CS-867 Spring 2021

Computer Vision

Assignment # 01

Interactive Foreground Segmentation

Instructor: Dr. Shahzor Ahmed

Submitted by: Monazza Qadeer Khan

Id: 206154

Dated: 3rd June, 2021

Interactive Foreground Segmentation

Important information regarding this assignment:

1) I have done this assignment using Google Colab. I've downloaded my notebook which is cotained in the zip folder as assignment_2.py. There's one other file named functions which has all the functions related to the tasks performed in this assignment.

2) I was facing some issue in file handling and reading of functions file, so, I created a github repository for that file and cloned it to colab notebook.

3) I have pasted the link below to my Drive folder which has all the files related to this assignment. It can be accessed through this link directly.

https://drive.google.com/drive/folders/1yX2_Ixp1AQU4UyrLpOpbHMqhBhiiobRB?usp=sharing

4) I first tried to crop out foreground for a patch size of 40×40 but as that patch corresponded to background in all the images, so, it returned background for actual background and for foreground it

returned zero value. It can be seen in the results section at the end of this document. The code in this report is for patch size of 40×40.

5) I generalized the code for any given image size but when I ran it in Colab it took hours even when I changed the hardware accelerator to TPU but still did not give the required results. The Colab notebook contains the generalized code, it can be seen there.

Code Specifications:

1) There are two files for this assignment, the colab notebook named **main_prog.ipynb**, it performs the required tasks of this assignment and a functions file named **functions.py**, in which all the functions used in main_prog are scripted.
2) No built-in function has been used, all functions are written from scratch. Function for each task is scripted in the functions.py file.
3) In the **main_prog file**, I have imported stroke image and input image on which the required task has to be performed. All stroke images are being stored in **`strokeimgs`** list and similarly, the input images are stored in **`oimgs.`** A loop is run in such a way that in each iteration a stroke image and input image is picked up.
4) Number of clusters used, k=64.

5) Description of functions:

- **`r_b_pixels:`** It is used to get red & blue stroke value indices from stroke images.
- **`Kmeans():`** It is used for red indexed pixels to get centroid and data points assigned to each cluster. The same is done for blue indexed pixels.
- **`Wk(), Ck():`** These are used to get weights of red pixel centroid i.e. it gives centroid values in original image. The same is done for blue pixels as well.
- **`p_of_oimPixels():`** It calculates probability of red pixels(foreground) in original image. Probability for blue pixels i.e. background is also found.
- **`fg_bg_assign():`** After assigning probabilities to both red and blue pixels, this function is used to assign label 1 if probability of red pixel is greater than that of blue pixel and label is zero is assigned if blue pixel is more probable.
- **`Show_fg():`** After each pixel is assigned its respective label either foreground (fg) or background (bg), two copies of original image are made i.e. one for each fg & bg, then this function is used to display image if the pixel values belong to fg, if it does not belong to fg then pixel label is replaced with 0 i.e. bg and display this modified image by .show() function.

- **`Show_bg:`** Above is repeated for blue pixel values, if the pixels belong to background they are displayed with this function if not then the pixel label is modified to 1 i.e. fg and display the modified image using .show() function.

`6) functions.py:` No built-in function is used in this assignment, all functions are written from scratch, they are briefly described above.

Code snippets can be seen in the following part of this document. Each command is accompanied by a brief description.

Code:

The command below mounts the drive contents to google colab's files section. I created a folder named Assignment_2_folder and all the content related to this assignment resides in this folder, it contains all the images files and this notebook.

```
[ ] from google.colab import drive
    drive.mount('/content/gdrive')
```

When we run above command we get the following link.

When we go to this link it asks for our permission to login to our Google Drive. We select our account and then we get a link to our Drive, we copy that link and paste it in above space. This way our Drive gets uploaded.

Drive has been uploaded as seen below.

Below command shows that drive has been uploaded.

```
[2] from google.colab import drive
    drive.mount('/content/gdrive')
```

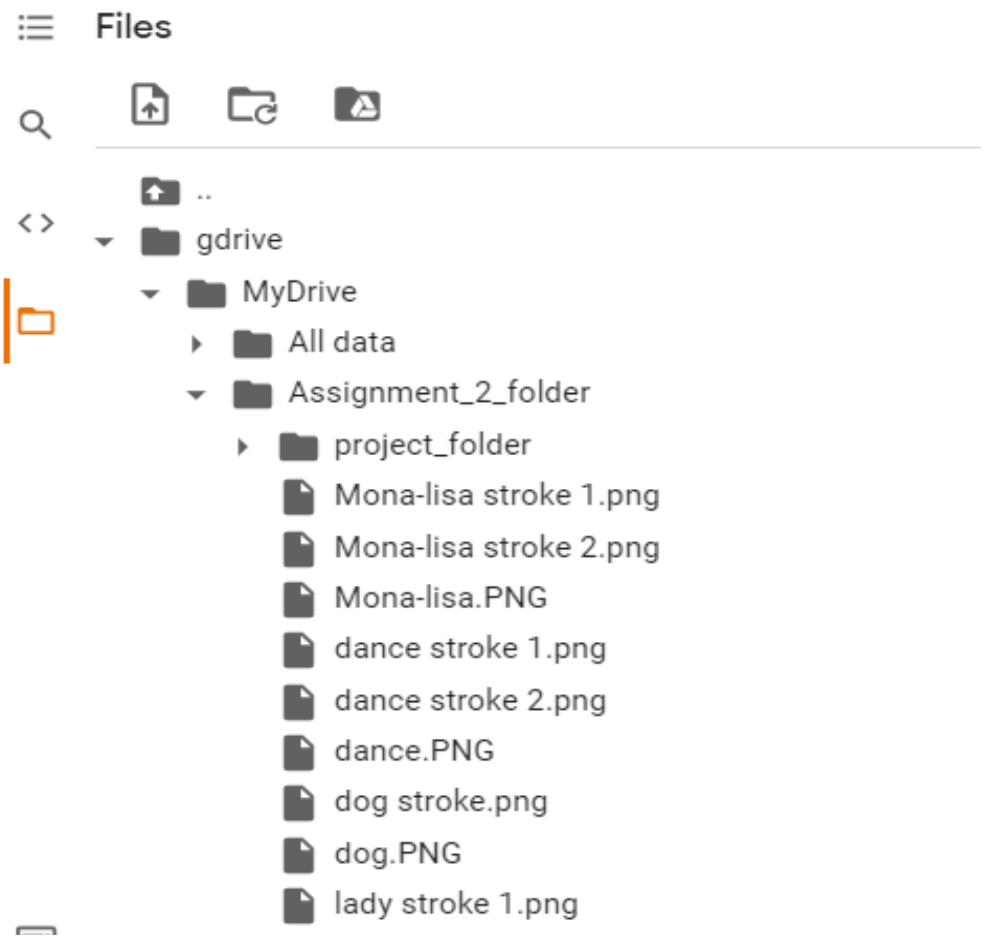
```
Mounted at /content/gdrive
```

Below command shows the contents that have been mounted to the files section of this notebook. It's just for confirmation.

```
[3] ! ls
```

```
gdrive  sample_data
```

Files section showing **Drive** contents.



This is a magic command, it changes our working directory. As I would be working in my Assignment_2_folder, so, I use this command before I start working.

```
[ ] %cd gdrive/My Drive/Assignment_2_folder
```

```
/content/gdrive/My Drive/Assignment_2_folder
```

This is a bit tricky part. Actually, I created a Github repository for this assignment as I was having some issue in file handling and reading directly from Google Colab.So, I created git repo and cloned it here. But we do it only once, when our required folder is cloned or copied in the destination folder it stays there unless you delete it yourself.

```
[ ] ! git clone https://github.com/monazza-qk92/project_folder.git
```

```
Cloning into 'project_folder'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (8/8), done.
```

Importing functions from functions.py file:

Now, we're done with the files uploading, so, the next thing is to import the functions file from the folder named "project_folder", which is inside our working directory i.e. Assignment_2_folder.

```
[6]  from project_folder.functions import *
```

All the functions are scripted in functions.py file, these functions are imported main main_prog.ipynb file using above command. As this functions file resides inside project_folder, thus, using project_folder.functions command to import from that folder. As I was having some error when I directly tried to import functions file to my notebook. For this reason, I created git repo and copied the required file from it.

Then we run the main code which is as below.

Main_prog.ipynb

```
from PIL import Image

strokeimgs = ['dance stroke 1.png','dance stroke 2.png','dog stroke.png',
              'lady stroke 1.png','lady stroke 2.png','Mona-lisa stroke 1.png',
              'Mona-lisa stroke 2.png','van Gogh stroke.png']
oimgs = ['dance.PNG','dance.PNG','dog.PNG','lady.PNG','lady.PNG',
         'Mona-lisa.PNG','Mona-lisa.PNG','van Gogh.PNG']

for ii in range(8):
     strokeimg = Image.open(strokeimgs[ii])
     oimg = Image.open(oimgs[ii])
     k = 64

     r_df, b_df,xs = r_b_pixels(strokeimg)
    #for red pixels i.e foreground
     rdf, rcentroids = kmeans(k, r_df)
     rwk = Wk(rcentroids,k, rdf,xs)
     rCkval = Ck(k,oimg,rcentroids)
     r_prob,oxs,oys = p_of_oimPixels(oimg,k,rCkval,rwk)
```

```
#for blue pixels i.e background
 bdf, bcentroids = kmeans(k, b_df)
 bwk = Wk(bcentroids,k, bdf,xs)
 bCkval = Ck(k,oimg,bcentroids)
 b_prob,oxs,oys = p_of_oimPixels(oimg,k,bCkval,bwk)

#assigning
 assign = fg_bg_assign(r_prob,b_prob)

 fgImg_copy = oimg.copy()
 bgImg_copy = oimg.copy()
 #show foreground image
 fg_oimg = show_fg(oxs,oys,fgImg_copy,assign)
 fg_oimg.show()
#show background image
 bg_oimg = show_bg(oxs,oys,bgImg_copy,assign)
 bg_oimg.show()
```

functions.py

```python
import matplotlib.pyplot as plt
import math
import numpy as np
import pandas as pd
import copy
def kmeans(k, df):
    max_x = max(df['x'])
    max_y = max(df['y'])
    # centroids[i] = [x, y]
    centroids = {
        i+1: [np.random.randint(0, max_x), np.random.randint(0, max_y)]
        for i in range(k)
    }
    print(centroids)
    ## Assignment Stage
    def assignment(df, centroids):
        for i in centroids.keys():
            # sqrt((x1 - x2)^2 - (y1 - y2)^2)
            df['distance_from_{}'.format(i)] = (
                np.sqrt(
                    (df['x'] - centroids[i][0]) ** 2
                    + (df['y'] - centroids[i][1]) ** 2
                )
            )
        centroid_distance_cols = ['distance_from_{}'.format(i) for i in centroids.keys()]
        df['closest'] = df.loc[:, centroid_distance_cols].idxmin(axis=1)
        df['closest'] = df['closest'].map(lambda x: int(x.lstrip('distance_from_')))
        return df
    df = assignment(df, centroids)
    print(centroids)
    ## Update Stage
    old_centroids = copy.deepcopy(centroids)
    def update(k):
        for i in centroids.keys():
            me = np.mean(df[df['closest'] == i]['x'])
            if(me>0):
                centroids[i][0] = me
            mee = np.mean(df[df['closest'] == i]['y'])
            if (mee > 0):
                centroids[i][1] = mee
        return k
    centroids = update(centroids)
    print(centroids)
    ## Repeat Assigment Stage
    df = assignment(df, centroids)
```

```
    # Continue until all assigned categories don't change any more
    while True:
        closest_centroids = df['closest'].copy(deep=True)
        centroids = update(centroids)
        df = assignment(df, centroids)
        if closest_centroids.equals(df['closest']):
            break
    print(centroids)
    return df, centroids

#r_b extration
def r_b_pixels(strokeimg):
    [xs, ys] = strokeimg.size
    r_xind = []
    r_yind = []
    b_xind = []
    b_yind = []
    rindalt=0
    bindalt=0
    for x in range(0, xs):
        for y in range(0, ys):
            # (4)  Get the RGB color of the pixel
            [r, g, b] = strokeimg.getpixel((x, y))
            if (r == 255):
                r_xind.insert(rindalt, x);
                r_yind.insert(rindalt, y);
                rindalt = rindalt + 1;
            if (b == 255):
                b_xind.insert(bindalt, x);
                b_yind.insert(bindalt, y);
                bindalt = bindalt + 1;

    r_df = pd.DataFrame({'x': r_xind, 'y': r_yind})
    b_df = pd.DataFrame({'x': b_xind, 'y': b_yind})
    return r_df, b_df,xs

#Computing Wk
def Wk(xcentroids,k, xdf,xs):
    lenCent = []
    wk = []
    lindalt = 0
    wkindalt=0
    centroiddatacontn = copy.deepcopy(xcentroids)
    for x in range(k):
        centroiddatacontn[x + 1][0] = (xdf[xdf['closest'] == x + 1]['x'])
        lenCent.insert(lindalt, len(centroiddatacontn[x + 1][0]));
        lindalt = lindalt + 1;
    for x in range(k):
        w = lenCent[x] / xs;
        wk.insert(wkindalt, w);
        wkindalt = wkindalt + 1;
    return wk

```

```
#for Ck pixels
def Ck(k,oimg,xcentroids):
    Ckval = []
    Ckvalind = 0
    for x in range(k):
        Ckval.insert(Ckvalind, oimg.getpixel((xcentroids[x+1][0],xcentroids[x+1][1])))
        Ckvalind = Ckvalind + 1;
    return Ckval

#p
def p_of_oimPixels(oimg,k,Ckval,wk):
    [oxs, oys] = oimg.size
    IpMinusCk = []
    IpMinusCkindex = 0
    prob = []
    pindex = 0
    for x in range(100, 140):
      for y in range(140, 180):
         [r,g,b] = oimg.getpixel((x, y))
         for z in range(k):
            [rr, gg, bb] = Ckval[z]
            dist = (r-rr)** 2+(g-gg)** 2+(b-bb)** 2

            # dist = numpy.linalg.norm(a - b)
            expval = math.exp(-1*(dist))
            p = wk[z]*(expval);
            IpMinusCk.insert(IpMinusCkindex, p);
            IpMinusCkindex = IpMinusCkindex +1;
         IpMinusCkindex=0;
         prob.insert(pindex, sum(IpMinusCk))
         pindex = pindex + 1
      pindex = 0
    print(prob)
    return prob,oxs,oys

def fg_bg_assign(r_prob,b_prob):
    assign = []
    for a in range(len(r_prob)):
        if (r_prob[a]>b_prob[a]):
            assign.insert(a, 1);
        else:
            assign.insert(a, 0);
    return assign

```

```
143  def show_fg(oxs,oys,oimg,assign):
144      pixind = 0;
145      for x in range(100, 140):
146          for y in range(140, 180):
147              if (assign[pixind] == 0):
148                  oimg.putpixel((x, y), 0)
149              pixind = pixind + 1
150      return oimg
151
152
153  def show_bg(oxs,oys,oimg,assign):
154      pixind = 0;
155      for x in range(100, 140):
156          for y in range(140, 180):
157              if (assign[pixind] == 1):
158                  oimg.putpixel((x, y), 0)
159              pixind = pixind + 1
160      return oimg
```

Results

Original Image: dance.PNG
Stroke Image: dance stroke 1.png

Background detected in 40x40.

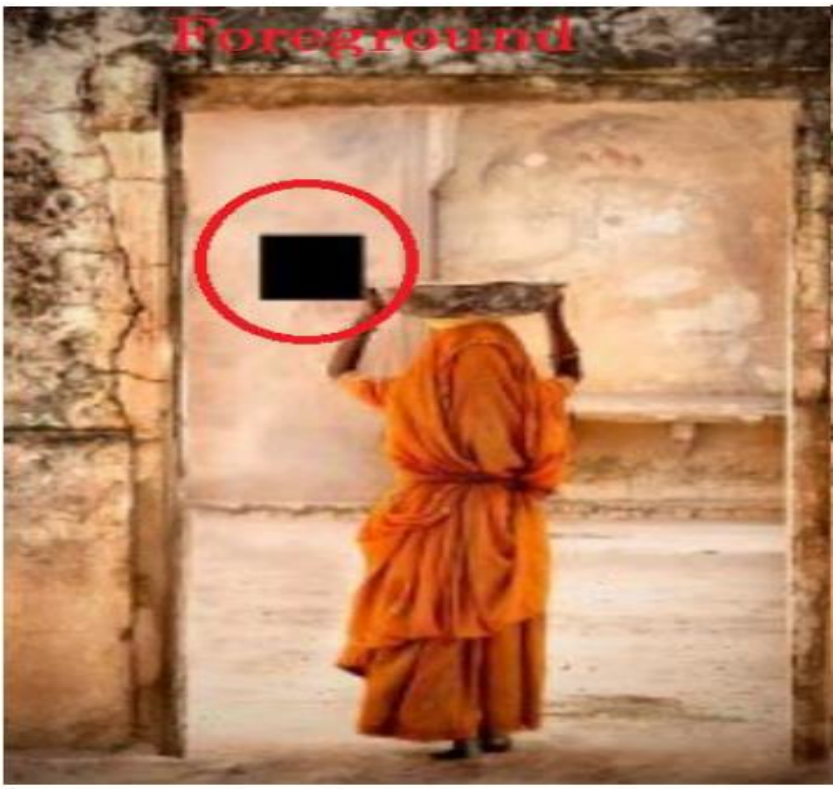
Original Image: dance.PNG
Stroke Image: dance stroke 2.png

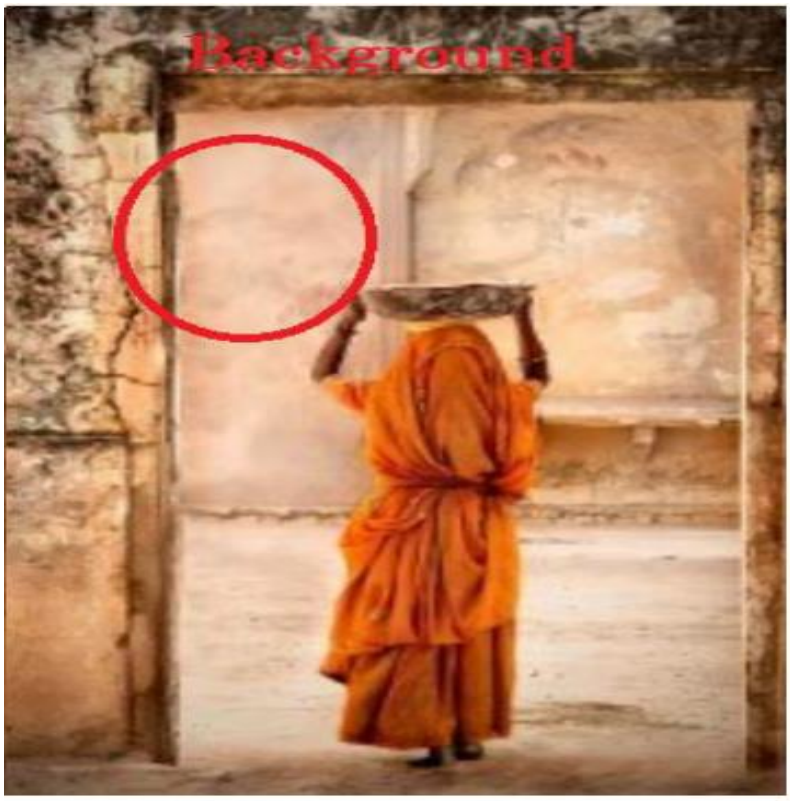
Background detected in 40x40.

Original Image: dog stroke.PNG
Stroke Image: dog.png

Background detected in 40x40.

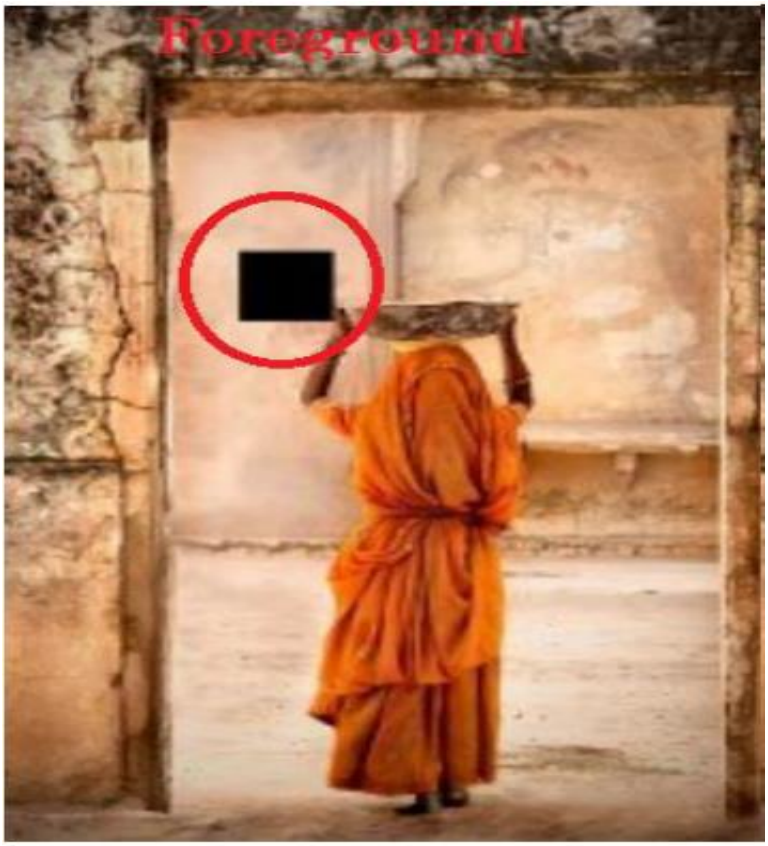
Original Image: lady.PNG
Stroke Image: lady stroke 1.png

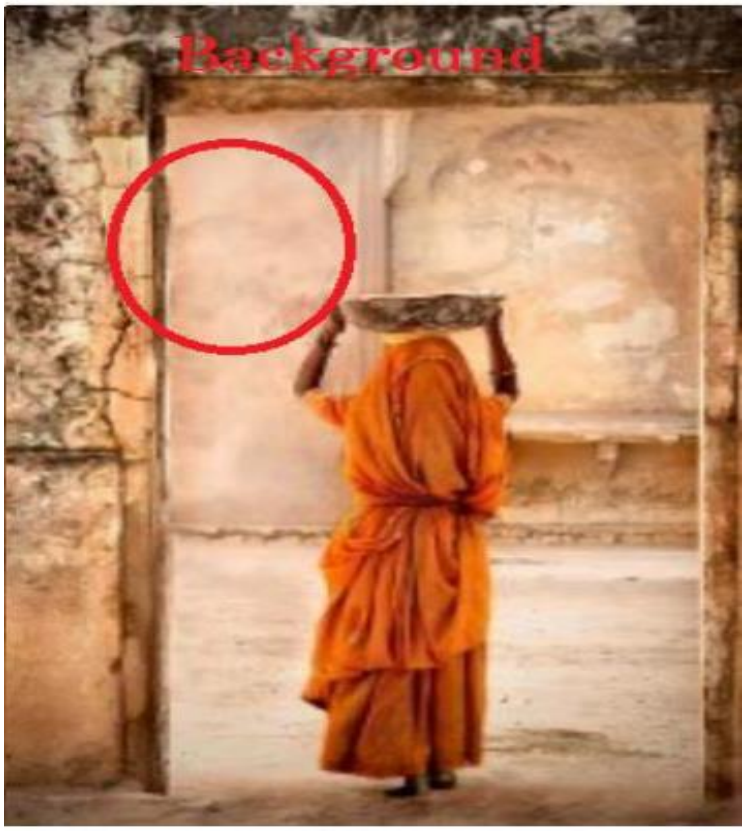




Background detected in 40x40.

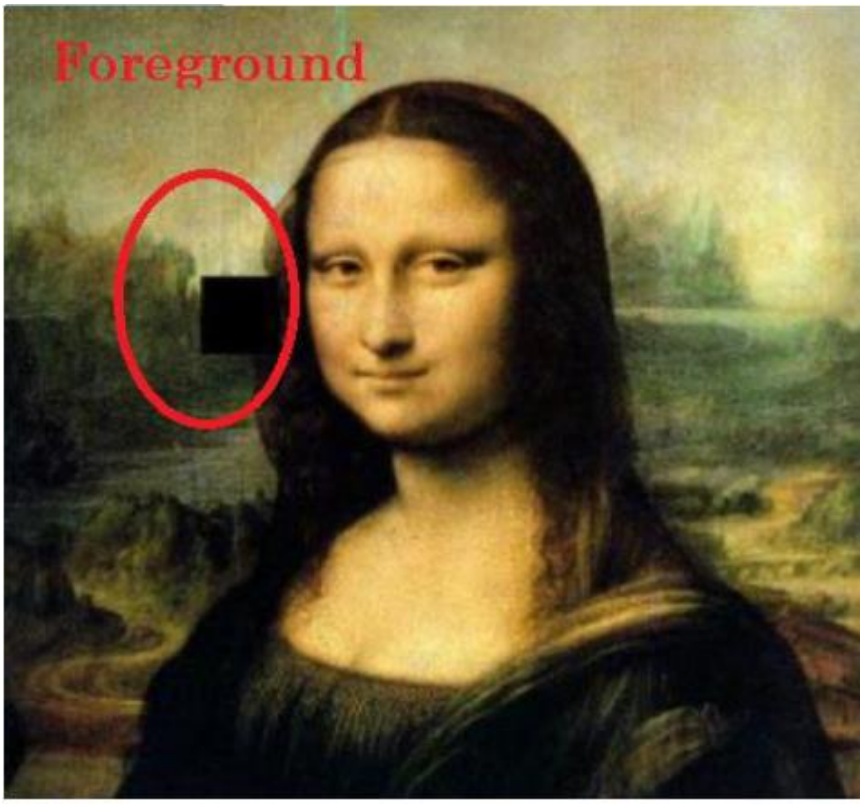
Original Image: lady.PNG
Stroke Image: lady stroke 2.png





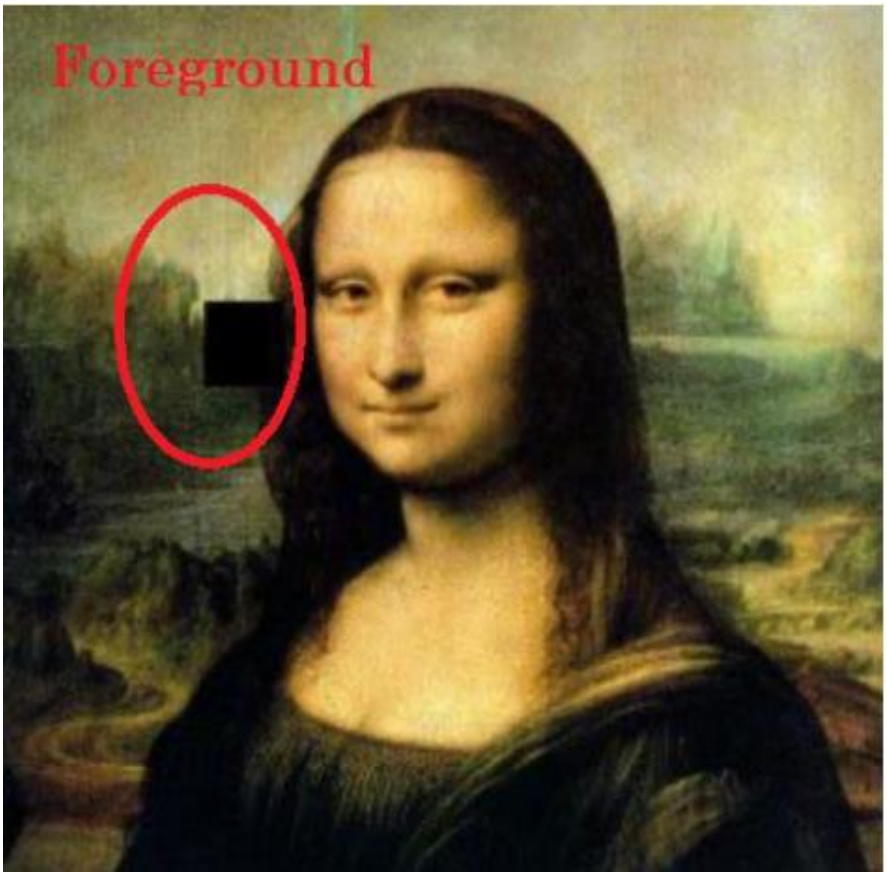
Background detected in 40x40.

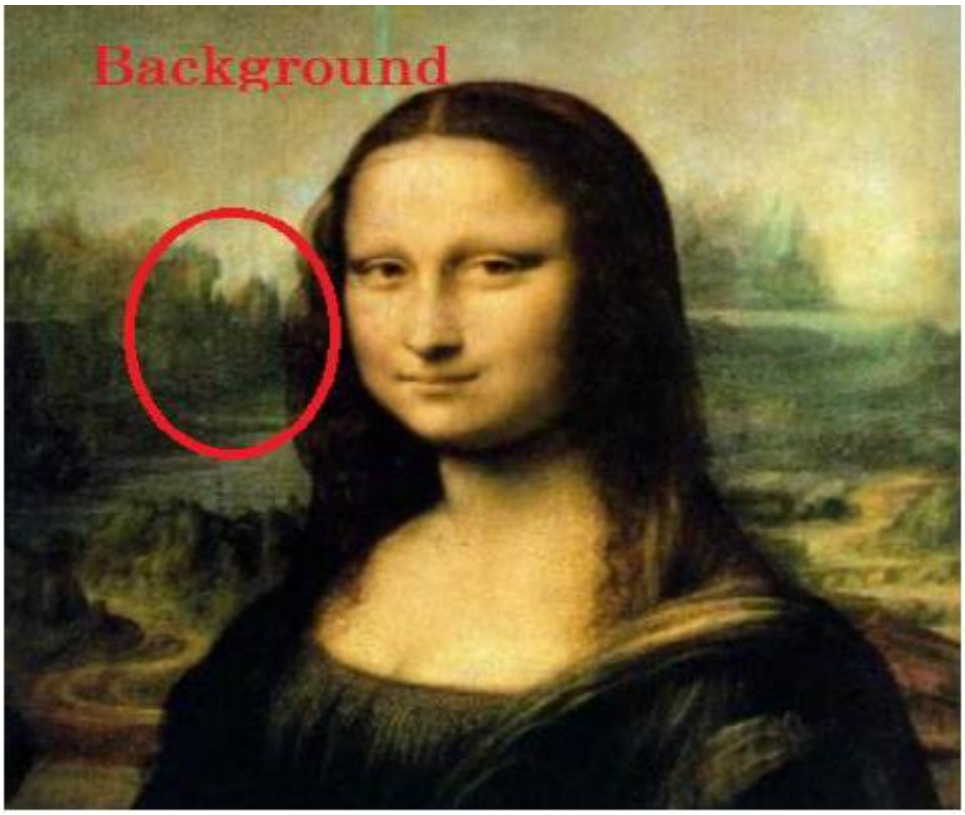
Original Image: Mona-lisa.PNG
Stroke Image: Mona-lisa stroke 1.png



Background detected in 40x40.

Image: Mona-lisa.PNG
Stroke Image: Mona-lisa stroke 2.png

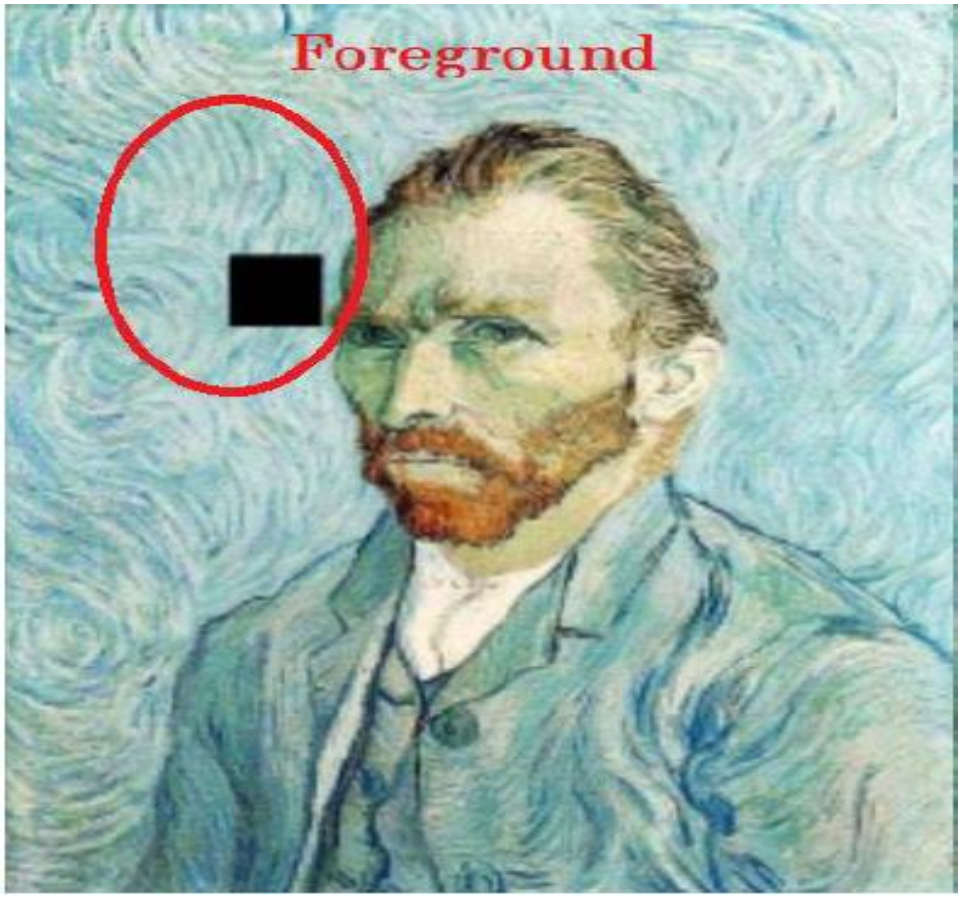


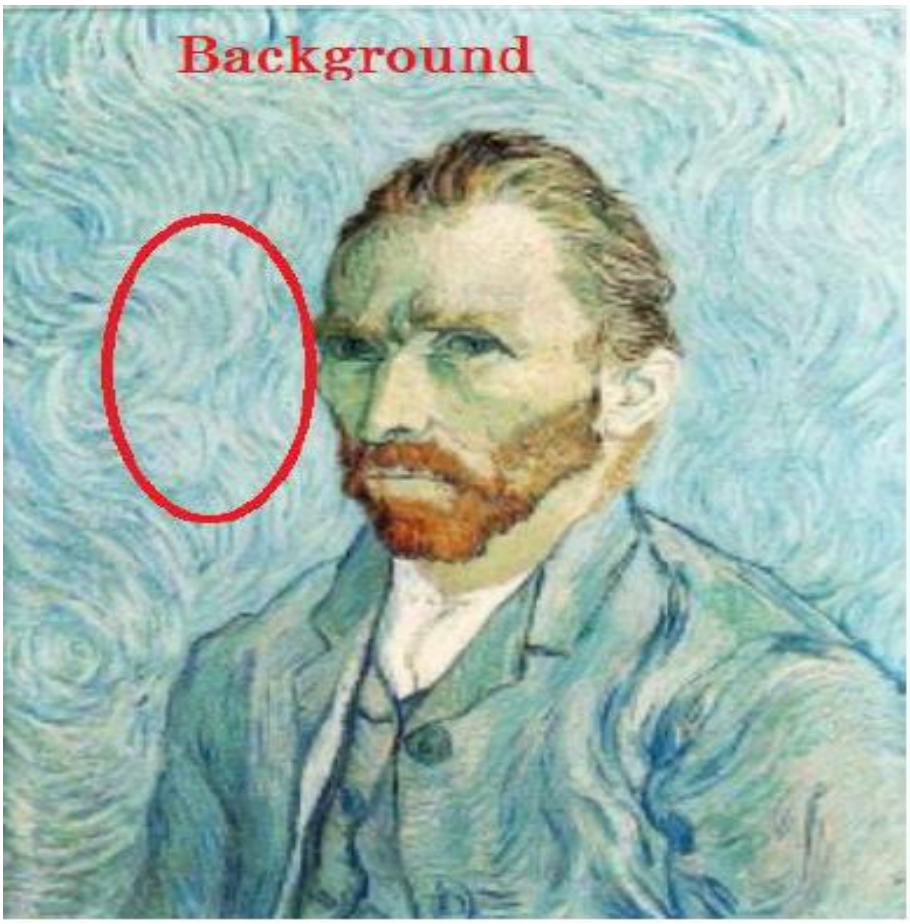


Background detected in 40x40.

Original Image: van Gogh.PNG

Stroke Image: van Gogh stroke.png






```
Background detected in 40x40.
```

Conclusion:

The code in this report is set for patch size of 40 × 40 specifically the location (100,140) in x-direction and (140, 180) in y-direction. Since, this location belongs to the background in all the images, so, the background is extracted even for the foreground area. For foreground, it returns zero values.

Actually, I first tried it for this patch size and then generalized it to any given size. I ran the generalized code but it took forever in execution, so, couldn't report the final results. The generalized code can be found in the zip folder while the code in this report is for 40 × 40 patch size.

**********THE END**********