Due date: October 3, 2016, 11:59 PM (Madison time). You have **four** late days — use it at as you wish. Once you run out of this quota, the penalty for late submission goes as follows for each subsequent 24 hour period: $\{10\%, 25\%, 50\%\}$. That is, if you submit this assignment on a Monday at 4:00 AM in the morning (if it were due on Friday at 11:59 PM), your submission is three days late. You can either use your late days quota (or let the penalty be applied). Clearly indicate in your submission if you seek to use the quota.

What to turn in:

1. Your submission should include your complete code base in an archive file (`zip, tar.gz`), example images you used to evaluate your implementation on (organized under different directories such as `q1/, q2/,` and so on), and a README describing how to run it.

2. A brief report (typed up, submit as a PDF file, NO handwritten scanned copies) describing what you implemented and known failure cases.

Notes from instructor:

- Start early!

- Stick with small images to start with to keep the running time manageable. There will be no penalty for (slow) running time. However, use this opportunity to learn about data structures that you think might be helpful. You may ask the TA or instructor for suggestions, and discuss the problem with others (minimally). But **all parts of the submitted code must be your own**. Matlab is slow with loops, and vectorized code is significantly faster.

- Submission instructions (for Canvas) and a number of images have been provided. Remember that the performance of your submitted code on these 'example images' is not sufficient for full credit. Try thinking about and accounting for degenerate cases. You should generate new images to evaluate the robustness of your code. Data we use to grade your submitted code is not included here.

Problem 1

(Convex Hull, Perimeter, Distances, Dilate, We consider 8-connected for all problems, **30pts**)

1. **(10pts)** Write a function called `imOut = mySegmenter(imIn).` `imIn` will be a gray-scale image containing a single region (blob) with varying intensity levels. The background will be dark. Your Matlab function should work similar to the Otsu's method for threshold calculation (or better) and compute the the foreground region of interest outputted as 1s in image `imOut`. All other pixels in `imOut` will be zero. In addition, your function should calculate and print some basic statistics of the region such as area (number of pixels) and diameter. You should write your own implementation for this functionality, usage of `regionprops, convhull, convhulln, qhull` from Matlab not permitted. If in doubt, ask!

   As a precursor to this task, you must write a function, `myPerimeter()` to perform a perimeter extraction of the region, this should be called from within `mySegmenter()`. The output of this function will be a $m \times 2$ matrix, where m is the number of points (pixels) on the perimeter, `bwperim, regionprops` not permitted. You need not print out all pixels that fall on the perimeter but may want to report this measure of the boundary length in the region statistics above.

2. **(10pts)** The second part of this problem deals with computing the "distance" of every pixel in the image to its closest point on the perimeter – in the form of a function called `imOut = myDT(imIn)`. The output from this process should be **two** colormap images: where the first corresponds to the distances for pixels *outside* the interior of the region and the second image will correspond to pixels which lie *inside* the interior of the segmented region.

3. **(10pts)** The image archive file consists of a set of images $\{1, 2, \cdots, n\}$. Associated with each image is a random variable saved in the array "predictor" of size n. If the images were participants in a study, the predictor variable can be thought of as a clinical measurement. First, look up Pearson's correlation, e.g., this link. Then, calculate correlation between the predictor variable and each of the image-wise summary statistics (e.g., area) you calculated above. Report your findings.

Problem 2

(Region finding, Labeling, **10pts**)

1. **(10pts)** Write a function `imOut = myRegionFinder(imIn)` to extract distinct regions from a binary image. The image will have more than one blob(s), your function must recognize the different blobs as distinct regions. The output of the function should be a gray scale image (same size as the input) where pixels in the same blob are assigned the same unique intensity value. Again, do not use `bwlabel, bwlabeln` explicitly but you are free to employ similar ideas as implemented in those functions. This should be based on a neighborhood system of your choice. (hint: look up the concept of "connected components").

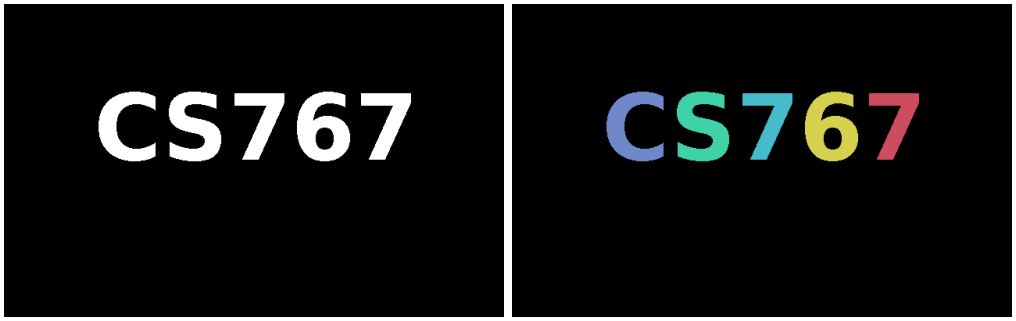

Figure 1: The number of connected comments: 5 , number of holes: 1.

Problem 3

(Edge detector, Edge linking, **30pts**)

1. **(15pts)** Write a gradient based edge detector. Your code should load in a gray-scale image (use imread and convert to a double array using `im2double`). You can display the image using functionality like `imagesc`. Once you have loaded in the image, you should smooth the image with a Gaussian filter and then compute horizontal and vertical derivatives using a derivative filter. The amount of smoothing is determined by the parameter σ of the Gaussian (which should be a parameter in your code). You can use `conv2` with the 'same' option to perform the required convolutions. Once you have computed the derivatives in the x and y directions, compute the gradient magnitude and orientation. Display the gradient magnitude as an image and the orientation using `quiver` functionality.

   Bonus points: To get nice results, come up with ways to remove cluttered edges.

2. **(15pts)** We want to extract the outer boundary of each object contained in one image. First, use `edge()` function (or your own edge detector) to obtain the edge map. Then use `imview()` function to manually locate one pixel located on the outer boundary of each object. Next, apply any technique you

can come up with to trace the boundary edge pixels. Try to get a boundary as complete and accurate as possible: it is OK if your results are not perfect (see below). Plot the final boundary you are able to detect and trace for each object. Discuss the success, failure, and possible ways for improvement in the report.

Note that boundary tracing is a non-trivial problem, involving many complicated issues such as noise, broken boundaries, ambiguous locations and directions, among others. Do not worry if you are not able to get a 'perfect' result — but try. We will grade this question on a curve.

Do not use the '`bwtraceboundary()` function provided by Matlab image processing library. The idea is for you to gain familiarity with the algorithm through the experiments in this homework.

Problem 4

(Integral Images, **30pts**)

In medical image analysis, one important application is to identify whether there is something interesting or abnormal in a given image. A common way to tackle this task is to extract local features from patches, followed by some statistical analysis. In this problem, we will compute certain simple summary measures from rectangular patches.

1. **(10pts)** Read the following paper before starting work on this problem. Paul A. Viola and Michael J. Jones, Robust Real-Time Face Detection. IJCV 2001. In your PDF report, summarize how you used ideas from this paper within your solutions to the following two sub-problems. Even if you ended up not implementing any functionality from this paper, you should nonetheless attempt to describe how you could have used certain concepts.

2. **(10pts)** Suppose we have a mask (divided into 2 rectangles) as shown in Figure 2, the summary measure we are interested in is the mean of the pixel intensities within the right rectangle subtracted from the mean of the pixel intensities within the left rectangle.

   Write a function to compute the rectangular features: `feature = getFeature(imIn, i, j, m, n)`, where `imIn` is a gray-scale image, (i, j) is the pixel at which to evaluate the mask (see Fig. 2), and $2m$ and n give the width and height of the smallest rectangle as shown in Figure 2. The origin of the mask is defined to be the upper left corner.

   After running your code on the example images, comment on your results. What can you say about the type of region that produces no response from the mask (i.e. it sums to approximately zero)? What can you say about the type of region that does produce a response from the mask?

3. **(10pts)** An important task for the problem of detection is how to choose the mask size. In this part, we will explore the influence of mask size with respect to rectangle feature.

   Suppose for any fixed mask size, we want to compute rectangle features for masks at all possible positions. Write a function that compute the histogram for one mask size. `featureHist = getFeatureHist(imIn, m, n, numBins)`. `numBins` is the number of bins used in computing the histogram.

   Plot five histograms with different mask size. If appropriate, summarize any observations you have about its behavior as a function of window size.

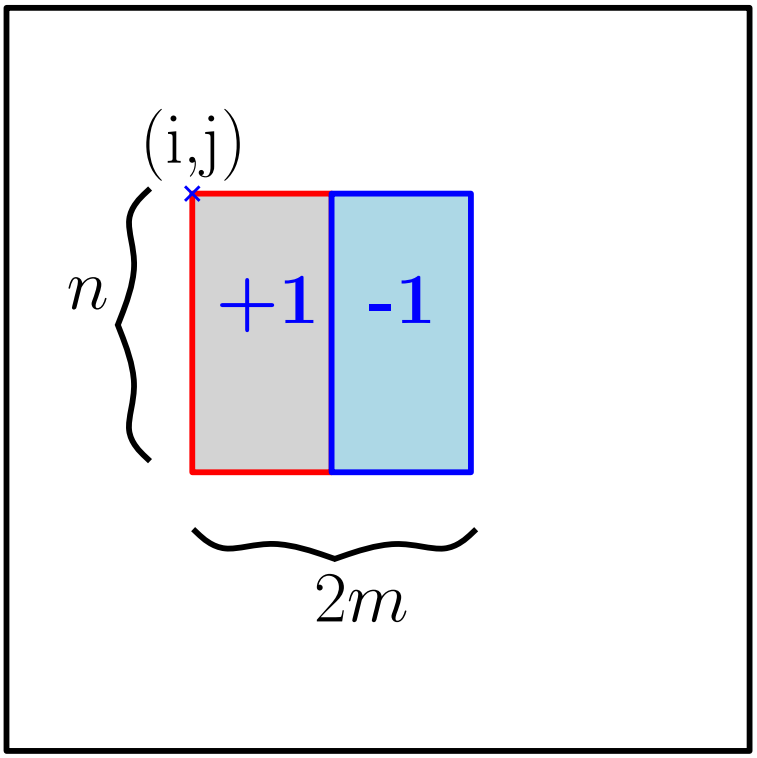

Figure 2: Illustration of integral image.