16-720 Computer Vision: Homework 2
Spatial Pyramid Matching for Scene Classification

Instructor: Martial Hebert
TAs: Xinlei Chen, Arne Suppé, Jacob Walker, Feng Zhou

Due: Thursday, October 3, 11:59:59.$\bar{9}$ a.m.

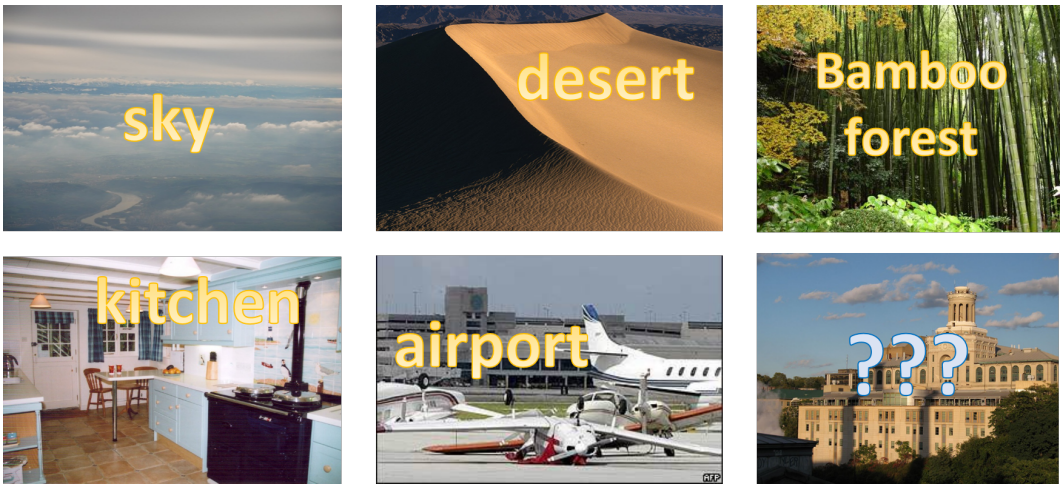


Figure 1: **Scene Classification:** Given an image, can a computer program determine where it was taken? In this homework, you will build a representation based on bags of visual words and use spatial pyramid matching for classifying the scene categories.

Instructions/Hints

1. Please pack your system and write-up into a single file named <**AndrewId**>**.zip**, see the complete submission checklist in the overview.
2. All questions marked with a **Q** require a submission.
3. **For the implementation part, please stick to the headers, variable names, and file conventions provided.**
4. **Start early!** This will take longer than the last homework and cannot be debugged as easily since there are multiple inter-connected components.
5. **Attempt to verify your implementation as you proceed:** If you don't verify that your implementation is correct on toy examples, you will risk having a huge mess when you put everything together.
6. If you have any question, please contact: **Xinlei Chen**, enderchen@cs.cmu.edu

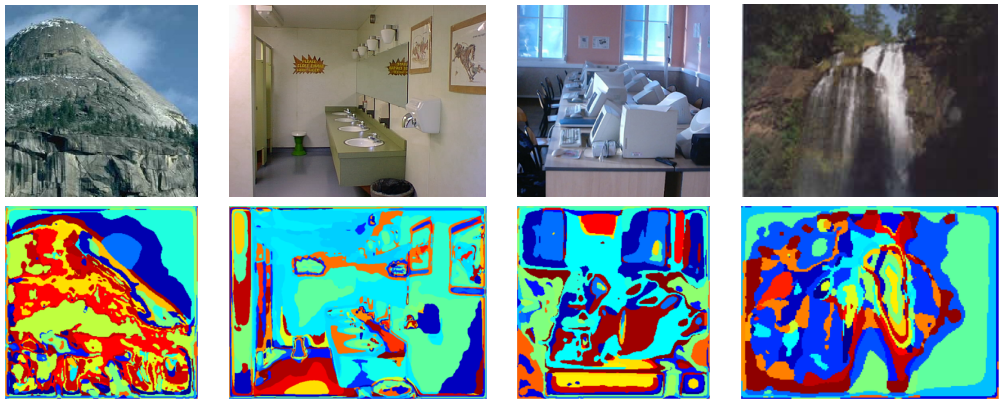


Figure 2: Visual words over images. You will use the spatially un-ordered distribution of visual words in a region (a bag of visual words) as a feature for scene classification, with some coarse information provided by spatial pyramid matching [5].

Overview

The bag-of-words (BoW) approach, which you learned about in class, has been applied to a myriad of recognition problems in computer vision. For example, two classic ones are object recognition [6, 9] and scene classification [7, 10][1].

Beyond that, the BoW representation has also been the subject of a great deal of study aimed at improving it, and you will see a large number of approaches that remain in the spirit of bag-of-words but improve upon the traditional approach which you will implement here. For example, two important extensions are pyramid matching [2, 5] and feature encoding [1].

What you will be doing:

1. In the first part, you will implement a scene classification system that uses the bag-of-words approach with its spatial pyramid extension. The paper that introduced the pyramid matching kernel [2] is:

   K. Grauman and T. Darrell. *The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features.* ICCV 2005. `http://www.cs.utexas.edu/~grauman/papers/grauman_darrell_iccv2005.pdf`

   Spatial pyramid matching [5] is presented at:

   S. Lazebnik, C. Schmid, and J. Ponce, *Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories*, CVPR 2006. `http://www.di.ens.fr/willow/pdfs/cvpr06b.pdf`

   You will be working with a subset of the SUN database[2]. The data set contains 1600 images from 8 scene categories like "kitchen", "sky" and "desert". And to build a recognition system, you will:

[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.

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

- first, take responses of a filter bank on images and build a dictionary of visual words;
- then, learn a model for the visual world based on the bag of visual words (with spatial pyramid matching [5]), and use nearest-neighbor to predict scene classes in a test set.

Note if this first part looks lengthy/daunting to you, **DON'T panic**! We provide you a step-by-step instruction to implement a working scene recognition framework, there are actually not that many lines of code to write. However, it may take *a few hours* to finish running the baseline system, so make sure to try **each component** on **a subset of the data set** first before putting everything together.

2. In the second part, we ask you to try to improve the performance of your algorithm via a number of different ways, and comment on the results you get.

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 matching system that you implement through this homework;
2. `custom/`: the custom system that you design to boost the performance of your system;
3. `application/`: (if you do the extra credit): an application that uses your recognition system;
4. a write-up (`.pdf` format).

When you submit, submit a copy of your system with the folder `images/` deleted, as well as any large temporary files that we did not ask you to create, and put it into a single file named <AndrewId>.zip.

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 5.

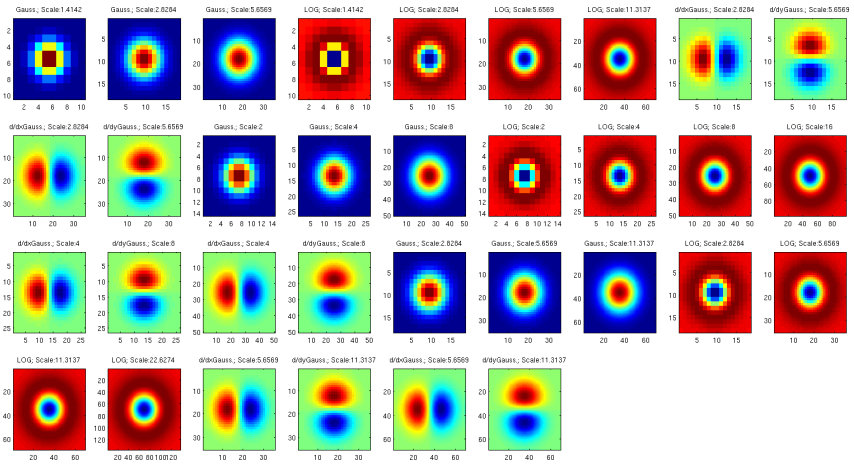


Figure 3: The provided multi-scale filter bank

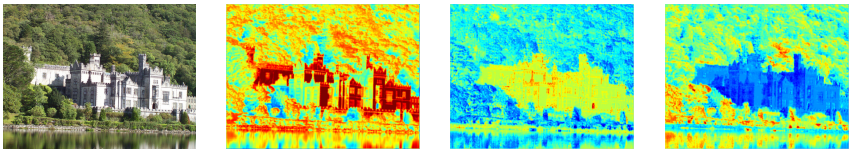


Figure 4: An input image and filter responses for each of the 3 channels

1 Representing the World with Visual Words

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] We have also provided you with a function to extract filter responses that takes a 3-channel RGB image and filter bank and returns the responses of the filters on the image.

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

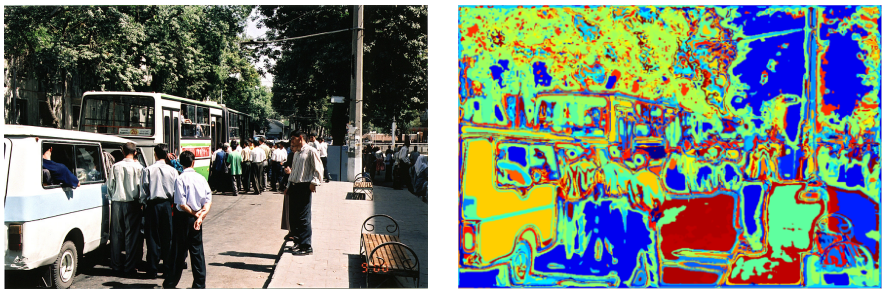
`filterResponses` is a $N \times M$ matrix, where N is the number of pixels in the input image, and M is the number of filter responses (three times the size of the filter bank, since you are applying it to a 3-channel image).

Q1.0 (5 points): 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.

1.1 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.

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



Color input Output map of visual words

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

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');
```

Q1.1 (15 points): 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 an image. 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 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.

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. If all goes poorly, you will have an error message[5]. 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.

1.2 Computing Visual Words

Q1.2 (15 points): 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)
```

[4]Try using `randperm`.

[5]Don't worry about "did-not-converge" errors.

`wordMap` is a matrix with the same width and height as I, where each pixel in `wordMap` is assigned the closest visual word of the filter response at the respective 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[6] 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.

2 Building a Recognition System

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 (colloquially, "name") the scene where the image was taken.

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.*, "kitchen", "sky") and then builds a model of how the data relates to the labels: "if blue, then sky". At test time, the computer takes features and uses these rules to infer the label: *e.g.*, "this is blue, so therefore it is sky".

In this assignment, we will use the simplest classification model: nearest neighbor. At test time, we will simply look at the query's nearest neighbor in the training set and transfer that label. In this example, you will be looking at the query image and looking up its nearest neighbor 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.

2.1 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.

Q2.1 (10 points): 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 a `H` × `W` image containing the IDs of the visual words

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

[7] Look into `hist` in MATLAB

- `dictionarySize` is the maximum visual word ID (*i.e.*, the number of visual words, the dictionary size)

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.

2.2 Multi-resolution: Spatial Pyramid Matching

Bag of words ignore the entire spatial structure of the image, which might be suboptimal. One way to alleviate this issue (and thus can often have a better representation) is to use spatial pyramid matching [5]. 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.

Here we will implement a popular scheme that chops 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 [5]), 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 for layer 0 and layer 1 the weight is set to 2^{-L}, and for the rest it is set 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.

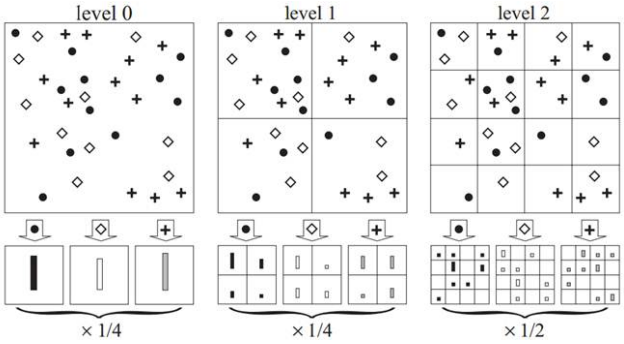


Figure 6: **Spatial Pyramid Matching:** From [5]. Toy example of a pyramid for L = 2. The image has three visual words, indicated by circles, diamonds, and crosses. We subdivide the image at three different levels of resolution. For each level of resolution and each channel, we count the features that fall in each spatial bin. Finally, weight each spatial histogram.

Q2.2 (20 points): 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 a `H` $\times$ `W` image containing the IDs of the visual words
- `dictionarySize` is the maximum visual word ID (*i.e.*, the number of visual words, the dictionary size)

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.

2.3 Comparing images

We will also need a way of comparing images to find the "nearest" instance in the training data. In this assignment, we'll 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" instance.

Q2.3 (10 points): Create the function `distanceToSet`

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

where `wordHist` is a $K\left(4^{(L+1)}-1\right)/3 \times 1$ vector and `histograms` is a $K\left(4^{(L+1)}-1\right)/3 \times T$ 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`[8].

2.4 Building A Model of the Visual World

Now that we've 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 `vision.mat`. Included will be:

1. `filterBank`: your filterbank.
2. `dictionary`: your visual word dictionary.
3. `featureTrs`: a $K\left(4^{(L+1)}-1\right)/3 \times N$ matrix containing all of the histograms of the N training images in the data set. I have a dictionary with 150 words and my `featureTrs` is 3150×1483.

[8] As a recommendation: unless you're 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.

4. `classTrs`: a $1 \times N$ vector containing the labels of each of the images. (*i.e.*, so that `featureTrs(:,i)` has label `classTrs(i)`).

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 `imTrs` for training, and the cell array of files `imTss` 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.

You may also wish to convert the labels to meaningful categories. Here is the mapping (a variable named `mapping` is included in `traintest.mat`):

1	2	3	4	5	6	7	8
airport	auditorium	bamboo_forest	campus	desert	football_field	kitchen	sky

Q2.4 (15 points): Write a script named `buildRecognitionSystem.m` that produces `vision.mat`, and submit it as well as any helper functions you write.

To qualitatively evaluate what you have done, we have provided a helper function 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)`, (*e.g.*, `guessImage('more/CMU-1.jpg')`)

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

Don't worry if you get a fair amount 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).

We have provided you with some images crawled from Google Image with query "Carnegie Mellon University" in `more/`. See what CMU is classified as by your system!

2.5 Quantitative Evaluation

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

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 8 classes, `C` will be 8×8. The accuracy, or percent of correctly classified images, is given by `trace(C) / sum(C(:))`.

Q2.5 (15 points): 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. Additionally, do not worry if your accuracy is low: with 8 classes, chance is 12.5%. To give you a more sensible number, my implementation *without* spatial pyramid matching gives an overall accuracy of $\tilde{6}5\%$.

3 Part II: Thoughts and More Improvement

We want you to experiment with a variety of ways to possibly improve results. Please place all of your data / experiments on improving performance in a new folder named `custom/`. 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.

Q3.1 (15 points): 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).

Q3.2 ($\geq$ 20 points): Below is a list of more things you can do to boost your performance. **Pick some subset of the following adding up to at least** 20 **points.** 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 didn't) the approach worked.** It is OK if you are not completely positive, but do give a plausible explanation **(at least three-five 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.

- **Better filter bank (5 points):** 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[9].

- **Better image similarity function (5 points):** histogram intersection is nice, but there might be different distance and similarity metrics you can use: *e.g.*, χ^2, Hellinger, Euclidean. Try at least one.

- **Different Features (15 points):** the bag of words representation is not tied to the particular features we're 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 person's 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. If you are unsure, please check with Xinlei Chen.

- **Encoding (15 points):** 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.

  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.

[9] Hint: `subplot`

4 Extra Credit - Your application

QX1 ($\leq$ 30 points):
Spatial pyramid matching with the bag of words representation is a powerful tool for generic image classification. Beyond boring object recognition and scene classification, there are a whole lot of interesting things to explore with this tool! For example, can it distinguish images from one dataset and images from another (*i.e.*, dataset bias [8]), memorable images from forgettable ones [4], paintings from different artists, product images and real world ones (*e.g.*, images on Amazon and pictures taken with a personal camera), images by professional photographers and ones found in a person's album? Collect a set of images and apply the framework to it as your baseline. Does it work? Can you modify part of the pipeline so that it works better? Write down your ideas and try at least one.

Place any extra-credit work/data in the folder named `application/`, and include a brief write-up of your collected data with your findings. 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.

References

[1] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *British Machine Vision Conference*, 2011.

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

[3] James Hays and Alexei A Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), 2007.

[4] Phillip Isola, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. What makes an image memorable? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 145–152, 2011.

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

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

[7] Laura Walker Renninger and Jitendra Malik. When is scene identification just texture recognition? *Vision research*, 44(19):2301–2311, 2004.

[8] A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 1521–1528, Washington, DC, USA, 2011. IEEE Computer Society.

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

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

5 HW2 Distribution Checklist

After unpacking `hw2.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.
- `more/`: a dataset containing some images that are collected from Google for qualitative evaluation.
- `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.