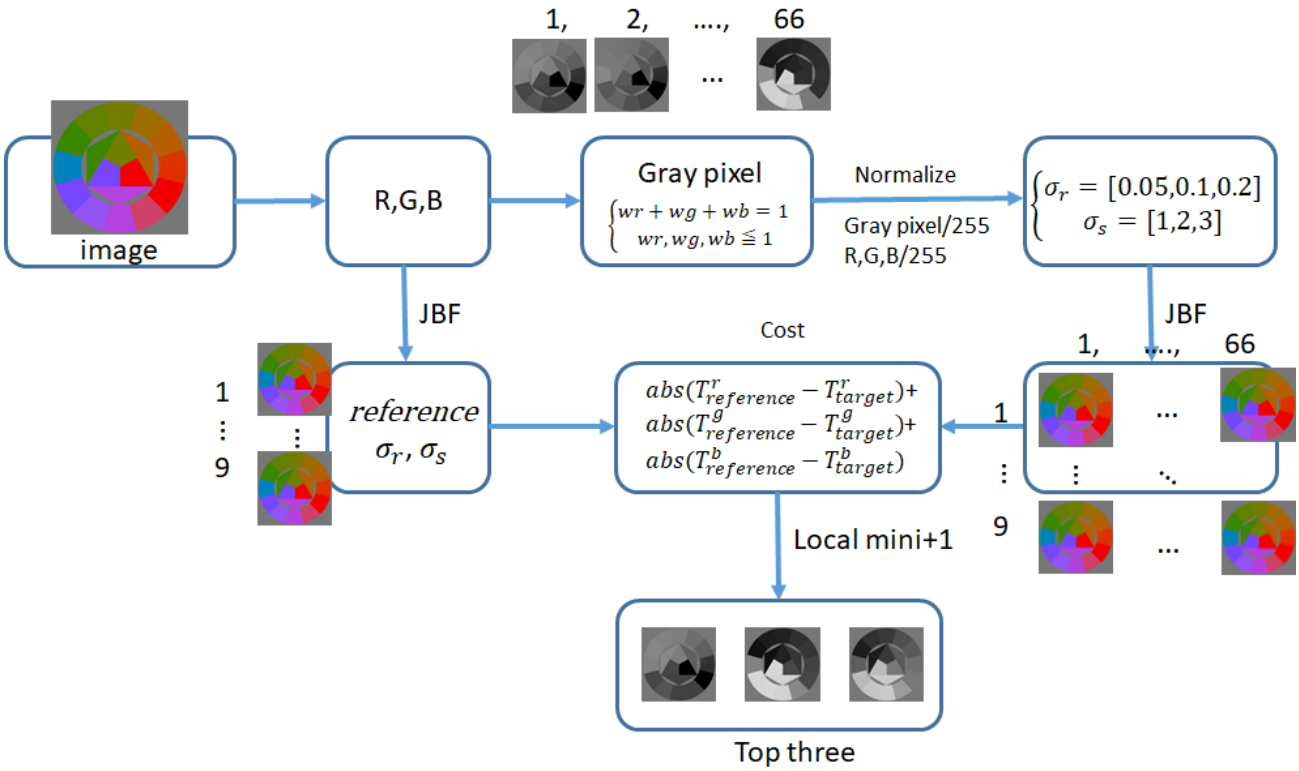
Assignment 1

ID: R06945011, NAME: Shih-Jou Chung(鍾詩柔), Department: BEBI

Input image: 2a,2b,2c

Aabstruct:



Design:

一開始將 testdata 的 RGB 和一個狀況下 wr,wg,wb 的 Graypixel 拿出來，並且正規化的除以 255，且在同一狀況下的σ_r, σ_s做 testdata 本身的 JBF 和在不同 weight 下轉成 Gray 的結果做 JBF，並進行兩者的比較，得到不同數值的 cost 後，取 local mini，只要是 local mini 就+1，做完全部後，看誰得票最多取前三。

JBF:分成 channel1 and channel3,原本的 testdata 利用 channel3 和 gray 利用 channel1。

$$F^T(I) = \frac{\sum_{q\in\Omega_p} G_s(p,q)G_r(T_p,T_q)I_q}{\sum_{q\in\Omega_p} G_s(p,q)G_r(T_p,T_q)}$$

$$G_r(T_p,T_q) = e^{-\frac{(T_p-T_q)^2}{2\sigma_r^2}}$$

channel 1

$$G_r(T_p, T_q) = e^{-\frac{(T_p^r - T_q^r)^2 + (T_p^g - T_q^g)^2 + (T_p^b - T_q^b)^2}{2\sigma_r^2}}$$

channel3

Testdata: Tp 為 window 的中心點(x,y)RGB 像素(normalize)；Tq 為(x-r,x+r)(y-r,y+r)的位置 RGB 像素(normalize)，Iq 為(x-r,x+r)(y-r,y+r)的位置 RGB 像素(non-normalize)

Gray: Tp 為 window 的中心點(x,y)gray 值(normalize)；Tq 為(x-r,x+r)(y-r,y+r)的位置 gray 值(normalize)，Iq 為(x-r,x+r)(y-r,y+r)的位置 RGB 像素(non-normalize)

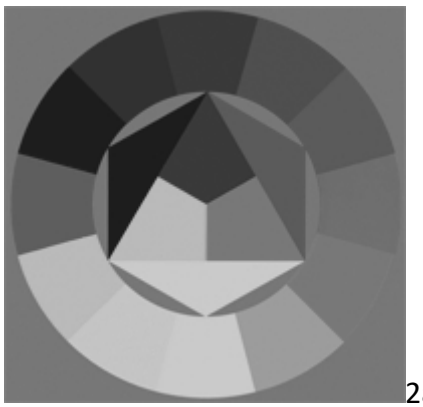
如果無正規化會讓σ_r，σ_s無法被顯現出來

Local mini: total 為(9,66)的 cost 矩陣，利用 argrelextrema(parameter, np.less)尋找 array 裡的 local mini 之後轉乘 array 並將他堆疊到 p 矩陣中，再利用 bincount(parameter)知道在各個 frame 當中票數最多的為何，再利用 np.argmax(parameter)找到最大的，將其設成零後在繼續找，直到找到第三個。

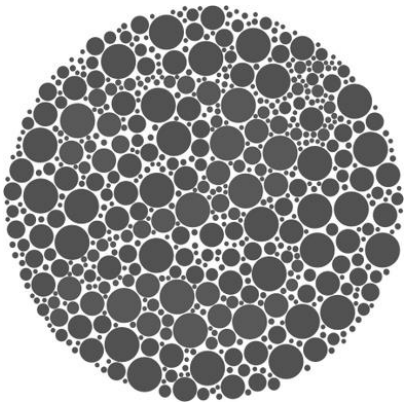
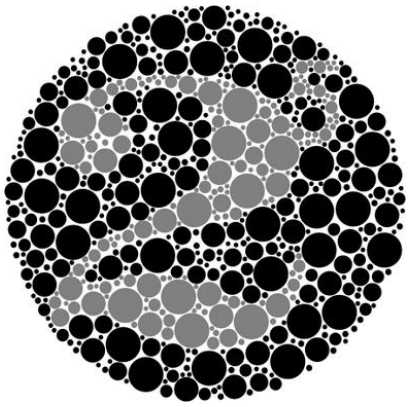
```
for a in range(9):
        mini= argrelextrema(total[a,:], np.less)
        o = np.asarray([a for b in mini for a in b])
        p = np.hstack((o,p))

p = p.astype(int)
s = np.bincount(p)
for a in range(3):
        u[a] = np.argmax(s)
        s[u[a]] = 0
```

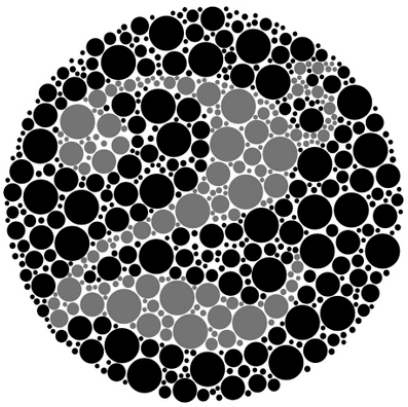
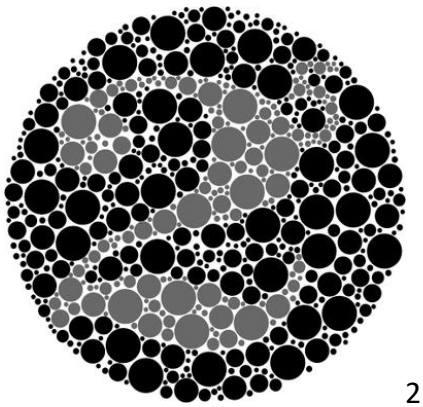
Results:


2a_y, 2a_y1



2a_y2, 2a_y3

wr	wg	wb
0.5	0.1	0.4
0.6	0	0.4
0.5	0	0.5



2b_y, 2b_y1



2b_y2, 2b_y3

wr	wg	wb
0	0	1
0.1	0	0.9
0.2	0	0.8


2c_y, 2c_y1


2c_y2, 2c_y3

wr	wg	wb
0.7	0.3	0
0.8	0.2	0
0.9	0.1	0

Code:

How to run? 將 main func 更改要 input 名稱即可與 output 的 2a/b/c_y0 看要哪一個，共 66 張 frame 會跑完，每跑完一個就會比較 reference 並將 cost 存入 total。

```
from PIL import Image
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import math
from scipy.signal import argrelextrema
from collections import Counter
```

```
def   joint_bilateral_filter(im,sigma_s,sigma_r,r,gray_pixel,RR,GG,BB):
        width,height = im.size
        FR = np.zeros((width,height), dtype=np.int)
        ima = Image.new('RGB',(width,height))
        FG = np.zeros((width,height), dtype=np.int)
        FB = np.zeros((width,height), dtype=np.int)

        FRrgb = np.zeros((width,height), dtype=np.int)
        imargb = Image.new('RGB',(width,height))
        FGrgb = np.zeros((width,height), dtype=np.int)
        FBrgb = np.zeros((width,height), dtype=np.int)

        RR = np.zeros((width+2*r,height+2*r), dtype=np.float)
        GG = np.zeros((width+2*r,height+2*r), dtype=np.float)
        BB = np.zeros((width+2*r,height+2*r), dtype=np.float)
        RR[r:(np.size(gray_pixel,0)+r),r:(np.size(gray_pixel,1)+r)] = R/255
        GG[r:(np.size(gray_pixel,0)+r),r:(np.size(gray_pixel,1)+r)] = G/255
        BB[r:(np.size(gray_pixel,0)+r),r:(np.size(gray_pixel,1)+r)] = B/255
        a,b = np.meshgrid(np.arange(-r,+r+1,1),np.arange(-r,+r+1,1))
        Gs = np.exp(-(a**2+b**2)/(2*sigma_s**2))

        gray_pixel1 = np.zeros((width+2*r,height+2*r), dtype=np.float)
        gray_pixel1[r:(np.size(gray_pixel,0)+r),r:(np.size(gray_pixel,1)+r)] = gray_pixel[:,:]
        for y in range(r,height+r):
                        for x in range(r,width+r):
                                a,b = np.meshgrid(np.arange(x-r,x+r+1,1),np.arange(y-r,y+r+1,1))
                                g = gray_pixel1[a[:,:],b[:,:]]
                                rr = RR[a[:,:],b[:,:]]
                                gg = GG[a[:,:],b[:,:]]
                                bb = BB[a[:,:],b[:,:]]
```

```
###3 channel
                                    Grrgb = np.exp(-(((rr-RR[x][y])**2+(gg-GG[x][y])**2+(bb-
BB[x][y])**2)/(2*sigma_r**2)))
                                    FuRrgb = sum(sum(Grrgb*Gs*rr))
                                    FuGrgb = sum(sum(Grrgb*Gs*gg))
                                    FuBrgb = sum(sum(Grrgb*Gs*bb))
                                    Fdrgb = sum(sum(Grrgb*Gs))
                                    FRrgb[x-r][y-r] = int(FuRrgb/Fdrgb*255)
                                    FGrgb[x-r][y-r] = int(FuGrgb/Fdrgb*255)
                                    FBrgb[x-r][y-r] = int(FuBrgb/Fdrgb*255)
                                    rgba2 = (FRrgb[x-r][y-r],FGrgb[x-r][y-r],FBrgb[x-r][y-r])
                                    imargb.putpixel((x-r,y-r),rgba2)
                                    ###1 channel
                                    Gr = np.exp(-((g-gray_pixel1[x][y])**2)/(2*sigma_r**2))
                                    FuR = sum(sum(Gr*Gs*rr))
                                    FuG = sum(sum(Gr*Gs*gg))
                                    FuB = sum(sum(Gr*Gs*bb))
                                    Fd = sum(sum(Gr*Gs))
                                    FR[x-r][y-r] = int(FuR/Fd*255)
                                    FG[x-r][y-r] = int(FuG/Fd*255)
                                    FB[x-r][y-r] = int(FuB/Fd*255)
                                    rgba = FR[x-r][y-r],FG[x-r][y-r],FB[x-r][y-r]
                                    ima.putpixel((x-r,y-r),rgba)
        ima.save("JTB"+str(i)+str(j)+"_"+str(number) + ".png")
        imargb.save("JTB_reference"+str(i)+str(j)+".png")

        return FR,FG,FB,FRrgb,FGrgb,FBrgb
```

```
def rgb2gray(im,R,G,B,w_R,w_G,w_B,number):
        width,height = im.size
        gray = Image.new('L',(width,height))
        gray_pixel = np.zeros((width,height), dtype=np.float)
        for y in range(height):
                for x in range(width):
                        rgba = (w_R*R[x][y]+    # R
                                w_G*G[x][y]+    # G
                                w_B*B[x][y])    # B
                        gray.putpixel((x,y),int(rgba))
                        gray_pixel[x][y]=(rgba)

        gray.save("gray"+str(number) + ".png")
        return gray_pixel/255

def rgb(im):
        width,height = im.size
        R = np.zeros((width,height), dtype=np.int)
        G = np.zeros((width,height), dtype=np.int)
        B = np.zeros((width,height), dtype=np.int)
        for y in range(height):
                for x in range(width):
                        rgba = im.getpixel((x,y))
                        R[x][y]=rgba[0]
                        G[x][y]=rgba[1]
                        B[x][y]=rgba[2]
                        im.putpixel((x,y),(R[x][y],G[x][y],B[x][y]))
        return R,G,B


def cost(FR,FG,FB,R,G,B,im):
    width,height = im.size
    total = 0.0
    for x in range(width):
        for y in range(height):
            total += abs(FR[x][y]-R[x][y])+abs(FR[x][y]-R[x][y])+abs(FR[x][y]-R[x][y])
    return total
```

```
if __name__ == '__main__':
        im = Image.open("2a.png")
        im.convert('L').save("2a_y.png")
        sigma_s = np.array([1,2,3])
        sigma_r = np.array([0.05,0.1,0.2])
        width,height = im.size
        r = 3*sigma_s    #ws = 2*r+1

        R,G,B = rgb(im)
        F = np.zeros((width,height), dtype=np.float)
        total = np.zeros((9,66), dtype=np.float)
        number =0
        p=[]
        u = [0,0,0]
        for    wr in np.arange(0,1.1,0.1):
                for wg in np.arange(1-wr,-0.1,-0.1):
                        wb = 1-wr-wg
                        if wb>=0:
                                n=0
                                gray_pixel = rgb2gray(im,R,G,B,wr,wg,wb,number)
                                for i in range(3):
                                        for j in range(3):
                                                FR,FG,FB,FR_r,FG_r,FB_r=
(joint_bilateral_filter(im,sigma_s[i],sigma_r[j],r[i],gray_pixel,R,G,B))
                                                total[n][number] =
cost(FR,FG,FB,FR_r,FG_r,FB_r,im)
                                n+=1
                                print("frame:"+ str(number))
                                number +=1
        for a in range(9):
                mini= argrelextrema(total[a,:], np.less)
                o = np.asarray([a for b in mini for a in b])
                p = np.hstack((o,p))

        p = p.astype(int)
        s = np.bincount(p)
        for a in range(3):
                u[a] = np.argmax(s)
                s[u[a]] = 0
```