Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2014

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 make reasonable progress towards completing the exercises and demonstrate the code you complete. **Also, you must submit your lab work to cuLearn by the end of the lab period**. (Instructions are provided in the *Wrap Up* section at the end of this handout.)

When you have finished all the exercises, call a TA, who will review the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

References

Course materials on image processing are posted on cuLearn.

Getting Started

Step 1: Create a new folder named Lab 4.

Step 2: Download lab_4.py, filters.py, Cimpl.py and the image (JPEG) files from cuLearn to your Lab 4 folder.

Step 3: Launch Wing IDE 101. Check the message Python displays in the shell window, and verify that Wing is running Python version 3.4. If another version is running, ask your instructor or a TA for help to reconfigure Wing to run Python 3.4.

Part 1 - Learning to Use the Cimpl Module

Exercise 1

Step 1: Open lab_4.py in a Wing IDE editor window. This module contains the definitions of the `maximize_red` and `make_sunset` functions that were presented in class.

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

Step 3: You're now going to use Cimpl to choose one of the image files, load it into memory and display it. You'll then call a filter to 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 can take 3 or 4 seconds (or longer) 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 this statement to select an image file and load it into memory:

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

You may have to move some of the windows on the screen in order to see the file chooser dialogue box. Select one of the JPEG files in your Lab 4 folder, then click the Open button. The contents of the selected image file will be decoded and stored in an object, then the object will be bound to variable `img`.

What is the type of the object that is bound to `img`? To find out, call Python's type function:

```
>>> type(img)
```

Will evaluating `img` in the shell display the image that is bound the variable? Try it:

```
>>> img
```

Clearly, evaluating `img` does not cause the image to be displayed.

Call `Cimpl`'s `show` function to open a window that displays the image:

```
>>> show(img)
```

The shell prompt won't reappear until you click the window's Close button.

Now call the `maximize_red` filter, passing it image that we want to modify:

```
>>> maximize_red(img)
```

Call `show` to display the modified image:

```
>>> show(img)
```

Before you move on to Part 2, review the material on image processing (lecture slides, etc.) that is posted on cuLearn. Make sure you understand what every statement in the basic filter pattern does. Read the definition of `maximize_red` and make sure you understand how that function implements the basic filter pattern.

Part 2 - Developing Some Simple Image Manipulation Functions

Exercise 2

In one of the lectures, we learned that a colour in the RGB colour model can be represented by a triplet of integer components, each of which ranges from 0 to 255. The first component is the quantity of red, the second component is the quantity of green, and the third component is the quantity of blue.

Each pixel in a display screen has three light sources placed close together, so we perceive them as a single "dot" in the display. One source emits red light, another emits green light and the third emits blue light. When a pixel is illuminated, the display hardware uses the RGB triplet for that pixel to set the intensity of light that is emitted from each of the three sources. Our vision system interprets the intensities of the red, green and blue light as one of the 16,777,216 colours that can be represented by an RGB triplet.

We can think of an RGB image as being composed of three monochromatic images (one consisting of shades of red, another consisting of shades of green, and the third consisting of shades of blue). These monochromatic images are often referred to as *channels*. We can build filters that modify an image to reveal the separate red, green and blue channels.

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

Step 2: Use the shell to test `red_channel`. To do this:

- Save the edited file, then click the Run button. (If you forget to do this, you won't be able to call `red_channel` from the shell, because you haven't saved the modified module and loaded it 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 `red_channel`. Remember to bind the `Image` object returned by `load_image` to a variable; e.g., `img`
- Call `red_channel` with `img` as an argument.
- After `red_channel` returns, display the modified image. What colours are in the modified image?

Step 3: In lab_4.py, define a function named `green_channel` that is passed an image as an argument. This function should set the red and blue components of each pixel to 0, leaving the green component unchanged.

Step 4: Use the shell to test `green_channel`; that is, load an image file, call `green_channel` to modify the image, and display the modified image. Remember to pass an original, unmodified image to `green_channel`, and not the modified image produced by `red_channel`! What colours are in the modified image?

Step 5: In lab_4.py, define a function named `blue_channel` that is passed an image as an argument. This function should set the red and green components of each pixel to 0, leaving the blue component unchanged.

Step 6: Use the shell to test `blue_channel`. What colours are in 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 one of the lectures, you learned that a simple way to quantify a pixel's brightness is to calculate the average of its red, green and blue components. A simple way to reduce an image's brightness by some percentage is to reduce every pixel's red, green and blue components by the same percentage.

In lab_4.py, define a function named `halve_brightness` that is passed an image as an argument. This function should reduce the image's brightness by 50% by setting the red, green and blue components of every pixel to half their current values.

Use the shell to test `halve_brightness`.

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 `swap_red_blue`.

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 filters.py in a Wing IDE editor window. This module contains the `grayscale` function that was presented in class.

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 `negative`.

Exercise 7 - Converting an Image to Grayscale, Revisited (Improving the Algorithm)

This exercise illustrates an important software engineering technique: sometimes, we implement a function using a simple algorithm that does what we want, and after we finish testing the function, we rework it to increase its efficiency or improve the results it produces.

Review the `grayscale` function that was presented in class. Recall that this filter converts a pixel's colour to a shade of gray by using this formula to calculate the pixel's *brightness* (the average of its red, green and blue components):

```
brightness = (red + green + blue) // 3
```

and setting all three components to this average.

This calculation is (almost) equivalent to this formula:

```
brightness = red × 0.3333 + green × 0.3333 + blue × 0.3333
```

In other words, the red, green and blue components are weighted equally when calculating the average, with each component contributing 33-and-a-third percent.

This way of calculating grayscale does not take into account how the human eye perceives brightness, because we perceive blue to be darker than red. A better way of averaging the three components is to change the weight of the red component (to 29.9%), the green component (to 58.7%) and the blue component (to 11.4%):

```
brightness = red × 0.299 + green × 0.587 + blue × 0.114
```

In filters.py, make a copy of your `grayscale` function, and rename this copy `weighted_grayscale`. **Don't modify your original grayscale filter.**

Edit `weighted_grayscale` to use the weighted averaging formula described earlier. For each pixel, calculate the weighted average of the pixel's red, green and blue components, then set the red, green and blue components to that value.

Interactively test your weighted grayscale filter. Compare the images produced by this function with the images produced your original `grayscale` function. In your opinion, which grayscale filter produces better-looking images?

Wrap-up

1. Remember to have a TA review your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the demo/sign-out sheet.

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. Before you leave the lab, log in to cuLearn and submit lab_4.py and filters.py (the files containing your solutions to Parts 2 and 3, not the unedited ones you downloaded form cuLearn!) To do this:

    3.1. Click the Submit Lab 4 link. A page containing instructions and your submission status will be displayed. After you've read the instructions, click the Add submission button. A page containing a File submissions box will appear. Drag lab_4.py and filters.py to the File submissions box. Do not submit another type of file (e.g., a zip file, a RAR file, a .txt file, etc.)

    3.2. After the icons for both .py files appear in the box, click the Save changes button. At

this point, the submission status of your file is "Draft (not submitted)". If you're ready to finish submitting the file, jump to Step 3.4. If you instead want to replace or delete your "draft" file submission, follow the instructions in Step 3.3.

3.3. You can replace or delete the file by clicking the Edit my submission button. The page containing the File submissions box will appear.

3.3.1. To overwrite a file you previously submitted with a file having the same name, drag another copy of the file to the File submissions box, then click the Overwrite button when you are told the file exists ("There is already a file called..."). After the icon for the file reappears in the box, click the Save changes button.

3.3.2. To delete a file you previously submitted, click its icon. A dialogue box will appear. Click the Delete button., then click the OK button when you are asked, "Are you sure you want to delete this file?" After the icon for the file disappears, click the Save changes button.

3.4. Once you're sure that you don't want to make any changes, click the Submit assignment button. A Submit assignment page will be displayed containing the message, "Are you sure you want to submit your work for grading? You will not be able to make any more changes." Click the Continue button to confirm that you are ready to submit your lab work. This will change the submission status to "Submitted for grading". **Make sure you do this before you leave the lab.**

Extra Practice

Exercise 8

We can design filters that hide an original image inside a modified image. Here is one approach:

- for every pixel, calculate the the pixel's brightness, using the same formula as a grayscale filter. Then, replace the red component with this brightness value, divided by 10.
- for every pixel, replace the green component with a random integer in the range 0 to 255, inclusive. Do the same thing with the blue component of every pixel.
    - the easiest way to generate a random integer is to call the `randint` function in Python's `random` module. For example:

      ```
      import random
      i = random.randint(a, b)
      ```

      `randint` returns a random integer `n` such that `a` <= `n` <= `b`.

Continued on the next page.

Define a function named `hide` that is passed an image and hides it, using the approach described here. All the steps can be placed inside a single loop; that is, your function doesn't need to iterate over the image several times.

The green and blue "snow" in the modified image will obscure the original image.

Exercise 9

Define a function named `recover` that is passed an image that was prepared by your `hide` filter. This function recovers an approximation of the original hidden image.