CS 3430: SciComp with Py
Assignment 8
Image Indexing and Retrieval

Vladimir Kulyukin
Department of Computer Science
Utah State University

March 17, 2018

1 Learning Objectives

1. Image Processing
2. OpenCV
3. BGR, HSV, and Grayscale (GSL) Color Spaces
4. Persistent Objects
5. I/O

2 Introduction

In this assignment, we will implement a simple image indexing and retrieval engine. An image retrieval task can be formulated as follows. Given a collection of images, index each image in the collection in terms of some features present in the image. This step is called *indexing*. The output of the indexing step is a collection index. This index can be an SQL database, a text file, or a persistent object such as a dictionary.

Given a collection index, we can take an image and find and display the image most similar to the input image. This step is called *retrieval*. Of course, we can relax the similarity retrieval requirement and retrieve the top 10/20/100 most similar images. But, for simplicity, we will keep the retrieval confined to just one image in this assignment.

3 Indexing Images

The attached zip archive for this assignment contains two image directories `car/` and `car_test/`. The directory `car/` contains 255 images of various streets in Logan that I took with a raspberry pi camera in my Wrangler for a lane detection project. We will use the images in `car/` to create three persistent index objects. The directory `car_test/` contains a smaller set of 23 images that we will use as input test images.

In the BGR space, an image I can be split into the Blue (B), Green (G), and Red (R) channels, as was shown in class. Each of these channels is also an image of the same dimensions as I. For each row r in I, we can compute three means: the mean value of the row in B, the mean value of the row in G, and the mean value of the row in R. The index of I is a list of 3-tuples $(\mu_r^B, \mu_r^G, \mu_r^R)$, where r ranges from 0 upto and the last row in I and μ_r^B, μ_r^G, μ_r^R are the means of the blue, green, and red values, respectively, for row r.

Let's look at a couple of simple example. Suppose that I is the following 2x3 image.

```
[[[b0, g0, r0], [b1, g1, r1], [b2, g2, r2]],
 [[b3, g3, r3], [b4, g4, r4], [b5, g5, r5]]]
```

After the image is split into the B, G, and R channels, the channels are as follows:

```
B = [[b0, b1, b2], [b3, b4, b5]]
G = [[g0, g1, g2], [g3, g4, g5]]
R = [[r0, r1, r2], [r3, r4, r5]]
```

The BGR index of I is then computed as follows:

```
bgri = [((b0+b1+b2)/3, (g0+g1+g2)/3, (r0+r1+r2)/3),
        ((b3+b4+b5)/3, (g3+g4+g5)/3, (r3+r4+r5)/3)],
```

where

1. $\mu_0^B = \frac{b0+b1+b2}{3}$;
2. $\mu_0^G = \frac{g0+g1+g2}{3}$;
3. $\mu_0^R = \frac{r0+r1+r2}{3}$;
4. $\mu_1^B = \frac{b3+b4+b5}{3}$;
5. $\mu_1^G = \frac{g3+g4+g5}{3}$;
6. $\mu_1^R = \frac{r3+r4+r5}{3}$.

Similarly, in the HSV space, the image I can be split into the Hue (H), Saturation (S), and Value (V) channels. Suppose, again, I is a 2x3 image. Then the three channels are as follows:

```
H = [[h0, h1, h2], [h3, h4, h5]]
S = [[s0, s1, s2], [s3, s4, s5]]
V = [[v0, v1, v2], [v3, v4, v5]]
```

The HSV index of I is then computed as

```
hsvi = [((h0+h1+h2)/3, (s0+s1+s2)/3, (v0+v1+v2)/3),
        ((h3+h4+h5)/3, (s3+s4+s5)/3, (v3+v4+v5)/3)]
```

What happens if an image is grayscale? If the image I is grayscale or is converted to grayscale, it will have only one value per pixel.

```
[[g0, g1, g2],
 [g3, g4, g5]]
```

In this case, the grayscale index of I consists of only two means:

```
gsli = [(g0+g1+g2)/3, (g3+g4+g5)/3]
```

Suppose that the image I is saved in `cars/16_07_02_14_21_00_orig.png` and its three indices (BGR, HSV, and grayscale) have been computed and saved as `bgri`, `hsvi`, and `gsli`, respectively. Each of the indices can be saved in a separate dictionary.

```
BGR_INDEX = {}
HSV_INDEX = {}
GSL_INDEX = {}
BGR_INDEX['cars/16_07_02_14_21_00_orig.png'] = bgri
HSV_INDEX['cars/16_07_02_14_21_00_orig.png'] = hsvi
GSL_INDEX['cars/16_07_02_14_21_00_orig.png'] = gsli
```

We can generalize the steps described above for a directory of images by implementing the function

```
def index_img_dir(imgdir):
   for imgp in generate_file_names(r'.+\.(jpg|png|JPG)', imgdir):
     print('indexing ' + imgp)
     index_img(imgp)
     print(imgp + ' indexed')
```

This function takes a directory of images, and then, through a call to the generator `generate_file_names`, indexes each image path in the directory. The function `generate_file_names` is a generator that takes a regular expression and generates the paths to files that match the regular expression. In the above example, the regular expression will match all files ending in `jpg`, `png`, or `JPG`. The function `index_img` computes the BGR index of an image saved in `imgp` and places the index in `BGR_INDEX` under the `imgp` key, computes the HSV index of the image and places it in `HSV_INDEX` under the `imgp` key, and computes the GSL index and stores it in `GSL_INDEX` under the `imgp` key. After all the indices are computed and saved in the three collection indices, i.e., i.e., `BGR_INDEX`, `HSV_INDEX`, and `GSL_INDEX`, the three collection indices, are pickled into the user specified files for future use. Here is the `__main__` we can do it with.

```
  ap = argparse.ArgumentParser()
  ap.add_argument('-imgdir', '--imgdir', required = True, help = 'image directory')
  ap.add_argument('-bgr', '--bgr', required = True, help = 'bgr index file')
  ap.add_argument('-hsv', '--hsv', required = True, help = 'hsv index file')
  ap.add_argument('-gsl', '--gsl', required = True, help = 'gsl index file')
  args = vars(ap.parse_args())

  if __name__ == '__main__':
    index_img_dir(args['imgdir'])
    with open(args['bgr'], 'wb') as bgrfile:
      pickle.dump(BGR_INDEX, bgrfile)
    with open(args['hsv'], 'wb') as hsvfile:
      pickle.dump(HSV_INDEX, hsvfile)
    with open(args['gsl'], 'wb') as gslfile:
      pickle.dump(GSL_INDEX, gslfile)
    print('indexing finished')
```

If we save our code in `image_index.py`, we can execute it as follows:

```
$ python image_index.py -imgdir car/ -bgr carbgr.pck -hsv carhsv.pck -gsl cargsl.pck
```

After the indexing is done, the files `carbgr.pck`, `carhsv.pck`, and `cargsl.pck` will contain the pickled dictionaries `BGR_INDEX`, `HSV_INDEX`, and `GSL_INDEX`, respectively.

4 Retrieving Images

OK, we've created and persisted image indices. How can we use them to retrieve images similar to a given one? Here is how. Given an input image I we 1) unpickle the pickled dictionaries `BGR_INDEX`, `HSV_INDEX`, and `GSL_INDEX`; 2) compute the BGR, HSV, and GSL indices of the input image I; 3) match the indices of the input image against the corresponsing indices in the unpickled dictionaries; and 4) return the paths to the three top matching images and the similarity scores, i.e., the path to the top BGR match, the path to the top HSV match, and the path to the top GSL match.

So, how do we actually compute the three matching scores? Let's continue with simple 2x3 images and suppose that our input image, I, has been transformed into the following three indexes: `bgri`, `hsvi`, and `gsli`.

```
bgri = [(b0, g0, r0)
        (b1, g1, r1)]

hsvi = [(h0, s0, v0),
        (h1, s1, v1)]

gsli = [gsl0, gsl1]
```

Recall that the values in the above three indexes are the means of the corresponding rows. For example, in `bgri`, the values in column 0 are the blue averges in rows 0 and 1 of I in the BGR space. The values in column 1 are the green averages in rows 0 and 1 of I in the BGR space. Finally, the averages in column 2 are the red averages in rows 0 and 1 of I in the BGR space.

In `hsvi`, the values in column 0 are the hue averges in rows 0 and 1 of I in the HSV space. The values in column 1 are the saturation averages in rows 0 and 1 of I in the HSV space. Finally, the averages in column 2 are the value averages in rows 0 and 1 of I in the HSV space.

In `gsli`, `gsl0` is the average of the grayscale values in row 0 of I in the GSL space and `gsl1` is the average of the grayscale values in row 1 of I in the GSL space.

Let's suppose that some image P is represented in the three unpickled indexes, i.e., `BGR_INDEX`, `HSV_INDEX`, and `GSL_INDEX`, as follows:

```
pbgr = [(pb0, pg0, pr0)
        (pb1, pg1, pr1)]

phsv = [(ph0, ps0, pv0),
        (ph1, ps1, pv1)]

pgsl = [pgsl0, pgsl1]
```

We compute the three matching scores between I and P, i.e., the scores in the BGR, HSV, and GSL spaces, column by column. Specifically, the BGR score is computed as follows:

$$blue\_sim = \frac{\sum_{i=0}^{1}(b_i \times pb_i)}{\sqrt{\sum_{i=0}^{2}(b_i^2)} \times \sqrt{\sum_{i=0}^{2}(pb_i^2)}} \tag{1}$$

$$green\_sim = \frac{\sum_{i=0}^{1}(g_i \times pg_i)}{\sqrt{\sum_{i=0}^{2}(g_i^2)} \times \sqrt{\sum_{i=0}^{2}(pg_i^2)}} \tag{2}$$

$$red\_sim = \frac{\sum_{i=0}^{1}(r_i \times pr_i)}{\sqrt{\sum_{i=0}^{2}(r_i^2)} \times \sqrt{\sum_{i=0}^{2}(rb_i^2)}} \tag{3}$$

Those of you who are familiar with linear algebra know that, by equation 1, `blue_sim` is the cosine similarity between the vectors `[b0, b1]` and `[pb0, pb1]`. By equation 2, `green_sim` is the cosine similarity between the vectors `[g0, g1]` and `[pg0, pg1]`. Finally, by equation 3, `red_sim` is the cosine similarity between the vectors `[r0, r1]` and `[pr0, pr1]`.

These equations have straightforward Py interpretations. Here is the Py interpreation of equation 1. The other two equations are interpreted similarly.

```
bgr_sim = (b0*pb0 + b1*pb1)/(sqrt(b0**2 + b1**2) * sqrt(pb0**2 + pb1**2))
```

The final BGR similarity score between I and P is computed as the average of `blue_sim`, `green_sim`, and `red_sim` and will be between 0 and 1.

The three hsv scores are computed in the same fashion.

$$hue\_sim = \frac{\sum_{i=0}^{1}(h_i \times ph_i)}{\sqrt{\sum_{i=0}^{2}(h_i^2)} \times \sqrt{\sum_{i=0}^{2}(ph_i^2)}} \tag{4}$$

$$saturation\_sim = \frac{\sum_{i=0}^{1}(s_i \times ps_i)}{\sqrt{\sum_{i=0}^{2}(s_i^2)} \times \sqrt{\sum_{i=0}^{2}(ps_i^2)}} \tag{5}$$

$$value\_sim = \frac{\sum_{i=0}^{1}(v_i \times pv_i)}{\sqrt{\sum_{i=0}^{2}(v_i^2)} \times \sqrt{\sum_{i=0}^{2}(pv_i^2)}} \tag{6}$$

The final HSV similarity score between I and P is computed as the average of `hue_sim`, `saturation_sim`, and `value_sim` and will be between 0 and 1.

The GSL similarity is computed as the cosine similarity between `gsl` and `pgsl`.

$$gsl\_sim = \frac{\sum_{i=0}^{1}(gsl_i \times pgsl_i)}{\sqrt{\sum_{i=0}^{2}(gsl_i^2)} \times \sqrt{\sum_{i=0}^{2}(pgsl_i^2)}} \tag{7}$$

In this fashion, we can compute the three similarity scores between the input image I and each image indexed in `BGR_INDEX`, `HSV_INDEX`, and `GSL_INDEX`. When all the scores are computed, the top one is selected for each space.

Suppose we implement the above image retrieval algorithm and save it in `image_retrieval.py`. Let's retrieve the three images most similar to `car_test/img01.png`.

```
$ python image_retrieval.py -i car_test/img01.png -bgr carbgr.pck -hsv carhsv.pck -gsl cargsl.pck
[('car/16_07_02_14_21_00_orig.png', 0.99999593095514727)]
[('car/16_07_02_14_21_00_orig.png', 0.99998668973015314)]
[('car/16_07_02_14_21_00_orig.png', 0.99999856308231527)]
```

The output prints the paths and the scores of the top matching images from the three indices. Figure 1 shows the four images displayed when the above command is executed. The window `Input` displays the original image. The window `BGR` displays the most similar image in the BGR index, i.e., `car/16_07_02_14_21_00_orig.png`. The window `HSV` displays the most similar image in the HSV index, i.e., `car/16_07_02_14_21_00_orig.png`. The window `GSL` displays the most similar image in the GSL image, i.e., `car/16_07_02_14_21_00_orig.png`.

Let's retrieve the images most similar to `car_test/img05.png`. The four images displayed in the windows are shown in 2.

```
$ python image_retrieval.py -i car_test/img05.png -bgr carbgr.pck -hsv carhsv.pck -gsl cargsl.pck
[('car/16_07_02_14_21_22_orig.png', 0.99836817514355269)]
[('car/16_07_02_14_21_24_orig.png', 0.99850466392576553)]
[('car/16_07_02_14_21_22_orig.png', 0.99841184648150316)]
```

Let's retrieve the images most similar to `car_test/img22.png`. The four images displayed in the windows are shown in 3.

Figure 1: Image img01.png and three images most similar to it.

Figure 2: Image img05.png and three images most similar to it.

```
$ python image_retrieval.py -i car_test/img22.png -bgr carbgr.pck -hsv carhsv.pck -gsl cargsl.pck
[('car/17_02_21_22_14_24_orig.png', 0.99826333103666121)]
[('car/17_02_21_22_14_14_orig.png', 0.99200445559785455)]
[('car/17_02_21_22_14_24_orig.png', 0.99832632604158333)]
```

5 What To Submit

Implement the functions in `image_index.py` and `image_retrieval.py` in Py2 and submit them via Canvas. Do not submit the image files. This is a Py2 assignment only, because the raspberry pi's images are configured to run with Py2 and OpenCV 3.0.0.

6 Additional Reading

For those of you who are mathematically inclined or just want to have some fun learning new stuff, here is an excellent wiki article on cosine similarity and its uses in various branches of math and science.

`https://en.wikipedia.org/wiki/Cosine_similarity`

Documentation on Py2 dictionaries is at the link below.

`https://docs.python.org/2/tutorial/datastructures.html`

Happy Hacking!

Figure 3: Image img22.png and three images most similar to it.