16720-A S18 Bag of Visual Words

Instructor: Kris Kitani

TAs: Leonid, Mohit, Arjun, Rawal, Aashi, Tanya

Due Feb 22, 2018 11:59 PM

Total Points: 100

Extra Credit Points: 30

Instructions

1. **Integrity and collaboration:** Students are encouraged to work in groups but each student must submit their own work. If you work as a group, include the names of your collaborators in your write up. Code should **NOT** be shared or copied. Please **DO NOT** use external code unless permitted. Plagiarism is strongly prohibited and may lead to failure of this course.

2. **Start early!** Especially those not familiar with MATLAB. Constructing the dictionary in **Q1.4**, as well as converting all the images to visual words in **Q2.1** can easily take up to 30 mins to run (if implemented well!). If you start late, you will be pressed for time while debugging.

3. **Questions:** If you have any question, please look at piazza first. Other students may have encountered the same problem, and is solved already. If not, post your question on the discussion board. TAs will respond as soon as possible.

4. **Write-up:** Items to be included in the writep are mentioned in each queestion, and summarized in the Writeup section. Please note that we **DO NOT** accept handwritten scans for your write-up in this assignment. Please type your answers to theory questions and discussions for experiments electronically.

5. **Handout:** The handout zip file contains 3 items. `hw2.pdf` is the assignment handout. `data` contains 1491 image files from the SUN image database, divided into 8 category folders. it also contains `traintest.mat`, which contains all the training and testing image paths, as well as the labels. `matlab` contains 3 scripts that you will make use of in this project.

6. **Submission:** The submission is on Gradescope, **you will be submitting both your writeup and code zip file**. The zip file, `<andrew-id.zip>`, contains your MATLAB implementations (including helper functions), results for extra credit (optional). Do not handin the image files we distributed in the handout zip, or any of the generated wordmap files. However you should handin the dictionary and vision mat files that you generate. **Note: You have to submit your writeup separately to Gradescope as** `<andrew-id.pdf>`.

Your final upload should have the files arranged in this layout:

`<AndrewID>.zip`

- <AndrewId>
  - ec
    - `computeIDF.m` (QX.2, *optional*)
    - `evaluateRecognitionSystem_IDF.m` (QX.2, *optional*)
    - `evaluateRecognitionSystem_SVM.m` (QX.1, *optional*)
    - `tryBetterFeatures.m` (QX.3, *optional*)
    - *Any other helper functions or external code*
  - matlab
    - `RGB2Lab.m` (*provided*)
    - `batchToVisualWords.m` (*provided*)
    - `buildRecognitionSystem.m` (Q2.3)
    - `createFilterBank.m` (*provided*)
    - `dictionaryHarris.mat` (Q1.3)
    - `dictionaryRandom.mat` (Q1.3)
    - `evaluateRecognitionSystem.m` (Q3.2)
    - `extractFilterResponses.m` (Q1.1)
    - `getDictionary.m` (Q1.3)
    - `getHarrisPoints.m` (Q1.2)
    - `getImageDistance.m` (Q3.1)
    - `getImageFeatures.m` (Q2.2)
    - `getRandomPoints.m` (Q1.2)
    - `getVisualWords.m` (Q2.1)
    - `visionHarris.mat` (Q2.3)
    - `visionRandom.mat` (Q2.3)
    - *Any other helper functions you need*

Please use the sample submission zip on Piazza to structure your handin.

7. TAs responsible for this assignment: Tanya Marwah (tmarwah@andrew.cmu.edu) and Rawal Khirodkar (rkhirodk@andrew.cmu.edu).

Overview

Perhaps the most common problem in computer vision is classification. Given an image that comes from a few fixed categories, can you determine which category it belongs to? For this assignment, you will be developing a system for scene classification. You have been given a subset of the SUN Image database consisting of eight scene categories. See Figure 1. In this assignment, you will build an end to end system that will, given a new scene image, determine which type of scene it is.

Figure 1: The eight categories you have been given from the SUN Image database

This assignment is based on an approach for document classification called **Bag of Words**. It represents a document as a vector or histogram of counts for each word that occurs in the document, as shown in Figure 2. The hope is that different documents in the same class will have a similar collection and distribution of words, and that when we see a new document, we can find out which class it belongs to by comparing it to the histograms already in that class. This approach has been very successful in Natural Language Processing, which is surprising due to its relative simplicity (we are throwing away the sequential relationships!). We will be taking the same approach to image classification. However one major problem arises. With text documents, we actually have a dictionary of words. But what *words* can we use to represent an image?

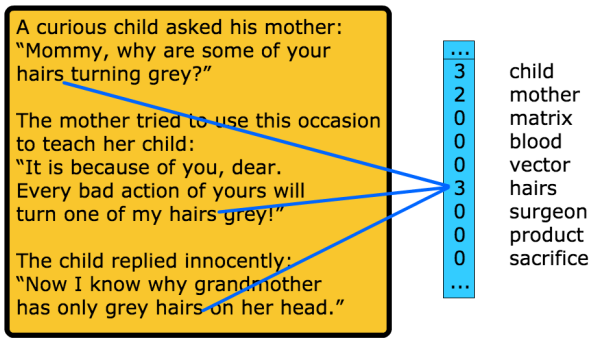

Figure 2: Bag of words representation of a text document

This assignment has 3 major parts.

- **Part 1**: build a dictionary of *visual words* from training data.
- **Part 2**: build the recognition system using visual word dictionary and training images.
- **Part 3**: evaluate the recognition system using test images.

In **Part 1**, you will use the provided filter bank to convert each pixel of each image into a high dimensional representation that will hopefully capture meaningful information, such as corners, edges etc. This will take each pixel from being a 3D vector of color values, to an nD vector of filter responses. You will then take these nD pixels from all of the training images and and run K-means clustering to find groups of pixels. Each resulting cluster center will become a visual word, and the whole set of cluster centers becomes our dictionary of visual words. In theory, we would like to use all pixels from all training images, but this is very computationally expensive. Instead, we will only take a small sample of pixels from each image. One option is to simply select α pixels from each one uniformly at random. Another option is to use some feature detector (Harris Corners for example), and take α feature points from each image. You will do both to produce two dictionaries, so that we can compare their relative performances. See Figure 3.

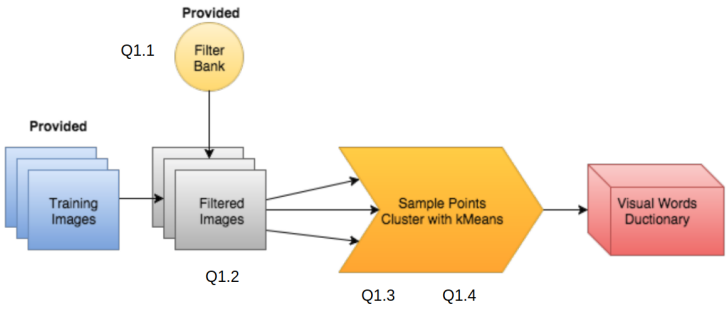

Figure 3: Flow chart of first part of the assignment that involves building the dictionary of visual words

In **Part 2**, the dictionary of visual word you produced will be applied to each of the training images to convert them into a wordmap. This will take each of the nD pixels in all of the filtered training images and assign each one a single integer label, corresponding to the closest cluster center in the visual words dictionary. Then each image will be converted to a "bag of words"; a histogram of the visual words counts. You will then use these to build the classifier (Nearest Neighbours, and SVM for extra credit). See Figure 4.

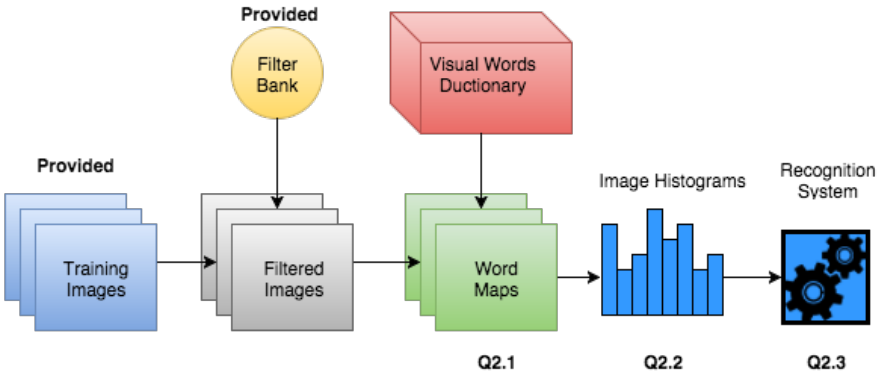

Figure 4: Flow chart of second part of the assignment that involves building the recognition system

In **Part 3** you will evaluate the recognition system that you built. This will involve taking the test images and converting them to image histograms using the visual words dictionary and the function you wrote in Part 2. Next, for nearest neighbor classification, you will use a histogram distance function to compare the new test image histogram to the training image histograms in order to classify the new test image. Doing this for all the test images will give you an idea of how good your recognition system. See Figure 5.

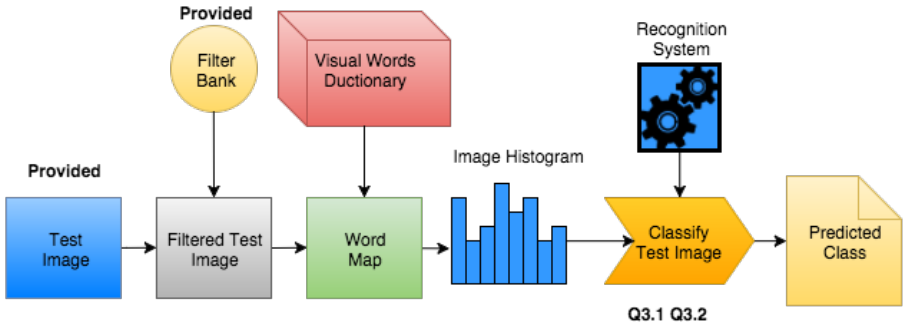

Figure 5: Flow chart of the third part of the project that involves evaluating the recognition system

In your write-up, we will ask you to include various intermediate result images, as well as metrics on the performance of your system.

Programming

Part 1: Build Visual Words Dictionary

Q1.1 Filter Bank (5 points)

We have provided you with a multi-scale filter bank that you will use to understand the visual world. You can create an instance of it with the following (provided) function:

```
function [filterBank] = createFilterBank().
```

`filterBank` is a cell array, with the pre-defined filters in its entries. We are using 20 filters consisting of 4 types of filters in 5 scales. Figure 6

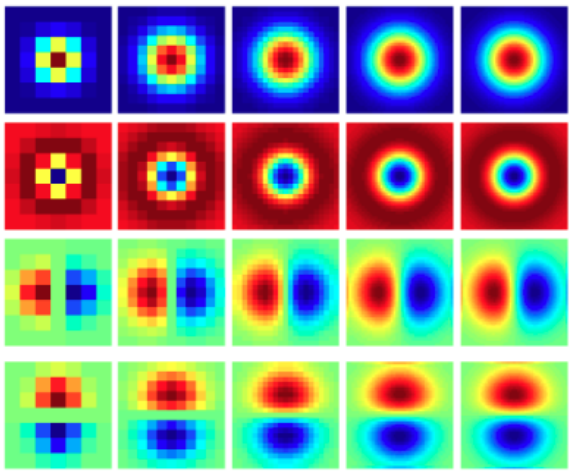

Figure 6: The provided multi-scale filter bank

In your writeup: What properties do each of the filter functions pick up? You should group the filters into broad categories (i.e all Gaussians, derivatives etc).

Q1.2 Extract Filter Responses (5 points)

Write a function to extract filter responses. Pass `filterBank` cell array of image convolution filters to your `extractFilterResponses` function, along with an image of size $H \times W \times 3$.

```
function [filterResponses] = extractFilterResponses(I, filterBank)
```

Your function should use the provided `RGB2Lab(..)` function to convert the color space of `Im` from RGB to Lab. Then it should apply all of the n filters on each of the 3 color channels of the input image. You should end up with $3n$ filter responses for the image. The final matrix that you return should have size $H \times W \times 3n$. Figure 7

In your writeup: Show an image from the dataset and 3 of its filter responses. Explain any artifacts you may notice. Also briefly describe the CIE Lab colorspace, and why we would like to use it. We did not cover the CIE Lab color space in class, so you will need to look it up online.

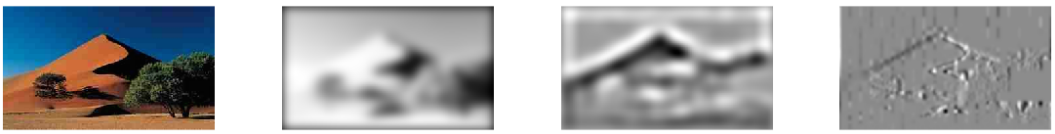

Figure 7: Sample desert image and a few filter responses (in CIE Lab colorspace)

Q1.3 Collect sample of points from image (10 points)

Write two functions that return a list of points in an image, that will then be clustered to generate visual words.

First write a simple function that takes an image and α value, and returns a matrix of size $\alpha \times 2$ of **random pixels locations** inside the image.

```
[points] = getRandomPoints(I, alpha)
```

Next write a function that uses the **Harris corner detection algorithm** to select keypoints from an input image, the algorithm is similar to one discussed in class.

```
[points] = getHarrisPoints(I, alpha, k)
```

Recall from class the Harris corner detector finds corners by building a covariance matrix of edge gradients within a region around a point in the image. The eigenvectors of this matrix point in the two directions of greatest change. If they are both large, then this indicates a corner. See class slides for more details.

This function takes the input image `I`. `I` may either be a color or grayscale image, but the following operations should be done on the grayscale representation of the image. For each pixel, you need to compute the covariance matrix

$$H = \begin{bmatrix} \sum_{p \in P} I_{xx} & \sum_{p \in P} I_{xy} \\ \sum_{p \in P} I_{yx} & \sum_{p \in P} I_{yy} \end{bmatrix}$$

where $I_{ab} = \frac{\partial I}{\partial a} \frac{\partial I}{\partial b}$. You can use a 3×3 or 5×5 window. It is a good idea to precompute the image's X and Y gradients, instead of doing them in every iteration of the loop. For the sum, also think about how you could do it using a convolution filter.

You then want to detect corners by finding pixels who's covariance matrix eigenvalues are large. Since its expensive to compute the eigenvalues explicity, you should instead compute the response function with

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 = \det(H) - k \operatorname{tr}(H)^2$$

`det` represents the determinant and `tr` denotes the trace of the matrix. Recall that when detecting corners with the Harris corner detector, corners are points where both eigenvalues are large. This is in contrast to edges (one eigenvalue is larger, while the other is small), and flat regions (both eigenvalues are small). In the response function, the first term becomes very small if one of the eigenvalues are small, thus making $R < 0$. Larger values of R indicates similarly large eigenvalues.

Instead of thresholding the response function, simply take the top α response as the corners, and return their coordinates. A good value for the k parameter is 0.04 - 0.06.

Note: You will be applying this function to about 1500 images, try to implement this without loops using vectorisation. However, there is no penalty for slow code.

In your writeup: Show the results of your corner detector on 3 different images.

Figure 8: Possible results for getRandomPoints and getHarrisPoints functions

Q1.4 Compute Dictionary of Visual Words (15 points)

You will now create the dictionary of visual words. Write the function:

```
function [dictionary] = getDictionary(imgPaths, alpha, K, method)
```

This function takes in an cell array of image paths. Load the provided `traintest.mat` to get a list of the training image paths. For every training image, load it and apply the filter bank to it. Then get α points for each training image an put it into an array, where each row represents a single n dimensional pixel (n = number of filters). If there are T training images, then you will build a matrix of size $\alpha T \times 3n$, where each row corresponds to a single pixel, and there are αT total pixels collected.

`method` will be a string either `'random'` or `'harris'`, and will determine how the α points will be taken from each image. Once you have done this, pass all of the points to matlab's K-means function.

```
[~, dictionary] = kmeans(pixelResponses, K, 'EmptyAction', 'drop')
```

The result of Matlab's `kmeans` function will be a matrix of size $K \times 3n$, where each row represents the coordinates of a cluster center. This matrix will be your dictionary of visual words.

For testing purposes, use $\alpha = 50$ and $K = 100$. Eventually you will want to use much larger numbers to get better results for the final part of the write-up.

This function can take a while to run. Start early and be patient. When it has completed, you will have your dictionary of visual words. Save this and the filter bank you used in a `.mat` file. This is your visual words dictionary. It may be helpful to write a `computeDictionary.m` which will do the legwork of loading the training image paths, processing them, building the dictionary, and saving the mat file. This is not required, but will be helpful to you when calling it multiple times.

For this question, you must produce two dictionaries. One named `dictionaryRandom.mat`, which used the random method to select points and another named `dictionaryHarris.mat` which used the Harris method. Both must be handed in.

Part 2: Build Recognition System

Q2.1 Convert image to wordmap (15 points)

Write a function to map each pixel in the image to its closest word in the dictionary.

```
[wordMap] = getVisualWords(I, dictionary, filterBank)
```

`I` is the input image of size $H \times W \times 3$. `dictionary` is the dictionary computed previously. `filterBank` is the filter bank that was used to construct the dictionary. `wordMap` is an $H \times W$ matrix of integer labels, where each label corresponds to a word/cluster center in the dictionary.

Use MATLAB function `pdist2` with Euclidean distance to do this efficiently. You can visualize your results with the function `label2rgb`.

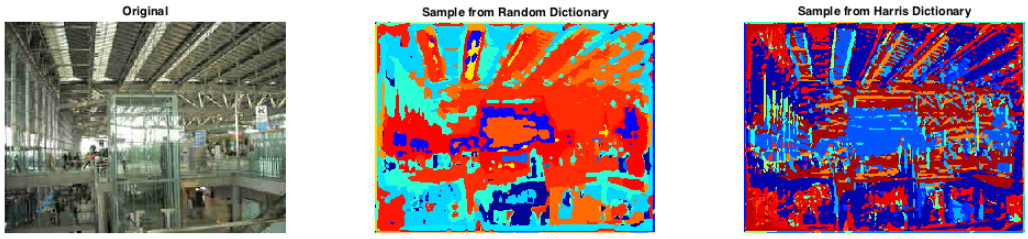

Figure 9: Sample visual words plot, made from dictionary using $\alpha = 50$ and $K = 50$

Once you are done, call the provided script `batchToVisualWords`. This function will apply your implementation of `getVisualWords` to every image in the training and testing set. The script will load `traintest.mat`, which contains the names and labels of all the data images. For each training image `data/<category>/X.jpg`, this script

will save the wordmap file `data/<category>/X.mat`. For optimal speed, pass in the number of cores your computer has when calling this function. This will save you from having to keep rerunning `getVisualWords` in the later parts, unless you decide to change the dictionary.

In your writeup: Show the wordmaps for 3 different images from two different classes (6 images total). Do this for each of the two dictionary types (random and Harris). Are the visual words capturing semantic meanings? Which dictionary seems to be better in your opinion? Why?

Q2.2 Get Image Features (10 points)

Create a function that extracts the histogram of visual words within the given image (i.e., the bag of visual words).

```
[ h ] = getImageFeatures(wordMap, dictionarySize)
```

`h` is a histogram of size $1 \times K$, where `dictionarySize` is the number of clusters K in the dictionary. `h(i)` should be equal to the number of times visual word `i` occurred in the wordmap.

Since the images are of differing sizes, the total count in `h` will vary from image to image. To account for this, L_1 normalize the histogram before returning it from your function (all entries in the histogram should sum to 1 after normalization).

Q2.3 Build Recognition System - Nearest Neighbours (10 points)

Now that you have build a way to get a vector representation for an input image, you are ready to build the vision scene classification system. This classifier will make use of nearest neighbour classification. Write a script `buildRecognitionSystem.m` that saves `visionRandom.mat` and `visionHarris.mat` and store in them the following:

1. `dictionary`: your visual word dictionary.
2. `filterBank`: filter bank used to produce the dictionary. This is a cell array of image filters.
3. `trainFeatures`: $T \times K$ matrix containing all of the histograms of visual words of the T training images in the data set.
4. `trainLabels`: $T \times 1$ vector containing the labels of each training image.

You will need to load the `train_imagenames` and `train_labels` from `traintest.mat`. Load `dictionary` from `dictionaryRandom.mat` and `dictionaryHarris.mat` you saved in part **Q1.4**.

Part 3: Evaluate Recognition System

Q3.1 Image Feature Distance (10 points)

For nearest neighbor classification you need a function to compute the distance between two image feature vectors. Write a function

```
[dist] = getImageDistance(hist1, hist2, method)
```

`hist1` and `hist2` are the two image histograms who's distance will be computed (with `hist2` being the target), and returned in `dist`. The idea is that two images that are very similar should have a very small distance, while dissimilar images should have a larger distance.

`method` will control how the distance is computed, and will either be set to `'euclidean'` or `'chi2'`. The first option tells to compute the euclidian distance between the two histograms. The second uses χ^2 distance. (Refer `pdist2` function in Matlab).

Alternatively, you may also write the function

```
[dist] = getImageDistance(hist1, histSet, method)
```

which, instead of the second histogram, it takes in a matrix of histograms, and returns a vector of distances between `hist1` and each histogram in `histSet`. This may make it possible to implement things more efficiently. Choose either one or the other to hand in.

Q3.2 Evaluate Recognition System (20 points)

Write a script `evaluateRecognitionSystem.m` that evaluates your nearest neighbor recognition system on the test images. Nearest neighbor classification assigns the test image the same class as the "nearest" sample in your training set. "Nearest" is defined by your distance function (`'euclidean'` or `'chi2'`).

Load `traintest.mat` and classify each of the `test_imagenames` files. Have the script report both the accuracy ($\frac{\#correct}{\#total}$), as well as the 8×8 confusion matrix `C`, where the entry `C(i,j)` records the number of times an image of actual class `i` was classified as class `j`.

The **confusion matrix** can help you identify which classes work better than others and quantify your results on a per-class basis. In a perfect classifier, you would have entries in the diagonal of `C`.

For each combination of dictionary (random or Harris) and distance metric (euclidean and χ^2), have your script print out the confusion matrix. (You may use `disp` or `fprintf`).

In your writeup:

- Include the output of `evaluateRecognitionSystem.m` (4 confusion matrices and accuracies. Do mention distance metric and dictionary types with them).
- How do the performances of the two dictionaries compare? Is this surprising?
- How about the two distance metrics? Which performed better? Why do you think this is?

Extra credit

QX.1 Evaluate Recognition System - Support Vector Machine (extra: 10 points)

In the class, you learnt that support vector machine (SVM) is a powerful classifier. Write a script, `evaluateRecognitionSystem_SVM.m` to do classification using SVM. Produce a new `visionSVM.mat` file with whatever parameters you need, and evaluate it on the test set.

We recommend you to use LIBSVM `http://www.csie.ntu.edu.tw/cjlin/libsvm/`, which is the most widely used SVM library, or the builtin SVM functions in matlab. Try at least two types of kernels.

In your writeup: Are the performances of the SVMs better than nearest neighbor? Why or why not? Does one kernel work better than the other? Why?

QX.2 Inverse Document Frequency (extra: 10 points)

With the bag-of-word model, image recognition is similar to classifying a document with words. In document classification, inverse document frequency (IDF) factor is incorporated which diminishes the weight of terms that occur very frequently in the document set. For example, because the term "the" is so common, this will tend to incorrectly emphasize documents which happen to use the word "the" more frequently, without giving enough weight to the more meaningful terms.

In the homework, the histogram we computed only considers the term frequency (TF), i.e. the number of times that word occurs in the word map. Now we want to weight the word by its inverse document frequency. The IDF of a word is defined as:

$$IDF_w = log \frac{T}{|\{d : w \in d\}|}$$

Here, T is number of all training images, and $|\{d : w \in d\}|$ is the number of images d such that w occurs in that image.

Write a script `computeIDF.m` to compute a vector `IDF` of size $1 \times K$ containing IDF for all visual words, where **K** is the dictionary size. Save **IDF** in `idf.mat`. Then write another script `evaluateRecognitionSystem_IDF.m` that makes use of the `IDF` vector in the recognition process. You can use either nearest neighbor or SVM as your classifier.

In your writeup: How does Inverse Document Frequency affect the performance? Better or worse? Does this make sense?

QX.3 Better pixel features (extra: 10 points)

The filter bank we provided to you contains some simple image filters, such as Gaussians, derivatives, and Laplacians. However you learned about various others in class, such as Harr wavelets, Gabor filters, SIFT, HOG etc... Try out anything you think might work. Include a script `tryBetterFeatures.m` and any other code you need.

Feel free to use any code or packages you find online, but be sure to cite it clearly. Experiment with whatever you see fit, just make sure the submission does not become too large.

In your writeup: What did you experiment with and how did it perform. What was the accuracy?

Writeup Summary

Q1.1 What properties do each of the filter functions pick up? You should group the filters into broad categories (i.e all Gaussians, derivatives etc).

Q1.2 Show an image from the dataset and 3 of its filter responses. Explain any artifacts you may notice. Also briefly describe the CIE Lab colorspace, and why we would like to use it. We did not cover the CIE Lab color space in class, so you will need to look it up online.

Q1.3 Show the results of your corner detector on 3 different images.

Q2.1 Show the wordmaps for 3 different images from two different classes (6 images total). Do this for each of the two dictionary types (random and Harris). Are the visual words capturing semantic meanings? Which dictionary seems to be better in your opinion? Why?

Q3.2 Comment on whole system:

- Include the output of `evaluateRecognitionSystem.m` (4 confusion matrices and accuracies).
- How do the performances of the two dictionaries compare? Is this surprising?
- How about the two distance metrics? Which performed better? Why do you think this is?

QX.1 Are the performances of the SVMs better than nearest neighbor? Why or why not? Does one kernel work better than the other? Why?

QX.2 How does Inverse Document Frequency affect the performance? Better or worse? Does this make sense?

QX.3 What did you experiment with and how did it perform. What was the accuracy?

References

[1] S. Lazebnik, C. Schmid, and J. Ponce, *Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories.* CVPR, 2006.