Homework3 , CS 682 Spring20
S M Hasan Mansur

- I have posted the results and scripts to http://mason.gmu.edu/~smansur4/cs682hw3/
- Login credentials:
  Username: testuser
  Password: testpass

Problem 1

For gray level edge detection I have used the **cv.Canny()** method. I computed gray level gradients using **cv.Sobel()** as these will be required in the next steps.

For color edges,I used **cv.Sobel()** to compute the gradients. This operation provides color gradients for 3 color channels (RGB). Then I splitted the **rx,gx,bx,ry,gy,by** components of the gradients.

Problem2

For gray images, I used the gradients from previous step (**sobel operation**) & **cv.cartToPolar()** method to compute magnitude and angle orientation. Then I used **np.histogram()** method to compute the histogram of gradients.The histogram uses 36 bins in X-axis. In the Y-axis, I plotted the count of all selected edges using the same value.

Problem3

For color images, using the provided formulas, I aggregated 3 color component gradients along the corresponding axis. Then I used the **cv.cartToPolar()** method to compute magnitude and angle orientation and **np.histogram()** method to compute the histogram of gradients, just like the way I did in Problem2.

Problem4

For histogram comparison, I wrote two functions histogram_intersection() & histogram_chi2(). For implementing intersection, I used the methods: np.true_divide(), np.sum(), np.minimum(), np.maximum(). For implementing chi square, I used the methods: np.nansum(), np.true_divide(), np.square(), np.subtract(), np.add().

Problem5

To compare all image pairs, I utilized the functions developed in the previous step (Problem4). As there were a total of 99 images, the results were kept in four 99x99 matrices (two for color histogram intersection & chi square, another two for gray histogram intersection & chi square). Finally, I plotted the results using matplotlib. In case of intersection, large values correspond to high similarity and for chi-squared measure, low values correspond to high similarity.

Extra Credit 1

First, I splitted the **rx,gx,bx,ry,gy,by** components of the gradients (which I got from sobel operation). Then I aggregated the 3 color component gradients along the corresponding axis. Finally, based on those values, I used the **quiver()** method from matplotlib to visualize the gradients.

Extra Credit 2 & 3

First, I splitted the **rx,gx,bx,ry,gy,by** components. Then I computed the eigenvalues and eigenvectors using the steps suggested in the provided slides. Finally I computed the color gradients using the eigenvalues and eigenvectors. For faster computation, I didn't write any python loop to compute eigenvalues and eigenvector. Rather I used numpy operations as suggested in the provided slide.

References:

1. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
2. https://jakevdp.github.io/PythonDataScienceHandbook/02.02-the-basics-of-numpy-arrays.html
3. https://docs.opencv.org/master/d5/d0f/tutorial_py_gradients.html
4. https://docs.opencv.org/master/da/d22/tutorial_py_canny.html
5. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.quiver.html
6. https://pythonforundergradengineers.com/quiver-plot-with-matplotlib-and-jupyter-notebooks.html
7. https://docs.opencv.org/master/d4/d70/tutorial_anisotropic_image_segmentation_by_a_gst.html