COMP9517: Computer Vision

2022 T3 Lab 1 Specification

Maximum Marks Achievable: 2.5

This lab is **worth 2.5% of the total course marks**.

The lab files should be submitted online.

Instructions for submission will be posted closer to the deadline.

Deadline for submission is Week 3, Thursday 29 September 2022, 13:00:00.

Objectives: This lab revisits important concepts covered in the Week 1 and Week 2 lectures and aims to make you familiar with implementing specific algorithms.

Materials: The sample images to be used in all the questions of this lab are available in WebCMS3. You are required to use OpenCV 3+ with Python 3+.

Submission: Question 4 below is assessable **after the lab**. Submit your source code for this question as a Jupyter notebook (.ipynb) which includes all output (see coding requirements below) by the above deadline. The submission link will be announced in due time. Questions 1-3 are exercises for yourself to get experience with image processing and will not be assessed. Only Question 4 will be assessed.

1. Contrast Stretching

Contrast is a measure of the range of intensity values in an image and is defined as the difference between the maximum pixel value and minimum pixel value. The full contrast of an 8-bit image is 255 (max) – 0 (min) = 255. Any value less than that means the image has lower contrast than possible. Contrast stretching attempts to improve the contrast of the image by stretching the range of intensity values using linear scaling.

Assume that I is the original input image and O is the output image. Let a and b be the minimum and maximum pixel values allowed (for an 8-bit image that means $a = 0$ and $b = 255$) and let c and d be the minimum and maximum pixel values found in I. Then the contrast-stretched image O is given by the function:

$$O(x, y) = (I(x, y) - c)\left(\frac{b - a}{d - c}\right) + a \qquad (1)$$

Question 1: Write an algorithm that performs contrast stretching as per Equation (1) above.

Read the given gray-scale image **Cat.png** and run your algorithm to see whether it indeed improves the image quality. The result should look like this:

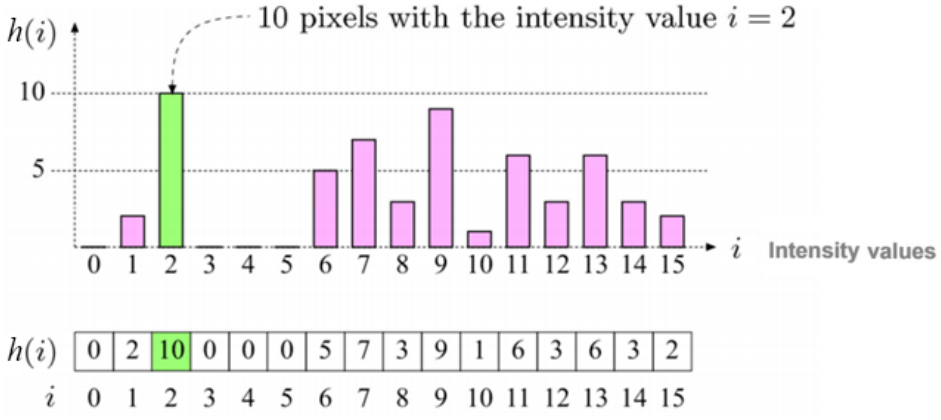
Input

Output

Also write an algorithm that finds the coordinates of the minimum pixel value and the coordinates of the maximum pixel value in an image. Do not use the existing library functions for these tasks but write your own code (see coding requirements below). Run it on both the input image and the output image and print the values of these pixels to confirm whether your contrast stretching algorithm works correctly.

2. Histogram Calculation

The histogram of an image shows the counts of the intensity values. It gives only statistical information about the pixels and removes the location information. For a digital image with L gray levels, from 0 to $L-1$, the histogram is a discrete function $h(i) = n_i$ where $i \in [0, L-1]$ is the ith gray level and n_i is the number of pixels with that gray level.



Question 2: Write an algorithm that computes the histogram of an image. Do not use existing library functions for computing the histogram but write your own code to perform this task (see coding requirements below). Then run your algorithm on **Cat.png** and its contrast enhanced version from Question 1 and visually compare the histograms (you may use existing functions to plot the histogram array computed by your algorithm).

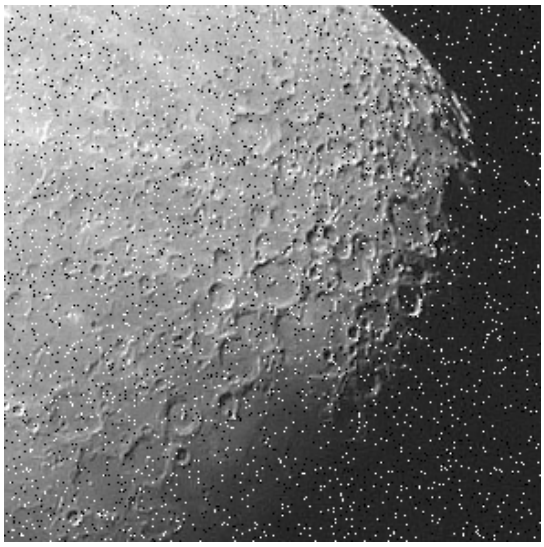
3. Image Smoothing

Reduction of noise (random intensity variations) in images can be achieved using image filtering. Salt and pepper noise, impulse noise, and Gaussian noise are some of the commonly observed types of noise in images. Different types of smoothing filters are suited for different types of noise. A smoothing filter is often called a low-pass filter as it allows low frequency components of the image to pass though (regions with similar intensity values) while suppressing high frequency components (edges or noise).

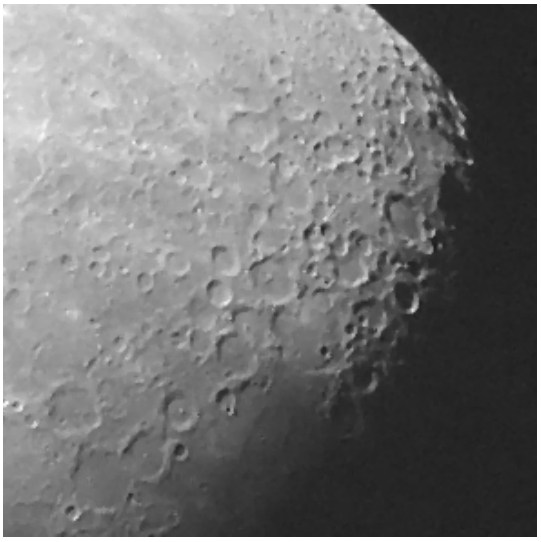
Question 3: Implement a mean (uniform) filter and a median filter. Do not use the existing library functions for convolution and image filtering but write your own code to perform this task (see coding requirements below). Perform noise removal on the input image **Moon.png**. Try both filters with different filter sizes, observe the differences in results, and decide which is the best filter and kernel size for this image.

Here is an example result (part of the image):

Input



Filtered



4. Edge Detection

Edges are an important source of semantic information in images. They occur in human visual perception at divisions between areas of different intensity, colour, or texture. A gray-scale image can be thought of as a 2D landscape with areas of different intensity living at different heights. A transition between areas of different intensity in an image I means there must be a steep slope, which we formalise as the gradient (vector):

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right) \tag{2}$$

As the image I is discrete, we need to approximate the continuous derivatives $\partial I/\partial x$ and

$\partial I/\partial y$ by finite differences. Simple examples of convolution kernels that perform finite differencing are the Sobel filters defined as follows:

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ and } S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Question 4 (2.5 marks): Write an algorithm that computes the two Sobel images $\partial I/\partial x \approx I * S_x$ and $\partial I/\partial y \approx I * S_y$ from an input image. Use the given image **CT.png** to test your algorithm. Do not use existing library functions for computing the Sobel images but write your own code to perform this task (see coding requirements below).

Notice that the calculations may produce negative output pixel values. Thus, make sure you use the right data types for the calculations and the output image.

After that, compute the gradient magnitude image:

$$\|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \tag{3}$$

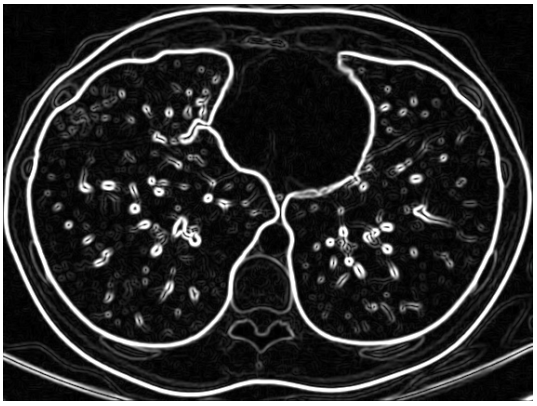
In other words, create a new output image having the same size as the input image and the Sobel images, and then for every pixel in the output image compute the value as the square root of the sum of the squared value of the Sobel image $\partial I/\partial x$ and the squared value of the Sobel image $\partial I/\partial y$ at that pixel position.

Here again, notice that the calculations may produce intermediate values outside the 8-bit range. Thus, make sure you use the right data types for the calculations.

The final result should look like this:

Input

Output



Coding Requirements

For all tasks in this lab, implement the required algorithms yourself, and do not use existing library functions (from OpenCV or any other packages) for these tasks. Using these functions instead of your own implementation will result in deduction of points.

Specifically, you should implement the following operations yourself using plain Python code without relying on existing functions that perform (part of) these operations directly:

- Find the minimum & maximum pixel values (write your own loop over the pixels).
- Perform contrast stretching (write your own loop over the pixels to modify their values).
- Count the number of pixels with a given value (to calculate the histogram of the image).
- Sort pixel values in a neighbourhood (to determine the median pixel value).
- Perform convolution of an image with a small kernel (write your own loops).
- Add, subtract, multiply, square (root) images pixelwise (write your own loops).

For other operations you may use existing library functions.

In future labs and in the group project you may use existing library functions, but in this lab (like in the assignment) the goal is to learn how basic image processing operations work at the pixel level and get experience implementing them yourself. Of course, you may use existing functions to verify the results of your own implementations.

Make sure that in your Jupyter notebook, the input images are readable from the location specified as an argument, and all output images and other requested results are displayed in the notebook environment. All cells in your notebook should have been executed so that the tutor/marker does not have to execute the notebook again to see the results.

Released: 21 September 2022