Computer Vision
CSCI-UA.0480-002
Assignment 3.

April 4, 2013

Introduction

This assignment looks at face and object recognition. Unfortunately, there are not too many textbooks for this area but Szeliski's book (link on course webpage) may be of some use.

The assignment contains two questions:

1. Face Recognition – using the Eigenfaces approach described in class. [*50 points*].
2. Scene classification – using the Bag of Words representation with a Nearest Neighbor classifier. [*50 points*].

Requirements

You should perform this assignment in Matlab. If you are not familiar with Matlab, I suggest you go through some of the tutorials posted on the course web page.

This assignment is due on **Tuesday April 23rd** by the start class (11am). The late policy is posted on the course webpage, so please start early and don't be afraid to ask for help.

The TA for the class is Pravish Sood (`pravish.sood@nyu.edu`). Please email him for help and assistance, or come to office hours (Thursday 12.30-1.30pm). If you think the assignment is not clear, or there is an error/bug in it, please contact the TA or myself.

You are allowed to collaborate with other students in terms discussing ideas and possible solutions. However you code up the solution yourself, i.e. you must write your own code. Copying your friends code and just changing all the names of the variables is not allowed! You are not allowed to use solutions from similar assignments in courses from other institutions, or those found elsewhere on the web.

Your solutions should be emailed to me (`fergus@cs.nyu.edu`) and the TA (`pravish.sood@nyu.edu`) as a single zip file, with the filename: `lastname_firstname_a3.zip`. This zip file should contain: (i) a PDF file `lastname_firstname_a3.pdf` with your report, showing output images for each part of the assignment and explanatory text, where appropriate; (ii) the source code used to generate the images (with code comments), along with a master script (named `master_a3.m`) that runs the code for each part of the assignment in turn.

1 Face Recognition

In this assignment the goal is to implement a nearest-neighbor recognition algorithm, using the Eigenface representation, presented in Lecture 15. It is also covered on page 589 of Szeliski's book.

Download the `faces.zip` file from the course webpage. This contains two Matlab files: `ORL_32x32.mat` and `train_test_orl.mat`. The former contains 400 faces from the Olivetti Face database, each being 32 by 32 pixels grayscale (in variable `fea`) and the label of each image (in variable `gnd`). The latter contains a set of indices to be used for training and testing.

Open Matlab and load the two files into memory. You can view the files with the following command: `figure; montage(reshape(fea'/255,[32 32 1 400]));`. To implement the algorithm, your code should be structured as follows:

1. Split `fea` and `gnd` into training and test sets using the indices in `trainIdx` and `testIdx`. Scale the images so that the intensities range from 0 to 1.

2. Center the training data, so that the per-pixel mean of across all images is zero.

3. Form C, the 1024 by 1024 covariance matrix.

4. Compute the first K principal components v of C using the `eigs` function, e.g. `[v,d]=eigs(C,K);`.

5. Plot out these principal components. They should look like the Eigenfaces in the slides. The `reshape` command may be useful.

6. Now project the centered training data into the PCA space using the principal components, yielding descriptors p.

7. To get a sense of what the model has captured, form the reconstruction of the face by projecting back into the image space using p and v. Do not forget to add the mean face back on again.

8. Now center and project the test data into the PCA space, giving descriptors q.

9. Perform a nearest-neighbor search for each of the descriptors in q to find the closest Euclidean descriptor in p. Assign the test image belonging to the query descriptor the label from this closest training image.

10. Measure the fraction of test images correctly classified.

11. Now repeat the whole scheme, varying the value of K. Plot a graph of the classification rate as a function of K.

You should turn in: your Matlab code, plot of classification rate vs K, a plot of the reconstructed training faces for $K = 20$.

For bonus points, extract some 32 by 32 patches from non-face images and try projecting them into face space with different values of K. Compare the reconstruction error to that of face images. Is there some threshold you can set that enables you to reliably tell if an image is a face or not?

2 Scene Classification

This part of the assignment gets you to implement part of a Bag-of-Visual-Words classifier, applied to 4 different scene classes (1="coast", 2="highway", 3="street" and 4="city").

Download the file: `qu2_data.zip` from the course webpage. This is a large file (∼800Mb), so may take some time. This file contains 800 images

of natural scenes (200 for each of the 4 classes). It also contains the pre-computed SIFT descriptors for each image, extracted on a dense grid (which is why the files are large). Finally, there are two skeleton code templates `example_vq_sift.m` and `example_classify_nn.m`.

Within the zip is the file `qu2_data.mat` which contains:

- The disjoint training and test image indices (`TRAIN_IND` and `TEST_IND`)
- Class labels ($1 \ldots 4$) for the training and test examples (`TRAIN_LABELS` and `TEST_LABELS`).
- A pre-computed dictionary for 200 visual words in the variable `dictionary`

The overall BoW pipeline consists of several stages: (i) extract dense SIFT; (ii) form visual word dictionary from training images; (iii) vector-quantize (VQ) SIFT descriptors to make BoW histograms for each image; (iv) train classifier on training set histograms and (v) test classifier on test set histograms.

The first two stages of this pipeline have already been done for you, and you should implement the remaining three stages.

2.1 Vector Quantization

You should write code that takes in the pre-computed SIFT file for each image and computes a Bag-of-Words histogram. To do this, each SIFT descriptor is assigned to the closest dictionary element and the corresponding histogram bin incremented by 1. The skeleton code `example_vq_sift.m` lays out this process in detail. You should fill in the blank sections of this file. The resulting output variable `HISTOGRAMS` should be (Number of images) $\times$ (Dictionary size). Your report should contain an image of this matrix, display using `figure; imagesc(HISTOGRAMS)`.

2.2 Bag-of-Words Classifier

To keep things simple, we will use a Nearest-Neighbor classifier that doesn't require any training as such, thus you only need to implement stage (v) of the pipeline above.

Working from the skeleton code in `example_classify_nn.m`, you should take each test image histogram (computed above) and classify it by finding

its ($K = 5$) nearest-neighbors and using their labels to vote on the predicted label.

Your function should output the fraction of the test images that whose label was predicted correctly.

For bonus marks, try to improve the performance of the classifier by using some of the different distance measures that were covered in the lectures.