Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2013

Lab 4 - Introduction to Image Processing

Objectives

- To develop some Python functions that manipulate digital images.

Attendance/Demo

To receive credit for this lab, you must demonstrate your work. When you have finished all the exercises, call your instructor or a TA., who will review the functions you wrote. For those who don't finish early, the TAs will ask you to demonstrate whatever functions you've completed, starting at about 30 minutes before the end of the lab period.

References

SYSC_1005_F13_Image_Processing_1.pdf (available on cuLearn)**.**

Instructions

Getting Started

Step 1: Create a new folder named Lab 4.

Step 2: Copy lab_4.py, Cimpl.py and the three image (JPEG) files to your Lab 4 folder.

Step 3: Launch Wing IDE 101.

Part 1 - Learning to Use the Cimpl Module

Exercise 1

Step 1: From the menu bar, select File > Open... and load lab_4.py. This module contains the `maximize_red` and `make_sunset` functions that were presented in class.

Step 2: Click the Run button. This will load the code in lab_4.py into the Python interpreter and check it for syntax errors.

Step 3: You're now going to choose one of the three images, display it, maximize the amount of red in the image, and display the modified image.

Important: When you execute a command in the Python shell, the message

Executing command. Please wait for result.

appears at the top of the Python Shell window. Functions that modify images will typically take 3 or 4 seconds to execute. **Do not type commands until the function has finished and the shell prompt (>>>) is displayed. If you type while a function is executing, there is a good chance that Wing IDE will hang.** If this happens, you will have to exit the IDE and relaunch it.

In the Python shell, type the following statements:

```
>>> img =load_image(choose_file())
>>> show(img)
>>> maximize_red(img)
>>> show(img)
```

The shell prompt won't reappear until you click the Close button of the window in which the image is displayed.

Review the image processing lecture slides that are posted on cuLearn. Read the definition of `maximize_red`, and make sure you understand what every statement in the function does.

Part 2 - Developing Some Simple Image Manipulation Functions

Exercise 2

Step 1: In lab_4.py, define a function named `remove_blue` that is passed an image as an argument. This function should set the blue component of each pixel to 0, leaving the red and green components unchanged.

Step 2: Save the edited file, then click the Run button. (If you forget to do this, you won't be able to call `remove_blue` from the shell, because you haven't loaded the modified module into the Python interpreter.) Notice that clicking Run resets the shell, which means you'll have to choose an image file and load the image before calling `remove_blue`. Remember to bind the `Image` object returned by `load_image` to a variable; e.g., `img`, then call `remove_blue` with `img` as an argument. After `remove_blue` returns, display the modified image.

Exercise 3

Read the definition of `make_sunset`. This function creates a sunset scene by reducing the green and blue components of every pixel by 30%.

Using the Python shell, load an image, call `make_sunset` to modify it, and display the modified image.

Exercise 4

In lab_4.py, define a function named `halve_colours` that is passed an image as an argument. This function should change the red, green and blue components of each pixel to half their current values. Use the shell to test your function.

Exercise 5

In lab_4.py, define a function named `swap_red_blue` that is passed an image as an argument. This function should swap each pixel's red and blue components. For example, if a pixel's red and blue components are 127 and 41, respectively, then the red and blue components should be set to 41 and 127, respectively. Use the shell to test your function.

Part 3 - Defining Some Filters for a Photo Editor Application

You'll now start developing some *image filtering* functions for an application that will allow you to edit digital photographs. All the functions will be stored in a module named `filters`.

If you don't finish this part during today's lab, you must finish it on your own time as "homework" and be prepared to demonstrate your solutions for Exercises 6 and 7 functions at the start of your next lab period.

Step 1: Open a new editor window and save it in a file named filters.py.

Exercise 6

In filters.py, define a function that is passed an image and converts it to a colour negative of the original image. The function header is:

```
def negative(img):
```

To create a negative image, we change the red, green and blue components of each pixel to the "opposite" of their current values. What do we mean by "opposite"? Recall that a component's value can range from 0 (lowest intensity) to 255 (highest intensity). If the red component is 0, its opposite value is 255. If the green component is 255, its opposite value is 0. If the blue component is 140 (slightly above the midpoint in the range of intensities), its opposite value is 115 (i.e., slightly below the midpoint). Using this information, derive a formula that maps v to $f(v)$, where v is the original value of a component, and $f(v)$ is the corresponding opposite value. After you've derived the formula, design the code for `negative`.

Use the shell to test your function.

Exercise 7

In filters.py, define a function that is passed an image and converts it to grayscale. The function header is:

```
def grayscale(img):
```

To create a grayscale image, we change each pixel's colour to an equivalent shade of gray. A colour is a shade of gray whenever the red, green and blue components have the same value. As such, RGB supports 256 shades of gray, from (0, 0, 0) to (255, 255, 255).

One strategy for converting an RGB colour to an equivalent shade of gray is to adjust the red, green and blue components, while keeping the *brightness* of the gray shade the same as the brightness of the original colour. To determine a colour's brightness, we calculate the average of its red, green and blue components. (The higher the average, the brighter the colour.) So, to change a pixel's colour to the corresponding shade of gray, we calculate the average value of its red, green and blue components, then set all three of the components to this average value. (Remember, RGB component values are stored as integers, so any fractional portion of the average will be lost when a colour is converted to gray. As such, a pixel's brightness may change slightly when you change it to gray, but it will be very close to the original brightness.)

Use the shell to test your function.

Wrap-up

1. Remember to have a TA review your solutions to the exercises and give you a grade (Satisfactory, Marginal or Unsatisfactory) before you leave the lab.
2. You'll be adding functions to your `filters` module during Lab 5, so remember to save a copy of your filters.py file (copy it to a flash drive, or email a copy to yourself, or store it on your M: drive - remember, files left on your desktop or in your Documents folder are not guaranteed to be there the next time you log in).
3. Log in to cuLearn, click the Submit Lab 4 link and submit lab4.py and filters.py. After you click the Add submission button, drag both files to the File submissions box. After icons for both files appear in the box, click the Save changes button. At this point, the submission status for your files is "Draft (not submitted)". You can resubmit one or both files by clicking the Edit my submission button. After you've finished uploading your files, remember to click the Submit assignment button to change the submission status to "Submitted for grading".

Posted: Tuesday, October 1, 2013