Contents

1 Basic Test Results 2

2 README 4

3 answer q1.txt 5

4 sol1.py 6

1 Basic Test Results

```
Archive:  /tmp/bodek.KDn6AM/impr/ex1/liavst2/presubmission/submission
  inflating: current/sol1.py
  inflating: current/README
  inflating: current/answer_q1.txt
ex1 presubmission script

  Disclaimer
  ----------
  The purpose of this script is to make sure that your code is compliant
  with the exercise API and some of the requirements
  The script does not test the quality of your results.
  Don't assume that passing this script will guarantee that you will get
  a high grade in the exercise

login:  liavst2

submitted files:


======================

==== README for ex1 ===

======================



=========================

=== List of submitted files ====

=========================



- README - this file.

- answer_q1.txt

- sol1.py - implementation of all the functions
answer to q1:

Answer for Q1:
-------------------------

The procedure will fail because we will get division by zero when
calculating the q values.
section 3.1
Reading images
section 3.3
Transforming rgb->yiq->rgb
Section 3.4
- Histogram equalization...
Section 3.5
- Image quantization...
all tests Passed.
- Pre-submission script done.

  Please go over the output and verify that there are no failures/warnings.
```

```
60      Remember that this script tested only some basic technical aspects of your implementation
61      It is your responsibility to make sure your results are actually correct and not only
62      technically valid.
```

2 README

```
liavst2

======================
==== README for ex1 ===
======================

==========================
=== List of submitted files ====
==========================

- README - this file.
- answer_q1.txt
- sol1.py - implementation of all the functions
```

3 answer q1.txt

```
Answer for Q1:
-------------------------

The procedure will fail because we will get division by zero when
calculating the q values.
```

4 sol1.py

```
1  ##################################################################
2  # FILE: sol1.py
3  # WRITER: Liav Steinberg
4  # EXERCISE : Image Processing ex1
5  ##################################################################
6
7  import numpy as np
8  from scipy.misc import imread
9  from matplotlib import pyplot as plt
10 from skimage.color import rgb2gray
11
12
13 def read_image(filename, representation):
14     ###########################################################
15     # reads an image with the given representation
16     ###########################################################
17     image = imread(filename).astype(np.float32) / 255
18     return image if representation == 2 else rgb2gray(image)
19
20
21 def imdisplay(filename, representation):
22     ###########################################################
23     # displays an image with the given representation
24     ###########################################################
25     image = read_image(filename, representation)
26     if representation == 1:
27         plt.imshow(image, cmap=plt.cm.gray)
28     elif representation == 2:
29         plt.imshow(image)
30     plt.show()
31
32
33 def rgb2yiq(imRGB):
34     ###########################################################
35     # converts from RGB image to YIQ using the conversion
36     # matrix showed in class.
37     ###########################################################
38     R, G, B = imRGB[:, :, 0], imRGB[:, :, 1], imRGB[:, :, 2]
39     Y = 0.299*R + 0.587*G + 0.114*B
40     I = 0.596*R - 0.275*G - 0.321*B
41     Q = 0.212*R - 0.523*G + 0.311*B
42     imRGB[:, :, 0], imRGB[:, :, 1], imRGB[:, :, 2] = Y, I, Q
43     return imRGB
44
45
46 def yiq2rgb(imYIQ):
47     ###########################################################
48     # converts from YIQ image to RGB using the inverted
49     # conversion matrix showed in class.
50     ###########################################################
51     Y, I, Q = imYIQ[:, :, 0], imYIQ[:, :, 1], imYIQ[:, :, 2]
52     R = Y + 0.956*I + 0.621*Q
53     G = Y - 0.272*I - 0.647*Q
54     B = Y - 1.106*I + 1.703*Q
55     imYIQ[:, :, 0], imYIQ[:, :, 1], imYIQ[:, :, 2] = R, G, B
56     return imYIQ
57
58
59 def histogram_equalize(im_orig):
```

```
    ######################################################
    # Performs histogram equalization on a given image,
    # according to the lecture algorithm.
    ######################################################
    isRGB, imYIQ = (len(im_orig.shape) == 3), None
    if isRGB:
        imYIQ = rgb2yiq(im_orig)
        im_orig = imYIQ[:, :, 0]
    #get the histogram of the image
    hist_orig, bins = np.histogram(im_orig.flatten(), 256)
    #compute the cumulative histogram
    cdf = np.cumsum(hist_orig)
    #normalize the cumulative histogram
    cdf = np.round(255 * (cdf - cdf[cdf>0].min()) / (cdf.max() - cdf[cdf>0].min()))
    #use linear interpolation of cdf to find new pixel values
    im_eq = np.interp(im_orig, np.linspace(0, 1, 256), cdf)
    #histogram the new image
    hist_eq, bins = np.histogram(im_eq.flatten(), 256)
    #if we got RGB, return it back to RGB
    if isRGB:
        imYIQ[:, :, 0] = im_eq / 255
        #using clip to zero the negative results of the transformation
        im_eq = np.clip(yiq2rgb(imYIQ), 0, 1)
    return im_eq, hist_orig, hist_eq


def quantize(im_orig, n_quant, n_iter):
    ######################################################
    # Performs image quantization on a given image,
    # n_iter iterations, according to the lecture algorithm.
    ######################################################

    def initialize(hist, nQuant):
        ######################################################
        # Initializes the initial division of the histogram
        ######################################################
        cdf = np.cumsum(hist)
        total_pixels = cdf.max()
        pixels_per_seg = round(total_pixels / nQuant)
        ZZ = [0]
        for ii in range(1, nQuant):
            ZZ.append(np.argmin(cdf < pixels_per_seg * ii))
        ZZ.append(256)
        return ZZ

    def calculate_Q(ZZ, hist, nQuant):
        ######################################################
        # Calculates the q value in each iteration
        ######################################################
        return [np.round(np.average(np.arange(int(ZZ[m]), int(ZZ[m+1])),
                                    weights=np.take(hist, np.arange(int(ZZ[m]), int(ZZ[m+1]))))
        )) for m in range(nQuant)]

    def calculate_error(ZZ, QQ, hist):
        ######################################################
        # Calculates the error value in each iteration
        ######################################################
        return np.dot(np.square(
                np.array(QQ)
                [np.digitize(np.arange(0, 256), ZZ) - 1] - np.arange(0, 256)), hist)

    ###########################################################################

    isRGB, imYIQ = (len(im_orig.shape) == 3), None
    #if its color, first change to yiq
    if isRGB:
        imYIQ = rgb2yiq(im_orig)
        im_orig = imYIQ[:, :, 0] * 255
```

```
    else:
        im_orig *= 255

    hist_orig, bins = np.histogram(im_orig, 256)
    error, Q, Z = [], [], initialize(hist_orig, n_quant)

    ###  quantization procedure ###

    for i in range(n_iter):
        prev_iteration_Z = Z
        prev_iteration_Q = Q
        Q = calculate_Q(Z, hist_orig, n_quant)
        Z[0] = 0
        for k in range(1, len(Z)-1):
            Z[k] = np.average([Q[k-1], Q[k]])
        error.append(calculate_error(Z, Q, hist_orig))

        #if already converged, do not proceed to the next iteration
        if prev_iteration_Z == Z and prev_iteration_Q == Q:
            break

    #creating lookup table
    LUT = np.zeros(256)
    for i in range(n_quant):
        indexes = np.array(np.arange(int(Z[i]), int(Z[i+1])))
        LUT[indexes] = Q[i]
    #taking the values along the axis
    im_quant = np.take(LUT.astype(np.int64), np.clip(im_orig, 0, 255).astype(np.int64))
    #changing back to color, if needed
    if isRGB:
        imYIQ[:, :, 0] = im_quant.astype(np.float32) / 255
        # without clip, as said in the forum!
        im_quant = yiq2rgb(imYIQ)
    return im_quant, error


def quantize_rgb(im_orig, n_quant, n_iter):
    ############################################################
    #--------------------- bonus --------------------------#
    ############################################################
    # Performs image quantization for full color images
    ############################################################
    R, G, B = im_orig[:, :, 0], im_orig[:, :, 1], im_orig[:, :, 2]
    #send each element to quantization
    quant_R, err_R = quantize(R, n_quant, n_iter)
    quant_G, err_G = quantize(G, n_quant, n_iter)
    quant_B, err_B = quantize(B, n_quant, n_iter)
    im_orig[:, :, 0], im_orig[:, :, 1], im_orig[:, :, 2] = quant_R, quant_G, quant_B
    im_orig *= 255
    ## setting the error vector properly ##
    # first, we adjust the three error vector to be in same length
    # by padding each vector with its last value
    max_len = np.max([len(err_R), len(err_G), len(err_B)])
    err_R.append(err_R[-1] for i in range(max_len - len(err_R)))
    err_G.append(err_G[-1] for i in range(max_len - len(err_G)))
    err_B.append(err_B[-1] for i in range(max_len - len(err_B)))
    # then, average every index in all of them
    err = [np.average([err_R[i], err_G[i], err_B[i]]) for i in range(max_len)]
    return im_orig, err
```