# COMP9517: Computer Vision

# 2022 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, Tuesday 14 June 2022, 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) with output images (.png) in a single zip file 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.

**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:

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

**Question 1:** Write an algorithm that performs contrast stretching as per Equation (1) above. Read the given gray-scale image **Kitten.png** and run your algorithm to see whether it indeed improves the image quality. The result should look like this:

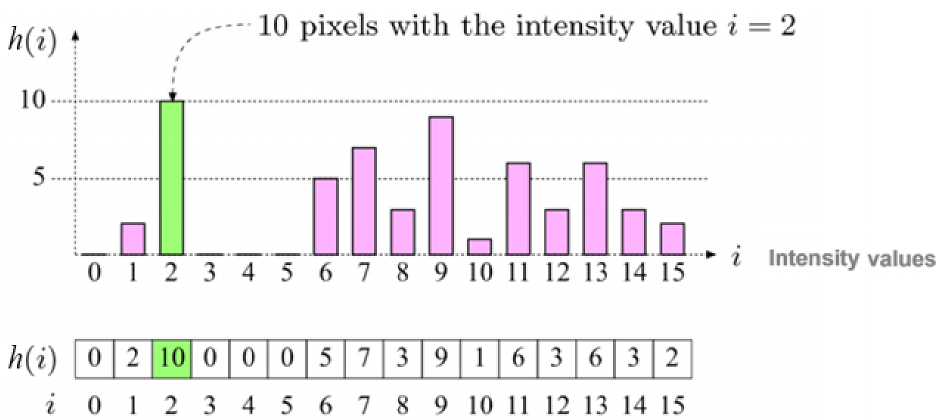
Input

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 built-in OpenCV functions for these tasks but write your own code. 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. Intensity Histogram

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 $i$th gray level and $n_i$ is the number of pixels with that gray level.




**Question 2:** Write an algorithm that computes and plots the histogram of an image. Do not use the built-in OpenCV functions for computing the histogram but write your own code to perform this task. Then run your algorithm on **Kitten.png** and its contrast-stretched version from Question 1 and visually compare the histograms.

**3. Image Edges**

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 3:** 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 the built-in OpenCV functions for computing the Sobel images but write your own code to perform this task. You may verify the output of your own algorithm by comparing with the output of built-in functions.

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:

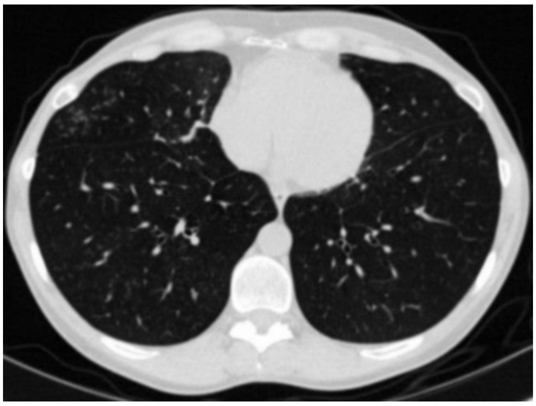
$$||\nabla I|| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \tag{3}$$

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.

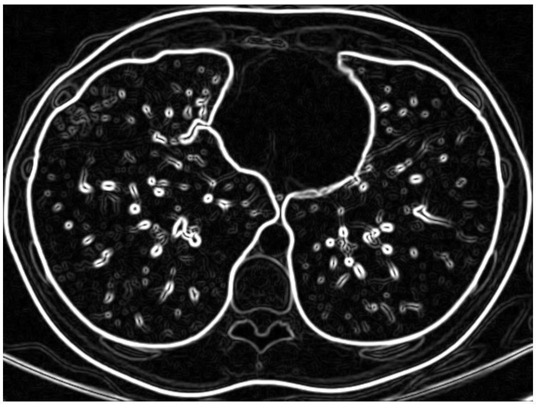
Here again, notice that the calculations may produce intermediate values outside the 8-bit range. Thus, make sure you use the right data types for the calculations.

The final result should look like this:

Input



Output



**4. Image Composition**

In both scientific and artistic image processing applications, images are often combined to create an output image with specific properties. Examples of this were shown in the last slides of the Image Processing Part 1 lecture. Typical image processing operations involved in this include both point operations (arithmetic operations and logical operations) and neighbourhood operations (image filtering).

**Question 4 (2.5 marks):** Write an algorithm that performs the following image processing steps to create a composed image of the two given input images **Cat.png** and **Dog.png**:

1. Apply edge detection to Cat.png using the Sobel filters (let's call the output image $C$).
2. Apply uniform filtering to Dog.png using a $5 \times 5$ kernel (let's call the output image $D$).
3. Compute a composed image $O(x, y) = 0.5 * C(x, y) + 0.5 * D(x, y)$.

Notice that the input images are RGB images, so the above steps must be applied to each channel to produce the composed RGB output image, which should look like this:

Cat

Dog

Composed

**Coding Requirements**

For all tasks, implement the required algorithms yourself, and **do not use 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.

In your Jupyter notebook, the input images should be readable from the location specified as an argument, and all output images and other requested results should be 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.

**Copyright:** UNSW CSE COMP9517 Team

**Released:** 7 June 2022