Malavika Pande
Mp3564

Note: I used a CPU as my hardware. All references are cited and documented in my program color_retrieval.py and in this file. I explain my decisions in this file and my code as well. My program can be executed with python3 color_retrieval.py. It writes to 4 csv files the query number, top 3 and worst target numbers and their score, and the total score.

Step 0

MyPreferences.txt is a text file located in the mp3564_HW2 directory.

Step 1

Decisions and Documentation

I chose to use OpenCV's calcHist method. For example, to calculate the histogram of a query image, I execute the following line:

```
hist_query = cv2.calcHist([query_im], [0,1,2],None, [16,16,16], [0,256,0,256,0,256])
```

The variable query_im refers to the query image. I create a 3D histogram by passing in [0,1,2] for the channels. My histogram size, the number of bins, is 16 per channel. This means there are 16^3 = 4096 possible bins a pixel could be in. I chose 16 because I wanted to create a frequency distribution of equal length: the bin size is 16 because 256/16 = 16.

This link: https://stats.stackexchange.com/questions/798/calculating-optimal-number-of-bins-in-a-histogram was helpful in understanding optimal number of bins. Also, I knew I needed to choose a number of bins such that the images are distinguished from one another but also contain pixels that intersect with one another. I tried out 8 and 20 as well, but both yielded total scores less than that of 16.

This link: https://stackoverflow.com/questions/46498041/is-the-opencv-3d-histogram-3-axis-histogram that Professor had referenced on Piazza was very helpful in understanding 3D histograms.

I have used the same format for *all steps*. In the column Query I have included the query image and identifying number. In columns T1, T2, T3, and T40 I also include the target image and its identifying number. In columns T1 Count, T2 Count, T3 Count, and T40 count, I include the *Crowd(q,t)* count.

System vs. Crowd Preferences

Query	T1	T1 Count	T2	T2 Count	T3	T3 Count	T40	T40 Count	Score
1	10	46	16	4	3		15	0	154
2	39	7	21	13	34		15	83	45
3	4	160	38	0	24		15	0	161
4	3	164	8	61	1		15	0	269
5	6	176	**7**	69	23		12	0	246
6	11	37	5	166	7		22	0	277
7	9	122	11	34	40		37	0	163
8	3	126	19	0	24		15	0	126
9	7	133	11	22	14		37	0	170
10	1	72	16	137	3		15	0	227
11	9	48	7	79	6		37	0	260
12	2	20	9	37	14		37	0	188
13	14	119	11	0	9		15	7	141
14	9	38	39	15	7		26	0	76

15	7	36	6	127	11		36	0	170
16	1	18	10	103	3		15	90	136
17	28	0	33	1	25		12	0	1
18	35	39	25	0	28		15	0	39
19	38	36	24	107	36		15	0	170
20	23	86	40	60	25		12	0	147
21	2	0	34	2	39		15	0	39
22	34	8	36	50	19		15	0	74
23	40	13	28	5	20		12	0	64
24	36	15	32	29	19		15	0	166
25	28	104	17	7	40		15	3	120
26	6	0	7	1	9		37	0	1
27	30	15	38	0	3		15	0	26
28	33	62	17	4	25		12	0	199

29	30	175	34	7	19		15	0	184
30	29	171	34	5	32		15	0	179
31	27	86	1	0	32		15	0	243
32	24	36	33	16	34		15	0	52
33	28	69	17	20	25		15	0	126
34	24	1	33	13	22		15	0	121
35	28	2	23	16	18		12	0	128
36	24	9	38	61	19		15	0	100
37	24	45	32	2	36		15	0	190
38	19	59	36	99	3		15	0	161
39	34	10	2	3	25		15	2	13
40	23	12	25	5	28		12	2	30

Grand Total Sum Score: 5382

System vs. Personal Preferences

Grand Total Intersection Score: 57

Discussion

For crowd-based accuracy, the system scored especially high and low on the following queries. I have shown the top 3 scores and the worst 3 scores:

High		**Low**	
Query	**Score**	**Query**	**Score**
6	277	17	1
4	269	26	1
11	260	39	13

The query with the highest score was query 6. Queries 17 and 26 had the lowest scores with a score of 1. I was not surprised that 26 scored low. 26 depicts a sunset with a wide distribution of colors. Queries 6, 4, and 11, notably, have less variety of colors. There is very little black background in all these images.

In terms of user satisfaction, the system scored high and low on the following queries:

High		**Low**	
Query	**Score**	**Query**	**Score**
4	3	15	0
1	2	16	1
2	2	17	1

As shown above, query 4 matched all my personal preferences. Unsurprisingly, query 4, one of the top scoring queries against the crowd data, was a top scoring query against the personal preference data. Query 17 scored the lowest in both categories. Query 15 failed to match any of my preferences. Overall, the machine performs well on histograms that are *not* uniformly distributed.

It is also important to note that a huge drawback of color-based histograms is that they do not consider the spatial information of pixels. This can result similar color distributions for different images.

Step 2:

Decisions and Documentation

I used OpenCV’s method to convert the image to a gray scale image. I used it because it is computationally efficient, and documented in OpenCV. This OpenCV tutorial: https://theailearner.com/tag/cv2-laplacian/ was very helpful for this step. I first smooth the image with a

Gaussian filter to reduce noise. Then, I convert the image to a grayscale image with cv2.cvtColor() and set the second parameter to cv2.COLOR_BGR2GRAY. Then, I find the found zero crossings with the cv2.Laplacian() method, setting depth equal to cv2.CV_16S and kernel size equal to 3. The two-step process is formally known as the Laplacian of Gaussian (LoG) operation.

I use CV_16S, a 16 bit signed short int, so I can save signed integers after the Laplacian filter is applied. Finally, I convert back to unsigned 8 bit so I can calculate the histogram of the image.

System vs. Crowd Preferences

Query	**T1**	**T1 Count**	**T2**	**T2 Count**	**T3**	**T3 Count**	**T40**	**T40 Count**	**Score**
1	3	104	8	150	10	46	26	0	300
2	6	21	14	19	12	14	26	26	54
3	1	69	8	92	10	15	26	0	176
4	7	5	9	0	16	19	26	0	24
5	31	0	15	30	7	69	26	0	99
6	2	1	14	0	20	0	26	0	1
7	16	2	9	122	13	2	26	0	126
8	1	151	3	126	10	12	26	0	289
9	16	8	11	22	7	133	26	0	163

10	1	72	3	18	8	17	26	0	107
11	9	48	16	6	7	79	26	0	133
12	14	131	15	10	31	0	26	0	141
13	7	11	16	5	9	22	26	0	38
14	12	116	15	48	2	9	26	0	173
15	31	0	5	42	12	0	26	0	42
16	9	18	7	4	11	3	26	0	25
17	38	0	25	0	24	3	10	0	3
18	23	52	38	0	32	9	10	0	61
19	24	107	38	36	39	0	26	1	143
20	6	0	22	51	2	0	10	0	51
21	29	25	34	2	25	0	10	0	27
22	35	1	34	8	29	64	10	0	73
23	18	67	38	0	17	49	26	1	116

24	19	122	38	13	17	6	26	2	141
25	17	7	28	104	24	0	10	0	111
26	30	0	35	25	29	0	10	0	25
27	4	2	40	0	9	0	26	0	2
28	25	133	40	23	24	1	10	0	157
29	21	38	34	7	22	63	10	0	108
30	29	171	21	40	35	1	10	0	212
31	5	0	15	0	7	0	26	2	0
32	34	0	38	1	17	15	10	0	16
33	32	15	18	22	23	2	10	0	39
34	21	36	29	48	22	107	10	0	191
35	22	4	29	1	37	4	10	0	9
36	29	5	25	3	22	41	10	0	49
37	35	20	36	143	22	9	10	1	172

38 	24 	27	17 	0	19 	59	26 	2	86
39 	19 	5	38 	60	40 	2	26 	0	67
40 	24 	0	28 	13	39 	13	26 	4	26

Grand Total Sum Score: 3776

System vs. User Preferences

Grand Total Intersection Score: 40

Discussion

For crowd-based accuracy, the system scored especially high and low on the following queries. I have shown the top 3 scores and the worst 3 scores:

High		**Low**	
Query	**Score**	**Query**	**Score**
1	300	31	0
8	289	6	1
30	212	27	2

Queries 1 and 8 have distinctively “rough” textures whereas query 30 has a very “smooth” texture. These queries may have performed well because they were easier to distinguish. Unlike the rest of the images, which depict fruits and vegetables for the most part, queries 31 and 27 depict abstract images. It is difficult for the system to classify the texture of these types of images because of how complex they are—they may not be rough *or* smooth.

In terms of user satisfaction, the system scored high and low on the following queries:

High		**Low**	
Query	**Score**	**Query**	**Score**
1	2	4	0
3	2	6	0
5	2	27	0

None of the queries matched all my personal preferences. Query 1 performed the best overall, and as expected (based on crowd performance scores) queries 6 and 27 performed the worst. The crowd performance results were somewhat similar to the user satisfaction results.

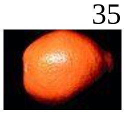
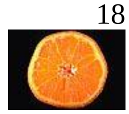
Step 3:

Decisions and Documentation

I used OpenCV cv.threshold() function. For every pixel, the same threshold value—what we determine to be "black— is applied. If a pixel is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value. I originally chose the threshold value to be 127.

In attempt to further optimize the threshold, I used Otsu Binarization. Interestingly, this increased my grand total score from 2853 to 3769. Otsu Binarization algorithm finds a threshold value that minimizes the weighted within-class variance given by the relation $\sigma^2_w(t)=q_1(t)\sigma^2_1(t) + q_2(t)\sigma^2_2(t)$. It finds a value of *t* such that the variances are minimal.

System vs. Crowd Preferences

Query	**T1**	**T1 Count**	**T2**	**T2 Count**	**T3**	**T3 Count**	**T40**	**T40 Count**	**Score**
1 	20 	0	29 	0	8 	150	26 	0	150
2 	26 	26	36 	0	37 	8	8 	1	34
3 	35 	0	39 	0	18 	0	7 	0	0
4 	29 	0	22 	0	39 	0	25 	0	0
5 	32 	0	31 	0	6 	176	14 	0	176
6 	32 	0	28 	0	31 	0	14 	0	0
7 	25 	0	6 	103	34 	0	13 	2	103

8	1	151	20	0	29	0	26	0	151
9	28	0	6	87	5	20	14	15	107
10	25	0	28	0	30	0	12	4	0
11	32	0	28	1	6	133	14	0	134
12	14	131	29	0	20	0	10	7	131
13	18	0	22	2	29	5	7	11	7
14	12	116	18	0	20	0	25	0	116
15	13	0	20	0	18	0	6	127	0
16	4	22	15	90	26	0	6	48	112
17	38	0	18	157	22	3	2	0	160
18	17	134	29	0	22	3	5	0	137
19	38	36	18	60	17	25	5	0	121
20	29	0	38	0	18	17	5	1	17
21	30	3	40	0	17	52	16	0	55

22	29	64	39	24	18	12	5	0	100
23	17	49	18	67	38	0	2	0	116
24	23	112	35	2	21	6	2	0	120
25	37	0	26	156	33	43	14	0	199
26	2	10	25	158	36	2	8	0	170
27	33	7	37	14	32	121	14	0	142
28	32	0	33	62	6	1	14	0	63
29	22	63	38	9	18	2	5	0	74
30	21	40	39	19	37	1	16	0	60
31	32	157	6	0	28	7	14	0	164
32	31	111	28	2	6	0	14	0	113
33	40	45	30	9	21	24	14	0	78
34	40	12	37	21	30	18	14	0	51
35	39	21	21	20	18	110	7	0	151

36 	37 	100	30 	2	35 	64	7 	0	166
37 	30 	3	36 	143	21 	2	16 	2	148
38 	17 	0	29 	19	18 	3	2 	0	22
39 	35 	36	22 	57	18 	8	7 	1	101
40 	21 	7	30 	0	39 	13	16 	3	20

Grand Total Sum Score: 3769

System vs. User Preferences

Grand Total Intersection Score: 25

Discussion

For crowd-based accuracy, the system scored especially high and low on the following queries. I have shown the top 3 scores and the worst 3 scores:

High		**Low**	
Query	**Score**	**Query**	**Score**
25	199	3	0
5	176	4	0
26	170	6	0

I thought it was fascinating how 25 performed the best. This might be because the shapes in this image are easily distinguishable from the rest of the images; for the most part, the rest of the images are singular objects.

In terms of user satisfaction, the system scored high and low on the following queries:

High	**Low**

Query	Score	Query	Score
19	2	1	0
5	1	2	0
7	1	3	0

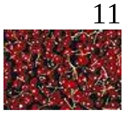
Queries 1, 2, and 3 failed to match any of my personal preferences.

Step 4

Decisions and Documentation

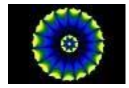
I chose my simplex vector based on each type of distance's performance against the Crowd.txt scores. The grand total scores ranked highest to lowest were color, with a score of 5382, texture, with a score of 3776, and shape, with a score of 3769. I found tested different values of a, b, c and ultimately found that a = 0.75, b = 0.125, and c = 0.125 yielded the optimal result. The simplex vector is thus (0.75, 0.125, 0.125). I set b and c the same because they had similar grand total scores, differing only by 7. I set a the highest because it had the highest grand total score by a wide margin, by at least 1606.

System vs. Crowd Preferences

Query	T1	T1 Count	T2	T2 Count	T3	T3 Count	T40	T40 Count	Score
1 	10 	46	3 	104	16 	4	26 	0	300
2 	39 	7	21 	13	25 	4	26 	26	40
3 	4 	160	38 	0	8 	92	26 	0	321
4 	3 	164	8 	61	1 	44	26 	0	269
5 	6 	176	7 	69	23 	1	26 	0	270
6 	11 	37	5 	166	7 	74	10 	0	277

7	9	122	11	34	40	7	26	0	192
8	3	126	19	0	1	151	26	0	289
9	7	133	11	22	6	87	26	0	175
10	1	72	3	18	16	137	26	0	227
11	9	48	7	79	6	133	26	0	181
12	2	20	14	131	9	37	26	0	156
13	14	119	11	0	9	22	26	0	33
14	9	38	39	15	13	82	26	0	169
15	7	36	11	7	6	127	26	0	205
16	10	103	1	18	3	15	26	0	143
17	28	0	33	1	25	0	11	0	1
18	35	39	17	134	23	52	26	2	188
19	38	36	24	107	36	27	26	1	170
20	23	86	40	60	17	17	11	0	148

21	34	2	39	37	29	25	15	0	27
22	34	8	36	50	19	16	11	0	59
23	40	13	28	5	20	46	26	1	129
24	36	15	19	122	32	29	26	2	150
25	28	104	17	7	40	9	16	3	120
26	6	0	40	10	33	35	10	0	102
27	38	0	30	15	3	11	26	0	13
28	33	62	25	133	17	4	13	0	160
29	30	175	38	9	34	7	11	0	199
30	29	171	34	5	37	1	11	0	200
31	27	86	32	157	5	0	26	2	86
32	24	36	33	16	34	0	11	0	52
33	28	69	17	20	34	9	16	0	104
34	22	107	24	1	33	13	11	0	114

35 	28 	2	18 	110	23 	16	1 	0	6
36 	24 	9	38 	61	19 	30	11 	0	111
37 	24 	45	36 	143	32 	2	11 	0	190
38 	19 	59	36 	99	24 	27	26 	2	185
39 	34 	10	22 	57	2 	3	26 	0	65
40 	23 	12	28 	13	25 	5	26 	4	30

Grand Total Sum Score: 6063

System vs. User Preferences

Grand Total Intersection Score: 61

Discussion

For crowd-based accuracy, the system scored especially high and low on the following queries. I have shown the top 3 scores and the worst 3 scores:

High		**Low**	
Query	**Score**	**Query**	**Score**
6	277	17	1
8	277	2	24
4	269	27	26

In terms of user satisfaction, the system scored high and low on the following queries:

High	**Low**

Query	Score	Query	Score
4	3	15	0
10	3	26	0
11	3	32	0

Step 5

Crowd-based performance

The grand total score is calculated by adding up the Crowd(q,t) scores for the top 3 target images per query image. The higher the score, the closer aligned my program is with crowd preferences.
So, the upper bound would be the sum of the 3 highest scoring target images per query image.

In my program, in my function get_upper_bound() I calculate the upper bound by retrieving the top 3 target images with the highest *Crowd(q,t)* counts per query image. Then, I sum the sum of the top 3 scores across all 40 rows.

a. The actual upper bound is therefore **9853**.

b. My final system grand total score was 6063. So, it came as close to 6063/9853 * 100 = **61.5 %.**

c. My personal preference grand intersection value was 61**.** So, it came as close to 61/120 = 61/120 = **51%.**

User-based performance

I effectively replace Crowd.txt and replace it with 40x40 matrix in my create_my_crowd() function. I redo Step 4 entirely except I score it against this new matrix.

Through trial and error, I tested various values for a, b, c to yield a maximum result. The new weighting vector is the same as before: a = 0.75, b = 0.125, c = 0.125. It's interesting that the values that I used for step 4 generated the maximum result. The total intersection score is 61, and 61/120 = **51%.** There is a **0%** change in difference from step 4.

References

https://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/laplace_operator/laplace_operator.html
https://stackoverflow.com/questions/46498041/is-the-opencv-3d-histogram-3-axis-histogram
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/laplace_operator/laplace_operator.html

https://www.pyimagesearch.com/2014/01/22/clever-girl-a-guide-to-utilizing-color-histograms-for-computer-vision-and-image-search-engines/
https://www.pyimagesearch.com/2014/07/14/3-ways-compare-histograms-using-opencv-python/