Image processing course – homework #1

imageprocessinghaifau@gmail.com

In this homework you will be writing python functions that use histograms, calculates the SLT transform and measures distances.

Please read the whole document before implementing functions and script.
You are provided with two files hw1_functions.py and hw1_script.py, please fill them with necessary implementation.

Getting started:

- Open a new PyCharm project.
- Install new packages using pip (as shown in the tirgul#1 presentation)
  - pip install numpy
  - pip install opencv-python
  - pip install matplotlib

hw1_functions.py:

1. **contrastEnhance(im, range):**

   This function maps an image to new range of gray values. It can be used to enhance contrast.
   mapping is linear in the form $g_{new} = a * g_{old} + b$
   (as taught in class)

   Input: im - a grayscale image in the range [0..255]
   range – range of grayvalues in the form [minVal, maxVal].
   Output: nim – the new grayscale image (same size as im) – a 2D numpy array.
   a,b - the parameters of the Tone Mapping that performs the mapping.
   Method: Maps image im such that the new image nim has values in the range given as input.
   Mapping is linear and is in the form $g_{new} = a * g_{old} + b$.
   Function returns mapped image as well as the parameters a and b.
   Example usage: nim, a, b = contrastEnhance(im, [10, 250]):

2. **Minkowski2Dist(im1,im2**)

This function measures the minkowski distance between (histograms of) two images. Uses p=2.

Input: im1, im2 – 2D numpy matrices. These are grayscale images with values in the range [0..255]
Output: d the distance value.
Method:
You may use np.histogram(im1, bins = 256, range=(0,255))
use numpy functions: sum, power, casting to float
Note: im1 and im2 need not be same size.

3. **MeanSqrDist(im1,im2)**

This function measures the mean square distance between two images

Input: im1, im2 – matrices of grayscale images of the same size in range [0..255].
Output: d the distance value.
Method:
Distance is the mean of the squared distances between corresponding pixels of the 2 IMAGES (not histograms). This function should be done with one line.
Make sure values are float!!

4. **sliceMat(im)**

This function builds the Slice Matrix, a binary valued matrix of size numPixel X 256.
Input: im - a 2D grayscale image matrix in the range [0..255]
Output: SL – a binary valued matrix (2D numpy array) of size numPixelx256 (numPixel is the number of pixels in im)

Method: The SL matrix contains the slices along the columns. Column i is associated with gray value i. The column i contains 1 in entry k if image im(:) has value i at index k.
That is: SL(k,i) == 1 if im(k) = i.
(see lesson slides).

You only need one loop in this function (loop over 256 gray levels).
Do not loop on the image pixels to create Boolean masks, use python indexing:

mask = im==200
this return a boolean mask of size image that holds true where im is equal to graylevel 200, and false everywhere else.

Use np.transpose if needed.

5. **SLTmap (im1,im2)**

Tone Maps image im1 to be as similar to im2.
Outputs the mapped image as well as the tone mapping function.

Input: im1, im2 - grayscale images of the same size in the range [0..255]
Ouput: nim – the new grayscale image (same size as im1).
TM – tone map = a 1x256 vector defining a mapping.

Method: Uses the SL matrix for im1 calculated using **sliceMat**.
For each slice (column) i, finds the gray values in im2 that correspond to values of 1 in the slice. These gray values are averaged to create the value inew to which gray scale i is mapped:
TM[i] = inew.
Image im1 is tone mapped using TM to create new image nim.

Do matrix multiplication using np.matmul.
Cast to float when calculating average.

6. **mapImage (im,tm)**

Maps image grayscale according to given toneMap. Returns new image. (use sliceMat function in this function).

Input: im - a grayscale image matrix in the range [0..255]
tm – tone map = a 1x256 (numpy array) defining a tone mapping.

Output: nim – the new grayscale image (same size as im).

Method: Map gray value i in image to new value given in tm(i).
use np.reshape to return nim of same size as im.

7. **sltNegative (im)**

maps image to it's negative (use mapImage function in this function)

Input: im - a grayscale image matrix in the range [0..255]

Output: nim – the negative grayscale image.

Method: using slt slicing and mapping, calculate negative image

Use np.arange(r) to create an array of values [0,1,2,3,…, r]

8. **sltThreshold (im, thresh)**

performs thresholding on image (use mapImage function in this function)

Input: im - a grayscale image matrix in the range [0..255]
thresh – threshold value – an int number

Output: nim – binary image (2D numpy array).

Method: using slt slicing and mapping , perform thresholding
(thresholding = each graylevel > thresh turns to 255 and each graylevel <= thresh turn to 0).
Use np.arange(r) to create an array of values [0,1,2,3,…, r]

hw1_script.py:

use: **import hw_functions** to load all functions to the script file.

a. Read darkImage.tif. Enhance contrast to maximum range (using contrastEnhance function).
   i. Display original image and enhanced image.
   ii. print the a and b values of the mapping.
   iii. Plot the Tone mapping (showMapping function is provided).
b. perform maximum contrast enhancing on an already enhanced (image from previous section). Show the difference between both images.
(do NOT simply display both images).
c. Test minkowski2Dist:
   i. Show that distance between image and itself is 0.

ii. Plot the distance between the image and a contrast enhanced version of the image:
Perform a loop of k=1:20 steps (to cover the whole range between min and max)
Each iteration the contrast enhanced version of the image is of contrast [min, min+ k * step].
The contrast is larger along the X axis (since we are increasing k with each iteration).
The Y axis is the minkowski2Dist between im and contrast enhanced im.

For example, if img have values [50,150]:
Iteration 1: enhanced image of contrast [50,55]
Iteration 2: enhanced image of contrast [50,60]
Iteration k=2: enhanced image of contrast [50,150]

d. Test sliceMat: show that sliceMat(im) * [0:255] == im
(do NOT simply display both images)

e. Show that sliceMat(im) * TM produces a tone mapped version of im.
Do so by calculating the maximum contrast enhancement of im. Find the Tone Map TM that creates this contrast enhancement. Compare the contrast enhanced im with sliceMat(im) * TM.
Also display im and its tone mapped version.

f. Test sltNegative **:**
produce the negative image of im.

g. Test sltThreshold:
produce a binary image by defining TM that performs thresholding of im, using threshold.

h. Test SLTmap:
   i. Choose 2 image im1 and im2. Display im1, SLTmap(im1,im2) and im2.
   ii. Show that the meanSqrDist distance between SLTmap(im1,im2) and im2 is smaller than the distance between im1 and im2.

i. Is SLTmap symmetric? That is does SLTmap(im1,im2) == SLTmap(im2,im1). If so, show computationally. display both images.

Student are required to submit two .py files:

- hw1_functions.py
- hw1_script.py

Tips:

- **hw1_functions.py has a print_IDs function (replace the students ids in it and use the function to print the ids in the start of your script).**
- when reading an image, pay attention of how many dimensions it has ([m,n] or [m,n,3]).
  use cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) to get rid of the 3 channels and have a grayscale image.
  Only when asked to view original image use the 3 channels image.

- pay attention to arrays types. For example, if an array is of type uint8, it's values are [0,255]. np.power(array,2) will also be of values [0,255]. Values won't exceed 255 unless you cast it to float instead of uint8.
  The casting should be done inside the functions.

- In python, you can return multiple variables from a function. You don't need to wrap them in a list.

  For example:
  Def function(a,b):
  C = a+b
  D = a-b
  Return C,D

- if asked to show two images in one section (for visual comparison). Do so by using:

  plt.figure()
  plt.subplot(1,2,1)
  plt.imshow(img1)
  plt.subplot(1,2,2)
  plt.imshow(img2)

  it will display two images in one figure (side by side).

- to calculate histogram of image:
  np.histogram(image, bins=256, range=(0,255))
  it returns an array. you can then divide the array by number of elements in image to get a normalized one.

- You can use functions from numpy such as min, max, power, sum,arange, round, ravel (this function turns an MxN matrix into MNx1 array, use it before calculating max,min… so that the min/max is done on all the values)
  read about all the functions mentioned, they're all used.

- all functions you're required to implement are less than 10 lines of code.
  try to simplify.

Good luck!