Computer Vision Mini Project 1:

Interactive Image Segmentation on Smart Phone using Processing Language*

Karan Daei-Mojdehi
Course Instructor: Dr. Ulas Bagci
Department of Computer Science, University of Central Florida

November 22, 2015

Outline

In this mini project, a region growing segmentation algorithm is implemented in Processing language which is mostly written in Java and then transferred to an android phone using Processing's android mode for performing segmentation task on static images.
Keywords:Computer Vision, Image Segmentation, Region Growing, Processing Language, Android

Introduction

The problem of image segmentation has been around for more that fifteen years and many successful algorithms have been proposed. This problem involves identifying regions of image which probably belong to different objects or scenes and outlining them. The output can later be used for other computer vision tasks like object detection, scene recognition, object classification, etc. Meanwhile interactive image segmentation is performing the same task with some help from a user or a third party application which provide initial seeds for the region to segment. One of

* www.processing.org

possible approaches to tackle this problem is region growing segmentation which will be explained briefly in next section. I chose this method for fast implementation considering limited available processing and memory resources on a mobile phone as well as limited computer vision libraries for Processing's android mode .

Algorithm: Region Growing Segmentation

Region growing is one of the very first methods which was used for segmentation task. Given the position of a pixel(initial seed) in an image, we explore eight neighboring pixels of the initial seed with goal of determining their resemblance to the initial seed. In order to quantify this resemblance, we compare intensity of two pixels in three RGV channels and define pixels' difference as euclidean distance of these three values. Neighboring pixels with a difference lower than a given threshold are considered alike. By this definition neighbors that are alike enough to the first pixel are included in our segment, else they are marked visited and left out of current segment. Each newly added pixel to the segment will have potential neighbors for our segment, therefore it is added to the 'to-visit' queue to be explored. In each step, a pixel is popped out of the queue and it's neighbors that are not visited yet will be examined; these pixel will have their intensities compared to the neighbor pixel (included in the segment) as well as the intensity of the initial seed so that intensities don't drift away too much from the starting pixel. Algorithm ends when no more candidates are left in the 'to-visit' queue and pixels included in the segment are outlined by some markings or contours.

Challenges

For implementation on an android phone, knowing that Processing is a high level language and for low level operations (like pixel difference calculations), first I decided to use OpenCV library for Processing which is a java wrapper to OpenCV's c/C++ libraries. Although OpenCV libraries can be loaded in Java mode perfectly, even after developing a library for android mode , using an android software development kit, complications arose when I tried to install them on an android phone. By inspecting the wrapper libraries I found out they use calls to native C/C++ libraries using a dynamic link library (dll) which can be used only in windows. In a second

try I used Processing's built in libraries for extracting pixels from images and used a recursive implementation of region growing, but due to limited stack size, for most of segmentation tasks the applications stack overflowed and crashed the application. In last attempt, I implemented a non-recursive version of the algorithm which used an IntList structure of Processing for keeping track of visited pixels. Although the stack overflow never arose again, the application was extremely slow on a phone. I used two tricks to enhance segmentation speed: first, instead of using rows and columns to access pixels in an image, I used a one dimensional array with correct coding that maps each pixel's row and column to a specific location in the array, and second, I replaced IntList structure for visited pixels, which required a considerable amount of time to determine whether a pixel has been visited, with a boolean array, same size as the image, that held a true or false flag for each pixel's mapped index. Although this requires slightly more spatial memory, it enhances segmentation speed tremendously. Rest of the challenges where minor ones which involved working with android drawing functions and handling hand gestures.

Working with the App

The android app is created for demonstration only and includes six built in images. In order to segment a region, it has to be touched, note the "performing segmentation..." text. Segmentation results will be shown (after the text disappears) by changing color of the segment (default: red). If segmentation is running out of regions, threshold can be reduced by flicking anywhere on the screen from top to bottom, the new threshold will be show by a fading text. The amount in threshold change scales with flick distance. On the other hand, threshold should be increased if segmentation is growing only partly in a region by flicking the screen from top to bottom. Current image can be changed by flicking the screen horizontally. Color used for outlining segment can be altered between red,green, and blue by pressing and holing any area on the screen, a fading text will declare color change.

Future Work

After successfully implementing region growing segmentation on an android phone, better (and of course more complicated) algorithms like GrabCut[1] can be implemented after integrating

off-the-shelf java libraries using android sdk. It will involve using graph structures and fast implementations of max-flow/min-cut algorithms from Boykov[2].

References

[1] Rother, Carsten, Vladimir Kolmogorov, and Andrew Blake(2004). "Grabcut: Interactive foreground extraction using iterated graph cuts", *ACM Transactions on Graphics* (TOG) 23, no. 3 (2004): 309-314.

[2] Boykov, Yuri, and Vladimir Kolmogorov(2004). "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision." *Pattern Analysis and Machine Intelligence, IEEE Transactions* on 26.9 (2004): 1124–1137.