CSCI 5561: Assignment #3

Scene Recognition

1 Submission

- Assignment due: Mar 29 (11:55pm)
- Individual assignment
- 2 page summary write-up with resulting visualization (more than 1 page assignment will be automatically returned.).
- Submission through Canvas.
- List of submission codes:
  - `GetTinyImage.m`
  - `PredictKNN.m`
  - `ClassifyKNN_Tiny.m`
  - `BuildVisualDictionary.m`
  - `ComputeBoW.m`
  - `ClassifyKNN_BoW.m`
  - `PredictSVM.m`
  - `ClassifySVM_BoW.m`
- DO NOT SUBMIT THE PROVIDED IMAGE DATA
- The function that does not comply with its specification will not be graded.
- You are allowed to use MATLAB built-in functions except for the ones in the Computer Vision Toolbox. Please consult with TA if you are not sure about the list of allowed functions.

CSCI 5561: Assignment #3

Scene Recognition

2 Overview

Figure 1: You will design a visual recognition system to classify the scene categories.

The goal of this assignment is to build a set of visual recognition systems that classify the scene categories. The scene classification dataset consists of 15 scene categories including office, kitchen, and forest as shown in Figure 1 [1]. The system will compute a set of image representations (tiny image and bag-of-word visual vocabulary) and predict the category of each testing image using the classifiers (k-nearest neighbor and SVM) built on the training data. A simple pseudo-code of the recognition system can found below:

Algorithm 1 Scene Recognition

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

For the knn classifier, step 3 and 4 can be combined.

CSCI 5561: Assignment #3

Scene Recognition

3 Scene Classification Dataset

You can download the training and testing data from here:
`http://www.cs.umn.edu/~hspark/csci5561/scene_classification_data.zip`

The data folder includes two text files (`train.txt` and `test.txt`) and two folders (`train` and `test`). Each row in the text file specifies the image and its label, i.e., (label) (image_path). The text files can be used to load images. In each folder, it includes 15 classes (Kitchen, Store, Bedroom, LivingRoom, Office, Industrial, Suburb, InsideCity, TallBuilding, Street, Highway, OpenCountry, Coast, Mountain, Forest) of scene images.

4 VLFeat Usage

Similar to HW #2, you will use VLFeat (`http://www.vlfeat.org/install-matlab.html`). You are allowed to use the following two functions: `vl_dsift` and `vl_svmtrain`.

CSCI 5561: Assignment #3

Scene Recognition

5 Tiny Image KNN Classification

(a) Image (b) Tiny Image

Figure 2: You will use tiny image representation to get an image feature.

`function [feature] = GetTinyImage(I, output_size)`
Input: `I` is an gray scale image, `output_size=[w,h]` is the size of the tiny image.
Output: `feature` is the tiny image representation by vectorizing the pixel intensity in a column major order. The resulting size will be `w×h`.
Description: You will simply resize each image to a small, fixed resolution (e.g., 16×16). You need to normalize the image by having zero mean and unit length. This is not a particularly good representation, because it discards all of the high frequency image content and is not especially invariant to spatial or brightness shifts.

`function [label_test_pred] = PredictKNN(feature_train, label_train, feature_test, k)`
Input: `feature_train` is a $n_{\text{tr}} \times d$ matrix where n_{tr} is the number of training data samples and d is the dimension of image feature, e.g., 265 for 16×16 tiny image representation. Each row is the image feature. `label_train`$\in [1, 15]$ is a n_{tr} vector that specifies the label of the training data. `feature_test` is a $n_{\text{te}} \times d$ matrix that contains the testing features where n_{te} is the number of testing data samples. `k` is the number of neighbors for label prediction.
Output: `label_test_pred` is a n_{te} vector that specifies the predicted label for the testing data.
Description: You will use a k-nearest neighbor classifier to predict the label of the testing data.

CSCI 5561: Assignment #3

Scene Recognition

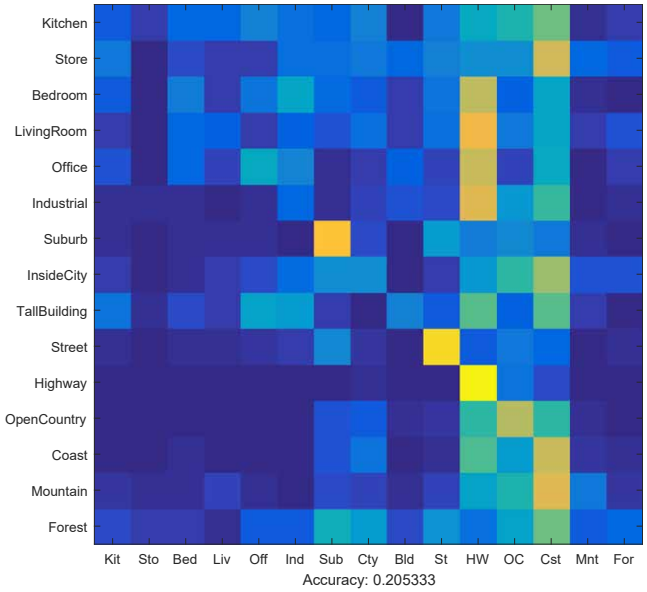

Figure 3: Confusion matrix for Tiny+KNN.

`function [confusion, accuracy] = ClassifyKNN_Tiny`
Output: `confusion` is a 15×15 confusion matrix and *accuracy* is the accuracy of the testing data prediction.
Description: You will combine `GetTinyImage` and `PredictKNN` for scene classification. Your goal is to achieve the accuracy >18%.

CSCI 5561: Assignment #3

Scene Recognition

6 Bag-of-word Visual Vocabulary

Figure 4: Each row represents a distinctive cluster from bag-of-word representation.

`function [vocab] = BuildVisualDictionary(training_image_cell, dic_size)`
Input: `training_image_cell` is a set of training images and `dic_size` is the size of the dictionary (the number of visual words).
Output: `vocab` lists the quantized visual words whose size is `dic_size`×128.
Description: Given a set of training images, you will build a visual dictionary made of quantized SIFT features. You may start `dic_size`=50. You can use the following built-in functions:

- `vl_dsift` from VLFeat.
- `kmeans` from MATLAB toolbox.

You may visualize the image patches to make sense the clustering as shown in Figure 4.

Algorithm 2 Visual Dictionary Building

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

CSCI 5561: Assignment #3

Scene Recognition

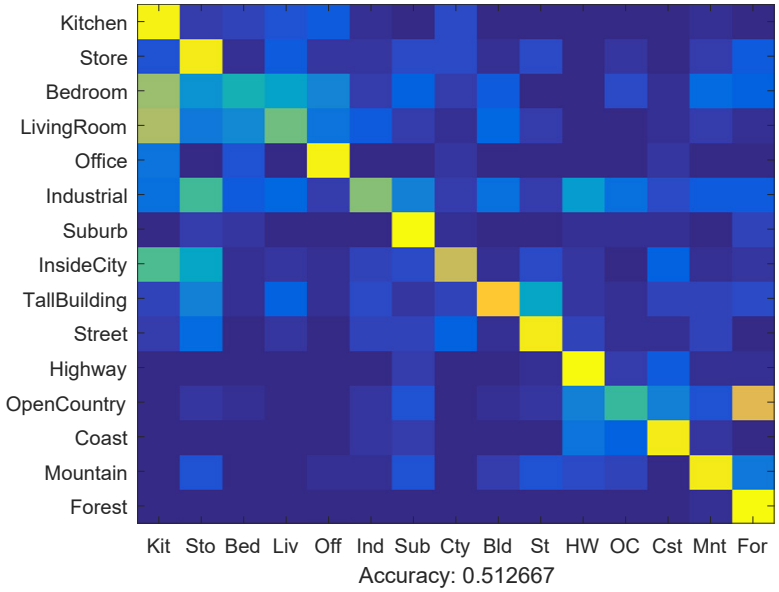

Figure 5: Confusion matrix for BoW+KNN.

`function [bow_feature] = ComputeBoW(feature, vocab)`
Input: `feature` is a set of SIFT features for one image, and `vocab` is visual dictionary.
Output: `bow_feature` is the bag-of-words feature vector whose size is `dic_size`.
Description: Give a set of SIFT features from an image, you will compute the bag-of-words feature. The BoW feature is constructed by counting SIFT features that fall into each cluster of the vocabulary. Nearest neighbor can be used to find the closest cluster center. The histogram needs to be normalized such that BoW feature has a unit length.

`function [confusion, accuracy] = ClassifyKNN_BoW`
Output: `confusion` is a 15×15 confusion matrix and *accuracy* is the accuracy of the testing data prediction.
Description: Given BoW features, you will combine `BuildVisualDictionary`, `ComputeBoW`, and `PredictKNN` for scene classification. Your goal is to achieve the accuracy >50%.

CSCI 5561: Assignment #3

Scene Recognition

7 BoW+SVM

`function [label_test_pred] = PredictSVM(feature_train, label_train, feature_test)`
Input: `feature_train` is a $n_{\text{tr}} \times d$ matrix where n_{tr} is the number of training data samples and d is the dimension of image feature. Each row is the image feature. `label_train`$\in [1, 15]$ is a n_{tr} vector that specifies the label of the training data. `feature_test` is a $n_{\text{te}} \times d$ matrix that contains the testing features where n_{te} is the number of testing data samples.
Output: `label_test_pred` is a n_{te} vector that specifies the predicted label for the testing data.
Description: You will use a SVM classifier to predict the label of the testing data. You don't have to implement the SVM classifier. Instead, you can use VLFeat `vl_svmtrain`. 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 'non-forest', '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 instance, if 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, and all of the other classifiers are even more negative, the test case would be classified as a kitchen even though none of the classifiers put the test case on the positive side of the decision boundary. When learning an SVM, you have a free parameter 'lambda' which controls how strongly regularized the model is. Your accuracy will be very sensitive to lambda, so be sure to test many values.

CSCI 5561: Assignment #3

Scene Recognition

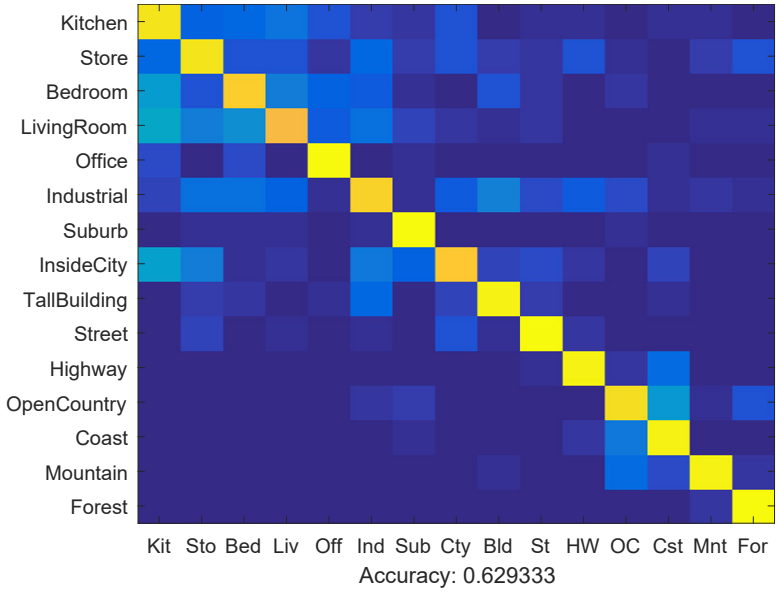

Figure 6: Confusion matrix for BoW+SVM.

`function [confusion, accuracy] = ClassifySVM_BoW`
Output: `confusion` is a 15×15 confusion matrix and *accuracy* is the accuracy of the testing data prediction.
Description: Given BoW features, you will combine `BuildVisualDictionary`, `ComputeBoW`, `PredictSVM` for scene classification. Your goal is to achieve the accuracy >60%.

References

[1] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," *CVPR*, 2006.