CSE 473 Computer Vision and Image Processing

Programming Assignment # 3:
Image Based Search Engine

Instructor Radhakrishna Dasari

Due Date: July 25th, 2016

Winnie Liang
wliang6@buffalo.edu
50038993

Image Based Search Engine

Introduction:

A dataset of 25 images is created with five landmarks located at University at Buffalo campus. This includes taken five different angle shots for each landmark: Clemens, CFA, Clock, Columns, Commons. The dataset is then inputted into an image based search engine program. The goal is that given a query image from an image from one of the five landmark categories, we can output the images that matches the same landmark category as the top 10 results.

Query Images

Clemens CFA Clock Columns Commons

Image Based Search Engine Concept:

The Image Based Search Engine is performed in the following manner:

We define the descriptor for the images like for example, the shape, texture or color. The image descriptor used is a color histogram which has red, green, blue channels. We will index the dataset by applying the descriptor to each image in the dataset to extract features. Indexing an image allows us to output a feature vector which is the numerical value of the image. With the feature vectors, we can compare the similarity of two images by using a distance metric such as the chi-squared distance. This is done between the query image and each individual images in the dataset which will then rank the images based on the similarity distance. Lastly, with the rankings, we can check the results to see if the corresponding images are ranked closest to the query images.

Programming approach:

The code is based off of Adrian Rosebrock's guide on building an image search engine in Python 2.7 and OpenCV 2.4.

1) Deciding 3D RGB Color Histogram as the image descriptor. There are 8 bins per red, green, blue channel. OpenCV stores feature vector as an (8, 8, 8) array and it will be flattened to (512,) to compute the similarity amongst the feature vectors. By making the histogram a constructor within a class, it allows each image to have the same parameters. OpenCV functions such as cv2.calcHist and cv2.normalize are called to create the 3D histogram and then normalizing it before flattening.
2) Applying the 3D RGB Color Histogram to each dataset image. Each image would result in an extracted histogram, then they are stored into a dictionary. The key is identified to be the image filename and the value is the corresponding histogram turned into feature vectors. The dictionary is written and saved to a pickle file. Glob function is also used to retrieve image paths from the command arguments inputted. OpenCV function cv2.imread is used to retrieve images from a path location. -- Indexing is complete.
3) Searching the indexed image dataset (pickle file). A Searcher class and a constructor is defined with the index (the pickle file) used as a parameter. A for-loop is constructed to loop over the index dictionary with the filenames and the feature vectors. To compare the histograms, chi-squared distance is used to calculate the distance between them before storing the results into a dictionary. The results are then sorted from the shortest distance (most relevant to query image) to longest distance.
4) Run the search and display results. Load the pickle file and the Searcher class. The search treats the current image as the query. Using a for loop, the top 10 results are written into two montage images which are empty matrices to hold the ranking image results. The chi-squared distances are also displayed.

Results and Factors:

As part of my results, most of the images from the dataset appeared in the top 10 results which corresponded to the query image. The accuracies range from 80% - 100% for all five images from the dataset appearing in the top 10 results. This means only 4 to a total of all 5 images appearing. The returned value next to the image filenames are the chi-squared distances of the image feature vectors in comparison to the query image. The shortest distance would appear at the top of the rank leading to the longest distance appearing at the bottom of the rank.

Clemens - 80% (4 out of all 5 images appearing in the top 10 rank out of 25)
CFA - 100% (5 out of all 5 images appearing in the top 10 rank out of 25)
Clock - 100% (5 out of all 5 images appearing in the top 10 rank out of 25)
Columns - 80% (4 out of all 5 images appearing in the top 10 rank out of 25)
Commons - 80% (4 out of all 5 images appearing in the top 10 rank out of 25)

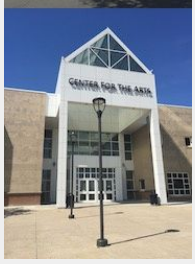
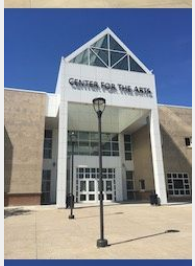
There are many factors that affected the results from running the images through the image based search engine. Image classification is still an open problem since the color histogram cannot provide enough information to provide an efficient classifier. One of the affecting factors can be the position or angle of which the images are taken from. The different standpoint can change the orientation of the featured landmark making it inaccurate when detecting the color bins for the histogram. Another factor can be due to the sunlight glare on the images from the series of light reflection bouncing on the camera lenses. Flares appear more noticeably when using a large aperture. Occlusions that blocks off a partial view from the landmark can also make it difficult to detect the landmark like in the clock image. A better implementation that can cover texture or shape can be done by using a SIFT descriptor instead of a color histogram. The noisy examples can be fixed due to SIFT being rotation invariant and scale invariant.

Rank Results 1-5, 6-10

Clemens	CFA	Clock
		



Rank Results 1-5, 6-10

Columns	Commons
	
         	         

Output from console:

The output displays the ranking of the image filenames based on their corresponding chi-squared distances. Also, the accuracy is simply calculated manually to see how many of the same matching image category appeared in the top 10 ranking. I also averaged the chi-squared distances of the images that matched the query.

CLEMENS

query: queries/Clemens5-query.jpg

1. Clemens4.jpg : 0.439
2. Clemens1.jpg : 0.468
3. Clemens8.jpg : 0.845
4. Clemens6.jpg : 0.952
5. Columns4.jpg : 0.959
6. Columns6.jpg : 1.033
7. Columns10.jpg : 1.241
8. CFA5.jpg : 1.244
9. Columns8.jpg : 1.254
10. Columns7.jpg : 1.254

Accuracy: ⅘ * 100 = 80%

Average chi-squared distances of the query matched images: 0.676

CFA

query: queries/CFA3-query.jpg

1. CFA4.jpg : 0.280
2. CFA2.jpg : 0.442
3. CFA1.jpg : 0.887
4. CFA5.jpg : 0.941
5. CFA7.jpg : 1.000
6. Columns10.jpg : 1.230
7. Columns6.jpg : 1.248
8. Columns8.jpg : 1.306
9. Commons5.jpg : 1.367
10. Columns7.jpg : 1.400

Accuracy: 5/5 * 100= 100%

Average chi-squared distances of the query matched images: 0.71

CLOCK

query: queries/Clock7-query.jpg

1. Clock8.jpg : 0.414
2. Clock9.jpg : 0.926
3. Clock5.jpg : 1.449
4. Commons7.jpg : 1.512
5. Clock4.jpg : 1.581
6. Clock6.jpg : 1.640
7. CFA1.jpg : 1.978
8. CFA4.jpg : 2.006
9. CFA2.jpg : 2.106

10. Columns6.jpg : 2.113

Accuracy: 5/5 * 100 = 100%

Average chi-squared distances of the query matched images: 1.202

COLUMNS

query: queries/Columns9-query.jpg

1. Columns10.jpg : 0.153

2. Columns8.jpg : 0.454

3. Columns7.jpg : 0.627

4. Columns6.jpg : 1.089

5. Commons2.jpg : 1.133
6. Commons5.jpg : 1.134
7. Commons6.jpg : 1.146
8. Commons1.jpg : 1.167
9. Clemens4.jpg : 1.229
10. CFA2.jpg : 1.237

Accuracy: ⅘ * 100 = 80%

Average chi-squared distances of the query matched images: 0.58075

COMMONS

query: queries/Commons4-query.jpg

1. Commons5.jpg : 0.045

2. Commons6.jpg : 0.087

3. Commons2.jpg : 0.380

4. Commons1.jpg : 0.738

5. Columns8.jpg : 0.802
6. Columns10.jpg : 1.069
7. CFA2.jpg : 1.244
8. CFA1.jpg : 1.259
9. CFA7.jpg : 1.447
10. CFA4.jpg : 1.453

Accuracy: ⅘ * 100 = 80%

Average chi-squared distances of the query matched images: 0.3125