Interactive Foreground Extraction using GrabCut Algorithm

Goal

In this chapter

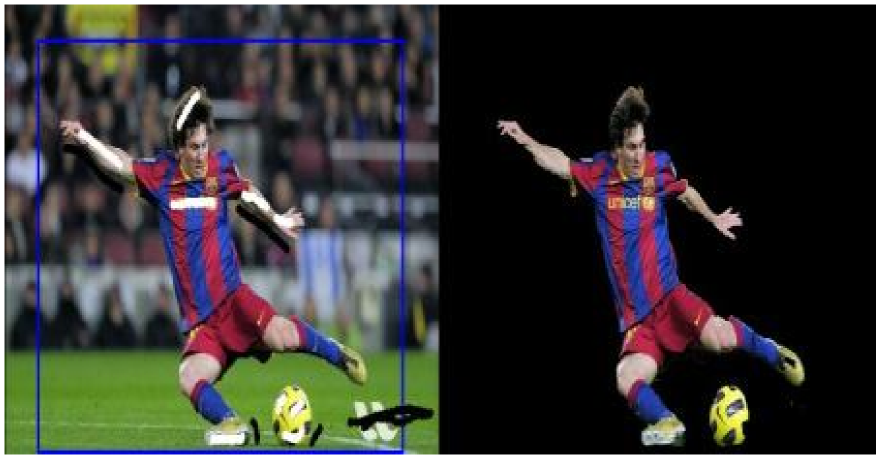
- We will see GrabCut algorithm to extract foreground in images
- We will create an interactive application for this.

Theory

GrabCut algorithm was designed by Carsten Rother, Vladimir Kolmogorov & Andrew Blake from Microsoft Research Cambridge, UK. in their paper, "GrabCut": interactive foreground extraction using iterated graph cuts . An algorithm was needed for foreground extraction with minimal user interaction, and the result was GrabCut.

How it works from user point of view ? Initially user draws a rectangle around the foreground region (foreground region should be completely inside the rectangle). Then algorithm segments it iteratively to get the best result. Done. But in some cases, the segmentation won't be fine, like, it may have marked some foreground region as background and vice versa. In that case, user need to do fine touch-ups. Just give some strokes on the images where some faulty results are there. Strokes basically says *"Hey, this region should be foreground, you marked it background, correct it in next iteration"* or its opposite for background. Then in the next iteration, you get better results.

See the image below. First player and football is enclosed in a blue rectangle. Then some final touchups with white strokes (denoting foreground) and black strokes (denoting background) is made. And we get a nice result.

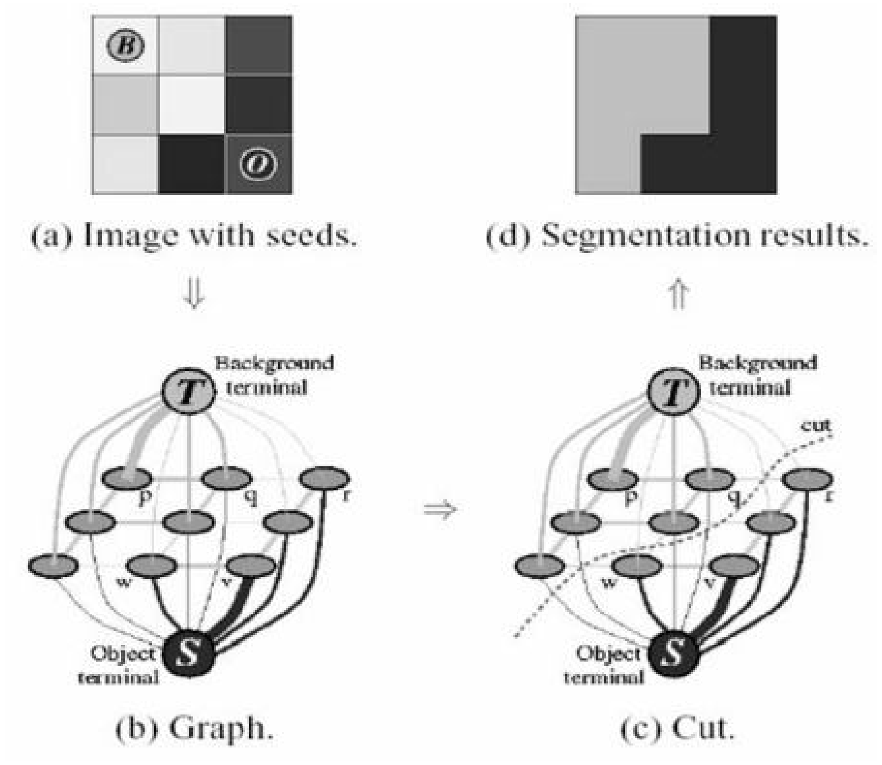


image

So what happens in background ?

- User inputs the rectangle. Everything outside this rectangle will be taken as sure background (That is the reason it is mentioned before that your rectangle should include all the objects). Everything inside rectangle is unknown. Similarly any user input specifying foreground and background are considered as hard-labelling which means they won't change in the process.
- Computer does an initial labelling depending on the data we gave. It labels the foreground and background pixels (or it hard-labels)
- Now a Gaussian Mixture Model(GMM) is used to model the foreground and background.
- Depending on the data we gave, GMM learns and create new pixel distribution. That is, the unknown pixels are labelled either probable foreground or probable background depending on its relation with the other hard-labelled pixels in terms of color statistics (It is just like clustering).
- A graph is built from this pixel distribution. Nodes in the graphs are pixels. Additional two nodes are added, **Source node** and **Sink node**. Every foreground pixel is connected to Source node and every background pixel is connected to Sink node.
- The weights of edges connecting pixels to source node/end node are defined by the probability of a pixel being foreground/background. The weights between the pixels are defined by the edge information or pixel similarity. If there is a large difference in pixel color, the edge between them will get a low weight.
- Then a mincut algorithm is used to segment the graph. It cuts the graph into two separating source node and sink node with minimum cost function. The cost function is the sum of all weights of the edges that are cut. After the cut, all the pixels connected to Source node become foreground and those connected to Sink node become background.
- The process is continued until the classification converges.

It is illustrated in below image (Image Courtesy: http://www.cs.ru.ac.za/research/g02m1682/)



image

Demo

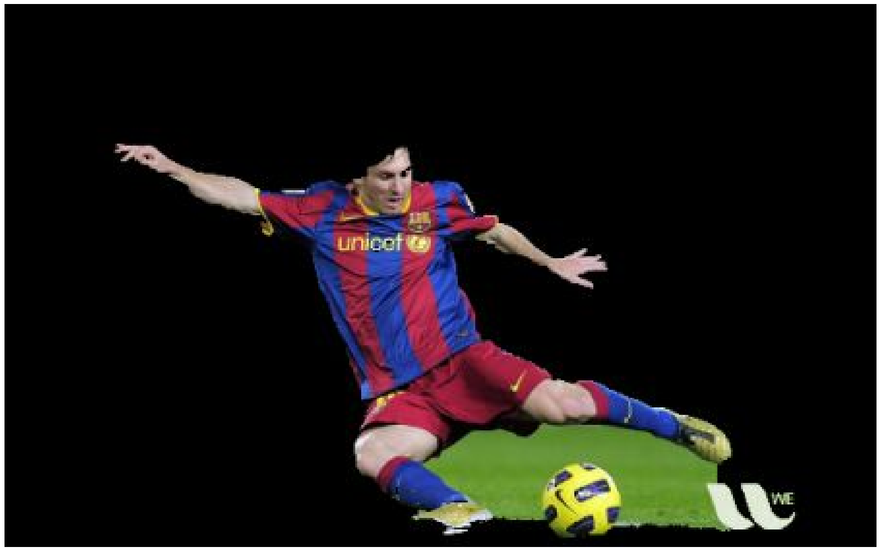
Now we go for grabcut algorithm with OpenCV. OpenCV has the function, **cv.grabCut()** for this. We will see its arguments first:

- *img* - Input image
- *mask* - It is a mask image where we specify which areas are background, foreground or probable background/foreground etc. It is done by the following flags, **cv.GC_BGD**, **cv.GC_FGD**, **cv.GC_PR_BGD**, **cv.GC_PR_FGD**, or simply pass 0,1,2,3 to image.
- *rect* - It is the coordinates of a rectangle which includes the foreground object in the format (x,y,w,h)
- *bdgModel*, *fgdModel* - These are arrays used by the algorithm internally. You just create two np.float64 type zero arrays of size (1,65).
- *iterCount* - Number of iterations the algorithm should run.
- *mode* - It should be **cv.GC_INIT_WITH_RECT** or **cv.GC_INIT_WITH_MASK** or combined which decides whether we are drawing rectangle or final touchup strokes.

First let's see with rectangular mode. We load the image, create a similar mask image. We create *fgdModel* and *bgdModel*. We give the rectangle parameters. It's all straight-forward. Let the algorithm run for 5 iterations. Mode should be ***cv.GC_INIT_WITH_RECT*** since we are using rectangle. Then run the grabcut. It modifies the mask image. In the new mask image, pixels will be marked with four flags denoting background/foreground as specified above. So we modify the mask such that all 0-pixels and 2-pixels are put to 0 (ie background) and all 1-pixels and 3-pixels are put to 1(ie foreground pixels). Now our final mask is ready. Just multiply it with input image to get the segmented image.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('messi5.jpg')
mask = np.zeros(img.shape[:2],np.uint8)

bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)

rect = (50,50,450,290)
cv.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)

mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img = img*mask2[:,:,np.newaxis]

plt.imshow(img),plt.colorbar(),plt.show()
```

See the results below:



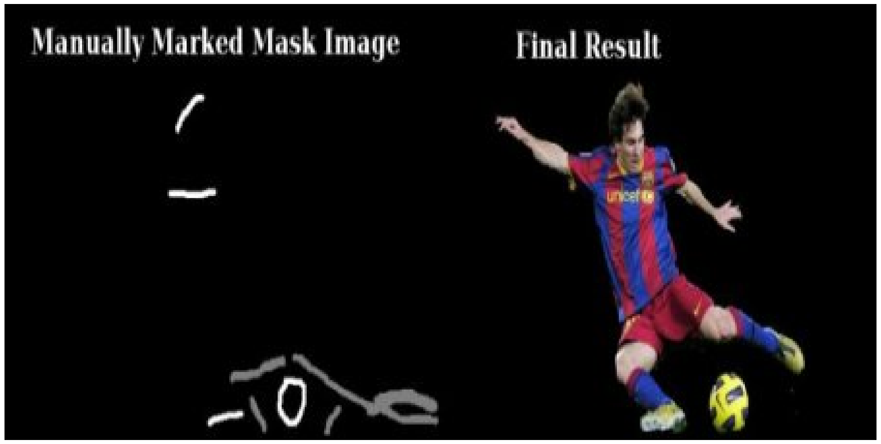
image

Oops, Messi's hair is gone. *Who likes Messi without his hair?* We need to bring it back. So we will give there a fine touchup with 1-pixel (sure foreground). At the same time, Some part of ground has come to picture which we don't want, and also some logo. We need to remove them. There we give some 0-pixel touchup (sure background). So we modify our resulting mask in previous case as we told now.

What I actually did is that, I opened input image in paint application and added another layer to the image. Using brush tool in the paint, I marked missed foreground (hair, shoes, ball etc) with white and unwanted background (like logo, ground etc) with black on this new layer. Then filled remaining background with gray. Then loaded that mask image in OpenCV, edited original mask image we got with corresponding values in newly added mask image. Check the code below:

```
# newmask is the mask image I manually labelled
newmask = cv.imread('newmask.png',0)

# wherever it is marked white (sure foreground), change mask=1
# wherever it is marked black (sure background), change mask=0
mask[newmask == 0] = 0
mask[newmask == 255] = 1

mask, bgdModel, fgdModel = cv.grabCut(img,mask,None,bgdModel,fgdModel,5,cv.GC_INIT_WITH_MASK)

mask = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img = img*mask[:,:,np.newaxis]
plt.imshow(img),plt.colorbar(),plt.show()
```

See the result below:



image

So that's it. Here instead of initializing in rect mode, you can directly go into mask mode. Just mark the rectangle area in mask image with 2-pixel or 3-pixel (probable background/foreground). Then mark our sure_foreground with 1-pixel as we did in second example. Then directly apply the grabCut function with mask mode.

Additional Resources

Exercises

1. OpenCV samples contain a sample grabcut.py which is an interactive tool using grabcut. Check it. Also watch this youtube video on how to use it.
2. Here, you can make this into a interactive sample with drawing rectangle and strokes with mouse, create trackbar to adjust stroke width etc.