Project Proposal

Finding Image Similarities: Histogram Method

Imara Willie Johnson

Principles of Data Mining CSCI 420.01

IWJ1586@rit.edu

Other Methods Used

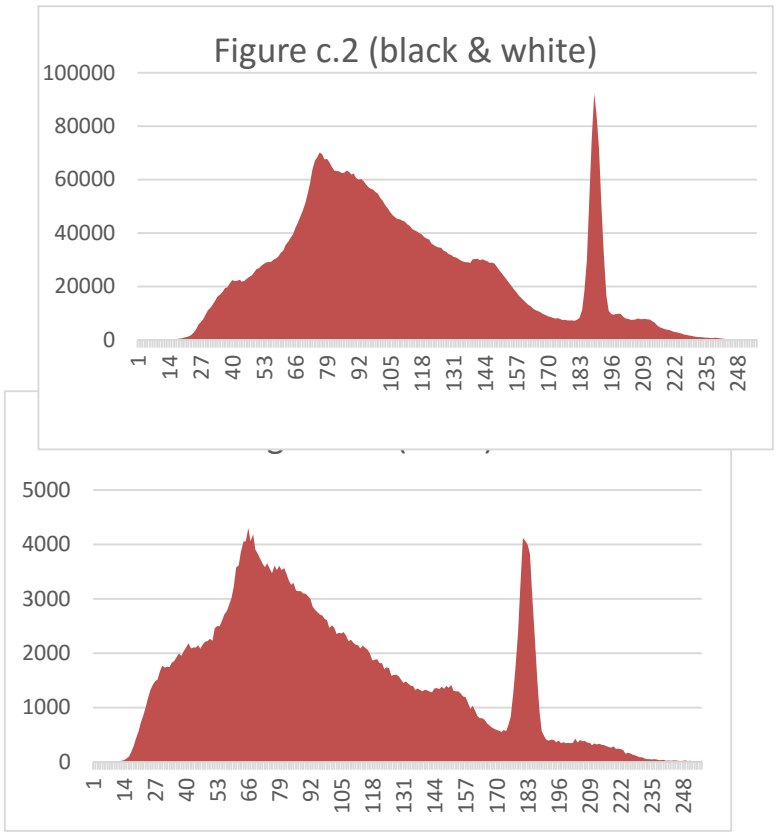
Finding Image similarities using histograms is a well-known, but as of late unpopular method because there exist more effective ones, like keypoint matching. Histogram analysis is mostly used as a tool in academic spheres, however opencv offers a function *compareHist*, which uses a specified distance metric to calculate the distance between two histograms.

Problems With This Method

The common problem with using histogram comparison as a tool to relate images, is that dissimilar images, can have similar histograms. A histogram is merely a measurement of the amount of pixels that correspond with each tonal value and so, the histogram of an image in color and the same image in greyscale are differ slightly but not in a way that is meaningful. For some applications this would be useful, but not for what I'm trying to do.

Figure c.1

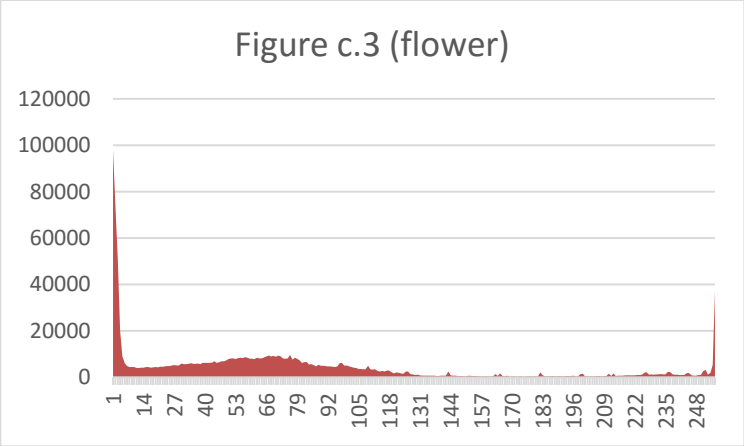
Figure c.2

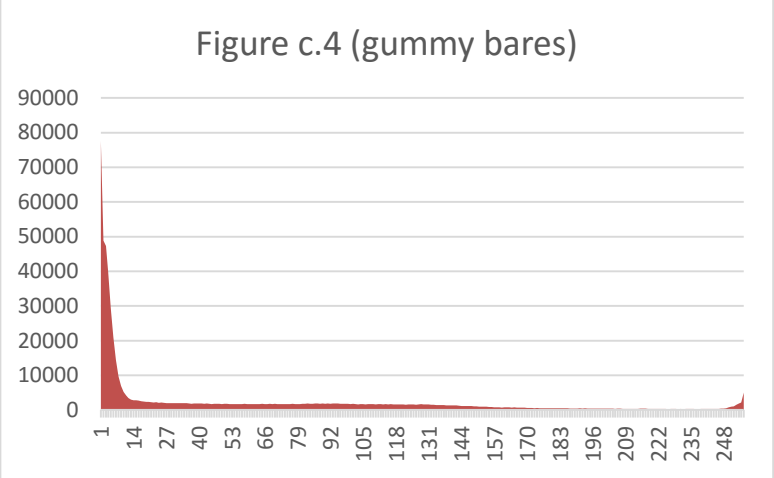


Another problem occurs when using color histograms; since histograms only measure the distribution of color, spatial differences are ignored and the histogram an image with a wide disbursement of a specific color, can be very similar to the histogram of an image with only one area with a high concentration of that color. Figure c.3 and Figure c.4 have very similar blue histograms.

Figure c.3 Flower

Figure c.4 gummy bares





Histogram analysis alone doesn't take into account objects, or edge detection in images, which would allow two dissimilar images to appear similar if only taking into account histograms.

Ethical Considerations

There are no ethical considerations for this topic.

Business Cases

There are no business cases for this topic.

Possible Difficulties Generating Data

The data itself is easily generated, as I can just take pictures with my phone. However, histograms are sensitive to quantization and therefore lossy image compression, so two identical images in JPEG and PNG format, will not compare well. So Images might need to be of the same format.

Cleaning Data

Cleanliness of the data depends on the camera, quantization effects the histogram, so different resolutions on different cameras will affect the results. For example, if the image has a high spatial frequency, aliasing may be a problem for differing cameras.

Accidental pictures, or pictures you did not intend to take, show up all the time, usually of the inside of your pocket. It may be a good idea to eliminate dark photos with few or no edges.

Techniques and Methods

Color Histogram Analysis: produce three histograms for an image's red, green, and blue values. Compare each to another image's corresponding histogram.

Bin-to-bin comparison: distances are found by comparing corresponding bins only.

Cross-bin comparison: distances are found by comparing all bins.

Joint Histograms: Multidimensional histograms that incorporate some set of attributes. Each entry is the number of pixels that contain some particular combination of attributes.

Texture Analysis: comparing texture of regions using a normalized gradient magnitude histogram and a normalized gradient orientation histogram.

Colored Coherence Vector: For each discretized color in an image, the ccv contains the amount of coherent and incoherent pixels. A coherent pixel, is a pixel that is in an area with a "large" contiguous group of pixels with the same color. This method solves the spatial problem in histogram comparison.

Considerations

Since similarity is subjective, there may be issues with the chosen threshold. Usage also plays a part into what should be defined as "similar" images, a user may be trying to find images with the same content. For example, when I visited south Dakota I took a million pictures of mount Rushmore, maybe I want to find all the pictures I took of mount Rushmore. Or maybe I'm making a portrait of Obama using pictures of American people, and I want to find headshots with a blue background. Or maybe I want to find all the pictures of a desk that I took, but I used different color filters on most of them. All these scenarios may require different definitions of similar.

Should I cluster?

My first instinct for this project was to do a cluster analysis on a set of images, to find groups of similar images. But I ran into a conceptual problem. If image A, is similar to image B, and Image B is similar to image C, is image A similar to image C by the same metric?

Image A

Image C

Image B

Distance Metrics

L_1 and L_2 norms, and χ^2 distance are used for bin-to-bin comparison.

Quadratic-Form and Earth Mover Distance are used for cross-bin comparison.

Attributes

Distance:

Attributes:

- Distribution of red values
- Distribution of green values
- Distribution of blue values
- Texture
- Color coherence

The distance matrices I will use are EMD for histograms, and L_1 for CCV. Total distance will be a summation of all individual attributes.

RGB distribution will be represented as three histograms

Texture can be described as gradient magnitude and orientation. to quantize and compare these values I will compute a normalized histogram of gradient magnitudes, and a normalized histogram of gradient orientations. To get the gradient histogram, I will use the Sobel operator, then apply the following equations for magnitude and direction respectively.

$$M = \sqrt{(G_x^2 + G_y^2)} \qquad \theta = \tan^{-1} 2(G_y, G_x)$$

To compute a color coherence vector, it helps to blur the image beforehand to reduce small differences. Define a color space as buckets with a range of intensities, then label each distinct color region (an area with at least 1 pixel of a certain color); count the number of pixels in each region, if it's less than some value *k*, those pixels are incoherent. Regions with large (>= k) pixel counts are classified as coherent. For each color, compute the number of coherent and non-coherent pixels. For this I will probably define a simple 256 color space.

I'm going to employ cluster analysis using DBSCAN, and find the ε and minPts that captures the most pictures that are similar, while minimizing the SSE. I will have sets of data with different cohesion measures (e.g. set of all similar images, set of all different images, set of mostly different images…) and utilize the *k*-fold cross-validation method.

Algorithms

- DBSCAN
- Expectation Maximization
- K-NN (for queries)

Final Report

Finding Image Similarities: Histogram Analysis

FISHA

By Imara W. Johnson

Abstract

Histogram analysis is a simple and easy way to determine similarity between two images, however it is also a method that is prone to inaccuracies. These inaccuracies can be reduced using a number of other attributes that may also represent image similarity, and when applied with data mining concepts, provides a machine capable of finding it in any data set.

Algorithms Used

For this project I used one dimensional classification and DBSCAN.

For one dimensional classification I found the best threshold to determine whether two images are similar or not based on the distance between them. I did this in order to have a query option, so that one could return all similar images in a given directory.

I used DBSCAN in order to, given a directory of image files, return groups of similar images.

I trained on 2,543 Images, because I had to clean the data so carefully, and tested on the same set of 340 images, to make sure I could quantify differences accurately.

Methods, Problems & Solutions

One Dimensional Classification

In order to find the best threshold, I had 10 groups of different images and for each group I took one image, out as a comparison image, and I labeled the images in that group similar, and I labeled all the other images in the other groups not similar. Then I averaged those 10 resulting thresholds.

The problem I ran into was data. I got my images from Flikr, which is a photo sharing website that sometimes allows you to download pictures. For this project I used pictures from the national park service Grand Canyon division, NASA, and NASA Goddard space flight center. The images were organized into albums, and the albums were usually composed of pictures that fall into the same category. For example, one album was of satellite images of an oil spill in the Gulf of Mexico. Initially, I declared all images within an album as similar, and images outside the album as not similar. Unfortunately, though they shared some value: content, date, place, event, photo device…etc., all images within an album

were not automatically similar, and all images not in the same album were dissimilar. So I had to go through a clean the data, so that groups fit the criteria.

DBSCAN

Initially I used the threshold I found from the 1D classifier as my **Eps** value, and 3 as my **Minpts** value. I found that my **Eps** was too large as it was clustering together dissimilar images. The problem was the same as I had expected and noted earlier in my paper, my **Eps** assumed that if

A is similar to B

&

B is similar to C

Then

A is similar to C

This is not true.

In order to decide which boarder points go to which cluster in the case of overlapping (which happened a lot), I calculated its distance from the center of each cluster and chose the smallest. This improved my cluster cohesion by a large amount, from when I simply chose the cluster that found the point first.

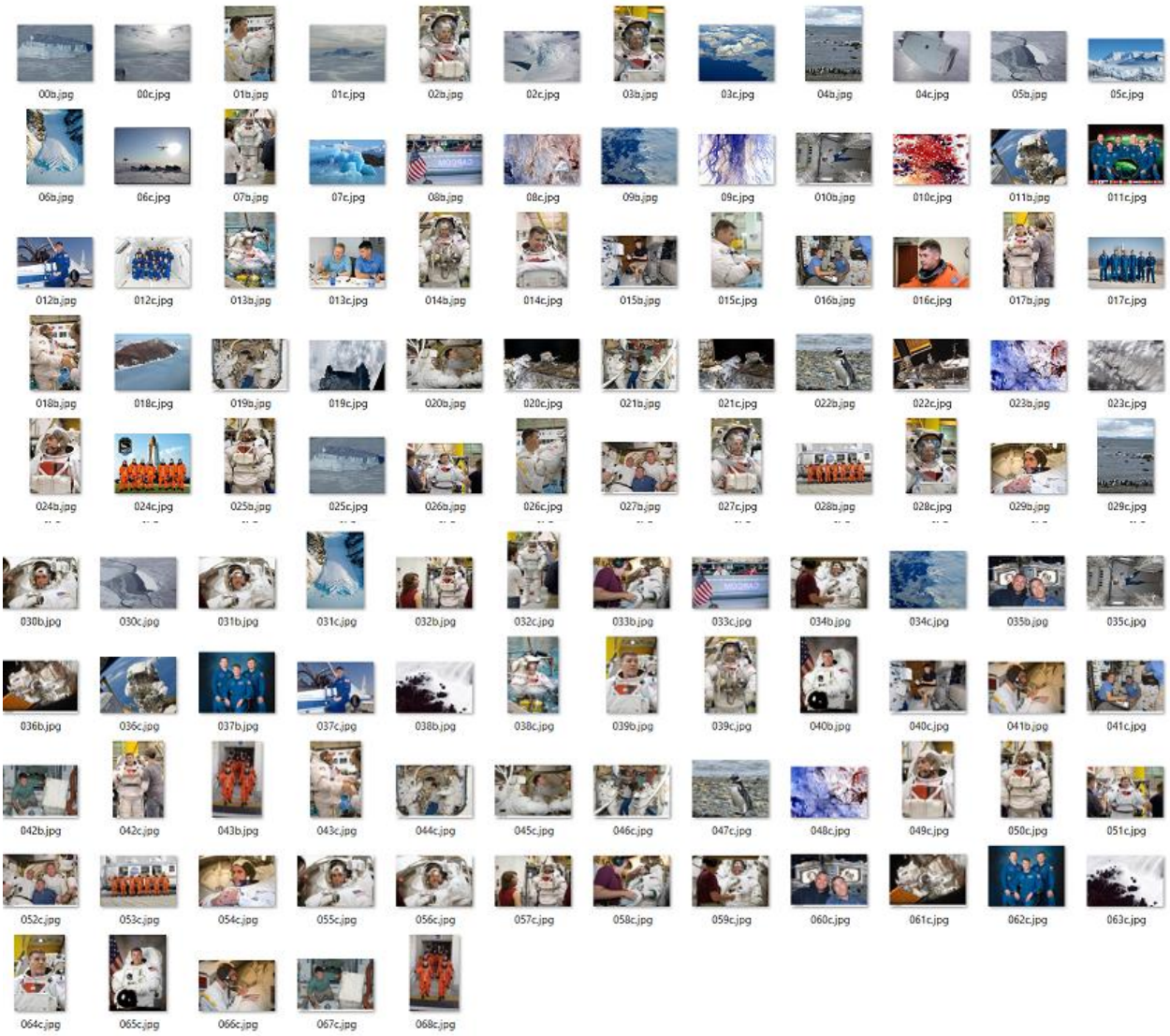
DBSCAN Results

Eps = 2.9, **Minpts** = 3

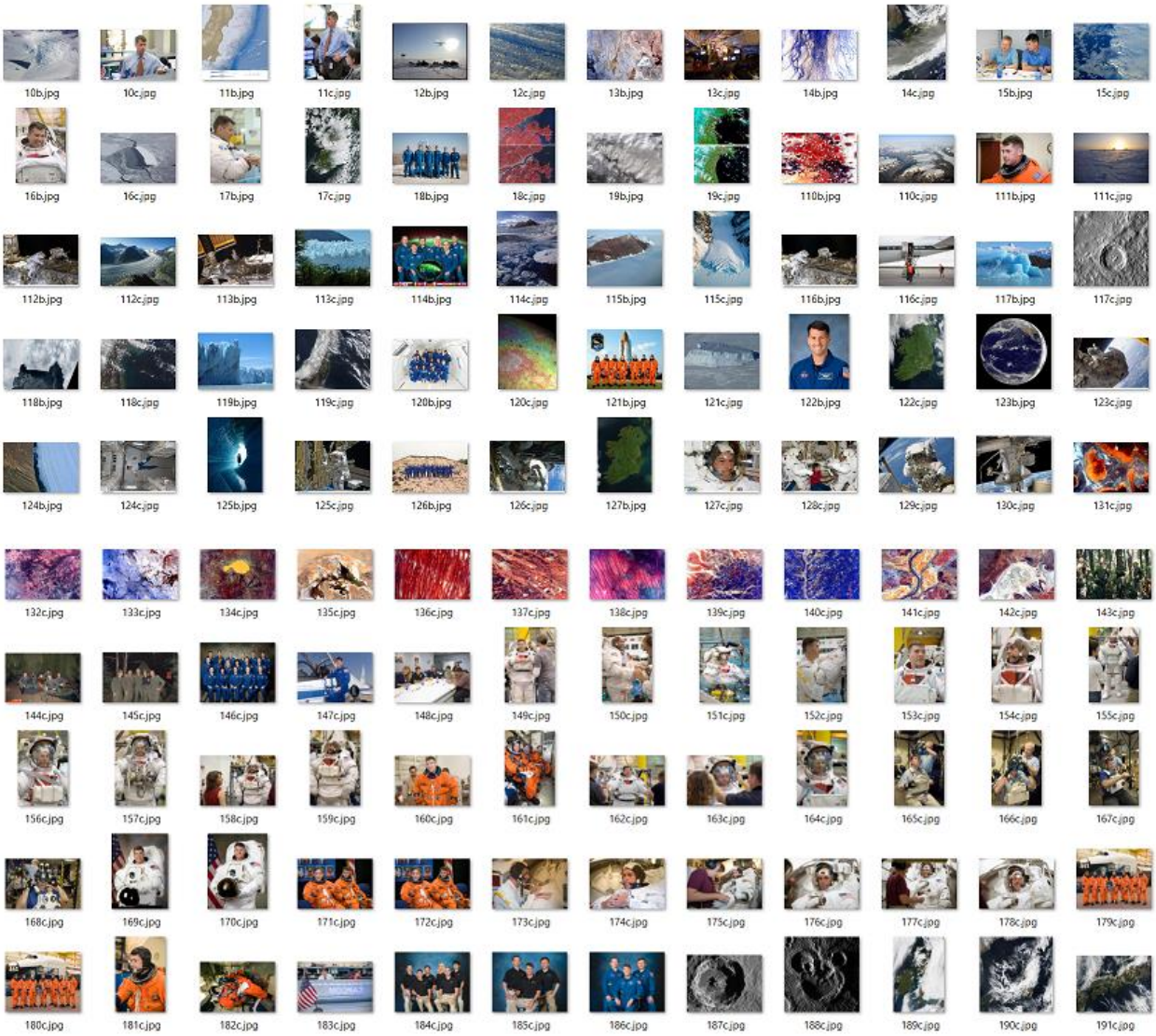
Cluster Number	Total Points	Number of Distinguishable groups
0	113	6
1	196	9

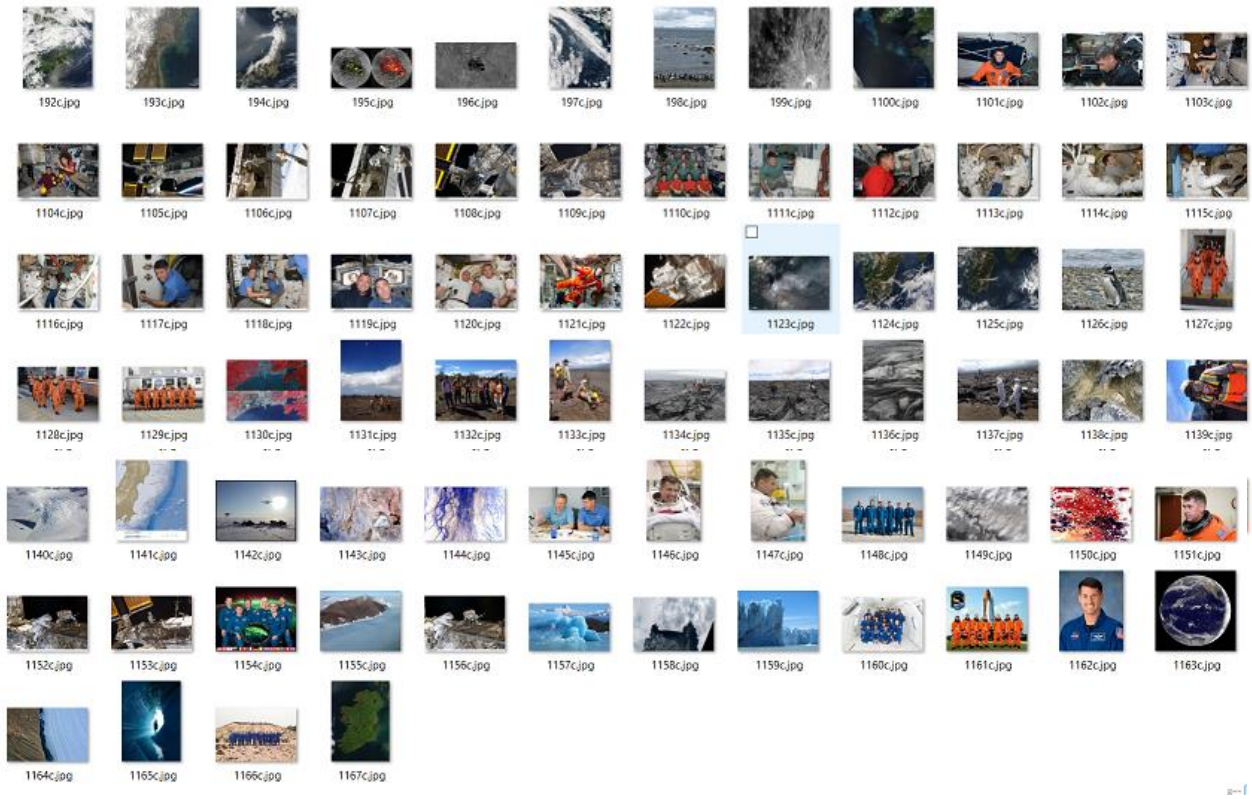
Total points (for all clusters) = 309

Cluster 0


00a.jpg
00c.jpg
01a.jpg
01c.jpg
02a.jpg
02c.jpg
03a.jpg
03c.jpg
04a.jpg
04c.jpg
05a.jpg
05c.jpg
06a.jpg
06c.jpg
07a.jpg
07c.jpg
08a.jpg
08c.jpg
09a.jpg
09c.jpg
010a.jpg
010c.jpg
011a.jpg
011c.jpg
012a.jpg
012c.jpg
013a.jpg
013c.jpg
014a.jpg
014c.jpg
015a.jpg
015c.jpg
016a.jpg
016c.jpg
017a.jpg
017c.jpg
018a.jpg
018c.jpg
019a.jpg
019c.jpg
020a.jpg
020c.jpg
021a.jpg
021c.jpg
022a.jpg
022c.jpg
023a.jpg
023c.jpg
024a.jpg
024c.jpg
025a.jpg
025c.jpg
026a.jpg
026c.jpg
027a.jpg
027c.jpg
028a.jpg
028c.jpg
029a.jpg
029c.jpg
030b.jpg
030c.jpg
031b.jpg
031c.jpg
032b.jpg
032c.jpg
033b.jpg
033c.jpg
034b.jpg
034c.jpg
035b.jpg
035c.jpg
036b.jpg
036c.jpg
037b.jpg
037c.jpg
038b.jpg
038c.jpg
039b.jpg
039c.jpg
040b.jpg
040c.jpg
041b.jpg
041c.jpg
042b.jpg
042c.jpg
043b.jpg
043c.jpg
044c.jpg
045c.jpg
046c.jpg
047c.jpg
048c.jpg
049c.jpg
050c.jpg
051c.jpg
052c.jpg
053c.jpg
054c.jpg
055c.jpg
056c.jpg
057c.jpg
058c.jpg
059c.jpg
060c.jpg
061c.jpg
062c.jpg
063c.jpg
064c.jpg
065c.jpg
066c.jpg
067c.jpg
068c.jpg

Cluster 1


10b.jpg
10c.jpg
11b.jpg
11c.jpg
12b.jpg
12c.jpg
13b.jpg
13c.jpg
14b.jpg
14c.jpg
15b.jpg
15c.jpg
16b.jpg
16c.jpg
17b.jpg
17c.jpg
18b.jpg
18c.jpg
19b.jpg
19c.jpg
110b.jpg
110c.jpg
111b.jpg
111c.jpg
112b.jpg
112c.jpg
113b.jpg
113c.jpg
114b.jpg
114c.jpg
115b.jpg
115c.jpg
116b.jpg
116c.jpg
117b.jpg
117c.jpg
118b.jpg
118c.jpg
119b.jpg
119c.jpg
120b.jpg
120c.jpg
121b.jpg
121c.jpg
122b.jpg
122c.jpg
123b.jpg
123c.jpg
124b.jpg
124c.jpg
125b.jpg
125c.jpg
126b.jpg
126c.jpg
127b.jpg
127c.jpg
128c.jpg
129c.jpg
130c.jpg
131c.jpg
132c.jpg
133c.jpg
134c.jpg
135c.jpg
136c.jpg
137c.jpg
138c.jpg
139c.jpg
140c.jpg
141c.jpg
142c.jpg
143c.jpg
144c.jpg
145c.jpg
146c.jpg
147c.jpg
148c.jpg
149c.jpg
150c.jpg
151c.jpg
152c.jpg
153c.jpg
154c.jpg
155c.jpg
156c.jpg
157c.jpg
158c.jpg
159c.jpg
160c.jpg
161c.jpg
162c.jpg
163c.jpg
164c.jpg
165c.jpg
166c.jpg
167c.jpg
168c.jpg
169c.jpg
170c.jpg
171c.jpg
172c.jpg
173c.jpg
174c.jpg
175c.jpg
176c.jpg
177c.jpg
178c.jpg
179c.jpg
180c.jpg
181c.jpg
182c.jpg
183c.jpg
184c.jpg
185c.jpg
186c.jpg
187c.jpg
188c.jpg
189c.jpg
190c.jpg
191c.jpg



Cluster Number	Total Points	Number of Distinguishable groups
0	126	8
1	11	2
2	45	5
3	5	1

Total points (for all clusters) = 187

Cluster 0

Cluster 1

Cluster 2

Cluster 3

Eps = 1.5, **Minpts** = 2

Cluster Number	Total Points	Number of Distinguishable Groups
0	5	1
1	5	1
2	15	2
3	39	1
4	6	1
5	13	3
6	6	1

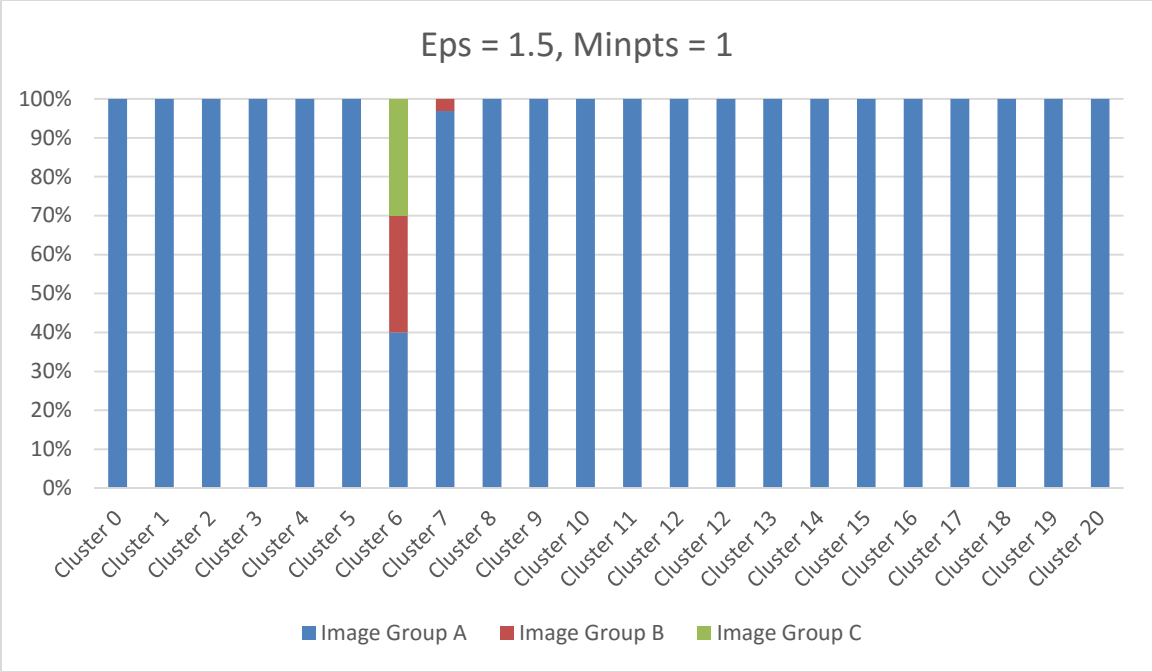
Total points (for all clusters) = 89

As you can see, decreasing the **Eps** to 1.5 significantly increased accuracy. I decided on 1.5 because it minimized the number of distinguishable groups in a cluster.

Eps = 1.5, **Minpts** = 1

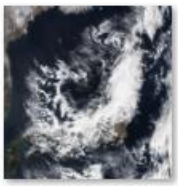
Cluster Number	Total Points	Number of Distinguishable Groups
0	5	1
1	3	1
2	5	1
3	3	1
4	2	1
5	3	1
6	10	3
7	32	2
8	2	1
9	3	1
10	3	1
11	11	1
12	2	1
13	2	1
14	2	1
15	2	1
16	2	1
17	5	1
18	3	1
19	2	1
20	2	1

Total Points (for all clusters) = 104



It occurred to me that by lowering the **Minpts** to 1, I would be reducing the amount of images categorized as noise, even is an image is similar only to 1 other image, I would still like to return them. Unfortunately, the amount of individual clusters that could be combined into larger clusters. For example, cluster 1, 4, 5, and 16 can all be combined into one,

Cluster 1


10b.jpg


10c.jpg


11c.jpg

Cluster 4

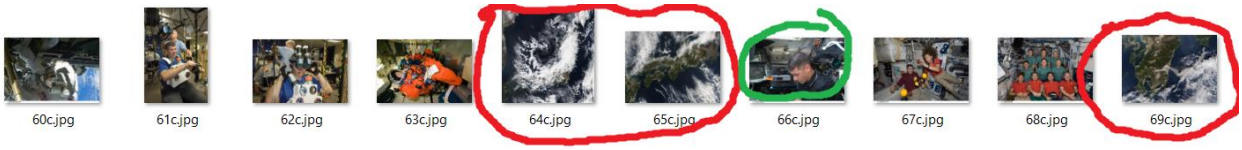

40c.jpg


41c.jpg

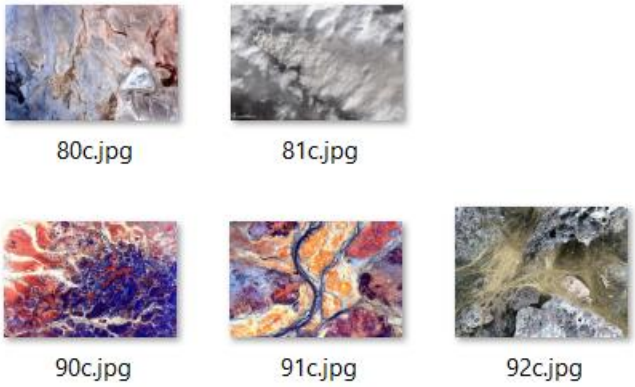
Cluster 5

Cluster 16

And cluster 6, incorrectly contains similar images, though I can understand how the images circled in red and the image circled in green can be clustered together:



Similarly, clusters 8 and 9 can be combined, and they also contain unstructured images, or images with a lack of clearly distinguishable components, which make their content difficult to identify at first:



This means that texture analysis, which I do using edges, plays a sizable part in the distance.

Cluster 1 and 5 are so similar that the issue must lie with my algorithm or distance logic, and not my **Eps** and **Minpts** choice.

Setting **Eps** to 1.6 was tempting because when **Eps** = 1.6 and **Minpts** = 1:

total points for all clusters is 141, reducing the amount of images labeled outliers.

The number of clusters with only 2-3 points goes from 75% to 50%

And there are only 2 clusters with more than 1 distinguishable group, cluster 13 and 14

However, these images are not completely dissimilar

<u>Cluster 13</u>

In cluster 13, the images with astronauts in orange suits (which I've circled in orange), correspond with the images of the seated portraits with the front row wearing orange shirts (circled in pink), which correspond with the blue circled images. And those correspond with the green images, and the green images correspond with the astronaut being suited up, circled in purple and magenta.

Cluster 14

In cluster 14, the coherence of white is clearly what connect these images.

Even though an **Eps** of 1.6 has arguably better clusters, in terms of total images returned and clusters which can be combined into bigger cluster, I still go with **Eps** = 1.5 because of better cluster cohesion, which is most important.

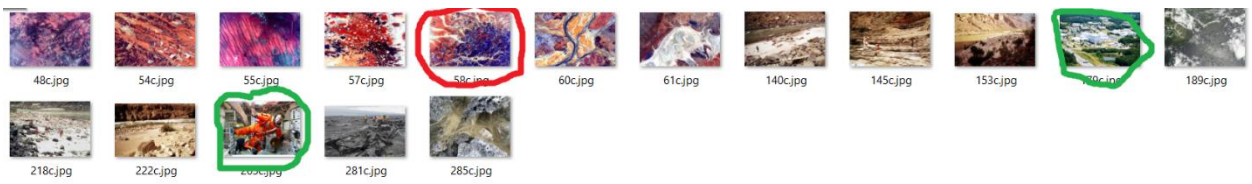
One Dimensional Classification Results

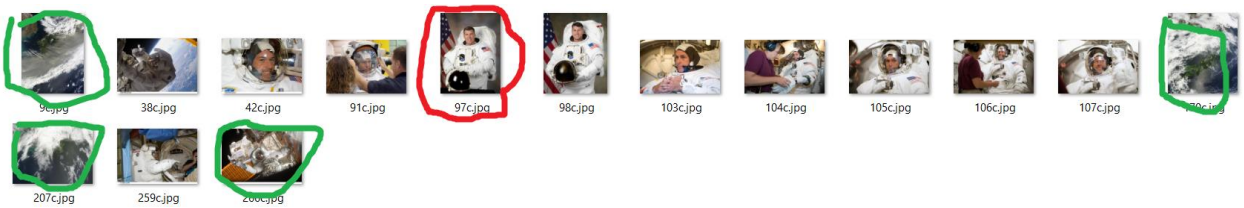
I used a threshold of 2.3

I found that this threshold finds all the related images, and some that could be left out, unfortunately, anything lower leaves out too much data.

Red is the comparison image

Green are outliers





Memory Problem

My initial plan for gathering an images attributes, was to read in the directory to my java program and store for EACH image object, the original image, the image after applying the Sobel filter in the horizontal direction, the image after applying the Sobel filter in the vertical direction, the image after applying a blur effect, along with the RGB histograms, magnitude and direction histograms, and the color coherence vectors. I ran into a memory problem if a tried to store one image object, with an original image over 250x250. So I used Matlab to do all my blurring and RGB histogram finding, and edge detection, which saved so much time and space. I also tried resizing the images, but I was afraid of the affect it would have on my calculations so I decided against it. My solution was to calculate everything as it was read in (magnitude, direction, color set, color coherence vectors), so I would not have to store all those objects.

Other Algorithms

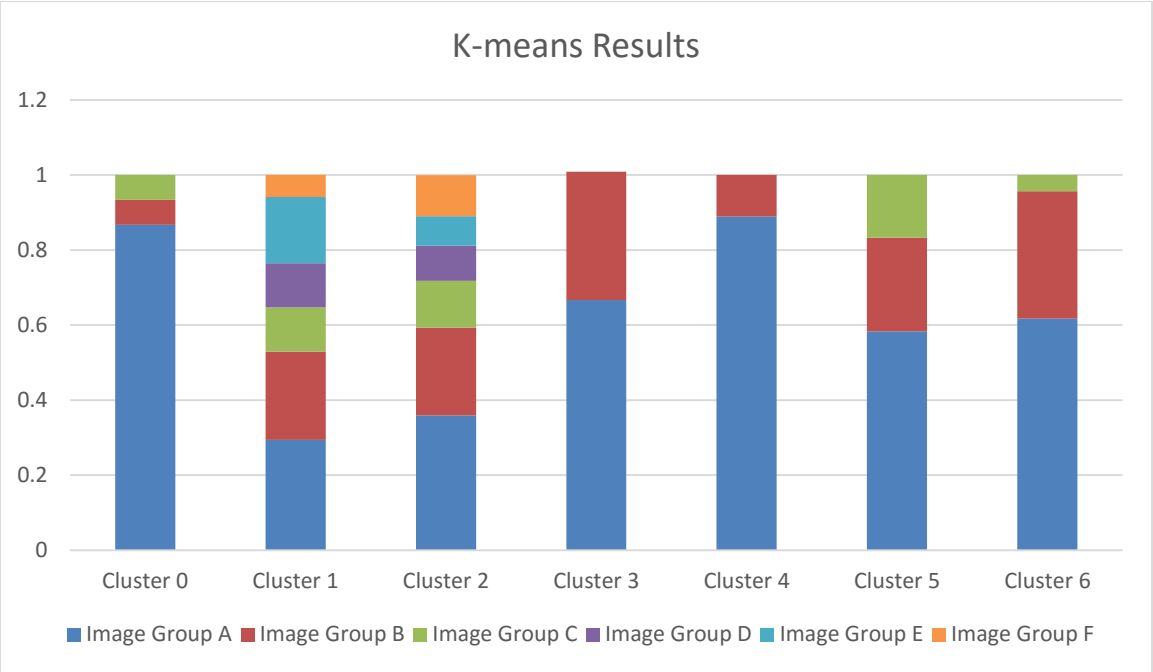
I wanted to try image segmentation, and object detection, which uses K-means. Unfortunately, I was unable to not overfit my classifier. I wanted to segment my images by texture, then assign them to some predefined texture range, I had difficulty finding descriptive texture ranges that fit all images. At first I tried to find what I wanted to separate; specific things like wood from brick, or more broad definitions like nature from urban. I would like to continue looking into image segmentation, but for this project I did not use it.

The third algorithm I tried started out as K-means and then turned into K-NN. So I found K by finding a group of images that were greater than or equal to some distance to each other image in the group. I then tried K-means however, I got huge blobs of data returned. Then I realized that I didn't need to re-evaluate my centroid every time a point was added, my initial seeds adequately scoped the data, so I treated them a classes.

K-Means Results

Cluster Number	Total Points	Number of Distinguishable Groups
0	15	3

1	34	6
2	64	6
3	36	2
4	9	2
5	12	3
6	47	3



Conclusions

Similarity is subjective, however, I think there were some things wrong in how I calculated distance.

Main Problems

- Distance tells me that it's similar, not how similar, because my attributes were not descriptive enough. This means that after a certain threshold, change in distance doesn't mean anything.

  For example, say my threshold = 2, and when I compare all the images in a directory to a specific image X, I am returned a group of 2 images [A, B]. A is more similar to X, but if I were to set my threshold lower and lower until there was only 1 image returned, either A or B could be left.

- My distance metric was bin-to-bin and not cross-bin. I used L1 distance because it was easier to normalize my answers. I believe this is why obviously similar images end up in differing classifications.

If I don’t take into account subject matter, my results are quite accurate. A way to tailor to subject matter is to add the image file name as an attribute, even if it is just a series of numbers, it is likely, especially if the images were taken or stored together, that their distance is indicative of subject matter similarities.