GrabCut

March 28, 2019

0.1 Name Nitin kandpal

0.2 Roll Num 2018802004

0.3 Assignment 4

0.4 Algorithm description

1. Read the image apply ROI.
2. Get the ROI of object and create MASK of foregrond background
3. Generate the graph for ROI image each pixel is node and edeges are similurity cost(need to calculate once.
4. Create 2 GMM model one for background and one for foreground
5. In ROI image get the log probability of a pixel (node) from back ground GMM and foreground GMM. It is weights connected to souce or sink to nodes(pixels.
6. Use graphcut maxflow algorithm to cut the graph from souce (foreground) to sink (background).
7. update the MASK and repeat step 4 5 ,6 untill graph cut converge.

```
In [3]: ### import the dependencis
        #%inline
        import matplotlib.pyplot as plt
        import cv2
        import numpy as np

        from sklearn.mixture import GMM

        import maxflow
```

0.4.1 Below is the function for popup the image and user can crop the object by mouse.

```
In [ ]: refPt = []
        cropping = False

        def click_and_crop(event, x, y, flags, param):
                # grab references to the global variables
                global refPt, cropping

                # if the left mouse button was clicked, record the starting
```

```
        # (x, y) coordinates and indicate that cropping is being
        # performed
        if event == cv2.EVENT_LBUTTONDOWN:
                refPt = [(x, y)]
                cropping = True

        # check to see if the left mouse button was released
        elif event == cv2.EVENT_LBUTTONUP:
                # record the ending (x, y) coordinates and indicate that
                # the cropping operation is finished
                refPt.append((x, y))
                cropping = False

                # draw a rectangle around the region of interest
                cv2.rectangle(image, refPt[0], refPt[1], (0, 255, 0), 2)
                cv2.imshow("image", image)


def crop_roi(image):

    clone = image.copy()
    cv2.namedWindow("image")
    cv2.setMouseCallback("image", click_and_crop)

    # keep looping until the 'q' key is pressed
    while True:
            # display the image and wait for a keypress
            cv2.imshow("image", image)
            key = cv2.waitKey(1) & 0xFF

            # if the 'r' key is pressed, reset the cropping region
            if key == ord("r"):
                    image = clone.copy()

            # if the 'c' key is pressed, break from the loop
            elif key == ord("c"):

                    break

    # if there are two reference points, then crop the region of interest
    # from teh image and display it
    if len(refPt) == 2:
                roi = clone[refPt[0][1]:refPt[1][1], refPt[0][0]:refPt[1][0]]

                cv2.imshow("ROI", roi)
                cv2.waitKey(0)

    # close all open windows
```

```
    cv2.destroyAllWindows()
    return refPt[0][0],refPt[1][0], refPt[0][1],refPt[1][1]
```

0.4.2 Below code will take the ROI and generate the binary mask

```
In [ ]: img = image.copy()
        [h,w,c] = image.shape

        ROI = crop_roi(image)
        ROI = refPt[0][0],refPt[1][0], refPt[0][1],refPt[1][1]      ## column and row




        ROI_img = img[ROI[2]:ROI[3],ROI[0]:ROI[1]]


        #cv2.namedWindow("image")
        #cv2.imshow("image",ROI_img)
        #cv2.waitKey(0)
        #cv2.destroyAllWindows()

        ROI_img = ROI_img.astype(float)
        ## make foreground white

        ### initial mask

        mask = np.zeros((h,w),dtype='uint8')
        mask[ROI[2]:ROI[3],ROI[0]:ROI[1]] = 255
```

0.4.3 Below function will take the image and binary masj and create 3 xn pixels vectors of foreground and background for GMM

```
In [ ]: def get_fg_bg_pixels_1D(image, mask):

            [h1,w1] = mask.shape
            img_R_1D = np.reshape(image[:,:,2],(h1*w1,1))
            img_G_1D = np.reshape(image[:,:,1],(h1*w1,1))
            img_B_1D = np.reshape(image[:,:,0],(h1*w1,1))

            mask_1D = np.reshape(mask,(h1*w1,1))


            ### get bg pixels from mask reshape to 1 d vector and concatenate to R G B channel
            bg_pixels  =  np.concatenate((np.reshape(img_R_1D[mask_1D==0],(-1,1)), np.reshape(im

            ### get FG pixels from mask reshape to 1 d vector and concatenate to R G B channel
```

```
        fg_pixels  =  np.concatenate((np.reshape(img_R_1D[mask_1D==255],(-1,1)), np.reshape(

        return np.array(bg_pixels).astype(float),np.array(fg_pixels).astype(float)
```

0.4.4 Train two gaussian mixetre model of 5 componenets

```
In [ ]: def gmm_fg_bg(bg_pixels,fg_pixels):


            bg_gmm = GMM(n_components=5).fit(bg_pixels)

            fg_gmm = GMM(n_components=5).fit(fg_pixels)


            #return bg_gmm.score_samples([[2.0,100.0,50]])

            return bg_gmm,fg_gmm
```

0.4.5 function to calculate the similurity cost between to neighbor pixels (distance of pixels is assumed to one)

```
In [ ]: def Dlink(ROI_img, row1,col1,row2,col2):

            B = 1

            D = 50*np.exp(-B*np.sum(np.abs(np.square(ROI_img[row1,col1]-ROI_img[row2,col2]))))

            return D
```

0.4.6 Construct graph with maxflow library

0.4.7 Create the graph.

```
In [ ]: g = maxflow.Graph[float]()

        nodeids = g.add_grid_nodes(ROI_img[:,:,0].shape)

        nodes = g.add_nodes(ROI_img[:,:,0].shape[0]*ROI_img[:,:,0].shape[1])

        nodes = nodes-np.min(nodes)
```

0.4.8 Assign graph to similurity cost and node cost with helper function we wrote above

```
In [ ]: roi_h,roi_w = ROI_img[:,:,0].shape

        for i in range (0,roi_h-1):
            for j in range(0,roi_w-1):
        #
```

```
            g.add_edge(nodes[nodeids[i,j]], nodes[nodeids[i+1,j]], Dlink(ROI_img,i,j,i+1,j),
            g.add_edge(nodes[nodeids[i,j]], nodes[nodeids[i,j+1]], Dlink(ROI_img,i,j,i,j+1),


        bg_pixels,fg_pixels = get_fg_bg_pixels_1D(img, mask)

        bg_gmm,fg_gmm = gmm_fg_bg(bg_pixels,fg_pixels)

        for i in range (0,roi_h-1):
            for j in range(0,roi_w-1):

                g.add_tedge(nodes[nodeids[i,j]],fg_gmm.score_samples([ROI_img[i,j]])[0], bg_gmm.
```

0.4.9 calculate the graph cut maxflow algorithm and update the binary mask

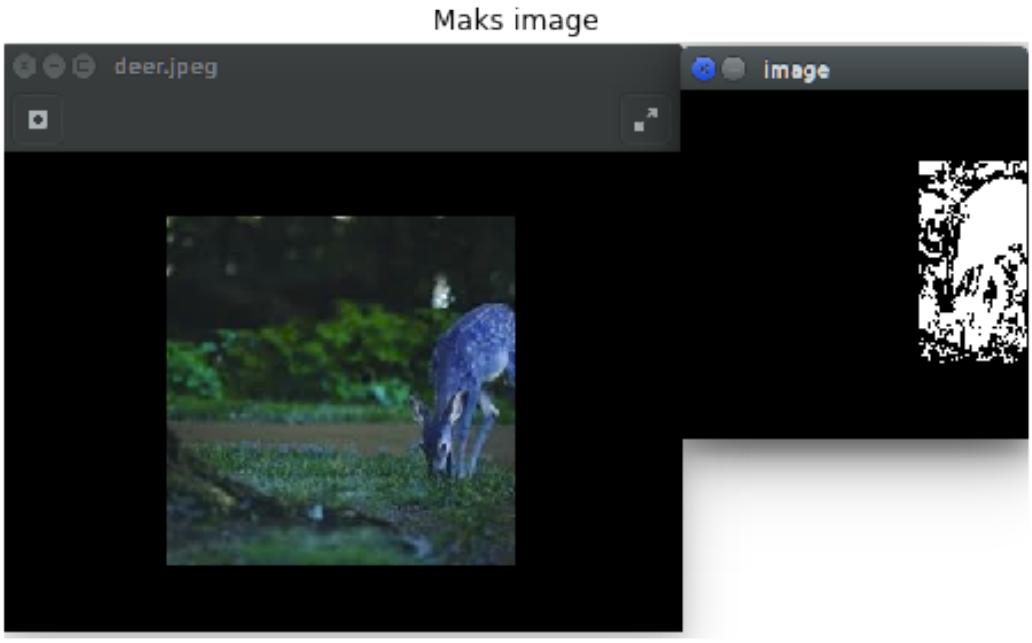
```
In [ ]: cost = g.maxflow()
        sgm = g.get_grid_segments(nodeids)
        mask[ROI[2]:ROI[3],ROI[0]:ROI[1]] = sgm.astype('uint8')*255
```

0.4.10 Below is the iterative process to udate the GMM models on updated mask and update the souce and sink to node weights (log probabilty) and perform graphcut. Repeat the proces untill converge.

```
In [ ]: ## uupdate the mask image
        iterate = 0

        cost = []
        while (iterate <5):
            bg_pixels,fg_pixels = get_fg_bg_pixels_1D(img, mask)
            bg_gmm,fg_gmm = gmm_fg_bg(bg_pixels,fg_pixels)
            for i in range (1,roi_h-1):
                for j in range(1,roi_w-1):
                    g.add_tedge(nodes[nodeids[i,j]],fg_gmm.score_samples([ROI_img[i,j]])[0], bg_



            #cost = g.maxflow()
            cost.append(g.maxflow())
            sgm = g.get_grid_segments(nodeids)
            mask[ROI[2]:ROI[3],ROI[0]:ROI[1]] = sgm.astype(int)*255
            iterate = iterate+1


In [4]: ## some result

In [5]: img1 = cv2.imread('deer_result.png')
        fig=plt.figure(figsize=(8, 8))
        columns = 1
```

```
        rows = 1
        fig.add_subplot(rows, columns, 1)
        plt.axis("off"),
        plt.title("Maks image")
        plt.imshow(img1)
```

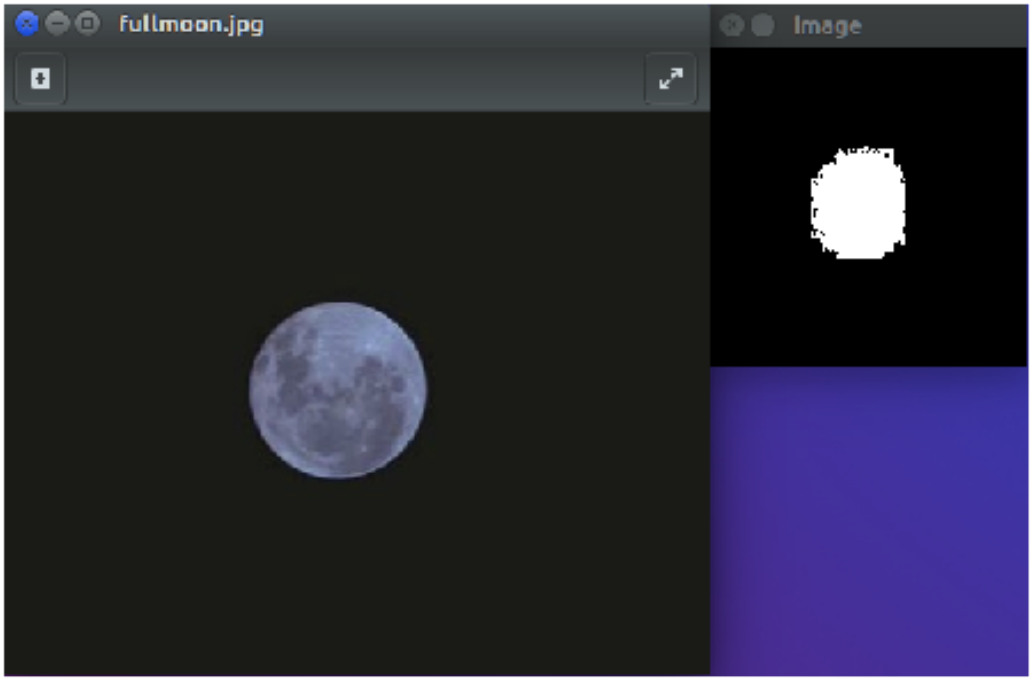
```
Out[5]: <matplotlib.image.AxesImage at 0x7f10f9969400>
```




```
In [6]: img1 = cv2.imread('moon_result.png')
        fig=plt.figure(figsize=(8, 8))
        columns = 1
        rows = 1
        fig.add_subplot(rows, columns, 1)
        plt.axis("off"),
        plt.title("Maks image")
        plt.imshow(img1)
```

```
Out[6]: <matplotlib.image.AxesImage at 0x7f10f87b4710>
```

Maks image



0.4.11 there is some bug in my code i have doubt in creating the similarity weigth matrix or either i missed something because many time when object are bigger in image segmentation is not converging