16-720J: Homework 2
Bag-of-Words for Scene Classification

Instructor - Gary Overett
TA - Yang Gao

24 September 2015 - Due Midnight Thursday 8 October 2015

Figure 1: **Scene Classification:** Given an image, can a computer program determine where it was taken? In this homework, you will build a representation based on bags of visual words for classifying the scene categories.

1 Instructions/Hints

1. **Warning:** It is expected that you write all the code for this homework by yourself. Please do not use external code unless otherwise mentioned.
2. Please pack your system and write-up into a single file named <NetID>.zip, see the complete submission checklist in Section 9.
3. Section 8 contains a list of files provided.
4. **For the implementation part, please stick to the headers, variable names, and file conventions provided. For theory questions, you dont need to write any code.**

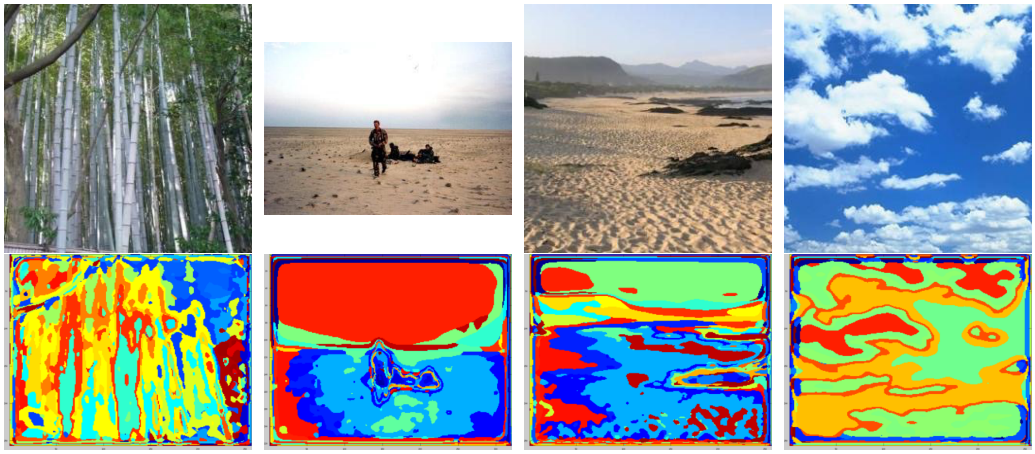


Figure 2: Visual words over images. You will use the spatially un-ordered distribution of visual words in a region (a bag of visual words) as a feature for scene classification, with some coarse information provided by spatial pyramid matching [4].

5. Start early! This will take longer than the last homework and cannot be debugged as easily since there are multiple inter-connected components.

6. **Attempt to verify your implementation as you proceed:** If you dont verify that your implementation is correct on toy examples, you will risk having a huge mess when you put everything together.

7. If you have any questions, please post them on the blackboard.

2 Overview

The bag-of-words (BoW) approach, which you learned about in class, has been applied to many recognition problems in computer vision. For example, object recognition [5, 7] and scene classification [6, 8][1]. Video Google `http://www.robots.ox.ac.uk/~vgg/research/vgoogle/` is a good demonstration of this.

This technique draws connection from Information Retreival, and has been studied extensively. Thus, there are many variants and extensions for the traditional approach explained in class. For example, two important extensions are pyramid matching [2, 4] and feature encoding [1].

What you will be doing:

1. You will take responses of a filter bank on images and build a dictionary of visual words (Section 4).

2. You will learn a model for the visual world based on a bag of visual words, and use nearest-neighbor to predict scene classes in a test set (Section 5).

[1]This homework aims at being largely self-contained; however, reading the listed papers (even without trying to truly understand them) is likely to be helpful.

3. Finally, we ask you to dig deep into visual words by trying to visualize them (Section 6).

If the assignment looks long ordifficult to you, **DONT panic**! We provide you with step-by-step instructions to implement a working scene recognition framework. There are not many lines of code to write. However, it may take a few hours for the base system to run, so make sure to try **each component** on a **subset** of the data set first before putting everything together. We have provided you with small subsets of the data for such testing (see Section 8).

Image Data You will be working with a subset of the SUN database2[2]. The data set contains 1600 images from 9 scene categories like "kitchen", "sky" and "desert".

A complete submission consists of a zip file with the following folders (please keep each system in a separate folder):

1. `baseline/`: the baseline spatial pyramid classification system that you implement through this homework;

2. `improved/`: (if you do the extra credit): the custom system that you design to boost the performance of your system;

3. a write-up (`.pdf` format).

We provide you with a number of functions and scripts in the hopes of alleviating some tedious or error-prone sections of the implementation.

You can find a list of files provided in Section 8. Please read these descriptions.

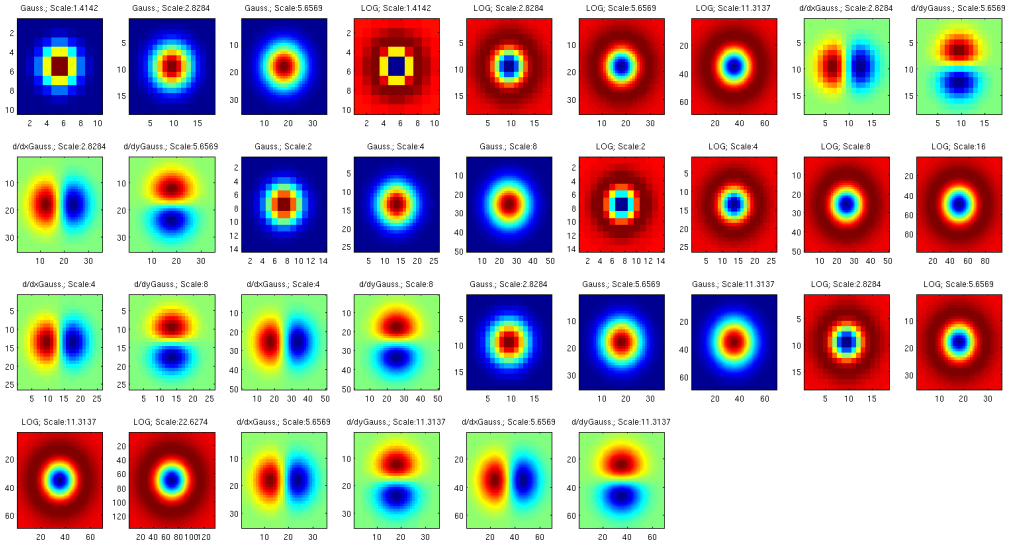


Figure 3: The provided multi-scale filter bank

[2] `http://groups.csail.mit.edu/vision/SUN/`

3 Warming up with some theory (10pts)

In this section, you will answer some questions about the most basic Bag-of-Words approach. We provide a suggested number of lines for each answer are mentioned.

Question 3.1 (3pts, 2-3 lines)

Given an $N \times N$ image, and an $h \times h$ Gaussian filter, how would you convolve the image and the filter in $O(h)$ instead of $O(h^2)$?

Question 3.2 (2pts, 2-3 lines)

Does the bag-of-words approach respect spatial information? Why?

Question 3.3 (5 pts, 2-3 lines)

For images, why is it a better idea to use filter responses as features rather than raw pixel values?

4 Representing the World with Visual Words (40pts)

We have provided you with a multi-scale filter bank that you will use to understand the visual world. You can get it with the following provided function:

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

`filterBank` is a cell array[3] Open the file `createFilterBank.m` to see how the filters were created. We have also provided you with a function to extract filter responses that takes a 3-channel RGB image and a filter bank and returns the responses of the filters on the image.

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

`filterResponses` is an $N \times M$ matrix, where N is the number of pixels in the input image, and M is the number of filter responses (three times the size of the filter bank, since you are applying it to a 3-channel image).

Question 4.1 (5 pts, 3-4 lines) Theory:

What properties do each of the filter functions pick up? You should group the filters into broad categories (i.e., all the Gaussians). Answer in your write-up.

[3]Look at MATLABs documentation for more details, but `filterBank{i}` is a 2D matrix, and `filterBank{i}` and `filterBank{j}` are not necessarily the same size.

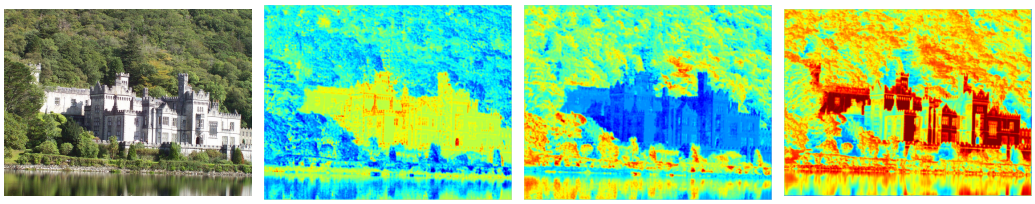
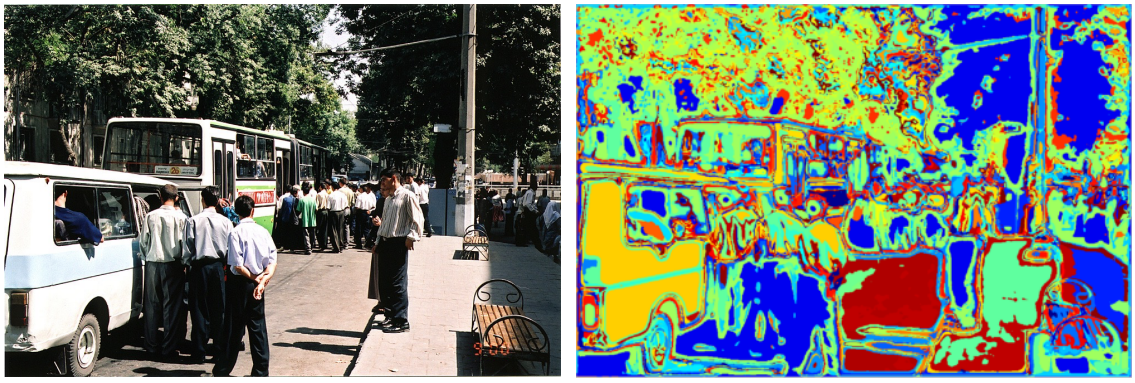


Figure 4: An input image and filter responses for each of the 3 channels



(a) Color Input (b) Output Map of Visual Words

Figure 5: A sample output, rendered with `imagesc`.

Creating Visual Words

You will now create a dictionary of visual words from the filter responses using k-means. After applying k-means, similar filter responses will be represented by the same visual word. You will use a dictionary with fixed-size. Instead of using all of the filter responses (**that can exceed the memory capacity of your computer**), you will use responses at α random pixels[4]. If there are T training images, then you should collect a matrix `filterResponses` over all the images that is $\alpha T \times N$, where N is the number of filter responses. 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(filterResponses, K, 'EmptyAction','drop');
```

Question 4.2 (15 pts)

You should write the following function to generate a dictionary given a list of images.

```
[filterBank, dictionary] = getFilterBankAndDictionary(imPaths)
```

As an input, `getFilterBankAndDictionary` takes a cell array of strings containing the full path to all images. You can load each file by iterating from `1:length(imPaths)`, and doing `imread(imPaths{i})`. Generate the αT filter responses over the training files and call

[4]Try using `randperm`.

k-means. A sensible initial value to try for K is between 100 and 300, and for α is between 50 and 150, but they depend on your system configuration and you might want to play with these values.

Question 4.3 (5 pts, 3-4 lines) Theory

How does the dictionary size affect the representation power of the bag-of-words pipeline, e.g., if I set $k = 10$ or $k = 10,000$? What is the problem of a dictionary that is too small or too large?

Once you are done with `getFilterBankAndDictionary`, call the provided script `computeDictionary`, which will pass in the file names, and go get a coffee. If all goes well, you will have a `.mat` file named `dictionary.mat` that contains the filter bank as well as the dictionary of visual words. Dont worry about "did-not-converge" errors. If it takes more than 10 minutes, reduce the number of clusters and samples. If you have debugging issues, try passing in a small number of training files manually.

Computing Visual Words

Question 4.4 (15 pts)

We want to map each pixel in the image to its closest word in the dictionary. Create the following function to do this:

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

`wordMap` is a matrix with the same width and height as I, where each pixel in `wordMap` is assigned the index of the closest visual word of the filter response at that pixel in I. We will use the standard Euclidean distance to do this; to do this efficiently, use the MATLAB function `pdist2`. A result is shown in Fig. 5.

Since this can be slow, we have provided a function `batchToVisualWords(numberOfCores)` that will apply your implementation of the function `getVisualWords` to every image in the training and testing set. This function will automatically[5] use as many cores as you tell it to use. For every image "`X.jpg`" in `images/`, there will be a corresponding file named "`X.mat`" in `wordmaps/` containing the variable `wordMap`. You are *highly* encouraged to visualize a few of the resulting word maps; they should look similar to the ones in Figs. 2,5.

5 Building a Recognition System (95pts)

We have formed a convenient representation for recognition. We will now produce a basic recognition system with spatial pyramid matching. The goal of the system is presented in Fig. 1: given an image, classify ("name") the scene where the image was taken.

[5]Interested parties should investigate `batchToVisualWords.m` and the MATLAB commands `matlabpool` and `parfor`.

Traditional classification problems follow two phases: training and testing. During training time, the computer is given a pile of formatted data (i.e., a collection of feature vectors) with corresponding labels (e.g., "grass", "sky") and then builds a model of how the data relates to the labels: "if green, then grass". At test time, the computer takes features and uses these rules to infer the label: e.g., "this is green, therefore it is grass".

In this assignment, we will use the simplest classification model: k-nearest neighbor. At test time, we will simply look at the querys top k nearest neighbors in the training set and transfer the majority label[6]. In this example, you will be looking at the query image and looking up its k nearest neighbors in a collection of training images whose labels are already known. This approach works surprisingly well given a huge amount of data, e.g., a very cool graphics applications from [3].

The components of any nearest-neighbor system are: features (how do you represent your instances?) and similarity (how do you compare instances in the feature space?). You will implement both.

Extracting Features

We will first represent an image with a bag-of-words approach. In each image, we simply look at how often each word appears.

Question 5.1 (10 pts)

Create a function `getImageFeatures` that extracts the histogram[7] of visual words within the given image (i.e., the bag of visual words).

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

As inputs, the function will take:

- `wordMap` is an $H \times W$ image containing the IDs of the visual words
- `dictionarySize` is the number of visual words in the dictionary.

As output, the function will return h, a `dictionarySize` $\times$ `1` histogram that is L_1 normalized, (i.e ., $\sum h_i = 1$). You may wish to load a single visual word map, visualize it, and verify that your function is working correctly before proceeding.

Question 5.2 Extra credit (5 pts 2-3 lines) Theory:

Why is it a good idea to normalize the histogram of visual words? (Hint: See Section 3.2 of [4])

[6] in case of a tie, use odd k'.

[7] Look at `hist` in MATLAB

Multi-resolution: Spatial Pyramid Matching

Bag of words ignores the entire spatial structure of the image. In many cases, this may not be desirable. One way to alleviate this issue (and thus can often have a better representation) is to use spatial pyramid matching. The general idea is to divide the image into a small number of cells, and concatenate the histogram of these cells to the histogram of the original image, with a proper weight per cell. We will be using the work of [4] as reference, and will point you to specific sections as needed.

Here we will implement a popular scheme that cuts the image into $2^l \times 2^l$ cells where l is the layer number. We treat each cell as a small image and count how often each visual word appears. This results in a histogram for every single cell in every layer. Finally to represent the entire image, we concatenate all the histograms together after normalization by the total number of features in the image. If there are L layers and K visual words, the resulting vector has dimensionality $K \sum_{l=0}^{L} 4^l = K \left(4^{(L+1)} - 1\right)/3$.

Now comes the weighting scheme. Note that when concatenating all the histograms, histograms from different levels are assigned different weights. Typically (in Section 3 of [4]), a histogram from layer l gets half the weight of a histogram from layer $l+1$, with the exception of layer 0, which is assigned a weight equal to layer 1. A popular choice is to set the weight for layer 0 and layer 1 to 2^{-L}, and for the rest to 2^{l-L-1} (e.g., in a three layer spatial pyramid, $L = 2$ and weights are set to 1/4, 1/4 and 1/2 for layer 0, 1 and 2 respectively, see Fig. 6). Note that the L_1 norm (absolute values of all dimensions summed up together) for the final vector is 1.

Question 5.3 Extra credit (5 pts 2-3 lines) Theory

Why do we weight different levels of the histogram differently? (Hint: See Section 3.1 of [4])

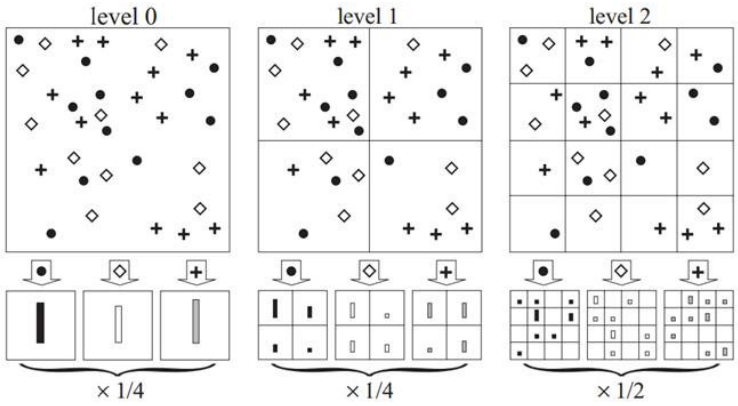


Figure 6: **Spatial Pyramid Matching:** From [4]. Toy example of a pyramid for $L = 2$. The image has three visual words, indicated by circles, diamonds, and crosses. We subdivide the image at three different levels of resolution. For each level of resolution and each channel, we count the features that fall in each spatial bin. Finally, weight each spatial histogram.

Question 5.4 (20 pts)

Create a function `getImageFeaturesSPM` that form a multi-resolution representation of the given image.

```
[h] = getImageFeaturesSPM(layerNum, wordMap, dictionarySize)
```

As inputs, the function will take:

- `layerNum` the number of layers in the spatial pyramid, i.e., $L + 1$
- `wordMap` is an $H \times W$ image containing the IDs of the visual words.
- `dictionarySize` is the number of visual words in the dictionary.

As output, the function will return `h`, a vector that is L_1 normalized. **Please use a 3-layer spatial pyramid** ($L = 2$) **for all the following recognition tasks**.

One small hint for efficiency: a lot of computation can be saved if you first compute the histograms of the *finest* layer, because the histograms of coarser layers can then be aggregated from finer ones.

Comparing Images

We will also need a way of comparing images to find the "nearest" instance in the training data. In this assignment, well use the histogram intersection similarity. The histogram intersection similarity between two histograms $x_{1:n}$ and $y_{1:n}$ is defined as $\sum_{i=1}^{n} \min(x_i, y_i)$, or `sum(min(x,y))` in MATLAB. Note that since this is a similarity, you want the *largest* value to find the "nearest" instances.

Question 5.5 (10 pts)

Create the function

```
[histInter] = distanceToSet(wordHist, histograms)
```

where `wordHist` is a K $(4^{(L+1)} - 1)/3 \times 1$ vector and histograms is a $K(4^{(L+1)} - 1)/3 \times T$ matrix containing T features from T training samples concatenated along the columns. This function returns the histogram intersection similarity between `wordHist` and each training sample as a $1 \times T$ vector. Since this is called every time you want to look up a classification, you want this to be fast, and doing a for-loop over tens of thousands of histograms is a very bad idea. Try `repmat` or (even faster) `bsxfun`. Unless youre experienced with MATLAB or confident, make sure your optimization works before moving on. Either use a few hand-made examples that you can manually verify or subtract the distances produced by the unoptimized and optimized examples.

Building A Model of the Visual World

Now that weve obtained a representation for each image, and defined a similarity measure to compare two spatial pyramids, we want to put everything up to now together.

You will need to load the training file names from `traintest.mat` and the filter bank and visual word dictionary from `dictionary.mat`. You will save everything to a `.mat` file named `trainOutput.mat`. Included will be:

1. `filterBank`: your filterbank.
2. `dictionary`: your visual dictionary.
3. `trainHistograms`: a K $(4^{(L+1)}-1)/3 \times N$ matrix containing all of the histograms of the N training images in the data set. I have a dictionary with 200 words and my `trainHistograms` is 4200×1279.
4. `trainImageLabels`: a $1 \times N$ vector containing the labels of each of the images. (i.e., so that `trainHistograms(:,i)` has label `trainImageLabels(i))`.
5. `classnames`: a $1 \times N$ cell array containing names of classes for each label. (similar to `traintest.mat`).

We have provided you with the names of the training and testing images in `traintest.mat`. You want to use the cell array of files `trainingImagePaths` for training, and the cell array of files `testImagePaths` for testing. *You cannot use the testing images for training.* To access the word maps created by `batchToVisualWords.m`, you might need function `strrep` to modify the file names.

1	2	3	4	5	6	7	8	9
bamboo_forest	basilica	dam	desert	kitchen	railroad	sky	supermarket	theatre

Question 5.6 (5 pts)

Write a function `createHistograms` to concatenate all the histograms into a single matrix (`histogram_size` $\times$ `num_images`). Refer to the script `trainSystem.m` for an idea of input/output.

```
outputHistograms = createHistograms(dictionarySize,imagePaths,wordMapDir)
```

Question 5.7 (5 pts)

Use the provided script `trainSystem.m` to produce `trainOutput.mat`, and submit the `.mat` as well as any helper functions you write. The script should run without any trouble if your implementations are correct. To qualitatively evaluate what you have done, we have provided a helper function `guessImage` that will let you get the predictions on a new image given the training data. This will give you a visual sanity check that you have implemented things correctly. Use the program as follows:

```
guessImage(absolutePathToImage)
# example - guessImage(images/bamboo forest/sun aaegrbxogokacwmz.jpg)
```

The program will load the image, represent it with visual words, and get a prediction based on nearest neighbor. The predictions will appear inside your MATLAB command window as text.

Dont worry if you get a fair number of wrong answers. Do worry if the program crashes while calling your code or if you get zero correct/all correct/all same answers. If you are getting 0% or 100% performance, go back and verify (visually) that each of your components produces correct output, or check that testing data are accidentally included during training (yet you can pick images from both training and testing set for debugging purposes).

Quantitative Evaluation

Qualitative evaluation is all well and good (and very important for diagnosing performance gains and losses), but we want some hard numbers.

Like we mentioned earlier, we use the k-nearest neighbor method for classification.

Question 5.8 (10 pts)

Write a function to classify the test samples.

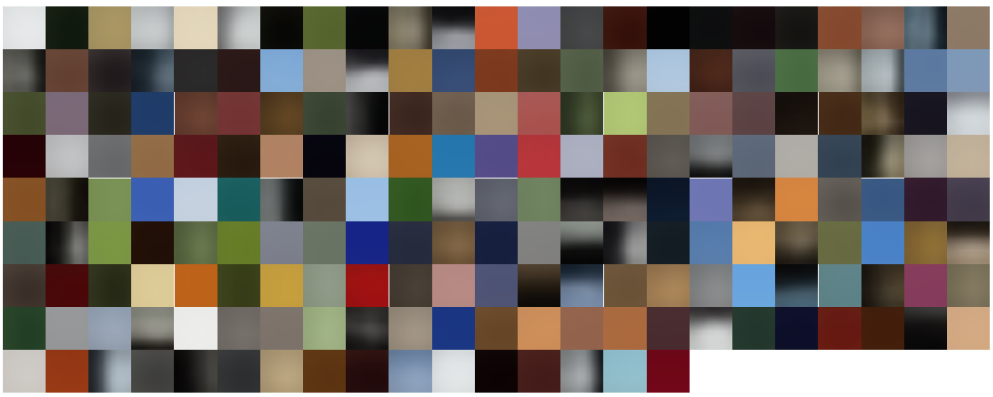


Figure 7: **Visualizing visual words:** Average 9×9 patches for visual words in the dictionary. These patches were created using a subset of 800 images of the training set, and the algorithm described in Section 6

```
[predictedLabel] = knnClassify(wordHist,trainHistograms,trainingLabels,k)
```

Load the corresponding test images and their labels, and compute the predicted labels of each. To quantify the accuracy, you will compute a confusion matrix `C`: given a classification problem, the entry `C(i,j)` of a confusion matrix counts the number of instances of class `i` that were predicted as class `j`. When things are going well, the elements on the diagonal of `C` are large, and the off-diagonal elements are small. Since there are 9 classes, `C` will be 9×9. The accuracy, or fraction of correctly classified images, is given by `trace(C) / sum(C(:))`.

Question 5.9 (10 pts)

Write a script named `evaluateRecognitionSystem.m` that tests the system and outputs the confusion matrix, and submit it as well as any helper functions you write. Report the confusion matrix for your results in your write-up. This does not have to be formatted prettily: if you are using LaTeX, you can simply copy/paste it from MATLAB into a `verbatim` environment. Compute the accuracy of classification and report that too. Use $k = 1$ for `knnClassify`. The script should print a 9×9 confusion matrix on the console (just use disp in MATLAB), followed by the accuracy.

Additionally, do not worry if your accuracy is low: with 9 classes, chance is 11.1%. To give you a more sensible number, my implementation *w*ithout spatial pyramid matching gives an overall accuracy of about 68%.

Question 5.10 (10 pts)

Vary the values of k in `knnClassify` and report the prediction accuracies. Is there any variation in the accuracies? Can you explain this variation? Plot a graph of k vs the classification accuracy.

Question 5.11 (5 pts, 2-3 lines): Theory

What are some practical/computational issues with using nearest neighbor classification for a large dataset?

6 Final thoughts on visual words (40 pts)

Now that we have a basic working pipeline with some hard numbers, it is time to dig into it. *Caution* Attempt this section only if you have finished all the above parts.

Visualizing the Visual Words

The first question we want to answer is what exactly does our dictionary represent? What do the visual words look like? To answer this question, we will write our own rough visualization procedure. The basic idea is to look at small patches of images that correspond to each of the dictionarys visual word. Briefly,

- For a visual word, look at small 9×9 pixel patches in which the center pixel gets assigned to this word. Find many such patches per visual word by going over multiple images.
- Take these 9×9 patches and compute their average. This average image patch is roughly a visualization of the visual word.
- Compute these "averaged patch" visualizations for each visual word in your dictionary. Thus for dictionary size N, you have N "averaged" patches.

Fig. 8 shows a few patches for a dictionary. Your patches will look different.

Figure 8: Input images and visualizations of top 10 visual words.

Question 6.1 (20 pts)

Implement the above "averaged patch" in the script visualizeWords.m. Computing this visualization will be time consuming, so be warned. Once you are done with this, plot your visualization like Fig. 7 and include it in your writeup. Try to understand what this visualization means, and describe it in 3-4 lines.

Figure generation tips: You may use `imdisp`[8] for displaying multiple images/patches. You may also use `export_fig`[9] for saving high quality images of your plots. Fig. 7 was plotted using `imdisp` and saved using `export_fig`.

Question 6.2 (10 pts)

For any 2 images from each class, look at the histograms of these images and plot the top 10 visual word patches (like in Fig. 8). Can you explain these visual words? (by top 10, I mean the visual words with top 10 highest values in the histogram. You can set $L = 0$ for spatial pyramid.)

Question 6.3 (10 pts, 4-5 lines): Dataset Expansion

Classifiers usually work better as the data set size increases, so one option is to just get more data. Yet with just the data we give you, can you think of a way to expand the data set without using external images? A popular way is to left-right flip the image. Will this work under the bag of words representation? How about bag of words with spatial pyramid matching? Why or why not? Include the answers in your write-up (no implementation required).

7 Extra Credit: Boosting performance (45 pts)

This section is for extra credit, and therefore **completely optional.**

We want you to experiment with a variety of ways to possibly improve results. Place any extra-credit work/data in the folder named `improved/},`and include a brief write-up of your

[8] `http://www.mathworks.com/matlabcentral/fileexchange/22387-imdisp`
[9] `http://www.mathworks.com/matlabcentral/fileexchange/23629-export-fig`

collected data with your findings. Use the same structure for training and testing (in fact, please copy from `baseline/})`and submit all scripts and helper functions. Restrict yourself to MATLAB and any of its common toolboxes. Also include 2 of your good results, as well as 2 failure cases, and some thoughts on them. Note that failures are often more illustrative and informative than successes.

Below is a list of more things you can do to boost your performance. For each addition you make (i.e., improved filter bank), report the results in a confusion matrix, and **explain why you think the results changed (or didnt)** It is OK if you are not completely positive, but do give a plausible explanation **(at least 4-5 full sentences of actual explanation)**: gains in understanding are much more important than gains in performance. *Important*: it is fine if performance does not change or decreases in terms of overall accuracy. Please report these results as well; they count as much as a performance gain.

For each section you attempt make a subdirectory inside `improved/`. e.g. If I attempt sections 5.1 and 5.3, my submission will have the directories improved/7.1/ and improved/7.3/.

Question 7.1 Better filter bank (10 pts)

Add filters to your filter bank. Explain in your write-up what each filter (or class of filters) is supposed to pick up on. Add at least two families of filters. Show your filter bank as we did in Fig. 3 and include this in the write-up[10].

Question 7.2 Better image similarity function (5 pts)

Histogram intersection is nice, but there might be different distance and similarity metrics you can use: e.g., $\mathcal{X}^2$, Hellinger, Euclidean. Try at least one.

Question 7.3 Different Features (15 pts)

The bag of words representation is not tied to the particular features were using: implement a new feature, and see what results you get. Feel free to choose the features talked in class. Clarification: You are free to use another persons code for feature extraction for SIFT, SURF, etc.; however, you must implement quantization (i.e., mapping to visual words). If your feature does not require quantization, you must implement it yourself.

Question 7.4 Encoding (15 pts)

The bag of words representation here follows a hard assignment that one feature is solely assigned to one visual word. Studies have shown that soft assignment can result in better performance [1]. Try one from the literature (provide the reference in your write-up) or devise your own.

[10] Hint: `subplot`

Question 7.5 Getting Fancy! (0 pts)

There are other classifiers which may improve performance (e.g., using SVMs or Random Forests); however, these require deeper knowledge about machine learning, and therefore will not count toward your credit. We will have a Hall of Fame for the best performers or the most creative ideas :)

References

[1] K. Chatfield, V. S. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *Proceedings of the British Machine Vision Conference (BMVC)*, volume 2, page 8, 2011.

[2] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1458–1465. IEEE, 2005.

[3] J. Hays and A. A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics (TOG)*, 26(3):4, 2007.

[4] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2169–2178. IEEE, 2006.

[5] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157. Ieee, 1999.

[6] L. W. Renninger and J. Malik. When is scene identification just texture recognition? *Vision research*, 44(19):2301–2311, 2004.

[7] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1800–1807. IEEE, 2005.

[8] J. Xiao, J. Hays, K. Ehinger, A. Oliva, A. Torralba, et al. Sun database: Large-scale scene recognition from abbey to zoo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3485–3492. IEEE, 2010.

8 HW2 Distribution Checklist

After unpacking `homework2.zip},`you should have a folder `hw2}`containing one folder for each system you might implement (`baseline,`custom, application). In the `baseline}`folder, where you will primarily work, you will find:

- `images/`: a directory containing `.jpg` images from SUN database.
- `batchToVisualWords.m`: a provided script that will run your code to convert all the images to visual word maps.
- `computeDictionary.m`: a provided script that will provide input for your visual word dictionary computation.
- `createFilterBank.m`: a provided function that returns a cell array of filters.
- `guessImage.m`: a provided script that will let the computer play the guessing-scene game.
- `extractFilterResponses.m`: a provided function that takes a 3-channel RGB image and filter bank and provides the filter responses at each pixel.
- `RGB2Lab.m`: a provided helper function.
- `traintest.mat`: a `.mat` file with the filenames of the training and testing set.
- `trainSystem.m`: a script that contains function calls to train the recognition system. The file `traintest.mat` contains the following variables

  `allImageLabels`: image labels for each image (both train and test).

  `allImagePaths`: paths for each image (both train and test).

  `classnames`: a cell array containing names of all the classes. `classnames(allImageLabels(i))` gives the class name for image `i`.

The file also contains paths/labels for `train/test` which you will be using. I have provided paths/labels for a small subset of the data in `smallTrain***/smallTest***`. **Use these only for debugging.**

9 HW2 Submission Checklist

Before you submit, please read and make sure that

- All your submission is inside a zip file named with your net id. e.g. if the net id is bovik, then `bovik.zip`.
- Your writeup is named with your net id. It is in the `pdf` format.
- Upload your `zip` file! No digital dropbox or links please.

- You have attempted all questions.
- Your submission contains at least the folder baseline with working code. Your code should assume only the existence of `traintest.mat`
- `trainSystem` followed by `evaluateRecognitionSystem` runs without any errors whatsoever. The final output of running these scripts is a 9×9 confusion matrix and accuracy.
- `visualizePatches.m` runs without any errors.
- The `images/` directory is deleted.
- All `.mat` files in baseline except `trainOutput.mat` are deleted.
- baseline folder cannot contain any 3rd party code. It contains code written for

  `createHistograms.m`

  `distanceToSet.m`

  `evaluateRecognitionSystem.m`

  `getFilterBankAndDictionary.m`

  `getImageFeatures.m`

  `getImageFeaturesSPM.m`

  `getVisualWords.m`

  `knnClassify.m`

  `visualizeWords.m`
- All images that you want us to see are in your pdf. We will not open individual images.
- `improved/` directories if you attempt extra credit. Again, make sure to have scripts `trainSystem`, `evaluateRecognitionSystem` which run without errors.
- `improved/` may contain small `.mat` files (¡ 10MB total) and 3rd party code for feature generation.