CS549/RBE549 Computer Vision
Scene Recognition with Bag of Words
Assignment 4

John S. Nafziger, Ph.D.
Adjunct Professor, Computer Science
jsnafziger@wpi.edu

Due: 27-November-2017 **NO EXCEPTIONS**

1 Overview

We will perform scene recognition with three different methods (Figure 1). We will classify scenes into one of 15 categories (Figure 2) by training and testing on the 15 scene database from (Lazebnik et al. 2006). Assignment 4 starter code and data available at DropBox Site: `https://www.dropbox.com/sh/itzz2unbt8w6e8s/AADkDbvmplyJdrdFWNoBwXaNa`

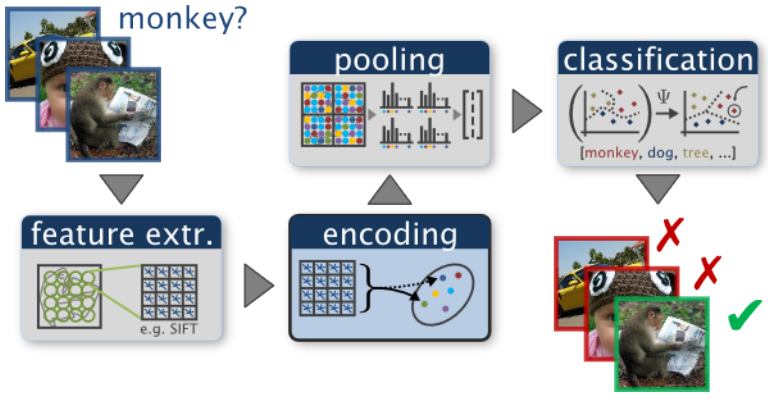

Figure 1: An example of a typical bag of words classification pipeline. Figure by Chatfield et al.

Figure 2: Example scenes from of each category in the 15 scene dataset. Figure from Lazebnik et al. 2006.

1.1 Task Overview

In this assignment, you are required to implement three scene recognition schemes. The `m-files` highlighted below are provided in starter code:

- Tiny images representation (`get_tiny_images.m`) and nearest neighbor classifier (`nearest_neighbor_classify.m`).
- Bag of words representation (`build_vocabulary.m, get_bags_of_words.m`) and nearest neighbor classifier.
- Bag of words representation and linear SVM classifier (`svm_classify.m`).

1.2 Useful and Forbidden Built-in MATLAB functions

- Potentially useful: `extractHOGFeatures()` and others, `kmeans(), fitclinear(), fitcsvm(), predict(), pdist2()`.
- Forbidden functions: `bagOfFeatures(), evaluateImageRetrieval()`, etc.

2 Scene Recognition Details

2.1 Tiny Images and Nearest Neighbor Classification

Start by implementing the tiny image representation and the nearest neighbor classifier. They are easy to understand, easy to implement, and run very quickly for our experimental

setup (less than 10 seconds). The tiny image feature, by Torralba et al., is one of the simplest possible image representations. One simply resizes each image to a small fixed resolution (we recommend 16x16). It works slightly better if the tiny image is made to have zero mean and unit length. This is not a particularly good representation, because it discards all of the high frequency image content and is not especially invariant to spatial or brightness shifts. Torralba et al. propose several alignment methods to alleviate the latter drawback, but we will not worry about alignment for this project. We are using tiny images simply as a baseline of performance with which to compare more sophisticated representations. See `get_tiny_images.m` for more details.

The nearest neighbor classifier is equally simple to understand. When tasked with classifying a test feature into a particular category, we simply find the "nearest" training example (L2 distance is a sufficient metric) and assigns to the test case the label of the nearest training example. The nearest neighbor classifier has many desirable features: it requires no training, it can represent arbitrarily complex decision boundaries, and it trivially supports multiclass problems. However, it is vulnerable to training noise, which can be alleviated by voting based on the K nearest neighbors (but you are not required to do so). Nearest neighbor classifiers also suffer as the feature dimensionality increases, because the classifier has no mechanism to learn which dimensions are irrelevant for the decision. See `nearest_neighbor_classify.m` for more details.

Together, the tiny image representation and nearest neighbor classifier will achieve 15-25% accuracy on the 15 scene database. For comparison, chance performance is ˜7%.

2.2 Bag of Words

Bag of words models are a popular technique for image classification inspired by models used in natural language processing. The model ignores or downplays word arrangement (spatial information in the image) and classifies based on a histogram of the frequency of visual words. The visual word "vocabulary" is established by clustering a large corpus of local features. See Szeliski chapter 14.4.1 for more details on category recognition with quantized features. In addition, 14.3.2 discusses vocabulary creation and 14.1 covers classification techniques.

After we have implemented a baseline scene recognition pipeline, we shall move on to a more sophisticated image representation: bags of quantized feature descriptors. Before we can represent our training and testing images as bags of feature descriptors, we first need to establish a *vocabulary* of visual words, which will represent the similar local regions across the images in our training database. We will form this vocabulary by clustering with k-means the many thousands of local feature vectors from our training set.

Note: This is not the same as finding the k-nearest neighbor feature descriptors! For example, given the training database, we might start by clustering our local feature vectors into k=50 clusters. The centroids of these clusters are our visual word vocabulary. For any new feature point we observe, we can find the nearest cluster to know its visual word. As it can be slow to sample and cluster many local features, the starter code saves the cluster

centroids and avoids recomputing them on future runs. *Be careful* of this if you re-run with different parameters. See `build_vocabulary.m` for more details.

Now we are ready to represent our training and testing images as histograms of visual words. For each image we will densely sample many feature descriptors. Instead of storing hundreds of feature descriptors, we simply count how many feature descriptors fall into each cluster in our visual word vocabulary. This is done by finding the nearest neighbor k-means centroid for every feature descriptor. Thus, if we have a vocabulary of 50 visual words, and we detect 220 features in an image, then our bag of words representation will be a histogram of 50 dimensions where each bin counts how many times a feature descriptor was assigned to that cluster and sums to 220. The histogram should be normalized so that image size does not dramatically change the bag of feature magnitude. See `get_bags_of_words.m` for more details.

You should now measure how well your bag of words representation works when paired with a nearest neighbor classifier. There are *many* design decisions and free parameters for the bag of words representation (number of clusters, sampling density, sampling scales, feature descriptor parameters, etc.) so performance might vary from 50% to 60% accuracy.

2.3 Multi-class SVM

The last task is to train 1-vs-all linear SVMs to classify the bag of words feature space. Linear classifiers are a simple learning model. The feature space is partitioned by a learned hyperplane and test cases are categorized based on which side of that hyperplane they fall on. Despite this model being far less expressive than the nearest neighbor classifier, it will often perform better. For example, maybe in our bag of words representation 40 of the 50 visual words are uninformative. They simply don't help us make a decision about whether an image is a 'forest' or a 'bedroom'. Perhaps they represent smooth patches, gradients, or step edges which occur in all types of scenes. The prediction from a nearest neighbor classifier will still be heavily influenced by these frequent visual words, whereas a linear classifier can learn that those dimensions of the feature vector are less relevant and thus downweight them when making a decision.

There are numerous methods to learn linear classifiers, but we will find linear decision boundaries with a support vector machine. You do not have to implement the support vector machine. However, linear classifiers are inherently binary and we have a 15-way classification problem. To decide which of 15 categories a test case belongs to, you will train 15 binary 1-vs-all SVMs. 1-vs-all means that each classifier will be trained to recognize 'forest' vs. 'non-forest', 'kitchen' vs. 'non-kitchen', etc. All 15 classifiers will be evaluated on each test case and the classifier which is most confidently positive "wins". E.G. if the 'kitchen' classifier returns a score of -0.2 (where 0 is on the decision boundary), and the 'forest' classifier returns a score of -0.3, and all of the other classifiers are even more negative, the test case would be classified as a kitchen even though none of the classifiers put the test case on the positive side of the decision boundary. When learning an SVM, you have a free parameter 'lambda' which controls the model regularization. Your accuracy will be very sensitive to

lambda, so be sure to test many values. See `svm_classify.m` for more details.

Now you can evaluate the bag of words representation paired with 1-vs-all linear SVMs. Accuracy should be from 50% to 60% depending on the parameters.

3 Code and Data Details

Assignment 4 starter code and data available at DropBox Site: `https://www.dropbox.com/sh/itzz2unbt8w6e8s/AADkDbvmplyJdrdFWNoBwXaNa`

- Initial performance: Running `projSceneRecBoW.m` unchanged will randomly guess the category of every test image. This achieves ~7% accuracy as, by chance with 15 classes, ~1 out of every 15 guesses is correct.
- Data: The starter codes trains and tests on 100 images from each category (i.e. 1500 training examples total and 1500 test cases total). In a real research paper, one would be expected to test performance on random splits of the data into training and test sets, but the starter code does not do this to ease debugging.
- Evaluation and Visualization: The starter code function `create_results_webpage.m` builds a confusion matrix and visualizes your classification decisions by producing a table of true positives, false positives, and false negatives as a webpage each time you run `projSceneRecBoW.m`.

4 Writeup

Please submit a project report (PDF) describing your algorithm and any decisions you made to write your algorithm a particular way. Use naming convention:
FirstName_LastName_HW4.pdf. Show and discuss the results of your algorithm.

- Report your classification performance for the three recognition pipelines: tiny images + nearest neighbor, bag of words + nearest neighbor, and bag of words + one vs. all linear SVM.
- For your best performing recognition setup, please include in your report the full confusion matrix and the table of classifier results produced by the starter code `results_webpage/index.ht`
- Combine the results html file, the code and project report into **FirstName_LastName_HW4.zip** before uploading to canvas. Include all the code you implement for each of the sections. Remember to set relative paths so that we can run your code out of the box.

5 Rubric (80 Points Total)

- +05 pts: Build tiny image features for scene recognition. (`get_tiny_images.m`)

- +10 pts: Nearest neighbor classifier. (`nn_classify.m`)
- +20 pts: Build a vocabulary from a random set of training features. (`build_vocabulary.m`)
- +20 pts: Build histograms of visual words for training and testing images. (`get_bags_of_words.m`)
- +20 pts: Train 1-vs-all SVMs on your bag of words model. (`svm_classify.m`)
- +05 pts: Writeup with design decisions and evaluation.

6 Notes

Lazebnik et al. 2006 is a great paper to read, although we will be implementing the *baseline method* the paper discusses (equivalent to the zero-level pyramid) and not the more sophisticated spatial pyramid (which is extra credit). For an excellent survey of modern feature encoding methods for bag of words models, please see Chatfield et al. 2011.

7 Computation Speed

Extracting features, clustering to build a universal dictionary, and building histograms from features can be slow. A good implementation can run the entire pipeline in less than 10 minutes, but this may be at the expense of accuracy (e.g., too small a vocabulary of visual words or too sparse a sampling rate). Save intermediate results and/or make use of Matlab's code sections functionality if you are trying to fine tune one part of the pipeline.

8 Credits

Project description and code by James Hays and Sam Birch. Figures in this handout from Chatfield et al. and Lana Lazebnik.