EE E6882 SVIA: Homework 1

Due on October 1, 2007

Shih-Fu Chang, Lexing Xie Monday 4:10-6:30

prepared by Eric Zavesky

1 Background

As the number of images and videos continues to increase, we will see more intelligent forms of image search applications develop. In this homework assignment, you will create a basic system that performs content-based image retrieval (CBIR); you must rank a set of images given a single query image. This homework will expose you to three essential tasks for indexing and searching video: feature extraction, distance metric choice, and performance analysis. You will be provided with skeleton sample code developed in Matlab and there are two opportunities to obtain bonus points towards your final course grade.

1.1 Dataset & Feature Extraction

This assignment uses a subset of images derived from a work that analyzes the performance of different low-level features for automatically annotating consumer photos[1]. This image set is derived from downloads from flickr[2] and Yahoo![3] so it should acclimate you to the challenges of an image search system. Your CBIR system will be searching the dataset for the best match to a small number of query images; dataset examples are shown in figure 1.

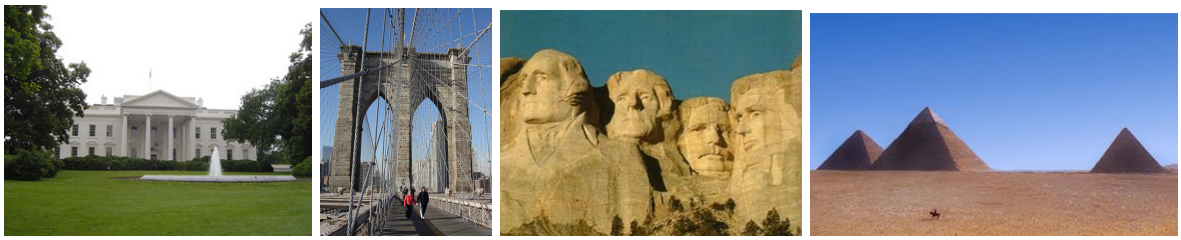

Figure 1: Example consumer photos of famous locations: the White House, the Brooklyn Bridge, Mount Rushmore, and the Pyramids at Giza.

Feature extraction is the process of analyzing and computing numerical representations of an image. Common low-level features used in the image processing community are color moments, edge direction histograms, Gabor or wavelet texture, and shape information.

To expedite system development, we have pre-computed low-level color moment and texture features and provide these files in the CourseWorks system. Both feature sets are formatted in a simple space delimited format (shown below), so you can easily import these into any programming environment of your choice.

```
<file name 1> <feature1> <feature2> <feature3> ... <feature N>
...
<file name M> <feature1> <feature2> <feature3> ... <feature N>
```

Figure 2: Example feature format for pre-computed features.

1.1.1 Dataset description

There are common themes among images in this dataset, some of which are shown in figure 1. We chose sets of images with distinct appearances but are not exactly the same content. This diversity of images is what one might expect from a real world dataset that could be directly acquried from the internet. Your CBIR system will be searching for images that match four specific locations. Specifically in the ground truth

[1] Lyndon Kennedy, Shih-Fu Chang, Igor Kozintsev. To Search or To Label?: Predicting the Performance of Search-Based Automatic Image Classifiers. In Multimedia Information Retrieval Workshop (MIR), Santa Barbara, CA, USA, 2006.

[2] http://flickr.com/

[3] http://images.search.yahoo.com/

file (whose format described in figure 4) you will find "concept codes" with values 1000, 2000, 3000, and 4000 that represent Mount Rushmore, the Pyramids at Giza, the Brooklyn Bridge, and the White House respectively. Although these collections originated from automatic downloads, we chose images that have a similar appearance (i.e. color or structure) but simultaneously present a challenge for your CBIR system.

1.2 Distance Metrics

Distance (or conversely, similarity) metrics are a core idea for any CBIR system. The most simplistic similarity metric is the L1 metric, which is also known as Manhattan distance, block distance, and Euclidian distance. The L1 distance metric is defined as the sum of the absolute value of differences for each feature dimension (N) of two samples.

$$d(x, y) = \Sigma_{i=1}^{N} |x_i - y_i|$$

To use a distance metric in a CBIR system, compute the distance between the query image x, and all of the images in the dataset y. Then, rank (or order) the images in the dataset from lowest to highest distance to present results for the query.

In this assignment you will implement the L1 distance metric as a **baseline** performance indicator. This means that your systems performance should usually do **at least as well** as the baseline and hopefully better. You can find other distance metrics in the materials presented in class and are free to experiment with any of them.

1.3 Performance Evaluation

Reporting the performance of any CBIR system allows others researchers to compare it to their own system on the same data set. While many different performance metrics exist in the information retrieval community, we will focus on precision and recall.

Precision indicates how well a system can measure similarity between relevant and irrelevant samples. Precision is equal to the number of relevant items returned divided by the number of total items returned. Recall indicates how well a system can find all relevant instances of a single class when looking through an entire dataset. Finally because this a CBIR system (looking for best matches to a query) and not a classifier system (learning one model for all data), we will calculate mean precision and mean recall for reporting.

$$\begin{aligned} precision(x, y) &= \frac{count(relevant \cap retrieved)}{count(retrieved)} \\ recall(x, y) &= \frac{count(relevant \cap retrieved)}{count(relevant)} \\ mean\_precision(x, y) &= \Sigma_{i=1}^{N} precision(x_i, y) \\ mean\_recall(x, y) &= \Sigma_{i=1}^{N} recall(x_i, y) \end{aligned}$$

where x is the query image and y is the entire data set. Please note that you should report the *mean* precision over all query images for which there exists a match. For this dataset that means that you should calculate precision and recall for each query image independently and then find their mean values among aggregated concepts, defined in section 1.1.1.

Finally, to get get a better understanding of how your CBIR system works at different depths, you should compute and graph the mean precision at the following depths: 1, 2, 5,10, 25, 50, 100. This maximum depth requires a trivial change to your precision algorithm that limits the depth of your precision calculation. For example, the equation for precision above now becomes the following.

$$\begin{aligned} precision(x, y, D) &= \frac{count(relevant \cap top\_D\_retrieved)}{D} \\ recall(x, y, D) &= \frac{count(relevant \cap top\_D\_retrieved)}{count(relevant)} \end{aligned}$$

The graph generated should measure precision vs. recall at different depths and it should look something like figure 3. In this figure the different lines are the different concepts analyzed, the vertical axis is the precision of analysis, and the horizontal axis is the mean recall over the samples with known matches (described more in section 1.3.1). Please note that you should generate **unique** graphs for each experiment variation (i.e. changing the feature modality or distance metric).

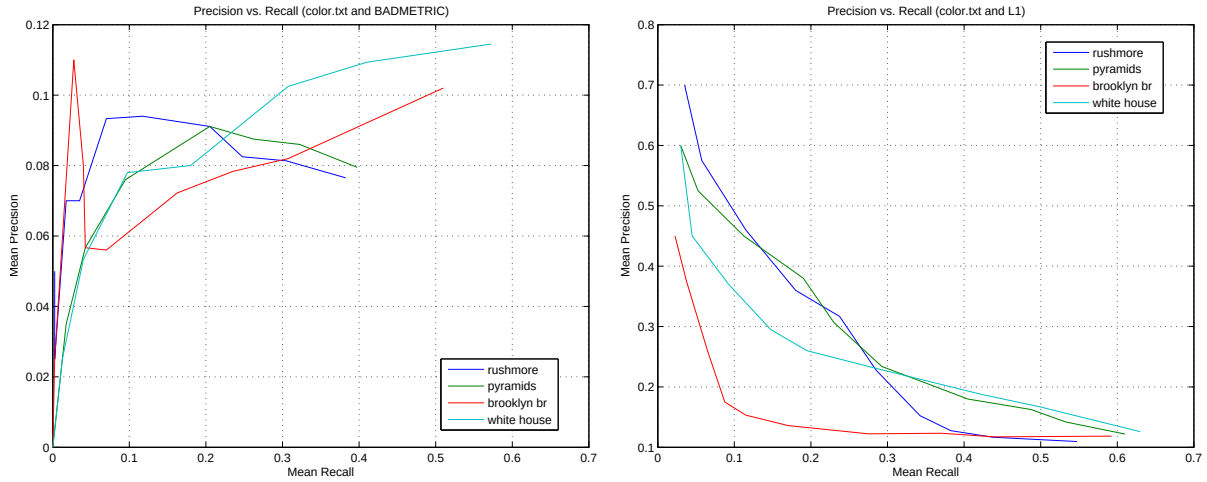

Figure 3: Example plot of mean precision across different CBIR system configurations using the color feature modality: left uses an arbitrary metric and right uses L1. Don't worry, these numbers do not reflect expectations for your own systems.

1.3.1 Ground truth

To evaluate the performance of a system you must have samples that have been labeled as relevant or irrelevant. This set of samples (and its labels) is often referred to as the *ground truth* or *golden standard* in different communities. A plain-text file will be provided in the format of figure 4 through CourseWorks that contains the ground truth for the given dataset.

```
<concept code 1> <query file name 1>
<concept code 2> <query file name 2>
...
<concept code M> < query file name M>

(Theoretical data example)
1000 005
3000 006      <-- note that only images with relevant concept labels are included
3000 017
3000 020
...
```

Figure 4: Example ground-truth format for evaluating CBIR system performance.

1.4 Downloading the data

All images, features, source code and even this document will be available via the CourseWorks site. Go to the "assignments" section of the CourseWorks site and you should find materials for homework 1 including one zip file containing images, source, and features and this document as a PDF file.

2 Bonus Opportunities

Bonus points will be awarded to your assignment for a deeper investigation of the CBIR topic.

2.1 Multi-modal Fusion

Multi-modal fusion is a highly researched topic that seeks to leverage the performance available from several individual feature modalities (i.e. color, texture, shape) by intelligently combining them. One simple multi-modal fusion technique is to use score averaging between different modalities. Assume that you have several different computable feature modalities m for the images in the dataset, a modality might consist of color, texture, or even text low-level features.

$$\text{modalities} = \{f_1, f_2, ...f_M\} \text{ where each } f_m \text{ can be a vector of features described in section 1.1}$$

Using a distance metric, you can compute scores of a modality m for a single query image x to all images y in the dataset.

$$scores_m = d(x_m, y_{im})$$

Finally, fuse the scores from multiple modalities by using some process, like $min(...)$, $max(...)$, or even $average(...)$. Ideally the final fusion process will provide the best score for each image match because it uses the computed scores from several diverse modalities.

$$scores_{final} = FUSION(scores_m, ..., scores_M)$$

For a bonus point on this assignment, you can provide performance of one run using multi-modal fusion along with a description of your fusion technique and why you chose this technique.

2.2 Class Competition (best CBIR system)

The last part of any CBIR system is the presentation of results to a user. While this topic will not be addressed in this assignment, it is important to keep it in mind when designing your CBIR system. For example, some typical topics to consider are: do you need to pre-compute results or does your system work in real-time, can the results of your system easily be delivered to another processing unit, and how many results should your system present/calculate so as to not overwhelm the end user?

For this assignment the best 100 results should be in a simple plain-text format so system performance can be easily assessed. This format consists of three entries per line; the file number of the query image, the file number of the dataset image, and the of this pair among images in the set, as shown in figure 5 below.

To give an objective assessment of each CBIR system, we will calculate the harmonic mean of the precision and recall averaged over all test images. In IR communities, this harmonic mean is referred to as the *F measure* and we are using F1. Using the formulas describe in section 1.3, F1 can be computed as described below. Results for each students CBIR system will be provided to them and the student with the highest F1 will be the class winner. *Please note: You may use the baseline approach (F1 with color and F1 with texture) for your submission, but every student will have access to these features, so you are encouraged to substitute your own, better approaches.* In the very unlikely event of a tie, the competition will be declared a friendly draw and no student will win the bonus points.

```
<query file name 1> <result file name> <result_rank 1 (best match)>
...
<query file name 1> <result file name> <result_rank 100 (worst match)>
...
<query file name M> <result file name> <rank of result (best to worst)>

(Theoretical data example)
200 101 1         (my best guess at matching query image 200)
200 102 2         (my second guess at matching query image 200)
200 105 3
...
201 104 100       (my worst/last guess at matching image 201)
```

Figure 5: Example result format to be submitted for performance evaluation.

$$all\_depth = \{1, ..., 100\}$$
$$mean\_average\_precision = \Sigma_{i=d}^{D} mean\_precision(x, y, all\_depth(d))$$
$$mean\_average\_recall = \Sigma_{i=d}^{D} mean\_recall(x, y, all\_depth(d))$$
$$F1 = \frac{2 * mean\_average\_precision * mean\_average\_recall}{mean\_average\_precision + mean\_average\_recall}$$

3 Grading Checkpoints

As part of the research process, individuals are required to rigorously discuss their assertions and insights about their findings. In a write-up in Microsoft Word or PDF format, please address each of the checkpoints outlined below.

- Discussion (written with supporting graphs if necessary)

  1. Inspect the images that you downloaded. What common trends or patterns do you observe that might be exploited among image features? (1 point)
  2. Derive two new features to process the images and analyze the performance of this feature. Only one of your features can be a form of normalizing an existing pre-computed feature. How does your feature capitalize on data that others did not? How can you leverage your feature in a large dataset (i.e. build a codebook or optimize its extraction). (1 point)
  3. Visually inspect results of your system, Without looking at the performance, what are your impressions of the returned results? Which features did you expect to perform and did they do so? Can you learn anything by also looking at the worst matches? Why or why not? (1 point)
  4. Now, also using empirical results as your evidence, discuss the strengths and weaknesses of the distance metrics used; at least the L1 metric (described above) and one other chosen by you. Why did you choose this secondary metric? (2 points)
  5. Inspecting your performance graphs, what does this indicate about CBIR operations at different depths? What do you expect to happen if the size (number of images) was increased in terms of inter-class confusion and feature discriminability? (2 points)

- Data files

  1. Graphical plots (like figure 3) of different CBIR permutations: at least two distance metrics (only one can be L1, as it is provided) at least two feature modalities (only one can be one of the pre-computed modalities). Please provide visual examples in your report of the top 3 matches

of the following imges: 080, 137, 141, and 192, which are the example images from figure 1. (1 point)

2. Your complete source code for this assignment with adequate documentation. (2 points)

- Bonus (optional)

1. Use the features for the test set and run your algorithm on the 40 query images in this text file. The best performance from the entire class will win these points. (+2 points)
2. Create a system that performs multi-modal fusion of scores that performs **better or roughly equal to** any single metric or modality alone. Clearly describe your reasoning for choosing this fusion system and explicitly document the formula or algorithm you used along with graphs of its performance like figure 3. (+1 point)

3.1 Submitting your work

To avoid the problems associated with email (large files, faulty servers, etc), we will only accept homework via the CourseWorks site. You can easily upload your prepared materials (the report, homework source code, and optional result file for the competition) into the "class files" section under "homework 1 submission" of the CourseWorks site. If you need technical assistance, slides have been prepared demonstrating how to do this and are available under the discussion topic called "How do I post my homework?" on CourseWorks.

3.2 Due-date extension for presenters

If you are a presenter in the first group of papers (October 8, 2007), you are given an extra two weeks for homework preparation; the homework will be due *October 15, 2007*. This extra time is provided so that you can focus on a rigorous preparation and discussion for your selected paper.

4 Skeleton example code in Matlab

This skeleton script is fully functional and is provided with the data to download from CoureWorks. Included in the script are methods to load data, perform an example feature extraction, compute a distance metric, calculate and plot performance, and prepare results for the class competition.

```
function run_cbir(imFeature);
% RUN_CBIR
%
% Example skeleton code written for EE6882, SVIA, 2007
% Columbia University, Prof. Shih-Fu Chang
% Written by Eric Zavesky, 09/07

% here are the settings for our experiment
isCompetitionMode = 0;                  % set to 1 and change feature modality for competition mode
distance_metric = 'BADMETRIC';     % 'L1' or 'BADMETRIC' or ?student provided
ground_truth = 'ground_truth.txt';
feature_modality = 'texture.txt'; % 'color.txt' or 'texture.txt' or 'GRAY' or student
doPrintSave = 0;

% what are the names of the files we're dealing with?
%   simply they are 000.jpg to 199.jpg inclusive, so we can make the
%   list of filenames with the command below.
imIdDatasetKnown = [0:199];

% load the ground truth
[imIdQuery, imConceptCode] = load_ground_truth(ground_truth);

% load the features or take what was passed in...
if (~exist('imFeature', 'var') || isempty(imFeature))
    imFeature = get_features(feature_modality, imIdDatasetKnown);
end;

% are we just doing the normal experiment or in competition mode?
if (isCompetitionMode)
    test_truth = 'ground_truth_test.txt';
    test_feature_modality = 'texture_test.txt'; % only 'color.txt' or 'texture.txt'

    % load the ground truth
    [imIdQuery, imConceptCode] = load_ground_truth(test_truth);
    % load the features or take what was passed in...
    imFeatureTest = get_features(test_feature_modality, imIdQuery);

    % combine the features with our normal features so the distance function works
    imFeature = [imFeature; imFeatureTest];
end;

% compute the scores for each query image (ROW) and database image (COLUMN)
imCbirScores = compute_image_scores(...
                distance_metric, imIdQuery, imIdDatasetKnown, imFeature);
finalScores = imCbirScores;

% PERFORM FEATURE FUSION HERE for BONUS POINTS!
% load OTHER features for different modalities?
% [imIdDataset, imFeature] = get_features('something new', imIdDatasetKnown);
% imCbirScores2 = compute_image_scores(...
%                 distance_metric, imIdQuery, imIdDatasetKnown, imFeature);
% finalScores = fuse_scores(imCbirScores, imCbirScores2);

% convert the scores to ranked lists
imCbirRanks = convert_to_ranked_list(finalScores, imIdDatasetKnown);

% are we just doing the normal experiment or in competition mode?
if (~isCompetitionMode)

    % close all existing figures
    close all;

    % draw out and save the requested images
    imDraw = [80, 137, 141, 192];
    for idxDraw=1:length(imDraw)
        imIdx = find(imIdQuery==imDraw(idxDraw));
        drawmatches('images', [imDraw(idxDraw) imCbirRanks(imIdx,1:3)]);
        % save the plot to file
        outputFile = sprintf('image%03d_%s_%s', ...
                     imDraw(idxDraw), feature_modality, distance_metric);
        outputFile = regexprep(outputFile, '[\/.-()]+', '_');
        if (doPrintSave)
            print(gcf, '-dpdf', [outputFile '.pdf']);
        end;
    end;

    % first, find the unique Concept codes, because we're aggregating by them
    uniqueConceptCodes = unique(imConceptCode(:,1));

    % define precision depths that we're interested in
    mandatoryDepths = [1, 2, 5, 10, 15, 25, 45, 60, 75, 100];
    foundPr = []; foundRe = [];

    for idxDepth =1:length(mandatoryDepths)
        currentDepth = mandatoryDepths(idxDepth);

        rowPr = []; rowRe = [];
        for idxConceptCode = 1:length(uniqueConceptCodes)
            % what's the current code (i.e. 1000, 2000, etc.)
            currentConceptCode = uniqueConceptCodes(idxConceptCode);

            % what are ALL of the possible images valid for this code?
            idxMatches = find(imConceptCode(:,1)==currentConceptCode);
            imIdMatches = imConceptCode(idxMatches,2);

            % compute the precision for all of these ranked lists where the
            %   query image came from one of the matches for this Concept code
            [newPr, newRe] = compute_PR(...
```

```
                imConceptCode(idxMatches,2), imCbirRanks(idxMatches,:), currentDepth);
            % add on the precision for this lcoation
            rowPr = [rowPr mean(newPr)];
            rowRe = [rowRe mean(newRe)];
        end;

        % append our mean precision to the kept vector for all depths
        foundPr = [foundPr; rowPr];
        foundRe = [foundRe; rowRe];
    end;

    % plot the results of this experiment in a neat graph
    figure;
    plot(foundRe, foundPr);
    xlabel('Mean Recall'); ylabel('Mean Precision'); grid on;
    %axis([0 1 0 1]);
    % build up our legend names
    legendSet = {};
    for currentCode=uniqueConceptCodes'
        switch(currentCode)
            case 1000
                legendSet = [legendSet; {'rushmore'}];
            case 2000
                legendSet = [legendSet; {'pyramids'}];
            case 3000
                legendSet = [legendSet; {'brooklyn_br'}];
            case 4000
                legendSet = [legendSet; {'white_house'}];
        end;
    end;
    legendSet = char(legendSet);
    legend(legendSet, 'Location','Best');

    title(sprintf('Precision vs. Recall (%s and %s)', ...
                  feature_modality, distance_metric));
    % save the plot to file
    outputFile = sprintf('%s_%s',feature_modality, distance_metric);
    % make a file easier to save
    outputFile = regexprep(outputFile, '[\.-()]+', '-');
    if (doPrintSave)
        print(gcf, '-dpdf', [outputFile '.pdf']);
        %print(gcf, '-dpng', [outputFile '.png']);
    end;

% is competition mode!
else
    write_test_submission_file('test_submission.txt', imIdQuery, imCbirRanks);
end;

disp('All done!');

%% -- sub functions listed below -- %%

% this sub-function loads the ground truth file
function [imIdQuery, imConceptCode] = load_ground_truth(ground_truth_filename)
    % matlab can easily read plain-text, space delineated files,
    %   no extra work needed here
    % [column1=Conceptid] [column2=queryid]
    fprintf('Loading GROUND TRUTH %s...\n', ground_truth_filename);
    [rawData] = load(ground_truth_filename);
    imIdQuery = rawData(:,2);
    imConceptCode = rawData;

% this sub-function loads the feature modality file
function [imFeature] = get_features(feature_modality, id_known_dataset)
    imIdDataset = [];   % not returned, but tracks what images were loaded

    % see which low-level feature modality we should load
    switch (feature_modality)
        case {'color.txt', 'color_test.txt'}
            fprintf('Loading COLOR_MOMENT_FEATURES: %s...\n', feature_modality);
            fprintf('  -- this might take a while!\n');
            [rawData] = load(feature_modality);
            imIdDataset = rawData(:,1);
            imFeature = rawData(:,2:end);

        case {'texture.txt', 'texture_test.txt'}
            fprintf('Loading TEXTURE_FEATURES: %s...\n', feature_modality);
            fprintf('  -- this might take a while!\n');
            [rawData] = load(feature_modality);
            imIdDataset = rawData(:,1);
            imFeature = rawData(:,2:end);

        otherwise
            fprintf('Computing custom features for images!\n');
            % we must load and compute fetures for each image...
            imIdDataset = id_known_dataset;
            imFeature = [];
            for idxKnown=1:length(id_known_dataset)
                % what is the filename of the current image?
                % NOTE: this code assumes you unpacked your images into
                %       a subdirectory called 'images'
                currentImageName = sprintf('%03d.jpg', id_known_dataset(idxKnown));
                currentImageName = ['images/' currentImageName];

                % first, load the current image with matlab's image load
                disp(['Processing: ' currentImageName]);
                imRgbData = imread(currentImageName);

                newVector = [];
                switch (feature_modality)
                    case 'GRAY'
                        % change the color depth to just gray images
                        if (size(imRgbData,3) == 3)
                            imGrayData = rgb2gray(imRgbData);
                        else
                            imGrayData = imRgbData;
```

```
                        end
                        % how many total pixels?
                        numPixels = size(imGrayData,1)*size(imGrayData,2);

                        % create three bins for a gray level histogram
                        % count the number of pixels with values below some
                        %   graylevel threshold...
                        % Our new feature vector for this image will look
                        % like this...
                        %   [ %pixel<0.33   %pixel>0.66     %pixel-otherwise
                        newHistogram = [...
                            length(find(imGrayData<0.33)) ...  % how many pixels
                            length(find(imGrayData>0.66))];
                        % compute "other" case between thresholds
                        newHistogram = [newHistogram ...
                            numPixels - (newHistogram(1)+newHistogram(2))];
                        % finally divide by total pixels to get a percentage
                        newVector = newHistogram / numPixels;

                    otherwise
                        % ENTER YOUR ALGORHTM HERE for BONUS POINTS!

                end;    % end secondary modality switch for REALTIME calculation

                % don't forget to add on our new feature vector
                imFeature = [imFeature; newVector];

            end;    % end iteration through provided images

    end;    % end primary modality switch

% compute the scores for each query image (ROW) against each database image (COL)
function [imCbirScores] = compute_image_scores(...
                        distance_metric, imIdQuery, imIdDataset, imFeature);
    imCbirScores = [];

    % we will return results for every query image...
    for idxQuery=1:length(imIdQuery)
        newScoreRow = [];       % will hold a new ROW of scores for the query image

        % NOTE: we need the +1 offset becase matlab counts from
        %       1 instead of 0 for matrix positions
        currentQueryImage = imIdQuery(idxQuery)+1;

        for idxDataset=1:length(imIdDataset)
            currentDatasetImage = imIdDataset(idxDataset)+1;

            % if we found ourself, just skip with a score of Infinity
            if (currentDatasetImage==currentQueryImage)
                newScoreRow = [newScoreRow Inf];
                continue;   % SKIP, go back to the FOR loop
            end;

            % see which distance metric we should use
            switch (distance_metric)
                case 'BADMETRIC'
                    % this is a very stupid metric that counts the number of
                    % feature bins GREATER minus the bins LESSER than
                    % the query image
                    numGreater = length(find(imFeature(currentQueryImage,:) ....
                                        > imFeature(currentDatasetImage,:)));
                    numLesser = length(find(imFeature(currentQueryImage,:) ....
                                        < imFeature(currentDatasetImage,:)));
                    % now normalize by the total number of bins
                    newScore = (numGreater-numLesser)...
                        /length(imFeature(currentQueryImage,:));
                    % save the new score
                    newScoreRow = [newScoreRow newScore];

                case 'L1'
                    % compute L1 as defined in homework
                    newScore = sum(abs(imFeature(currentQueryImage,:) ....
                                    -imFeature(currentDatasetImage,:)));
                    % save the new score
                    newScoreRow = [newScoreRow newScore];

                otherwise
                    % ENTER YOUR ALGORHTM HERE
                    if (isempty(newScoreRow) && isempty(imCbirScores))
                        fprintf('Uh_oh,_did_you_implement_the_second_student_metric,_yet?\n');
                    end;
                    % random value between zero and one
                    newScoreRow = [newScoreRow rand(1)];
            end;
        end;    % end dataset for loop

        % append the new row onto the returned scores
        imCbirScores = [imCbirScores; newScoreRow];
    end;    % end query id for loop


% go from a list of scores to a set of ranked lists
function imCbirRanks = convert_to_ranked_list(finalScores, imIdDatasetKnown)
    imCbirRanks = [];
    numQuery = size(finalScores,1);
    for idxQuery=1:numQuery
        % sort the scores
        [sortScore, sortIdx] = sort(finalScores(idxQuery,:));
        % find the new ranked ids using the sorted index as reference
        sortedIds = imIdDatasetKnown(sortIdx);
        imCbirRanks = [imCbirRanks; sortedIds];
    end;

% this sub-function computes precision of the ranked lists for each query
function [imCbirP, imCbirR] = compute_PR(imIdsMatch, imCbirRanks, currentDepth)
    imCbirP = []; imCbirR = [];
    for idxQuery = 1:size(imCbirRanks,1)
```

```
        % get all result images from this depth
        topNresults = imCbirRanks(idxQuery,1:currentDepth);

        % now find the intersection for the precision
        numMatches = length(intersect(topNresults,imIdsMatch));

        % save the comptued precision
        imCbirP = [imCbirP numMatches];
        imCbirR = [imCbirR numMatches];
    end;
    % noramlized all precisions by this depth
    imCbirP = imCbirP/currentDepth;
    imCbirR = imCbirR/length(imIdsMatch);

% utility function to draw three images together
function drawmatches(imDirectory, targetIm)
    figure; %(2);
    numCells = length(targetIm);
    for idxIm=1:numCells
        srcIm = imread(sprintf('%s/%03d.jpg', imDirectory, targetIm(idxIm)));
        subplot(1,numCells,idxIm), imshow(srcIm);
        title(sprintf('image_%03d', targetIm(idxIm)));
    end;

% utility function do create TEST submission format
function write_test_submission_file(outputFile, imQueryId, imRanking)
    outputData = [];
    for idxQuery=1:length(imQueryId)
        currentQueryId = imQueryId(idxQuery);

        % replicate our column indicating query id
        newRows = ones(100,1)*currentQueryId;
        % grab the top 100 results from our ranking matrix
        newRows = [newRows imRanking(idxQuery,1:100)'];
        % finally, put in the rank order that we are guessing
        newRows = [newRows [1:100]'];

        % done, plop this onto the results list
        outputData = [outputData; newRows];
    end;

    % handy matlab function to save our our ASCII file
    save(outputFile, 'outputData', '-ASCII');
```