**ESE 358 Computer Vision, Fall 2021**

**Electrical and Computer Engineering, Stony Brook University**

**Murali Subbarao**

**Project 1. Basic operations on RGB pixel values**

**References:**

1. [Python Tutorial (brown.edu)](https://cs.brown.edu/courses/csci1430/2021_Spring/resources/python_tutorial/)
2. <https://scikit-image.org/docs/stable/user_guide/numpy_images.html?highlight=image%20io>
3. [Python Numpy Tutorial (with Jupyter and Colab) (cs231n.github.io)](https://cs231n.github.io/python-numpy-tutorial/)
4. [numpy.pdf (ucsb.edu)](https://sites.engineering.ucsb.edu/~shell/che210d/numpy.pdf)

Understand the code given below at the end, run it, and modify it to complete this project

1. Read a given RGB image A (8 bits/pixel) in jpeg format. The size of A will be 512x512.
2. Display the image, and display the images corresponding to the three channels R,G,B, as three separate images, and write these images RC, GC, BC, in jpeg format.
3. Compute the gray-level image AG by taking the average of the three channels R,G, and B, at each pixel.
4. Compute the histogram of the images RC, GC, BC, and AG, and plot them on the monitor.
5. (1 point) Threshold the image AG to obtain a binary image AB. Read the threshold TB as input at runtime (e.g. TB=100), with all pixels less than T being assigned 0 and others being assigned 255. Display the output image AB.
6. (1 point) Simple edge detection: For the image AG of size MxN, read in a threshold value TE (e.g. TE =15), compute the gradient along rows Gx(m,n) = AG(m,n+1)-AG(m,n) [except at the last pixel of each row m; it is set to zero] compute the gradient along columns Gy(m,n) = AG(m+1,n)-AG(m,n) [except at the last pixel of each column n; it is set to zero]. Then compute the gradient magnitude at each pixel as

GM(m,n)= sqrt( Gx(m,n)^2+Gy(m,n)^2).

Threshold the gradient magnitude with TE to compute an edege image AE as

AE(m,n)=255 if GM(m,n)>TE; else AE(m,n)=0.

Display the edge image AE.

1. (2 points) Compute and display an image pyramid of AG with image sizes half, quarter, and 1/8th the size of AG, by taking the average of 2x2 image blocks to obtain the corresponding pixels in the successive images AG2, AG4, and AG8. Display the 3 output images.
2. Submit the following on Blackboard: Create a Microsoft docx format file that contains-- source code, displayed input, output, for each part above. Also, submit a zip file will all files including source files, input and output image files.

SKIP THE FOLLOWIN TWO PARTS

1. Read another RGB image B (8bits/pixel) in jpeg format similar to A (e.g. A and B can a stereo image pair with cameras at different positions and directions), and compute its gray-level image BG by taking the average of the three channels R,G, and B, at each pixel.
2. Divide AG uniformly into 16x16 image bocks. Each block is identified by the row and column indices of the pixel at its top left corner. For each block in AG, find the most similar block in BG using the minimum of the sum of squared-differences of corresponding pixels.

Create an output image given by the absolute difference between the corresponding blocks of AG and BG, and display the result as a gray-level image.

# Understand the following code and modify it.

#

# -\*- coding: utf-8 -\*-

"""

Created on Fri Aug 13 13:24:56 2021

Original template:

https://cs.brown.edu/courses/csci1430/2021\_Spring/resources/python\_tutorial/

Edited by

@author: murali subbarao, sbu, ece

"""

# Read in original RGB image.

import numpy as np

from skimage import io, color

import matplotlib.pyplot as plt

rgbImage = io.imread('olympics1.jpg')

(m,n,o) = rgbImage.shape

# Extract color channels.

redChannel = rgbImage[:,:,0] # Red channel

greenChannel = rgbImage[:,:,1] # Green channel

blueChannel = rgbImage[:,:,2] # Blue channel

# Create an all black channel.

allBlack = np.zeros((m, n), dtype=np.uint8)

# Create color versions of the individual color channels.

justRed = np.stack((redChannel, allBlack, allBlack), axis=2)

justGreen = np.stack((allBlack, greenChannel, allBlack),axis=2)

justBlue = np.stack((allBlack, allBlack, blueChannel),axis=2)

# Recombine the individual color channels to create the original RGB image again.

recombinedRGBImage = np.stack(( redChannel, greenChannel, blueChannel),axis=2)

plt.imshow(recombinedRGBImage)

plt.show()

print(justRed.shape)

print(justRed.dtype)

print(justRed.size)

print(justRed[0:5, 0:5, 0])

io.imsave('justRed1.jpg' , justRed)

# image values not normalized to be in 0.0 to 1.0) range

imageFloat1 = rgbImage.astype(np.float32)

plt.imshow(imageFloat1)

plt.show()

print(imageFloat1[0:5, 0:5 , 0])

#normalize the image and display

from skimage import img\_as\_float

floatImage2 = img\_as\_float(rgbImage)

plt.imshow(floatImage2)

plt.show()

print(floatImage2[0:5, 0:5, 0])

for i in range(5,10) :

for j in range(20,25) :

print(floatImage2[i,j,0])

print('\n')

#compute the histogram

hist=np.zeros(256)

for i in range(m) :

for j in range(n) :

hist[rgbImage[i,j,0]] += 1

#print(hist)

import matplotlib.pyplot as plt

#plot example

x = np.arange(0, 5, 0.1)

y = np.sin(x)

plt.plot(x, y)

#plot the histogram

plt.plot(hist)

#grayscale

plt.imshow(greenChannel, cmap=plt.cm.gray)

#######

import matplotlib.pyplot as plt

from skimage import data

from skimage.color import rgb2gray

original = data.astronaut()

grayscale = rgb2gray(original)

fig, axes = plt.subplots(1, 2, figsize=(8, 4))

ax = axes.ravel()

ax[0].imshow(original)

ax[0].set\_title("Original")

ax[1].imshow(grayscale, cmap=plt.cm.gray)

ax[1].set\_title("Grayscale")

fig.tight\_layout()

plt.show()