ECSE-415 Introduction to Computer Vision

Assignment #1: Template Matching

Due: Friday, Feb. 12, 2016, 11:59pm

Please submit your assignment solutions electronically via the myCourses assignment dropbox. The solutions should be in PDF format. Attempt all parts of this assignment. Directives about the report is given in Section 3. Associated code files should be commented and submitted, so that they can be decompressed and run for testing.

This assignment is out of a total of **100 points**. It is designed to familiarize the students with writing image processing algorithms in Matlab. The goal of the assignment is to help the students obtain a deeper understanding of matching images using template matching. The assignment requires the students to use a given Matlab template code and modify it as described below. A set of images is included with this assignment.

The student is expected to write his/her own code. To submit your code, submit the m-file script that includes the template code (see the next section) and all the required scripts and functions.

Assignments received up to 24 hours late will be penalized by 30%. Assignments received more than 24 hours late will not be marked.

Overview of the Template Code

For this assignment, you are given 13 template images of playing cards. You are also given 4 test images. For each test image, you are asked to apply two template matching techniques in order to find the best matching location for each template image within the test images: Sum of Square Differences (SSD) and Normalized Cross Correlation (NCC). You should then create an output image (for each test case) after applying each template matching methods. The output image contains the original input test image, overlaid with the matched label of the template card names. An example is shown in Figure 1.

Please use the code template in Table 1 for writing your code (you can download the complete "Assignment1.m" file from myCourses). The `Assignment1(.)` function accepts the full path/name of an input test image, matches all the template images and displays two output images, one for SSD and the other for NCC.

Table 1: Template code for Assignment 1

```
function [] = Assignment1(image_name)
%Assignment1 template
% image_name       Full path/name of the input image (e.g. 'Test Image (1).JPG')

%% Load the input RGB image

%% Create a gray-scale duplicate into grayImage variable for processing

%% List all the template files starting with 'Template-' ending with '.png'
templateFileNames = dir('Template-*.png');
%% Get the number of templates (this should return 13)
numTemplates = length(templateFileNames);
%% Set the values of SSD_THRESH and NCC_THRESH

%% Initialize two output images to the RGB input image

%% For each template, do the following
for i=1:numTemplates
    %% Load the RGB template image, into variable T

    %% Convert the template to gray-scale

    %% Extract the card name from its file name (look between '-' and '.' chars)
    cardNameIdx1 = findstr(templateFileNames(i).name,'-') + 1;
    cardNameIdx2 = findstr(templateFileNames(i).name,'.') - 1;
    cardName = templateFileNames(i).name(cardNameIdx1:cardNameIdx2);

    %% Find the best match [row column] using Sum of Square Difference (SSD)
    [SSDrow, SSDcol] = SSD(grayImage, T, SSD_THRESH);

    % Overlay the card name on the best match location on the SSD output image

    %% Find the best match [row column] using Normalized Correlation Coefficients (NCC)
    [NCCrow, NCCcol] = NCC(grayImage, T, NCC_THRESH);

    % Overlay the card name on the best match location on the NCC output image

end

%% Display the output images

end

%% Implement the SSD-based template matching here
function [SSDrow, SSDcol] = SSD(grayImage, T, SSD_THRESH)
end

%% Implement the NCC-based template matching here
function [NCCrow, NCCcol] = NCC(grayImage, T, NCC_THRESH)
end
```

The code starts by loading the input image and creating a gray-scale version of it for processing (you should write the necessary code for these). The code then scans the current directory, looking for all the files having `'Template-*.png'` file names. This effectively lists all the template files into the `templateFileNames` variable. You can access each template file by indexing this variable, e.g. `templateFileNames(i).name`. Using the length of this variable, the code finds and stores the number of template files into the `numTemplates` variable. Before matching each template with the test image, you need to set up two variables, i.e. the thresholds for accepting/rejecting a matched template (`SSD_THRESH` and `NCC_THRESH`), and initialize the output images to the RGB test images. Try to optimize the thresholds values to have the lowest false positive while having a decent true positive.

Using a for loop on each template image, the code first loads and converts each template image into a gray-scale version (you should write the necessary code for these), extracts its label (using its file name) into the `cardName` variable and proceeds by passing the gray-scale test image, the gray-scale template image and the corresponding threshold to the `SSD(.)` and the `NCC(.)` functions. You are required to write the code for the `SSD(.)` and the `NCC(.)` functions, which are described in Sections 1 and 2, respectively. In both functions, the template image should be run over the entire test image and the best matched location is returned. These functions return $(row, column)$ of the matched location (if any).

You then need to augment the output images by printing the template card name (use `cardName` variable) on the matched location. Use Matlab's built-in `insertText(.)` function to put numbers and strings on images. It will be possible that your approach will lead to more than one template matching the same card in the test image. In this case, the labels would be placed on top of each other, making them hard to read. To overcome this problem, try to print the card names at the best match location, with a random integer offset (e.g. 30 pixels) to make things easier to see (see Figure 2). You can use Matlab's built-in `randi(.)` function to generate random integers. Also, try to use a small font size, e.g. 6. After the for-loop, you should display the two output images (use a subplot or two separate figures).

Figure 1: A test image (left) and its corresponding output image (right).

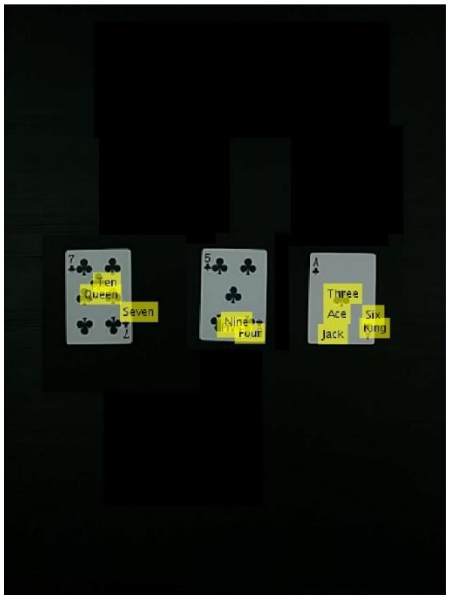

Figure 2: Dealing with overlapping card names by randomly offsetting the labels.

1 SSD-based Template Matching (20 pts)

The `SSD(.)` function accepts the following three arguments as inputs: `grayImage` (the gray-scale test image), `T` (the gray-scale template image), and `SSD_THRESH` (the threshold above which a match is rejected). To compute the SSD value for each pixel in the input image, use the following equation:

$$SSD(row, col) = \sum_{m,n}(T[m,n] - grayImage[row+m, col+n])^2. \quad (1)$$

For this part, you are **not allowed** to use any convolution, filtering, or template matching function provided by Matlab. Instead, you are required to implement the SSD between the template

and the test images in the 2D spatial domain (using nested for-loops and pixel operations). Note that based on your implementation, the code might take some time to run. After computing the SSD matrix, find the pixel with the minimum SSD value as the potential match location. Now we can apply the threshold, `SSD_THRESH`. If the SSD value of the match location is below the threshold, you should accept it as a valid matched point. The function should return the location (the row and the column) of the matched point.

2 NCC-based Template Matching (30 pts)

The `NCC(.)` function accepts the following three arguments as inputs: `grayImage` (the gray-scale test image), `T` (the gray-scale template image), and `NCC_THRESH` (the threshold below which a match is rejected). To compute the NCC value for each pixel in the input image, use the following equation:

$$NCC(row, col) = \frac{\sum_{m,n}(T[m,n]-\bar{T})(grayImage[row+m,col+n]-\overline{grayImage}_{row,col})}{\sqrt{\sum_{m,n}(T[m,n]-\bar{T})^2 \sum_{m,n}(grayImage[row+m,col+n]-\overline{grayImage}_{row,col})^2}}, \quad (2)$$

Where $\bar{T}$ is the mean of the template image and $\overline{grayImage}_{row,col}$ denotes the mean of an image patch, centered on the (row, col) pixel.

For this part, you are **not allowed** to use any convolution, filtering, or template matching function provided by Matlab. Instead, you are required to implement the NCC between the template and the test images in the 2D spatial domain (using nested for-loops and pixel operations). Note that based on your implementation, the code might take some time to run. After computing the NCC matrix, find the pixel with the maximum NCC value as the potential match location. Now we can apply the threshold, `NCC_THRESH`. If the NCC value of the match location is above the threshold, you should accept it as a valid matched point. The function should return the location (the row and the column) of the matched point.

3 Report

For the report, answer the following questions.

1) Display and compare the output images (overlaid images for the SSD- and the NCC-based template matching) of each test image. Based on your observations, which method is more suitable to template matching and why? (10 pts)
2) Use Matlab's built-in `imnoise(.)` function to add zero-mean Gaussian noise with the following variances $\sigma^2 = \{0.01, 0.03, 0.05, 0.1, 0.3, 0.5, 0.8, 1.0, 1.5, 2.0\}$ to the "Test Image (1).png" image.

   Plot the mean SSD and the mean NCC versus the noise variance for this image. The mean values are computed as the average of the SSD and the NCC values of the best match location over all the templates (do not consider the threshold values here).

   Compare the performance of SSD and NCC in dealing with noise. You do not need to submit the code for this question. (30 pts)

3) Explain how we can normalize the SSD equation given in (1) to compensate for possible intensity changes between the test images and the templates. Give the normalized equation. (10 pts)
4) **Bonus question (15 pts)**: Implement the normalized SSD method (add the new function `SSDnormed(grayImage, T, SSD_THRESH)` to the template code), display and compare its output images with the regular SSD method.