UNIVERSITY OF TEXAS AT ARLINGTON
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

6367 **COMPUTER VISION** **SPRING 2020**

ASSIGNMENT 6 (100 POINTS)
ASSIGNED: 4/21/2020 DUE: 5/7/2020

This assignment constitutes 10% of the course grade. You must work on it individually and are required to submit a PDF report along with the MATLAB scripts described below. Your programs must not use computer vision functions provided by MATLAB unless otherwise specified.

1 Overview: Scene Recognition

Figure 1: Example images from the scene category dataset. The starred categories originate from Oliva and Torralba [2].

In this assignment, the goal is to build a set of visual recognition systems that classify scene categories [1]. The provided scene classification dataset consists of 15 categories, Figure 1. The system will compute a set of image representations (tiny images and bag-of-words (BoW) visual vocabulary), and predict the category of each test image using a set of classifiers (k-nearest neighbors (KNN) and support vector machines (SVM)) built on the training data. Pseudocode of the scene recognition system is shown in Algorithm 1.

Algorithm 1 Scene Recognition

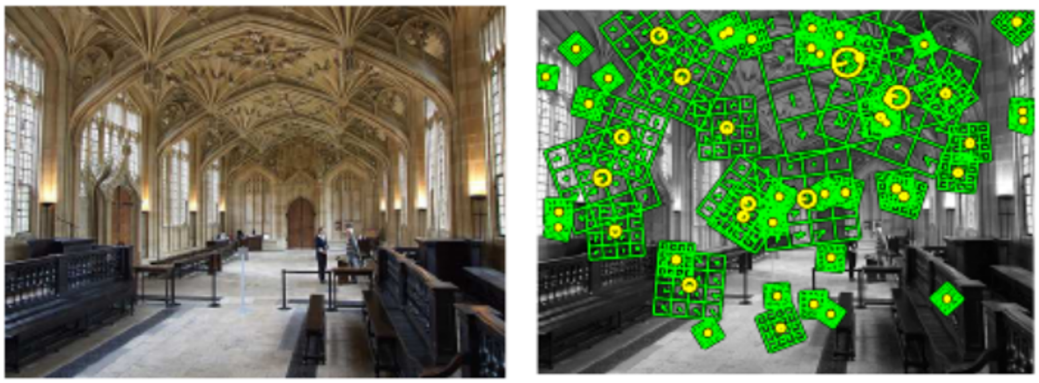
1: Load the training and testing images
2: Build image representations from the training data
3: Train a classifier using the representations of the training images
4: Classify the test data
5: Compute the accuracy of the test data classification

For the KNN classifier, steps 3 and 4 can be combined.

2 Scene Classification Dataset

The scene classification dataset, contained in 'scene_classification_dataset.zip', includes two text files ('train.txt' and 'test.txt') and two directories (`train` and `test`). Each row of the text file specifies the image and its label, i.e., (label) (image path). The text files may be used to load images. In each folder there are 15 classes (Kitchen, Store, Bedroom, LivingRoom, Office, Industrial, Suburb, InsideCity, TallBuilding, Street, Highway, OpenCountry, Coast, Mountain, Forest) of scene images.

3 VLFeat Usage



(a) (b)

Figure 2: An input image (a) and examples of detected SIFT features along with their descriptors (b).

A key skill to learn in computer vision is the ability to use open source code which allows you avoid reinventing the wheel. For this assignment you will use the VLFeat software by Vedaldi and Fulkerson [3] to extract SIFT features. Install VLFeat as follows:

- Setup VLFeat in MATLAB using these instructions: https://www.vlfeat.org/install-matlab.html
- Double check the installation by running `vl_demo_sift_basic`

Note that you will use this library only for SIFT feature extraction and its visualization. All other visualizations and algorithms must be implemented with your own code. Using VLFeat, you can extract keypoints and their associated descriptors as shown in Figure 2.

Practice using VLFeat to visualize SIFT features with scale and orientation. You may want to read the following tutorial: https://www.vlfeat.org/overview/sift.html. In this assignment, you are allowed use the following two functions: `vl_dsift` and `vl_svmtrain`.

4 Tiny Image Representation and K-Nearest Neighbors Classification

(a) (b)

Figure 3: An input image (a) and its tiny image representation (b).

Problem 1 (20 + 20 + 10 = 50 points)

In this problem, you will compute a set of tiny image representations and predict the category of each test image using a KNN classifier.

(a) Write a MATLAB function, `[feature] = get_tiny_image(I, output_size)`, that simply re-sizes an input image to a small, fixed resolution (e.g., 16 × 16). The first input, `I`, is a gray scale image and the second input, `output_size = [w,h]`, is the size of the resulting tiny image. The output, `feature`, is the tiny image representation created by vectorizing the pixel intensity in column major order. The resulting normalized image (zero mean and unit length) will be of size $w \times h$. Note that this is not a particularly good representation since it discards all of the high frequency image content and lacks invariance to spatial and brightness shifts.

(b) Write a MATLAB function, `[label_test_pred] = predict_knn(feature_train, label_train, feature_test, k)`, that uses a KNN classifier to predict the label of the testing data. The first input, `feature_train`, is a $n_{tr} \times d$ matrix where n_{tr} is the number of training data samples and d is the dimension of image features, e.g., 265 for a 16 × 16 tiny image representation. Each row is an image feature. The second input, `label_train` $\in [1, 15]$, is a n_{tr} vector that specifies the label of the training data. The third input, `feature_test`, is a $n_{te} \times d$ matrix that contains the testing features where n_{te} is the number of test data samples. The last input, k, is the number of neighbors for label prediction. The output, `label_test_pred`, is a n_{te} vector that specifies the predicted label for the testing data.

(c) Write a MATLAB function, `[confusion, accuracy] = classify_knn_tiny`, that combines `get_tiny_image` and `predict_knn` for scene classification. The first output, `confusion`, is a 15 × 15 confusion matrix. The second output, `accuracy`, is the accuracy of the test data prediction. Your goal is to achieve an accuracy $> 18\%$.

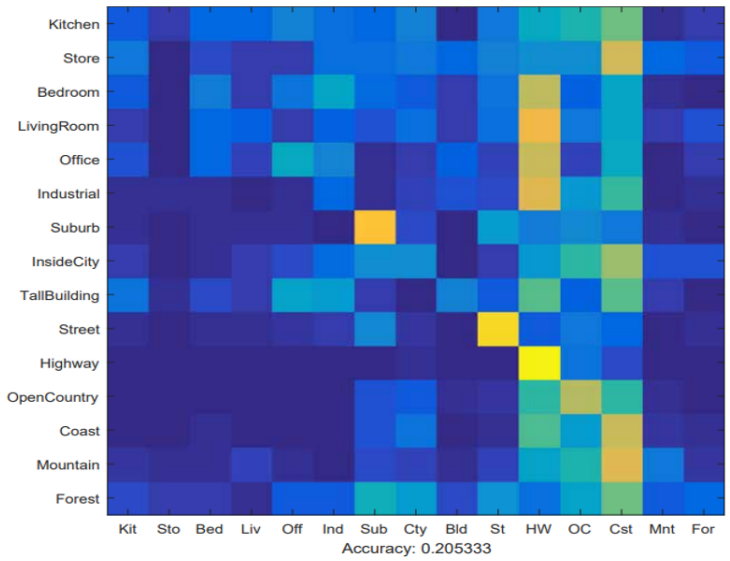


Figure 4: Confusion matrix for tiny image and KNN classification.

Submission Instructions: *For part (a) submit the MATLAB function* `classify_tiny_image`, *for part (b) submit the MATLAB function* `predict_knn`, *and for part (c) submit the MATLAB function* `classify_knn_tiny`. *Submit a MATLAB scrip 'problem_1.m' along with any other files necessary to run your functions. Include the confusion matrix and classification accuracy (Figure 4) by embedding them in the report, do not submit the figure separately.*

5 Bag-of-Words Visual Vocabulary and K-Nearest Neighbors Classification

Figure 5: Each row represents a distinctive cluster from a bag-of-words representation.

Problem 2 (20 + 20 + 10 = 50 points)

In this problem, you will compute a set of image representations using a BoW visual vocabulary and predict the category of each test image using a KNN classifier (Algorithm 2).

Algorithm 2 Visual Dictionary Building

1: For each image, compute dense SIFT features over a regular grid
2: Build a pool of SIFT features from all the training images
3: Find the cluster centers from the SIFT pool using k-means clustering
4: Return the cluster centers

(a) Write a MATLAB function, `[vocab] = build_visual_dictionary(training_image_cell, dict_size)`, that builds a visual dictionary of quantized SIFT features. The first input, `training_image_cell`, is a set of training images and the second input, `dict_size`, is the size of the dictionary (i.e., the number of visual words). You may start with `dict_size = 50` and you can use the following builtin functions:

- `vl_dsift` from VLFeat
- `kmeans` from the MATLAB Statistics and Machine Learning Toolbox

You may visualize the image patches to make sense of the clustering as shown in Figure 5.

(b) Write a MATLAB function, `[bow_feature] = compute_bow(feature, vocab)`, that computes the BoW feature vector. The first input `feature` is a set of SIFT features for one image, and the second input `vocab` is a visual dictionary. The output, `bow_feature`, is the BoW feature vector whose size is `dict_size`. The BoW feature vector is constructed by counting SIFT features that fall into each cluster of the vocabulary. Nearest neighbors can be used to find the closest cluster center. The histogram needs to be normalized such that the BoW feature vector has unit length.

(c) Write a MATLAB function, `[confusion, accuracy] = classify_knn_bow`, that combines `build_visual_dictionary`, `compute_bow`, and `predict_knn` for scene classification given BoW features. The first output, `confusion`, is a 15×15 confusion matrix. The second output, `accuracy`, is the accuracy of the test data prediction. Your goal is to achieve an accuracy $> 50\%$.

Submission Instructions: *For part (a) submit the MATLAB function* `build_visual_dictionary`, *for part (b) submit the MATLAB function* `compute_bow`, *and for part (c) submit the MATLAB function* `classify_knn_bow`. *Submit a MATLAB scrip 'problem_2.m' along with any other files necessary to run your functions. Include the confusion matrix and classification accuracy (Figure 6) by embedding them in the report, do not submit the figure separately.*

Extra Credit (10 + 10 = 20 points)

In this problem, you will compute a set of image representations using a BoW visual vocabulary and predict the category of each test image using a SVM classifier.

(a) Write a MATLAB function, `[label_test_pred] = predict_svm(feature_train, label_train, feature_test)`, that uses a SVM classifier to predict the label of the test data. The first input, `feature_train`, is a $n_{tr} \times d$ matrix where n_{tr} is the number of training data samples and d is the dimension of image feature. Each row is an image feature. The second input, `label_train` $\in [1, 15]$, is a n_{tr} vector that specifies the label of the training data. The third input, `feature_test`, is a $n_{te} \times d$ matrix that contains the test features

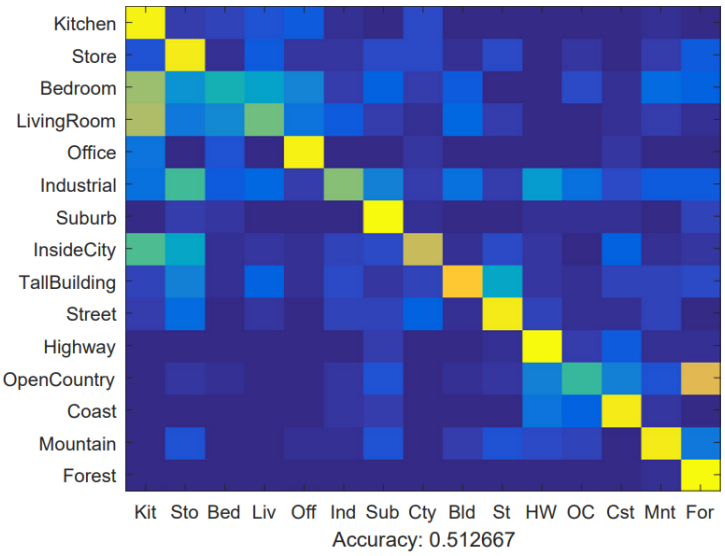


Figure 6: Confusion matrix for BoW and KNN classification.

where n_{te} is the number of test data samples. The output, `label_test_pred`, is a n_{te} vector that specifies the predicted label for the test data.

You do not have to implement the SVM classifier. Instead, you can use `vl_svmtrain` in VLFeat. Linear classifiers are inherently binary and we have a 15-way classification problem. To decide which of 15 categories a test case belongs to, you will train 15 binary 1-vs-all SVMs. 1-vs-all means that each classifier will be trained to recognize 'forest' vs. 'nonforest', 'kitchen' vs. 'non-kitchen', etc.

All 15 classifiers will be evaluated on each test case and the classifier which is most confidently positive "wins". For example, suppose that the 'kitchen' classifier returns a score of -0.2 (where 0 is on the decision boundary) and the forest classifier returns a score of -0.3. If all of the other classifiers are even more negative, then the test case would be classified as 'kitchen' even though none of the classifiers placed the test case on the positive side of the decision boundary. When learning a SVM, you have a free parameter 'lambda' which controls how strongly regularized the model is. Your accuracy will be very sensitive to the choice of lambda, therefore be sure to test many values.

(a) Write a MATLAB function, `[confusion, accuracy] = classify_svm_bow`, that combines `build_visual_dictionary`, `compute_bow`, and `predict_svm` for scene classification given BoW features. The first output, `confusion`, is a 15×15 confusion matrix. The second output, `accuracy`, is the accuracy of the test data prediction. Your goal is to achieve an accuracy $> 60\%$.

Submission Instructions: *For part (a) submit the MATLAB function* `predict_svm`, *and for part (b) submit the MATLAB function* `classify_svm_bow`. *Submit a MATLAB scrip 'extra_credit.m' along with any other files necessary to run your functions. Include the confusion matrix and classification accuracy (Figure 7) by embedding them in the report, do not submit the figure separately.*

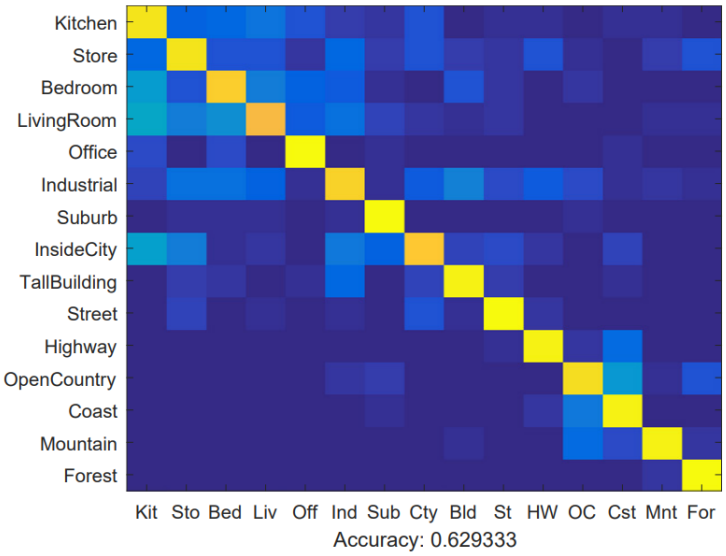


Figure 7: Confusion matrix for BoW and SVM classification.

References

[1] S. Lazebnik, C. Schmid and J. Ponce. "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories." IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), vol. 2, pp. 2169-2178. 2006.

[2] A. Oliva and A. Torralba. "Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope." International Journal of Computer Vision, vol. 42, no. 3, pp. 145-175. 2001.

[3] A. Vedaldi and B. Fulkerson. "VLFeat: An Open and Portable Library of Computer Vision Algorithms." 18th ACM International Conference on Multimedia, pp. 1469-1472. 2010