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

Lab 5 - Image Processing, Continued: Developing Filters that Selectively Modify Pixels

Objectives

To learn the syntax and semantics of Python's `if`, `if-else` and `if-elif-else` statements, by developing functions that selectively modify pixels in 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 5.

Step 2: Copy filters.py (the file containing your solutions to Lab 4, Part 3, not the original file obtained from cuLearn) to your Lab 5 folder. You will also need to copy Cimpl.py and the three image (JPEG) files from last week's lab to this folder. Finally, download filters_if_statements.py from cuLearn. This file contains the filters that were presented last week's lectures.

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.

Step 4: Open filters.py in a Wing IDE editor window. Don't delete the `negative` , `grayscale` and `weighted_grayscale` functions from Lab 4!

Step 5: First, you'll add copies of the filters presented in last week's lectures to filters.py. Open filters_if_statements.py in a Wing IDE editor window. Copy/paste the definitions of the `solarize`, `black_and_white` and `black_and_white_and_gray` filters from filters_if_statements.py

to filters.py. Close filters_if_statements.py.

Exercise 1 - Modifying the Solarizing Filter (`if` Statements)

Currently, `solarize` compares each RGB component to 128. This number was chosen as the threshold because it's halfway between the largest and smallest component values (0 and 255).

Change this filter so that each component is changed only if its value is less than a specified integer threshold value. You will need to edit the function header, adding a parameter named `threshold`:

```
def solarize(img, threshold):
```

Use the Python shell to interactively test your function with threshold values 64, 128 and 192. (Remember to reload the original image each time before you call the function; otherwise, you'll be modifying an image that has already been solarized.) What effect does increasing the threshold have?

Predict what the modified image will look like if 0 is passed as the threshold value. Call `solarize` with this argument, and see if your prediction was correct.

What threshold value will cause `solarize` to create the same effect as your `negative` function from Lab 4? Try it!

Exercise 2 - Extreme Contrast (Using `if-else` Statements)

Review the definition of the `black_and_white` filter. Make sure you understand how the `if-else` statement determines if a pixel's colour will be changed to black or white. (cuLearn has a link to an Online Python Tutor example that will help you visualize the execution of this statement.)

In filters.py, define a function that is passed an image and maximizes the contrast between pixels (light pixels are made lighter, and dark pixels are made darker). The function header is:

```
def extreme_contrast(img):
```

This function will change the red, green and blue components of each pixel. If a component's value is between 0 and 127 (i.e., it's relatively dark), the component is changed to 0. If a component's value is between 128 and 255 (i.e., it's relatively light), the component is changed to 255. Use these new component values to create the pixel's new colour.

- How many different colours could there be in an image that has been modified by this filter? Explain your answer.

When defining this function, use `if-else` statements. Do not use `if` statements or `if-elif-else` statements.

Use the shell to interactively test your function.

Exercise 3 - Sepia Tinting (Using `if-elif-else` Statements)

Review the definition of the `black_and_white_and_gray` filter. Make sure you understand how the `if-elif-else` statement determines if a pixel's colour is changed to black or white or gray. (cuLearn has a link to an Online Python Tutor example that will help you visualize the execution of this statement.)

I'm sure you've seen old black-and-white (actually, grayscale) photos that, over time, have gained a yellowish tint. We can mimic this effect by creating sepia-toned images.

In filters.py, define a function that is passed an image and transforms it by sepia-tinting it.The function header is:

```
def sepia_tint(img):
```

There are several different ways to sepia-tint an image. One of the simplest approaches involves determining if each pixel's red component lies in a particular range of values, and if it does, modifying the pixel's red and blue components, leaving the green component unchanged. Here's the algorithm:

First, we convert the image to grayscale, because old photographic prints were grayscale. (To do this, your function must call your `grayscale` function. Don't replicate the grayscale algorithm in `sepia_tint`.) After this conversion, each pixel is a shade of gray; that is, its red, green and blue components are equal.

Next, we tint each pixel so that it's a bit yellow. In the RGB system, yellow is a mixture of red and green. To make a pixel appear slightly more yellow, we can simply decrease its blue component by a small percentage. If we also increase the red component by the same percentage, the brightness of the tinted pixel is essentially unchanged. The amount by which we change a pixel's red and blue components depends on whether the pixel is a dark gray, a medium gray, or a light gray:

- If the red component is less than 63, the pixel is in a shadowed area (it's a dark gray), so the blue component is decreased by multiplying it by 0.9 and the red component is increased by multiplying it by 1.1;
- If the red component is between 63 and 191 inclusive, the pixel is a medium gray. The blue component is decreased by multiplying it by 0.85 and the red component is increased by multiplying it by 1.15;
- If the red component is greater than 191, the pixel is in a highlighted area (it's a light gray), so the blue component is decreased by multiplying it by 0.93 and the red component is increased by multiplying it by 1.08.

When defining this function, use `if-elif-else` statements. Do not use `if` statements or `if-else` statements.

Use the shell to interactively test your sepia-tinting function.

Exercise 4 - Range Checking

In filters.py, define a *helper function* named `_adjust_component` that is passed the value of a pixel's red, green or blue component. (Yes, the function's name begins with an underscore. A convention followed by Python programmers is to use a leading underscore to indicate that a function is intended for internal use only; in other words, that it should only be called by functions in the same module.) Here is the function header:

```
def _adjust_component(amount):
```

Note: parameter `amount` is a value of type `int`, not an image or a value of type `Color`.

If `amount` is between 0 and 63, inclusive, the function returns 31. If `amount` is between 64 and 127, inclusive, the function returns 95. If `amount` is between 128 and 191, inclusive, the function returns 159. If `amount` is between 192 and 255, inclusive, the function returns 223. Your function should assume that the integer bound to `amount` will always lie between 0 and 255, inclusive; in other words, it does not need to check if `amount` is negative or greater than 255.

Important:

- the value returned by this function must be an `int`, not a `float` or a `str` (character string);
- this function does not print anything; i.e., it must not call Python's `print` function.

Interactively test your function; for example, try:

```
>>> _adjust_component(10)
31
>>> _adjust_component(85)
95
>>> _adjust_component(142)
159
>>> _adjust_component(230)
223
```

These test cases use values picked from each of the four quadrants, but four tests aren't sufficient. It's also a good idea to have test cases that check values at the upper and lower limits of each quadrant. Interactively test your function to verify that:

- `_adjust_component(0)` and `_adjust_component(63)` return 31;
- `_adjust_component(64)` and `_adjust_component(127)` return 95;
- `_adjust_component(128)` and `_adjust_component(191)` return 159;
- `_adjust_component(192)` and `_adjust_component(255)` return 223.

Finish this exercise before your attempt Exercise 5.

Exercise 5 - Posterizing an Image

Posterizing is a process in which we change an image to have a smaller number of colours than the original. Here is one way to do this:

Recall that a pixel's red, green and blue components have values between 0 and 255, inclusive. Suppose we divide this range into four equal-size quadrants: 0 to 63, 64 to 127, 128 to 191, and 192 to 255. We can use the _adjust_component function you wrote for Exercise 4 to determine the quadrant in which a component lies and return the value that is in the middle of the quadrant.

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

```
def posterize(img):
```

For each pixel, change the red component to the midpoint of the quadrant in which it lies. Do the same thing for the green and blue components. In other words, for each pixel, your function must call your `_adjust_component` function three times (once for each of the pixel's components).

Interactively test your posterizing function.

Exercise 6 - Five-Colour Images (Primary Colours, plus Black and White)

In filters.py, define a function that is passed an image and changes each pixel's colour to white, black, or one of three primary colours (red, green and blue). The function header is:

```
def simplify(img):
```

For every pixel, the function compares the pixel's red, green and blue components to one or two threshold values:

- If all three components are above the high threshold (200), the pixel's colour is changed to white (255, 255, 255);
- If all three components are below the low threshold (50), the pixel's colour is changed to black (0, 0, 0).

If the pixel's colour isn't changed to white or black, the function performs more comparisons on the components:

- If the red component is greater than both the green component and the blue component, the pixel's colour is changed to red; i.e., the RGB triple (255, 0, 0);
- If the green component is greater than both the red component and the blue component, the pixel's colour is changed to green; i.e., the RGB triple (0, 255, 0);
- Otherwise, the pixel's colour is changed to blue; i.e., the RGB triple (0, 0, 255). (This handles

the case where the blue component is greater than both the red component and the green component, plus all the other cases that can occur; for example, two of a pixel's three components are equal, all three of its components are equal, etc.)

Hint: this function doesn't require many lines of code if you use an `if-elif` statement, and use one or more of Python's Boolean operators (`and`, `or`, `not`) in the conditions. If you're not sure how to start, review the `swap_black_white` function presented in class.

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. If you've finished all the exercises, your `filters` module should contain 10 filters: `negative`, `grayscale` and `weighted_grayscale` (from Lab 4) and `solarize`, `black_and_white`, `black_and_white_and_gray`, `extreme_contrast`, `sepia_tint`, `posterize` and `simplify`. (Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.)

   You'll be adding functions to your `filters` module during Lab 6, 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 filters.py. To do this:

   3.1. Click the Submit Lab 5 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 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 icon for filters.py files appears 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.**