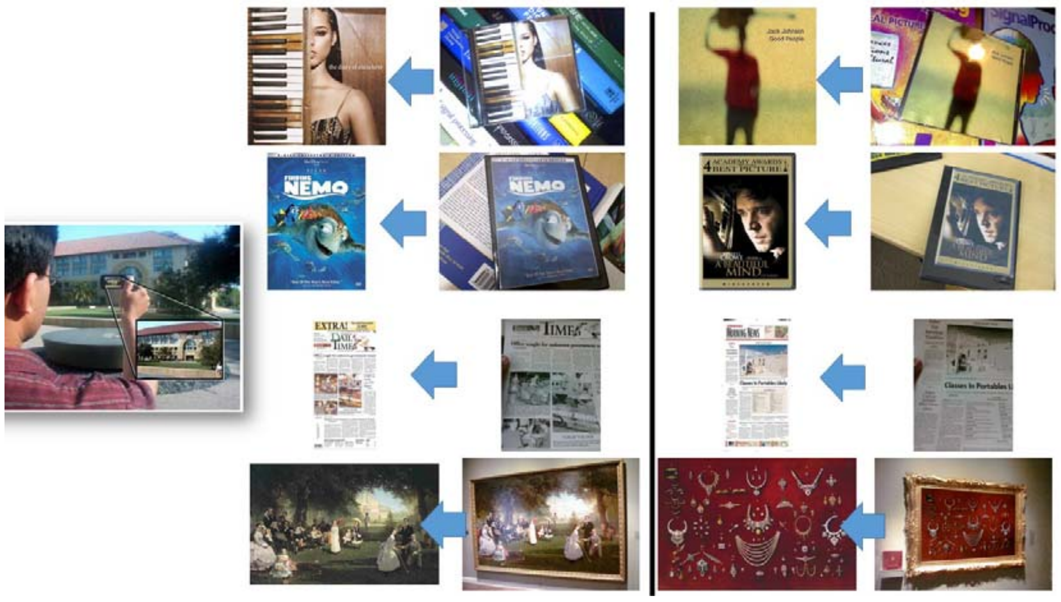
CS 376 Computer Vision
Spring 2018
Assignment 4

Out: Tuesday April 3
Due: Tuesday, April 17 11:59 PM



Visual search from mobile phone photos

For this problem, you will implement visual search methods to retrieve relevant database images for a query image taken on a mobile phone containing an object of interest, such as a book, printed material, video, or landmark building.

Data

We are providing image data and some starter code for this assignment. You can access the image dataset and pre-computed SIFT features here `/v/filer4b/v45q002/data`. The data takes about 4G, so you should *not* try to copy it to your home directory. Just point to the data files directly in your code.

There are four directories containing four different types of images: *video frames*, *print*, *landmarks*, and *book covers*. For the video frame type, a reference image is from a video and the query is from a photo taken of a screen playing that video. For the print type, the reference is a clean file of, e.g., a newspaper page, and the query is a mobile phone photo of the physical newspaper. For the landmarks type, the images contain building facades outdoors. Finally, the book covers type consists of reference catalog images of a book and query images of mobile phone photos of the book amidst clutter.

Within each of these directories, there are several sub-directories. One of them is always "`Reference`". That directory contains the "database" images. The other directories are named by camera types; they correspond to the different mobile phone cameras used to capture a set of queries. For example, an "`iPhone`" sub-directory contains query images that were taken with an iPhone. They are typically noisier, lower-resolution captures of the object compared to its Reference view.

In each of those directories (Reference, iPhone, etc.) you will find jpg images and precomputed SIFT features. The latter is saved as a `.mat` file per image. For example, image `001.jpg` has the associated file `001.jpg.sift.mat`. In each `.mat` file, there are **two key variables**:

```
frames 4 x n     // contains location, scale, orientation data for the n SIFT patches
                 // column i contains (x, y, scale, orientation) for the i-th SIFT patch

desc   128 x n // contains the 128-dimensional SIFT descriptors as columns
```

where n is the number of interest points that were detected for that image.

Note that the naming is significant: `book_covers/Reference/001.jpg` is a picture of the same object instance as `book_covers/iPhone/001.jpg`. You will need to use this alignment once you **evaluate** your method, to score how frequently it is retrieving the correct database image as its top result(s).

Provided code

These are the provided code files, available in the shared `/v/filer4b/v45q002/code` directory.

- `a4starter.m`: Run this first and make sure you understand the data format and the files (jpgs/mat) as described above. It is a script that shows a loop of data files, and how to access each descriptor via `frames` and `desc`. It also shows how to use some of the other functions below.

- `displaySIFTFeatures.m`: given SIFT descriptor info, it draws the patches on top of an image

- `getPatchFromSIFTParameters.m`: given SIFT descriptor info, it extracts the image patch itself and returns as a single image.

- `showMatchingPatches.m`: given SIFT descriptor info and images, together with matches computed between some number of their features, display the matched patches. This is useful to visualize what matches your code has computed.

- `showLinesBetweenMatches.m:` an alternative display function that shows lines connecting the matches (no patches overlaid).

- `dist2.m`: a fast implementation of computing pairwise distances between two matrices for which each row is a data point

- `kmeansML.m`: a faster k-means implementation that takes the data points as columns

You are not required to use any of these functions, but you will probably find them helpful.

Also download the VLFeats library here: http://www.vlfeat.org/. Follow their instructions for setup. For Matlab, this entails calling >>> run('VLFEATROOT/toolbox/vl_setup'); where VLFEATROOT is the directory where you downloaded the code. See http://www.vlfeat.org/install-matlab.html. The provided scripts use functions from VLFeats for SIFT feature extraction and displaying matched features. The provided script `a4starter.m` will demonstrate some of the provided display code as well as a simple single feature match.

What to implement and discuss in the writeup

Write one script for each of the following (along with any helper functions you find useful), and in your pdf writeup report on the results, explain, and show images where appropriate. Run/read the file `a4starter.m` to see how to access the provided data and display and extract patches and their matches (which you'll need to compute).

1. **Raw descriptor matching [20 pts]:** Given two images, load their SIFT descriptors and compute the nearest neighbor matches. Note, no visual vocabulary should be used for this one. Name your script `rawDescriptorMatches.m.` Use it to demonstrate matching on one image pair containing the same object: for each patch in the first image, extract the best available match in the second image in terms of lowest Euclidean distance in SIFT space. Display the results as a figure. If the number of descriptors is so high that it looks cluttered, randomly downsample those you include in the display.

2. **Visualizing the vocabulary [20 pts]:** Build a visual vocabulary with k-means. Use a representative (large but manageable) corpus of descriptors from the images. Display example image patches associated with two of the visual words. Choose two words that are distinct to illustrate what the different words are capturing, and display enough patch examples so the word content is evident (e.g., say 10-15 patches per word displayed). See provided helper functions `getPatchFromSIFTParameters` and `kmeansML`, and feel free to re-use your code from the textons assignment. Explain what you see. Name your script `visualizeVocabulary.m.`

3. **Full frame queries [20 pts]:** Write code to do bag-of-words matching, using a linear scan through all the "database" reference frames. The similarity measure is the normalized inner product between two bag of words histograms. After testing your code for bag-of-words visual search, choose 2 different mobile phone image queries from the entire dataset (from any of the types: print, book cover, etc.) to serve as interesting successful queries, and 1 other query to serve as an interesting failure case query. Display the *M*=5 most similar reference images to each of these queries (in rank order) based on the normalized scalar product between their bag of words histograms. Briefly explain the results. Name your script `bagOfWordsQueries.m`

4. **Quantitatively evaluate your results for retrieval [20 pts]:** Using the bag of words method from part 3 above, evaluate your code systematically on all the provided data. In particular, process each query image (all non-Reference images) in turn, searching through *all* Reference images for their bag of words similarity. Sort the Reference images by their similarity. Then report the average "top-K" accuracy for each type of images, for K=1,3,5. That is, record the fraction of book cover queries for which the correct Reference image appears in your top K retrieved results; do the same for each type in turn (print, landmarks, book covers, video frames). You can informally experiment with the vocabulary size to understand the sensitivity of your results and select a reasonable value. Report a table with the numerical results in your writeup. Analyze the performance. Which types fare better/worse with this approach, and why? Discuss your findings and use illustrative examples where relevant. Name your script `quantitativeEvaluation.m`

5. **Spatial verification [20 pts]:** Write code to perform spatial verification via an affine alignment with RANSAC. The verification stage should return the number of inliers found for a given pair of images. Then, for a given query image of your choosing, run this spatial verification stage on a shortlist of the top *T* ranked Reference images according to the bag of words similarity. Rerank that shortlist by the inlier counts. Choose a query case where this verification improves the rank of the correct Reference image. Show the matches before and after verification. Display the results and discuss briefly. Name your script `spatialVerification.m.`

III. OPTIONAL : Extra credit (up to 10 points each, max 15 points total)

- **[10 pts] Stop list and tf-idf**. Implement a stop list to ignore very common words, and apply tf-idf weighting to the bags of words. Create an experiment to illustrate the impact on your results and discuss.

- **[5 pts] Add a "ratio test" functionality** to part 1 above that discards ambiguous neighbor matches (see lecture 13). Display the results for a matched pair of images where the ratio test appears valuable.

- **[10 pts] Quantitatively evaluate** the results of adding spatial verification to the bag of words shortlist. Compare the performance before and after, again broken down per image type.

- **[5 pts] Augment the descriptor set** with complementary interest operator(s) beyond the DoG used for the pre-computed features, and discuss the results. http://www.robots.ox.ac.uk/~vgg/research/affine/

- **Have something else you want to try?** Run it by us.

Acknowledgements

The image data we are using comes from the Stanford Mobile Visual Search Dataset:

Chandrasekhar, Vijay and Chen, David and Tsai, Sam and Cheung, Ngai-Man and Chen, Huizhong and Takacs, Gabriel and Reznik, Yuriy and Vedantham, Ramakrishna and Grzeszczuk, Radek and Bach, Jeff and Girod, Bernd. (2013). Stanford Mobile Visual Search Dataset. Stanford Digital Repository. Available at: http://purl.stanford.edu/rb470rw0983

The figure above is taken from the paper "The Stanford Mobile Visual Search Data Set" by Chandrasekhar etal., ACM MMSys 2011.

Submission instructions: what to hand in

Please make a single zip file to submit on Canvas with:

- Your well-documented Matlab code .m files named as specified above.

- A pdf file containing
  - Your name and CS login ID at the top.
  - Your image results and accompanying explanations.
  - (optional): Any results and descriptions for extra credit portions. For max credit explain what you implemented and also interpret the results. Please indicate clearly in the pdf what parts of the extra credit you did and which submitted code files are involved.