CS 231A Computer Vision (Winter 2015)
Problem Set 4

Due: March 5th, 2015 (11:59pm)

1 Implicit Shape Model: Application of Generalized Hough Transform (20 points)

For this problem, you will be going through the training and testing of an implicit shape model to both classify and detect objects of two classes, cars and trucks. The pictures and codewords in this example are very abstract, but are intended to build the intuition behind implicit shape models and the use of the generalized hough transform to localize object centers.

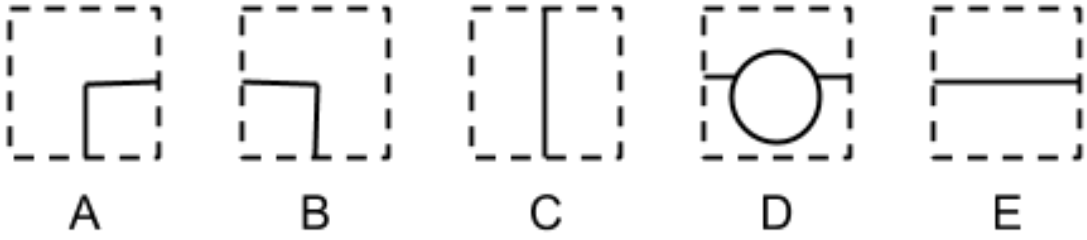

Figure 1: Code Words

(a) Using the codewords shown in Figure 1, match the patches shown in each grid cell for both the "Car" training image shown in Figure 2 (a) and the "Truck" image shown in (b). The object centers are shown as a red dot, and the grid spacing is 2 units. The two possible displacement vectors for each code word, R_1=[x1,y1] and R_2 = [x2,y2], should be measured from the middle of each matching grid cell to the the red dot, positive directions are to the right and up. Note that some codewords may have less than two displacement vectors, since a codeword can appear less than twice in the training images, so R_2 may be unused. If a codeword does not appear, leave both possible displacement vectors blank. No codeword appears more than twice. Patches that do not match the 5 codewords should be ignored. Populate the codebook shown in Table 1 and submit it.

(b) Using the two learnt models of car and track (i.e. the codebook tables that you generated in Part a), identify which object is present in the query image (Figures 3 and 4) and identify its location. You can do that using the generalized Hough Transform by voting for the object center for both the "Car" and "Truck" object classes. You can ignore

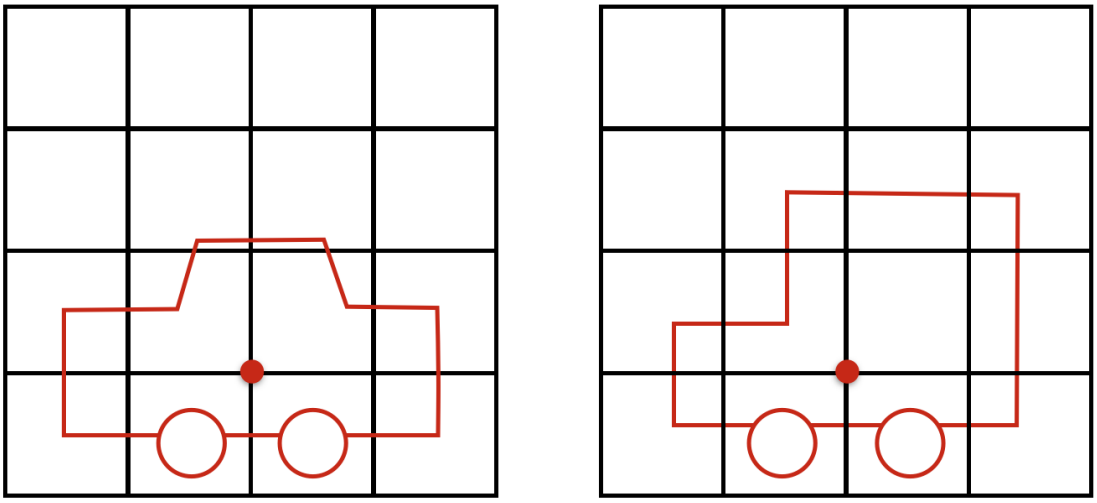

Figure 2: Training images for "Car" and "Truck" classes.

	Car		Truck	
	R_1	R_2	R_1	R_2
A				
B				
C				
D				
E				

Table 1: Codebook for "Car" and "Truck" objects.

patches that do not match any codeword. The two query figures are identical; use them to plot the hough votes for each object class. Please state which class is most likely based on the consensus of object center votes. Please turn in plots of the votes for the object centers for both classes. In case the votes overlap, indicate how many votes each point represents.

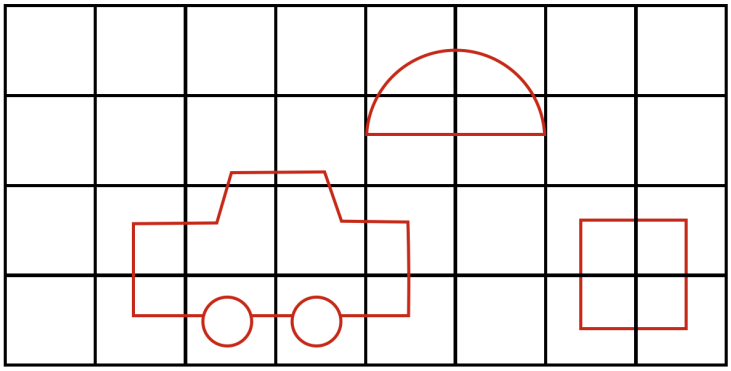

Figure 3: Query Image 1: "Car"

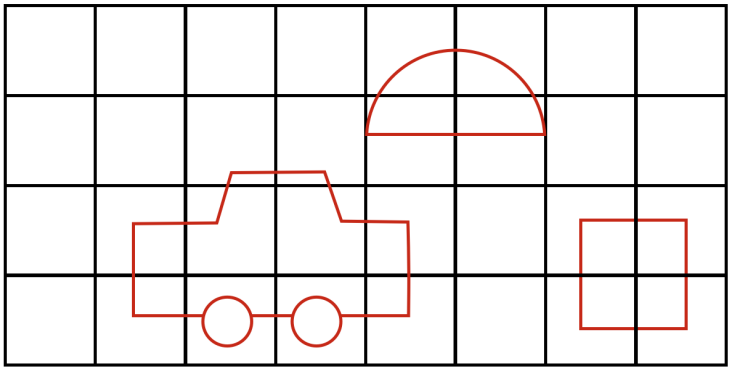

Figure 4: Query Image 2: "Truck"

2 Image classification (80 points)

The goal of this programming project is to implement a simple, effective method for classifying images using the Bag-of-Words (BoW) model. This is going to be an end-to-end system that, given multiple classes of training images, trains a model and classify test images.

The image classification will be performed on a subset of the Caltech-101 dataset, which will be downloaded automatically the first time you run the skeleton code (you will need to connected to the internet for this). The main Matlab function files are **p2.m**.

In this project we will use the open source VLFeat library, which implements popular computer vision algorithms. The installation guide for VLFeat in MATLAB can be found here: `http://www.vlfeat.org/install-matlab.html`. Follow the "one-time setup" section in the guide to make sure it's correctly installed. After that, you need to set VLFEAT_PATH in **p2.m** to be the VLFeat root path.

In the BoW model, there are four main components:

- **Feature extraction**. We need a way to describe image features with high-dimensional descriptors. For this assignment, you will use the dense SIFT feature descriptor. The dense SIFT feature descriptor extracts SIFT features from a dense sampling of keypoints in a given image. It typically uses a grid structure and computes the features at all grid point locations. For more information, refer to `http://www.vlfeat.org/api/dsift.html`. In

this project, you do not need to implement the descriptor yourself.

- **Visual word dictionary**. This is also called a "codebook" in the BoW model. We need to convert features to "codewords", a representation of several similar features. A simple way to generate a dictionary is *k-means clustering* over all extracted features, where the centroid of clusters are used as codewords. We then approximate features with the centroid of the cluster it belongs to (the codeword it is closest to). We will use this k-means clustering approach in this project.

- **Image representation**. Given the visual dictionary, each image is represented by a histogram of visual words that features in the image are assigned to. In this way, an image is represented as a distribution of features rather than the extracted features themselves. The distribution is later used to train a multi-class 1-vs-all SVM classifier.

  In addition, we add one more wrinkle. To encode the local spatial distribution of features, we introduce the idea of Spatial Pyramid Matching. In Spatial Pyramid Matching, we partition the image as shown in Fig 5. For each sub-block, we build a spatial histogram. At Level k, the image is partitioned into 2^{2k} sub-blocks. So, level 0 represents the full image, level 1 partitions the image into 4 sub-blocks and so on. Once we extract spatial histograms for all sub-blocks in all levels, we concatenate the histograms and represent the image as a single concatenated histogram.

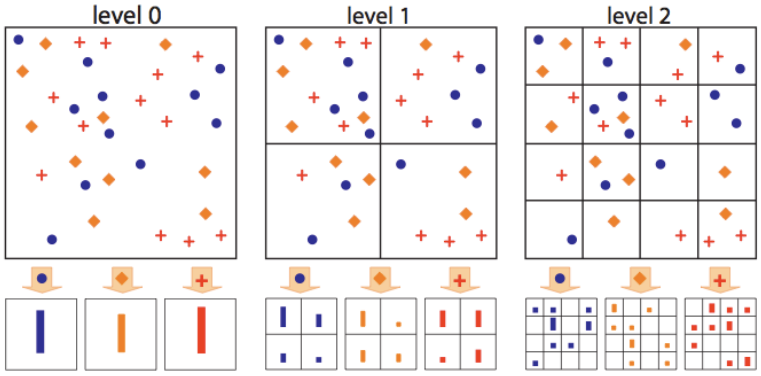

Figure 5: Overview of the Spatial Pyramid Kernel in which histograms of interest points (denoted by the three markers) are created for each subdivision in a 2-level pyramid. In level-0, we create a histogram for all of the features in the full image. In level-1, we create a histogram for each of the 4 sub-blocks. All of these histograms are then concatenated to form the representation of this image.

- **Machine learning**. Since images are now represented by visual histograms from the BoW model, we can use a discriminative model to train and classify the images. Support Vector Machines (SVM) are one of the most popular choices for this task and will be used in this project. This part is already provided, and you don't have to write code on it.

We provide the skeleton code in **p2.m**. Please fill in the empty functions to complete the end-to-end image classification system. These are listed in a section below.

i There are several parameters in **p2.m** you have to set before writing the code. See the "User configuration" section in **p2.m**.

- VLFEAT_PATH. This should be set to the root path of your installed VLFeat.
- N_CLASSES. This controls how many image classes from the dataset are used. This value is directly related to the time of execution, so you should start by setting it to a small number (e.g. 2) for ease of debugging. When you are confident that your implementation is correct, set this to 10 for a more general result.
- N_WORDS_CHOICES. This array gives the number of codewords to use in the dictionary. For debugging, we recommend setting it to a single value, meaning that only N_WORDS = 400 will be used. Multiple values in an array will cause the system to run multiple times with different values of N_WORDS. You may need to modify this to give analysis on different dictionary sizes.
- MAX_LEVEL_CHOICES. This array defines levels of the spatial pyramids to be used. For instance, if the array is $[0, 1]$, then image classification will be run twice, once with level 0 and the second time with levels 0 and 1. You may need to modify this to report analysis on different number of levels.

ii There are several functions for you to implement. We provide a **read_image** function to read in an image. Please use this function to load images, as it takes care of normalizing the image. Specifically, you need to implement certain portions of the following files. Open up these files and follow their instructions to complete. Submit the code for grading.

- **extract_dense_sift.m**
- **create_dictionary.m**
- **spatial_pyramid_single.m**
- **create_histograms.m**
- **create_confusion_matrix.m**

Once you have completed these functions, run **p2.m** to train and evaluate the BoW model. Report the training and testing error as well as the confusion matrix.

For your reference, a correct implementation with N_CLASSES = 2, N_WORDS = 400, MAX_LEVEL = 0, should have an accuracy around 80%. Using N_CLASSES = 10, N_WORDS = 700, MAX_LEVEL = 2, the accuracy for a correct implementation is also around 80%.

iii Report the accuracy of the test set using 2 classes, 200 words, and only level 0. Report the accuracy of the test set using 10 classes and the same number of words and levels as before. Give a sentence or two on why these results makes sense.

iv Using 3 classes and only level 0, plot the accuracy with respect to using 5, 300, 600 and 900 codewords. Report the number of classes that gives the highest accuracy of the test set. Give a sentence or two on why these results makes sense.

v Using 3 classes, plot the accuracy with respect to using 0, 1, and 2 levels of spatial pyramid. For clarification, using just level 0 is 1 level. Using levels 0 and 1 is 2 levels. Report the number of levels that gives the highest accuracy of the test set. Give a sentence or two on why these results makes sense. Use 600 codewords for the all the experiments.