COMP9517: Computer Vision

2023 T2 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, Wednesday 14 June 2023, 18:00:00.

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, so you do not have to submit your code for these. 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 maximum possible 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 image **Landscape.png** and run your algorithm to see whether it indeed improves the contrast. Notice that this is a colour image, which has three channels (R, G, B), so you need to somehow apply your algorithm to the three channels.

There are different possibilities here. The most straightforward would be to apply it separately to each of the channels. Does this yield a good contrast-stretched colour image?

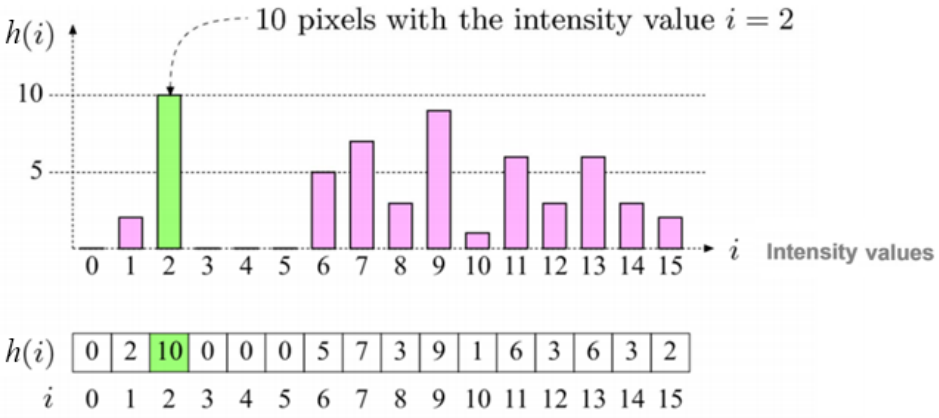
Alternatively, you could convert the colour image to a different colour space, such as HSV, and calculate the mapping function (1) on the value (V) channel, and then apply it to the original image channels (R, G, B). Does this yield a better contrast-stretched colour image?

Example: Input Output

Also write an algorithm that finds the coordinates of the first pixel (scanned from top-left to bottom-right) having the minimum value and the coordinates of the first pixel having the maximum value found in any given image channel. Do not use the existing library functions for these tasks but write your own code (see coding requirements below). Run your algorithm on both the input image and the output image and print the values of these two 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 the given image **Baboon.png** and plot the histograms of the R, G, B channels (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 given image **Noisy.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.

Example: 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 intensities at different heights. The Laplacian is a second-order derivative operator that can be used to find edges. It emphasizes pixels in areas of strong intensity changes and deemphasizes pixels in areas with slowly varying intensities. A simple 3×3 pixel convolution kernel that approximates the Laplacian operator is the following:

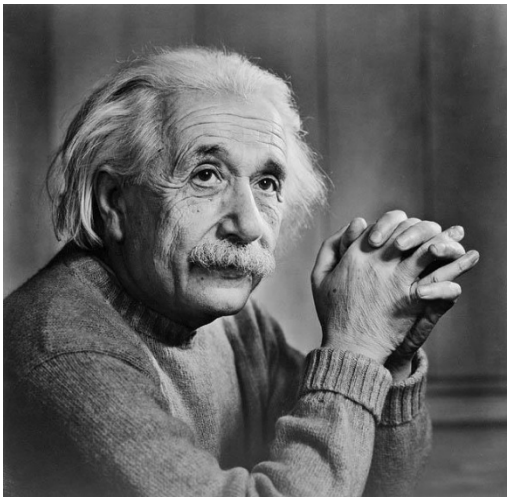
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Question 4 (2.5 marks): Write an algorithm that computes the Laplacian image of an input image using the above kernel. Use the given image **Einstein.png** to test your algorithm. Do not use existing library functions for computing the Laplacian image but write your own code to perform this task (see coding requirements below).

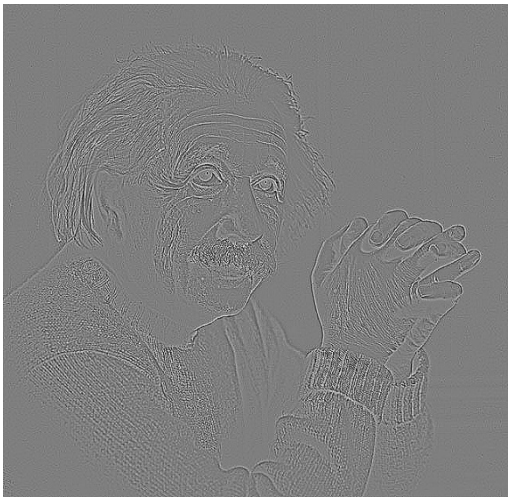
Notice that the calculations may produce negative output pixel values. Thus, make sure you use the right data types for the calculations and for the output image, and the right intensity mapping to display the output image (see the example below).

The final result should look like this (the output image has been contrast-enhanced):

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, unless stated otherwise. Using these functions instead of your own 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).
- Perform convolution of an image with a small kernel (write your own loops).
- Apply basic mathematical operations to images (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: 7 June 2023