Image and Vision Computing: Lab 3

In this lab we will classify photo images into different scene types (Kitchen, Bedroom, Office, etc). It includes two parts that will talk through using two different representations (raw images, and SIFT Bag of Words) and two different classifiers (KNN and SVM) to classify. Specifically, we will work on classifying the 15 categories of images in the Lazebnik et al 2006 CVPR paper, and work towards building one of the baseline methods evaluated in that paper (SIFT Bag of Words with SVM).

Acknowledgements: The idea for this task is borrowed from the computer vision homework assignment set by Prof. James Hayes at Brown. This is a simplified version of James' original assignment.

Setup

For the SIFT BoW and SVM tools used in the second part of the lab we will use the vl_feat addon library (www.vlfeat.org). For your convenience this is included pre-compiled in the lab package, but if you want to do this work on your own PC, you may need to download, compile and install vlfeat separately. For the impatient you can skip this step at first while working on Part 1, it only needs to be done before doing Part 2. To setup the prepackaged version of vl_feat, go to the root folder of the lab and execute:

```
run('vlfeat/toolbox/vl_setup')
```

This will add the vl_feat library to your current matlab path.

Part 1: Tiny images and NN classifier.

The first task is to implement a simple version of the nearest neighbor classifier for image classification. Load up `LAB3_task1.m`. In this file you already have the code to find all the image files on disk (`get_image_paths`), read them from disk and store them as small 16x16 thumbnails (`tiny_images`), and summarise the outputs of the classification (`create_results_webpage`). Take a look at these files and make sure you understand how they work.

Here you already have the representation and your main task is to implement a nearest neighbor classifier in `nearest_neighbour`. This routine should accept the NxD array of train images `train_image`, the Nx1 cell array of train image labels (`train_labels`) corresponding to the input images, and the MxD array of test images (`test_image`). It should output `predicted_labels`, the Mx1 cell array of test image labels estimated by your NN classifier. Recall that a NN classifier estimates the label of an input test image by: 1. Comparing the testing image to all training images, 2. Finding the most similar (closest) training image, 3. Predicting the label of that closest training image.

Once you have generated your output predictions, the create results routine will compare the predictions to the true image categories and generate the output summary that can be seen by viewing `results_webpage/index.html`. The

output summary shows the overall accuracy, a confusion matrix of which clasess were mixed up, and some examples of the classifications made.

Interpreting the results: A basic implementation of a 1-NN classifier here should get about 18% accuracy. This is not great, but significantly higher than the chance level of 1/15=6.6%.

Hint: 1. Check out the matlab routine pdist2, it will do most of the work for you. Other functions that may be useful are min(), sort(), unique(), and strcmp(). 2. Start by implementing a 1-NN classifier for simplicity, you can upgrade it to K-NN later.

Bonus: The basic 1-NN classifier can be improved in various ways to get higher than 18% performance. Try: (i) Using more of the training data than the fraction used in the example code, (ii) Varying the image thumbnail size used, (iii) Standardizing the scale of the data before classification (look at function zscore), (iv) Exploring what other distance metrics pdist2 can use.

Part 2: SIFT BOW and SVM

The second task is to implement a simple version of the SIFT Bag of Words SVM classifier. Load up `LAB3_task2.m`. In this file you already have the code to find all the image files on disk (`get_image_paths`), and classify the encoded images with SVM (`svm`). What's missing here is the methods to train the codebook (`vocab`) and to use the learned codebook to turn each image into a bag of words representation (`bag_of_words`).

Building the Codebook

In `codebook.m`, you should loop over all the image paths, load up each image, and call a sift routine from vl_feat such as `vl_sift` (interest point sift) or `vl_dsift` (dense sift). The second return value of these functions contains the sift descriptors at each keypoint. Stack up the descriptors for all images and then pass them to a K-means function such as matlab's built in `kmeans` or `vl_kmeans` (faster!) to build the dictionary. It can then return the constructed dictionary. You can use the parameter num_words to tune the size of the codebook lerned.
Hint: For fast debugging, you can use the attribute 'fast' on vl_dsift, use a small number of words, and a smaller fraction of the training images. Once you got everything running, you can use more images and a larger codebook and it will take a few minutes to run.

Use the Codebook to Represent Images as Bag of Words

In bag_of_words you should read all the images again, and use the learned dictionary to encode them. 1. Extract features of each image (eg `vl_dsift` represents each interest point as a 128D SIFT vector). 2. Project them to the codebook (so each interest point is now represented as a scalar indicating the cluster it is closest to. Try `vl_alldist2`, or `pdist2` and `min`). 3. And then

encode each image by a histogram (try matlab's `hist`). So `bag_of_words` should return a Nxnum_words sized matrix representing the BoW histogram for each of the N images.

SVM Classification

Example SVM code is already provided which you can inspect and improve if you wish. The example code uses `vl_svmtrain()` to train and test the SVM model. Unlike the matlab built in SVM library, this function can be use as the training function and testing function. In testing process, it generates the labels for the test data. The example of testing with vl_svmtrain() is below.

```
[~,~,~, scores] = vl_svmtrain(test_image_feats', label_emtpy, 0, 'model',w, 'bias', b, 'solver', 'none');
```

As the basic SVM is a linear binary classifier, but we need to classify 15 classes, we train 15 1-vs-all SVMs. The SVM regularization parameter LAMBDA is important, you can try to select one manually or apply cross validation to select one.

Interpreting the results: A basic implementation of a SIFT-BOW + SVM classifier here should get about 26% accuracy, higher than the 18% from 1-NN.

Bonus: If you reached this far you have implemented a simple SIFT-BOW pipeline. Correctly tuning all parts of this pipeline (well tuned SIFT extraction, well tuned codebook, well cross-validated SVM) should be able to push the accuracy to around 70%.

Bonus: Alternatively, you can try plugging in a deep feature extractor and see how it performs.