Digital Image Processing
Assignment # 2

Submitted by

Ali Akram 260336
Minhaj Khokhar 265439

DEPARTMENT OF ELECTRICAL ENGINEERING

College of Electrical and Mechanical Engineering (CEME)

Task Distribution

Ali Akram

- Main function, K-mean function, Seed pixels function, report.

Minhaj Khokhar

- Likelihood of pixels, updating centroids, generating index.

Segmentation

In this assignment we implemented interactive image cut-out / segmentation approach called Lazy Snapping. We were given several test images, along with corresponding auxiliary images depicting the foreground and background “seed” pixels marked with red and blue brushstrokes, respectively. Our program exploited these partial human annotations in order to compute a precise figure-ground segmentation.

Code

Libraries

```
import cv2
import numpy as np
import time
from PIL import Image
```

Main Function

```
def main():

    stroke_img1 = Image.open(r'C:\Users\k-ali\Desktop\FYP\fyp\ML sem\dip\assign 
2\dance stroke 1.png')
    stroke_img2 = Image.open(r'C:\Users\k-ali\Desktop\FYP\fyp\ML sem\dip\assign 
2\dance stroke 2.png')
    original_img = cv2.imread(r'C:\Users\k-ali\Desktop\FYP\fyp\ML sem\dip\assign 
2\dance.PNG')
    N = 64

    Segmentation(original_img, stroke_img1, N)
    Segmentation(original_img, stroke_img2, N)




if __name__ == '__main__':
    main()
```

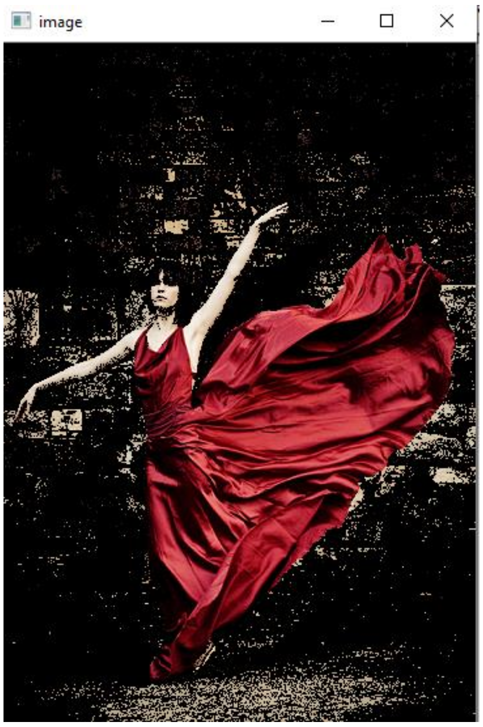
Results

Dance Img

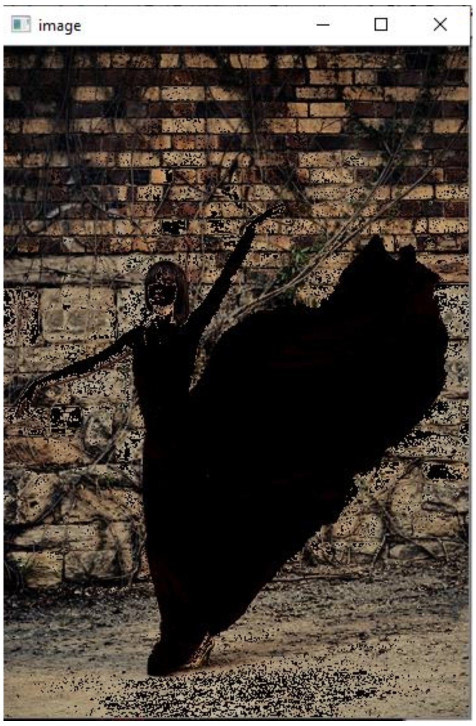
For N = 64

With Stroke img 1

Foreground

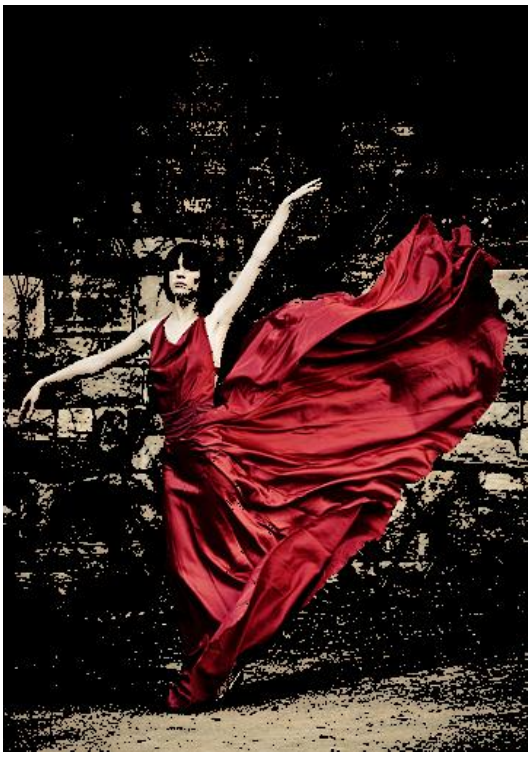


Background

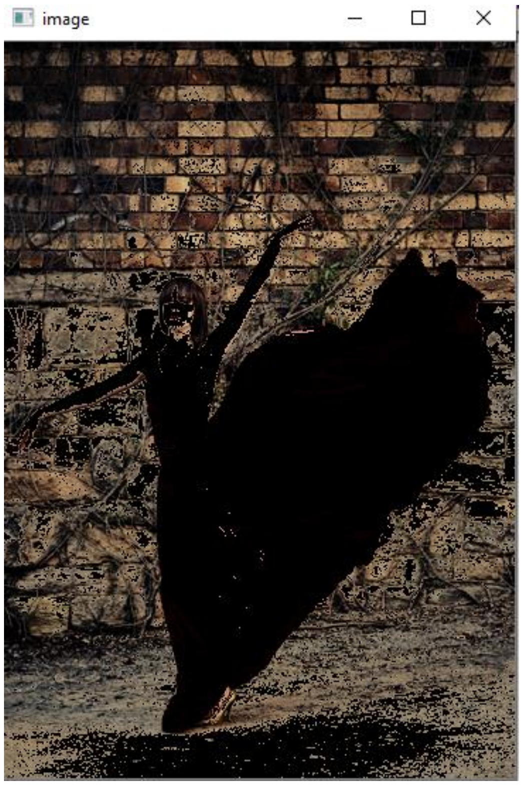


With stroke img 2

Foreground

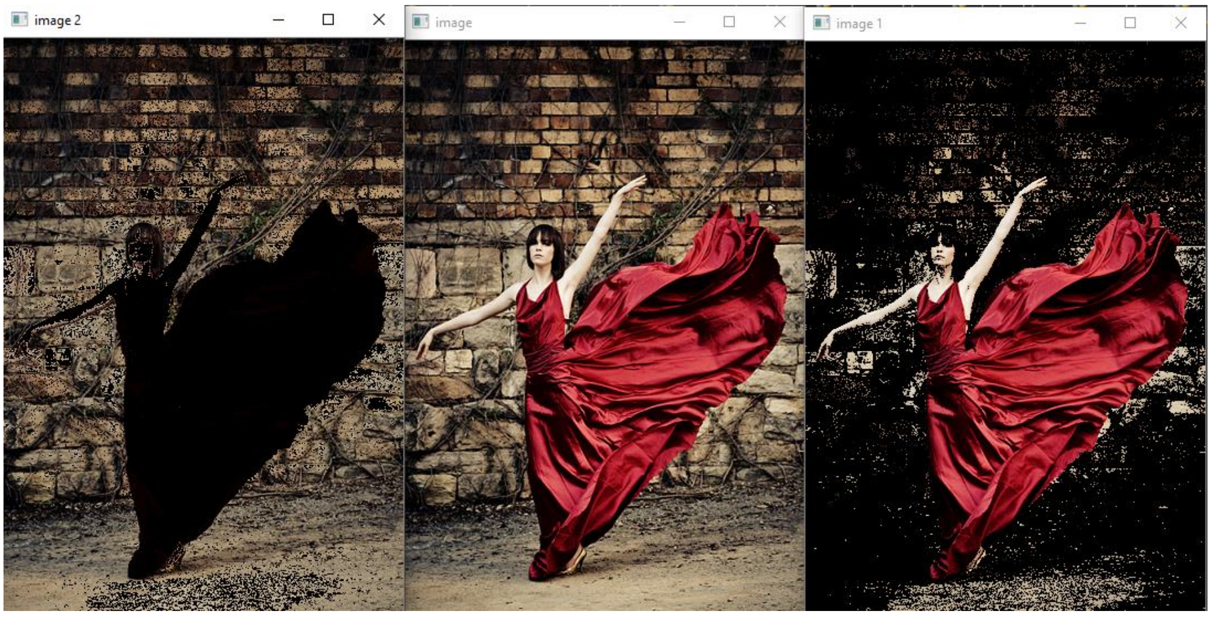


Background

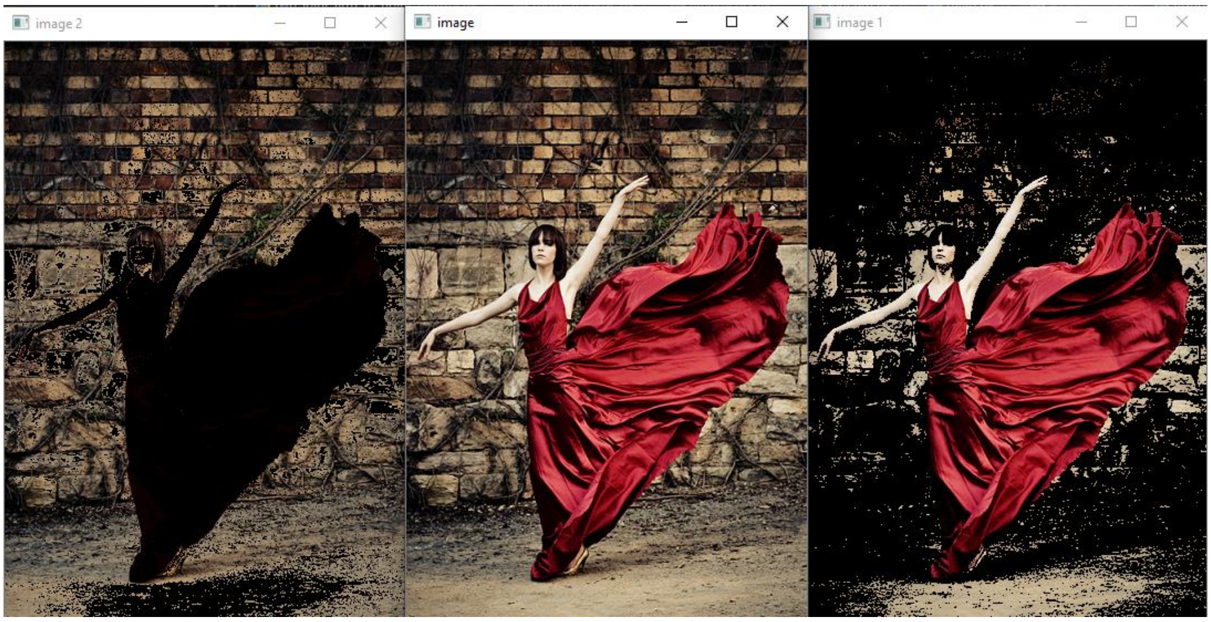


So, we saw that here in this image stroke image 1 gave us better results then stroke image 2.

When N = 25 with stroke image 1



When N = 25 with stroke image 2



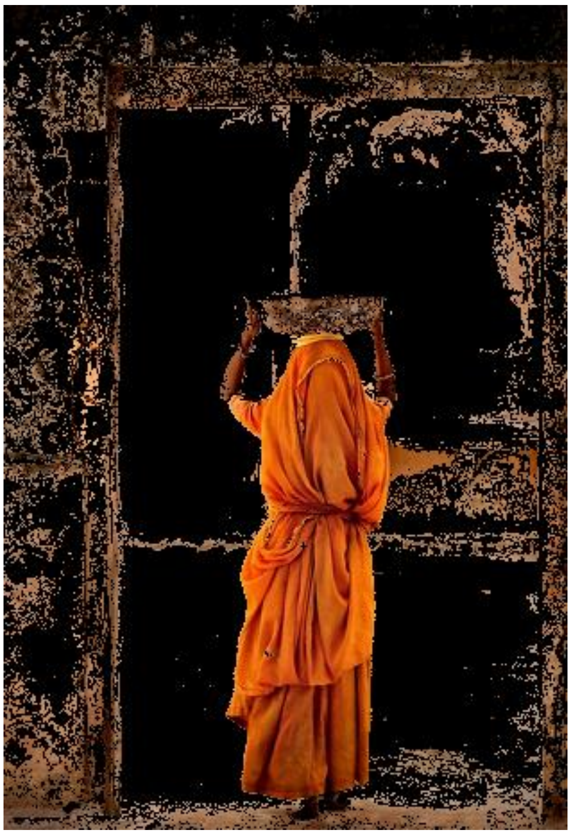
So we saw that difference is very little but N = 64 is slightly better than N = 25, also when N = 64 it took long time to show results.

Lady Img

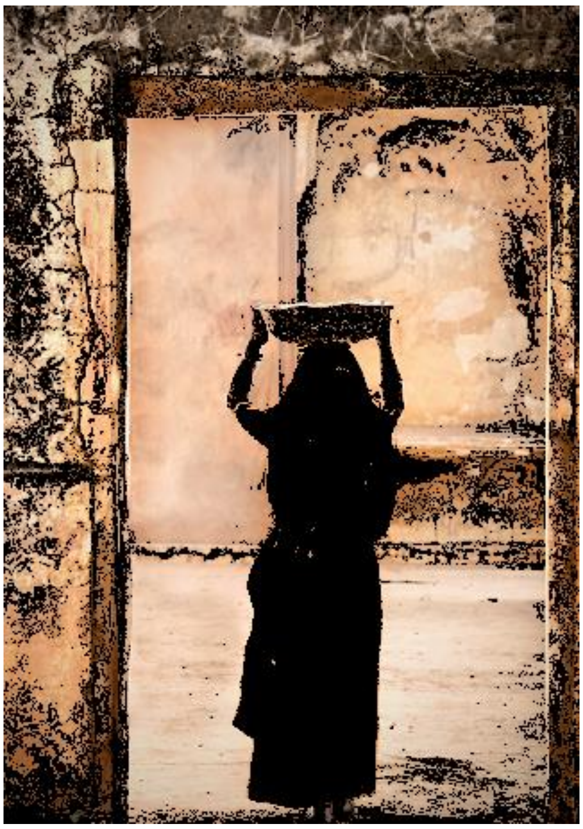
For N = 64

With Stroke img 1

Foreground

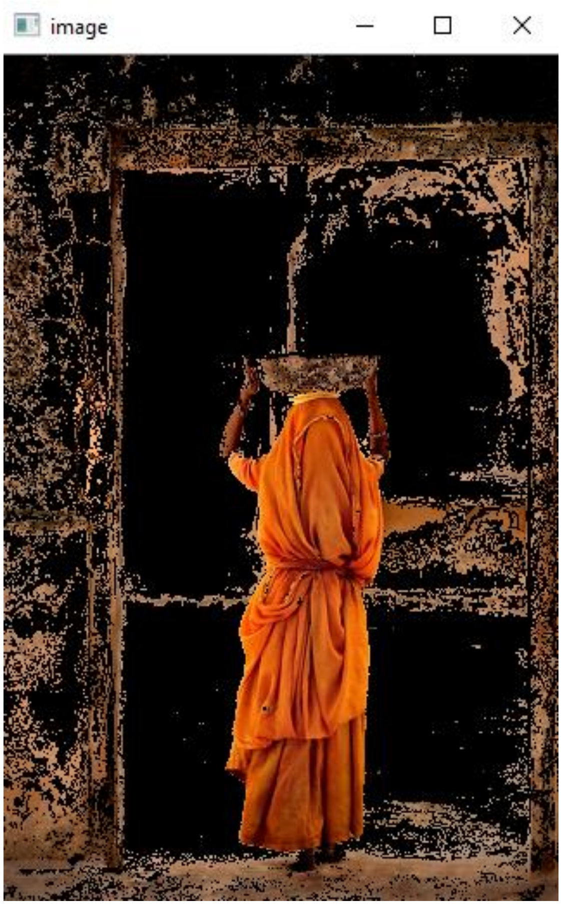


Background

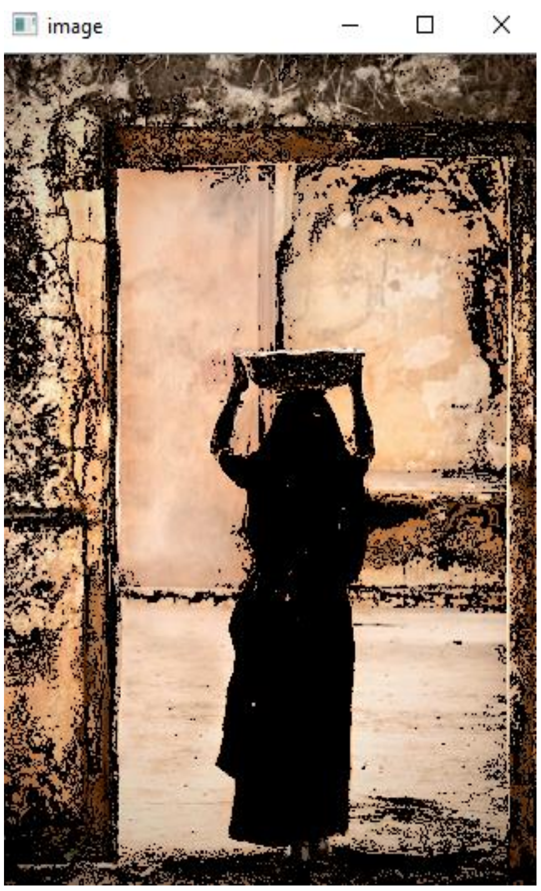


With Stroke img 2

Foreground

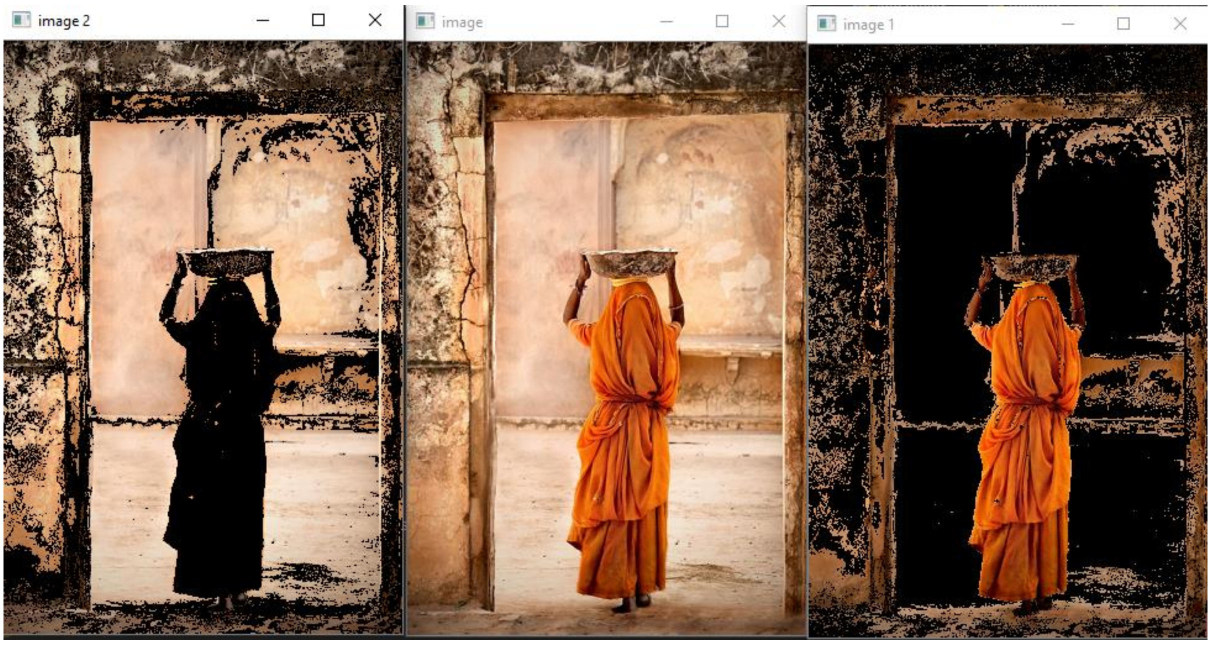


Background

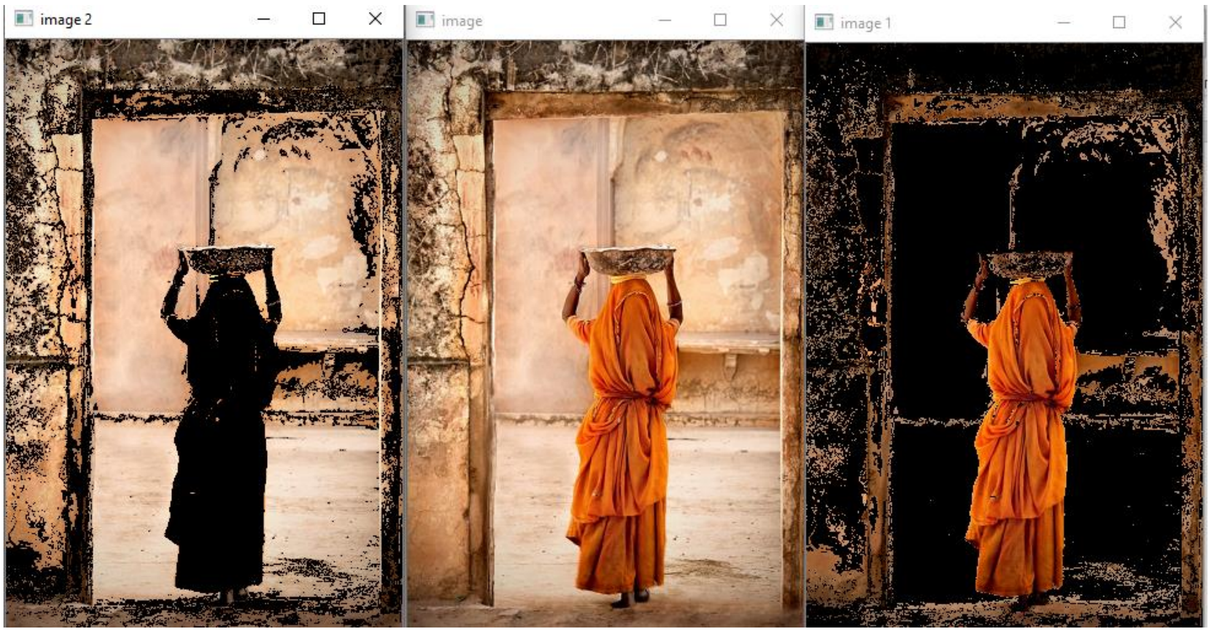


So, we saw that stroke image 2 gave us better results then stroke image 1.

When N = 10 with stroke image 1



When N = 10 with stroke image 2



We saw a big difference in the processing time of N = 64 vs N = 10

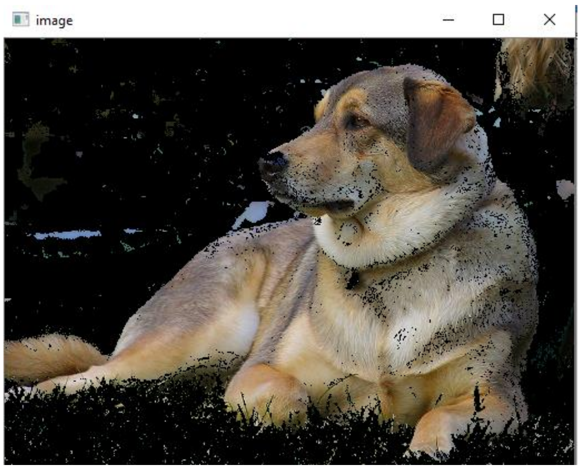
process time here in this case in minutes is 2.82

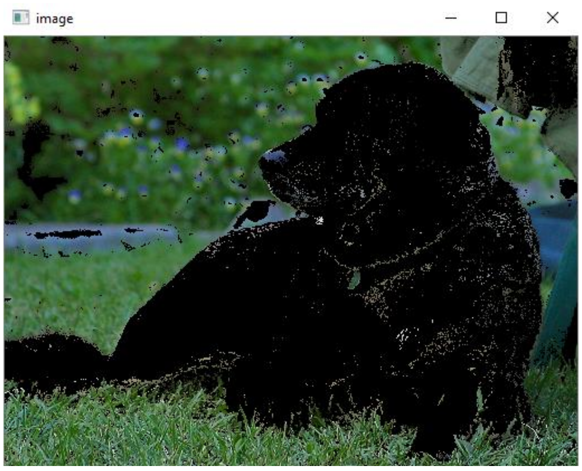
Dog Img

For N = 64

Foreground

Background

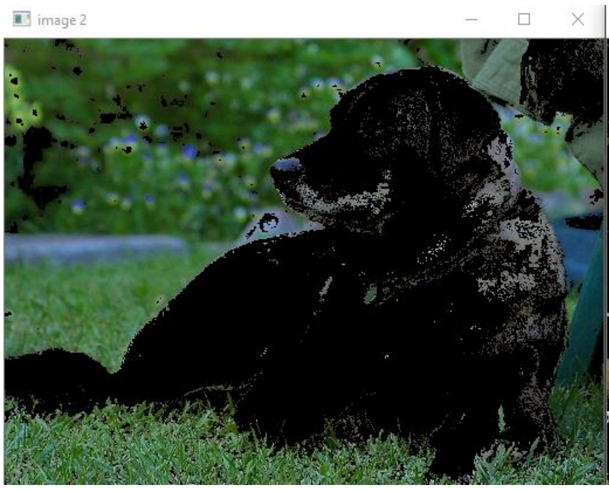


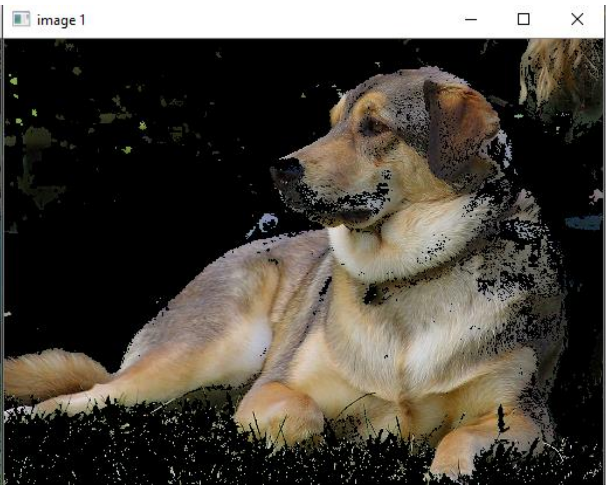


For N = 10

Background

Foreground





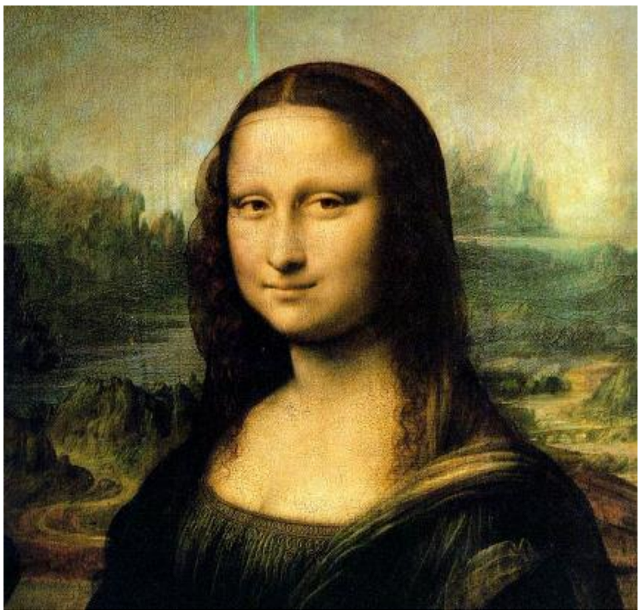
So, we can clearly see that foreground and background of N = 64 is better than N = 10.

When N is 10 some foreground is not correctly segmented, at 64 white fur of dog is more visible

The computation time of N = 64 is way more.

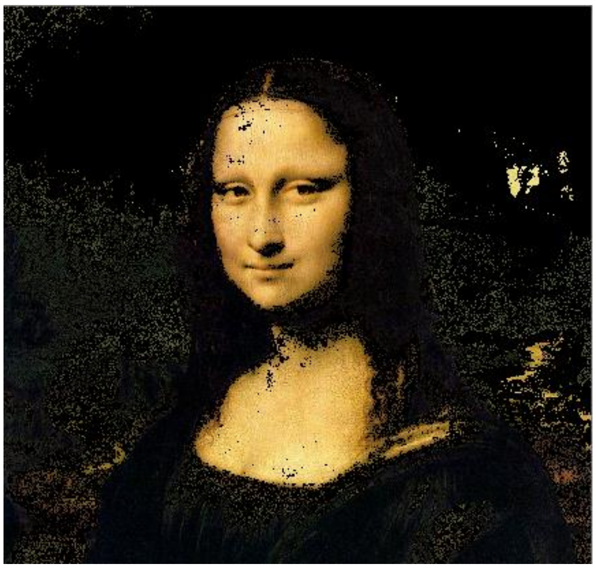
Monalisa Img

For N = 64

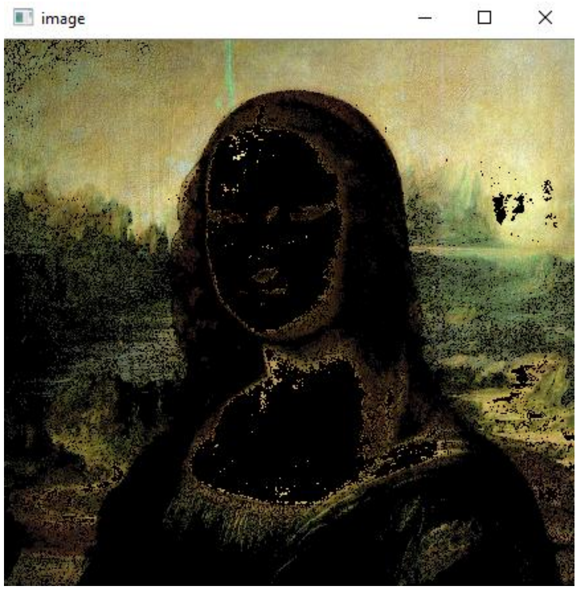


With Stroke img 1

Foreground

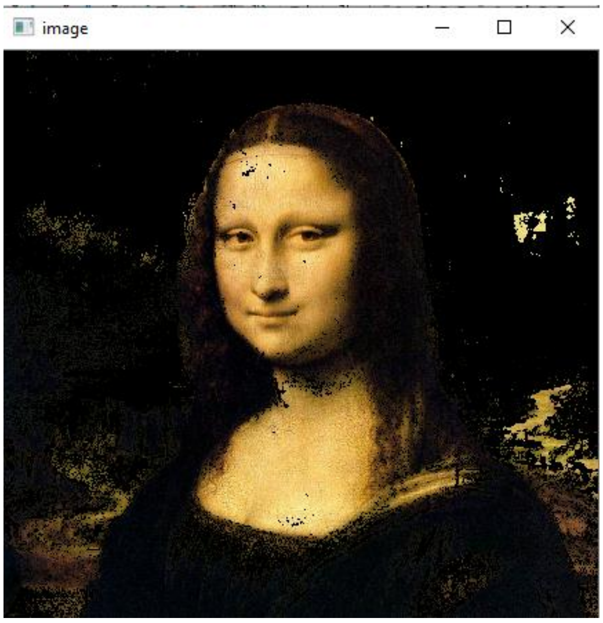


Background

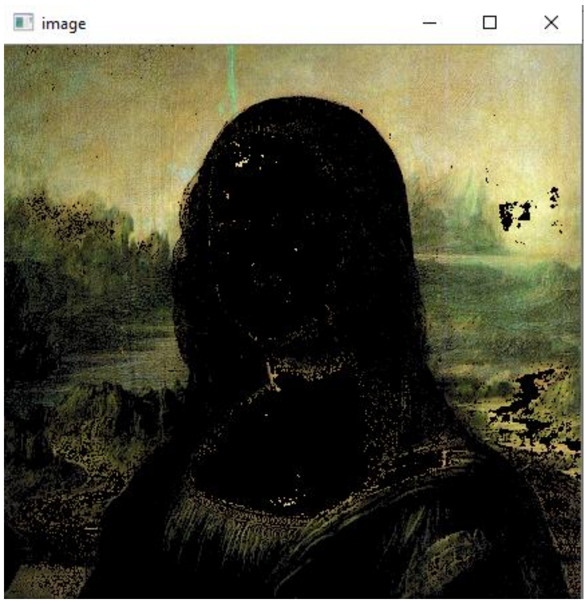


With Stroke img 2

Foreground



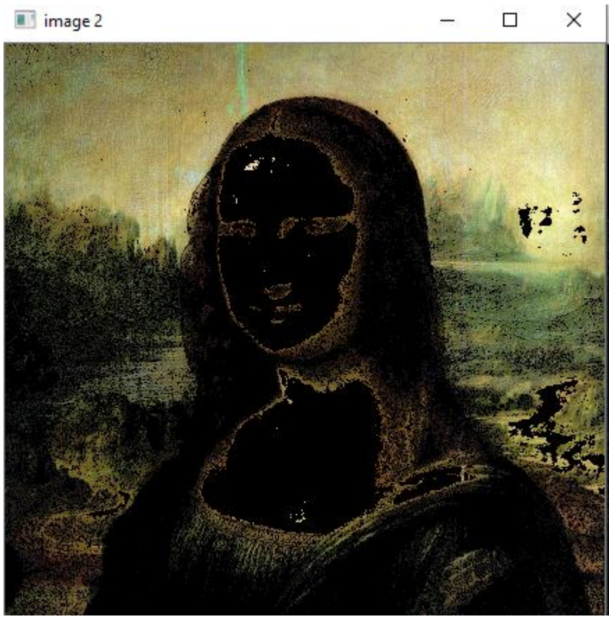
Background

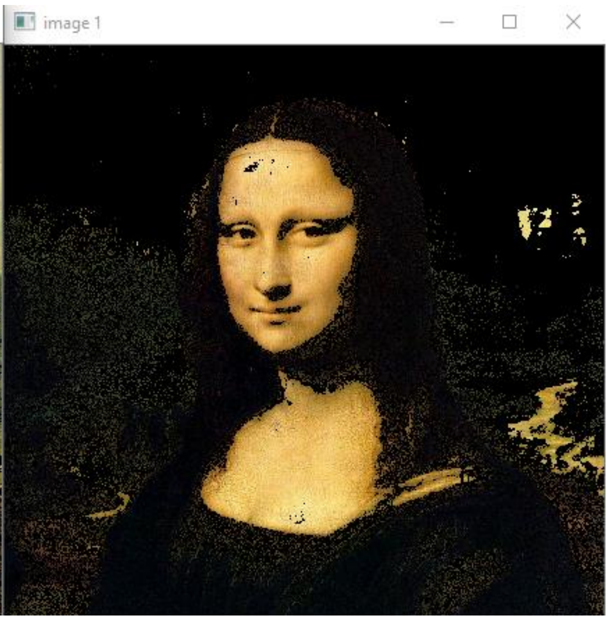


So, we saw that stroke image 2 gave us better results then stroke image 1.

For N = 10

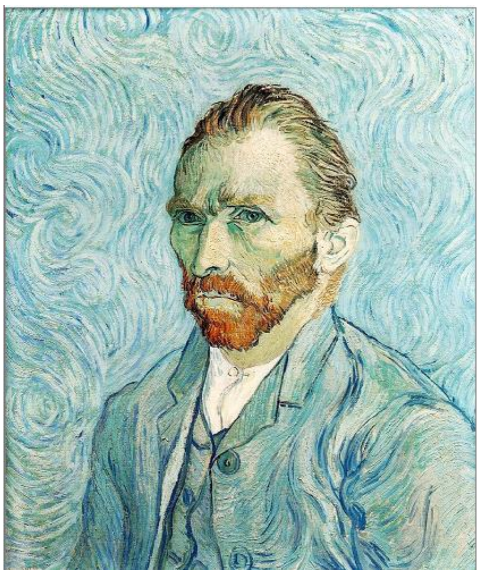
With Stroke img 1





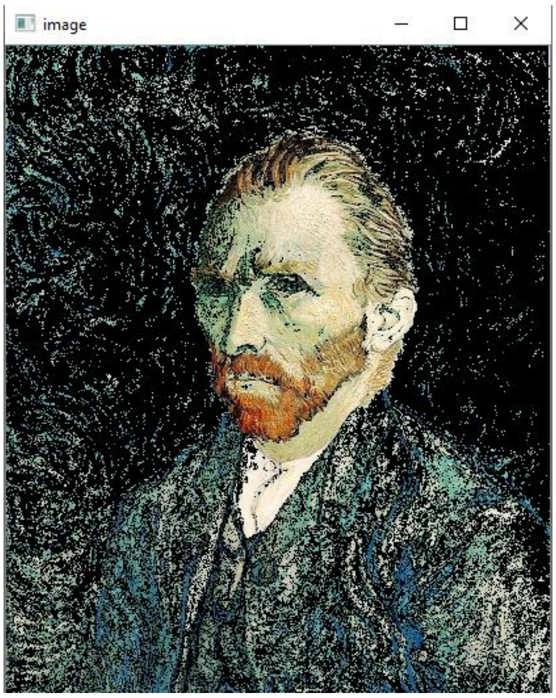
Van Gogh Img

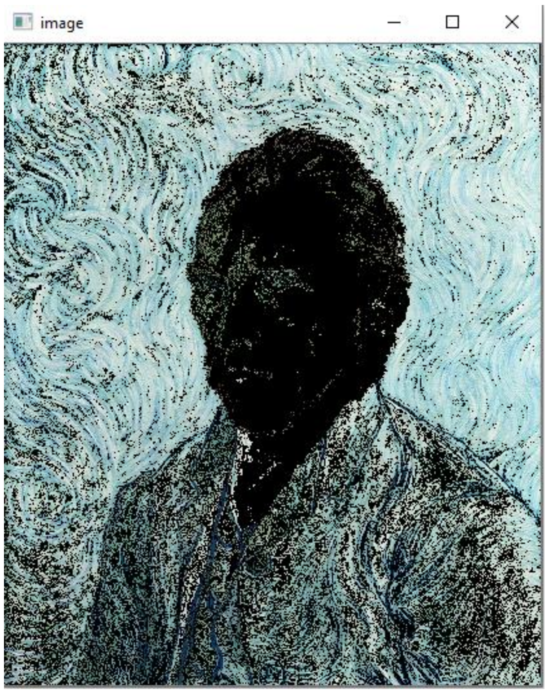
For N = 64



Foreground

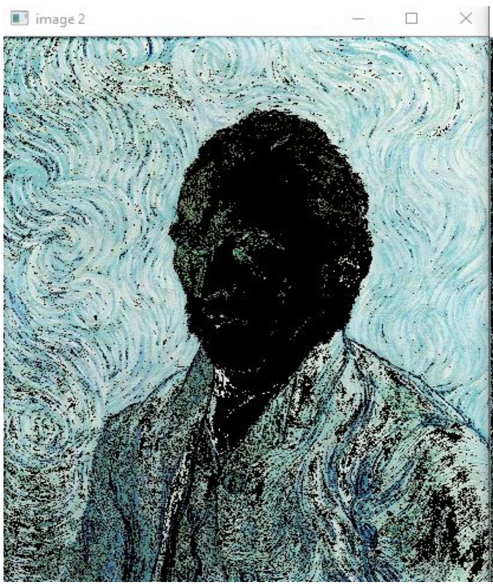
Background

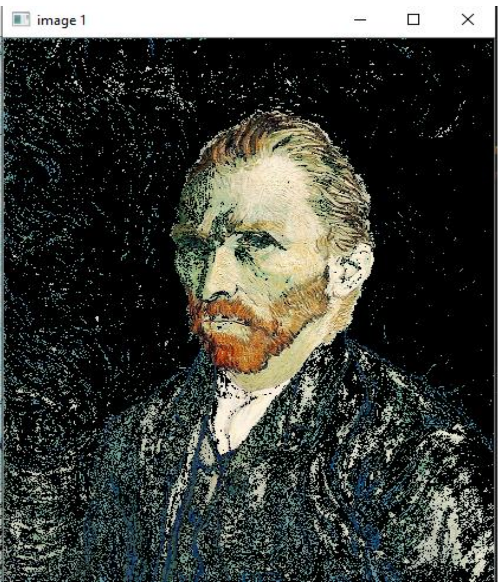




Despite the fact that foreground and background images have same colors, it did pretty well in segmenting face. However due to similarity of coat and background colors it struggle to segment them correctly

For N = 5





Code

Segmentation function

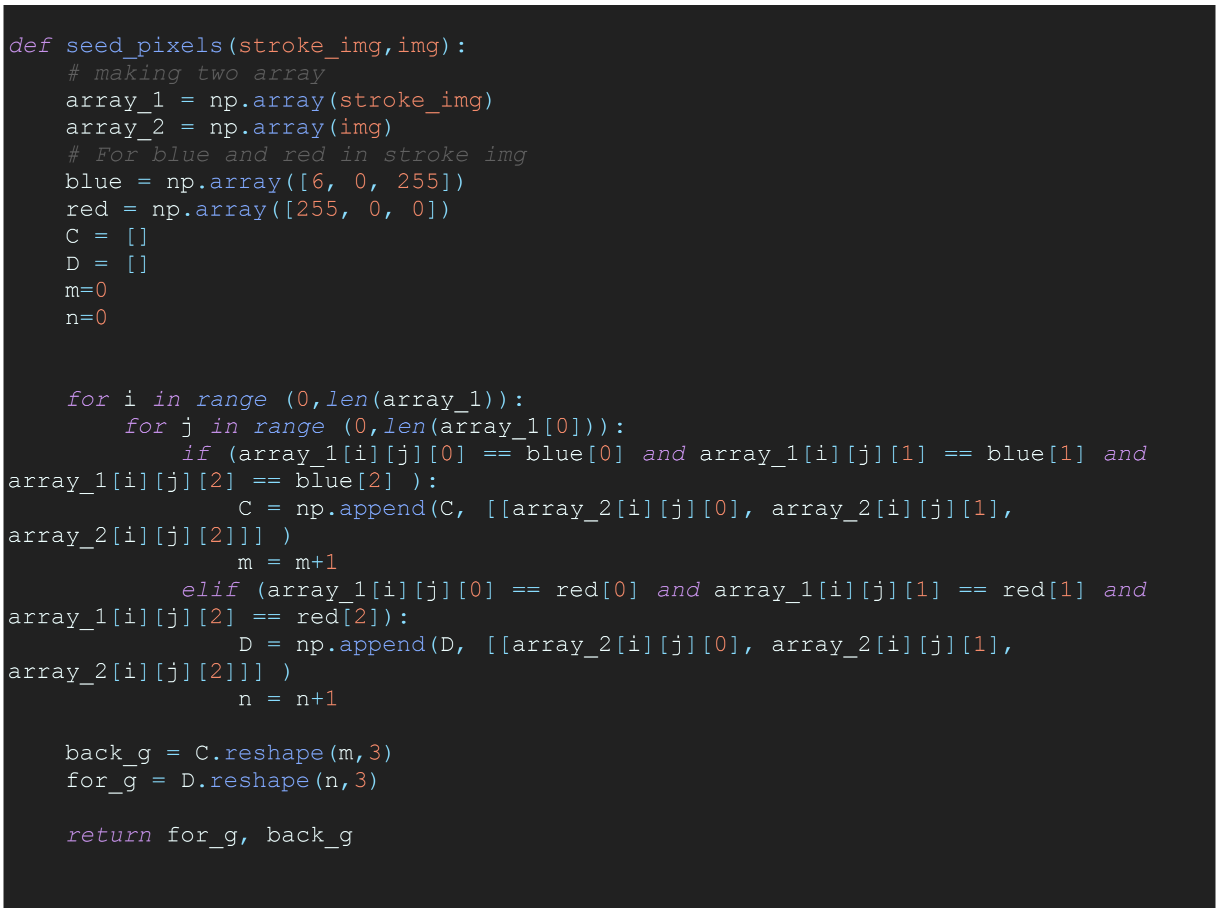
Segmentation function takes input of original image, stroke image and N.

It computes k mean of original image using stroke image and display results of foreground and background segmentation.

```
def Segmentation(img,stroke_img ,k):
    start = time.time()
    s_img = stroke_img
    for_g, back_g = seed_pixels(s_img, img)

    centroid_f, index_f = kmean(k, for_g)
    centroid_b, index_b = kmean(k, back_g)

    b = likelihood(img, centroid_f, index_f, centroid_b, index_b)
    display(b, img)

    end = time.time()
    print("process time in minutes")
    print((end-start)/60)
```

Seed pixel function

Seed pixel function gets pixels of stroke image whether it belong to foreground and background and return them as an array.



```
def seed_pixels(stroke_img,img):
    # making two array
    array_1 = np.array(stroke_img)
    array_2 = np.array(img)
    # For blue and red in stroke img
    blue = np.array([6, 0, 255])
    red = np.array([255, 0, 0])
    C = []
    D = []
    m=0
    n=0


    for i in range (0,len(array_1)):
        for j in range (0,len(array_1[0])):
            if (array_1[i][j][0] == blue[0] and array_1[i][j][1] == blue[1] and
array_1[i][j][2] == blue[2] ):
                C = np.append(C, [[array_2[i][j][0], array_2[i][j][1],
array_2[i][j][2]]] )
                m = m+1
            elif (array_1[i][j][0] == red[0] and array_1[i][j][1] == red[1] and
array_1[i][j][2] == red[2]):
                D = np.append(D, [[array_2[i][j][0], array_2[i][j][1],
array_2[i][j][2]]] )
                n = n+1

    back_g = C.reshape(m,3)
    for_g = D.reshape(n,3)

    return for_g, back_g
```

K-mean function

it assigns centroids randomly, assign index to each centroid and then takes average for a new centroids.

It keeps computing until new centroids are equal to old centroids or iteration is more than 200

```
def kmean(k,g):

    centroids = generate_centroids(k, g)
    cent = np.empty_like(centroids)
    iterations = 0
    while not np.array_equal(cent, centroids):
        if iterations <= 200:
            iterations += 1
            cent = centroids
            index = get_index(g, centroids)
            centroids = update_centroid(g, index, k)
        else:
            cent = centroids
    print('total number of iterations:')
    print(iterations)

    return centroids, index
```

Likelihood function

It calculates Likelihood of pixels of the image whether it belong to its foreground or background region and then return binary image in which 1 is assigned to higher probability of fore ground pixels and 0 is assigned for higher probability of back ground pixel.

Display function

It takes as input binary image and original image and display the results.

```python
def display(b,img):

    img_1 = np.array(img.copy())
    img_2 = np.array(img.copy())

    for i in range(len(b)):
        for j in range(len(b[0])):
            if b[i][j][0] == 0:
                img_1[i][j] = [0,0,0]
    for i in range(len(b)):
        for j in range(len(b[0])):
            if b[i][j][0] != 0:
                img_2[i][j] = [0,0,0]

    cv2.imshow('image', img)
    cv2.imshow('image 1', img_1)
    cv2.imshow('image 2', img_2)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```