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

Lab 6 - Image Processing, Continued: Advanced Techniques

Objectives
To develop more Python functions that manipulate digital images.

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

References
Image processing lecture notes and examples are posted on cuLearn.

Instructions

Getting Started
Step 1: You can use your Lab 6 folder from last week's lab. This folder should contain Cimpl.py, the three image (JPEG) files, and the filters.py module you've been developing over the past couple of weeks.

Step 2: Launch Wing IDE 101 and open filters.py. Don't delete the filter functions you developed during Labs 4 and 5!

Exercise 1 - Converting an Image to Grayscale, Revisited (Improving the Algorithm)
This exercise illustrates an important software engineering technique: sometimes, we first implement a function using a simple algorithm that does (approximately) what we want, and once the function is working, we rework it to increase its efficiency or improve the results it produces.

Review the `grayscale` function you wrote for Lab 4. Recall that this filter converts a pixel's colour to a shade of gray by using this formula to calculate the average of its red, green and blue components:

(red + green + blue) / 3

and setting all three components to this average.

This calculation is (almost) equivalent to this formula:

(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 weights of the red component (to 29.9%), the green component (to 58.7%) and the blue component (to 11.4%):

(red × 0.299 + green × 0.587 + blue × 0.114)

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 average 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?

Exercise 2 - Edge Detection

Edge detection is a technique that results in an image that looks like a pencil sketch, by changing the pixels' colours to black or white.

In filters.py, define a function that is passed an image and transforms it using edge detection. The function header is:

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

Parameter `threshold` is a positive real number.

A simple algorithm for performing edge detection is: for every pixel that has a pixel below it, check the *contrast* between the two pixels. If the contrast is high, change the top pixel's colour to black, and if the contrast is low, change the top pixel's colour to white.

One way to calculate the contrast between two colours is to calculate the average of the red, green and blue components of the top pixel (with all three components weighted equally), and subtract the average of the red, green and blue components of the pixel below it. We then calculate the absolute value of this difference. If this absolute value is greater than the filter's threshold attribute, the contrast between the two pixels is high, so we change the top pixel's colour to black; otherwise, the contrast between the two pixels is low, so we change the top pixel's colour to white.

Hint: your function needs to process one row of pixels at a time, and for a given pixel, it will need to access the pixel below it. Don't use the

```
for x, y, col in img:
```

pattern that you used in your other filters. Instead, use nested for loops to generate the (x, y) coordinates of pixels, and call `get_color` to get the `Color` objects for individual pixels. For examples of how do this, have a look at the functions that were presented in lectures during Week 5 and 6, which are posted on cuLearn..

Interactively test your edge detection filter. When calling the function, use 10.0 as the threshold. Repeat your tests with different thresholds. What range of threshold values result in images that look good to your eyes?

Exercise 3 - Improved Edge Detection
Make a copy of your `detect_edges` function, and rename this copy `detect_edges_better`. Don't modify your original edge detection filter.

Edit `detect_edges_better` so that it checks the contrast between each pixel and the pixel below it *as well as the pixel to the right of it*. As before, we calculate the contrast of two pixels by averaging the red, green and blue components of each pixel, subtracting the averages, and calculating the absolute value of the difference. We change a pixel's colour to black only if the contrast between the pixel and the one below it is high (i.e., the absolute value of the difference exceeds the filter's threshold attribute) *or* the contrast between the pixel and the one to the right of it is high. Otherwise, we change the pixel's colour to white.

Interactively test your function, using 10.0 as the threshold. In your opinion, does this filter do a better job of edge detection than the one you developed for Exercise 2?

Exercise 4 - Changing the Blurring Filter
From cuLearn, download the Python file containing the simple blurring filter. (The design and code of this filter was presented during the October 15 lecture.) Open the file in Wing IDE 101, and copy/paste the `blur` function into filters.py.

Currently, the filter calculates the component values for the new colour of each blurred pixel by averaging the components of the pixel, the pixel above it, the pixel below it, the pixel to the left of it, and the pixel to the right of it.

Another way to create the new colour for each blurred pixel is to average the pixel's red, green and blue components with the corresponding components of the 8 pixels that surround it; i.e., the new colour is based on the colours of 9 pixels.

Modify `blur` to do this. **Do not call `get_color` 9 times**. Instead, define one pair of nested for loops to calculate running sums of the red, green and blue components of the 9 pixels (a total of three sums), then use these sums to calculate the averages. In the body of the inner for loop, there will be exactly one call to `get_color`.

Interactively test your blurring filter. In your opinion, does this filter do a better job of blurring than the

one developed in class?

Exercise 5 (Challenge) - Flipping an Image
In filters.py, define a function that is passed an image and transforms it by flipping it through an imaginary vertical line drawn through the center of the image. The function header is:

```
def flip(img):
```

For example, if you had an image of a person looking to the right, the flipped image would show the person looking to the left.

You'll need to use nested loops.Within each row, the colour of a pixel some distance from the left side of the image must be swapped with the colour of the pixel the same distance from the right side of the image. This exercise is presented as a challenge, so I'm not going to provide the formulas that determine the (x, y) coordinates of the pairs of pixels that will have their colours swapped. That's something you have to figure out. Before writing any code, spend some time sketching diagrams and deriving the formulas. Hint: the body the inner loop is short (less than 10 lines of code). I'll present my solution after students all the lab sections have had time to work on this exercise.

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 need your `filters` module during Lab 7, 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 6 link and submit filters.py. After you click the Add submission button, drag the file to the File submissions box. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status for your files is "Draft (not submitted)". You can resubmit the file by clicking the Edit my submission button. After you've finished uploading the file, remember to click the Submit assignment button to change the submission status to "Submitted for grading".

Posted: Wednesday, October 15, 2013