Problem Set 5

CMSC 733
Assigned Tuesday November 25, 2014
Due Tuesday, December 9, 2014

Bag of visual words image classification

For this problem set you will design and implement the baseline algorithm for bag-of-words classification. The dataset we are going to use is Caltech 101, which can be found at: 101_ObjectCategories.tar.gz (131Mbytes). To help you start and focus on algorithm design, we provide ps5.m with skeleton code that handles data import and organization. Simply download and extract the data to the same folder as the ps5.m file. Adjust the 'dataDir' variable according to your path to the Caltech 101 data.

You are expected to fill all places marked by 'TODO' in the ps5.m file. Everything should be implemented from scratch, except as noted below. (Note, it is not allowed to consult or make use of code that you might find on-line, beyond the code we provided).

Note: we provide the skeleton code to help you develop the algorithm more easily. You are free to not follow the exact format for each function. If you depart from this format, please document your choices.

1. **20 points** Write a function to extract feature descriptors from an image. This should have the form:

```
function descr = computeFeature(im,method)
```

'im' is the image, 'method' is either 'sift' or 'filter-bank'. You have to implement **BOTH**.

'sift':

You can use 'vl_feat' to calculate SIFT descriptors in the image. You need to have the vl_feat software package installed, which you should have already done for the Mosaicing project.

'filter-bank':

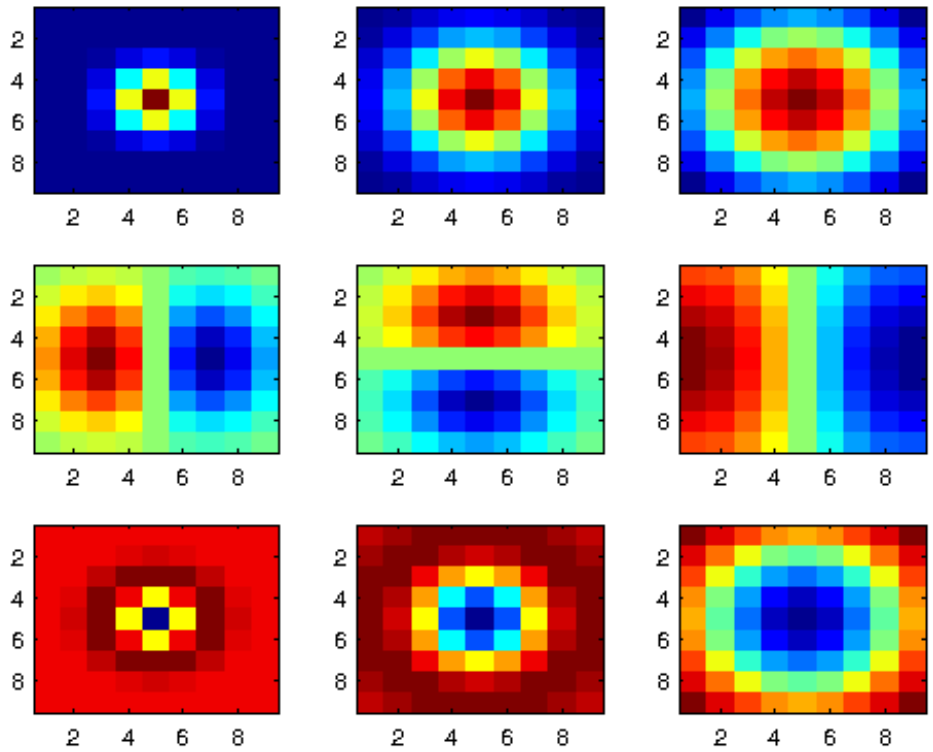
Consult this paper:
http://research.microsoft.com/pubs/67408/criminisi_iccv2005.pdf (last paragraph, Section 3).

The filter bank is made of 3 Gaussians (σ = 1,2,4), 4 Laplacian of Gaussians (σ = 1,2,4,8) and 4 first order derivatives of Gaussians (σ = 2,4 for x and y). You need to first convert the image to L*a*b space. You can do that by using MATLAB built-in function. See reference here:
http://www.mathworks.com/help/images/converting-color-data-between-color-spaces.html

After conversion, you can now convolve the image with each filter and aggregate responses for all pixels together to form the descriptor.

Specifically, the resulting 'descr' should be a 17-by-M matrix, where M = size(im,1)*size(im,2).

You can try different sizes and variances for those filters to achieve best performance. Your filter bank might look like this:



Test your filters on some image and show the response map.

2. **20 points** Create a function that will learn vocabulary/dictionary/visual words from training data. This can have the form:

```
function vocab = getVocab(numWords,Train,…)
```

You can use K-means to cluster the descriptors collected from training data. You can use 'vl_kmeans' function in the vlfeat package for this task. Here K equals 'numWords'. 'vocab' is a d-by-numWords matrix, where d is the length of the visual descriptor. 'Train' is a structure that stores feature descriptors for all classes (See code for detailed comments). '…' means you may add other arguments if necessary. For example, you can pass in a parameter called 'ratioPerClass' to control the portion of data used in each class to learn the vocabulary.

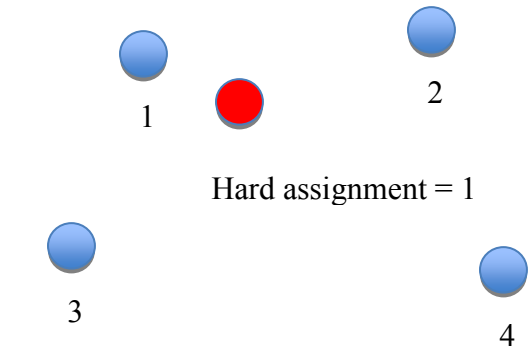
3. **20 points** Create a function that produces a histogram of visual words for a given image (the encoding step). This could have the form:

```
function hist = encoding(vocab,descr,method,…)
```

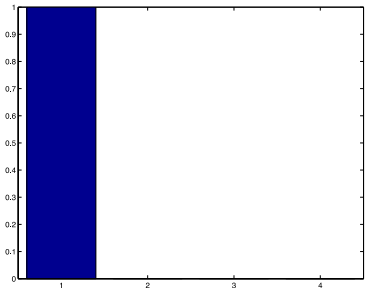
'vocab' is the vocabulary we just learned. 'descr' is the descriptors extracted from an image. The method is either 'nearest-neighbor' or 'local-encoding', you have to implement **BOTH**:

'nearest-neighbor':

Each descriptor is assigned to the closest visual words in vocab. This is a hard assignment. See the following example for illustration. The red ball indicates one testing feature. Blue balls are learned visual words. In nearest-neighbor encoding, we set the histogram to be [1 0 0 0] for this feature, since #1 visual word is closest to the testing feature.



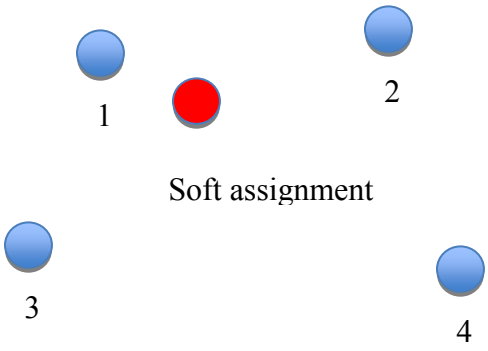
Histogram:

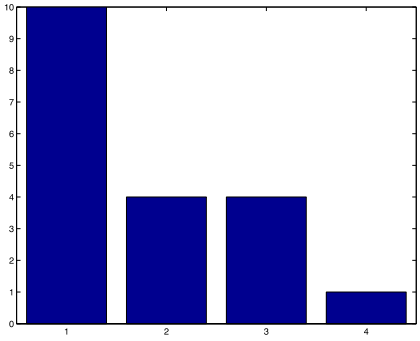


'local-encoding':

Instead, we can assign the k-nearest visual words to a descriptor, and produce a weighted histogram based on their distance to the descriptor. This is more like soft assignment. This method shares similar ideas with Locality-constrained Linear Coding (LLC), but is much simpler.

You may find local-encoding working better than nearest-neighbor in some classes. For example, you might see a 10% accuracy improvement in class 'rhino'.



Histogram: 

In this example, we have the same testing feature and visual words as above. Instead of hard assignment, we first compute the distance from the testing feature to the k-nearest visual words. Suppose k=4 for this case, we then build the histogram to be [10 4 4 1] so that every visual word contributes to the histogram. The weight of contribution reflects how similar each visual word is to the testing feature.

You can try different ways to calculate the weight. For example, you can take the reciprocal of Euclidean distance, or fit a Gaussian kernel for each visual word.

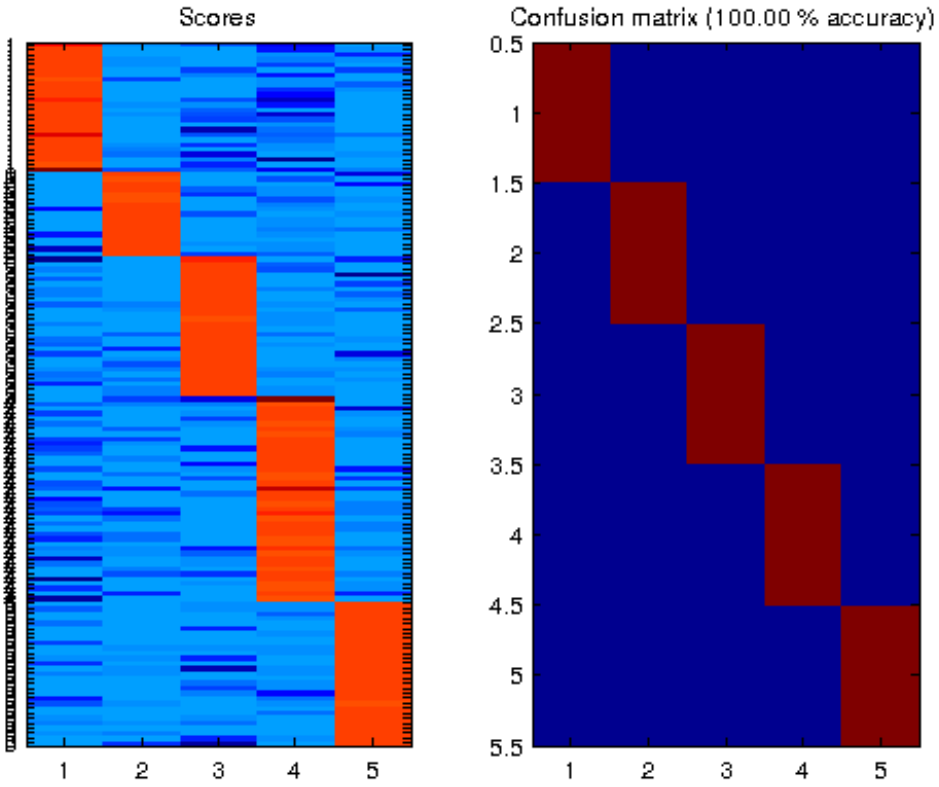
4. **40 points** Now you can train a linear SVM classifier and run the test experiment.

   You can download the liblinear software package for the classifier: http://www.csie.ntu.edu.tw/~cjlin/liblinear/

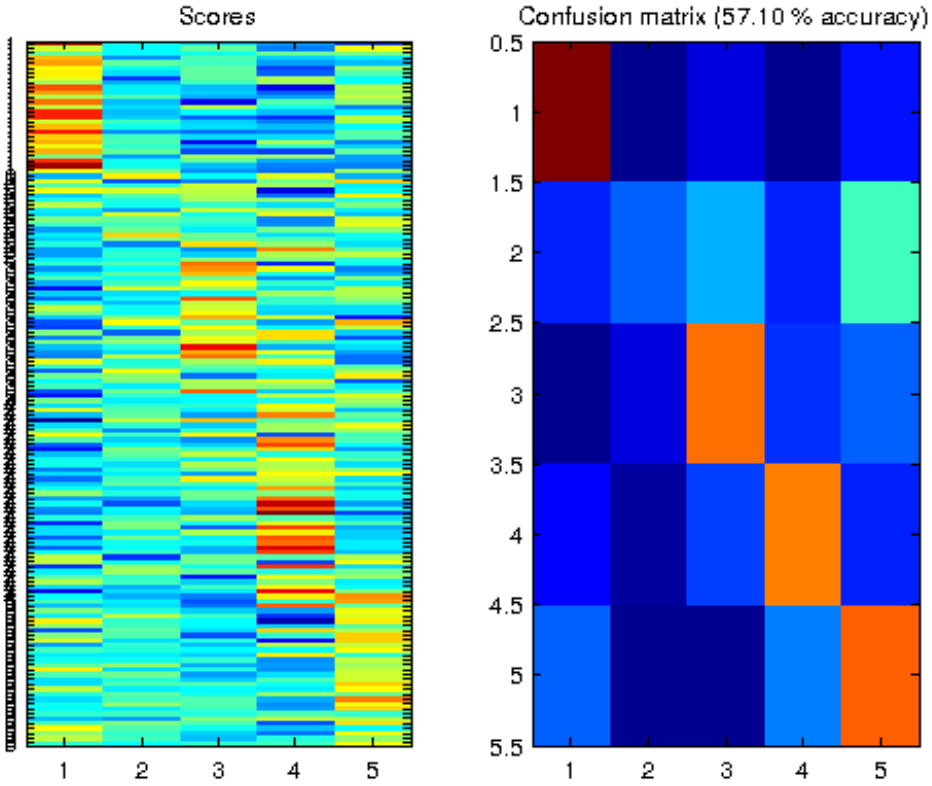
   You should try to find the best parameter to use in liblinear. You will be using liblinear's MATLAB interface. Use 'train' to train your classifier. After that, you should be able to calculate the response of the classifier by taking the inner product between the trained weight vector and the testing feature. See 'liblinear-1.96/matlab/README' file for instruction in installation and usage.

You are provided with a visualization function ‘showConfus’ to show the confusion matrix of the classification results, which might look like this (in a 5 class example):

Training validation



Testing



You can test on a subset of Caltech 101. Try different configurations of parameters, and report the best result (e.g., number of words, choice of filter bank, encoding choice, SVM params…). Of course, don't expect to get 100% correct!

5. **Extra Credit (up to 10 points):** Excellent classification results have been achieved using features extracted by deep learning. Caffe provides deep networks that have already been trained, making their use relatively simple. For extra credit, use these features instead of SIFT descriptors, and compare their effectiveness. Caffe is described at: http://caffe.berkeleyvision.org/. Instructions on installing Caffe can be found at: http://caffe.berkeleyvision.org/installation.html. Some information about its pretrained models are at: http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/classification.ipynb. Unfortunately, installing caffe is not trivial, as it has many dependencies. For this reason, you are free to work together or seek help from friends just in getting it installed (mention all sources of help). We would have made this problem required, but the number of possible platforms and versions of software make it impractical for us to help you install Caffe.

6. **Extra Credit (up to 10 points):** So far we are totally discarding the spatial information of descriptors. This information has proven to be useful in image classification. Spatial Pyramid Matching (SPM) is a well-known method to overcome the loss of spatial information. Take a look at the paper: http://www-cvr.ai.uiuc.edu/ponce_grp/publication/paper/cvpr06b.pdf

Try to implement SPM (it is actually very simple, once you understand it) and report the performance difference.

7. **Extra Credit (up to 10 points):** Andrea Vedaldi et al. described a method to calculate explicit feature map using Chi square kernel for histograms: http://www.robots.ox.ac.uk/~vedaldi/assets/pubs/vedaldi11efficient.pdf

   They show that bag of words classification in Caltech 101 can benefit from using this kernel mapping. You can try and report the performance difference.