CS 4476: Computer Vision, Spring 2022
PS1

Instructor: Judy Hoffman

Due: Thursday, January 27, 11:59 pm ET

Instructions

1. The assignment must be done in Python3. No other programming languages are allowed.
2. Fill your answers in the report PPT provided and submit a pdf version of it named like so FirstName_LastName_PS1.pdf on Gradescope. Please do not modify the layout of the boxes provided in the report. Fit your answers within the space provided.
3. Please enter your code in the designated areas of the template Python files. Please do not add additional functions/imports to the files. Points will be deducted for any changes to code/file names, use of static paths and anything else that needs manual intervention to fix.
4. Please submit your code and output files in a zipped format, using the helper script `zip_submission.py` with your GT username as a command line argument (using `--gt_username`), to Gradescope. See the command below. Please do not create subdirectories within the main directory. The `.zip_dir_list.yml` file contains the required deliverable files, and `zip_submission.py` will fail if all the deliverables are not present in the root directory. Feel free to comment and uncomment them as you complete your solutions.

   ```
   $ python zip_submission.py --gt_username <your_gt_username>
   ```
5. For the implementation questions, make sure your code is bug-free and works out of the box. Please be sure to submit all main and helper functions. Be sure to not include absolute paths. Points will be deducted if your code does not run out of the box.
6. If plots are required, you must include them in your Gradescope report and your code must display them when run. Points will be deducted for not following this protocol.
7. Ensure that you follow the instructions very carefully.

Note: There is no extra credit for this assignment. Students are expected to complete it entirely(100 points total: 54 Report + 46 Code).

Setup

Note that we will be using a new conda environment for this project!

1. Install Miniconda. It doesn't matter whether you use Python 2 or 3 because we will create our own environment that uses 3 anyways.
2. Open the terminal
   (a) On Windows: open the installed **Conda prompt** to run the command.

   (b) On MacOS: open a terminal window to run the command
   (c) On Linux: open a terminal window to run the command

3. Navigate to the folder where you have the project

4. Create the conda environment for this project

   (a) On Windows: `conda env create -f proj1_env_win.yml`
   (b) On MacOS: `conda env create -f proj1_env_mac.yml`
   (c) On Linux: `conda env create -f proj1_env_linux.yml`

5. Activate the newly created environment, use the command `conda activate cv_proj1`

Introduction

Read through the provided Python, NumPy and Matplotlib introduction code and comments:
http://cs231n.github.io/python-numpy-tutorial/ or
https://filebox.ece.vt.edu/~F15ECE5554ECE4984/resources/numpy.pdf. Open an interactive session in Python and test the commands by typing them at the prompt. (Skip this step if you are already familiar with Python and NumPy.)

After reading the required documentation in the above section, to test your knowledge ensure that you know the outputs of the following commands without actually running them.

```
    > import numpy as np
(a) > x = np.random.permutation(1000)
(b) > a = np.array([[11,22,33],[40,50,60],[77,88,99]])
    > b = a[2,:]
(c) > a = np.array([[11,22,33],[40,50,60],[77,88,99])
    > b = a.reshape(-1)
(d) > f = np.random.randn(5,1)
    > g = f[f>0]
(e) > x = np.zeros(10)+0.5
    > y = 0.5*np.ones(len(x))
    > z = x + y
(f) > a = np.arange(1,100)
    > b = a[::-1]
```

1 Basic Numpy [20 points (5 each)]

Write a few lines of code to do each of the following. Copy and paste your code into the report.

1.1 Use `numpy.random.rand` to return the roll of a six-sided die over N trials.

1.2 Let y be the vector: `y = np.array([11, 22, 33, 44, 55, 66])`. Use the reshape command to form a new matrix `z` that looks like this: `[[11,22],[33,44],[55,66]]`

1.3 Use the `numpy.max` and `numpy.where` functions to set `x` to the maximum value that occurs in `z` (above), and set `r` to the row number (0-indexed) it occurs in and `c` to the column number (0-indexed) it occurs in.

1.4 Let `v` be the vector: `v = np.array([1, 4, 7, 1, 2, 6, 8, 1, 9])`. Set a new variable `x` to be the number of 1's in the vector `v`.

2 Array/Matrix Operations [25 points (5 each)]

Load the 100x100 matrix `inputAPS1Q2.npy` which is the matrix A. Fill the template functions in the script `PS1Q2.py` to load `inputAPS1Q2.npy` and perform each of the following actions on `A`. Submit the file `PS1Q2.py`.

2.1 Plot all the intensities in `A`, sorted in decreasing value. Provide the plot in your report. (Note, in this case we don't care about the 2D structure of `A`, we only want to sort the list of all intensities.) To ensure consistency, you may use the gray colormap option.
hint : Use matplotlib's imshow and ensure to set the *aspect* appropriately

2.2 Display a histogram of `A`'s intensities with 20 bins. Again, we do not care about the 2D structure. Provide the histogram in your report.

2.3 Create and return a new matrix `X` that consists of the bottom left quadrant of `A`. (Use function `prob_2_3` and return `X`.)

2.4 Create and return a new matrix `Y`, which is the same as `A`, but with `A`'s mean intensity value subtracted from each pixel. (Use function `prob_2_4` and return `Y`.)

2.5 Create and return a new matrix `Z` that represents a color image the same size as `A`, but with 3 channels to represent R, G and B values. Set the values to be red (i.e., $R = 1$, $G = 0$, $B = 0$) wherever the intensity in `A` is greater than a threshold $\texttt{t}$ = the average intensity in `A`, and black everywhere else. (Use function `prob_2_5` and return `Z`.)

3 Image Manipulations [36 points (6 each : 4 Code + 2 Report)]

The input color image `inputPS1Q3.jpg` has been provided. Fill the template functions in the script `PS1Q3.py` to perform the following transformations. Avoid using loops. Submit the file `PS1Q3.py`. Provide all the resultant images in your report.

Note: In every part, the image that is returned from your function must have integer values in the range [0, 255] i.e uint8 format.

3.1 Load the input color image and swap its red and green color channels. Return the output image.
(Use function `prob_3_1` and return `swapImg`.)

3.2 Convert the input color image to a grayscale image. Return the grayscale image.
(Use function `prob_3_2` and return `grayImg`.)

Perform each of the below transformations on the grayscale image produced in part `2` above. When plotting them, make sure you use the gray colormap option.

3.3 Convert the grayscale image to its negative image, in which the lightest values appear dark and vice versa. Return the negative image. (Use function `prob_3_3` and return `negativeImg`.)

3.4 Map the grayscale image to its mirror image (flipping it left to right). Return the mirror image. (Use function `prob_3_4` and return `mirrorImg`.)

3.5 Average the grayscale image with its mirror image (typecast the image to `float` or `double` from `uint8` to avoid overflow when adding). Return the averaged image. (Use function `prob_3_5` and return `avgImg`.)

3.6 Create a matrix `noise` whose size is same as the grayscale image, containing random numbers in the range [0, 255]. Add `noise` to the grayscale image, then clip the resulting image to have a maximum value of 255. Return the clipped image and the noise matrix. (Use function `prob_3_6` and return `noisyImg, noise`)

Tips: Do the necessary typecasting (`uint8` and `double`) when working with or displaying the images. If you can't find some functions in numpy (such as rgb2gray), you can write your own function. For example:

```
def rgb2gray(rgb):
  return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])
```

4 Understanding Color [19 points]

4.1 The same color may look different under different lighting conditions. Images `indoor.png` and `outdoor.png` are two photos of a same Rubik's cube under different illuminances. Load the images and plot their R, G, B channels separately as grayscale images using matplotlib's `imshow()` (use gray colormap). Then convert them into LAB color space using `cv2.cvtColor()` or `skimage.color` and plot the three channels again. Include the plots in your report. (Use function `prob_4_1`) [points - 7 Report]

4.2 In the section below, we have explained the process behind translating the cubical colorspace of RGB to the cylinder of hue, saturation, and value. Read through the explanation and fill in the template function in `PS1Q4.py` to load `inputPS1Q4.jpg`, convert the image from RGB to HSV, and return the final HSV image. Submit the final image to the report. You may use for loops for this question. You are not allowed to use library function to do the conversion. [points - 7 Code + 5 Report]

Note: To ensure consistency, do the necessary typecasting (`double`) and transform the image to values between `[0,1]` before performing the below operations.

So far we've been focusing on RGB and grayscale images. But there are other colorspaces out there too we may want to play around with. Like Hue, Saturation, and Value (HSV).

Hue can be thought of as the base color of a pixel. Saturation is the intensity of the color compared to white (the least saturated color). The Value is the perception of brightness of a pixel compared to black. You can try out this demo to get a better feel for the differences between these two colorspaces.

Now, to be sure, there are lots of issues with this colorspace. But it's still fun to play around with and relatively easy to implement. The easiest component to calculate is the Value, it's just the largest of the 3 RGB components:

$$V = max(R, G, B)$$

Next we can calculate Saturation. This is a measure of how much color is in the pixel compared to neutral white/gray. Neutral colors have the same amount of each three color components, so to calculate saturation we see how far the color is from being even across each component. First we find the minimum value

$$m = min(R, G, B)$$

Then we see how far apart the min and max are:

$$C = V - m$$

and the Saturation will be the ratio between the difference and how large the max is:

$$S = C/V$$

Except if R, G, and B are all 0. Because then V would be 0 and we don't want to divide by that, so just set the saturation 0 if that's the case.

Finally, to calculate Hue we want to calculate how far around the color hexagon our target color is.

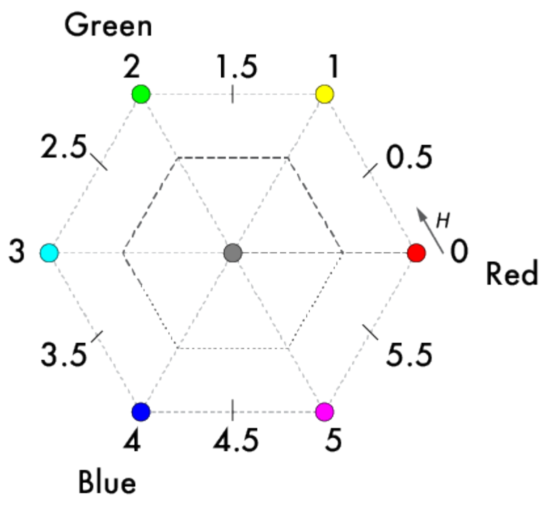

Figure 1: Color Hexagon

We start counting at Red. Each step to a point on the hexagon counts as 1 unit distance. The distance between points is given by the relative ratios of the secondary colors. We can use the following formula from Wikipedia:

$$H' = \begin{cases} undefined & \text{C} = 0 \\ \frac{G-B}{C} & \text{if V} = \text{R} \\ \frac{B-R}{C} + 2 & \text{if V} = \text{G} \\ \frac{R-G}{C} + 4 & \text{if V} = \text{B} \end{cases}$$

$$H = \begin{cases} \frac{H'}{6} + 1 & \text{if H' } < 0 \\ \frac{H'}{6} & \text{otherwise} \end{cases}$$

There is no "correct" Hue if C = 0 because all of the channels are equal so the color is a shade of gray, right in the center of the cylinder. However, for now let's just set H = 0 if C = 0 because then your implementation will match ours. (Use function `prob_4_2` and return `HSV`.)

Deliverable Checklist

1. 1.1-1.4 (report) code snippets for each question.
2. 2.1-2.5 (code/files) `PS1Q2.py`, (report) corresponding plot for 2.1 & 2.2.
3. 3.1-3.6 (code/files) `PS1Q3.py`, (report) 6 images.
4. 4.1 (report) display the 2 plots (RGB, LAB)
5. 4.2 (code/files) `PS1Q4.py`, (report) `Resultant HSV image`

Submission Instructions

- Submit the code as zip on Gradescope at **PS1 - Code**
- Submit the report as PDF on Gradescope at **PS1 - Report**

There is no submission to be done on Canvas.

This assignment is adapted from the following 3 sources:

PS0 assignment of Kristen Grauman's CS 376: Computer Vision at UT Austin

HW1 assignment of David Fouhey's EECS 442: Computer Vision at University of Michigan.

HW0 assignment of Joseph Redmon's CSE 455: Computer Vision at University of Washington.