Computer Vision
Fuzzy C-Means & SLIC Algorithms

Table of Contents

Question 1

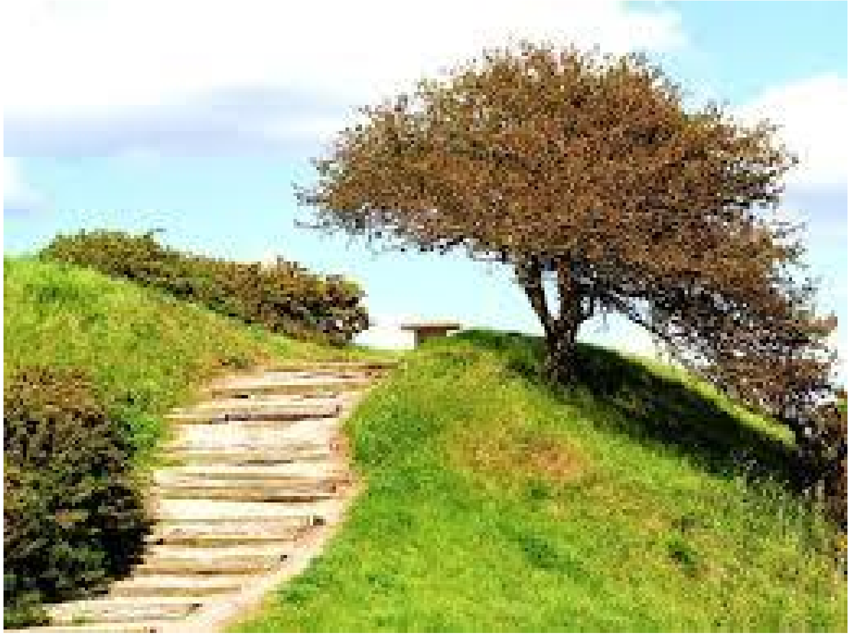
Write a program to implement a region segmentation algorithm using the fuzzy c-means algorithm on normalized 'RGBxy' data of an image. Merge stray (isolated) pixels (or very-small-regions) to their surrounding regions. [3 marks]

Answer

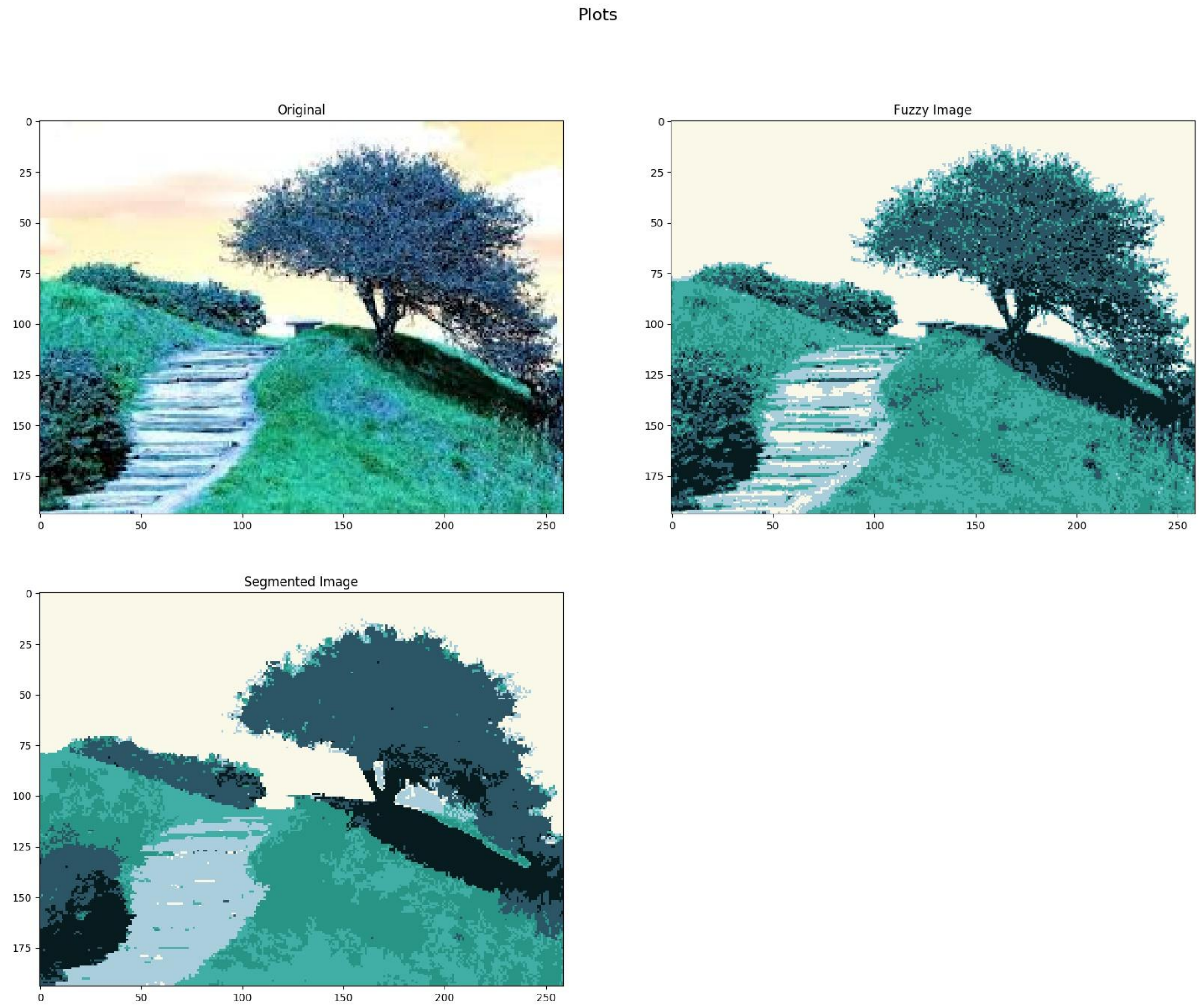
Algorithm:

1. Read the Image
2. Run the fuzzy cmeans algorithm with cluster size predefined (say 6) on the image to cluster the image on the basis of the colors.
3. The image is iterated and a dfs algorithm is used to find the size of the islands on the segmented image. Dfs algorithm also stores the boundary pixels to that connected component (color) in the boundary dictionary.
4. If the size of the island (of pixels) is smaller than the threshold then its color is replaced with the majority color in the boundary pixels using the boundary dictionary .

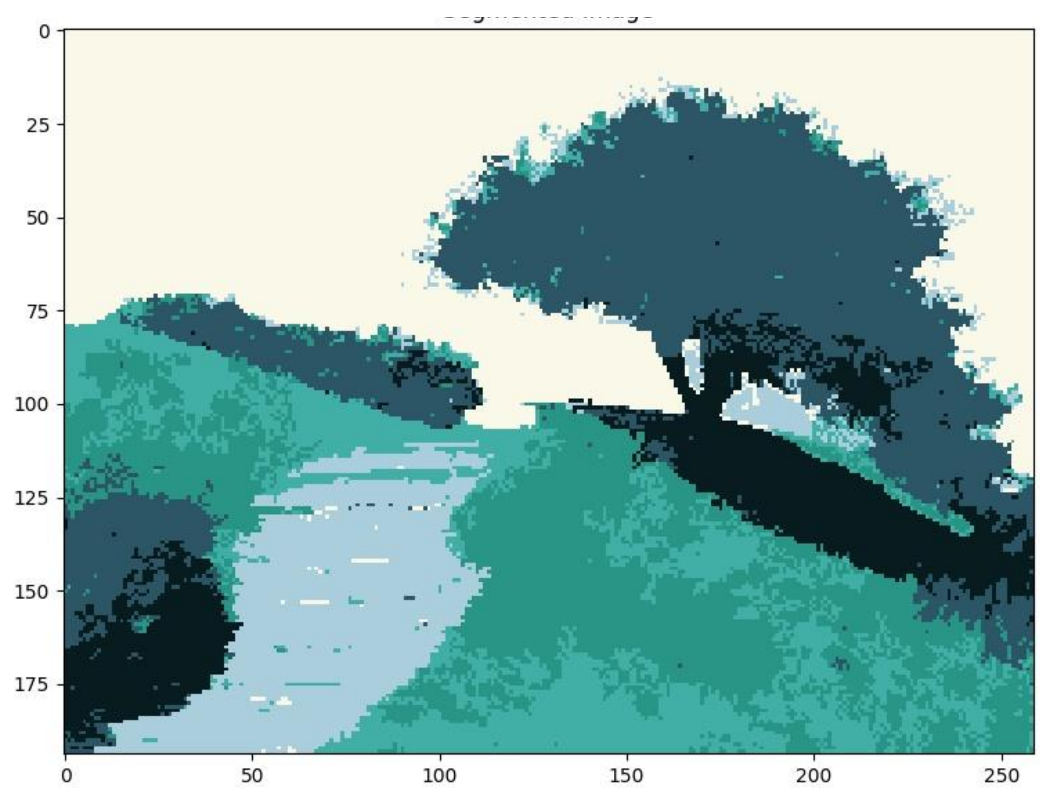
Input Image:



Output Image:



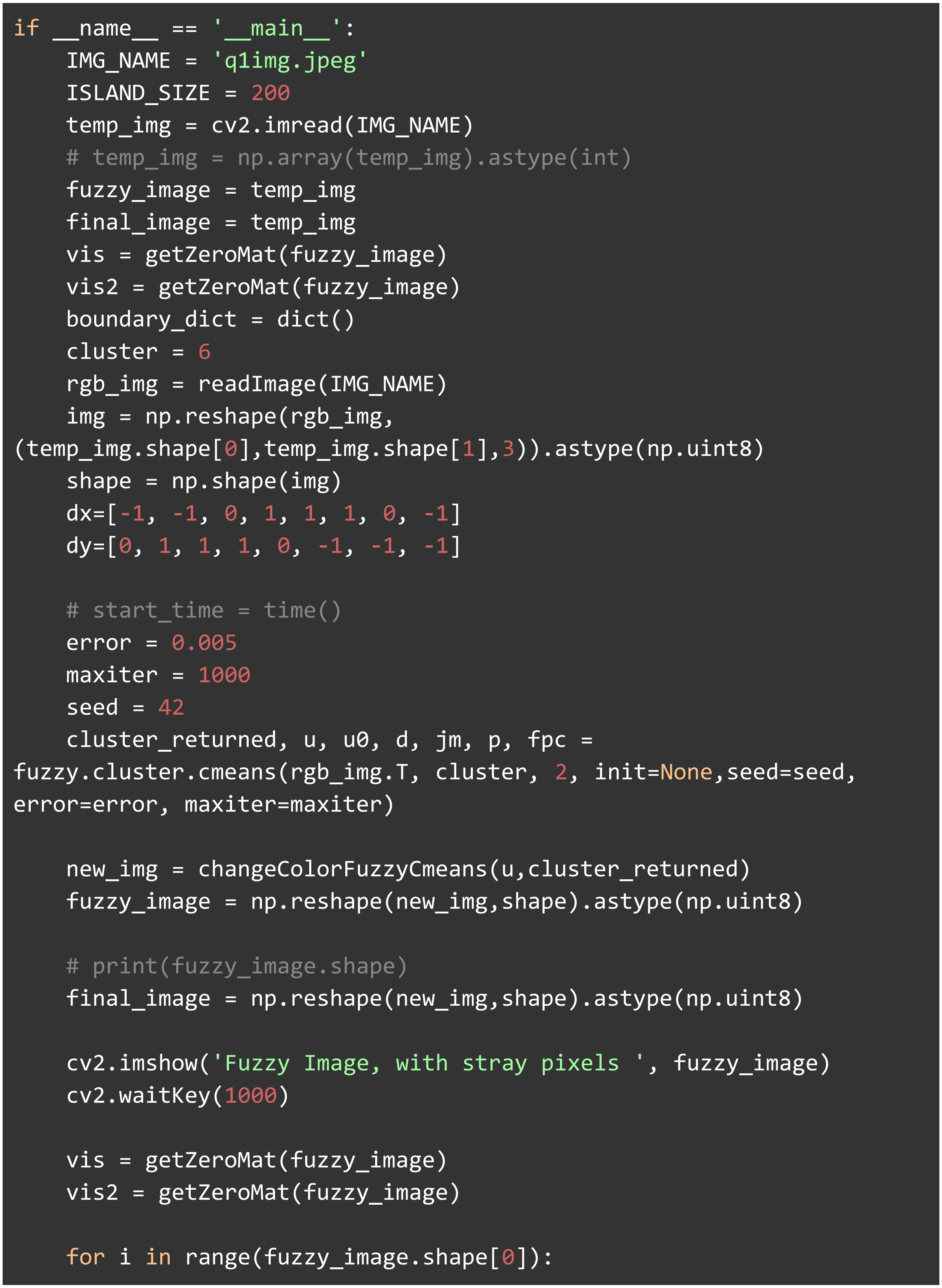
Segmented Image without Stray Pixels



Code:

```
import os
import cv2
import sys
import numpy as np
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzzy
from time import time
sys.setrecursionlimit(10**7)

def readImage(image_name):
    img = cv2.imread(image_name)
    # img = np.array(img).astype(int)
    print('Image shape is ',img.shape)
    norm_img = np.zeros(img.shape)
    final_image = cv2.normalize(img,  norm_img, 0, 255,
cv2.NORM_MINMAX)
```

```
    rgb_img = img.reshape((final_image.shape[0] * final_image.shape[1], 3))
    return rgb_img

def dfsAlgo(i,j,x,y,z,dx,dy):
    count=1
    for index in range(len(dx)):
        new_i = i + dx[index]
        new_j = j + dy[index]
        if (new_i<fuzzy_image.shape[0] and new_i >=0 and new_j<fuzzy_image.shape[1] and new_j>=0):
            if(x != fuzzy_image[new_i][new_j][0] or y != fuzzy_image[new_i][new_j][1] or z != fuzzy_image[new_i][new_j][2]):
                rgb_tuple = (fuzzy_image[new_i][new_j][0],fuzzy_image[new_i][new_j][1],fuzzy_image[new_i][new_j][2])
                if rgb_tuple in boundary_dict:
                    boundary_dict[rgb_tuple] += 1
                else:
                    boundary_dict[rgb_tuple] = 1
            elif vis[new_i][new_j]==0:
                vis[new_i][new_j] = 1
                count += dfsAlgo(new_i,new_j,x,y,z,dx,dy)
    return count

def changeColorFuzzyCmeans(cluster_members,clusters):
    image = []
    for pix in cluster_members.T:
        image.append(clusters[np.argmax(pix)])
    return image

def showPlots(img,fuzzy_image,segmented_image):
    fig = plt.figure(figsize=(20, 15))
    fig.add_subplot(221)
    plt.title('Original')
    # plt.set_cmap('gray')
    plt.imshow(img)

    fig.add_subplot(222)
    plt.title('Fuzzy Image')
    # plt.set_cmap('gray')
    plt.imshow(fuzzy_image)

    fig.add_subplot(223)
    plt.title('Segmented Image without Stray Pixels')
    # plt.set_cmap('gray')
    plt.imshow(segmented_image)

    fig.suptitle('Plots', fontsize=16)
    plt.savefig('Ans1.jpg')
    plt.show()

def mergeStrayPixels(i,j,majority_surrounding,x,y,z,dx,dy):
    for index in range(len(dx)):
        new_i = i + dx[index]
        new_j = j + dy[index]

        if(new_i<fuzzy_image.shape[0] and new_i >=0 and
new_j<fuzzy_image.shape[1] and new_j>=0):
            if(final_image[new_i][new_j][0] ==
majority_surrounding[0] and final_image[new_i][new_j][1] ==
majority_surrounding[1] and final_image[new_i][new_j][2] ==
majority_surrounding[2]):
                continue
            if(x == fuzzy_image[new_i][new_j][0] and y ==
fuzzy_image[new_i][new_j][1] and z == fuzzy_image[new_i][new_j][2]):
                final_image[new_i][new_j][0] =
majority_surrounding[0]
                final_image[new_i][new_j][1] =
majority_surrounding[1]
                final_image[new_i][new_j][2] =
majority_surrounding[2]

mergeStrayPixels(new_i,new_j,majority_surrounding,x,y,z,dx,dy)

def getZeroMat(fuzzy_image):
    return np.zeros((fuzzy_image.shape[0],fuzzy_image.shape[1]))
```



```
if __name__ == '__main__':
    IMG_NAME = 'q1img.jpeg'
    ISLAND_SIZE = 200
    temp_img = cv2.imread(IMG_NAME)
    # temp_img = np.array(temp_img).astype(int)
    fuzzy_image = temp_img
    final_image = temp_img
    vis = getZeroMat(fuzzy_image)
    vis2 = getZeroMat(fuzzy_image)
    boundary_dict = dict()
    cluster = 6
    rgb_img = readImage(IMG_NAME)
    img = np.reshape(rgb_img,
(temp_img.shape[0],temp_img.shape[1],3)).astype(np.uint8)
    shape = np.shape(img)
    dx=[-1, -1, 0, 1, 1, 1, 0, -1]
    dy=[0, 1, 1, 1, 0, -1, -1, -1]

    # start_time = time()
    error = 0.005
    maxiter = 1000
    seed = 42
    cluster_returned, u, u0, d, jm, p, fpc =
fuzzy.cluster.cmeans(rgb_img.T, cluster, 2, init=None,seed=seed,
error=error, maxiter=maxiter)

    new_img = changeColorFuzzyCmeans(u,cluster_returned)
    fuzzy_image = np.reshape(new_img,shape).astype(np.uint8)

    # print(fuzzy_image.shape)
    final_image = np.reshape(new_img,shape).astype(np.uint8)

    cv2.imshow('Fuzzy Image, with stray pixels ', fuzzy_image)
    cv2.waitKey(1000)

    vis = getZeroMat(fuzzy_image)
    vis2 = getZeroMat(fuzzy_image)

    for i in range(fuzzy_image.shape[0]):
        for j in range(fuzzy_image.shape[1]):
            if(vis[i][j]==0):
                x = final_image[i][j][0]
                y = final_image[i][j][1]
                z = final_image[i][j][2]
                boundary_dict = {(x,y,z):0}
                vis[i][j]=1
                area_of_island = dfsAlgo(i,j,x,y,z,dx,dy)
                if(area_of_island<=ISLAND_SIZE):
                    majority_surrounding = max(boundary_dict,key = 
boundary_dict.get)
                    final_image[i][j][0] = majority_surrounding[0]
                    final_image[i][j][1] = majority_surrounding[1]
                    final_image[i][j][2] = majority_surrounding[2]

mergeStrayPixels(i,j,majority_surrounding,x,y,z,dx,dy)

    showPlots(temp_img,fuzzy_image,final_image)
    # print('Fuzzy time for cluster',cluster,'is',time() - 
start_time,'seconds')
    # print()
    print('Completed the program..')
    cv2.imshow('Final Output',final_image)
    cv2.waitKey(5000)
```

Question 2 and Question 3

(are done together)

Q2) Write a program to obtain the spatial and contrast cues using SLIC superpixels of an image instead of pixels. [3 marks]

Q3) Implement the separation measure discussed in Sec III.B.1 of the following paper to obtain quality scores for the two cues obtained in Q2. Use these quality scores as weights while performing the weighted sum of the two cues for getting the final saliency cue. [4 marks]

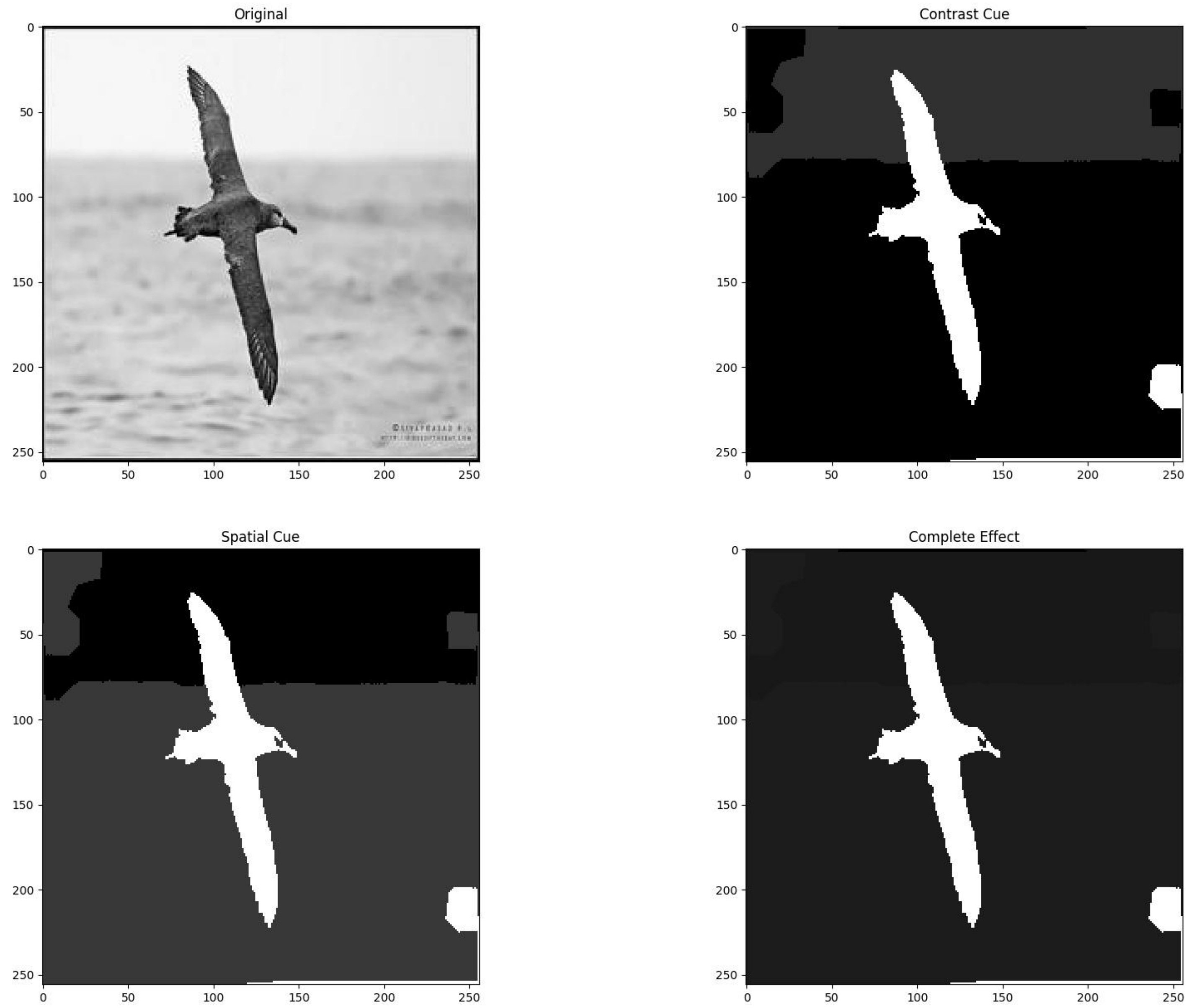
Answer

Algorithm:

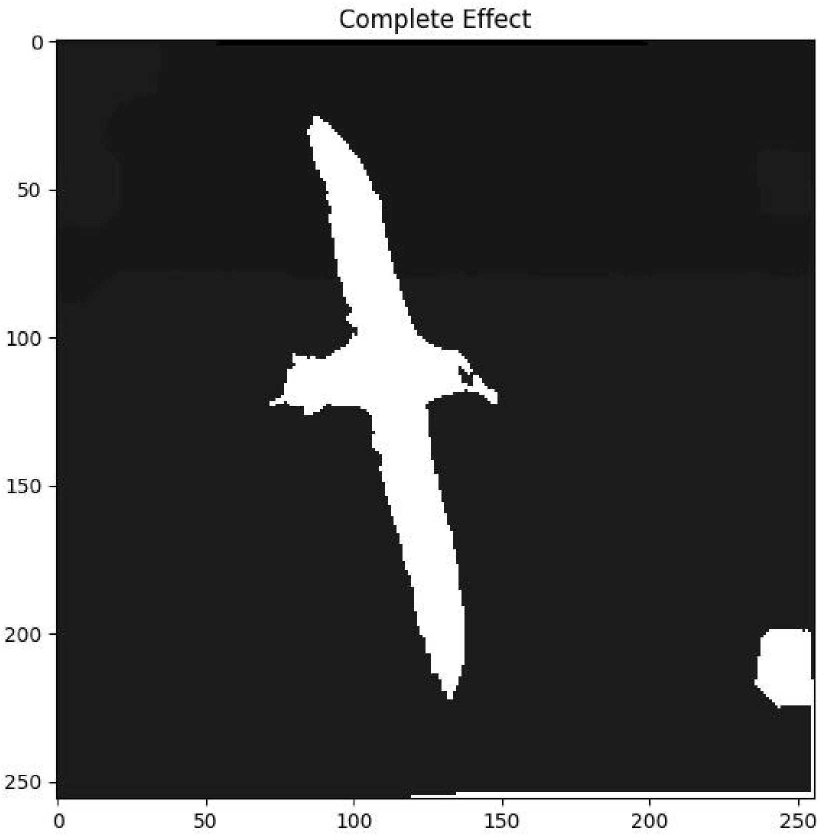
1. Read the image
2. Perform slic on it and get the superpixels.
3. Compute the coordinates and color of superpixels by averaging the pixels in it.
4. Perform k means (clustering algorithm) on the color of superpixels (kmeans runs for variable clusters at start and chooses the best cluster hyperparameter on its own).
5. Calculate the contrast que of the image
6. Calculate the spatial que of the image.
7. Find separation measure (phi values) using the matrix of contrast que and then spatial que.
    - Run Otsu threshold on the normalised matrix to find foreground and background.
    - Calculate the phi values using the separation measure formula implemented from the paper.
8. Find the final image by combining the contrast and spatial matrix using the phi values as weights. Normalise the matrix and multiply by 255.
9. Display all the results in the form of the plots.

Input Image:

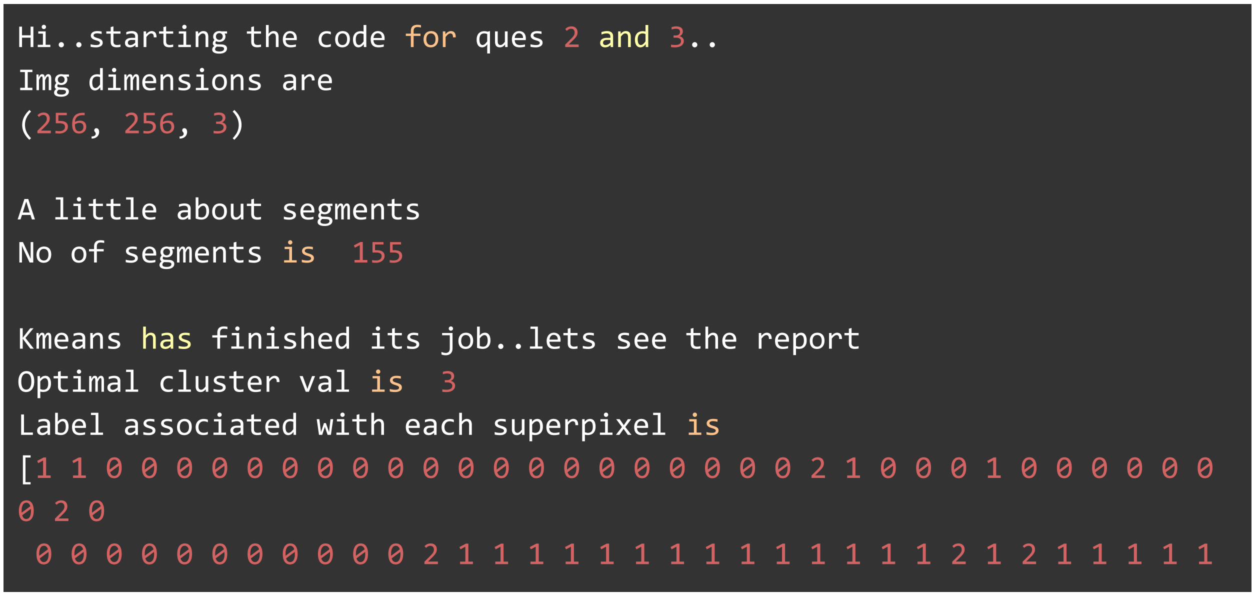
Output Plots (present in the folder also):



Final Image (formed using phi values and contrast and spatial matrix)



Output on terminal:



```
Hi..starting the code for ques 2 and 3..
Img dimensions are
(256, 256, 3)

A little about segments
No of segments is  155

Kmeans has finished its job..lets see the report
Optimal cluster val is  3
Label associated with each superpixel is
[1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 0 0 0 1 0 0 0 0 0 0
0 2 0
 0 0 0 0 0 0 0 0 0 0 0 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1
2 2 1
 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
1 1 1
 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
 1 1 1 1 1 1 1]

Contrast cue for each cluster (cluster of superpixels) is
(normalisation is done later)
{0: 68.55829848057388, 1: 34.591030398705065, 2: 208.98134593475277}

Spatial cue for each cluster (cluster of superpixels) is
(normalisation is done later)
{0: 0.04857264772851222, 1: 0.1054137145789455, 2:
0.3018705271868717}

Phi value for Contrast =  0.9999999700909099
Phi value for Spatial =  0.999996870790117

Plotting the results now
```

Code:

```python
import cv2
import numpy as np
from skimage.segmentation import slic
from sklearn.cluster import KMeans
from math import sqrt,exp, log, log10
from matplotlib import pyplot as plt
from scipy import integrate



# returns a map with labels as keys and list of corresponding pixel
coordinates with that label
def makeSuperPixelMap(superpixel_labels):
     smap = {}
```

```
    for i in range(superpixel_labels.shape[0]):
        for j in range(superpixel_labels.shape[1]):
            key = superpixel_labels[i][j]
            if key==0:
                continue
            value = (i,j)
            if key in smap:
                smap[key].append(value)
            else:
                smap[key]=[value]
    return smap

def findSuperPixelColors(img,smap):
    sp_color_map={}
    keys = smap.keys()

    for key in smap.keys():
        color = np.zeros(3)
        for pixel in smap[key]:
            # print(img[pixel[0]][pixel[1]])
            color[0] += img[pixel[0]][pixel[1]][0]
            color[1] += img[pixel[0]][pixel[1]][1]
            color[2] += img[pixel[0]][pixel[1]][2]
            # print('color is ',color)
        color = color/len(smap[key])

        sp_color_map[key] = np.array(color).astype(int)
    return sp_color_map

def findSuperPixelCoords(smap):
    sp_coord_map={}
    keys = smap.keys()

    for key in smap.keys():
        coord = (0,0)
        for pixel in smap[key]:
            coord = (coord[0] + pixel[0], coord[1]+pixel[1])
        coord = (int(coord[0]/len(smap[key])) ,
int(coord[1]/len(smap[key])))
```

```
        sp_coord_map[key] = coord
    return sp_coord_map

def l2norm(vec):
    return sqrt(vec[0]**2 + vec[1]**2 + vec[2]**2)

def convertDictValuestoNumpy(dictValues):
    xsize = len(dictValues)
    ysize = dictValues[0].size
    nparr = np.zeros((xsize,ysize))
    for i in range(xsize):
        for j in range(ysize):
            nparr[i][j] = dictValues[i][j]
    return nparr

def calcContrastCue(labels,unique_labels,cluster_centers):
    # total no of pixels = no of superpixels
    N = len(labels)
    contrast_que = {}
    for i in range(len(unique_labels)):
        kth_label = unique_labels[i]
        kth_mean = cluster_centers[kth_label]
        con_value = 0

        for j in range(len(unique_labels)):
            other_label = unique_labels[j]
            other_mean = cluster_centers[other_label]
            n = list(labels).count(other_label)
            if kth_label!=other_label:
                con_value += l2norm(kth_mean-other_mean) * 
(n/N)

        contrast_que[kth_label] = con_value

    return contrast_que

def 
calcSpatialCue(img,sp_coord_map,labels,unique_labels,cluster_centers)
```

```
:
    spatial_cue = {}
    h,w, channels = img.shape
    center_o = (h/2,w/2)
    d = sqrt(h**2 + w**2)

    for i in range(len(unique_labels)):
        kth_label = unique_labels[i]
        all_pixels = []
        # iterate through the labels, if same then iterate through
the pixels
        for j in range(len(labels)):
            if kth_label==labels[j]:
                #j+1
                all_pixels.append(sp_coord_map[j+1])

        n = list(labels).count(kth_label)
        # sigma = np.std(all_pixels)
        sigma = sqrt(d/2)
        c = sigma * sqrt(2*22/7)
        spatial_value=0
        for point in all_pixels:
            val = sqrt((point[0]-center_o[0])**2 +
(point[1]-center_o[1])**2)
            spatial_value += 1/exp(val/c)
        spatial_value = spatial_value * (1/n)

        spatial_cue[kth_label] = spatial_value
    return spatial_cue

# forms images using the cue values given
def formImage(img,cue,labels,smap):
    cue_values = list(cue.values())
    max_value = max(cue_values)
    min_value = min(cue_values)
    cue_values = np.array(cue_values) - min_value
    # max_value = np.max(cue_values)
    cue_values = cue_values/max_value
    cue_values = cue_values *255
    i=0
    for key in cue:
        cue[key] = cue_values[i]
        i+=1
    # print('Normalized cue is')
    # print(cue)
    # print()

    new_img = np.zeros(img.shape)
    for i in range(len(labels)):
        label = labels[i]
        # i+1
        pixels = smap[i+1]

        for pixel in pixels:
            new_img[pixel[0]][pixel[1]] = int(cue[label])
    return new_img

# displays all the plots
def showPlots(img,contrast_image,spatial_image,final_image):

    fig = plt.figure(figsize=(20, 15))
    fig.add_subplot(221)
    plt.title('Original')
    plt.set_cmap('gray')
    plt.imshow(img)

    fig.add_subplot(222)
    plt.title('Contrast Cue')
    plt.set_cmap('gray')
    plt.imshow(contrast_image)

    fig.add_subplot(223)
    plt.title('Spatial Cue')
    plt.set_cmap('gray')
    plt.imshow(spatial_image)

    fig.add_subplot(224)
    plt.title('Complete Effect')
    plt.set_cmap('gray')
    plt.imshow(final_image)

    fig.suptitle('Plots', fontsize=16)
    plt.savefig('Ans2&3.jpg')
    plt.show()


def ostuThresholding(mat):
    MIN_PIXEL_SUM = 10000000000
    OTSU_THRESH = -1
    unique_values = np.unique(mat)

    for i in range(len(unique_values)):
        temp_thresh = unique_values[i]
        pos1 = mat>=temp_thresh
        pos2 = mat<temp_thresh

        l1 = np.array([])
        l2 = np.array([])

        if pos1.any():
            l1 = np.array(mat[pos1]).ravel()
        if pos2.any():
            l2 = np.array(mat[pos2]).ravel()

        val=1000000
        if len(l1)!=0 and len(l2)!=0:
            val = np.var(l1)*l1.shape[0] +
np.var(l2)*l2.shape[0]
        elif len(l1)!=0:
            val = np.var(l1)*l1.shape[0]
        else:
            val = np.var(l2)*l2.shape[0]
        # print(val,MIN_PIXEL_SUM)

        if(val<MIN_PIXEL_SUM):
            OTSU_THRESH=temp_thresh
            MIN_PIXEL_SUM =val

    return OTSU_THRESH

def normalizeMat(mat):
    min_value = np.min(np.min(mat))
    max_value = np.max(np.max(mat))
    mat = mat - min_value
    mat = mat/max_value
    return mat

# finds phi, seperation measure
def findPhi(mat):
    mat = normalizeMat(mat)

    otsu_thres = ostuThresholding(mat)
    # print('OTSU IS',otsu_thres)

    # color_values = list(sp_color_map.values())
    # np_color_values = np.array(color_values)
    pos1 = mat>=otsu_thres
    pos2 = mat<otsu_thres

    mean_f = 0
    mean_b = 0
    var_f = 0
    var_b = 0

    if pos1.any():
        l1 = np.array(mat[pos1]).ravel()
        mean_f = np.mean(l1)
        var_f = np.var(l1)
    if pos2.any():
        l2 = np.array(mat[pos2]).ravel()
        mean_b = np.mean(l2)
        var_b = np.var(l2)
    var_f+=0.01
    var_b+=0.01
```

```
	# print('mean and var is ',mean_f,mean_b,var_f,var_b)
	term1 = (mean_b*var_f - mean_f*var_b)/(var_f - var_b)
	term2 = sqrt(var_f*var_b)/(var_f-var_b) * sqrt( (mean_f -
mean_b)**2 - 2*(var_f-var_b)*(log(sqrt(var_b))-log(sqrt(var_f))))

	tempz1 = term1 + term2
	tempz2 = term1 - term2


	# print('zstar values are coming out to be
',tempz1,tempz2,'\n')

	z_star = tempz1
	# max(tempz1,tempz2)


	func_f = lambda x:
exp(-1*((x-mean_f)**2)/var_f)/sqrt(var_f*2*22/7)
	func_b = lambda x:
exp(-1*((x-mean_b)**2)/var_b)/sqrt(var_b*2*22/7)

	overlap = integrate.quad(func_f, 0, z_star)[0] +
integrate.quad(func_b,z_star,1)[0]
	# print('Loss is ',overlap)

	#set no of bins
	Loss = abs(overlap)
	gamma = 10
	phi = 1/(1+log10(1+gamma*Loss))

	return phi

def calcDis(x1,y1,a,b,c):
	d = abs((a*x1+b*y1+c))/sqrt(a**2 + b**2)
	return d

def performKmeans(sp_color_list):
	clusters_list = np.arange(15)+1
	dist_points_from_cluster_center = []
```

```
      for clusters in clusters_list:
            kmeans = KMeans(n_clusters=clusters, random_state=0)
            kmeans.fit(sp_color_list)
            dist_points_from_cluster_center.append(kmeans.inertia_)

      a = dist_points_from_cluster_center[0] -
dist_points_from_cluster_center[-1]
      b = clusters_list[-1] - clusters_list[0]
      c1 = clusters_list[0] * dist_points_from_cluster_center[-1]
      c2 = clusters_list[-1] * dist_points_from_cluster_center[0]
      c = c1-c2

      distance_from_line =[]
      for i in range(len(clusters_list)):

distance_from_line.append(calcDis(clusters_list[i],dist_points_from_c
luster_center[i],a,b,c))

      # find index with max value
      index  = distance_from_line.index(max(distance_from_line))
      optimal_cluster = clusters_list[index]

      kmeans = KMeans(n_clusters=optimal_cluster, random_state=0)
      kmeans.fit(sp_color_list)
      labels = kmeans.labels_
      cluster_centers = kmeans.cluster_centers_

      return (optimal_cluster,labels,cluster_centers)



if __name__ == '__main__':

      print('Hi..starting the code for ques 2 and 3..')

      # image_name ='iiitd.jpg'
      image_name = 'q2&3bird.jpg'

      img = cv2.imread(image_name)
```

```
    # img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    print('Img dimensions are')
    print(img.shape)
    print()

    #*************** Question 2

    n_segments=163
    compactness=20
    convert2lab=True
    segments = slic(img, n_segments=n_segments,
start_label=1,convert2lab=convert2lab,compactness=compactness)

    print('A little about segments')
    print('No of segments is ',np.max(np.max(segments)))
    # print(type(segments))
    # print(len(segments[0]))
    # print(segments)
    print()

    smap = makeSuperPixelMap(segments)
    # print('Now lets see what we created')
    # print(smap.keys())
    # print()

    sp_coord_map = findSuperPixelCoords(smap)
    # print('Whats the associated coord for the superpixel')
    # print(sp_coord_map)
    # print()

    sp_color_map = findSuperPixelColors(img,smap)
    # print('Whats the associated color for the superpixel')
    # print(sp_color_map)
    # print()

    # used for clustering
    sp_color_list = list(sp_color_map.values())
    # sp_color_list = convertDictValuestoNumpy(sp_color_list)
    # print(len(sp_color_list))
    # print('converting these values to list now')
    # print(type(sp_color_list))
    # print(sp_color_list)
    # print()

    # clusters= 4
    # kmeans = KMeans(n_clusters=clusters, random_state=0)
    # kmeans.fit(sp_color_list)
    # labels = kmeans.labels_
    # cluster_centers = kmeans.cluster_centers_
    optimal_cluster_val,labels,cluster_centers =
performKmeans(sp_color_list)
    unique_labels = list(set(labels))
    print('Kmeans has finished its job..lets see the report')
    print('Optimal cluster val is ',optimal_cluster_val)
    print('Label associated with each superpixel is ')
    print(labels)
    # print(unique_labels)
    # print(cluster_centers[0])
    print()

    contrast_que =
calcContrastCue(labels,unique_labels,cluster_centers)
    print('Contrast cue for each cluster (cluster of superpixels)
is (normalisation is done later) ')
    print(contrast_que)
    print()

    spatial_cue =
calcSpatialCue(img,sp_coord_map,labels,unique_labels,cluster_centers)
    print('Spatial cue for each cluster (cluster of superpixels) is
(normalisation is done later) ')
    print(spatial_cue)
    print()

    # ********** Question 3
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    contrast_image = formImage(img,contrast_que,labels,smap)
    phi_contrast = findPhi(contrast_image)
    spatial_image = formImage(img,spatial_cue,labels,smap)
    # print('spatial_image is ')
    # print(spatial_image)
    # print()

    phi_spatial = findPhi(spatial_image)

    print('Phi value for Contrast = ',phi_contrast)
    print('Phi value for Spatial = ',phi_spatial)
    print()


    final_image = contrast_image * phi_contrast +  spatial_image *
phi_spatial
    max_value = np.max(np.max(final_image))
    min_value = np.min(np.min(final_image))
    final_image = ((final_image - min_value)/max_value)*255

    print('Plotting the results now')
    print()
    showPlots(img,contrast_image,spatial_image,final_image)
```