Contents

1 Basic Test Results

```
ex1/
ex1/README.md
ex1/answer_q1.txt
ex1/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

=== Check Submission ===

README file:

tal.porezky
sol1.py
answer_q1.txt



Answer for Q1:
in case of a segment which contains no pixels, the process will fail because
segment with no pixels would mean p(z) = 0 for that segment, and therefore
lead the algorithm to crash since we will divide by 0 when calculating q_i.

In order for the algorithm not to crash we can divide the segments so that
would contain same number of pixels in each segment, instead of dividing them
in z equal parts of 255.
=== Load Student Library ===

Loading...

=== Section 3.1 ===

Reading images...

=== Section 3.3 ===

Transforming rgb->yiq->rgb

=== Section 3.4 ===

- Histogram equalization...

=== Section 3.5 ===

- Image quantization...


=== Presubmission Completed Successfully ===




    Please go over the output and verify that there were 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 ex1/README.md

```
tal.porezky
sol1.py
answer_q1.txt
```

3 ex1/answer q1.txt

```
1  in case of a segment which contains no pixels, the process will fail because
2  segment with no pixels would mean p(z) = 0 for that segment, and therefore
3  lead the algorithm to crash since we will divide by 0 when calculating q_i.
4
5  In order for the algorithm not to crash we can divide the segments so that
6  would contain same number of pixels in each segment, instead of dividing them
7  in z equal parts of 255.
```

4 ex1/sol1.py

```python
"""
Ex1 - Image processing.
Name: Tal Porezky
ID: 311322499
C.S.E: tal.porezky
Email: tal.porezky@mail.huji.ac.il
"""

# IMPORTS #
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from imageio import imread, imwrite

# CONSTANTS #
GRAY_INDEX = 1
RGB_INDEX = 2

RGB_TO_YIQ_MATRIX = np.array([[0.299, 0.587, 0.114],
                              [0.596, -0.275, -0.321],
                              [0.212, -0.523, 0.311]])


def read_image(filename, representation):
    """
    function which reads an image ~Lle and converts it into a given
    representation.
    :param filename: the filename of an image on disk (could be grayscale or
    RGB).
    :param representation: representation code, either 1 or 2 defining
    whether the output should be a grayscaleimage (1) or an RGB image (2).
    If the input image is grayscale, we won't call it with representation = 2.
    :return: rgb or grayscale img
    """
    image = imread(filename)
    new_image = image.astype(np.float64)
    new_image /= 255
    if representation == GRAY_INDEX:
        new_image = rgb2gray(new_image)
    return new_image


def imdisplay(filename, representation):
    """
    Display an image in a given representation.
    :param filename: the filename of an image on disk (could be grayscale or
    RGB).
    :param representation: representation code, either 1 or 2 defining
    whether the output should be a grayscaleimage (1) or an RGB image (2).
    :return: show the image. return nothing
    """
    plt.figure()
    plt.axis('off')
    image = read_image(filename, representation)
    if (len(image.shape) == 2):
        plt.imshow(image, cmap='gray')
    else:
        plt.imshow(image)
    plt.show()
```

```
60
61
62  def rgb2yiq(imRGB):
63      """
64      Convert image from rgb to yiq
65      :param imRGB: an image in rgb protocol
66      :return: an yiq image.
67      """
68      return np.dot(imRGB, RGB_TO_YIQ_MATRIX.T)
69
70
71  def yiq2rgb(imYIQ):
72      """
73      convert image from yiq to rgb
74      :param imYIQ: an image in yiq protocol
75      :return: an rgb image
76      """
77      return np.dot(imYIQ, np.linalg.inv(RGB_TO_YIQ_MATRIX).T)
78
79
80  def histogram_equalizer_helper(im_orig):
81      """
82      helper to the histogram_equalize function
83      :param im_orig: the image to equalize
84      :return:  equlized image [0-255], the histogram of the original image,
85      and the histogram of the modified image.
86      """
87      im_orig = (im_orig * 255).astype(np.uint8)
88      hist_orig, bins = np.histogram(im_orig,
89                                     bins=256,
90                                     range=(0, 255))
91      cumulative_histogram = np.cumsum(hist_orig)
92      normalized_cumulative_histogram = cumulative_histogram / \
93                                        cumulative_histogram[-1]
94      max_grey_level = 255
95      modified_by_max_grey_cumulative_histogram = normalized_cumulative_histogram * \
96                                                  max_grey_level
97
98      for min_val_for_zero in modified_by_max_grey_cumulative_histogram:
99          if min_val_for_zero == 0:
100             continue
101         else:
102             modified_by_max_grey_cumulative_histogram -= min_val_for_zero
103             break
104
105     assert(max(modified_by_max_grey_cumulative_histogram) != 0)
106     modified_by_max_grey_cumulative_histogram *= (255 / max(modified_by_max_grey_cumulative_histogram))
107
108     rounded_histo = modified_by_max_grey_cumulative_histogram.astype(np.uint8)
109
110     equalized_image = rounded_histo[im_orig].astype(np.float64)
111
112     return equalized_image, hist_orig, np.histogram(equalized_image,
113                                                     bins=256,
114                                                     range=(0, 255))[0]
115
116
117 def histogram_equalize(im_orig):
118     """
119     this function manage the histogram equalize program. it deal with an
120     RGB images.
121     :param im_orig: the image to equalize
122     :return: equlized image [0-255], the histogram of the original image,
123     and the histogram of the modified image.
124     """
125
126     if len(im_orig.shape) == 2:
127         eq_image, hist_orig, histo_eq = histogram_equalizer_helper(im_orig)
```

```
    else:
        yiq_image = rgb2yiq(im_orig)
        gray_scale_img = yiq_image[:, :, 0]
        eq_image_gray_scale, hist_orig, histo_eq = histogram_equalizer_helper(gray_scale_img)
        eq_image_gray_scale /= 255
        yiq_image[:, :, 0] = eq_image_gray_scale
        yiq_image *= 255
        eq_image = yiq2rgb(yiq_image)

    return [eq_image / 255, hist_orig, histo_eq]


def z_0_calculator(histo, n_quant):
    """
    this function initiate the z array at the first time
    :param histo: an histogram of the image
    :param n_quant: the number of shades in the output image
    :return: an array of the z points
    """
    assert (n_quant >= 1)
    z = np.zeros(n_quant + 1, dtype=np.uint8)
    cum = np.cumsum(histo)
    num_of_pixels = cum[-1]
    pixels_in_segment = num_of_pixels / n_quant
    for seg_idx in range(1, n_quant):
        z[seg_idx] = np.where(cum >= seg_idx * pixels_in_segment)[0][0]
    z[-1] = 255
    return z


def q_calculator(histo, bin, z):
    """
    this function find the best q points for a current iteration
    :param histo: an histogram of the image
    :param bin: all of the possible points in the image
    :param z: an array of the z points n a current iteration
    :return: an array of the q points
    """
    q = np.zeros(len(z) - 1, dtype=np.float64)
    for i in range(1, len(z)):
        product_up = np.round(histo[z[i - 1]: z[i]] * bin[z[i - 1]: z[i]])
        sum_up = np.sum(product_up)
        sum_down = np.sum(histo[z[i - 1]: z[i]])
        q[i - 1] = np.round(sum_up / sum_down)
    return q


def z_calculator(q):
    """
    this function find the z points according to the q points
    :param q: an np array of the q points
    :return: an np array of the z points
    """
    z = np.zeros(len(q) + 1, dtype=np.uint8)
    for i in range(1, len(z) - 1):
        z[i] = np.ceil((q[i - 1] + q[i]) / 2)
    z[-1] = 255 #
    return z


def error_calculator(q, z, histo, bin):
    """
    this function find the error rate for a current iteration
    :param q: an np array of the q points
    :param z: an np array of the z points
    :param histo: an histogram of the image (0-255)
    :param bin: all of the possible values in the image (0-255)
    :return: an rate of the error for a currant iteration
```

```
    """
    seg_score = np.zeros(len(q))
    for i in range(len(q)):
        temp_sum = np.power(q[i] - bin[z[i]: z[i + 1]] , 2) * \
                   histo[z[i]: z[i + 1]]
        seg_score[i] = np.ceil(np.sum(temp_sum))
    iter_score =  np.sum(seg_score)
    return iter_score


def quantize_helper(im_orig, n_quant, n_iter):
    """
    quantize function helper.
    :param im_orig: is the input grayscale or RGB image to be quantized (
    oat64 image with values in [0; 1]).
    :param n_quant: is the number of intensities your output im_quant image
    should have.
    :param n_iter: is the maximum number of iterations of the optimization
    procedure (may converge earlier.)
    :return: [im_quant, error] such that:
    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.
    """"""
    histo, bins = np.histogram(im_orig, bins=np.arange(257))
    z_0 = z_0_calculator(histo, n_quant)
    q = q_calculator(histo, bins, z_0)
    error = list()

    error.append(error_calculator(q, z_0, histo, bins))
    z = z_0
    for i in range(1, n_iter):
        z = z_calculator(q)
        if np.array_equal(z, z_0):
            break
        q = q_calculator(histo, bins, z)
        error.append(error_calculator(q, z, histo, bins))
        z_0 = z

    lookup = np.zeros(257)
    for i in range(n_quant):
        lookup[z[i]: z[i + 1] + 1] = q[i]

    image = np.interp(im_orig, bins, lookup)

    return image, error


def quantize(im_orig, n_quant, n_iter):
    """
    function that performs optimal quantization of a given grayscale or RGB image.
    :param im_orig: is the input grayscale or RGB image to be quantized (
    oat64 image with values in [0; 1]).
    :param n_quant: is the number of intensities your output im_quant image
    should have.
    :param n_iter: is the maximum number of iterations of the optimization
    procedure (may converge earlier.)
    :return: [im_quant, error] such that:
    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.
    """
    im_orig = im_orig * 255
    if len(im_orig.shape) == 3:
        yiq_img = rgb2yiq(im_orig)
        image = yiq_img[:, :, 0]
        image, error = quantize_helper(image, n_quant, n_iter)
        uni_image = np.array(
```

```
264                 [image.T, yiq_img[:, :, 1].T, yiq_img[:, :, 2].T]).T
265             image = yiq2rgb(uni_image)
266 
267         else:
268             image, error = quantize_helper(im_orig, n_quant, n_iter)
269 
270         return [image / 255, error]
271 
```