COMS 4753

Assignment 2

Daisy Ye

yy3131

Step 0: myPreferences.txt

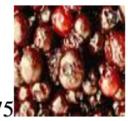
01 08 04 03
02 14 34 39
03 04 08 01
04 03 08 01
05 06 07 11
06 15 07 09
07 09 06 15
08 01 03 04
09 06 15 05
10 16 08 01
11 05 06 15
12 14 39 13
13 14 12 09
14 13 12 09
15 06 05 09
16 10 03 04
17 18 23 31
18 17 23 32
19 24 28 22
20 23 22 40
21 22 24 19
22 21 20 34
23 20 24 18
24 19 23 21
25 26 33 28
26 25 33 28
27 31 32 19
28 25 26 20
29 30 39 22
30 29 39 22
31 32 17 27
32 24 31 19
33 25 26 28
34 21 02 22
35 18 36 39
36 38 19 35
37 08 01 36
38 36 39 35
39 21 22 38

40 20 35 17
(Note: This is also in the code)

Step 1: Color

1. System vs. Crowd Preferences

Overall Score: 11561


12

09, crowd(12, 09): 75

02, crowd(12, 02): 30

14, crowd(12, 14): 226 Score: 331

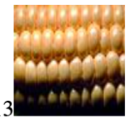

13

14, crowd(13, 14): 201

10, crowd(13, 10): 47

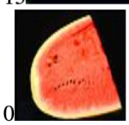
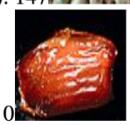
39, crowd(13, 39): 10 Score: 258


14

39, crowd(14, 39): 30

02, crowd(14, 02): 13

13, crowd(14, 13): 147 Score: 190


15

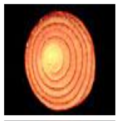
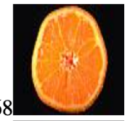
07, crowd(15, 07): 72

20, crowd(15, 20): 0

40, crowd(15, 40): 0 Score: 72

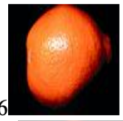

16

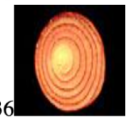
10, crowd(16, 10): 240

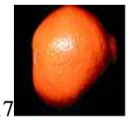
01, crowd(16, 01): 39

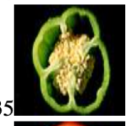
03, crowd(16, 03): 24 Score: 303


17

18, crowd(17, 18): 268

25, crowd(17, 25): 8

35, crowd(17, 35): 66 Score: 342

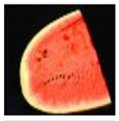
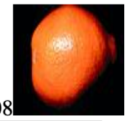
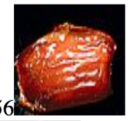
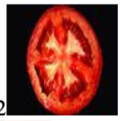

18

17, crowd(18, 17): 236

25, crowd(18, 25): 0

35, crowd(18, 35): 117 Score: 353

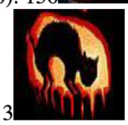

19

24, crowd(19, 24): 235

38, crowd(19, 38): 76

36, crowd(19, 36): 60 Score: 371


20

35, crowd(20, 35): 108

40, crowd(20, 40): 156

23, crowd(20, 23): 142 Score: 406


21

34, crowd(21, 34): 3

28, crowd(21, 28): 3

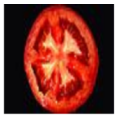
25, crowd(21, 25): 0 Score: 6

22

34, crowd(22, 34): 20

38, crowd(22, 38): 4

36, crowd(22, 36): 84 Score: 108


23 **40,** crowd(23, 40): 16 **35,** crowd(23, 35): 53 **28,** crowd(23, 28): 8 Score: 77


24 **36,** crowd(24, 36): 31 **38,** crowd(24, 38): 46 **19,** crowd(24, 19): 222 Score: 299


25 **17,** crowd(25, 17): 16 **18,** crowd(25, 18): 7 **28,** crowd(25, 28): 178 Score: 201


26 **07,** crowd(26, 07): 0 **40,** crowd(26, 40): 16 **11,** crowd(26, 11): 0 Score: 16

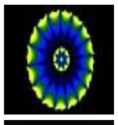

27 **32,** crowd(27, 32): 207 **33,** crowd(27, 33): 12 **24,** crowd(27, 24): 32 Score: 251


28 **33,** crowd(28, 33): 149 **21,** crowd(28, 21): 3 **25,** crowd(28, 25): 228 Score: 380

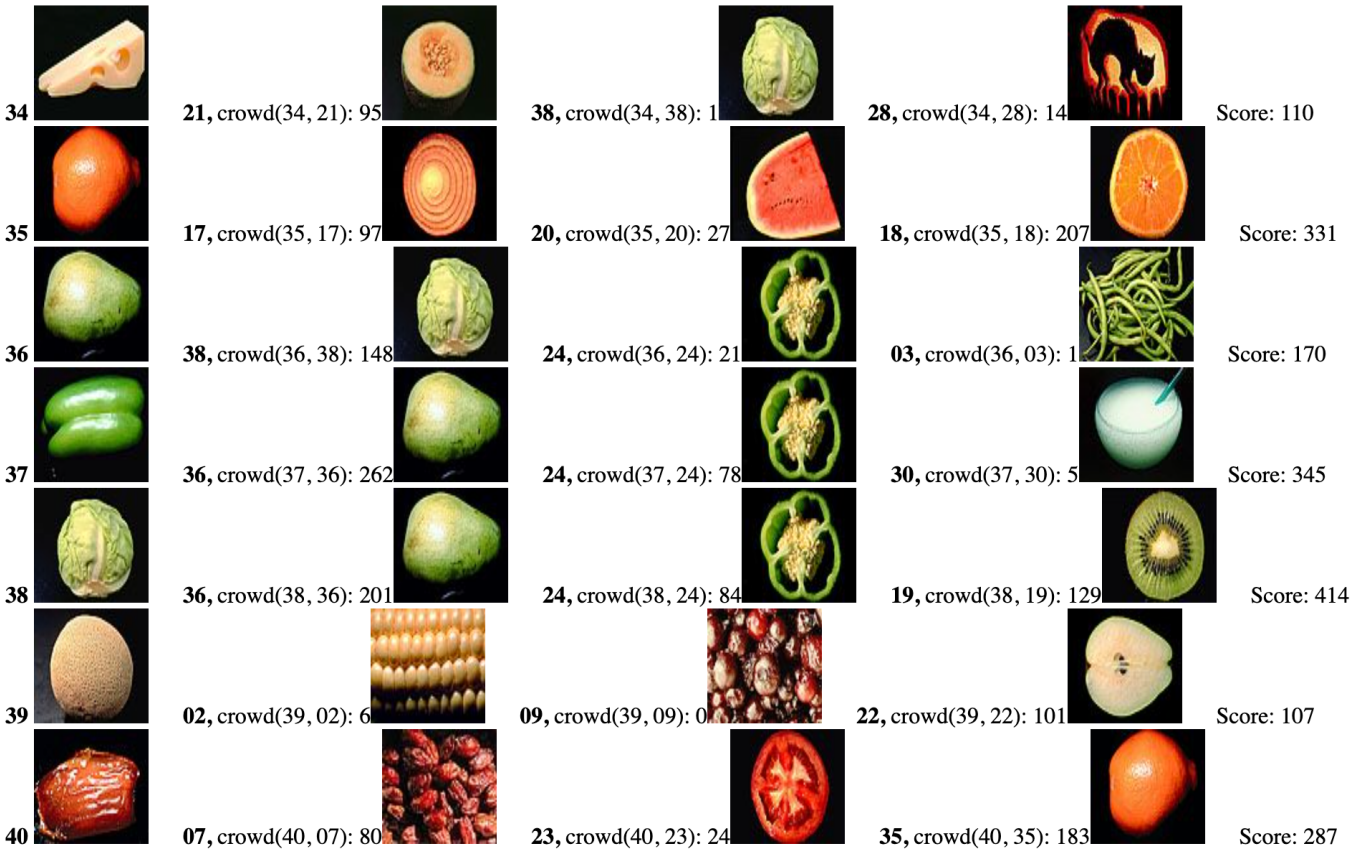

29 **30,** crowd(29, 30): 311 **38,** crowd(29, 38): 9 **36,** crowd(29, 36): 18 Score: 338


30 **29,** crowd(30, 29): 305 **36,** crowd(30, 36): 29 **38,** crowd(30, 38): 10 Score: 344


31 **28,** crowd(31, 28): 16 **33,** crowd(31, 33): 25 **32,** crowd(31, 32): 281 Score: 322


32 **27,** crowd(32, 27): 110 **24,** crowd(32, 24): 71 **28,** crowd(32, 28): 10 Score: 191


33 **28,** crowd(33, 28): 149 **27,** crowd(33, 27): 2 **40,** crowd(33, 40): 86 Score: 237



Overall Score: 11561

2. System vs. Personal Preferences
    a. The intersection value is 44
3. Justification

I will first explain the algorithm before going into my bin partition choice. I used the OpenCV calcHist function that takes an image and creates its histogram given the number of channels and the bin partition for each channel. I will discuss the specifics for such bin partition later. Then, I use the histograms to calculate the L1 norm between two images. Afterwards, I choose the three images with the lowest L1 norm distance and choose those as the final result.

My final choice of how to partition each channel is [2, 6, 4], which means I separated blue into 2 bins, green into 6 bins, and red into 4 bins. I arrived at this result because people have a harder time seeing differences in blue and they can see differences in green easily, and red is somewhere in the middle. However, I did some exploration before finally figuring out this combination.

The first combination was [3, 2, 2], which means 3 bins for blue, 2 bins for green, and 2 bins for red. The overall score of this result is 7548. I learned from this combination that the values for the channels are too low. For example, for green and red, there are only 4 colors for each channel. And then I tried an even partition of the bins [4, 4, 4], which means 4 bins for each channel. The overall score of the result rose to 9655. The third combination I tried was [2, 7, 4],

which means 2 bins for blue, 7 bins for green, and 2 bins for red. The overall score of this result is 11238, which is a lot higher than the last one, proving the fact that people have an easier time seeing differences in green.

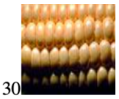
Before I arrived at my final result, I also realized that the number of pixels in the picture is about 5000. After doing some math, 2 to the 12th power is slightly over 4000, which makes it close to 5000. Therefore, my goal became to find a combination with the sum of 12. I modified the [2, 7, 4] by subtracting a bin from blue, and surprisingly arrived at a better overall score of 11561.

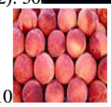
I believed that for this step, my algorithm worked well. 11561/25200 gives the percentage of 46%. Given how this is only one distance and the highest score possible is 72%, I think it is a pretty reliable algorithm.

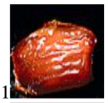
Step 2: Texture

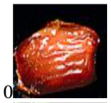
1. System vs. Crowd Preferences

Overall Score: 7182

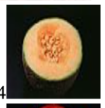

12 
14, crowd(12, 14): 226 
15, crowd(12, 15): 12 
02, crowd(12, 02): 30 Score: 268

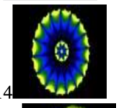

13 
16, crowd(13, 16): 7 
04, crowd(13, 04): 6 
15, crowd(13, 15): 10 Score: 23

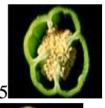

14 
15, crowd(14, 15): 92 
40, crowd(14, 40): 1 
12, crowd(14, 12): 210 Score: 303

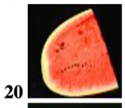
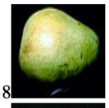
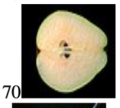

15 
14, crowd(15, 14): 29 
40, crowd(15, 40): 0 
13, crowd(15, 13): 1 Score: 30


16 
13, crowd(16, 13): 4 
04, crowd(16, 04): 43 
07, crowd(16, 07): 7 Score: 54


17 
38, crowd(17, 38): 3 
34, crowd(17, 34): 0 
21, crowd(17, 21): 104 Score: 107

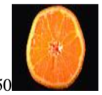
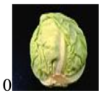

18 
32, crowd(18, 32): 14 
38, crowd(18, 38): 0 
23, crowd(18, 23): 85 Score: 99


19 
24, crowd(19, 24): 235 
39, crowd(19, 39): 1 
38, crowd(19, 38): 76 Score: 312

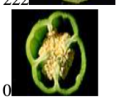
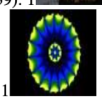

20 
36, crowd(20, 36): 8 
28, crowd(20, 28): 20 
22, crowd(20, 22): 70 Score: 98

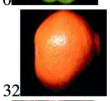
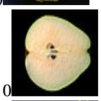
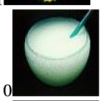

21 
34, crowd(21, 34): 3 
29, crowd(21, 29): 27 
30, crowd(21, 30): 5 Score: 35

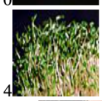
22
35, crowd(22, 35): 14
36, crowd(22, 36): 84
37, crowd(22, 37): 2 Score: 100

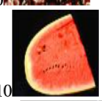
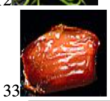

23 
18, crowd(23, 18): 150 
39, crowd(23, 39): 2 
38, crowd(23, 38): 0 Score: 152


24 
19, crowd(24, 19): 222 
38, crowd(24, 38): 46 
39, crowd(24, 39): 1 Score: 269

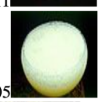

25 
24, crowd(25, 24): 0 
38, crowd(25, 38): 0 
32, crowd(25, 32): 1 Score: 1

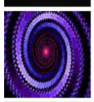

26 
35, crowd(26, 35): 32 
22, crowd(26, 22): 0 
30, crowd(26, 30): 0 Score: 32

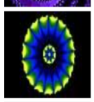
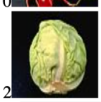
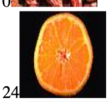

27 
09, crowd(27, 09): 0 
03, crowd(27, 03): 12 
01, crowd(27, 01): 4 Score: 16


28 
20, crowd(28, 20): 10 
40, crowd(28, 40): 33 
25, crowd(28, 25): 228 Score: 271

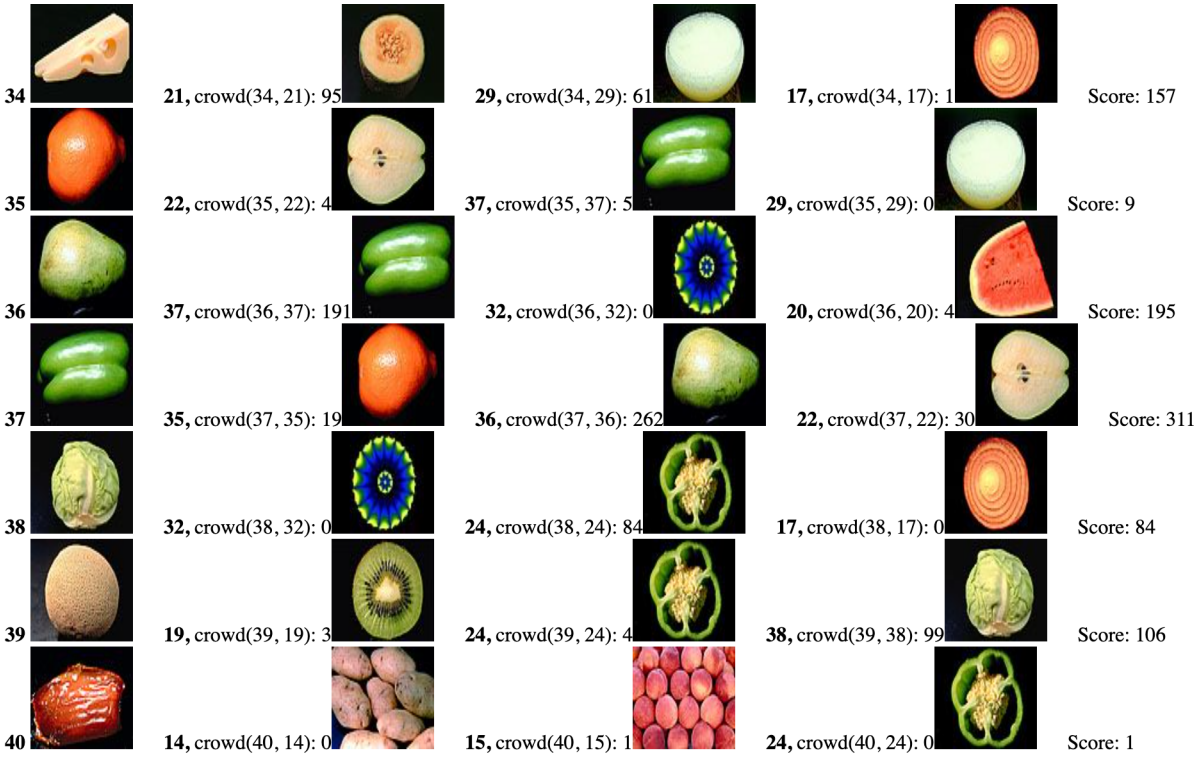

29 
30, crowd(29, 30): 311 
21, crowd(29, 21): 63 
34, crowd(29, 34): 10 Score: 384


30 
29, crowd(30, 29): 305 
21, crowd(30, 21): 59 
34, crowd(30, 34): 10 Score: 374


31 
05, crowd(31, 05): 0 
07, crowd(31, 07): 0 
09, crowd(31, 09): 1 Score: 1


32 
38, crowd(32, 38): 2 
18, crowd(32, 18): 24 
29, crowd(32, 29): 3 Score: 29

33
17, crowd(33, 17): 28
32, crowd(33, 32): 11
34, crowd(33, 34): 10 Score: 49



Overall Score: 7182

2. System vs. Personal Preferences
    a. The intersection value is 28
3. Justification

I will first explain the algorithm for texture and then justify the bin partition that I used. First, I converted all images from their original form to 8 byte grayscale form. I did this by dividing each RGB value of a pixel by 3 and rounding it to the nearest integer. The resulting grayscale image looks like this:


(image 21 – gray)

Then I used the built-in OpenCV laplacian method to convert the grayscale image to its laplacian form. I chose the OpenCV laplacian method instead of doing the usual calculation because I tried both methods and they led to the same result. And the OpenCV laplacian method is much cleaner and faster than my method. However, one can find the function that does the

laplacian conversion in my code and it's named calcLap(gray).

 (image 21 – laplacian)

Afterwards, I call the OpenCV function convertScaleAbs to convert the result to all non-negative values and to make sure it is 8 bytes. Lastly, I call calcHist and use the laplacian images to produce a histogram that is then used to calculate texture normalized distance.

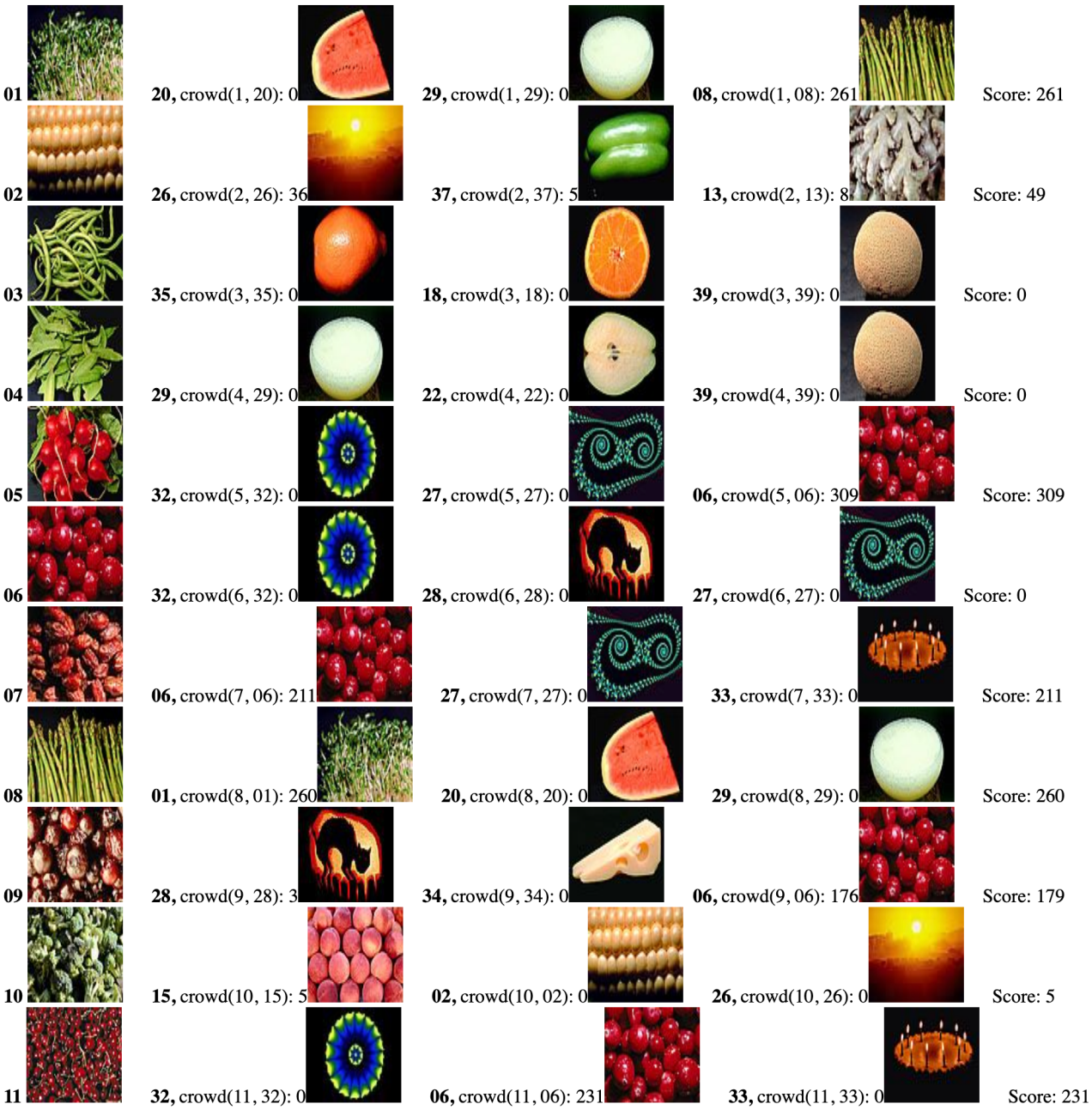
Now, I will discuss the bin partition for calcHist. Because the laplacian images use only one channel, there is only one value for the bin, instead of three. First, I tried the bin size of [5] as an experiment, and the total score is 6227. I realized that it's lower than what I desire because 5 is not a power of 2. For example, if the bin size were 8, it is 2^3, which will provide a cleaner bin partition. The next bin size is [8], and the total score rose to 6478. However, I did not settle with this score because I wanted to see what other multiples of 2 can get. Therefore, I tried the bin size of [16], which gave me a total score of 6336. And the bin size of [32] gave me a value of 6248. Finally, the bin size of [64] gave me a value of 7182. This is how I arrived at the final bin size of [64], which means that it is 2^6. I think this gave me the highest score because it is not only a multiple of 2, but it also creates enough bins for the laplacian images to be categorized more correctly. Because the laplacian images are lower resolution, it depends on a higher bin partition value to distinguish the differences.

I believe that the algorithm for this step works okay, worse than step 1, but still acceptable. 7182/25200 gives the percentage of 29%. Even though it is much lower than color, it makes sense because people use color more than texture when judging the similarity between images.

Step 3: Shape

1. System vs. Crowd Preference

Overall Score: 7208



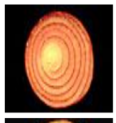
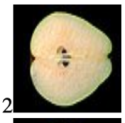

12 
13, crowd(12, 13): 175 
14, crowd(12, 14): 226 
15, crowd(12, 15): 12 Score: 413

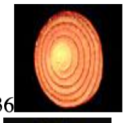
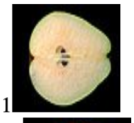

13 
15, crowd(13, 15): 10 
14, crowd(13, 14): 201 
12, crowd(13, 12): 230 Score: 441

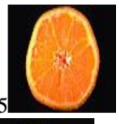
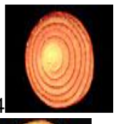

14 
13, crowd(14, 13): 147 
15, crowd(14, 15): 92 
12, crowd(14, 12): 210 Score: 449

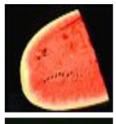
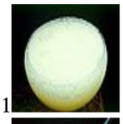
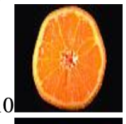

15 
13, crowd(15, 13): 1 
14, crowd(15, 14): 29 
12, crowd(15, 12): 7 Score: 37

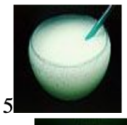
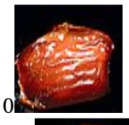

16 
15, crowd(16, 15): 129 
13, crowd(16, 13): 4 
14, crowd(16, 14): 3 Score: 136


17 
38, crowd(17, 38): 3 
18, crowd(17, 18): 268 
22, crowd(17, 22): 2 Score: 273

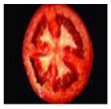
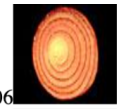
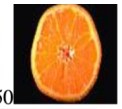

18 
17, crowd(18, 17): 236 
38, crowd(18, 38): 0 
22, crowd(18, 22): 1 Score: 237

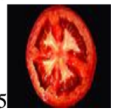
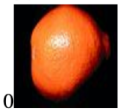
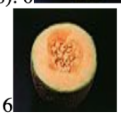

19 
38, crowd(19, 38): 76 
18, crowd(19, 18): 65 
17, crowd(19, 17): 34 Score: 175

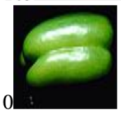

20 
29, crowd(20, 29): 1 
38, crowd(20, 38): 0 
18, crowd(20, 18): 10 Score: 11

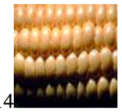
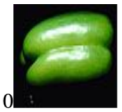
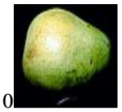

21 
30, crowd(21, 30): 5 
17, crowd(21, 17): 99 
40, crowd(21, 40): 0 Score: 104

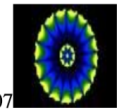
22
29, crowd(22, 29): 113
18, crowd(22, 18): 18
17, crowd(22, 17): 9 Score: 140

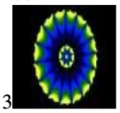
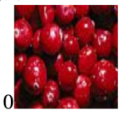

23 
17, crowd(23, 17): 106 
18, crowd(23, 18): 150 
38, crowd(23, 38): 0 Score: 256

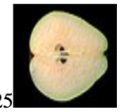
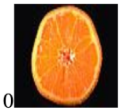

24 
23, crowd(24, 23): 165 
35, crowd(24, 35): 0 
21, crowd(24, 21): 6 Score: 171

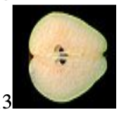
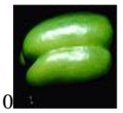

25 
37, crowd(25, 37): 0 
26, crowd(25, 26): 292 
33, crowd(25, 33): 96 Score: 388

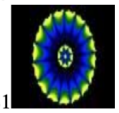

26 
02, crowd(26, 02): 14 
37, crowd(26, 37): 0 
36, crowd(26, 36): 0 Score: 14

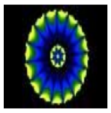
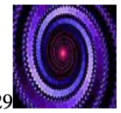
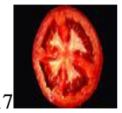

27 
32, crowd(27, 32): 207 
33, crowd(27, 33): 12 
06, crowd(27, 06): 0 Score: 219


28 
32, crowd(28, 32): 3 
33, crowd(28, 33): 149 
06, crowd(28, 06): 0 Score: 152

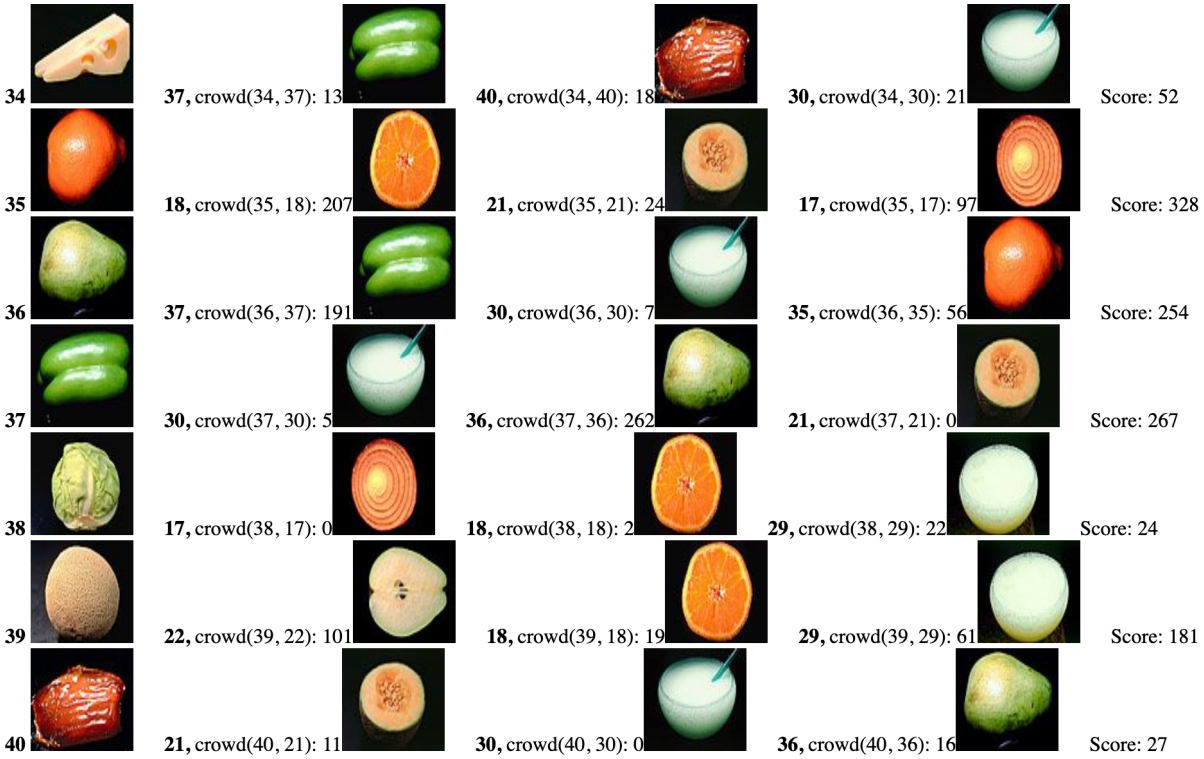

29 
22, crowd(29, 22): 125 
38, crowd(29, 38): 9 
18, crowd(29, 18): 0 Score: 134


30 
22, crowd(30, 22): 113 
21, crowd(30, 21): 59 
37, crowd(30, 37): 0 Score: 172


31 
32, crowd(31, 32): 281 
06, crowd(31, 06): 1 
28, crowd(31, 28): 16 Score: 298


32 
06, crowd(32, 06): 0 
31, crowd(32, 31): 229 
23, crowd(32, 23): 17 Score: 246

33
40, crowd(33, 40): 86
21, crowd(33, 21): 18
37, crowd(33, 37): 0 Score: 104



Overall Score: 7208

2. System vs. Personal Preferences
    a. The overlap is 28
3. Justification

I will first explain the algorithm, and then explain my choice of "blackness". I first converted all gray images into black and white form, and this is where the threshold for blackness comes into play. I made every pixel that is larger than the threshold value into black, and every pixel that is smaller than the threshold value into white. The resulting shape looks like this:


(image 21 – shape)

Between each image, I calculated the normalized distance between them by looking at if each pixel matches.

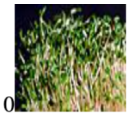
I tried several threshold values before I arrived at the final value 71. The first number I tried was 45, and it gave me a score of 6264. I then adjusted the value to 49, and it gave me a score of 6517. Then, I elevated the value significantly to 71, and it gave me the value of 7208. At this point, I was not sure how large of a value is enough for it to keep increasing, so I tried 95, and it resulted in 6406, which is lower than 71. I eventually settled with 71 because I think it is high enough to categorize grayish colors as white. Moreover, because the images are taken professionally with a black background, it is safer to assume a higher threshold value as the pixel is probably the background.

I believe the algorithm for this step also worked okay. By itself, it gives the accuracy of 29%. However, it also makes sense because people do not judge the similarity of images by shape itself. If two items are both circular but different color and shape, it is unlikely that they will be seen as similar.

Step 4: Symmetry

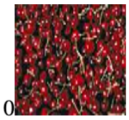
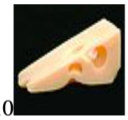
1. System vs. Crowd Preference

Overall Score is 5262


12

01, crowd(12, 01): 0

14, crowd(12, 14): 226

03, crowd(12, 03): 0 Score: 226


13

11, crowd(13, 11): 0

34, crowd(13, 34): 10

05, crowd(13, 05): 4 Score: 14


14

03, crowd(14, 03): 0

15, crowd(14, 15): 92

08, crowd(14, 08): 0 Score: 92


15

08, crowd(15, 08): 0

05, crowd(15, 05): 93

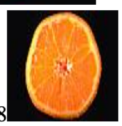
03, crowd(15, 03): 0 Score: 93


16

25, crowd(16, 25): 0

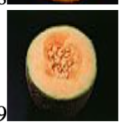
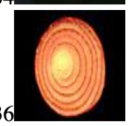
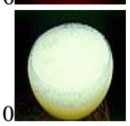
04, crowd(16, 04): 43

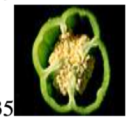
07, crowd(16, 07): 7 Score: 50

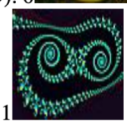

17

18, crowd(17, 18): 268

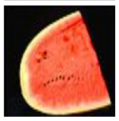
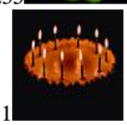
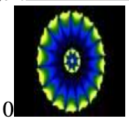
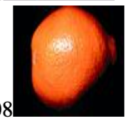
21, crowd(17, 21): 104

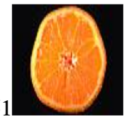
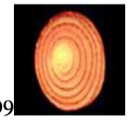
26, crowd(17, 26): 6 Score: 378


18

21, crowd(18, 21): 109

17, crowd(18, 17): 236

29, crowd(18, 29): 0 Score: 345


19

24, crowd(19, 24): 235

28, crowd(19, 28): 4

27, crowd(19, 27): 1 Score: 240


20

33, crowd(20, 33): 11

32, crowd(20, 32): 0

35, crowd(20, 35): 108 Score: 119


21

18, crowd(21, 18): 151

17, crowd(21, 17): 99

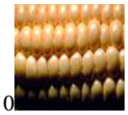
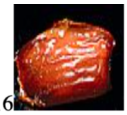
29, crowd(21, 29): 27 Score: 277

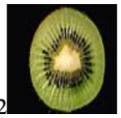
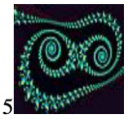
22

39, crowd(22, 39): 67

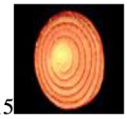
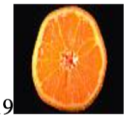
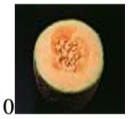
38, crowd(22, 38): 4

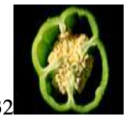
30, crowd(22, 30): 21 Score: 92

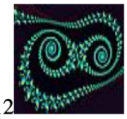
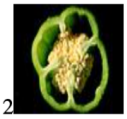
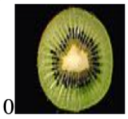

23 
06, crowd(23, 06): 9 
02, crowd(23, 02): 0 
40, crowd(23, 40): 16 Score: 25

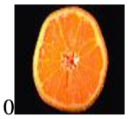
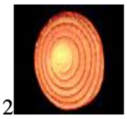

24 
19, crowd(24, 19): 222 
28, crowd(24, 28): 4 
27, crowd(24, 27): 5 Score: 231

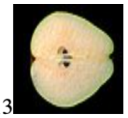
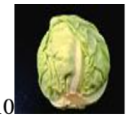

25 
16, crowd(25, 16): 0 
07, crowd(25, 07): 0 
04, crowd(25, 04): 0 Score: 0

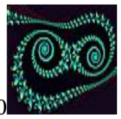
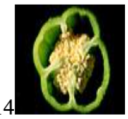

26 
17, crowd(26, 17): 15 
18, crowd(26, 18): 19 
21, crowd(26, 21): 0 Score: 34

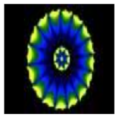
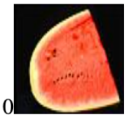
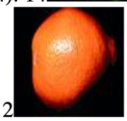

27 
28, crowd(27, 28): 30 
24, crowd(27, 24): 32 
19, crowd(27, 19): 21 Score: 83


28 
27, crowd(28, 27): 12 
24, crowd(28, 24): 2 
19, crowd(28, 19): 0 Score: 14

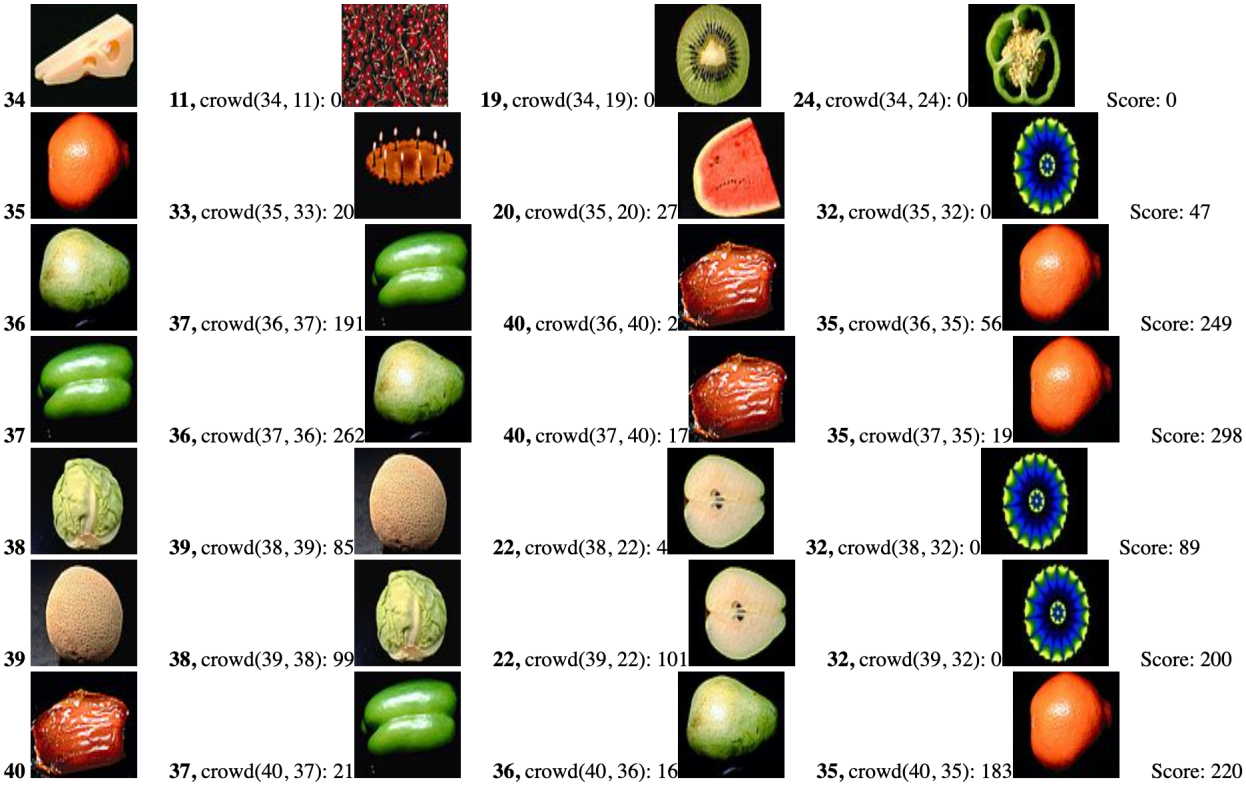

29 
21, crowd(29, 21): 63 
18, crowd(29, 18): 0 
17, crowd(29, 17): 2 Score: 65


30 
22, crowd(30, 22): 113 
39, crowd(30, 39): 42 
38, crowd(30, 38): 10 Score: 165


31 
27, crowd(31, 27): 160 
28, crowd(31, 28): 16 
24, crowd(31, 24): 14 Score: 190


32 
20, crowd(32, 20): 0 
33, crowd(32, 33): 17 
35, crowd(32, 35): 2 Score: 19

33
20, crowd(33, 20): 7
35, crowd(33, 35): 99
32, crowd(33, 32): 11 Score: 117



Overall Score: 5262

2. System vs. Personal Preferences
    a. The intersection value is 22
3. Justification

The algorithm for calculating symmetry is as follows: I first converted the image to black and white, with a new threshold value that I will explain later. Afterwards, I calculated the symmetric distance by looking at how the pixel values match between two symmetric columns. The value is then normalized and the closest three images is determined by how symmetrical or asymmetrical they are compared to the target image. Those that are most similar in symmetric distances are chosen.

I used a new black threshold for this part because the original value 71 gives only a score of 3843. I think this is because shape focuses on the foreground vs. background and a slightly lower threshold is enough to distinguish the differences. However, symmetry is more sensitive and a slightly higher black threshold will make sure that more gray colors are counted as white. The next value I tried therefore, was 80, which gave a result of 4593. Afterwards, I tried 90, and it seems like it is too high of a value as the result decreased to 3495. Therefore, I narrowed my range between 71 and 90. I tried each value between this range and the highest I found was 82, which gave the the score of 5262.

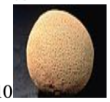
This step has the lowest accuracy of 5262/25200 = 20%, which means the algorithm is the least reliable out of all steps. However, it is still understandable because symmetry is not a common way to judge the similarity of images.

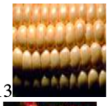
Step 5: Overall

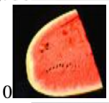
1. System vs. Crowd Preference

Overall Score: 12757

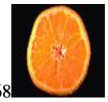
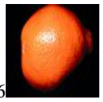

12 
14, crowd(12, 14): 226 
09, crowd(12, 09): 75 
13, crowd(12, 13): 175 Score: 476

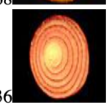
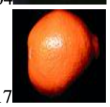

13 
14, crowd(13, 14): 201 
16, crowd(13, 16): 7 
39, crowd(13, 39): 10 Score: 218

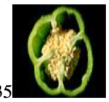
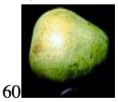

14 
13, crowd(14, 13): 147 
02, crowd(14, 02): 13 
39, crowd(14, 39): 30 Score: 190

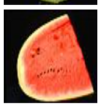
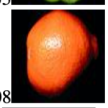
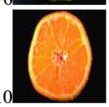
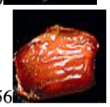

15 
14, crowd(15, 14): 29 
07, crowd(15, 07): 72 
20, crowd(15, 20): 0 Score: 101

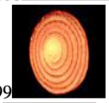

16 
10, crowd(16, 10): 240 
04, crowd(16, 04): 43 
01, crowd(16, 01): 39 Score: 322


17 
18, crowd(17, 18): 268 
21, crowd(17, 21): 104 
35, crowd(17, 35): 66 Score: 438

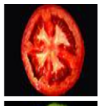
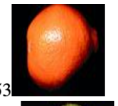
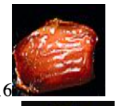
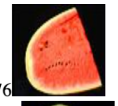

18 
17, crowd(18, 17): 236 
35, crowd(18, 35): 117 
21, crowd(18, 21): 109 Score: 462


19 
24, crowd(19, 24): 235 
38, crowd(19, 38): 76 
36, crowd(19, 36): 60 Score: 371

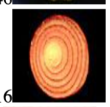

20 
35, crowd(20, 35): 108 
18, crowd(20, 18): 10 
40, crowd(20, 40): 156 Score: 274

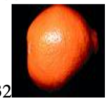
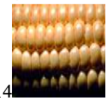
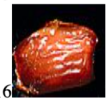

21 
17, crowd(21, 17): 99 
34, crowd(21, 34): 3 
18, crowd(21, 18): 151 Score: 253

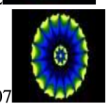
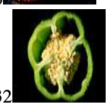
22
38, crowd(22, 38): 4
29, crowd(22, 29): 113
36, crowd(22, 36): 84 Score: 201

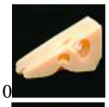
23  **35,** crowd(23, 35): 53  **40,** crowd(23, 40): 16  **20,** crowd(23, 20): 76  Score: 145

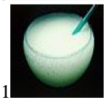
24  **19,** crowd(24, 19): 222  **38,** crowd(24, 38): 46  **36,** crowd(24, 36): 31  Score: 299

25  **28,** crowd(25, 28): 178  **17,** crowd(25, 17): 16  **34,** crowd(25, 34): 5  Score: 199

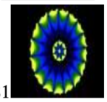
26  **35,** crowd(26, 35): 32  **02,** crowd(26, 02): 14  **40,** crowd(26, 40): 16  Score: 62

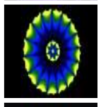
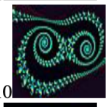
27  **32,** crowd(27, 32): 207  **28,** crowd(27, 28): 30  **24,** crowd(27, 24): 32  Score: 269

28  **33,** crowd(28, 33): 149  **34,** crowd(28, 34): 0  **24,** crowd(28, 24): 2  Score: 151

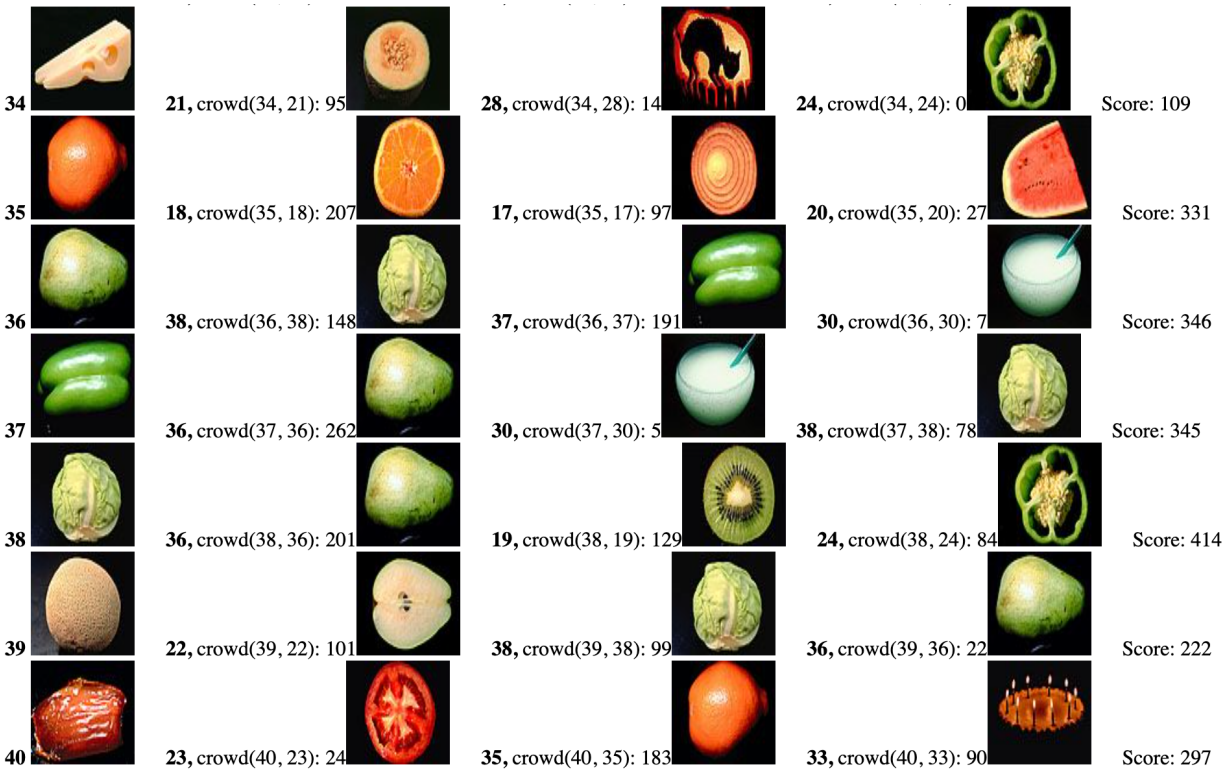
29  **30,** crowd(29, 30): 311  **38,** crowd(29, 38): 9  **22,** crowd(29, 22): 125  Score: 445

30  **29,** crowd(30, 29): 305  **36,** crowd(30, 36): 29  **38,** crowd(30, 38): 10  Score: 344

31  **28,** crowd(31, 28): 16  **27,** crowd(31, 27): 160  **32,** crowd(31, 32): 281  Score: 457

32  **27,** crowd(32, 27): 110  **33,** crowd(32, 33): 17  **24,** crowd(32, 24): 71  Score: 198

33 **40,** crowd(33, 40): 86 **28,** crowd(33, 28): 149 **21,** crowd(33, 21): 18 Score: 253



Overall Score: 12757

2. System vs. Personal Preference
    a. The overlap score is 48.
3. Justification

The algorithm for step 5 works as follows: for each image, I access the color distance, texture distance, shape distance, and symmetry distance to all other images. The I multiplied each distance by its corresponding standard simplex value. Lastly, I chose the three images with the least distance to a given image.

The standard simplex value that I settled with was [0.39, 0.21, 0.23, 0.17] for color, texture, shape, and symmetry respectively. Color has the highest value because it is the most accurate one and it is the most common way that images are compared. My second highest value is shape because it has the second highest value and it is also an important way to compare images. Symmetry has the least importance because first it is harder to detect minor symmetric differences, and second it is not as common to compare images based on symmetry.

However, before I arrived at my final answer. I also tried several different approaches. The first combination I tried was [0.5, 0.15, 0.2, 0.15], which gave the result of 12251. I tried this combination because I believed that color should be occupy 50% of our decision making when comparing images. However, the result was not as high as the final result. The second combination I tried was [0.4, 0.2, 0.3, 0.1], which gave the result of 12087. I think this is because the score for symmetry is too low and the score for shape is too high. Eventually I found the

combination by dividing each grand score by the total sum of grand scores. For example, I divided 11561/(11561+7182+7208+5262) to get a rough estimation of how much color should occupy.

This is the most accurate step and it means the algorithm worked well. It provides an accuracy percentage of 12757/25200 = 51%. Given the highest possible overall percentage is 72%, the algorithm is working well. This is because it is the only step that considers the combination of color, texture, shape, and symmetry. And it is only when the program considers all those elements that the most accurate result appears.

Step 6: MyCrowd Overall

1. Justification

First, I wanted to see which metric I lean towards the most. Therefore, I tried making one distance dominate. If the images are compared entirely based on color, the grand score is 103/240. If entirely based on texture, the grand score is 59/240. If entirely based on shape, the grand score is 61/240. If entirely based on symmetry, the grand score is 44/240. From this, one can see that the score is highest with color, then shape, then texture, then symmetry. This means that when I am adjusting, I should prioritize the distances in that order. I tried the combination [0.7, 0.1, 0.15, 0.05] for color, texture, shape, and symmetry respectively and received a total score of 102. Then I tried, the same combination as before [0.39, 0.21, 0.23, 0.17], which gave a total score of 106. I also tried the combination of [0.42, 0.21, 0.23, 0.14] to see if increasing the importance of color and decreasing the importance of symmetry will give me a better value. However, the score lowered to 103. Eventually, I settled with the combination [0.39, 0.21, 0.23, 0.17].

In my case, because the weight vector is exactly the same as the weight vector for the class, it does not contribute that much to the improvement of my algorithm. However, it does reconfirm the weight vector from the previous step, which is really important as it shows that my ways of judging the similarity between images is similar to how others do it. And it also confirms that what makes most people happy will make me happy too.

Citation

1. https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html
2. https://techtutorialsx.com/2019/04/13/python-opencv-converting-image-to-black-and-white/
3. https://pyimagesearch.com/2021/04/28/opencv-image-histograms-cv2-calchist/