EC – 413 Computer Vision
Lazy Snapping

Azka Rehman (arehman.ee38ceme) – 174227
Mushkbar Fatima (fmushkbar.ee38ceme) – 175540

May 17, 2020

Contribution to work

- Extraction of seed pixels - Mushkbar Fatima
- K means clustring -Azka Rehman
- Probability Function - Mushkbar Fatima
- Report - Azka Rehman

Contents

1 Problem Description

Algorithm of lazy snapping was required by implementing following functions:

- Extraction of seed pixels
- K means clustring
- Calculation of probability

2 Important Functions and Libraries

Coding is done in python.

Following libraries are used while coding this assignment.

- numpy
- math
- OpenCV
- sys
- matplotlib
- pyplot
- PIL
- time
- copy

3 Discussion of Code

3.1 Main Function

Main function takes image, stroke image and k as input and performs following functions:

- Extracts seed pixels
- Perform k means clustring on them
- Find probability for each pixel in an image
- Displays the results

It prints shape of array of seed pixels,total number of iteration for centroids updating of foreground and background. It displays foreground and background extraction of image and total time of lazy snapping on whole image.

3.2 Seed Pixels Extraction:

- Each pixel of a stroke image is traversed to check whether it's belonged to blue or red pixel.
- All the blue and red pixels of a stroke image are placed in separate arrays.

3.3 K means clustering:

There are three main steps for k means clustering.

- Assign centroids randomly.
- Assign index of centroids to seed pixels.
- Update centroids by taking average of seed pixels for each index of centroids.
- This function puts conditions on iterations as well as values of centroids. it will keep computing new centroids until old centroid becomes equal to new centroids. If iterations are more than 200 then the function will proceed with the last updated centroids.
- The indexing function also puts condition while averaging. If no index is assigned to a centroid, it will replace the value of that centroid with any pixel value of seed pixels. This is a prevention to avoid infinity values.

3.4 Calculation of probabilities:

- In this part, the likelihood of all the pixels of the image whether it belong to its fore ground or back ground region is calculated.
- And then a binary image is formed, in which 1 is assigned to higher probability of fore ground pixels and 0 is assigned for higher probability of back ground pixel.

3.4.1 Calculation

- K means function passed two arrays: an array of centroids and an array of data points assigned to respective cluster. For example, indexed 0 data point is assigned to 4th cluster.
- Further, an array consists of total number of data points in respective cluster and sum of that array is calculated. Later, it will be used for weight of each cluster.
- For loop is traversed to reach each pixel of an original image and each pixel is assigned to an 1x3 array iteratively for further calculations.
- Then again for loop is run to calculate the probability of pixel related to fore ground and background region in an image.
- And then binary image is formed based on higher probabilities assigned to pixel.

4 Code Listings

4.1 Importing Libraries

```
import numpy as np
import math
import cv2
import sys
from matplotlib import pyplot as plt
from PIL import Image
import array
import time
import copy
```

4.2 Main Function

Listing 1: Main function

```
def lazySnapping(image,si,k):
    s_image=si
    st=time.time()
    f,b=extract_seed_pixels(s_image,image)
    cf,i_f=kmeanalgo(k,f)
    cb,ib=kmeanalgo(k,b)
    bw=calc_prob(image,cf,i_f, cb, ib)
    img,fog,bag=mydisplay(bw,image)
    ed=time.time()
    print("process time in minutes")
    print((ed-st)/60)
    return fog,bag
```

4.3 Extraction of Seed Pixels

Listing 2: seed pixel extraction

```
def extract_seed_pixels(stroke_im,im):
    arr=np.array(stroke_im)
    arr1=np.array(im)
    blue =np.array([6, 0, 255])
    red =np.array([255, 0, 0])
    m=0
    n=0
    C=[]
    D=[]
    for i in range (0,len(arr)):
        for j in range (0,len(arr[0])):
            if (arr[i][j][0] == blue[0] and arr[i][j][1] == blue[1]
                                        and arr[i][j][2] == blue[2] ):
                C=np.append(C, [[arr1[i][j][0], arr1[i][j][1], arr1[i][j][2]]] )
```

```
            m=m+1
        elif (arr[i][j][0] == red[0] and arr[i][j][1] == red[1]
                                    and arr[i][j][2] == red[2]):
            D=np.append(D, [[arr1[i][j][0], arr1[i][j][1], arr1[i][j][2]]] )
            n=n+1
print(C)
bg=C.reshape(m,3)
fg=D.reshape(n,3)
print(bg.shape)
print(fg.shape)
return fg, bg
```

4.4 K Means Clustring

Listing 3: K means clustring

```
def kmeanalgo(k,arr):
    centroids=generate_centroids(k,arr)
    cent_1=np.empty_like(centroids)
    it=0
    while not np.array_equal(cent_1,centroids):
        if it<=200:
            it+=1
            cent_1=centroids
            index=get_index(arr,centroids)
            centroids=update_centroid(arr,index,k)
        else:
            cent_1=centroids
    print('total number of iterations:')
    print(it)
    return centroids,index
```

4.4.1 Generation of Random Centroids

Listing 4: Centroid Generation

```
def generate_centroids(k,arr):
    hi=0
    centroid=[]
    while hi < k*3:
        r=np.random.randint(np.amin(np.amin(arr)), np.amax(np.amax(arr)))
        if r not in centroid:
            hi += 1
            centroid=np.append(centroid,r)
    c=centroid.reshape(k,3)
    return c
```

4.4.2 Generation of Index

Listing 5: Indexing

```
def get_index(arr,cent):
    imd=[]
    for w in range (0,len(arr)):
        dis=[]
        for i in range (len(cent)):
            d=arr[w]-cent[i]
            dd=(np.linalg.norm(d))
            dis=np.append(dis,dd)
        imd=np.append(imd,np.argmin(dis))
    return imd
```

4.4.3 Updating Centroids

Listing 6: Centroid upgration

```
def update_centroid(arr,ind,k):
    avg=[]

    for i in range (k):
        hh=[]
        m=0
        for j in range(len(arr)):
            if ind[j]==i:
                hh=np.append(hh,arr[j])
                m+=1
        if len(hh)!=0:
            hh=hh.reshape(m,3)
            avg=np.append(avg,(sum(hh)/len(hh)))

        else:
            avg=np.append(avg,arr[i])

    avg=avg.reshape(k,3)
    return avg
```

4.5 Calculation of Probability

Listing 7: Calculation of Probability

```
def calc_prob(image,fore_cent, fore_dp_in_cent, back_cent, back_dp_in_cent):
    imag1 = image
    im1=np.array(imag1)
    print(im1.shape)

    fww=weight(fore_cent, fore_dp_in_cent)
    bww=weight(back_cent, back_dp_in_cent)

    b_w=np.zeros([len(im1), len(im1[0]),3])
    imgg=b_w
```

```
    p1=np.zeros([1,3])
    for i in range(0, len(im1)):
        for j in range(0, len(im1[0])):
            p1[0][0]=im1[i][j][0]
            p1[0][1]=im1[i][j][1]
            p1[0][2]=im1[i][j][2]
            add=0

            for k in range(0, len(fore_cent)):
                wk= fww[k]
                norm1=(np.linalg.norm(p1-fore_cent[k]))
                expo=math.exp(-(norm1))
                add=add+ (wk* expo)
            fore_ground=add
            add_1=0

            for m in range(0, len(back_cent)):
                wk_1= bww[m]
                norm_1=(np.linalg.norm(p1-back_cent[m]))
                expo_1=math.exp(-norm_1)
                add_1=add_1+ (wk_1* expo_1)
            back_ground=add_1

            if fore_ground > back_ground:
                b_w[i][j]=[255,255,255]

            else:
                b_w[i][j]=[0,0,0]
    return b_w
```

4.5.1 Calculation of Weights

```
def weight(cent,index):
    tsp=len(index)
    k=len(cent)
    num=[]
    for i in range (k):
        l=0
        for j in range (tsp):
            if index[j]==i:
                l+=1
        num=np.append(num,l)
    weigh=num/tsp
    return weigh
```

4.6 Display

Listing 8: Display of results

```
def mydisplay(bw,image):
```

```
im=np.array(image.copy())
im1=np.array(image.copy())
for i in range (len(bw)):
    for j in range (len(bw[0])):
        if bw[i][j][0]==0:
            im[i][j]=[0,0,0]
for i in range (len(bw)):
    for j in range (len(bw[0])):
        if bw[i][j][0]!=0:
            im1[i][j]=[0,0,0]
dst = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
dst1 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
dst2=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(dst2)
plt.show()
plt.imshow(dst)
plt.show()
plt.imshow(dst1)
plt.show()
return dst2,dst,dst1
```

5 Results

5.1 Lady

For Lady stroke 1:
k=31

If k is less i.e. 5, we can see a clear result in back ground and foreground but feet of lady are not visible.
If K is increased e.g. 64 background and foreground difference doesn't appear to be good enough. So, k=31 was selected after hit and trial. Feet of lady are visible too and difference of foreground and background was good enough.

For Lady stroke 2:
k=31

Background inside arms of lady is more separated. It is visible from figure 2 and 3 that results are much better for strokes 2 around arms of lady. k=31 was chosen after hit and trial as it gives better results.

Figure 1: Lady

(a) Lady Foreground

(b) Lady Background

Figure 2: Lazy Segmentation using strokes 1

(a) Lady Foreground

(b) Lady Background

Figure 3: Lazy Segmentation using strokes 2

5.2 Van Gogh

k=64

Selection of less k results in poor results as this figure contains many colours so, more value of k results in more clusters of colors.

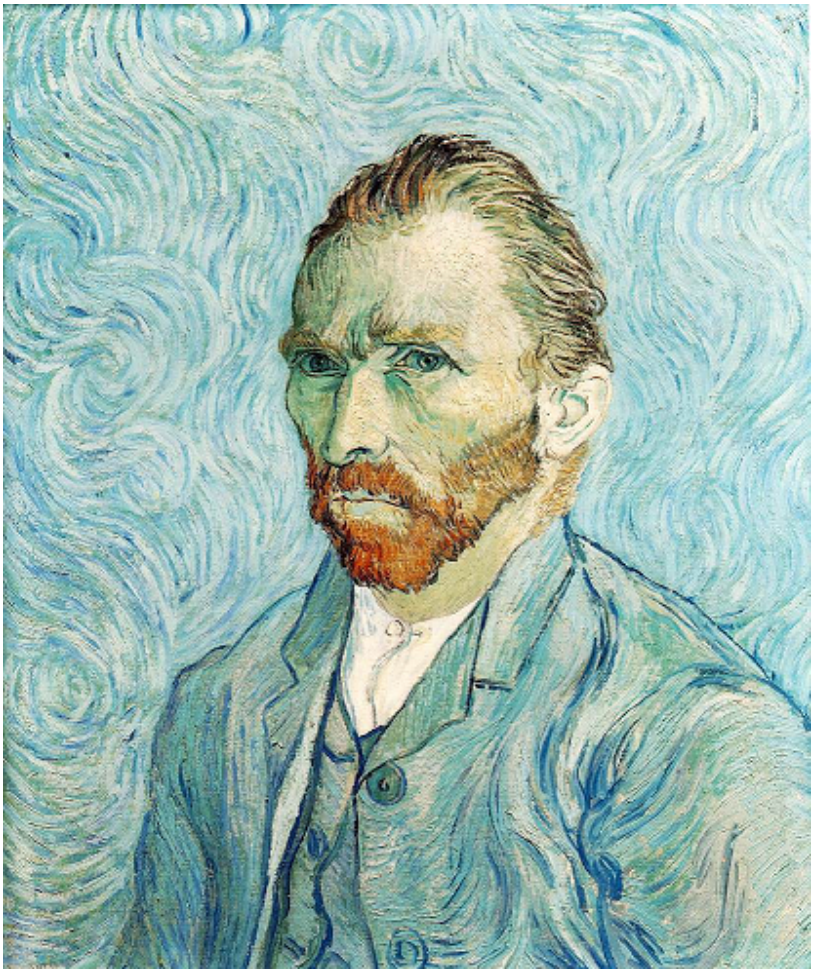
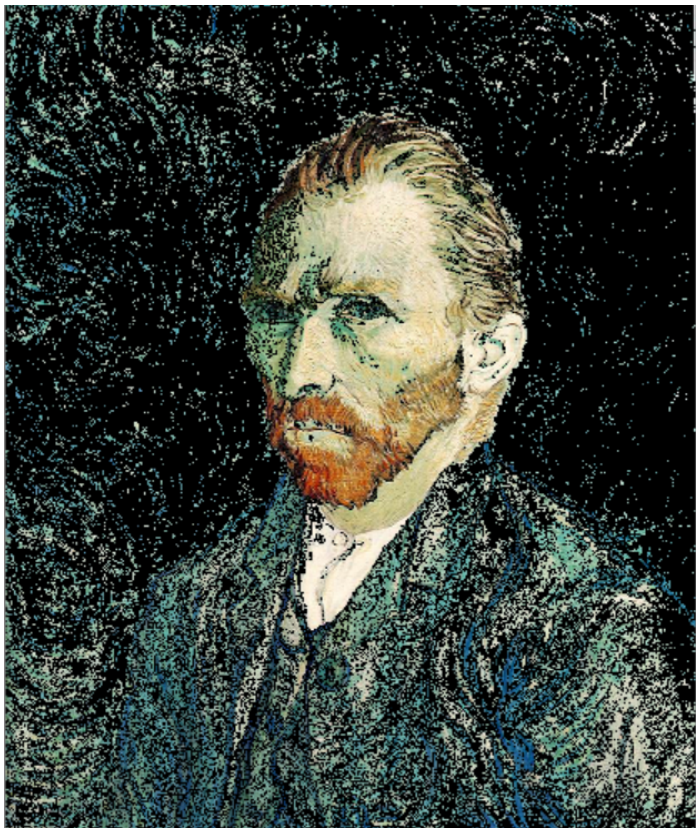


Figure 4: Van Gogh



(a) van Gogh foreground

(b) van Gogh Background

Figure 5: Lazy Segmentation of van Gogh

5.3 Dog

k=64

Selection of less k results in poor results as this figure contains many colours so, more value of k results in more clusters of colors.

Figure 6: Dog

(a) Dog foreground

(b) Dog Background

Figure 7: Lazy Segmentation of Dog

5.4 Mona-lisa

For Mona-Lisa stroke 1:
k=51

If K is increased e.g. background and foreground difference appear to be good enough.
So, k=51 was selected after hit and trial. Difference of foreground and background was good enough.

For Mona-lisa stroke 2:
k=51

Different strokes gives different results. Increasing strokes increases the process time but gives better results as seen in figures below.

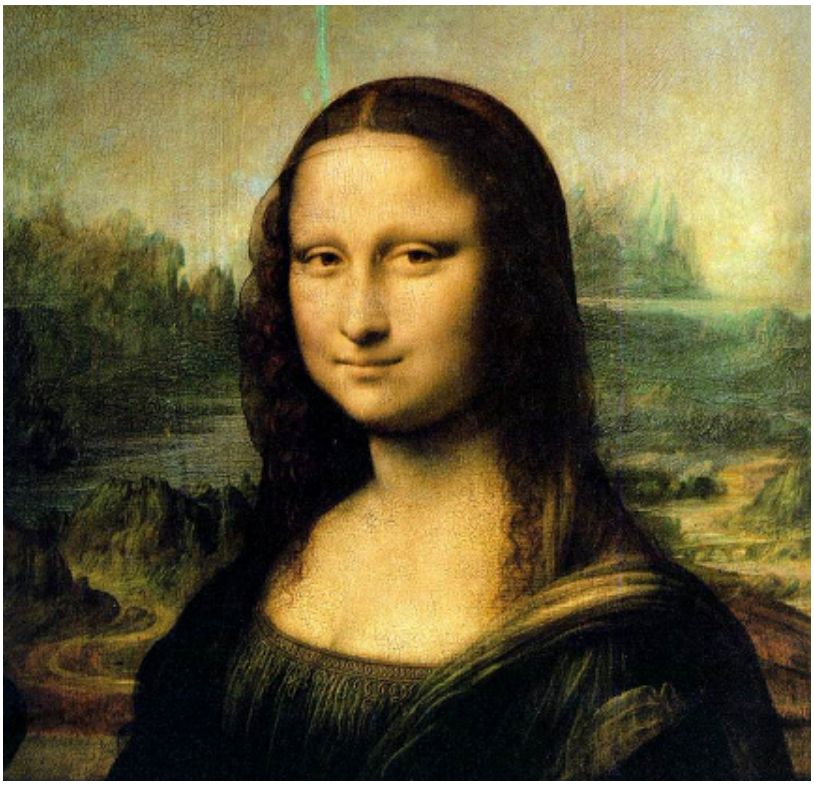
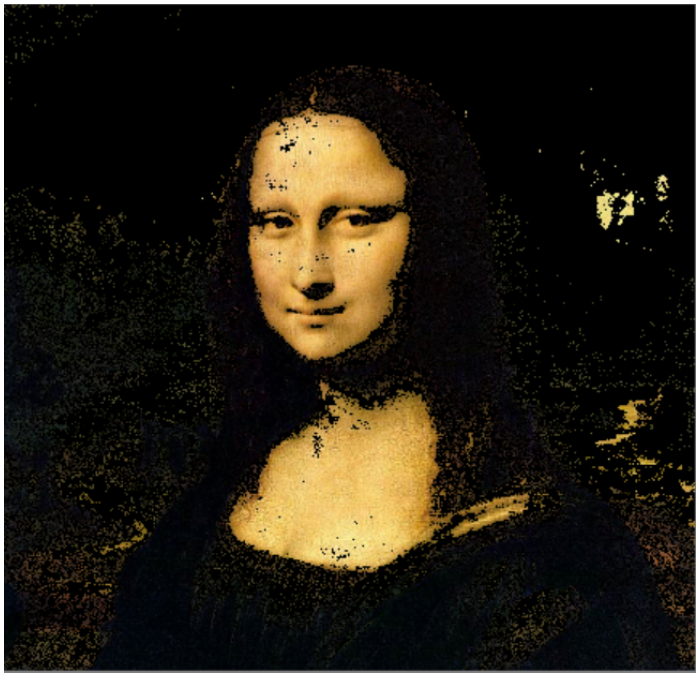


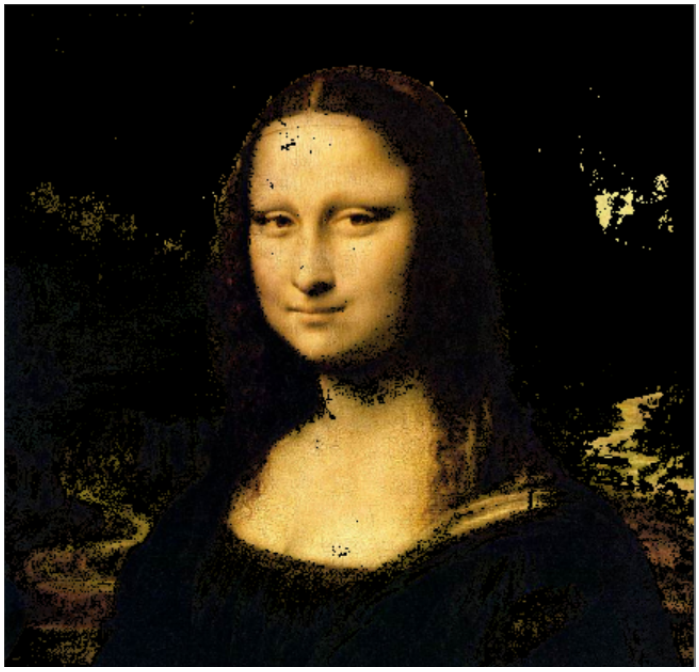
Figure 8: Mona-lisa



(a) Lady Foreground

(b) Lady Background

Figure 9: Lazy Segmentation using strokes 1



(a) Mona-lisa Foreground

(b) Mona-lisa Background

Figure 10: Lazy Segmentation using strokes 2

5.5 Dancing Lady

For Dancing Lady stroke 1:
k=29

If K is increased e.g. 29 background and foreground difference appear to be good enough. So, k=29 was selected after hit and trial. Difference of foreground and background was good enough.

For Dancing Lady stroke 2:
k=31

Different strokes gives different results. Increasing strokes increases the process time but gives better results as seen in figures below.

Figure 11: Dancing Lady

(a) Dancing Lady Foreground

(b) Dancing Lady Background

Figure 12: Lazy Segmentation using strokes 1

(a) Dancing Lady Foreground

(b) Dancing Lady Background

Figure 13: Lazy Segmentation using strokes 2

6 Important Findings

Following are some important observations done while implementing the algorithm of given task:

- Some images give better result for less k because they contain less colours. Other images produce better results with higher k.
- Increasing strokes gives better results but also increases the process time.
- Process time increases with increase in k.
- Process time is less for images with less colour variation.
- Working with arrays also increases the process time as compared to working with lists in a loop. For example if a function takes 0.02 seconds for a loop with lists.append, it takes 0.8 seconds for the same function for a loop with array.append.