CS 3430: SciComp with Py
Assignment 9
Image Indexing and Retrieval with Color Image Histograms in RGB and HSV Spaces

Vladimir Kulyukin
Department of Computer Science
Utah State University

March 23, 2018

1 Learning Objectives

1. Image Histograms
2. Histogram Similarity Functions
3. Image Matching and Retrieval
4. Matplotlib
5. OpenCV
6. Persistent Objects

2 Introduction

In this assignment, we will implement another image indexing and retrieval method. However, unlike in the previous assignment, where we explored image retrieval with feature vectors that consist of mean row values of `B`, `G`, `R` or `H`, `S`, `V` pixel values, this assignment will focus on using `RGB` and `HSV` histograms to index and retrieve images. We will also index many more images than in the previous assignment.

3 Indexing Images with Color Histograms

I have posted 8 zip archives as Canvas announcements for this assignment. The archives are:

- `hw09_images_01.zip`;
- `hw09_images_02.zip`;
- `hw09_images_03.zip`;
- `hw09_images_04.zip`;
- `hw09_images_05.zip`;
- `hw09_images_06.zip`;
- `hw09_images_07.zip`;
- `hw09_test_images.zip`.

After you download these archives, place them into the `images` directory on your pi. The archives contain the images of various streets in Logan familiar to you from the previous assignment. These are the images taken from a pi camera in my Wrangler. I periodically drive around with my grad students to update our road archives that we use in various research projects. The archives also contain lunch tray images from several Logan schools. I received them from Dr. Heidi Wengreen, a USU nutrition professor, with whom I collaborated on a food texture recognition project. These are the images we will use for persistent image indexing. The archive `hw09_test_images.zip` contain two subdirectories: `car_test` and `food_test` that we will use for image retrieval.

Histograms are worth considering any time one needs to retrieve images similar to a given image from a database of images or finding an object in an image. Histogram matching is based on the implicit assumption that similar images have similar histograms. Of course, this is not always true. For example, a blue ball will have a histogram similar to a blue cube and quite different from the histogram of a red ball. Therefore, histograms may not be sufficient for accurate image retrieval without additional geometrical or topological information.

Implement the function `hist_index_img`. This function takes an image path `imgp`, a string specification of the color space (either `'rgb'` and `'hsv'`), and the bin size, i.e., the number of bins in each color channel in the histogram. Recall that the bin size parameter is used in the OpenCV `cv2.calcHist()` function. For example, if you want to use 8 bins for each channel to index the image `img`, the call to `cv2.calcHist()` looks as follows:

```
cv2.calcHist([img], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
```

The function `hist_index_img` places the key-value pair `(imgp, norm_hist)` in the dictionary `HIST_INDEX`. Depending on the value of `color_space`, the `norm_hist` is either the normalized and flattened `RGB` histogram of the image in `imgp` or the normalized and flattened `HSV` histogram of that image.

```
HIST_INDEX = {}

def hist_index_img(imgp, color_space, bin_size=8):
  global HIST_INDEX
  ## your code
  pass
```

Use your impelementation of `hist_index_img` to write the function `hist_index_img_dir` that uses `hist_index_img` to index each image in a given directory with the specified color space and bin size. The latter two parameters are specified on the command line.

```
# a few mandatory command line parameters
ap = argparse.ArgumentParser()
ap.add_argument('-imgdir', '--imgdir', required=True, help='image directory')
ap.add_argument('-hist', '--hist', required=True, help='histogram index file')
ap.add_argument('-bin', '--bin', required=True, help='histogram bin size')
ap.add_argument('-clr', '--clr', required=True, help='color space')
args = vars(ap.parse_args())

def hist_index_img_dir(imgdir, color_space, bin_size):
  # your code
  pass

# this is the main block that computes HIST_INDEX and pickles it in the file
# specified in the command line parameter -hist.
if __name__ == '__main__':
  hist_index_img_dir(args['imgdir'], args['clr'], int(args['bin']))
  with open(args['hist'], 'wb') as histpick:
    pickle.dump(HIST_INDEX, histpick)
  print('indexing finished')
```

Here is a test run that computes the normalized flattened 16-bin `RGB` histograms of each image in `images` and persists them in `rgb_hist16.pck`.

```
$ python hist_image_index.py -imgdir images/ -clr rgb -hist rgb_hist16.pck -bin 16
...
indexing images/16_07_02_14_50_48_orig.png
images/16_07_02_14_50_48_orig.png indexed
images/16_07_02_14_37_38_orig.png
images/16_07_02_14_37_38_orig.png indexed
images/123473019.JPG
images/123473019.JPG indexed
indexing finished
```

Here is a test run that computes the normalized flattened 8-bin HSV histograms of each image in `images` and persists the resultant dictionary in `hsv_hist8.pck`.

```
$ python hist_image_index.py -imgdir images/ -clr hsv -hist hsv_hist8.pck -bin 8
...
indexing images/16_07_02_14_50_48_orig.png
```

```
images/16_07_02_14_50_48_orig.png indexed
images/16_07_02_14_37_38_orig.png
images/16_07_02_14_37_38_orig.png indexed
images/123473019.JPG
images/123473019.JPG indexed
indexing finished
```

After you implement `hist_image_index.py`, run the following indexing operations to create four persistent pickle indices: `rgb_hist8.pck`, `rgb_hist16.pck`, `hsv_hist8.pck`, `hsv_hist16.pck`.

```
$ python hist_image_index.py -imgdir images -clr rgb -hist rgb_hist8.pck -bin 8
$ python hist_image_index.py -imgdir images -clr rgb -hist rgb_hist16.pck -bin 16
$ python hist_image_index.py -imgdir images -clr hsv -hist hsv_hist8.pck -bin 8
$ python hist_image_index.py -imgdir images -clr hsv -hist hsv_hist16.pck -bin 16
```

4 Retrieving Images

Let's move on to image retrieval. Write the Py program `hist_index_retrieval.py` that finds and displays the top 3 images similar to the user-specified image by computing the similarity scores between this image and all images in a given persisted histogram index. The top three images and the input image are displayed in four separate matplotlib figures. The similarity scores of the top three matches in the command window. Let's start with the following self-explanatory input parameters.

```
if __name__ == '__main__':
  ap = argparse.ArgumentParser()
  ap.add_argument('-ip', '--imgpath', required = True, help = 'image path')
  ap.add_argument('-hist', '--hist', required = True, help = 'hist index file')
  ap.add_argument('-bin', '--bin', required = True, help = 'hist bin size')
  ap.add_argument('-clr', '--clr', required = True, help= 'color space')
  ap.add_argument('-sim', '--sim', required = True, help = 'hist similarity')
  args = vars(ap.parse_args())
```

The main block of `hist_image_retrieval.py` looks as follows.

```
if __name__ == '__main__':
  with open(args['hist'], 'rb') as histfile:
    HIST_INDEX = pickle.load(histfile)
  sim_list = compute_hist_sim(inhist_vec, HIST_INDEX)
  for imagepath, sim in sim_list:
    print(imagepath + ' --> ' + str(sim))
  show_images(inimg, sim_list)
```

In the main, we unpickle the histogram index specified in `-hist` into `HIST_INDEX`. The function `compute_hist_sim` takes the normalized and flattened histogram of the input image specified by the user with `-ip` and returns the similarity list of the three top matches. Each match is a 2-tuple, the first element of which is an image path and the second element is a float similarity score. The function `show_images` takes the image `inimg` read from `-ip` and the top 3 matches in `sim_list` and displays four images in four separate matplotlib figures.

Below is a test run where we use the 8-bin rgb index to find the top 3 similar images by using the histogram intersection similarity function, i.e., `cv2.HISTCMP_INTERSECT`. Recall the four OpenCV similarity metrics covered in Lecture 17: `correlation`, `chi square`, `intersection`, and `bhattacharrya`. Your implementation should allow the user to specify each of the four similarity metrics with the `-sim` parameter to `hist_image_retrieval.py`. Use the following strings in command line: `correl` for `correlation`, `chisqr` for `chi square`, `inter` for `intersection`, and `bhatta` for `bhattacharrya`. Remember that for `correlation` and `intersection`, the higher the score, the closer the match whereas, for `chi square` and `bhattacharrya`, the lower the score, the closer the match.

```
$ python hist_image_retrieval.py -ip food_test/img01.JPG -clr rgb -hist rgb_hist8.pck -bin 8 -sim inter
```

Your command line output from the above call should look something like this.

```
images/123461762.JPG --> 2.69072864504
images/123465049.JPG --> 2.63319342056
images/123476552.JPG --> 2.51518283323

match 1 images/123461762.JPG
match 2 images/123465049.JPG
match 3 images/123476552.JPG
```

The figures of the input and top match images are in Figure 1. Note that the match figure title displays the similarity score. The figures of the second and third matches are in Figure 2.

Figure 1: Input image (Figure 1) and first match image (Figure 2)

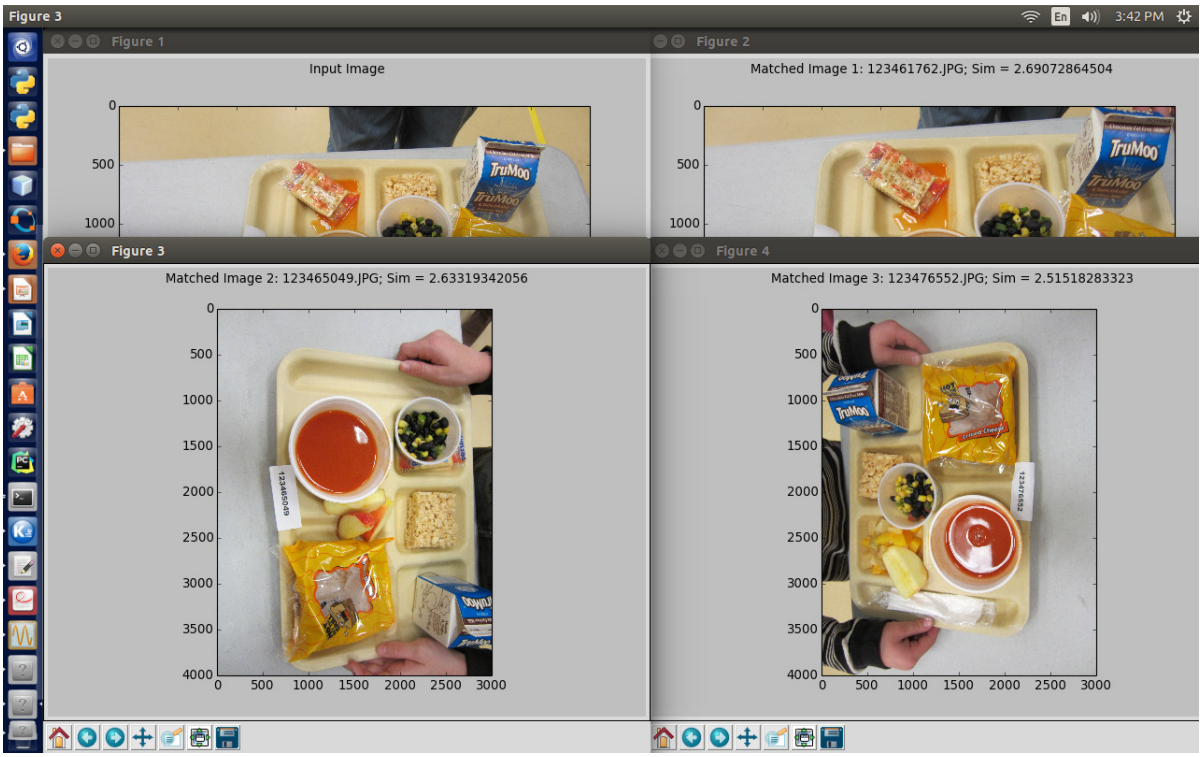

Figure 2: Second match image (Figure 3) and third match image (Figure 4)

5 What To Submit

Save your persistent 8- and 16-bin `RGB` and `HSV` dictionaries in `rgb_hist8.pck`, `rgb_hist16.pck`, `hsv_hist8.pck`, and `hsv_hist16.pck`. Place your source code files, `hist_image_index.py` and `hist_image_retrieval.py`, and your four pck files in `hw08.zip` and submit it via Canvas.

Happy Hacking!