Problem Set 1: Images as Functions

(arrays or matrices of numbers)

Description

This problem set is really just to make sure you can load an image, manipulate the values, produce some output, and submit the code along with the report. Note that autograded problems will be marked with a (*).

It is expected that you have set up your environment properly. All problem sets will require the following libraries: NumPy, SciPy and OpenCV 3.

If you are having problems, look for information on Piazza, as someone may have resolved it, if not, post a question with detailed specifics of your problem. We want to emphasize the goal of this problem set is to get you all set up so we can do more in later weeks.

Please do not use absolute paths in your submission code. All paths should be relative to the submission directory. Any submissions with absolute paths are in danger of receiving a penalty!

Learning Objectives

- Learn to load, display, and save images.
- Study how images can be represented as functions.
- Identify the difference between an RGB and Monochrome / Grayscale images.
- Apply linear algebra concepts to manipulate image pixel values.
- Perform basic statistical operations in arrays.
- Introduce the concept of noise in an image.

Obtaining the Starter Files:

```
Obtain the starter code from canvas under files.
```

Programming Instructions

Your main programming task is to complete the api described in the file **ps1.py**. The driver program **experiment.py** helps to illustrate the intended use and will output the files needed for the writeup.

Write-up Instructions

Create **ps1_report.pdf** - a PDF file that shows all your output for the problem set, including images labeled appropriately (by filename, e.g. ps1-1-a-1.png) so it is clear which section they are for and the small number of written responses necessary to answer some of the questions (as indicated). You are required to use do the following when creating your report:

- Use the template provided.
- PLEASE be sure to add your name and email, exactly as it is on CANVAS as we need to know who did this work. ALSO, please change the term to current term of the class (SPRING 2018, etc.)

We require PDF only and will not accept a word document, latex, of ppt or any other format.

How to submit:

1. To submit your code, in the terminal window run the following command:
   `python submit.py ps01`
2. To submit the report, input images for part 5, and experiment.py, in the terminal window run the following command:
   `python submit.py ps01_report`
3. Submit your report pdf to gradescope.

YOU MUST SUBMIT your report separately, i.e., two submissions for the code and the report, respectively.
Only your last submission before the deadline will be counted for each of the code and the report.

For the code the following lines will appear:

```
GT Login required.
Username : <GT username (same as T-square)>
Password: <GT password>
Save the jwt?[y,N] <either y or N if you want to save your credentials>
```

You should see the autograder feedback in the terminal window. Additionally, you can look at a history of all your submissions at https://bonnie.udacity.com/

Grading

The assignment will be graded out of 100 points. The code portion (autograder) represents 60% of the grade and the report the remaining 40%.

Assignment Questions

1. Input images [20 pts]

a. Pick two interesting images to use. Name them **ps1-1-a-1.png** and **ps1-1-a-2.png.** Place them in the same directory as the submit.py file. They should be color, rectangular in shape (NOT square). The first and second images should be wide and one tall, respectively. You might find some classic vision examples here, or you may use your own. Make sure the image width or height each does not exceed 512 pixels and that it is at least 100 pixels.

   Code: In the file experiment.py, complete the image paths.
   Report:Place your interesting images (wide and tall images) **ps1-1-a-1.png** and **ps1-1-a-2.png** in the writeup.

2. (*) Color planes [15 pts]

a. Swap the green channel and blue channel of image 1
   Code: implement swap_green_blue()
   Report: place your swapped channel image as **ps1-2-a-1.png** in writeup

b. Make a monochrome image (img1_green) created by selecting the green channel of image 1. (Your monochrome image must be a 2D array)

Code: implement extract_green()
Report: place your green monochrome image as **ps1-2-b-1.png** in writeup

c. Make a monochrome image (img1_red) created by selecting the red channel of image 1. (Your monochrome image should be a 2D array)
Code: implement extract_red()
Report: place your red monochrome image as **ps1-2-c-1.png** in writeup

3. (*) Replacement of pixels [5 pts]

Note: For this, use ps1-2-b-1.png from 2-b as your monochrome image

a. Insert the center square region of 100x100 pixels of the monochrome version of image 1 into the center of a monochrome version of image 2.
Code: implement copy_paste_middle()
Report: place your new image created as **ps1-3-a-1.png** in writeup

4. (*) Arithmetic and Geometric operations [20 pts]

a. Compute the min, max, mean, and standard deviation of pixel values in the monochrome image.
Code: implement image_stats()
Report: insert the values of the min, max, mean, and stddev in the writeup

b. Subtract the mean from all pixels in the monochrome image, then divide by standard deviation, then multiply by the scaling factor 10 if your image is 0 to 255 or 0.05 if your image ranges from 0.0 to 1.0. Now, add the mean back into the product.
Code: implement center_and_normalize()
Report: The manipulated image as **ps1-4-b-1.png** in writeup

c. Shift img1_green to the left by 2 pixels.
Code: implement shift_image_left()
Report: The shifted image as **ps1-4-c-1.png** in writeup

d. Subtract the shifted version of img1_green from the original img1_green, and save the difference image.
Code: implement difference_image()
Report: The difference image as **ps1-4-d-1.png** in write up (make sure that the values are proper, e.g., do not exceed the limits of the image, when you write the image so that you can see all relative differences)

5. (*) Noise [20 pts]

a. Using Image1, start adding Gaussian noise to the pixels in the green channel. Increase sigma until the noise is visible but doesn't overwhelm the image. The full RGB image should be visible, just with some added noise in the green channel.
Code: implement add_noise(), modifying sigma in experiment.py to create visible noise
Report: The noisy green channel image as **ps1-5-a-1.png** in writeup

b. Apply that same amount of noise to the blue channel, and observe how it affects the image.
Report: The noisy blue channel image as **ps1-5-b-1.png** in writeup

6. Discussion [20 pts]

Report: Answer the questions below in the writeup.

a. Use the image **southafricaflagface.png** and look at all three channels individually as monochrome. Between all color channels, which channel most resembles a grayscale conversion of the original? Why is this? Does it matter if you use other images? (For this problem, you will have to read a bit on how the eye works/cameras to discover which channel is more prevalent and widely used)
b. What does it mean when an image has negative pixel values stored? Why is it important to maintain negative pixel values?
c. In question 5, noise was added to the green channel and also to the blue channel. Which looks better to you? Why? What sigma was used to detect any discernible difference?