Contents

1 Basic Test Results 2

2 README 3

3 answer q1.txt 4

4 sol1.py 5

1 Basic Test Results

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

submitted files:


==== README for ex1 ===



List of submitted files:



README - this file

sol1.py - python3 code
answer to q1:
Answer to question Q1 in section 3.5:

The reason it will fail is because by finding the next q_i, we devide by zero since the sum over the pixel
in some segment is zero
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.
  Remember that this script tested only some basic technical aspects of your implementation
  It is your responsibility to make sure your results are actually correct and not only
  technically valid.
```

2 README

```
ITAMAKATZ

==== README for ex1 ===

List of submitted files:

README - this file
sol1.py - python3 code
```

3 answer q1.txt

```
1  Answer to question Q1 in section 3.5:
2
3  The reason it will fail is because by finding the next q_i, we devide by zero since the sum over the pixel
4  in some segment is zero
```

4 sol1.py

```
import numpy as np
from scipy.misc import imread as imread
from skimage.color import rgb2gray
import matplotlib.pyplot as plt

def read_image(filename, representation):

    im = imread(filename)
    # check if it is a B&W image
    if(representation == 1):
        im = rgb2gray(im)
    # convert to float and normalize
    return im.astype(np.float32) / 255

def imdisplay(filename, representation):
    # check if it is a B&W or color image
    if(representation == 1):
        plt.imshow(read_image(filename, representation), plt.cm.gray)
    else:
        plt.imshow(read_image(filename, representation))
    plt.show()

def rgb2yiq(imRGB):
    # define the transformation matrix
    trans_mat = np.array([[0.299, 0.587, 0.114], [0.596, -0.275, -0.321], [0.212, -0.523, 0.311]])
    # make a deep copy of original image
    imYIQ = imRGB.copy()
    # calc transformation
    for i in range(0, 3):
        imYIQ[:, :, i] = trans_mat[i, 0] * imRGB[:, :, 0] + trans_mat[i, 1] * imRGB[:, :, 1] + \
                         trans_mat[i, 2] * imRGB[:, :,2]

    return imYIQ

def yiq2rgb(imYIQ):
    # define the transformation matrix
    trans_mat = np.linalg.inv(np.array([[0.299, 0.587, 0.114], [0.596, -0.275, -0.321], [0.212, -0.523, 0.311]]))
    # make a deep copy of original image
    imRGB = imYIQ.copy()
    # calc transformation
    for i in range(0, 3):
        imRGB[:, :, i] = trans_mat[i, 0] * imYIQ[:, :, 0] + trans_mat[i, 1] * imYIQ[:, :, 1] + \
                         trans_mat[i, 2] * imYIQ[:, :,2]

    return imRGB

def histogram_equalize(im_orig):

    yiq = None

    # check if it is a B&W or color image
    if(im_orig.ndim == 2):
        im = im_orig
    else:
        # transform to the YIQ space
        yiq = rgb2yiq(im_orig)
        im = yiq[:, :, 0]

    # calc the histogram of the image
```

```
    hist_orig, bins = np.histogram(im.flatten(), 256)
    # compute the cumulative histogram
    cumulative_histogram = np.cumsum(hist_orig)
    # find first m for which S(m) != 0
    m_val = (cumulative_histogram[cumulative_histogram > 0])[0]
    # apply linear stretching
    cumulative_stretch = np.round(255 * (cumulative_histogram - m_val) / (cumulative_histogram[-1] - m_val))

    if(im_orig.ndim == 2):
        # apply the look up table to the image
        im_eq = np.interp(im.flatten(), bins[:-1], cumulative_stretch).reshape(im.shape)
        #  calc the histogram of the enhanced image
        hist_eq, bins2 = np.histogram(im_eq, 256)
    else:
        # apply the look up table to the image
        yiq[:, :, 0] = np.interp(im.flatten(), np.linspace(0, 1, 256, True), cumulative_stretch).reshape(im.shape) / 255
        #  ensure the values after the transformation are in the [0,1] range by "clipping"
        im_eq = np.clip(yiq2rgb(yiq), 0, 1)
        #  calc the histogram after the clipping
        # yiq = rgb2yiq(yiq2rgb(im_eq))
        hist_eq, bins2 = np.histogram(yiq[:, :, 0].flatten(), 256)

    return [im_eq, hist_orig, hist_eq]

def quantize (im_orig, n_quant, n_iter):
    yiq = None

    # check if it is a B&W or color image
    if(im_orig.ndim == 2):
        im = im_orig * 255
    else:
        # transform to the YIQ space
        yiq = rgb2yiq(im_orig)
        im = yiq[:, :, 0] * 255

    # calc the histogram of the image
    hist, bins = np.histogram(im.flatten(), 255)
    # compute the cumulative histogram
    cumulative_histogram = np.cumsum(hist)

    # initialize the arrays
    z_arr = np.zeros(n_quant + 1,)
    q_arr = np.zeros(n_quant,)
    err = np.zeros(n_iter,)

    # calc z_arr: divide the z indexes so each segment has am equal amount of pixels
    init_z_step = np.floor(cumulative_histogram[-1] / n_quant)
    index_array = np.arange(len(cumulative_histogram))
    for i in range(1, n_quant):
        z_arr[i] = index_array[cumulative_histogram > init_z_step * i][0]
    z_arr[-1] = 255

    for i in range(n_iter):
        # save current z,q arrays to check in case we converged
        prev_z_arr = np.copy(z_arr)
        prev_q_arr = np.copy(q_arr)

        # calc the new q_arr
        for k in range(n_quant):
            numerator, denominator = 0, 0
            for z in range(int(z_arr[k]), int(z_arr[k + 1])):
                numerator += z * hist[z]
                denominator += hist[z]
            q_arr[k] = np.round(numerator / denominator)

        # calc the new z_arr
        z_arr[0] = 0
        for k in range(1, n_quant):
```

```
128             z_arr[k] = np.average([q_arr[k - 1], q_arr[k]])
129
130         indexes = np.digitize(np.linspace(0, 254, 255), z_arr) - 1
131         err[i] = np.dot(np.square(q_arr[indexes] - np.arange(0, 255)), hist)
132
133         # check in case we converged
134         if np.array_equal(prev_z_arr, z_arr) and np.array_equal(prev_q_arr, q_arr):
135             # save only the relevant (none zero) values of err
136             err = err[:i]
137             break
138
139     # calc the lookup table
140     lookup_table = np.zeros(256,)
141     for i in range(n_quant):
142         lookup_table[np.arange(int(z_arr[i]), int(z_arr[i + 1]))] = q_arr[i]
143
144     # check if it is a B&W or color image
145     if(im_orig.ndim == 2):
146         # take only relevant values from the lookup table
147         im_quant = np.take(lookup_table, (im * 255).astype(np.int32))
148
149     else:
150         # take only relevant values from the lookup table
151         yiq[:, :, 0] = np.take(lookup_table, im.astype(np.int32)) / 255
152         # convert back to grb space
153         im_quant = yiq2rgb(yiq)
154
155     return im_quant, err
156
157 def quantize_rgb(im_orig, n_quant, n_iter):
158
159     #  init vecs and mat
160     im_quantize = np.copy(im_orig)
161     err = np.zeros(n_iter,)
162
163     # send all 3 color dimensions to quantize()
164     for i in range(3):
165         # normalize for B&W manipulation in quantize()
166         send_im = im_orig[:, :, i] / 255
167         im_quantize[:, :, i], new_err = quantize(send_im, n_quant, n_iter)
168         # save all errors together
169         err += new_err
170
171     # find averaged error while leaving the empty values
172     err = err[err > 0] / 3
173     # de-normelize (due to the iter)
174     im_quantize *= 255
175
176     return im_quantize, err
```