Assignment 2
Computer Vision 16–385, Spring 15
Due Date: Mar/6/2015, 23:59:59
Total Points: 100, Extra credit: 20

Given an image, can a computer program determine where it was taken? In this assignment, you will build an image representation based on bags of visual words and learn to use support vector machine to classify scene categories. You will implement a recognition system similar to [1].

1 Instructions

1. If you have any question, ask on piazza or contact:
   Minghuang Ma, **minghuam@andrew.cmu.edu.**

2. **Start early!** Especially those not familiar with MATLAB.

1.1 Submitting Your Assignment

Your submission for this assignment will comprise of a write-up (theory and experiments), the code for your MATLAB implementation, and the outputs of intermediate steps (wordmaps, and some mat file). The name of the write-up should be `<AndrewId>.pdf`. Each of the MATLAB functions you write as described in Section 2 along with any extra helper functions you wrote should be in a folder named `matlab`. Make sure all the files needed for your code to run (except data) are included. Remember to delete the `images/` folder before submission.

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

- <AndrewId>
  - `<AndrewId>.pdf`
  - wordmaps
    - `your word maps`
  - matlab
    - `batchToVisualWords.m` *(provided)*
    - `buildRecognitionSystem.m`
    - `batchComputeDictionary.m.m` *(provided)*
    - `dictionary.mat`
    - `evaluateRecognitionNN.m`
    - `evaluateRecognitionSVM.m`
    - `extractSIFTResponses.m`
    - `getDictionary.m`

    - * getImageFeatures.m
    - * getVisualWords.m
    - * dataset.mat *(provided)*
    - * vision.mat
    - * release
        - · code you downloaded online
- application *(optional for extra credit)*
    - * your own images
    - * your own code

Please pack your system and write-up into a single file named <AndrewId>.zip and upload to the Blackboard.

2 Programming

2.1 Build the Bag-of-Word Model

Q2.1.1 Extract Filter Response (10 points)

Write a function to extract dense-SIFT responses.

```
function [siftResponses] = extractSIFTResponses(I, binSize, step)
```

In this function, you do *not* have to implement dense-SIFT by yourself. Find a dense-SIFT implementation online and wrap it in this function. Remember to cite the source in your write-up.

I is the input image. binSize is size to extract SIFT response. step is step size in pixels. siftResponses is a $N \times M$ matrix of **doubles**, where N is the number of pixels from which SIFT are extracted (depending on binSize and step), and M is the feature's dimension (128 for SIFT).

For the baseline implementation, I used VLFeat package with binSize = 4 and step = 8.

Q2.1.2 Compute Dictionary of Visual Words (15 points)

You will now create a dictionary of visual words from the SIFT responses using k-means. After applying k-means, similar SIFT responses will be represented by the same visual word.

```
function [dictionary] = getDictionary(imgPaths)
```

Given training image paths imgPaths, you will perform k-means on sampled filter (SIFT) responses to get a dictionary. Instead of using all of the SIFT responses (that can exceed the memory capacity of your computer), you will use responses at α random pixels of each image. If there are T training images, then you should collect a matrix

`siftResponses` over all the images that is of size $\alpha T \times 128$. Then, to generate a visual words dictionary with K words, you will cluster the responses with k-means using the built-in MATLAB function kmeans as follows:

```
[unused, dictionary] = kmeans(siftResponses, K, EmptyAction,drop)
```

For the baseline implementation, I used $K = 200$ and $\alpha = 100$.

Once you are done with `getDictionary`, call the provided script `batchComputeDictionary`, which will pass in the file names and save the result. This will take some time. If all goes well, you will have a `.mat` file named `dictionary.mat` that contains the dictionary of visual words.

Q2.1.3 Get Visual Words (15 points)

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

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

`I` is the input image. `dictionary` is the dictionary computed previously. Use `load dictionary.mat` to get it. Each pixel in `wordMap` stores the index of the closest visual word to the SIFT response at the respective pixel. For example, pixel $[i, j]$ in the `wordMap` is assigned to k if row k of the dictionary is the closest to the `siftResponse` at pixel $[i, j]$. Use MATLAB function `pdist2` with Euclidean distance to do this efficiently. With `step = 1`, you should get a result similar to that in Fig. 1.

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. For every image `"X.jpg"` in `images/`, there will be a corresponding file named `"X.mat"` in `wordmaps/` containing the variable wordMap. You may want to adjust the `numOfCores` in the script.

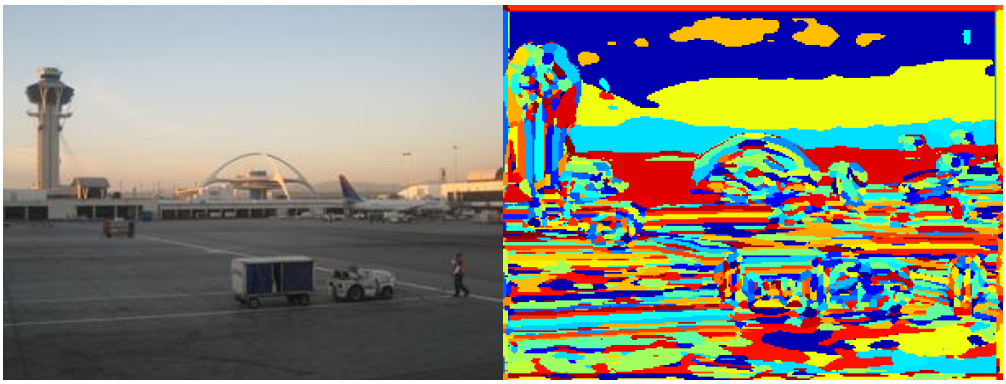

Figure 1: Word map example with original image.

Q2.1.4 Get Image Feature (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 $1 \times dictionarySize$, where `dictionarySize` is the first dimension size of the dictionary. Each element in `h` counts the number of that visual word in the `wordMap`. h should be L_1 normalized (`sum(h) = 1`).

2.2 Build a Recognition System

Q2.2.1 Collect Training Data (10 points)

Now that we have obtained a representation for each image, we want to build a model for classification. The first step is to collect the training data. Write a script `buildRecognitionSystem.m` that produces `vision.mat` containing:

1. `dictionary`: your visual word dictionary.
2. `trainFeatures`: $T \times K$ matrix containing all of the histograms of the T training images in the data set.
3. `trainLabels`: $T \times 1$ vector containing the labels of each training image.

You will need to load the training file names (`trainPaths`) and training class labels (`trainLabels`) from `dataset.mat`. Use MATLAB function `strrep` to get the path of the word map from the image path. Load the `dictionary` from `dictionary.mat`.

Q2.2.2 Build Recognition System – Nearest Neighbor (20 points)

Write a script `evaluateRecognitionNN.m` to do classification using nearest neighbor. You will need to load your training data from `vision.mat`. Load the testing file names (`testPaths`) and testing class labels (`testLabels`) from `dataset.mat`. For each testing image, compute its histogram and find the nearest histogram in `trainLabels`. Assign the corresponding training label as your prediction.

Try both euclidean distance and χ^2 distance and **compare the results in your write-up.** You should discuss why the performances are different. Also, **report the confusion matrices C in your write-up.** The entry $C(i, j)$ of a confusion matrix counts the number of instances of class i that were predicted as class j.

My implementation gives overall accuracies 70% using euclidean distance and 77% using χ^2 distance. Don't be obsessed with the numbers. You can get full credit as long as you implementation is correct.

Q2.2.3 Build Recognition System – Support Vector Machine (20 points)

In the class, you learn that support vector machine (SVM) is a powerful classifier. Write a script `evaluateRecognition_SVM.m` to do classification using SVM. Again load data from `vision.mat` and `traintest.mat`. We recommend you to use LIBSVM `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`, which is the most widely used SVM software.

Try at least two types of kernel and **compare the results in your write-up.** Are the performances better than nearest neighbor? Why or why not?

3 Extra Credit

Q3.1X k-Nearest Neighbors (5 points)

In your Nearest Neighbor implementation, you pick the closest member from the training dataset. Usually, the performance can be improved by looking at k nearest neighbors. Implement a k-Nearest Neighbors classifier and draw a curve by varing `k`. Examplain your result in 2-3 sentences.

Q3.2X Classify Your Own Dataset (15 points)

The system you built in this homework is actually very powerful! Find your own application or dataset and apply your system. Place any work/data in the folder named `application/`. Some interesting application including classifying paintings from different artists, images from different dataset [2], images from different source (images on Amazon and pictures taken with a personal camera [3]). You can also try your system on standard object recognition dataset like Caltech101 `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`. Discuss your results in the write-up.

References

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

[2] A. Torralba and A. A. Efros, *Unbiased look at dataset bias*. CVPR, 2011.

[3] K. Saenko, B. Kulis, M. Fritz and T. Darrell, *Adapting Visual Category Models to New Domains*. ECCV, 2010.