CPS109 - Fall 2014
Assignment 2: Where's Waldo?
Due Date: November 14, 2014

template image

search image

Figure 1: Where's Waldo?

Problem description:

The aim of this assignment is to give you practice writing methods, calling methods, using (nested) loops ... and find Waldo! To do the assignment, you need classes that come with the supplementary textbook, Introduction to Computing and Programming with Java, by Guzdial and Ericson, used in Lab 5. You can copy the Guzdial classes to a directory of your choosing with the command:
`scp -r moon.scs.ryerson.ca:/usr/courses/cps109/guzdial/bookClasses .`

In this assignment we consider an instance of the *template matching* problem. Template matching is an image processing technique for finding a small part(s) of an image that matches a template image. It can be used in a variety of real world situations, including as a component of a quality control system in manufacturing, a way to autonomously navigate a mobile robot and image manipulation (e.g., blurring an image and extracting edges from an image). Your task is to find Waldo, given by a template image, in a large cluttered image, given by the search image, see Fig. 1.

In the following, we use $S[row, col]$ to denote the search image, where $[row, col]$ represent the pixel coordinates in the search image. We use $T[row_t, col_t]$ to denote the template, where $[row_t, col_t]$

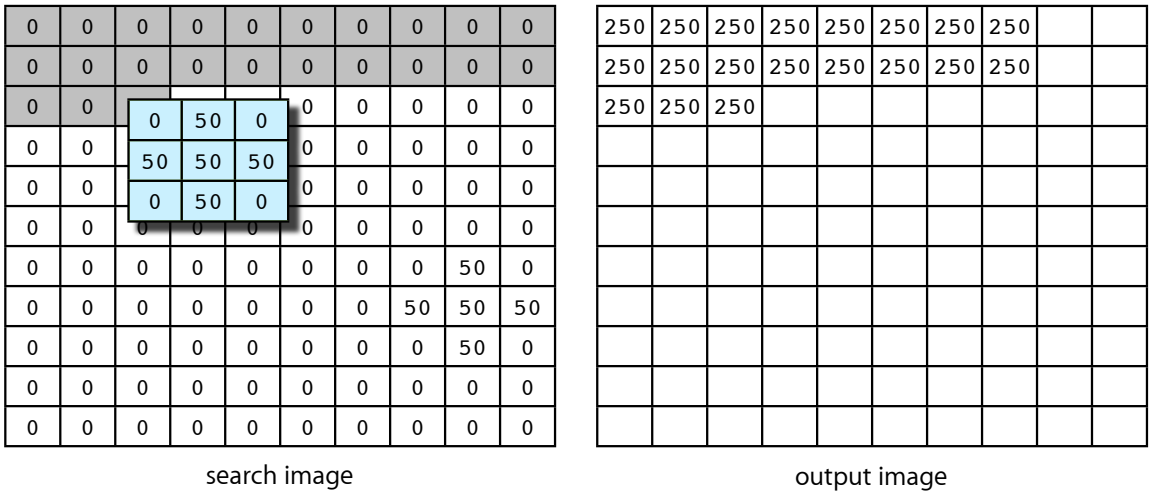

Figure 2: Overview of template matching. The blue image overlaid over the search image represents the template image. The gray search pixels indicate those that have been traversed. The output image is computed by "sliding" the template over the search image from left-to-right, top-to-bottom and computing the SAD match between the portion of the search image under the template image.

represent the pixel coordinates in the template. Both images are assumed to be grayscale. We then simply move the top-left corner of the template $T[row_t, col_t]$ over each $[row, col]$ point in the search image and calculate the sum of the absolute differences (SAD) between the pixel values in $S[row, col]$ and $T[row_t, col_t]$ over the area spanned by the template and store this value in the corresponding $[row, col]$ position in the output image. Upon completion of this process, the $[row, col]$ position in the output image with the minimum value corresponds to the best match of the template in the search image.

In other words, for each position $[row, col]$ in the search image (see Fig. 2 for an overview):

1. Place the template over the search image such that the top-left corner of the template is aligned with the current search image position.

2. Compute the match score between the template and search images:

   (a) Subtract the pixel intensities of the corresponding points in the template and search images.

   (b) Take the absolute value of each difference and add them all up.

3. Store the result in the output image at position $[row, col]$.

DO NOT compute the match score around the right and bottom edges of the image, since the template will not completely lie within the image and thus the match score will be undefined, see Figs. 3 and 4. The width (height) for the right (bottom) undefined region corresponds to the width (height) of the template less one. Instead, set the values of the output image for the undefined regions to some large value.

Your task is to implement:

1. A template matching method, `sadMatch()`, that takes as parameters two images: (i) template and (ii) search. The method returns an output 2D array containing the computed template match scores.

2. A method, `find2DMin()`, that takes as parameter the output 2D array of the template matcher. The method returns a java.awt.Point which contains the row and column location of the minimum match score.

3. A method, `displayMatch()`, that takes as parameters the search image and 2D location of the minimum match score. The method displays the search image with a bounding box that is the same size of the template (in our case Waldo) of width 3 pixels outlining the location of the template in the search image.

4. A driver class, `MatcherTester`, that reads in the template image (`waldo.png`) and search image (`scene.png`) and calls `sadMatch()`, `find2DMin()` and `displayMatch()`.

Marking rubric (out of 15):

1 points Documentation: Use of Javadoc comments for every class, method and constructor.

1 points Coding style: Use of proper indentation, proper capitalization for variable names and classes and methods, proper mnemonic variable names, use of constants rather than magic numbers.

1 points Driver class, `MatcherTester`.

6 points Template matching method, `sadMatch()`.

3 points Find minimum match score method, `find2DMin()`.

3 points Display detected template method, `displayMatch()`.

1 bonus points Adapt your code to accommodate colour images. Use `waldo.jpg` as the template image and `scene.jpg` for the search image.

Submit:

Submit to Blackboard all your Java files and a screenshot showing the image that is produced by `displayMatch()`. For those that attempt the bonus, also submit your adapted code and a screenshot showing the image that is produced of the detected template.

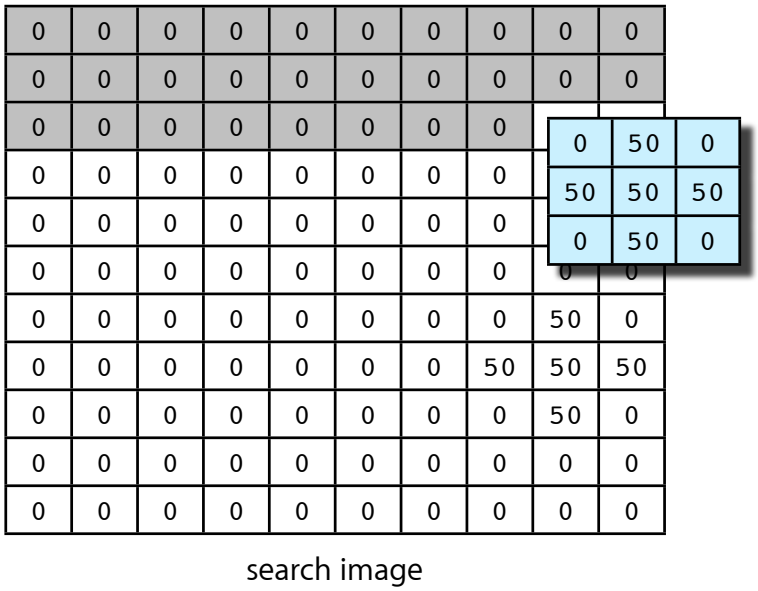

Figure 3: Undefined template match score. The blue image partially overlaid over the search image represents the template image. The gray search pixels indicate those that have been traversed. Since there are template pixels beyond the boundary of the search image, the match score computation is undefined.

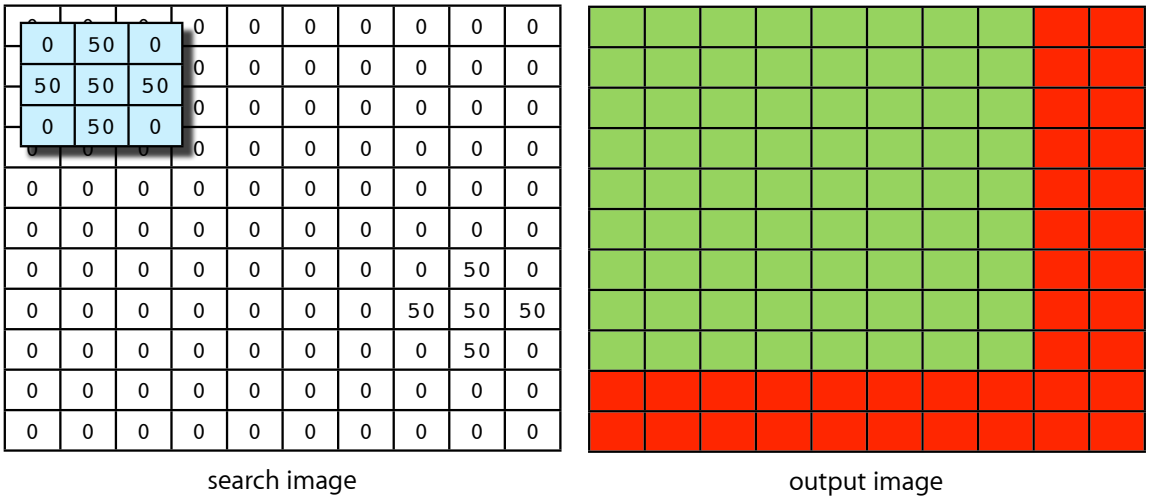

Figure 4: Undefined regions for match computation. The blue image overlaid over the search image represents the template image. The green and red pixels in the output image denote the regions where the match score is defined and undefined, respectively.