EN.600.461/661 – Computer Vision
Fall 2016
Homework #3
Due: 11:59 PM, Thu, November 17, 2016

All solutions (e.g., code, README write-ups, output images) should be zipped up as **HW3_yourJHED.zip** and posted on Blackboard, where 'yourJHED' is your JHED ID (ie.tkim60). Only basic MATLAB/Python functions are allowed, unless otherwise specified. If you are unsure of an allowable function, please ask on Piazza before assuming! You will get no credit for using a "magic" function to answer any questions where you should be answering them. When in doubt, ASK!

Programming Assignment

1) *Scene Recognition with Bag of Words*

a. We wish to build a system that can recognize 6 scene categories: *buildings, food, people, faces, cars, and trees*. For each label, we have prepared for you a collection of 201 pictures of each category. The root dataset folder contains two folders: *train/* and *test/*. Under the *train*/ directory, you will find a directory for each of the scene categories. You are to use the images under the *train/* directory to train your recognition system and likewise for the images in *test/* .

Here is the dataset:
https://www.dropbox.com/s/g7zelnuomw5cc0a/CV_Assignment3.zip?dl=0

b. *Training*: For each scene class (6 of them), use all the images in the train split and the remaining for testing (so you will have 6 classes *150 images/class for training and 6 classes * 51 images/class for testing). To train, do as follows:

i. For each image, compute SIFT (ORB features for python-ers for technical reasons) features using the supplied source code

Python:

```
import cv2
import matplotlib.pyplot as plt
cv2.ocl.setUseOpenCL(False)
img = cv2.imread('PATH TO YOUR FILE')
orb = cv2.ORB_create()
keypoints = orb.detect(img,None)
keypoints, descriptors = orb.compute(img, keypoints)
# where 'descriptors' is a n-by-32 descriptors for n
feature points in the 'img'
```

```
whyNoSIFTinOpenCV3= cv2.drawKeypoints(img, keypoints, 0,
color=(0,255,0))
plt.imshow( cv2.cvtColor(whyNoSIFTinOpenCV3,
cv2.COLOR_BGR2RGB))
plt.show()
```

Matlab:

```
We will be using a very useful library called VLFeat which
allows us to use SIFT features. Follow the instructions
below. (Instructions are straightforward and I tried
setting this up as well without much pain)

http://www.vlfeat.org/install-matlab.html
```

ii. Collect SIFT/ORB features for all training images into one large vector (i.e., all features from all images and all scene classes together into one vector). Run K-Means with 800 cluster centers on the SIFT/ORB vectors and store the resulting centers (i.e., your "codebook"). ***Is 800 a good choice for size of the codebook? Would it be better to use more, less, or neither? Please explain.***

Python:

```
# Assume 'features' is a n-by-128  numpy.ndarray where # n
is the number of all features to cluster, 128 is the
dimension of each feature.

# where label contains labels for all features and
# center has all 800 cluster centers.

features = np.float32(features)
criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
ret,label,center=cv2.kmeans(features,800,None,criteria,10,
cv2.KMEANS_RANDOM_CENTERS)
```

Matlab:

```
% Assume 'features' is a n-by-d double matrix where
% the specs are the same as above.

[labels,centers] = kmeans(features,800);
```

iii. For each training image, use the SIFT/ORB features from each and the codebook that you just created, and compute a Bag-of-Words encoding vector for each and store them in memory with the scene class associated with each. You may choose how to store these so you can search them easily (i.e., a dictionary, 2 lists, one of features and one of the associated scene class as a string for each, etc.)

iv. For each image in your test set: compute SIFT/ORB features, use your codebook from training, compute a Bag-of-Words encoding, and use a nearest-neighbor search to your training features to label the best scene class. Report your error and accuracy results.

c. (***For 661 Graduate Students only***) Repeat the above experiment, however instead of nearest-neighbor classification we will use a Support Vector Machine (SVM) with a linear kernel. This question is designed to get you familiar with common tools to perform classification and NOT to get you to rewrite the guts of an SVM algorithm! These are useful tools in the research community. ***Please compare and contrast these results to your nearest-neighbor approach. Was it better, worse, or the same? Why or why not?***

For Python, SVMs (along with many other useful machine learning algorithms) are built in to scikit. You can easily install this with "pip install scikit-learn". For MATLAB this comes "for free" (including the Academic Version) with the Statistics and Machine Learning Toolbox.

Python:

```
from sklearn.svm import LinearSVC
classifier = LinearSVC(random_state=0, C=1.0, loss='hinge',
penalty='l2')
classifier.fit(features, labels)

# 'features' is np.ndarray where each row is a feature
vector. The corresponding index in 'labels' is the class
label of the feature vector.

# In test time, use classifier.predict(X), where X is an
np.ndarray where each row is a test feature.
```

Matlab:

Since most versions of Matlab do not provide multi-class support for SVM, we have included a Matlab script that implements one-vs-the-rest classification using binary SVM classfiers.

[result] = multisvm(trainData, trainLabels, testData)

 # trainData is an nxd matrix, where each row corresponds to an observation feature vector, trainLabels is the nx1 vector of training labels and testData is the test data matrix in the same format as trainData.

2) Stereo Matching and Reconstruction

a. Look at the enclosed stereo rectified images ("*scene_l.bmp"* and "*scene_r.bmp*"), and assume the epilines are horizontal lines such that the y-coordinates of a point in one image matches a horizontal line at that same y-coordinate in the other image. Compute a depth image by stereo matching using normalized cross-correlation with a fixed window size of 15x15 pixels.

b. Using your depth image, create a 3D point cloud and store as a text file (one 3D point per row). ***Did you get any areas in your depth map? Why? Please explain.***

Baseline: 100 mm
Focal length: 400 pixels

3) Camera Calibration

Let P be the projection matrix for camera C, as given below:

$$P = KR[I_{3\times3}| - C],$$

$$\text{with } K = \begin{bmatrix} 1072 & 0 & 500 \\ 0 & 1072 & 390 \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.7071 & -0.7071 \\ 0 & 0.7071 & 0.7071 \end{bmatrix} \quad C = \begin{bmatrix} 25 \\ 50 \\ 0 \end{bmatrix}.$$

a. Supposing square pixels, what are the focal length (in pixel units) and image center of the camera?

b. What are the orientation and position of the camera with respect to the world coordinate system (explain briefly)?

c. Supposing the camera C is now rotated by 10 degrees around the world X camera axis from its initial position, provide the new matrix R (explain briefly).

d. Supposing the camera C is now further translated by 10 units along the world positive X axis, provide the new vector C (explain briefly).

4) Fundamental Matrix and Epipolar Lines (For 661 Graduate Students only)

The purpose of this exercise is to estimate and display epipolar lines in the two images hopkins1.jpg (I1) and hopkins2.jpg (I2). Implement this in: ***`fundamental.m`*** **or** ***`fundamental.py`***

a. Find the SIFT/ORB features in both images and match the features (similar to what we did in assignment 2). Display the correspondences.

b. Find the fundamental matrix using the previously matched features. Use the method described in the lecture notes. *In order to get credit for this question, you may not simply call a vision library to compute the matrix for you. You must estimate it yourself!*

c. Using the fundamental matrix computed above, draw the epipolar lines for a few selected features of your choice. (Randomly pick eight features in the left image and draw the eight corresponding epipolar lines on the right image).