Image Processing – 67829 – Exercise 1

Image Representations, Intensity Transformations, and Quantization

Due date: 13/11/2019

1 Overview

The main purpose of this exercise is to get you acquainted with NumPy and some image processing facilities. This exercise covers:

- Loading grayscale and RGB image representations.
- Displaying figures and images.
- Transforming RGB color images back and forth from the YIQ color space.
- Performing intensity transformations: histogram equalization.
- Performing optimal quantization

2 Background

Before you start working on the exercise it is recommended for those of you using Numpy for the first time to go over a manual or read the notes from Moodle. **Relevant reading material:** You can read about power law transformations and histogram equalization in Gonzalez and Woods book.

3 The Exercise

3.1 Reading an image into a given representation

Write a function which reads an image file and converts it into a given representation. The function should have the following interface:

`read_image(filename, representation)`

having the following input arguments:
`filename` - the filename of an image on disk (could be grayscale or RGB).
`representation` - representation code, either 1 or 2 defining whether the output should be a grayscale image (1) or an RGB image (2). If the input image is grayscale, we won't call it with `representation = 2`.

This function returns an image, make sure the output image is represented by a matrix of type `np.float64` with intensities (either grayscale or RGB channel intensities) normalized to the range $[0, 1]$. You will find the function `rgb2gray` from the module `skimage.color` useful, as well as `imread` from `scipy.misc`. We won't ask you to convert a grayscale image to RGB.

3.2 Displaying an image

Write a function to utilize `read_image` to display an image in a given representation. The function should have the following interface:

`imdisplay(filename, representation)`

where `filename` and `representation` are the same as those defined in `read_image`'s interface. The function should open a new figure and display the loaded image in the converted representation.

3.3 Transforming an RGB image to YIQ color space

Write two functions that transform an RGB image into the YIQ color space (mentioned in the lecture) and vice versa. Given the red (R), green (G), and blue (B) pixel components of an RGB color image, the corresponding luminance (Y), and the chromaticity components (I and Q) in the YIQ color space are linearly related as follows:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

The two functions should have the following interfaces:

`rgb2yiq(imRGB)`
`yiq2rgb(imYIQ)`

where the inputs of the functions are $height \times width \times 3$ `np.float64` matrices. In the RGB case, imRGB is in the $[0, 1]$ range, the red channel is encoded in `imRGB[:,:,0]`, the green in `imRGB[:,:,1]`, and the blue in `imRGB[:,:,2]`. Similarly for YIQ, `imYIQ[:,:,0]` encodes the luminance channel Y, `imYIQ[:,:,1]` encodes I, and `imYIQ[:,:,2]` encodes Q. Unlike the RGB case, while the Y channel is in the [0,1] range, the I and Q channels are in the $[-1, 1]$ range (though they do not span it entirely). Each of the functions returns an image having the same dimensions as the input.

3.4 Histogram equalization

Write a function that performs histogram equalization of a given grayscale or RGB image. The function should have the following interface:

```
histogram_equalize(im_orig)
```

where:
`im_orig` - is the input grayscale or RGB float64 image with values in $[0, 1]$.
The function returns a list [`im_eq, hist_orig, hist_eq`] where
`im_eq` - is the equalized image. grayscale or RGB float64 image with values in $[0, 1]$.
`hist_orig` - is a 256 bin histogram of the original image (array with shape (256,)).
`hist_eq` - is a 256 bin histogram of the equalized image (array with shape (256,)).
If an RGB image is given, the following equalization procedure should only operate on the Y channel of the corresponding YIQ image and then convert back from YIQ to RGB. Moreover, the outputs `hist_orig` and `hist_eq` should be the histogram of the Y channel only. The required intensity transformation is defined such that the gray levels should have an approximately uniform gray-level histogram (i.e. equalized histogram) stretched over the entire $[0, 1]$ gray level range. Make sure you stretch the equlized histogram if it only covers partly that range. You may use the NumPy functions `histogram` and `cumsum` to perform the equalization. Also note that although `im_orig` is of type `np.float64`, you are required to internally perform the equalization using 256 bin histograms. After equalizing histogram of Y and converting the image back to RGB there may be some values outside the range $[0, 1]$.

3.5 Optimal image quantization

Write a function that performs optimal quantization of a given grayscale or RGB image.

The function should have the following interface:

```
quantize (im_orig, n_quant, n_iter)
```

where:
`im_orig` - is the input grayscale or RGB image to be quantized (float64 image with values in $[0, 1]$).
`n_quant` - is the number of intensities your output `im_quant` image should have.
`n_iter` - is the maximum number of iterations of the optimization procedure (may converge earlier.)
And the output is a list [`im_quant, error`] where
`im_quant` - is the quantized output image.
`error` - is an array with shape (`n_iter`,) (or less) of the total intensities error for each iteration of the quantization procedure.

If an RGB image is given, the quantization procedure should only operate on the Y channel of the corresponding YIQ image and then convert back from YIQ to RGB. Each iteration of the quantization process is composed of the following two steps:

- Computing `z` - the borders which divide the histograms into segments. `z` is an array with shape (`n_quant`+1,). The first and last elements are 0 and 255 respectively.
- Computing `q` - the values to which each of the segments' intensities will map. `q` is also a one dimensional array, containing `n_quant` elements.

More comments:

- You should perform the two steps above `n_iter` times unless the process converges.
- You should find `z` and `q` by minimizing the total intensities error. The closed form expressions for `z` and `q` can be found in the lecture notes.
- The quantization procedure needs an initial segment division of [0..255] to segments, `z`. If a division will have a gray level segment with no pixels, the procedure will crash **(Q1: Why?)**. In order to overcome this problem, you are required to set the initial division such that each segment will contain approximately the same number of pixels.
- The exact calculation of the output `error` is given to you in the lecture notes.
- Convergence is defined as a situation where the values of `z` have not been changed during the last iteration. You should stop iterating in case of convergence.
- In order to plot the error as a function of the iteration number use the command `plt.plot(error)`, where the module was imported using `import matplotlib.pyplot as plt`. As a sanity check make sure the error graph is monotonically descending.

- For simplicity, you may assume that we won't ask you to perform quantization of an image to more than a third of the total number of gray-shades.
- In general we will not check your quantization on extreme cases. For example, you may assume there is no bin in the histogram that contains more pixels than the initial number of pixels per segment.
- **Do not** clip the values of the quantized image, as clipping will introduce additional intensities. It's OK to return an image with some of the values greater than 1 or smaller than 0.

3.5.1 Bonus - 10 points

If you find this exercise too easy, you are welcome to try and perform quantization for full color images. You will have to find the methods yourself. If you choose to do so, put your code in a function called `quantize_rgb(im_orig, n_quant)` that returns just the quantized image `im_quant`. (Since we do not specify the algorithm you should use, then there is no point in returning an "error" array like we instructed you for the regular `quantize` function.)

4 Specific Guidelines

Pay attention to the following guidelines:

1. Test your program on the provided example images and on other images you may find or create. Try not to use very big images (more than 1000X1000 pixels) for fast computations.
2. Document all interfaces, code blocks, and crucial parts of the program (but not every single line).
3. You must write vectorial code where possible (try to avoid using for-loops). Sections $3.1, 3.2, 3.4$ should be done without loops at all. In section 3.3 you may loop over the 3 channels, but not over the pixels of the image. In section 3.5 you will need to use a loop for performing the iterations. You can also loop over `n_quant`, but you should not loop over 256 or over the pixels of the image.
4. In sections $3.3, 3.4, 3.5$ the output images' type should be float64 in the range $[0, 1]$. You can assume that the input images are float64 in the range $[0, 1]$. Please notice that by saying range $[0, 1]$ we mean that the values are between 0 and 1, there is no demand that the smallest number is 0 and the largest one is 1. The above is strictly for RGB and grayscale images, and obviously not for YIQ images.
5. You can query the type and shape of an array using its attributes `dtype` and `shape`, respectively.

6. Perform reasonable checks to the program input arguments. Your program may only crash in case of serious abuse by the user.

7. Write reusable code. This will make solving this and the following exercises much simpler.

8. You may add new functions to your implementation.

9. Only import packages mentioned in the Tirgul slides, or from this exercise description, except for the bonus where you can import other function you like.

10. If you use magic functions (starting with %) don't forget to remove them before submission.

5 Submission

Submission instructions are given in the "Submissions Guidelines" document published on the course web page. Please read and follow them carefully. Any update to those guidelines will be posted in the news forum, so be sure you follow the forum.

1. You should include a README.md (do not name it README.txt or README) file formatted according to the course submission guidelines. In particular, it should contain your login, and list of submitted files, in separate lines.

2. Answer question **Q1** in a simple text file named 'answer_q1.txt' and add it to your submission.

3. Put all your code in one file, it should be called 'sol1.py'. This file should contain only functions and global constant variables, we will import this file, so don't use any top level code or global variables. If you wish to make your code executable for your own tests, then you should write a separate file and import your 'sol1.py' file as we will in our testers.

4. Do not zip your files! The submission to Moodle is done entirely by the git submit command.

Good luck and enjoy!