range(0,len(data)):
```

```
        data[i]=data[i]*data[i]
    return data
```

1.2.2. The vertical type

The Python code calculating the squared of differences of the vertical type is **running for 13 seconds** and is shown as follows:

```
def ver_p(arr_):
    m,n,j=len(arr_),len(arr_.T),0
    data=[0]*(m*n)
    for p in range(0,n):
        for i in range(0,m-1):
            data[j]=int(arr_[i+1,p]-arr_[i,p])
            j+=1
    for i in range(0,len(data)):
        data[i]=data[i]*data[i]
    return data
```

1.2.3. The 4-type

The Python code calculating the squared of differences of the 4-type is **running for 42.5 seconds** and is shown as follows:

```
def n4_p(arr_):
    m,n=len(arr_),len(arr_.T)
    data=[0]*(m*n)
    # On the border
    data[0],data[1]=round((arr_[0,1]-
2*arr_[0,0]+arr_[1,0])/2),round((arr_[0,n-2]-2*arr_[0,n-1]+arr_[1,n-1])/2)
    data[2],data[3]=round((arr_[m-2,0]-2*arr_[m-1,0]+arr_[m-
1,1])/2),round((arr_[m-1,n-2]-2*arr_[m-1,n-1]+arr_[m-2,n-1])/2)
    j=4
    for i in range(1,m-1):
        data[j]=round((arr_[i-1,0]-3*arr_[i,0]+arr_[i,1]+arr_[i+1,0])/3)
        data[j+1]=round((arr_[i-1,n-1]-3*arr_[i,n-1]+arr_[i,n-2]+arr_[i+1,n-
1])/3)
        j+=2
    for i in range(1,n-1):
        data[j]=round((arr_[0,i-1]-3*arr_[0,i]+arr_[1,i]+arr_[0,i+1])/3)
        data[j+1]=round((arr_[m-1,i-1]-3*arr_[m-1,i]+arr_[m-2,i]+arr_[m-
1,i+1])/3)
        j+=2
    # Off the border
    for i in range(1,m-1):
        for p in range(1,n-1):
            data[j]=round((arr_[i,p-1]+arr_[i+1,p]+arr_[i-1,p]+arr_[i,p+1]-
4*arr_[i,p])/4)
            j+=1
```

```
    for i in range(0,len(data)):
        data[i]=data[i]*data[i]
    return data
```

1.2.4. The D-type

The Python code calculating the squared of differences of the D-type is **running for 43.2 seconds** and is shown as follows:

```
def nd_p(arr_):
    m,n=len(arr_),len(arr_.T)
    data=[0]*(m*n)
    # On the border
    data[0],data[1],data[2],data[3]=arr_[1,1]-arr_[0,0],arr_[1,n-2]-
arr_[0,n-1],arr_[m-2,1]-arr_[m-1,0],arr_[m-2,n-2]-arr_[m-1,n-1]
    j=4
    for i in range(1,m-1):
        data[j]=round((arr_[i-1,1]-2*arr_[i,0]+arr_[i+1,1])/2)
        data[j+1]=round((arr_[i-1,n-2]-2*arr_[i,n-1]+arr_[i+1,n-2])/2)
        j+=2
    for i in range(1,n-1):
        data[j]=round((arr_[1,i-1]-2*arr_[0,i]+arr_[1,i+1])/2)
        data[j+1]=round((arr_[m-2,i-1]-2*arr_[m-1,i]+arr_[m-2,i+1])/2)
        j+=2
    # off the border
    for i in range(1,m-1):
        for p in range(1,n-1):
            data[j]=round((arr_[i-1,p-1]+arr_[i-1,p+1]+arr_[i+1,p-
1]+arr_[i+1,p+1]-4*arr_[i,p])/4)
            j+=1
    for i in range(0,len(data)):
        data[i]=data[i]*data[i]
    return data
```

1.2.5. The 8-type

The Python code calculating the squared of differences of the 8-type is **running for 56 seconds** and is shown as follows:

```
def n8_p(arr_):
    m,n=len(arr_),len(arr_.T)
    data=[0]*(m*n)
    # On the border
    data[0]=round((arr_[0,1]+arr_[1,0]+arr_[1,1]-3*arr_[0,0])/3)
    data[1]=round((arr_[0,n-2]+arr_[1,n-2]+arr_[1,n-1]-3*arr_[0,n-1])/3)
    data[2]=round((arr_[m-2,0]+arr_[m-2,1]+arr_[m-1,1]-3*arr_[m-1,0])/3)
    data[3]=round((arr_[m-2,n-1]+arr_[m-2,n-2]+arr_[m-1,n-2]-3*arr_[m-1,n-
1])/3)
    j=4
```

```
    for i in range(1,m-1):
        data[j]=round((arr_[i-1,0]+arr_[i-
1,1]+arr_[i,1]+arr_[i+1,1]+arr_[i+1,0]-5*arr_[i,0])/5)
        data[j+1]=round((arr_[i-1,n-1]+arr_[i-1,n-2]+arr_[i,n-2]+arr_[i+1,n-
2]+arr_[i+1,n-1]-5*arr_[i,n-1])/5)
        j+=2
    for i in range(1,n-1):
        data[j]=round((arr_[0,i-1]+arr_[1,i-
1]+arr_[1,i]+arr_[1,i+1]+arr_[0,i+1]-5*arr_[0,i])/5)
        data[j+1]=round((arr_[m-1,i-1]+arr_[m-2,i-1]+arr_[m-2,i]+arr_[m-
2,i+1]+arr_[m-1,i+1]-5*arr_[m-1,i])/5)
        j+=2
    # off the border
    for i in range(1,m-1):
        for p in range(1,n-1):
            data[j]=round((arr_[i-1,p-1]+arr_[i-1,p]+arr_[i-1,p+1]+arr_[i,p-
1]+arr_[i,p+1]+
                          arr_[i+1,p-1]+arr_[i+1,p]+arr_[i+1,p+1]-
8*arr_[i,p])/8)
            j+=1
    for i in range(0,len(data)):
        data[i]=data[i]*data[i]
    return data
```

1.3. Histogram

1.3.1. The horizontal type

The Python code, visualizing histograms of the horizontal-type neighbor for 4 types of differences (intensity, RGB, HSV, and Lab), is shown as follows:

```
def plot_histogram_hor(arr1,arr2):      # Horizontal
    data_hor,mark_4=arr1,['horizontal']
    data_hor_mean=round(np.mean(data_hor))
    data_hor_sigma=round(np.std(data_hor,ddof=1))
    num_bins=100
    fig,ax=plt.subplots()

n_4,bins_4,patches_4=ax.hist(data_hor,num_bins,range=[0,8000],edgecolor='bl
ack',facecolor='green',histtype='bar',density=True)
    ax.set_xlabel('Differences')
    ax.set_ylabel('Frequency')
    ax.set_title(r'Histogram of Squared Differences: $\mu=%d$,
$\sigma=%d$' %(data_hor_mean,data_hor_sigma))
    fig.savefig('p_%s.png' %(''.join(arr2+mark_4)))
```

After running my code, I can acquire histograms of 4 types of differences (intensity, RGB, HSV, and Lab) respectively.

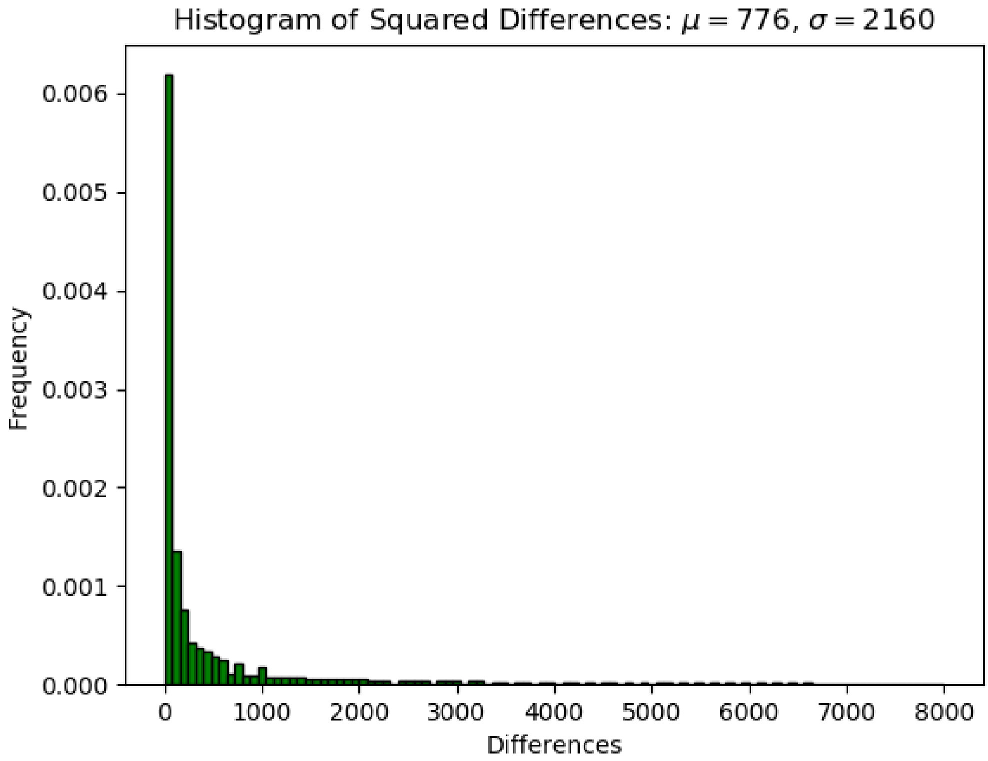

Fig. 1. The histogram of squared horizontal-type differences of intensity

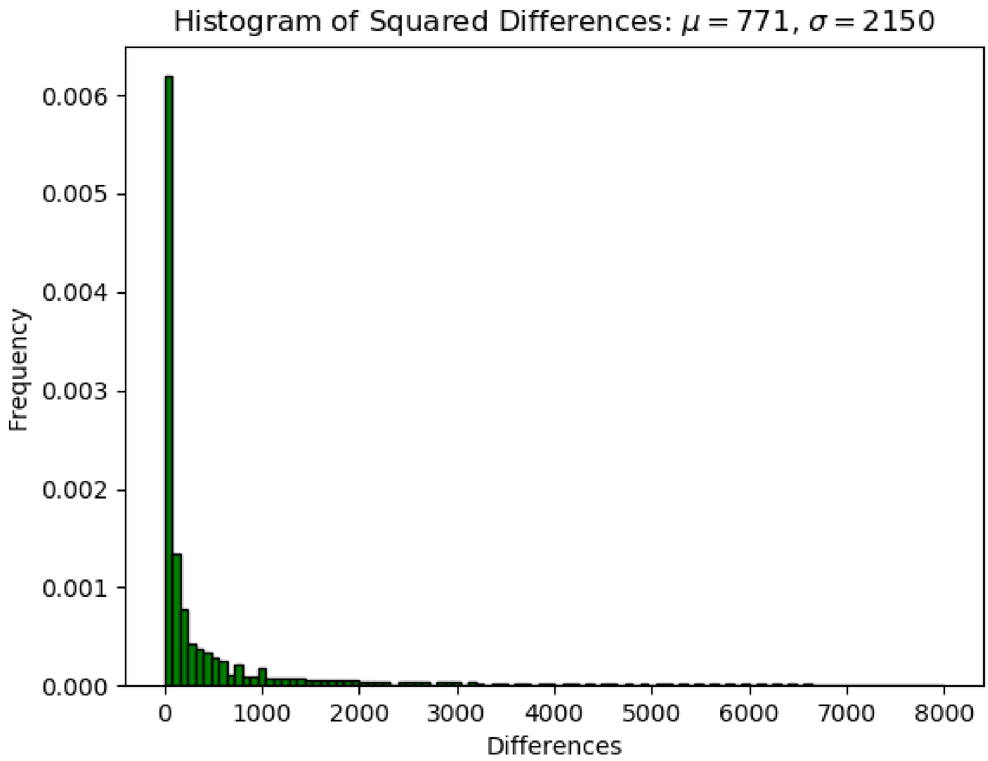

Fig. 2. The histogram of squared horizontal-type differences of RGB

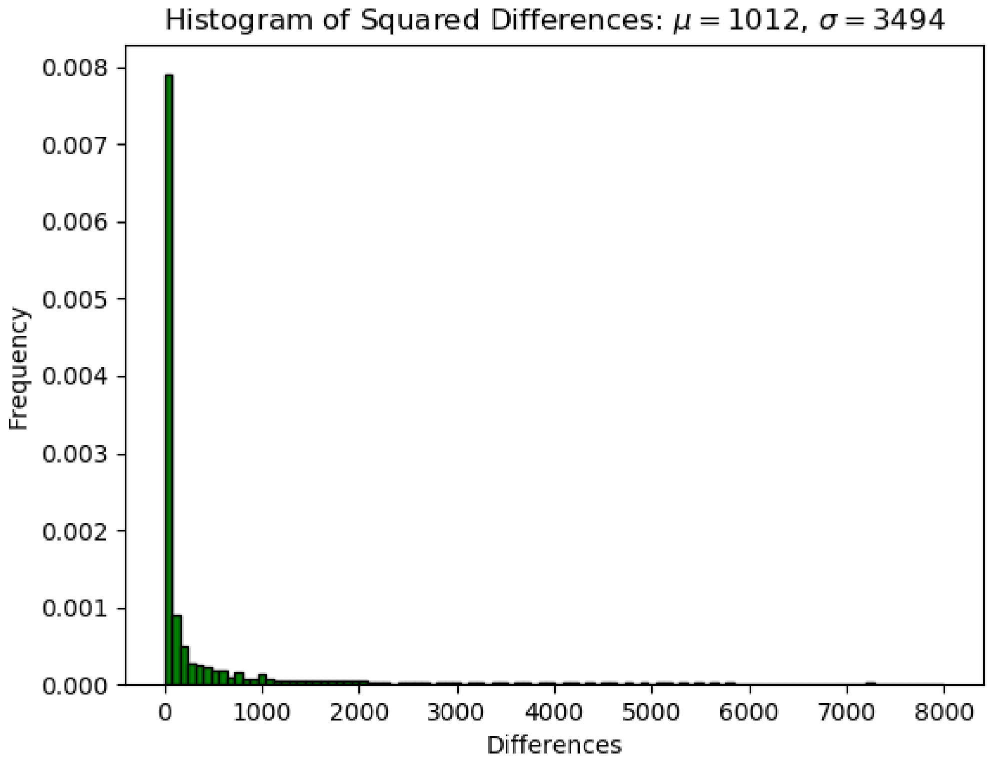

Fig. 3. The histogram of squared horizontal-type differences of HSV

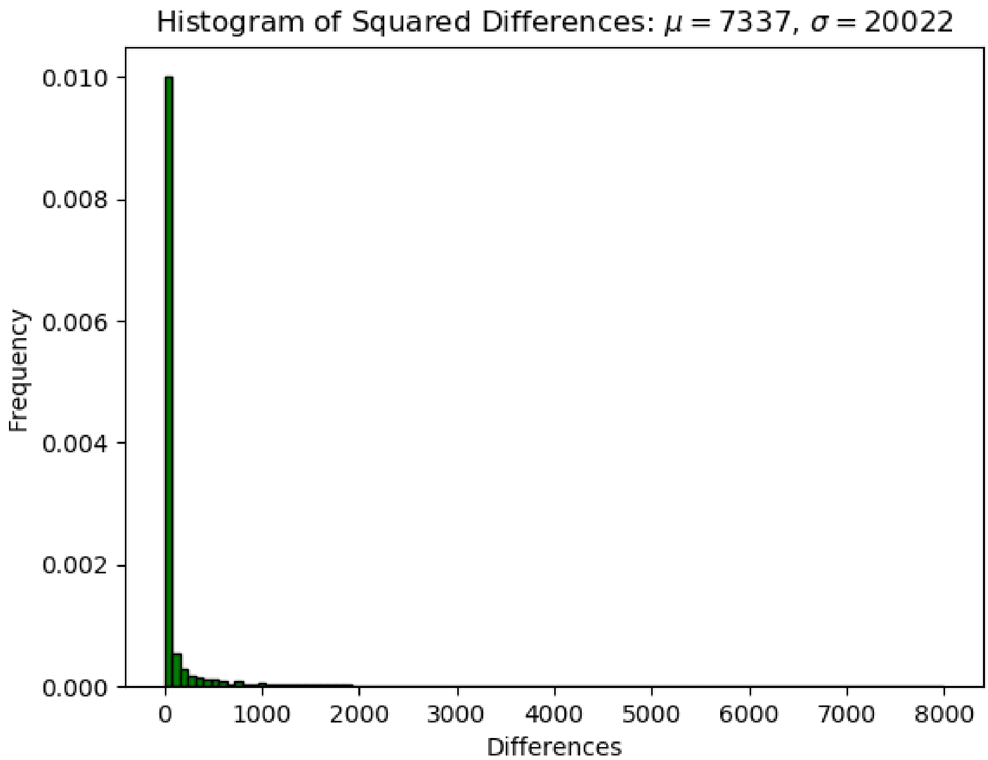

Fig. 4. The histogram of squared horizontal-type differences of Lab

1.3.2. The vertical type

The Python code, visualizing histograms of the vertical-type neighbor for 4 types of differences (intensity, RGB, HSV, and Lab), is shown as follows:

```
def plot_histogram_ver(arr1,arr2):      # Vertical
    data_ver,mark_4=arr1,['vertical']
    data_ver_mean=round(np.mean(data_ver))
    data_ver_sigma=round(np.std(data_ver,ddof=1))
    num_bins=100
    fig,ax=plt.subplots()

n_4,bins_4,patches_4=ax.hist(data_ver,num_bins,range=[0,8000],edgecolor='bl
ack',facecolor='green',histtype='bar',density=True)
    ax.set_xlabel('Differences')
    ax.set_ylabel('Frequency')
    ax.set_title(r'Histogram of Squared Differences: $\mu=%d$,
$\sigma=%d$' %(data_ver_mean,data_ver_sigma))
    fig.savefig('p_%s.png' %(''.join(arr2+mark_4)))
```

After running my code, I can acquire histograms of 4 types of differences (intensity, RGB, HSV, and Lab) respectively.

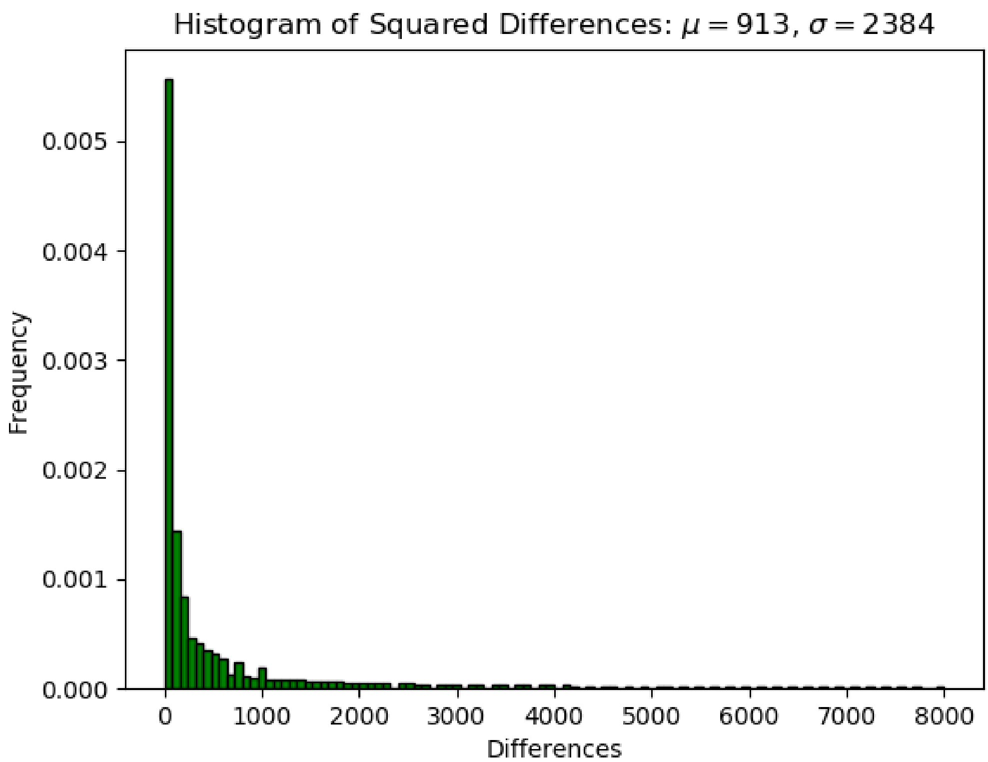

Fig. 5. The histogram of squared vertical-type differences of intensity

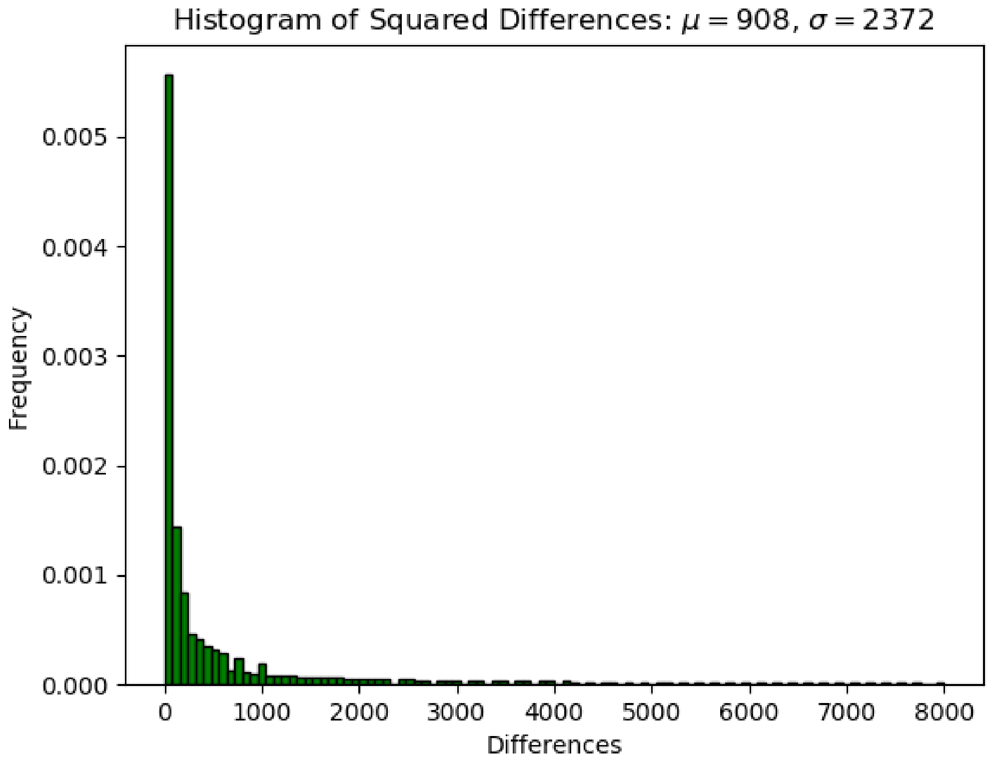

Fig. 6. The histogram of squared vertical-type differences of RGB

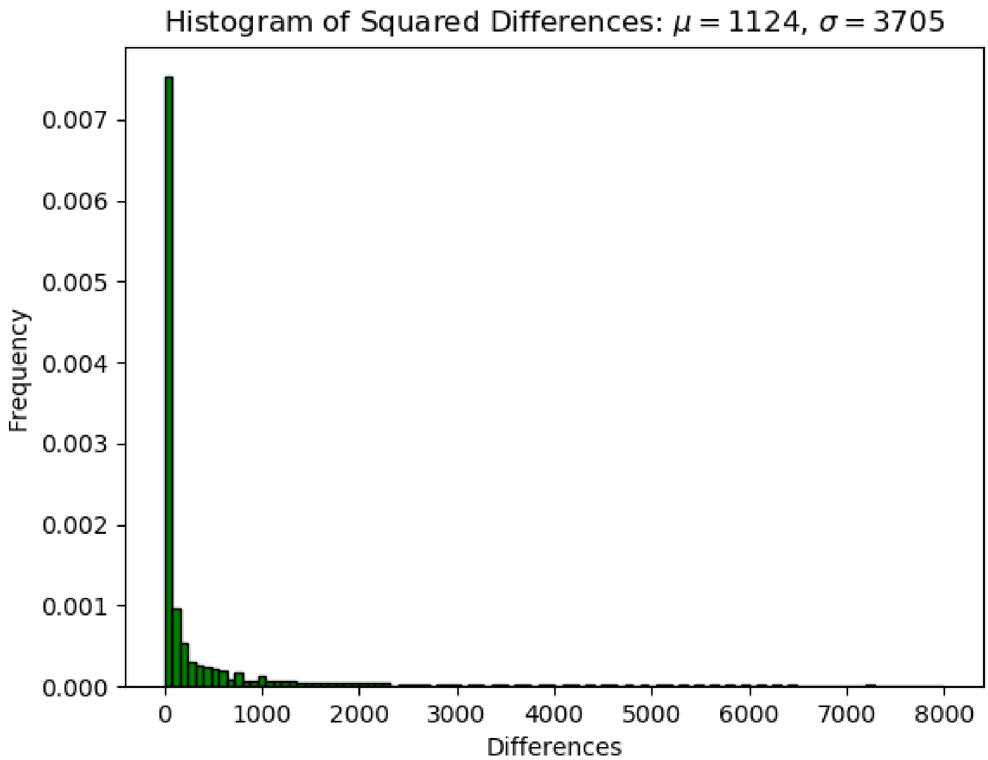

Fig. 7. The histogram of squared vertical-type differences of HSV

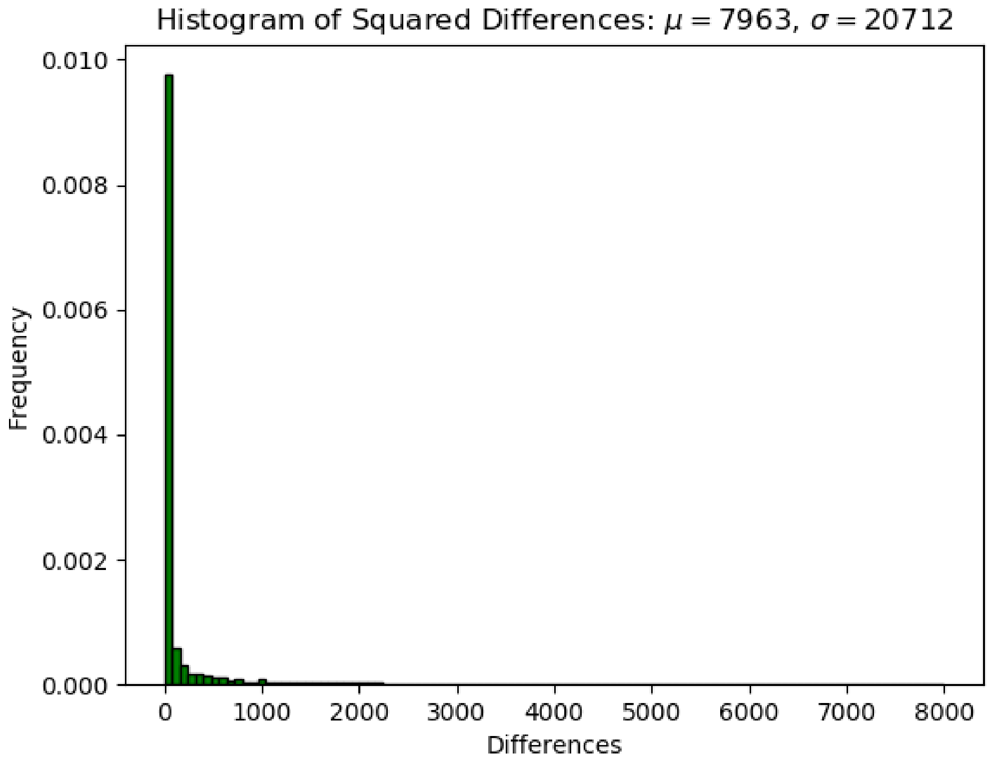

Fig. 8. The histogram of squared vertical-type differences of Lab

1.3.3. The 4-type

The Python code, visualizing histograms of the 4-type neighbor for 4 types of differences (intensity, RGB, HSV, and Lab), is shown as follows:

```
def plot_histogram_4(arr1,arr2):        # 4-Neighbors
    data_4,mark_4=arr1,['four']
    data_4_mean=round(np.mean(data_4))
    data_4_sigma=round(np.std(data_4,ddof=1))
    num_bins=100
    fig,ax=plt.subplots()

n_4,bins_4,patches_4=ax.hist(data_4,num_bins,range=[0,8000],edgecolor='blac
k',facecolor='green',histtype='bar',density=True)
    ax.set_xlabel('Differences')
    ax.set_ylabel('Frequency')
    ax.set_title(r'Histogram of Squared Differences: $\mu=%d$,
$\sigma=%d$' %(data_4_mean,data_4_sigma))
    fig.savefig('p_%s.png' %(''.join(arr2+mark_4)))
```

After running my code, I can acquire histograms of 4 types of differences (intensity, RGB, HSV, and Lab) respectively.

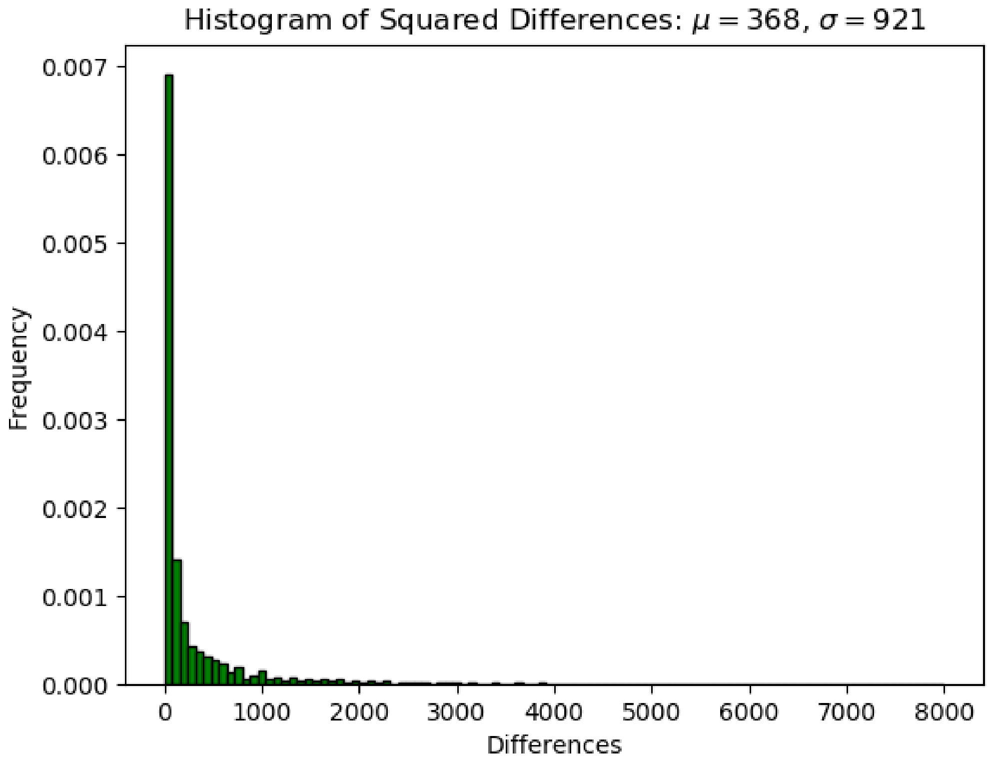

Fig. 9. The histogram of squared 4-type differences of intensity

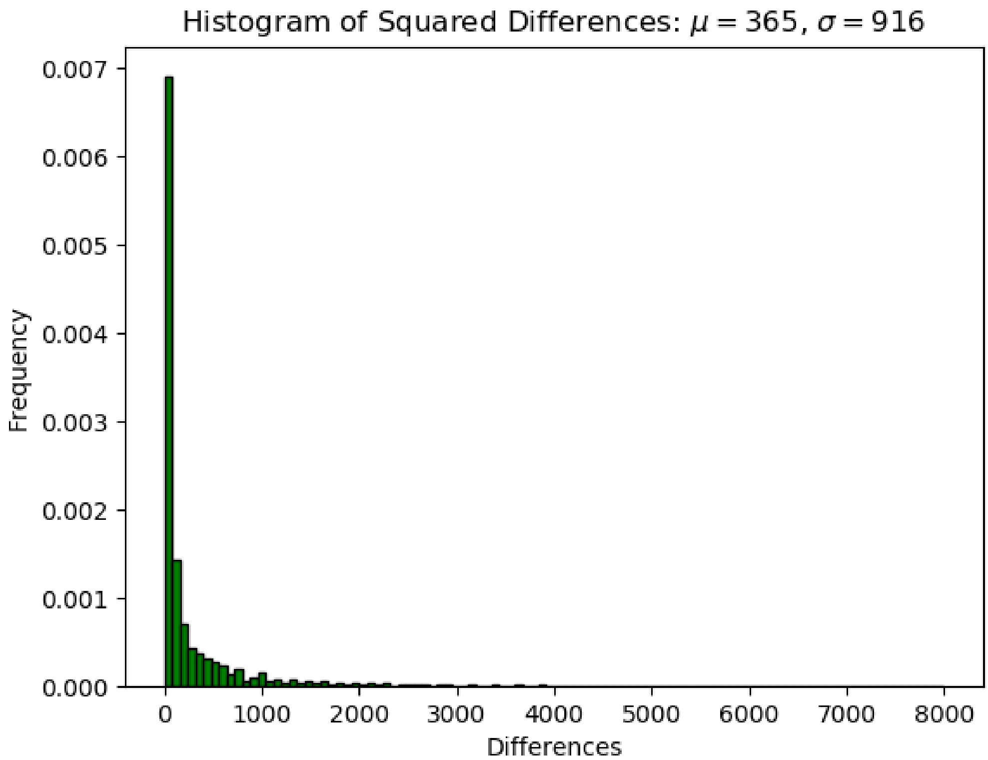

Fig. 10. The histogram of squared 4-type differences of RGB

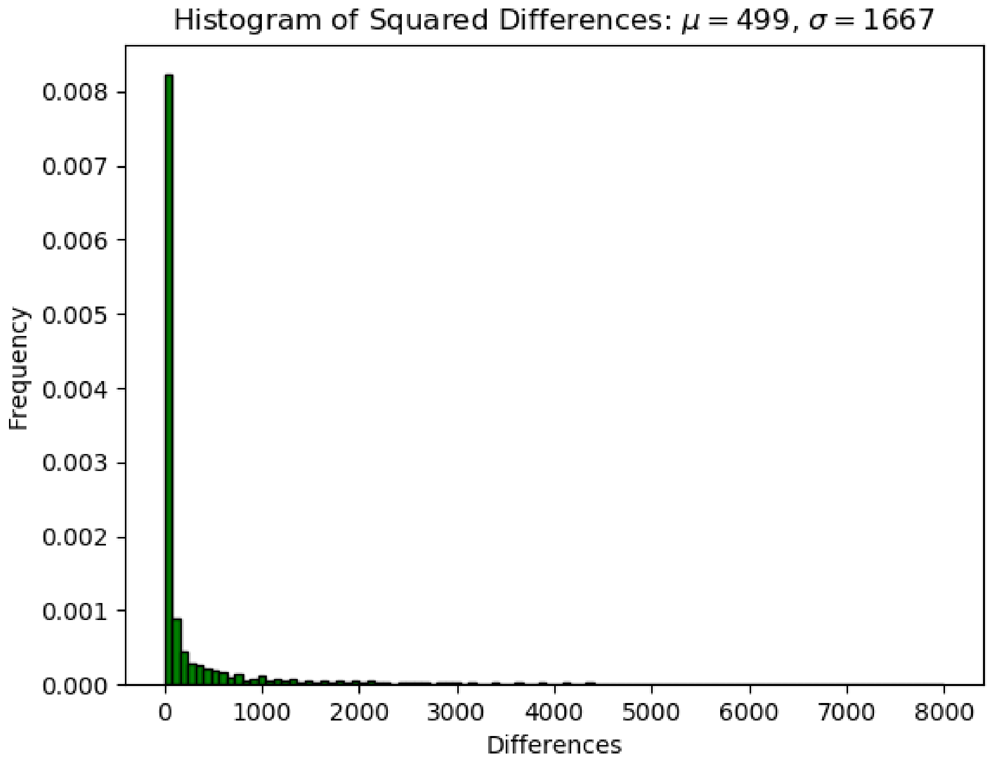

Fig. 11. The histogram of squared 4-type differences of HSV

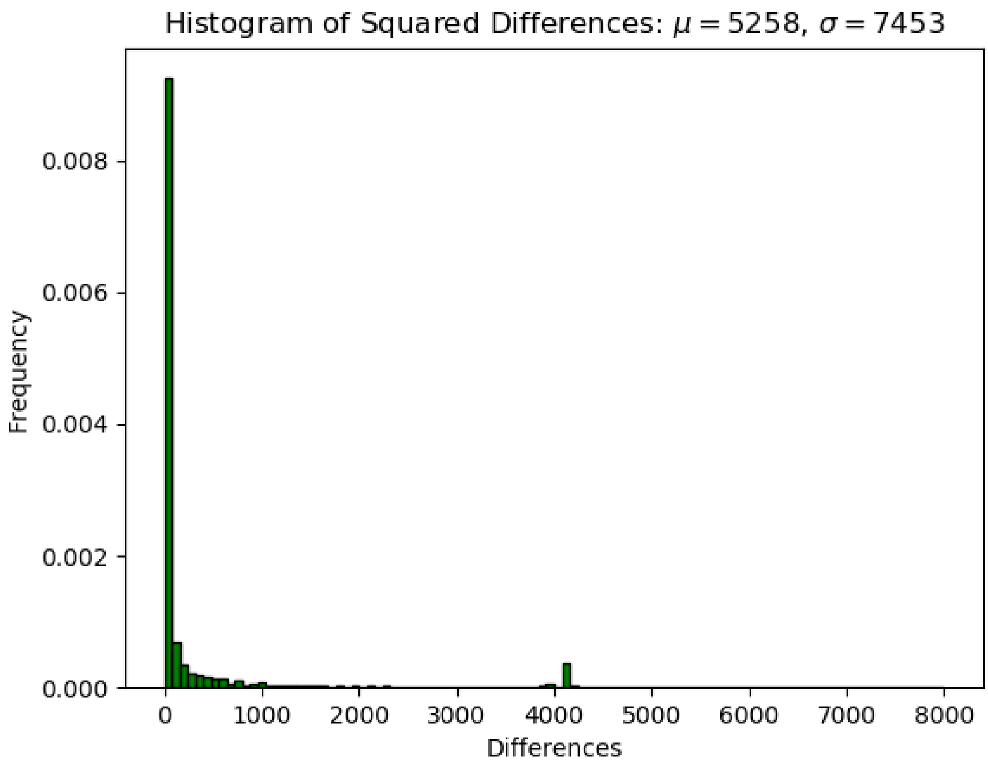

Fig. 12. The histogram of squared 4-type differences of Lab

1.3.4. The D-type

The Python code, visualizing histograms of the D-type neighbor for 4 types of differences (intensity, RGB, HSV, and Lab), is shown as follows:

```
def plot_histogram_d(arr1,arr2):        # Diagonal-Neighbors
    data_d,mark_d=arr1,['diagonal']
    data_d_mean=round(np.mean(data_d))
    data_d_sigma=round(np.std(data_d,ddof=1))
    num_bins=100
    fig,ax=plt.subplots()

n_d,bins_d,patches__d=ax.hist(data_d,num_bins,range=[0,8000],edgecolor='bla
ck',facecolor='green',histtype='bar',density=True)
    ax.set_xlabel('Differences')
    ax.set_ylabel('Frequency')
    ax.set_title(r'Histogram of Squared Differences: $\mu=%d$,
$\sigma=%d$' %(data_d_mean,data_d_sigma))
    fig.savefig('p_%s.png' %(''.join(arr2+mark_d)))
```

After running my code, I can acquire histograms of 4 types of differences (intensity, RGB, HSV, and Lab) respectively.

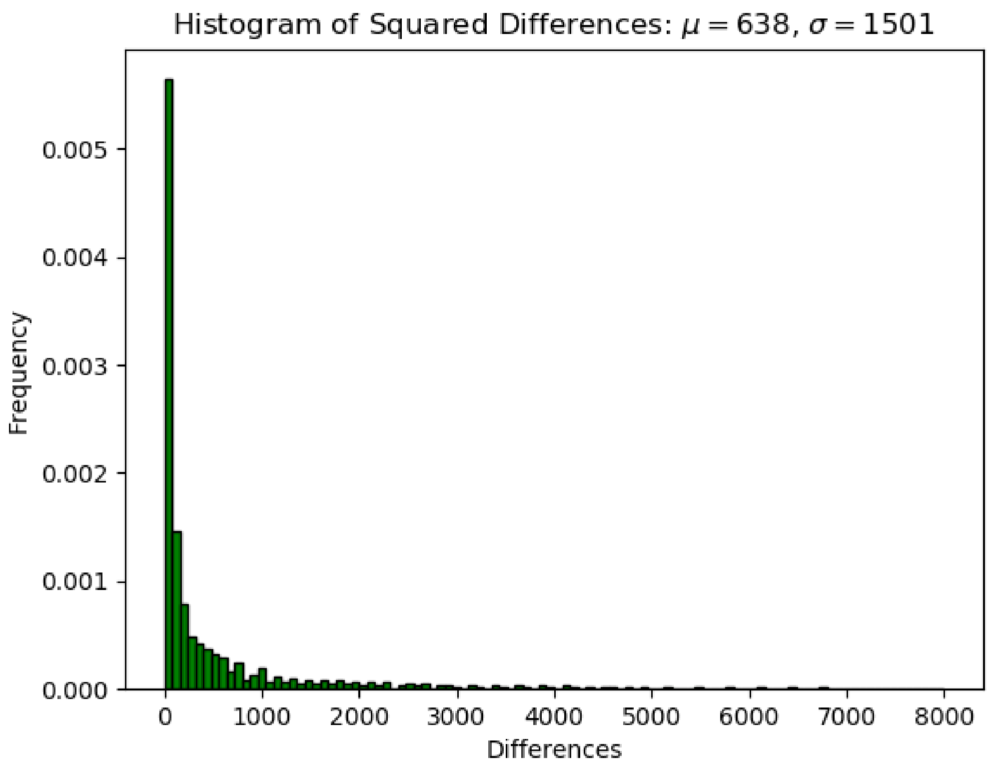

Fig. 13. The histogram of squared D-type differences of intensity

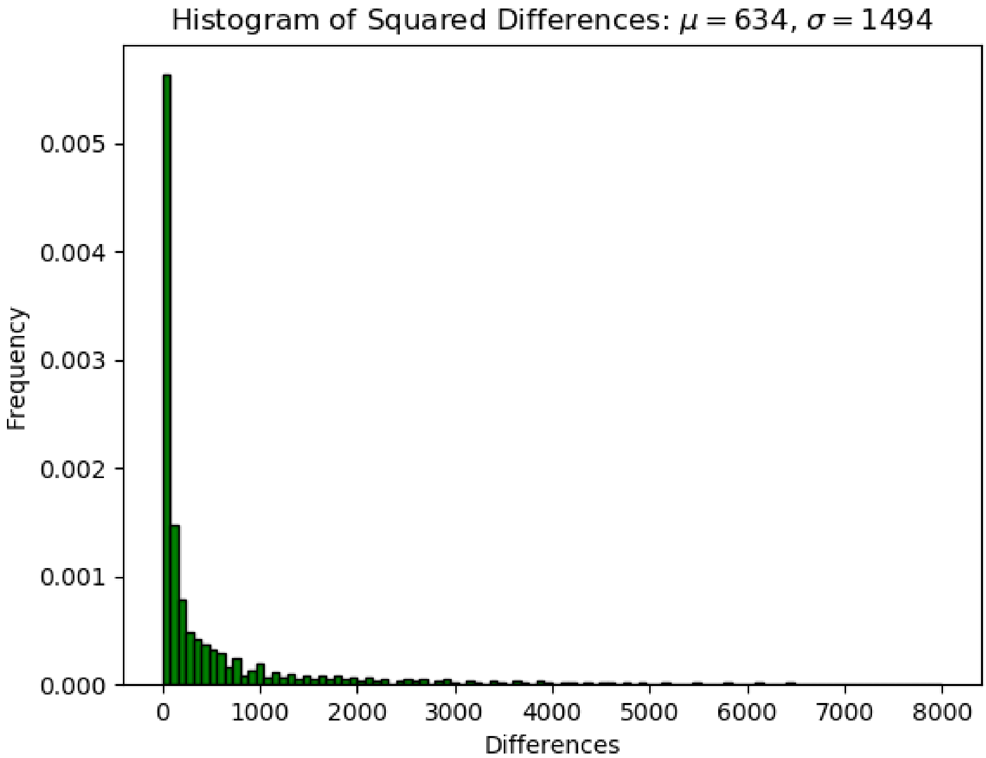

Fig. 14. The histogram of squared D-type differences of RGB

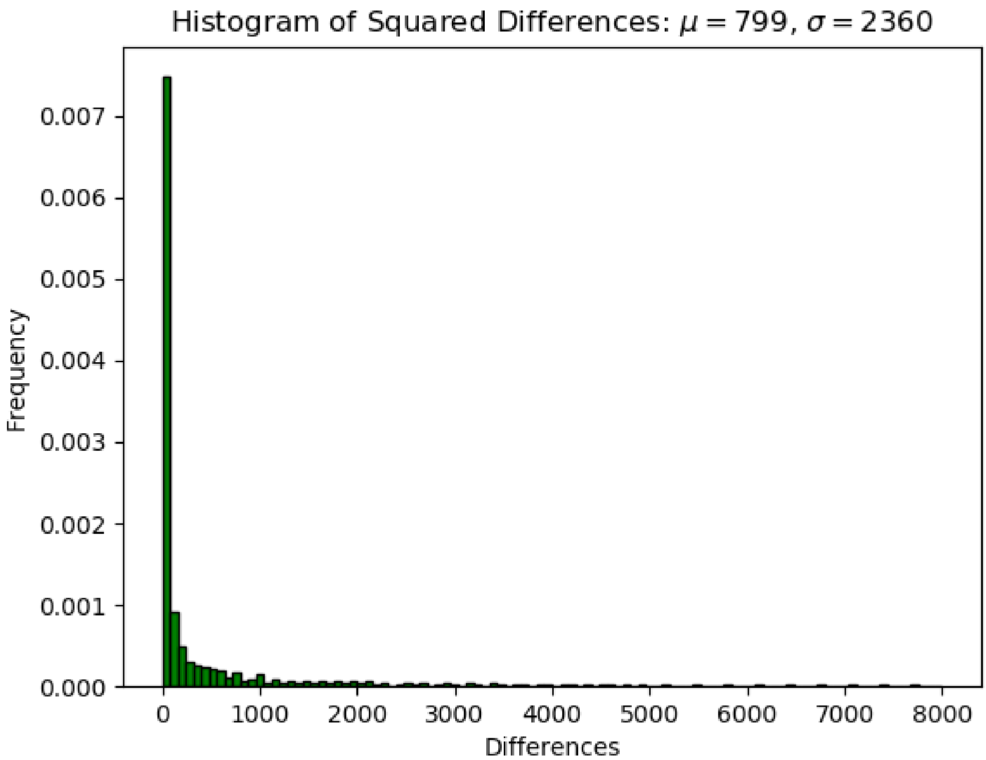

Fig. 15. The histogram of squared D-type differences of HSV

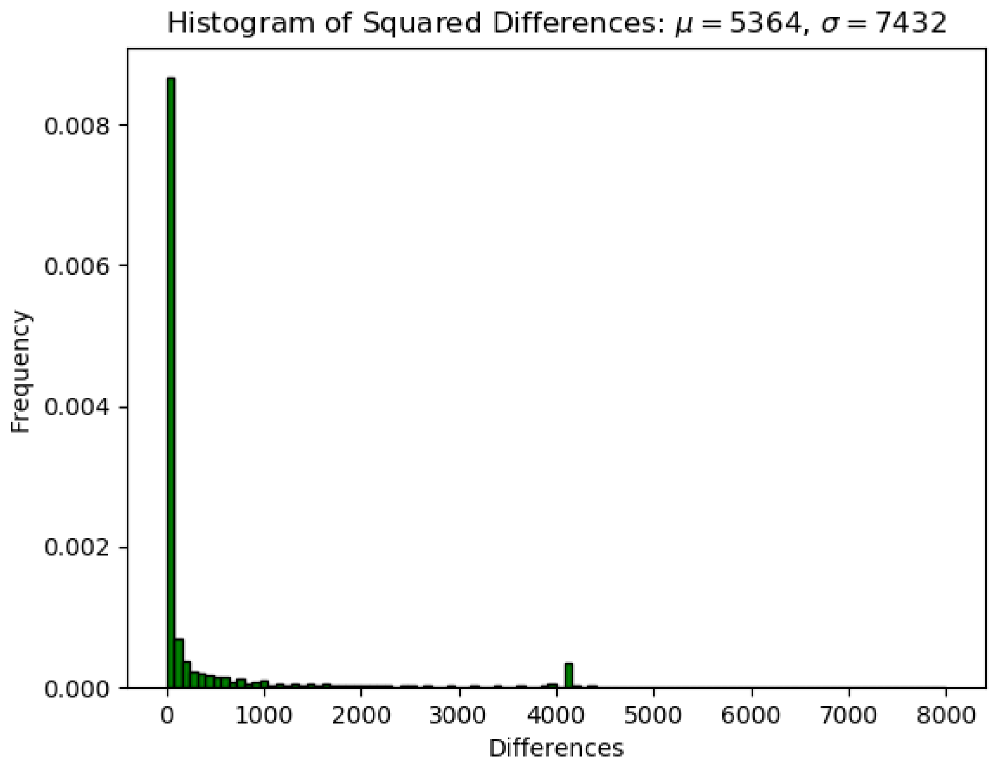

Fig. 16. The histogram of squared D-type differences of Lab

1.3.5. The 8-type

The Python code, visualizing histograms of the horizontal-type neighbor for 4 types of differences (intensity, RGB, HSV, and Lab), is shown as follows:

```
def plot_histogram_8(arr1,arr2):        # 8-Neighbors
    data_8,mark_8=arr1,['eight']
    data_8_mean=round(np.mean(data_8))
    data_8_sigma=round(np.std(data_8,ddof=1))
    num_bins=100
    fig,ax=plt.subplots()

n_8,bins_8,patches_8=ax.hist(data_8,num_bins,range=[0,8000],edgecolor='blac
k',facecolor='green',histtype='bar',density=True)
    ax.set_xlabel('Differences')
    ax.set_ylabel('Frequency')
    ax.set_title(r'Histogram of Squared Differences: $\mu=%d$,
$\sigma=%d$' %(data_8_mean,data_8_sigma))
    fig.savefig('p_%s.png' %(''.join(arr2+mark_8)))
```

After running my code, I can acquire histograms of 4 types of differences (intensity, RGB, HSV, and Lab) respectively.

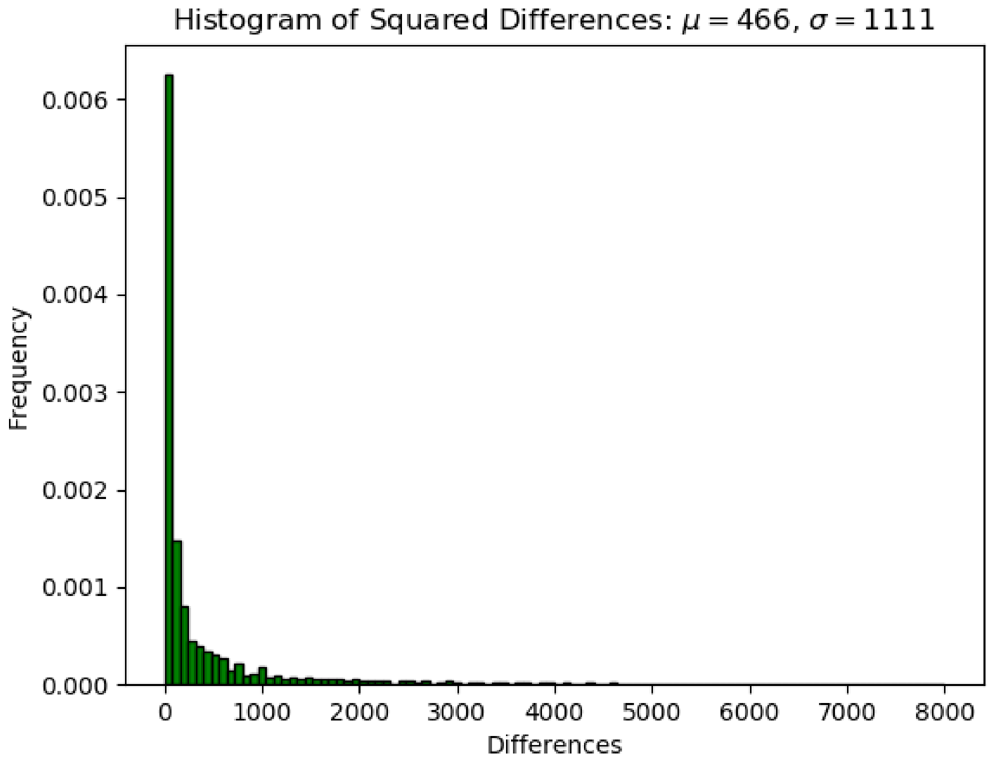

Fig. 17. The histogram of squared 8-type differences of intensity

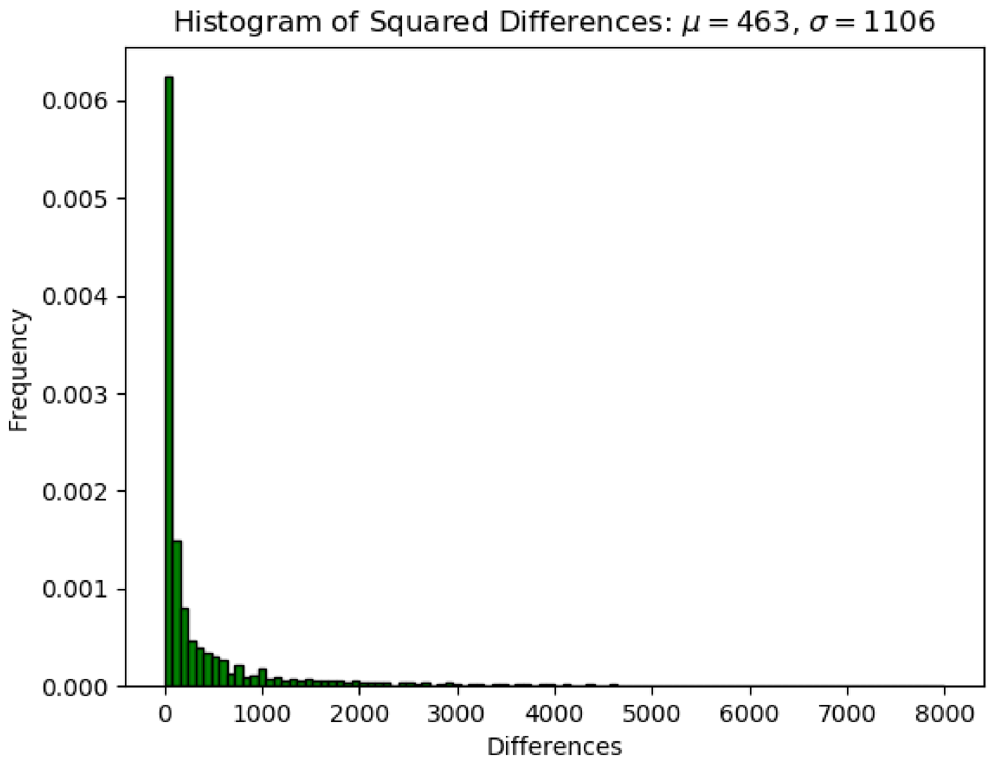

Fig. 18. The histogram of squared 8-type differences of RGB

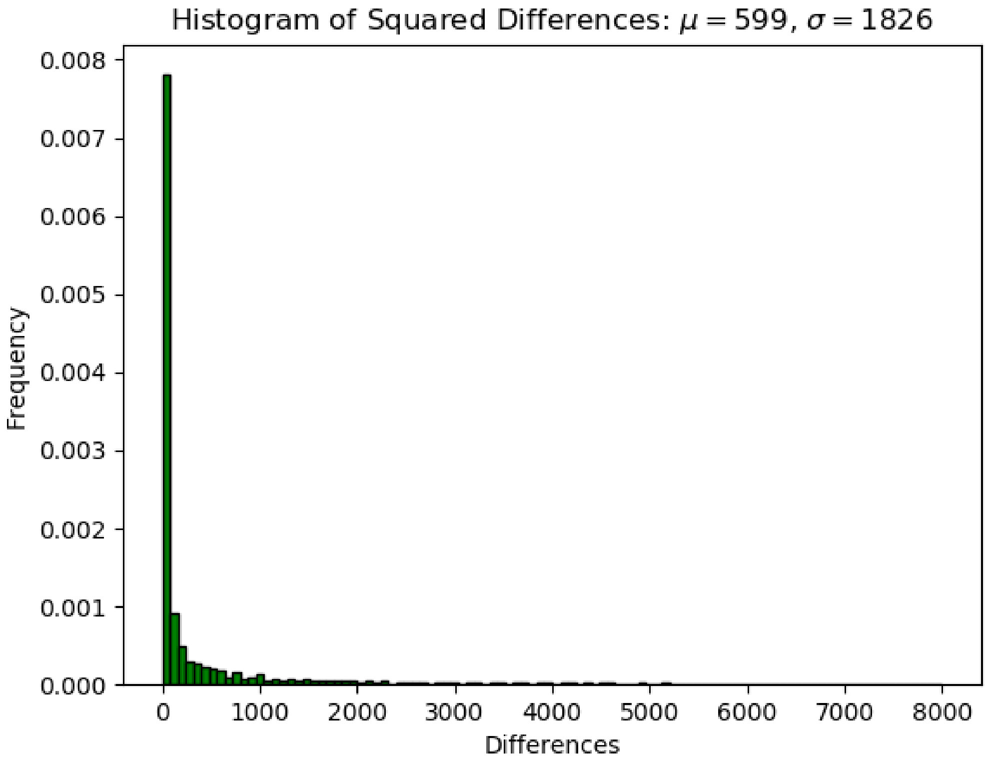

Fig. 19. The histogram of squared 8-type differences of HSV

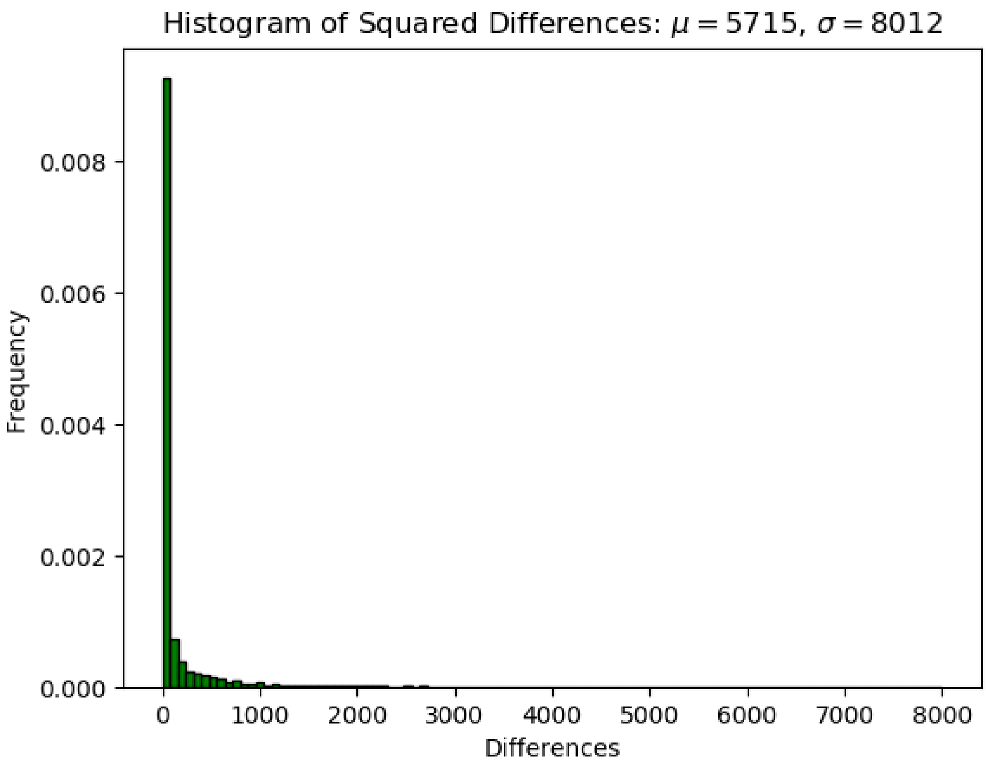

Fig. 20. The histogram of squared 8-type differences of Lab

2. Problem2

Given an image segment, try to find the shortest path.

2.1. Main codes

2.1.1. The 4-path

The Python code computing the lengths of the shortest 4-path between p and q is shown as follows:

```
def find_4_path(im_seg,arr_begin,arr_end,v):
    m,n=len(im_seg),len(im_seg.T)
    if m<=2 and n<=2:
        print("You don't have to use the function!")
        return True
    graph={}
    # On the border
    graph[(im_seg[0,0],0,0)]=[]
    if im_seg[0,1] in v: graph[im_seg[0,0],0,0].append([im_seg[0,1],0,1])
    if im_seg[1,0] in v: graph[im_seg[0,0],0,0].append([im_seg[1,0],1,0])
    graph[(im_seg[m-1,0],m-1,0)]=[]
    if im_seg[m-1,1] in v: graph[im_seg[m-1,0],m-1,0].append([im_seg[m-
1,1],m-1,1])
    if im_seg[m-2,0] in v: graph[im_seg[m-1,0],m-1,0].append([im_seg[m-
2,0],m-2,0])
    graph[(im_seg[0,n-1],0,n-1)]=[]
    if im_seg[0,n-2] in v: graph[im_seg[0,n-1],0,n-1].append([im_seg[0,n-
2],0,n-2])
    if im_seg[1,n-1] in v: graph[im_seg[0,n-1],0,n-1].append([im_seg[1,n-
1],1,n-1])
    graph[(im_seg[m-1,n-1],m-1,n-1)]=[]
    if im_seg[m-2,n-1] in v: graph[im_seg[m-1,n-1],m-1,n-
1].append([im_seg[m-2,n-1],m-2,n-1])
    if im_seg[m-1,n-2] in v: graph[im_seg[m-1,n-1],m-1,n-
1].append([im_seg[m-1,n-2],m-1,n-2])

    if m>2:
        for i in range(1,m-1):
            graph[(im_seg[i,0],i,0)]=[]
            if im_seg[i-1,0] in v: graph[im_seg[i,0],i,0].append([im_seg[i-
1,0],i-1,0])
            if im_seg[i+1,0] in v:
graph[im_seg[i,0],i,0].append([im_seg[i+1,0],i+1,0])
            if im_seg[i,1]   in v:
graph[im_seg[i,0],i,0].append([im_seg[i,1],i,1])
            graph[(im_seg[i,n-1],i,n-1)]=[]
            if im_seg[i-1,n-1] in v: graph[im_seg[i,n-1],i,n-
1].append([im_seg[i-1,n-1],i-1,n-1])
            if im_seg[i+1,n-1] in v: graph[im_seg[i,n-1],i,n-
```

```
1].append([im_seg[i+1,n-1],i+1,n-1])
            if im_seg[i,n-2]   in v: graph[im_seg[i,n-1],i,n-
1].append([im_seg[i,n-2],i,n-2])
    if n>2:
        for j in range(1,n-1):
            graph[(im_seg[0,j],0,j)]=[]
            if im_seg[0,j-1] in v:
graph[im_seg[0,j],0,j].append([im_seg[0,j-1],0,j-1])
            if im_seg[0,j+1] in v:
graph[im_seg[0,j],0,j].append([im_seg[0,j+1],0,j+1])
            if im_seg[1,j]   in v:
graph[im_seg[0,j],0,j].append([im_seg[1,j],1,j])
            graph[(im_seg[m-1,j],m-1,j)]=[]
            if im_seg[m-1,j-1] in v: graph[im_seg[m-1,j],m-
1,j].append([im_seg[m-1,j-1],m-1,j-1])
            if im_seg[m-1,j+1] in v: graph[im_seg[m-1,j],m-
1,j].append([im_seg[m-1,j+1],m-1,j+1])
            if im_seg[m-2,j]   in v: graph[im_seg[m-1,j],m-
1,j].append([im_seg[m-2,j],m-2,j])
    # Off the border
    if m>2 and n>2:
        for i in range(1,m-1):
            for j in range(1,n-1):
                graph[(im_seg[i,j],i,j)]=[]
                if im_seg[i-1,j] in v:
graph[im_seg[i,j],i,j].append([im_seg[i-1,j],i-1,j])
                if im_seg[i+1,j] in v:
graph[im_seg[i,j],i,j].append([im_seg[i+1,j],i+1,j])
                if im_seg[i,j-1] in v:
graph[im_seg[i,j],i,j].append([im_seg[i,j-1],i,j-1])
                if im_seg[i,j+1] in v:
graph[im_seg[i,j],i,j].append([im_seg[i,j+1],i,j+1])

    m_begin,n_begin=arr_begin[0],arr_begin[1]
    pre_pixel={}
    search_queue=deque()
    search_queue+=[[im_seg[m_begin,n_begin],m_begin,n_begin]]
    search_queue+=graph[im_seg[m_begin,n_begin],m_begin,n_begin]
    searched1,searched2=[],[[im_seg[m_begin,n_begin],m_begin,n_begin]]

    for element in graph[im_seg[m_begin,n_begin],m_begin,n_begin]:
        element=tuple([element[1],element[2]])
        pre_pixel[element]=[]
        pre_pixel[element].append([m_begin,n_begin])
```

```
    while search_queue:
        pixel_next=search_queue.popleft()
        if pixel_next not in searched1:
            if pixel_next[1]==arr_end[0] and pixel_next[2]==arr_end[1]:
                key_list,value_list=[],[]
                for key,value in pre_pixel.items():
                    key_list.append(list(key))
                    value_list.append(list(value[0]))
                order=[]
                index0=0
                for index in range(0,len(key_list)):
                    if key_list[index]==list(q):
                        order.append(key_list[index])
                        index0=index
                order=enum_(p,key_list,value_list,order,index0)
                new_order=[]
                for i in range(0,len(order)):
                    new_order.append(order.pop())
                print("The 4-type path is: ",new_order)
                print("The length of the shortest 4-type path is:
",len(new_order)-1)
                return True
            for index,element in
enumerate(graph[pixel_next[0],pixel_next[1],pixel_next[2]]):
                if element in searched2:

graph[pixel_next[0],pixel_next[1],pixel_next[2]].pop(index)
            for element1 in
graph[pixel_next[0],pixel_next[1],pixel_next[2]]:
                if element1 not in searched2:
                    element1=tuple([element1[1],element1[2]])
                    pre_pixel[element1]=[]
                    pre_pixel[element1].append([pixel_next[1],pixel_next[2]])
            for element2 in
graph[pixel_next[0],pixel_next[1],pixel_next[2]]:
                searched2.append([element2[0],element2[1],element2[2]])
            search_queue+=graph[pixel_next[0],pixel_next[1],pixel_next[2]]
            searched1.append(pixel_next)

        else: continue
    print("Sorry! The particular path doesn't exist.")
    return False
```

2.1.2. The 8-path

The Python code computing the lengths of the shortest 4-path between p and q is shown as follows:

```
def find_8_path(im_seg,arr_begin,arr_end,v):
    m,n=len(im_seg),len(im_seg.T)
    if m<=2 and n<=2:
        print("You don't have to use the function!")
        return True
    graph={}
    # On the border
    graph[(im_seg[0,0],0,0)]=[]
    if im_seg[0,1] in v: graph[im_seg[0,0],0,0].append([im_seg[0,1],0,1])
    if im_seg[1,0] in v: graph[im_seg[0,0],0,0].append([im_seg[1,0],1,0])
    if im_seg[1,1] in v: graph[im_seg[0,0],0,0].append([im_seg[1,1],1,1])
    graph[(im_seg[m-1,0],m-1,0)]=[]
    if im_seg[m-1,1] in v: graph[im_seg[m-1,0],m-1,0].append([im_seg[m-
1,1],m-1,1])
    if im_seg[m-2,0] in v: graph[im_seg[m-1,0],m-1,0].append([im_seg[m-
2,0],m-2,0])
    if im_seg[m-2,1] in v: graph[im_seg[m-1,0],m-1,0].append([im_seg[m-
2,1],m-2,1])
    graph[(im_seg[0,n-1],0,n-1)]=[]
    if im_seg[0,n-2] in v: graph[im_seg[0,n-1],0,n-1].append([im_seg[0,n-
2],0,n-2])
    if im_seg[1,n-1] in v: graph[im_seg[0,n-1],0,n-1].append([im_seg[1,n-
1],1,n-1])
    if im_seg[1,n-2] in v: graph[im_seg[0,n-1],0,n-1].append([im_seg[1,n-
2],1,n-2])
    graph[(im_seg[m-1,n-1],m-1,n-1)]=[]
    if im_seg[m-2,n-1] in v: graph[im_seg[m-1,n-1],m-1,n-
1].append([im_seg[m-2,n-1],m-2,n-1])
    if im_seg[m-1,n-2] in v: graph[im_seg[m-1,n-1],m-1,n-
1].append([im_seg[m-1,n-2],m-1,n-2])
    if im_seg[m-2,n-2] in v: graph[im_seg[m-1,n-1],m-1,n-
1].append([im_seg[m-2,n-2],m-2,n-2])

    if m>2:
        for i in range(1,m-1):
            graph[(im_seg[i,0],i,0)]=[]
            if im_seg[i-1,0] in v: graph[im_seg[i,0],i,0].append([im_seg[i-
1,0],i-1,0])
            if im_seg[i+1,0] in v:
graph[im_seg[i,0],i,0].append([im_seg[i+1,0],i+1,0])
            if im_seg[i,1]   in v:
graph[im_seg[i,0],i,0].append([im_seg[i,1],i,1])
```

```
            if im_seg[i-1,1] in v: graph[im_seg[i,0],i,0].append([im_seg[i-1,1],i-1,1])
            if im_seg[i+1,1] in v: graph[im_seg[i,0],i,0].append([im_seg[i+1,1],i+1,1])
            graph[(im_seg[i,n-1],i,n-1)]=[]
            if im_seg[i-1,n-1] in v: graph[im_seg[i,n-1],i,n-1].append([im_seg[i-1,n-1],i-1,n-1])
            if im_seg[i+1,n-1] in v: graph[im_seg[i,n-1],i,n-1].append([im_seg[i+1,n-1],i+1,n-1])
            if im_seg[i,n-2]   in v: graph[im_seg[i,n-1],i,n-1].append([im_seg[i,n-2],i,n-2])
            if im_seg[i-1,n-2] in v: graph[im_seg[i,n-1],i,n-1].append([im_seg[i-1,n-2],i-1,n-2])
            if im_seg[i+1,n-2] in v: graph[im_seg[i,n-1],i,n-1].append([im_seg[i+1,n-2],i+1,n-2])
    if n>2:
        for j in range(1,n-1):
            graph[(im_seg[0,j],0,j)]=[]
            if im_seg[0,j-1] in v: graph[im_seg[0,j],0,j].append([im_seg[0,j-1],0,j-1])
            if im_seg[0,j+1] in v: graph[im_seg[0,j],0,j].append([im_seg[0,j+1],0,j+1])
            if im_seg[1,j]   in v: graph[im_seg[0,j],0,j].append([im_seg[1,j],1,j])
            if im_seg[1,j-1] in v: graph[im_seg[0,j],0,j].append([im_seg[1,j-1],1,j-1])
            if im_seg[1,j+1] in v: graph[im_seg[0,j],0,j].append([im_seg[1,j+1],1,j+1])
            graph[(im_seg[m-1,j],m-1,j)]=[]
            if im_seg[m-1,j-1] in v: graph[im_seg[m-1,j],m-1,j].append([im_seg[m-1,j-1],m-1,j-1])
            if im_seg[m-1,j+1] in v: graph[im_seg[m-1,j],m-1,j].append([im_seg[m-1,j+1],m-1,j+1])
            if im_seg[m-2,j]   in v: graph[im_seg[m-1,j],m-1,j].append([im_seg[m-2,j],m-2,j])
            if im_seg[m-2,j-1] in v: graph[im_seg[m-1,j],m-1,j].append([im_seg[m-2,j-1],m-2,j-1])
            if im_seg[m-2,j+1] in v: graph[im_seg[m-1,j],m-1,j].append([im_seg[m-2,j+1],m-2,j+1])
    # Off the border
    if m>2 and n>2:
        for i in range(1,m-1):
            for j in range(1,n-1):
                graph[(im_seg[i,j],i,j)]=[]
```

```
                if im_seg[i-1,j]   in v:
graph[im_seg[i,j],i,j].append([im_seg[i-1,j],i-1,j])
                if im_seg[i+1,j]   in v:
graph[im_seg[i,j],i,j].append([im_seg[i+1,j],i+1,j])
                if im_seg[i,j-1]   in v:
graph[im_seg[i,j],i,j].append([im_seg[i,j-1],i,j-1])
                if im_seg[i,j+1]   in v:
graph[im_seg[i,j],i,j].append([im_seg[i,j+1],i,j+1])
                if im_seg[i-1,j-1] in v:
graph[im_seg[i,j],i,j].append([im_seg[i-1,j-1],i-1,j-1])
                if im_seg[i+1,j-1] in v:
graph[im_seg[i,j],i,j].append([im_seg[i+1,j-1],i+1,j-1])
                if im_seg[i-1,j+1] in v:
graph[im_seg[i,j],i,j].append([im_seg[i-1,j+1],i-1,j+1])
                if im_seg[i+1,j+1] in v:
graph[im_seg[i,j],i,j].append([im_seg[i+1,j+1],i+1,j+1])

    m_begin,n_begin=arr_begin[0],arr_begin[1]
    pre_pixel={}
    search_queue=deque()
    search_queue+=[[im_seg[m_begin,n_begin],m_begin,n_begin]]
    search_queue+=graph[im_seg[m_begin,n_begin],m_begin,n_begin]
    searched1,searched2=[],[[im_seg[m_begin,n_begin],m_begin,n_begin]]

    for element in graph[im_seg[m_begin,n_begin],m_begin,n_begin]:
        element=tuple([element[1],element[2]])
        pre_pixel[element]=[]
        pre_pixel[element].append([m_begin,n_begin])

    while search_queue:
        # searched.append(pixel_next)
        pixel_next=search_queue.popleft()
        if pixel_next not in searched1:
            if pixel_next[1]==arr_end[0] and pixel_next[2]==arr_end[1]:
                key_list,value_list=[],[]
                for key,value in pre_pixel.items():
                    key_list.append(list(key))
                    value_list.append(list(value[0]))
                order=[]
                index0=0
                for index in range(0,len(key_list)):
                    if key_list[index]==list(q):
                        order.append(key_list[index])
                        index0=index
```

```
                order=enum_(p,key_list,value_list,order,index0)
                new_order=[]
                for i in range(0,len(order)):
                    new_order.append(order.pop())
                print("The 8-type path is: ",new_order)
                print("The length of the shortest 8-type path is:
",len(new_order)-1)
                return True

            for index,element in
enumerate(graph[pixel_next[0],pixel_next[1],pixel_next[2]]):
                if element in searched2:

graph[pixel_next[0],pixel_next[1],pixel_next[2]].pop(index)
                for element1 in
graph[pixel_next[0],pixel_next[1],pixel_next[2]]:
                    if element1 not in searched2:
                        element1=tuple([element1[1],element1[2]])
                        pre_pixel[element1]=[]

pre_pixel[element1].append([pixel_next[1],pixel_next[2]])
                for element2 in
graph[pixel_next[0],pixel_next[1],pixel_next[2]]:
                    searched2.append([element2[0],element2[1],element2[2]])

search_queue+=graph[pixel_next[0],pixel_next[1],pixel_next[2]]
                searched1.append(pixel_next)

        else: continue
    print("Sorry! The particular path doesn't exist.")
    return False
```

2.1.3. The m-path

The Python code computing the lengths of the shortest 4-path between p and q is shown as follows:

```
def find_m_path(im_seg,arr_begin,arr_end,v):
    m,n=len(im_seg),len(im_seg.T)
    if m<=2 and n<=2:
        print("You don't have to use the function!")
        return True
    graph={}
    # On the border
    graph[(im_seg[0,0],0,0)]=[]
    if im_seg[0,1] in v:
        graph[im_seg[0,0],0,0].append([im_seg[0,1],0,1])
    if im_seg[1,0] in v:
```

```
        graph[im_seg[0,0],0,0].append([im_seg[1,0],1,0])
    if im_seg[0,1] not in v and im_seg[1,0] not in v and im_seg[1,1] in v:
        graph[im_seg[0,0],0,0].append([im_seg[1,1],1,1])
    graph[(im_seg[m-1,0],m-1,0)]=[]
    if im_seg[m-1,1] in v:
        graph[im_seg[m-1,0],m-1,0].append([im_seg[m-1,1],m-1,1])
    if im_seg[m-2,0] in v:
        graph[im_seg[m-1,0],m-1,0].append([im_seg[m-2,0],m-2,0])
    if im_seg[m-1,1] not in v and im_seg[m-2,0] not in v and im_seg[m-2,1]
in v:
        graph[im_seg[m-1,0],m-1,0].append([im_seg[m-2,1],m-2,1])
    graph[(im_seg[0,n-1],0,n-1)]=[]
    if im_seg[0,n-2] in v:
        graph[im_seg[0,n-1],0,n-1].append([im_seg[0,n-2],0,n-2])
    if im_seg[1,n-1] in v:
        graph[im_seg[0,n-1],0,n-1].append([im_seg[1,n-1],1,n-1])
    if im_seg[0,n-2] not in v and im_seg[1,n-1] not in v and im_seg[1,n-2]
in v:
        graph[im_seg[0,n-1],0,n-1].append([im_seg[1,n-2],1,n-2])
    graph[(im_seg[m-1,n-1],m-1,n-1)]=[]
    if im_seg[m-2,n-1] in v:
        graph[im_seg[m-1,n-1],m-1,n-1].append([im_seg[m-2,n-1],m-2,n-1])
    if im_seg[m-1,n-2] in v:
        graph[im_seg[m-1,n-1],m-1,n-1].append([im_seg[m-1,n-2],m-1,n-2])
    if im_seg[m-2,n-1] not in v and im_seg[n-1,m-2] not in v and im_seg[m-
2,n-2] in v:
        graph[im_seg[m-1,n-1],m-1,n-1].append([im_seg[m-2,n-2],m-2,n-2])

    if m>2:
        for i in range(1,m-1):
            # on the left
            graph[(im_seg[i,0],i,0)]=[]
            if im_seg[i-1,0] in v: graph[im_seg[i,0],i,0].append([im_seg[i-
1,0],i-1,0])
            if im_seg[i+1,0] in v:
graph[im_seg[i,0],i,0].append([im_seg[i+1,0],i+1,0])
            if im_seg[i,1]   in v:
graph[im_seg[i,0],i,0].append([im_seg[i,1],i,1])
            if im_seg[i-1,0] not in v and im_seg[i+1,0] not in v and
im_seg[i,1] not in v:
                if im_seg[i-1,1] in v:
graph[im_seg[i,0],i,0].append([im_seg[i-1,1],i-1,1])
                if im_seg[i+1,1] in v:
graph[im_seg[i,0],i,0].append([im_seg[i+1,1],i+1,1])
```

```
            # on the right
            graph[(im_seg[i,n-1],i,n-1)]=[]
            if im_seg[i-1,n-1] in v: graph[im_seg[i,n-1],i,n-
1].append([im_seg[i-1,n-1],i-1,n-1])
            if im_seg[i+1,n-1] in v: graph[im_seg[i,n-1],i,n-
1].append([im_seg[i+1,n-1],i+1,n-1])
            if im_seg[i,n-2]   in v: graph[im_seg[i,n-1],i,n-
1].append([im_seg[i,n-2],i,n-2])
            if im_seg[i-1,n-1] not in v and im_seg[i+1,n-1] not in v and
im_seg[i,n-2] not in v:
                if im_seg[i-1,n-2] in v:
graph[im_seg[i,0],i,0].append([im_seg[i-1,n-2],i-1,n-2])
                if im_seg[i+1,n-2] in v:
graph[im_seg[i,0],i,0].append([im_seg[i+1,n-2],i+1,n-2])
    if n>2:
        for j in range(1,n-1):
            # on the top
            graph[(im_seg[0,j],0,j)]=[]
            if im_seg[0,j-1] in v:
graph[im_seg[0,j],0,j].append([im_seg[0,j-1],0,j-1])
            if im_seg[0,j+1] in v:
graph[im_seg[0,j],0,j].append([im_seg[0,j+1],0,j+1])
            if im_seg[1,j]   in v:
graph[im_seg[0,j],0,j].append([im_seg[1,j],1,j])
            if im_seg[0,j-1] not in v and im_seg[0,j+1] not in v and
im_seg[1,j] not in v:
                if im_seg[1,j-1] in v:
graph[im_seg[i,0],i,0].append([im_seg[1,j-1],1,j-1])
                if im_seg[1,j+1] in v:
graph[im_seg[i,0],i,0].append([im_seg[1,j+1],1,j+1])
            # on the underneath
            graph[(im_seg[m-1,j],m-1,j)]=[]
            if im_seg[m-1,j-1] in v: graph[im_seg[m-1,j],m-
1,j].append([im_seg[m-1,j-1],m-1,j-1])
            if im_seg[m-1,j+1] in v: graph[im_seg[m-1,j],m-
1,j].append([im_seg[m-1,j+1],m-1,j+1])
            if im_seg[m-2,j]   in v: graph[im_seg[m-1,j],m-
1,j].append([im_seg[m-2,j],m-2,j])
            if im_seg[0,j-1] not in v and im_seg[0,j+1] not in v and
im_seg[1,j] not in v:
                if im_seg[m-2,j-1] in v:
graph[im_seg[i,0],i,0].append([im_seg[m-2,j-1],m-2,j-1])
                if im_seg[m-2,j+1] in v:
graph[im_seg[i,0],i,0].append([im_seg[m-2,j+1],m-2,j+1])
```

```
    # Off the border
    if m>2 and n>2:
        for i in range(1,m-1):
            for j in range(1,n-1):
                graph[(im_seg[i,j],i,j)]=[]
                if im_seg[i-1,j] in v:
graph[im_seg[i,j],i,j].append([im_seg[i-1,j],i-1,j])
                if im_seg[i+1,j] in v:
graph[im_seg[i,j],i,j].append([im_seg[i+1,j],i+1,j])
                if im_seg[i,j-1] in v:
graph[im_seg[i,j],i,j].append([im_seg[i,j-1],i,j-1])
                if im_seg[i,j+1] in v:
graph[im_seg[i,j],i,j].append([im_seg[i,j+1],i,j+1])
                if im_seg[i-1,j] not in v and im_seg[i+1,j] not in v and
im_seg[i,j-1] not in v and im_seg[i,j+1] not in v:
                    if im_seg[i-1,j-1] in v:
graph[im_seg[i,0],i,0].append([im_seg[i-1,j-1],i-1,j-1])
                    if im_seg[i-1,j+1] in v:
graph[im_seg[i,0],i,0].append([im_seg[i-1,j+1],i-1,j+1])
                    if im_seg[i+1,j-1] in v:
graph[im_seg[i,0],i,0].append([im_seg[i+1,j-1],i+1,j-1])
                    if im_seg[i+1,j+1] in v:
graph[im_seg[i,0],i,0].append([im_seg[i+1,j+1],i+1,j+1])

    m_begin,n_begin=arr_begin[0],arr_begin[1]
    pre_pixel={}
    search_queue=deque()
    search_queue+=[[im_seg[m_begin,n_begin],m_begin,n_begin]]
    search_queue+=graph[im_seg[m_begin,n_begin],m_begin,n_begin]
    searched1,searched2=[],[[im_seg[m_begin,n_begin],m_begin,n_begin]]
    for element in graph[im_seg[m_begin,n_begin],m_begin,n_begin]:
        element=tuple([element[1],element[2]])
        pre_pixel[element]=[]
        pre_pixel[element].append([m_begin,n_begin])

    while search_queue:
        pixel_next=search_queue.popleft()
        if pixel_next not in searched1:
            if pixel_next[1]==arr_end[0] and pixel_next[2]==arr_end[1]:
                key_list,value_list=[],[]
                for key,value in pre_pixel.items():
                    key_list.append(list(key))
                    value_list.append(list(value[0]))
                order=[]
```

```
                index0=0
                for index in range(0,len(key_list)):
                    if key_list[index]==list(q):
                        order.append(key_list[index])
                        index0=index
                order=enum_(p,key_list,value_list,order,index0)
                new_order=[]
                for i in range(0,len(order)):
                    new_order.append(order.pop())
                print("The m-type path is: ",new_order)
                print("The length of the shortest m-type path is:
",len(new_order)-1)
                return True
            for index,element in
enumerate(graph[pixel_next[0],pixel_next[1],pixel_next[2]]):
                if element in searched2:

graph[pixel_next[0],pixel_next[1],pixel_next[2]].pop(index)
            for element1 in
graph[pixel_next[0],pixel_next[1],pixel_next[2]]:
                if element1 not in searched2:
                    element1=tuple([element1[1],element1[2]])
                    pre_pixel[element1]=[]
                    pre_pixel[element1].append([pixel_next[1],pixel_next[2]])
            for element2 in
graph[pixel_next[0],pixel_next[1],pixel_next[2]]:
                searched2.append([element2[0],element2[1],element2[2]])
            search_queue+=graph[pixel_next[0],pixel_next[1],pixel_next[2]]
            searched1.append(pixel_next)

        else: continue
    print("Sorry! The particular path doesn't exist.")
    return False
```

2.2 The input argument

An image segment matrix, a predefined set V, two pixel locations p and q inside the image, the path type (4-, 8-, or m-path). The Python code is as follows:

```
m=int(input('Please set the number of row of your image: \n'))
n=int(input('Please set the number of column of your image: \n'))
A=[]
for i in range(0,m):
   A.append([])
   for j in range(0,n):

      A[i].append(int(input('Please input pixel values: \n')))
```

```
p=[]
for i in range(0,2):
      p.append(int(input('Please set your inial point: \n')))
q=[]
for i in range(0,2):
      q.append(int(input('Please set your final point: \n')))
i=int(input('Please set the length of  your predefined V: \n'))
V=[]
for i in range(0,i):
   V.append(int(input('Please set your V: \n')))
path_type=input('Please set your path type: \n')

A=np.array(A)
p,q=tuple(p),tuple(q)
if path_type=='4':
    find_4_path(A,p,q,V)
elif path_type=='8':
    find_8_path(A,p,q,V)
elif path_type=='m':
    find_m_path(A,p,q,V)
else:
    print("Your input is incorrect!")
```

2.3. Implement the function

2.3.1. Problem (a) and (b)

The image segment matrix is

$$\begin{bmatrix} 3 & 1 & 2 & 1 \\ 2 & 2 & 0 & 2 \\ 1 & 2 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix}.$$

The initial point p is (3, 0) and the ending point is q (0, 3).

(a) When V is {0, 1}:

The Python code of 4-type runs for 5.5 seconds.

```
Sorry! The particular path doesn't exist.

Process finished with exit code 0
```

The Python code of 8-type runs for 2.9 seconds.

```
The 8-type path is:  [[3, 0], [3, 1], [2, 2], [1, 2], [0, 3]
The length of the shortest 8-type path is:  4

Process finished with exit code 0
```

The Python code of m-type runs for 2.7 seconds.

```
The m-type path is:  [[3, 0], [3, 1], [3, 2], [2, 2], [1, 2], [0, 3]]
The length of the shortest m-type path is:  5

Process finished with exit code 0
```

(b) When V is {1, 2}:

The Python code of 4-type runs for 2.4 seconds.

```
The 4-type path is:  [[3, 0], [2, 0], [1, 0], [1, 1], [0, 1], [0, 2], [0, 3]]
The length of the shortest 4-type path is:  6

Process finished with exit code 0
```

The Python code of 8-type runs for 2.5 seconds.

```
The 8-type path is:  [[3, 0], [2, 0], [1, 1], [0, 2], [0, 3]]
The length of the shortest 8-type path is:  4

Process finished with exit code 0
```

The Python code of m-type runs for 2 seconds.

```
The m-type path is:  [[3, 0], [2, 0], [1, 0], [1, 1], [0, 1], [0, 2], [0, 3]]
The length of the shortest m-type path is:  6

Process finished with exit code 0
```

(c) When V is {0, 1, 2}:

The Python code of 4-type runs for 2 seconds.

```
The 4-type path is:  [[3, 0], [3, 1], [3, 2], [3, 3], [2, 3], [1, 3], [0, 3]]
The length of the shortest 4-type path is:  6

Process finished with exit code 0
```

The Python code of 8-type runs for 2.5 seconds.

```
The 8-type path is:  [[3, 0], [2, 1], [1, 2], [0, 3]]
The length of the shortest 8-type path is:  3

Process finished with exit code 0
```

The Python code of m-type runs for 2 seconds.

```
The m-type path is:  [[3, 0], [3, 1], [3, 2], [3, 3], [2, 3], [1, 3], [0, 3]]
The length of the shortest m-type path is:  6

Process finished with exit code 0
```

2.3.2. More examples

The image segment matrix is

$$\begin{bmatrix} 3 & 1 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 \\ 1 & 2 & 2 & 0 & 1 \\ 0 & 3 & 2 & 1 & 2 \\ 1 & 2 & 1 & 1 & 2 \end{bmatrix}.$$

(a) When V is {1, 2}, the initial point p is (4, 0) and the ending point is q (0, 4).

```
The 4-type path is:  [[4, 0], [4, 1], [4, 2], [4, 3], [4, 4], [3, 4], [2, 4], [1, 4], [0, 4]]
The length of the shortest 4-type path is:  8
The 8-type path is:  [[4, 0], [4, 1], [3, 2], [2, 2], [1, 3], [0, 4]]
The length of the shortest 8-type path is:  5
The m-type path is:  [[4, 0], [4, 1], [4, 2], [4, 3], [4, 4], [3, 4], [2, 4], [1, 4], [0, 4]]
The length of the shortest m-type path is:  8
```

(b) When V is {1, 2}, the initial point p is (0, 0) and the ending point is q (4, 4).

```
The 4-type path is:  [[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [1, 4], [2, 4], [3, 4], [4, 4]]
The length of the shortest 4-type path is:  8
The 8-type path is:  [[0, 0], [1, 1], [2, 2], [3, 3], [4, 4]]
The length of the shortest 8-type path is:  4
The m-type path is:  [[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [1, 4], [2, 4], [3, 4], [4, 4]]
The length of the shortest m-type path is:  8
```

(c) When V is {0, 1, 2}, the initial point p is (1, 0) and the ending point is q (4, 4).

```
The 4-type path is:  [[1, 0], [2, 0], [3, 0], [4, 0], [4, 1], [4, 2], [4, 3], [4, 4]]
The length of the shortest 4-type path is:  7
The 8-type path is:  [[1, 0], [1, 1], [2, 2], [3, 3], [4, 4]]
The length of the shortest 8-type path is:  4
The m-type path is:  [[1, 0], [2, 0], [3, 0], [4, 0], [4, 1], [4, 2], [4, 3], [4, 4]]
The length of the shortest m-type path is:  7
```