Image Processing

Exercise 1: Image Representations and Point Operations

Due date: 18/04/2020

1 Overview

The main purpose of this exercise is to get you acquainted with Python's basic syntax and some of its 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 Before you start!

There is a file added called **ex1_main.py**, this is how I'll run your program (not exaclty, but if this runs smoothly, my autoTester will work fine), make sure it works without any changes. That mean, that **all** your code should be in the file **ex1_utils.py**, that is imported to **ex1_main.py**, except the **gammaDisplay** which should be in **gamma.py** file.

Add a the following function to **ex1_utils.py** so I'll be able to grade the right person.

```
def myID() -> np.int:
    """
```

```
    Return my ID (not the friend's ID I copied from)
    :return: int
    """
    return 123456789
```

3 Background

Before you start working on the exercise it is recommended for those of you using Python for the first time to go over the first week's exercise class notes. Those already familiar with Python may still find the "OpenCV/Numpy" part of the documentation useful. **Relevant reading material:** You can read about power law transformations and histogram equalization in Gonzalez and Woods book.

4 The Exercise

4.1 Reading an image into a given representation

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

```
def imReadAndConvert(filename:str, representation:int)->np.ndarray:
    """
    Reads an image, and returns and returns in converted as requested
    :param filename: The path to the image
    :param representation: grayscale(1) or RGB(2)
    :return: The image np array
    """
```

Make sure that the output image is represented by a matrix of class `np.float` with intensities (either grayscale or RGB channel intensities) normalized to the range $[0, 1]$. You will find the function `cv2.cvtColor` useful, as well as `dtype` which will return the type of the array. We won't ask you to convert a grayscale image to RGB.

4.2 Displaying an image

Write a function that utilizes `imReadAndConvert` to display a given image file in a given representation. The function should have the following interface:

```
imDisplay(filename:str, representation:int)->None:
    """
    Reads an image as RGB or GRAY_SCALE and displays it
    :param filename: The path to the image
    :param representation: grayscale(1) or RGB(2)
    :return: None
    """
```

The function should open a new figure window and display the loaded image in the converted representation. Also use the `plt.imshow` function to display the image. Don't forget that matplotlib assumes RGB!

4.3 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:

```
transformRGB2YIQ(imRGB:np.ndarray)->np.ndarray:
    """
    Converts an RGB image to YIQ color space
    :param imgRGB: An Image in RGB
    :return: A YIQ in image color space
    """
```

```
transformYIQ2RGB(imYIQ:np.ndarray)->np.ndarray:
    """
    Converts an YIQ image to RGB color space
    :param imgYIQ: An Image in YIQ
    :return: A RGB in image color space
    """
```

where both `imRGB` and `imYIQ` are $height \times width \times 3$ `double` matrices. In the RGB case the red channel is encoded in `imRGB[:,:,1]`, the green in `imRGB[:,:,2]`, and the blue in `imRGB[:,:,3]`. Similarly for YIQ, `imYIQ[:,:,1]` encodes the luminance channel Y, `imYIQ[:,:,2]` encodes I, and `imYIQ[:,:,3]` encodes Q.

4.4 Histogram equalization

Write a function that performs histogram equalization of a given grayscale or RGB image. The function should also display the input and the equalized output image. The function should have the following interface:

```
histogramEqualize(imOrig:np.ndarray)->(np.ndarray,np.ndarray,np.ndarray):
    """
    Equalizes the histogram of an image
    :param imgOrig: Original Histogram
    :return: (imgEq,histOrg,histEQ)
    """
```

where:
`imOrig` - is the input grayscale or RGB image to be equalized having values in the range $[0, 1]$.

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. The required intensity transformation is defined 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. You may use the numpy functions `np.histogram`, `np.cumsum` etc. to perform the equalization **but not** `cv2.equalizeHist()`. Also note that although `imOrig` is of type `float`, you are required to internally perform the equalization using 256

bin histograms. On completion, `histogramEqualize` should output the equalized image `imEq`, a 255 bin histogram of the original image `histOrig`, and a 255 bin histogram of the equalized image `histEq`.

4.5 Optimal image quantization

Write a function that performs optimal quantization of a given grayscale or RGB image. The function should also display:

- The input image
- The quantize output image
- An error plot representing the MSE for each iteration in the quantization procedure.

The function should have the following interface:

```
quantizeImage(imOrig:np.ndarray, nQuant:int, nIter:int)->(List[np.ndarray],List[float]):
    """
    Quantized an image in to **nQuant** colors
    :param imOrig: The original image (RGB or Gray scale)
    :param nQuant: Number of colors to quantize the image to
    :param nIter: Number of optimization loops
    :return: (List[qImage_i],List[error_i])
    """
```

If an RGB image is given, the following quantization procedure should only operate on the Y channel of the corresponding YIQ image and then convert back from YIQ to RGB.

Each iteration in the quantization process contains two steps:

- Finding `z` - the borders which divide the histograms into segments. `z` is a vector containing `nQuant+1` elements. The first and last elements are 0 and 255 respectively.
- Finding `q` - the values that each of the segments' intensities will map to. `q` is also a vector, however, containing `nQuant` elements.

More comments:

- You should perform the two steps above `nIter` times.

- You should find `z` and `q` by minimizing the total intensities error. The close 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 grey level segment with no pixels, procedure will crash **(Why?)**. In order to overcome this problem, we suggest to set the initial division such that each segment will contain approximately the same number of pixels.
- The output `error` is a vector with `nIter` elements (or less in case of converges). Each element is the total intensities error in a current iteration. The exact error calculation is given to you in the lecture notes.
- Please notice that your function should plot the error as a function of the iteration number using the command `plt.plot(error)`. As a sanity check make sure that the error graph is monotonically descending.

* Notice that after quantization (with the YIQ transform), the RGB image will have more than the desired colors, try to think why that happens. To get better results you can use the KMeans algorithm, it's a bit more complicated, so do it only if you have time left, but the results and the knowledge are worthwhile (contact me for details).

4.6 Gamma Correction

Write a function that performs gamma correction on an image with a given γ.

For this task, you'll be using the OpenCV funtions `createTrackbar` to create the slider and display it, since it's OpenCV's functions, the image will have to be represented as BGR.

```
def gammaDisplay(self, img_path:str, rep:int)->None:
    """
    GUI for gamma correction
    :param img_path: Path to the image
    :param rep: grayscale(1) or RGB(2)
    :return: None
    """
```

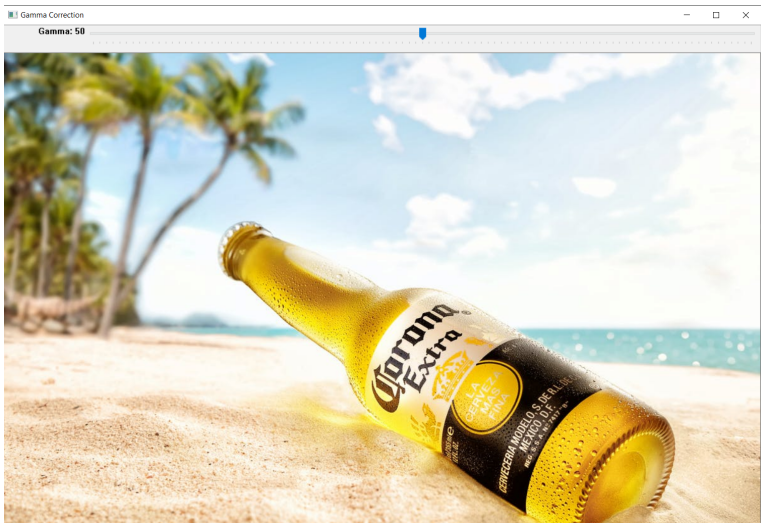

Figure 1: GammaGui screenshot

The slider's values should be from 0.01 to 2 with resolution 0.01 but since the OpenCV trackbar has only integer values, think how to convert it to float when applying the gamma correction.

Useful Links:

- OpenCV trackbar example
- Gamma Correction Wikipedia

5 Specific Guidelines

Pay attention to the following guidelines:

1. Test your programs on the provided example images and on other images you may find or create.
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).
4. Perform reasonable checks to the program input arguments. You can use a try-catch block if you prefer. Your program may only crash in case of serious abuse by the user.
5. Write reusable code. This will make solving this and the following exercises much simpler.
6. Mention in your README file the Python version and platform on which you have tested your program on. List the files you are submitting and a short description of each. Also list in the

README file the functions you've written with a short description of each. You should also answer the question from section 4.5.

6 Submission

Submission instructions will be published later in the course web site. Please read and follow them carefully. In this exercise you are also required to submit two test images that you have created (or downloaded), named `testImg1.jpg` and `testImg2.jpg` (different from those that we've supplied) and the python script `ex1.py`. Explain in your README file why you designed them in this way and how they were used to examine your implementation.

Good luck and enjoy!