COMS W4735y: Visual Interfaces to Computers
Spring 2015
Assignment 2: Due March 12, 2015

Visual Information Retrieval: Design Specifications

The goal of this assignment is to write and analyze algorithms that explore different ways of deciding the degree of similarities amongst actual images.

These images, of fruits and vegetables plus some distractors, are available on Courseworks, in both JPG format and in PPM format. They are all of the same small size, and they all use‘ black backgrounds. Their generic file names are iNN.jpg or iNN.ppm, where NN is from 01 to 40. The JPG files are for use with any display you chose to write, as some packages require JPGs. However, the PPM files are much easier to manipulate within code, and they follow the PPM P6 color byte standard, as follows.

1. The first line is the "magic number" for one of eight PPM formats. For all 40 images it is the string "P6", the magic number indicating that the PPM format is in color and stored in byte format.
2. The second line is a comment line beginning with "#". In this case, it is an encoded string placed in the image by its source to track unauthorized use, which in all cases is "#JDONtLJDO", the magic number for the company Wernher Krutein Productions, Inc. (The instructor has paid a royalty for the rights to use these images in the course.)
3. The third line has two strings indicating the width and height, respectively, of the image, which in all cases is "89 60", meaning 89 columns and 60 rows (thumbnail size).
4. The fourth line has a string indicating the maximum value of each pixel coordinate, which in all cases is "255", meaning each pixel is represented in single bytes.
5. The next 60 lines each consist of one image row, where each image row consists of 89 pixels in byte format, so each row takes up 89*3 bytes encoded in binary, not as a readable string.
6. Each pixel consists of three bytes, one each for R, G, B, in that order, where 0 is zero color and 255 is full color.

1 Step 1 (5 points): Gross color matching

Write an algorithm that compares two of these images strictly on their total color distributions. That is, generate their three dimensional color histograms, and compare the histograms using the normalized L1 comparison, leading to a value in the interval [0,1], where 0 means no similarity whatsoever (no shared colors at all) and 1 means total similarity (image2 is really just the same pixels as image1, but possibly spatially scrambled).

You will have to make two important decisions. First, you must decide how rich or sparse to make your histogram space, since you should quickly discover that representing the RGB axes as full 0 to 255 bins (a total of 256*256*256 = 16M bins) is not only costly, it also makes image comparison impossible, since the 5K pixels in a given image usually won’t fall into many of the same bins that the 5K pixels of the other image do. Second, you must decide if you want to ignore all "black" pixels, and how and when to do so, since all of the images, except for a few of the distractors, have a "black" background. This can cause false matches simply on the basis of how much background similarity there is. (Really, this is effectively a measure of the size of the object(s) in the foreground.) Note that you have to decide exactly what "black" means.

For each of the 40 images, you must return the three images most *like* it in color distribution and the three images most *unlike* it. You should output these as display of image septuples, that is, in each of 40 rows, there should be seven images. The first is the image iNN. Then its most similar image, then its next most similar image, the the third most similar image. Then the third most dissimilar image, then the second most dissimilar image, then the most dissimilar image. (You can do this in gray scale if you wish.) To help the grader, in this step you should also indicate the identifying number of the image next to each image.

Then, display which group of four images are the most alike throughout the entire database, and which group of four images are the most unlike. Note that your output will strongly depend on your two decisions about histogram space and background, so you must carefully document your code as to what you decided and why. You will also have to decide and document what it means for four images to be alike or un-alike, since all that you will be computing is simple pairwise image comparisons.

Please note that if you use any black box algorithms, you much document why you selected them and how you tuned them.

2 Step 2 (5 points): Gross texture matching.

Write an algorithm which first converts the color images into black and white ones (that's easy: (R+G+B)/3) and then measures their gray scale textural similarity. There are many ways to do this, but you should use the following, which measures "edginess distribution".

First, create from each of the gray scale images a "Laplacian" image. To do this for an image, create a new image in which each pixel's value is equal to eight times the original image's pixel value, less the sum of the eight neighbors of the original image's pixel. What this does is to find local changes. For example, if the image is smooth in color in a region, then all pixels are more or less equal in value, and this results in a value near zero. Even if there is a slow steady increase in image intensity in some direction, it is not hard to show that this too results in a value near zero. But if there is an image edge or image texture, then the value departs from zero, either positively or negatively.

Now, you can form a one-dimensional histogram of these edginess values. Smooth imagery will have a lot of values near zero. Textured imagery will have a relative abundance of values that are highly positive and highly negative. A textured image that is within a black background will have some zero-like values (from the black) and some high values (from the texture). It is up to you to decide how to quantize this histogram, similar to what you did in Step 1, and, again, how to handle the background. Use the normalized L1 comparison again to compute textural similarity, so that again all pairwise comparisons lie in the unit interval, [0, 1], meaning [as different as possible, identical].

For each of the 40 images, you must again return seven images as in Step 1, and then the two groups-of-four as in Step 1, except based on this texture measure. Note again that your output will strongly depend on your quantization and background decisions, so you must carefully document your code as to what you decided and why.

Please note that if you use any black box algorithms, you much document why you selected them and how you tuned them.

3 Step 3 (5 points): Combine similarities, and cluster.

First, combine your measures of color similarity (C) and texture similarity (T) in a reasonable way. You already know that both similarities lie in the interval [0,1]. One reasonable way to combine them is by way of a linear sum: S = r*T + (1-r)*C, where r itself is in the interval [0,1]. If r=0, this is a pure color comparison, and if r=1, a pure texture comparison. It is up to you to pick (and defend!) a "good" value of r, based on what you see in the imagery.

Second, cluster the images in the following way. Each of the 40 images starts off in its own cluster. Compare each pair of images to find the "nearest" pair according to your measure S, where distance is defined to be 1-S, then combine these two nearest to a single new cluster. At the end of this first step, you now have 39 clusters: 38 singletons and 1 pair. Then find the two nearest clusters, again, except now you have to be able to compare clusters that have more than one image in them.

Define "nearness" to be the closest that *the farthest* image in one cluster comes to *the farthest* image in the second cluster. That is, closeness of clusters is the worst closeness of all their individual pairwise comparisons. (In clustering theory, this is called "complete link".) For example, if your first cluster consisted of two images, one that was mostly pink and one that nearly pure white, then the distance of this cluster from an image that was nearly pure black would be determined by the nearly pure white one, since that is image that is the most different from the nearly pure black one. So at the end of this second step, you will now have 38 clusters, which might consist of 37 singletons and 1 triple, or 36 singletons and 2 pairs. Repeat this process, which decreases the total number of clusters by 1 at each step until you have exactly 7 clusters. Then, display them.

For example, suppose the images image1, image2, image3, and image4 have the following total distance measures, D = 1-S, between image pairs.

```
0.0  0.5  0.1  0.2
0.5  0.0  0.4  0.6
0.1  0.4  0.0  0.3
0.2  0.6  0.3  0.0
```

The clustering process goes:

```
{{1}, {2}, {3}, {4}}
{{1,3}, {2}, {4}}
{{1,3,4}, {2}}
{{1,2,3,4}}
```

Clustering code is available in many places on the internet. But if you use someone else’s package, or translate a package into the language you are using, you must document your borrowing.

Now, having written similarity based on "complete link", there is a second way of defining cluster similarities that is related, called "single link". The definition of "nearness" is changed to be: the closest that *the nearest* image in one cluster comes to *the nearest* image in the second cluster.

Do the clustering again using single link, stop at y clusters, and display the results. It is not hard to show that complete link at each step gives clusters that are cliques (that is, in every cluster, every image is related to every other image), whereas single link at each step approximates the stages of growth of a minimum spanning tree (that is, in every cluster, each image is very close to at least one other image, but not necessarily to any of the others). This means the results will vary. So, examine your two results and compare them; usually you should find that the single link method gives worse results.

4 Step 4 (5 points): Creative step.

This step will depend on having three good patient friends.

Each friend does the same three things; basically, they all do something like the previous three steps. This will take about a half hour or more for each friend, so pick wisely. Note that you are using three friends simply for statistical purposes. It would be better if you had all your Facebook friends pitch in, but we'll just go with three friends, since this is a learning exercise, not a real psychology experiment. Note also that you can start the assignment here even before you design and code your system, since what your friends do is completely independent of what your system will do.

First, show a friend the images and give him or her some way to record the answers. For the first task, ask them for each of the 40 images to record the number of the image that is most like it and the number of the image that is most unlike it in color. This gives you a total of 80 numbers. For the second task, do the same, except for just texture. For the third task, ask the friend to group the images into exactly 7 clusters, however they think best. For this task, it is probably a good idea to print out copies of the images and cut them up, so they are about the size of matchbooks,and can be spatially arranged on a table. (If you want, you can do a complete user interface for this, where the friend can drag images around on a screen, but that is going to be a lot more work, particularly if the interface also uses visual gestures for control!) When the friend is done, record what is in each group.

Now for the creativity step. For each of the three tasks, write an algorithm that returns a value measuring how similar your output was to each friend. Defend your measure of task similarity, and discuss the results. If you want, you can do the user studies first, then go back and adjust your three Steps to get better agreement with one or all of your friends.

This Step is open-ended, and you will not get extra credit for having (or simulating) friends who happen to agree well with your system. But rather, you will get credit for how reasonable your definition of system performance is, and for your interpretation of what the results tell you about your system. For example, one easy way to define performance for the first and second tasks, given a specific friend, is: the system gets one point for each time at least one of your friend's answers for image iNN is in the six images your system picked for image iNN. Then, total score for that task and that friend is points/40, and overall score for the system on that task is the average of these three friends' total scores. But there are other ways, too.

For the third task, it gets a bit more tricky and more computer science-y. You have to determine how to measure the similarity between two partitions of a set. This is tricky, even if the number of subsets for both partitions is the same (here, it is 7). This can be done in a number of ways, and in fact, it is often done in more than one way, since one number is not quite right--just as "precision" and "recall" are usually used together for measurements in information retrieval.

So, for example, if there are four images called a, b, c, d, there are exactly 7 partitions of this set into 2 subsets: a|bcd, b|acd, c|abd, d|abc, ab|cd, ac|bd, and ad|bc. You probably want some way to say that a|bcd and ab|cd are "closer" than a|bcd and b|acd are. Again, if you find some code or modify some code, you still have to give credit, and explain why you chose it for your measurement.

5 Checklist of deliverables

1. Your assignment is to be submitted to Courseworks electronically as a single pdf.
2. Your assignment consists first of a writeup with examples, then a listing of all the code used. Any code that you did not write yourself has to be documented with a statement about its source and an explanation of why you have permission to use it.
3. For all steps, your writeup explains: what design choices you made and why, what algorithms you used, what you observed in the output, and how well you believe your design worked.
4. For all steps, your examples show thumbnails and image numbers together, in order to display the results. For step 4, this can get a bit tedious, so you might want to write a general program that creates pictorial output given image numbers.
5. For step 1, you must say what you did about bin size and background black, and some commentary about your results. Your output is a table that is 40 by 7 images, plus most similar group-of-four plus most dissimilar group-of-four, plus an explanation on how you defined these concepts for a group-of-four.
6. For step 2, you must say what your texture measure is, how you binned it, what you did about black, and some commentary about your results. Your output is a table like that in Step 1, and again an explanation.
7. For step 3, you must say what your linear sum is and how you decided to set it, how you make your clusters, some commentary about your results, and what you notice about complete versus single link. Your output is two clusterings, each with exactly 7 clusters.
8. For step 4, you must show the results of your three friends on each task, what and how you determined to be a good measure of comparison between your system's results and that of your friends, and a discussion of what your performance results mean about your system (or about your friends!!),
9. Please use Piazza, early and often. It is a good way to explore and learn, from and with the rest of the class. And, you can use it anonymously, so you don't have to be afraid of looking stupid or unprepared. But, please start the assignment early, so that you can give yourself--and your friends--enough time to do so.