COMS W4735 Assignment 2b
Crowd-sourcing due: Feb. 18, 2023 @ 2:25pm
Experiments and write-up due: Mar. 9, 2023 @ 2:25pm

This document describes the System steps for Ass. 2.

Data for this assignment requires three files, two which are provided, and one of which you have already created as part of Ass.2a crowd-sourcing.

Here are the file descriptions

1. Images.zip

   A directory of actual images of fruits and vegetables, plus some distractors, are available on Courseworks in both JPG format and in PPM format. They all are of the same small size, they all use the same black backgrounds, and they all have been professionally created to have good visual composition. 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, since some display packages require JPGs. However, depending on which image processing package you use, the PPM files may be much easier to manipulate within code, since they follow the simple PPM P6 color byte standard, and give exact integer values, as follows.

   (a) 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.

   (b) 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.

   (c) 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. These are small by smartphone standards, but big enough to be challenging for visual interface purposes.

   (d) 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 of precision.

   (e) 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 a total of 89 * 3 bytes encoded in binary, not as a readable string.

   (f) 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.

2. Crowd.txt

   Once people have submitted their files for the crowd-sourcing experiment in Ass. 2a, the files will be compiled into a 40 x 40 integer-valued matrix, in a text file called Crowd.txt on Courseworks, in the same directory as the image files. Each entry. $Crowd(q, t)$–that is, the "query row" q and the "target column" t of the text–aggregates the Crowd's total "Borda counts", measuring how well people think a given target image t is a match for the query image q. See Ass. 2a for more detail.

3. MyPreferences.txt

   This file was created and uploaded by you, and it holds *your own* first, second, and third place votes for each of the 40 images. This also is described in Ass. 2a.

   Since this is part of Ass. 2b, you need to make this file part of what you submit.

1 Step 1 (5 points): Color distance

This first Step in your Content-Based Image Retrieval system is about the most important image similarity cue, color. But first, you need to know some details about the imagery.

1.1 Some motivation about color

Although humans tend to perceive color according to hue, saturation, and value (HSV, "cortex" dimensions), this experiment will give a reasonable approximation by using the raw red, green, and blue camera data (RGB, "retina" dimensions).

1.2 Color computation

This Step asks you to write an algorithm that compares two images based on simple measures of their color across the total image, and to evaluate the quality of your algorithm by comparing its responses to what is in Crowd.txt.

The explanation for this step is a bit lengthy. but it gives some common procedures that are used in all five steps, which will then be simply referenced rather than repeated in the later steps.

First, you need to generate each image's three dimensional color histogram, as explained in class. Then, you compare two images' histograms by using the "normalized L1 distance", as explained in class. This leads to a value in the interval $[0, 1]$, where 0 means no distance at all (Image1 is really just the same pixels as Image2, but possibly spatially scrambled), and where 1 means perfect distance (there are no shared colors at all between the images).

But, as explained in class, you *must decide* how rich or sparse to make your histogram space, since you will quickly discover that representing the RGB axes with 255 partitions each (a total of 256 * 256 * 256 = 16M bins) is not only costly, it also makes image comparison impossible. This is because the 5K pixels (60 * 89 = 5,340) of Image1 usually then won't fall into many of the exact same bins that the 5K pixels of Image2 will.

One way to approach this is to "round off" the values of each coordinate by using only some of their leading bits. So, for example, if you just used the leftmost bit of each of the three colors, you

would be able to recognize eight colors ($8 = 2^{(1+1+1)}$), since there would be only two partitions of each axis. A pixel then is either red enough or not, green enough or not, blue enough or not. The eight colors then are black, red, green, blue, yellow, cyan, magenta, and white. This too is not useful, as eight colors are too few to make good distinctions. Using the leftmost two bits, though, gives 64 colors ($64 = 2^{2+2+2}$), which is about at the level of people's ability to name them, but experimentation will show that it is not quite good enough, either, since many of those 64 colors won't appear at all.

Note that you don't have to choose the same number of bits for all three colors, particularly since it is well-known that people can't see differences in blue as well as they can see differences in green (red is somewhere in the middle). In general, you would then have $2^{(R+G+B)}$ bins, if R, G, and B represent the number of leading bits you use for each color.

So, your job is to find a "Goldilocks" choice for the partition of each axis. You wnat your color similarity model to end up most like that which matches the crowdsource data in Crowd.txt.

Some details:

- The normalized L1 distance is given by:

$$L1(ImageC1, ImageC2) = \sum_{(r,g,b)} |ImageC1(r,g,b) - ImageC2(r,g,b)| \,/(2{*}rows{*}cols)$$

  where $rows$ and $cols$ in this assignment are 60 and 89, respectively.

- For each of the 40 query images, you will need to find the three target images most *like* it in color distribution, that is, the three target images that have the smallest normalized L1 distance to the query image. Then, to get a *score* for each query image q, add up the counts given in Crowd.txt for each of the three target images t_1, t_2, t_3 that your algorithm selects. Note that you should treat this scoring as just a simple unweighted sum; you don't have to do the Borda thing:

$$Score(q) = Crowd(q, t_1) + Crowd(q, t_2) + Crowd(q, t_3)$$

1.3 Visualization and performance evaluation

To explore and defend both accuracy and user satisfaction, provide the following:

1. *System versus crowd preferences, using scores*: To show the accuracy of your system, output your results for this Step as a display of 40 rows of four small (thumbnail) images each, in the order q, t_1, t_2, t_3. Label each query image q with its identifying number (01 to 40). Label each target image t with both its number and its $Crowd(q, t)$ count. Label each row with its $Score(q)$. Label the entire display with the grand total sum of all the $Scores$. These are the major results of this Step concerning crowd-based accuracy. Note that you have three "levels" of scoring: one score for each target, one score for each query, and one score for the system.

2. *System versus personal preferences, using set intersection* To get a sense of how much your own system would make you happy as a user, for each row, simply count *how many* images are in common between the three best targets that your system computed, and the three targets that you had submitted as your personal ground truth. For each query image q, this will be a number from 0 to 3. You don't have to display these individual numbers. But just add these up as a grand intersection value, from 0 to 120, and report. This is the major result of this Step concerning (your) user satisfaction.

3. *Justification* Since your output will strongly depend on your decisions, you must document them, both in your write-up and *in your code*. Talk abut what you decided to use, and why. One way to do this is to show the results of some of your explorations: what didn't work until you found something that did. Please note that your client (the graders) will be suspicious if you only give a single result and proclaim it "the best".

   Please also note that if you use any open source computer vision algorithms, you must document *in your code* where they came from, why you selected them, and how you tuned them.

2 Step 2 (4 points): Texture distance

This Step in your Content-Based Image Retrieval system is about another important image similarity cue: texture.

2.1 Some motivation about texture

There is good psychophysical evidence that texture is mostly conveyed by intensity variations. So, first convert all the full color (24 bit) images to their black and white intensity ("gray scale") counterparts. This is easy to do: $I = (R+G+B)/3$; note that I is now just 8 bits. Then, you need to compute the textural distance between two gray scale images. There are many ways to do this, but you should use the following, which is computes an "edginess distribution", and is well-suited to these images.

2.2 Texture computation

First, create from each of the gray scale images their "Laplacian" image. To do this, create a new image in which each pixel's Laplacian 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. It looks roughly like this, where you multiply gray scale values by the given integers, and add up the result:

```
1   1  1
1  -8  1
1   1  1
```

What this does is it finds 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, as if it were a color. Smooth imagery will have a lot of values near zero. Textured imagery will have a relative abundance of values that are highly positive and, necessarily, also highly negative. It is not hard to show that the values will tend to be symmetrically distributed on either side of zero. So you can simplify the histogram by collecting only the absolute values of the Laplacian pixels.

You *must decide* how to quantize this histogram, like what you did in Step 1. Use a properly modified version of the normalized L1 distance again to compute textural distance, so that again all pairwise image comparisons lie in the unit interval, $[0, 1]$, where 0 means no distance at all (ImageI1 is exactly as "edgy" as ImageI2), and where 1 means perfect distance (one image is perfectly smooth but the other is violently changing all over).

2.3 Visualization and performance evaluation

Same as in Step 1: rows of images and scores for the Crowd, and a single happiness value for you, and your justification.

3 Step 3 (2 points): Shape distance: fore- vs. back-ground

This Step in your Content-Based Image Retrieval system is about yet another very important image similarity cue, shape, but one which unfortunately has been embarrassingly neglected in computer vision.

3.1 Some motivation about shape

The world is three-dimensional, but there is good evidence that we recognize objects by matching their two-dimensional images against a collection of previously seen two-dimensional images. Because of this, boundaries are very important. This assignment therefore will cheat by noting that the images in this assignment have been deliberately selected so that they are well composed against a black background, and therefore the black versus non-black boundary gives a fairly accurate object silhouette. This is especially easy since many of the images have been cropped to display the object.in the exact center of the image.

So, first convert the intensity images you created in Step 2 to binary black (background) versus white (foreground) objects. This is easy to do by saying $N = I > m$, that is, the value of N, which represents the biNary object, is determined by whether it is brighter than some magic value representing "black", which you *must decide*. Note that too small of an m makes everything foreground, and too large of an m makes everything background.

3.2 Foreground-background computation

At this point, you have Color images (24 bits/pixel or less) used for color, Intensity images (8 bits/pixel) used for texture, and biNary images (1 bit/pixel), used for shape. Although it would be

more realistic to trace the black versus white border (like you probably did in Ass. 1) non-black border, there are simpler ways that compute boundary overlap. And, of course, some of the images don't really have a border at all.

You should use the following, which is computes the "normalized overlap distance", and is reasonable approximation given the composition of these images. It is given by:

$$Overlap(ImageN1, ImageN2) = \sum_{(x,y)} |ImageN1(x,y) \neq ImageN2(x,y)| \,/(rows * cols)$$

where, again, $rows$ and $cols$ in this assignment are 60 and 89, respectively.

This basically counts the number of pixels at which the images disagree on being foreground versus background. Shapes that have similar sizes and boundaries will have relatively few such pixels. This shape distance also lies in the unit interval, $[0, 1]$, where 0 means no distance at all (Image1 is has exactly the same object as Image2), and where 1 means perfect distance (one image is the binary negative of the other).

3.3 Visualization and performance evaluation

Same as in Step 1: rows of images and scores for the Crowd, and a single happiness value for you, and your justification.

4 Step 4 (2 points): Shape distance: symmetry

This Step in your Content-Based Image Retrieval system is about a curiously human capability for shape, that of symmetry detection.

4.1 Some further motivation about shape

Humans have a strong ability to recognize left-right symmetry about a vertical axis. The speculation is that it is primarily useful for face recognition: you can easily recognize someone from only their left or right half of their face. Further speculation is that departures from symmetry are often associated with illness. How often have you compared something on the left half of your body to something on the right side, before asking: "Do you think this is swollen?". This health test may also give cues to the safety of dealing with, or even eating, plants and animals.

The vertical axis is ultimately determined from the ever-present force of gravity. So, conversely, humans aren't so good at horizontal symmetry, though. The speculation is that there has never been a good need for it. So for this step, we will only do symmetry about a vertical axis.

4.2 Symmetry computation

First, start with the same biNary images for this shape step as you used in the previous step. Except now, you should "fold the image in half", and see how closely the two sides agree.

Note that each image has 89 columns. Using indexing starting from 1, the left half of an image consists of columns 1 through 44, and the right half of columns 46 to 89. (You can ignore

column 45.). Form a normalized symmetry score in the following way. For each column, compare it against its symmetric twin (column 1 twins with column 89, column 2 twins with column 88, . . . column 44 twins with column 46). Each twin computes how many pixels agree, from 0 to 60. Find the total symmetric agreement over the full image, and then normalize it in the usual way so that you get a distance again in the range [0, 1]. Except here, a distance of 0 means perfect symmetry and a distance of 1 means anti-symmetry (for example, the left half is all black and the right half is all white).

But now note that this Step depends on your definition of "black" that you used in the previous step. But what is good for boundary might not be so good for symmetry. So maybe you might want to do use a different "black". If you do, make sure you justify it.

4.3 Visualization and performance evaluation

Same as in Step 1: rows of images and scores for the Crowd, and a single happiness value for you, and your justification, especially about "black".

5 Step 5 (3 points): Overall distance

5.1 Some motivation about "gestalt" perception

Psychologists know that the experience of human perception is highly non-linear. But you will ignore their wisdom by simply doing what you can, rather that by doing what you should. You will combine the four unit distances in to one overall system distance by doing a linear combination instead.

5.2 Gestalt computation

Combine your measures of color distance (C), texture distance (T), shape distance (S), and symmetrY distance (Y) in a reasonable way. You already know that these distances all lie in the interval $[0, 1]$. One reasonable way to combine them is by way of a linear sum: $P = a*C+b*T+c*S+d*Y$ where a, b, c, and d are all non-negative, and $a+b+c+d = 1$. This vector (a, b, c, d) is called the "standard simplex".

Note that if $a = 1$, this is a pure color comparison; if $b = 1$ this is a pure texture comparison, if $c = 1$ this is a pure shape comparison, and if $d = 1$ this is a pure symmetry comparison. Further, since all of C, T, S, and Y are in the interval [0, 1] it is not hard to show that this linear sum (technically, a "dot product") also is in the same interval.

You *must decide* and justify a "good" value for (a, b, c, d), based on the ground truth in Crowd.txt. Please note that even if you have perfectly optimized your C, T, S, and Y distances, this does not guarantee that combining them will lead to optimal total system performance. This is because these different kinds of visual distances may interact in funny ways; for example, small textured shapes may not be readily distinguishable by their color, etc. On the other hand, it is not hard to show by a simple example that there are cases where this dot product increases accuracy.

One good heuristic here is to start with a weighting vector that proportionately rewards each unit according to its individual success.

5.3 Visualization and performance evaluation

Same as in Step 1: rows of images and scores for the Crowd, and a single happiness value for you, and your justification.

6 Step 6 (1 points): Crowd versus you

This section attempts to maximize your own happiness.by "personalizing" your system. Replace Crowd.txt with a very sparse 40x40 matrix that was crowd-sourced based only on you. That is, each row has a single 3, a single 2, and a single 1.

Now, redo Step 5–but keeping your decisions in Steps 1 through 4 the same–by simply searching for the values of (a, b, c, d) that make *you* happiest. Report these values, and comment why you think they differ from what you got for Step 5.

7 Checklist of deliverables

Your assignment is to be submitted to Courseworks electronically as a single pdf.

Your assignment consists first of a write-up 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.

For all steps, your write-up 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.

For all steps, your examples should show thumbnails with image numbers nearby, in order to display the results accessibly.

1. For "Step 0", make sure you include your MyPreferences.txt file.

2. For Step 1, say what you did to find a "good" three-dimensional histogram bin definition. Show a 40 by 4 image display of q, t_1, t_2, t_3, t_{40}, plus individual target scores, row scores, and grand total score. Show your personal happiness score. Justify how you got your results.

3. For Step 2, say similar things to Step 1, except that the histogram here is one-dimensional. Same visualization, happiness, and justification requirements.

4. For Step 3, say similar things to Step 1, except that the search here is for a good definition of "black". Same visualization, happiness, and justification requirements.

5. For Step 4, say similar things to Step 1, except that the search here is for a different definition of "black". Same visualization, happiness, and justification requirements.

6. For Step 5, say similar things to Step 1, except that the search here is for a good simplex vector. Same visualization, happiness, and justification requirements.

7. For Step 6, report the new weighting vector, and discuss the improvement that personalization gives.