Introduction

This assignment addresses some of the topics we covered in class. There are three parts to it and the goals are:

1. To become familiar with reading, manipulating, displaying and writing images to file, and more specifically, using filters for image processing (30 points).

2. To implement template matching from scratch and test out the different matching techniques discussed in class (60 points).

3. To get familiar with the steps involved in implementing the Canny Edge Detector (10 points).

Download the homework1.zip file from myCourses
`Contents`→`Programming_assignments`→`Homework1`; this contains the instructions, starter code and data needed for the assignment.

Requirements

You should perform this assignment using Python along with any image library of your choice, and it is due on **Sunday September 13th by 11:59pm**. You are required to submit your code in a Jupyter notebook along with a brief report containing short write-ups based on the questions in the assignment. Your solutions should be zipped and uploaded to myCourses via Assignments before the deadline.

Your submitted zipped file for this assignment should be named **LastnameFirstname_hw1.zip** and should contain at least two files - *LastnameFirstname_hw1.pdf* and *LastnameFirstname_hw1.ipynb*. Feel free to submit any other auxiliary files including new images as needed. We should be able to execute your code for each part of the assignment from your Jupyter notebbok. Include a `Readme` file if necessary (especially if using several external libraries).

Failure to follow this convention could result in delays in getting your grade.

Problem 1. Image sharpening(30 points)

The goal of this problem is to get you to become familiar with basic image manipulations, including reading in images, writing out their modified versions and running filters with various variance σ values. For this part of the assignment, from the folder `images`, you are to read in the image *timnit.png* shown in Figure 1(left).

We have provided some helper files for you based on the `skimage` image library. These files help you (1) load an image and (2) build an $n \times n$ Gaussian filter with variance σ.

1) Load your image of choice. You may use either the image provided for you or take one of your own to demonstrate your image sharpening skills.

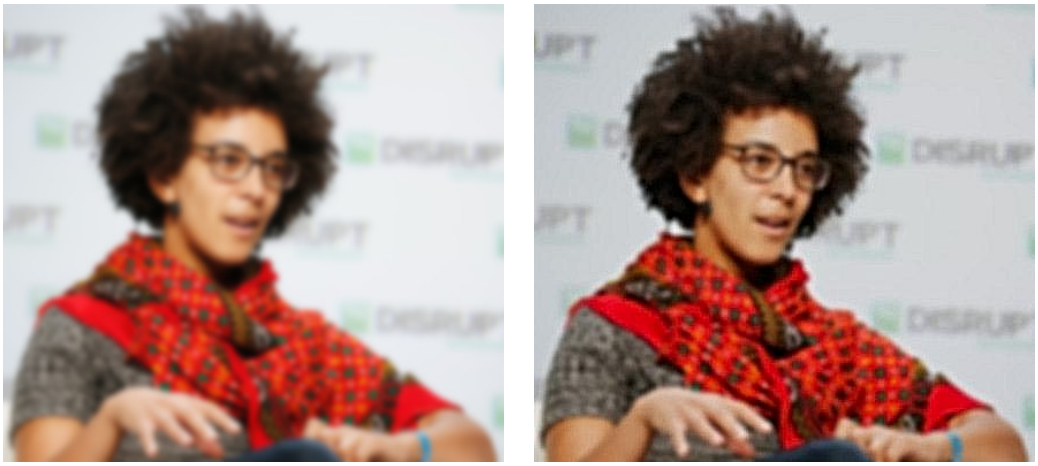


Figure 1: The left image shows a blurred original image and the right one is the result after sharpening.

2) Create a Gaussian filter g (the filter size that worked best on the timnit image (shown) is about 30-50). Use various values of σ starting from 1 till about 5. In your report, discuss if you see a difference in sharpening with the varying values of σ

3) Next, convert your image to *L.a.b.* color scale. The command to do this is given in the notebook.

4) Extract the first channel to get the intensity-only image to get a new image L.

5) Convolve this new image L with the filter g to get a smoothed image *Simg*.

6) Create a new image by multiplying L by a small constant r and *Simg* by another small constant s, then subtract; i.e. $r * L - s * Simg$

7) Normalize your new image so that its values are between 0 and 100; *L.a.b.* values need to fall in this range.

8) Recombine this new L image with the previous a and b channels of the *L.a.b.* image in step #3. This line of code is already provided for you.

9) Reconvert the *L.a.b.* image back to RGB with the command given in the Jupyter notebook. This code is already provided for you.

10) Examine your newly sharpened image and save it to file. Also display the original and sharpened image side-by-side in the notebook.

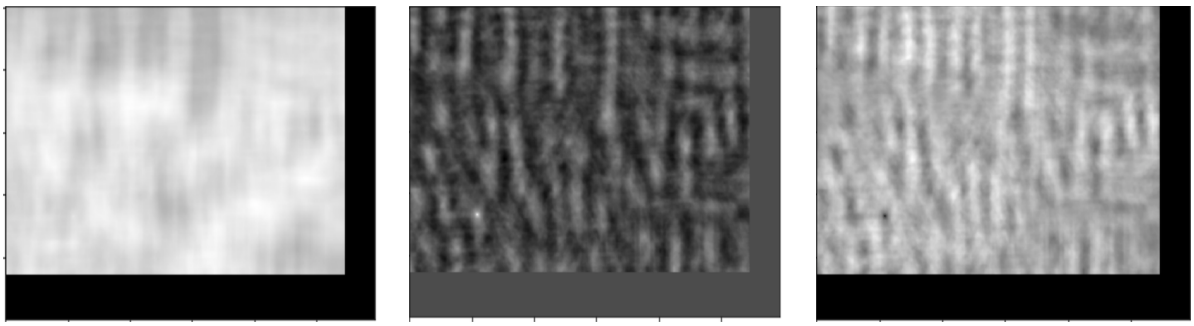
Note: You will have to find the appropriate values of the filter size $n \times n$, σ and the constants r and s that give you the best sharpened image. I used trial-and-error but you might be able to find a better way. But remember that these values might depend on the image being evaluated.
Your submission for this part should contain the all the code (including any auxilliary files) used to address this problem. Try to keep everything in the Jupyter notebook. We will run

your code on our own test images. Your PDF report should contain a short write-up on the influence of the window sizes and σ on your results.Also include pictures of various results you obtained to illustrate this.

Problem 2. Finding Waldo - Template Matcing (60 points)

(a) (b) (c)

Figure 2: (a) the original image; (b) the waldo template to be matched (up-scaled for demonstration only) (c) results after the 3 algorithms are run. Zoom in to see the 3 results at the arrows



(a) (b) (c)

Figure 3: (a) correlation map; (b) normalized cross correlation map (c) SSD map

For this second problem, we will be working with the popular cartoon character Waldo. The goal here is to try to match the given template across the image to find the location of the best match. Waldo hopefully, should be located at that point. But this is based on the correctness of the algorithm used. Your tasks for this problem will therefore be:

1) Load the image and template and convert them to grayscale. Code is provided to do this. Note that this process is NOT the same as the colorspace conversion from the previous problem. Take note of the different names of the color and grayscale images.

2) Write and call a cross-correlation function to find the best match for Waldo. If the symbol for correlation is $\circ$; if the image is represented by I and the template by f, then we want to compute $Z = I \circ f$. The tentative location of Waldo is at $\max(Z)$.

3) Use the helper function `draw_patch` to place a circular cyan patch on the newly computed location of Waldo on the original color image. Also, display the image of the correlation map Z (Figure 3a). obtained, alongside this image showing the location of Waldo (Figure 2c).

4) Write and call a normalized cross-correlation function to find the best match for Waldo. To compute the normalized version of a signal X:

$$\hat{X} = \frac{X - \bar{X}}{\sqrt{\sum (X - \bar{X})^2}}$$

where $\bar{X}$ is the mean of X. Hence, normalize both I and f, and then compute $Z = \hat{I} \circ \hat{f}$. The tentative location of Waldo here is at $\max(Z)$.

5) Again, use the helper function `draw_patch` to place a circular green patch on the newly computed location of Waldo in the color image. Also, display the image of the normalized correlation map Z (Figure 3b). obtained, alongside this image.

6) Write and call an SSD function to find the best match for Waldo. The formula for SSD between two signals is:

$$Z = \sum (I - f)^2$$

The tentative location of Waldo here is at $\min(Z)$. Note that this is the **min** value and not the max as in the other functions.

7) Lastly, use the helper function 'draw_patch' to place a circular red patch on the newly computed location of Waldo in the color image. Also, display the image of the SSD map Z (Figure 3c). obtained, alongside this image.

8) Briefly discuss your template matching results in the PDF file.

Do not use the library equivalent of the matching functions; write your own.

As in Problem 1, your submission should contain the all the code used to address this problem. We will run the code on other test images. Your PDF report should contain your responses to the questions/discussions solicited in the problem.

Problem 3. Canny Edge Detection (10 points)

Review the article at:
`https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123`

Indicate in your report whether or not you read the article.

This assignment is due on **Sunday September 13th by 11:59pm**.