CS 558: Homework Assignment 2
Due: October 17, 6:00pm

Philippos Mordohai
Department of Computer Science
Stevens Institute of Technology
pmordoha@stevens.edu

Collaboration Policy. Homeworks will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems. Use of the Internet is allowed, but should not include searching for previous solutions or answers to the specific questions of the assignment. I will assume that, as participants in a graduate course, you will be taking the responsibility of making sure that you personally understand the solution to any work arising from collaboration.

Late Policy. A total of 3 late-days will be allowed in the entire semester without penalty. Days will be counted by rounding up, e.g. being 5 minutes late will be counted as one of the late-days. Assignments will no longer be accepted after all late-days have been used.

Deliverables. In a zip file include:

1. The complete **Jupyter notebook** with all cells executed.
2. The above in **pdf format**.
3. A **report in pdf format** explaining what you did and answering the questions in each problem.

Please read the following list carefully:

- You should not change the provided function specifications, but you can add your own functions as needed.
- You are not allowed to use **any** library functions that operate on the images other than to read and write the files and to display images on the screen. (See the NumPy tutorial for examples.) Math functions from NumPy are allowed, but no filtering, convolution, **resizing**, edge detection etc. functions that have not been implemented by you may be used. Using the **Numpy pad** function for zero-padding, and similar functions from other packages, is also not allowed.

Guidelines and Hints

- Specify the path to the input as in the skeleton code and append specific filenames in separate lines, so that we can replace them with our own. In other words, preserve the structure of the skeleton code.
- If you develop using a single Python script, you still have to submit a notebook by populating the boxes. Under the *Runtime* menu, there is an option for executing all cells.
- Do not activate *playground mode* so that your changes to the notebook are preserved.
- **Keep track of what x and y correspond to.** Which one spans the rows and which one spans the columns of the image. Inconsistent notation and usage can cause trouble.
- Avoid potential divisions by 0 by adding a small number to denominators when needed.
- Optimizing the code will make noticeable differences in runtime and allow you to develop faster.

Problem 1: Simple Template Matching (35 points)

Inputs: logo38.png, match_clean.png, match2.png, ..., match16.png

The objective of this problem is to detect a small Stevens logo (stored in logo38.png) in a larger image corrupted by salt and pepper noise using the **Sum of Absolute Differences (SAD)** and **Normalized Cross Correlation (NCC)**. The template appears twice in the inputs and has not been rotated or scaled. The goal is to find the correct translation and report the coordinates of the top left corner. **Detecting either instance of the logo is sufficient** in this problem. In other words, the goal is to pick the window with the best score or cost.

SAD is a cost implemented as follows:

$$SAD(x,y) = \sum_{k,l} |f[y+l, x+k] - t[l,k]|, \tag{1}$$

where f is the image and t is the template. The summation covers the entire width and height of the template, and (x, y) are the top left corner. Small values are indicators of good matches.

NCC is a score implemented as follows:

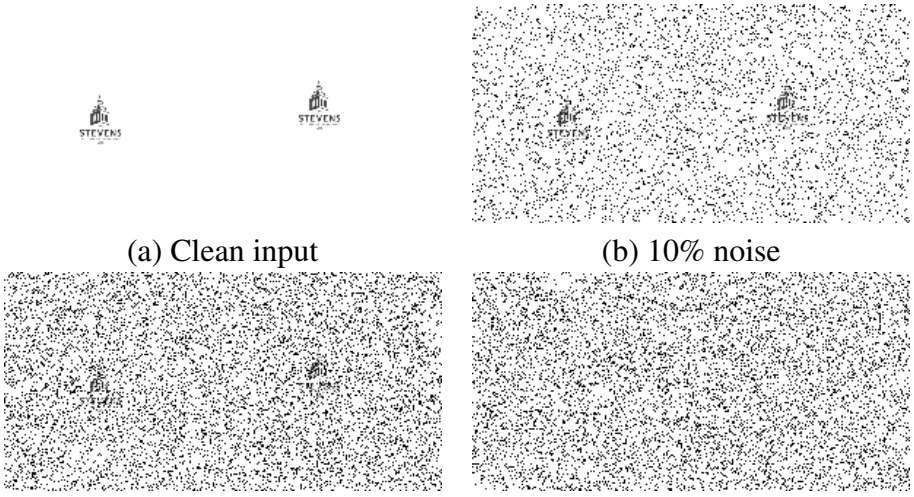
$$NCC(x,y) = \frac{\sum_{k,l}(f[y+l, x+k] - \bar{f}_{x,y})(t[l,k] - \bar{t})}{\sqrt{\sum_{k,l}(f[y+l, x+k] - \bar{f}_{x,y})^2 \sum_{k,l}(t[l,k] - \bar{t})^2}} \tag{2}$$

where $\bar{f}_{x,y}$ is the mean of f in the **window** under consideration and $\bar{t}$ is the mean of the template. NCC is a score and large values are indicators of good matches. Make sure that values are between -1 and 1, up to numerical precision. (This function can be optimized by identifying repeated computations and not repeating them. This will not receive credit, but will make development faster.)

The sequence of input images was generated by adding noise to the clean image as follows:

```
# add salt and pepper noise
for i in range(1,18):
  img = np.copy(img_clean)
  snp_corruption = 0.05*i
  mask_s = np.random.random(img.shape)
  mask_p = np.random.random(img.shape)
  img[mask_s < snp_corruption] = 0
  img[mask_p > 1-snp_corruption] = 255
  imshow(img)
  fname = 'match' + repr(i) + '.png'
  cv2.imwrite(os.path.join(rootpath, fname), img)
```

Only even steps of i are provided with the assignment. As a result, match2.png corresponds to 10% noise, match4.png corresponds to 20% noise etc. See also Fig. 1.



(a) Clean input (b) 10% noise

(c) 20% noise (d) 80% noise

Figure 1: Some of the inputs for Problem 1

Report: make a table or a graph showing the SAD and NCC values of the detected template, with the highest NCC or lowest SAD, for each noise level. Do not include trials in which the template was not detected. **Discuss** the performance of the two methods and provide a plausible for why one of them is more effective than the other.

Problem 2: Multi-scale Template Matching (35 points)

Inputs: logo38.png, multiscale.png

In this problem, four instances of the template have been blended with a background image, a map, as shown in Fig. 2. Two of those are small scale, provided in logo38.png, one is medium

scale, which is twice as large in width and height, and one is large scale, which is four times larger than the template in width and height. These are shown in Fig. 2(a).

Before being blended with the map, each instance of the template underwent an affine transformation separately. The new intensity of a pixel was computed by

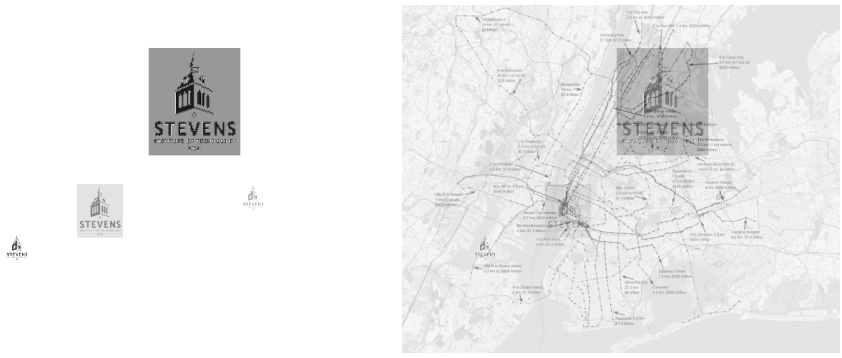
$$I'[y, x] = aI[y, x] + b. \tag{3}$$

The parameters a and b are intentionally not provided. The size of each instance was not altered during these operations.

The objective is to find **all four instances** at the three different scales. Compared to Problem 1, the challenge is implementing a way to detect the second best instance of the template at the small scale.

As you make the necessary scale adjustments by factors of 2 and 4, do not forget to low-pass filter appropriately. You can re-use your code from the previous assignment, but see the comment above about numpy.pad().

The same SAD and NCC functions from Problem 1 should be used directly.



(a) Templates only (b) Input image

Figure 2: The templates at the three different scales in isolation (not provided with the assignment) and the input image for Problem 2

Report: Observe whether SAD or NCC works better and provide a plausible explanation for this. Looking closely at Fig. 2 should be helpful for answering this question.

Problem 3: Hough Transform for Circles (30 points)

Inputs: circles.png

The input image contains 5 circles with 60 points each, as shown in Fig. 3. The radius was set to 100 in all cases, while the center varies. The objective is to detect the centers of the circles, taking advantage of the known radius, using a Hough transform.

To accomplish the objective, you must adjust the size of the accumulator bins, the step size used for generating the votes, as well as the number of votes required for detecting one circle. All circles can be detected simultaneously.

Report: Explain how the settings of the three parameters above affect the results. When are circles missed? How accurate is the estimated center?

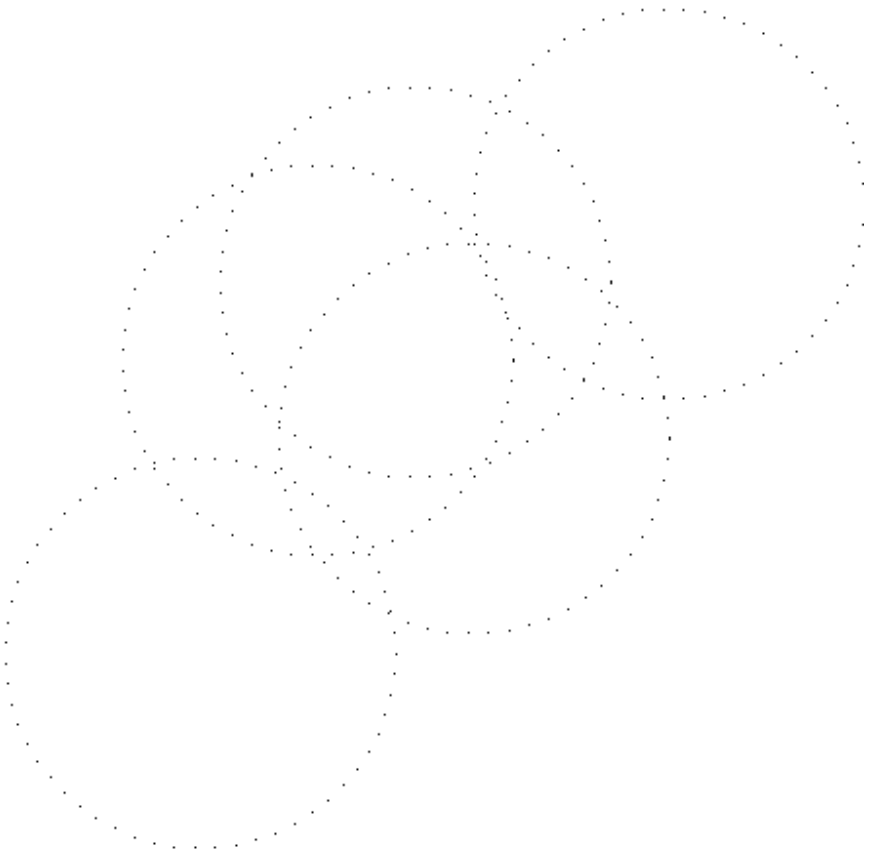


Figure 3: Circles to be detected via a Hough tansform