Project 3 Writeup

Project Overview

In this project, scene recognition was performed using three different methods, starting with very simple methods – tiny images and nearest neighbor classification – and then move on to more sophisticated methods – bags of words and linear classifiers learned by support vector machines. The accuracy for the BoW/SVM can reach 76.7%, while for BoW/NN, Tiny/SVM, Tiny/NN are 63.1%, 25%, 22.2%, respectively.

Implementation Detail

The implementation order is following the implementation strategy part on project website. The accuracy results will be discussed in the Result section. Any implementations for extra credits will be discussed in the Extra Credit section.

All image operations have been optimized for parallel using multiple CPUs with the help of `Parallel()` from `joblib` package, which reduces the execution time dramatically to 22s on Gradescope, compared to the means of other students' execution time, 200s or so.

Explaination for implementation of get_tiny_images()

First, images are collected using `imread()` using one for loop. Then, the resize operation are done in `Parallel()` by calling the `tiny_feature()` which will resize the image to 16×16 shape.

Explaination for implementation of build_vocabulary()

Similar to `get_tiny_images`, images are collected in one for loop in the begining. Then I have implemented four descriptors to construct the feature space for the images, hog, gaussian_pyramid, gist and daisy. The last three will be discussed in the Extra Credit section.

The detail for hog descriptors is in `hog_feature()` function, `hog()` function from `sklearn` package is used and the parameters have been optimized for the best performance, e.g. `pixels_per_cell_dim` is 8 and `cells_per_block_dim` is 2. Then the features returned from `hog()` are reshaped to (-1, `cells_per_block_dim` × `cells_per_block_dim` × 9)

Then the features obtained for all images are clustered using `MiniBatchKmeans()` with the number of clusters set as 200, i.e. the size of our vocabulary is 200.

Explaination for implementation of get_bags_of_words()

Similar to `build_vocabulary`, images are collected in the begining. To utilized the `Parallel` function, I implement `generate_feats` to encode every image with the vocabulary we generated in the previous function.

To make it consistent with the construction of vocabulary, descriptors type here are the same as those used in `build_vocabulary`. Once the features for each image are obtained. The euclidean distances are calculated using `cdist`. The each feature is labeled by the nearest word. Then we simply count the histogram of the feature labels and return it as the final feature of the image. ”soft assignment” is also implemented and will be discussed in Extra Credit section.

Explaination for implementation of svm_classify()

Multiple SVM methods have been tested here, such as poly, sigmoid, rbf and che-sqr. The results will be discussed in Extra Credit section.

Explaination for implementation of nearest_neighbor_classify()

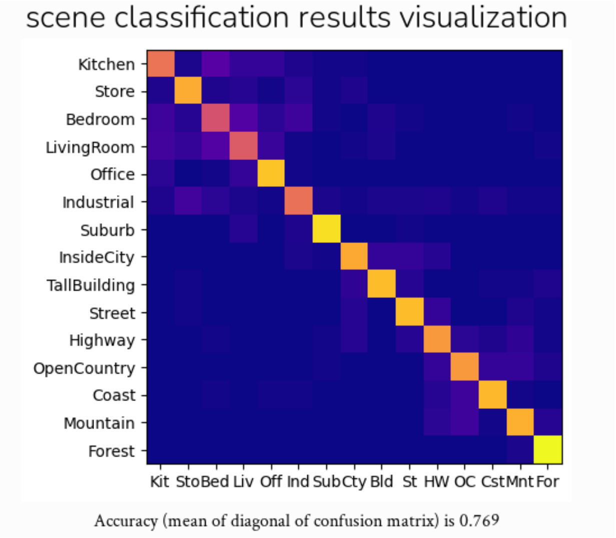
In this function, the feature of the image is labeled by the nearest point. For k points, the label of the image will be the label that appears most times among these nearest k points. Different k values have been tested and weighted votes have been implemented to improve the performance of kNN method. The detail will be discussed in Extra Credit section.

Result

Here we presented the result with optimized parameters. The code here is the same as what we submit to Gradescope, i.e. the SVM method, the hog parameters etc. For `bag_of_words`, we use hog and gaussian_pyramid together to construct vocabulary and encode images. In `get_bags_of_words`, we use soft assignment to obtain the feature of the image. We use poly SVM for `svm_classify` and $k = 15$ with weighted votes for `nearest_neighbor_classify`

Models	Accuracy(%)
bag_of_words/SVM	78.267
bag_of_words/NN	63.333
tiny_image/SVM	24.333
tiny_image/NN	22.467

Table 1: Performance of different methods in provided dataset



Accuracy (mean of diagonal of confusion matrix) is 0.769

Extra Credit (Optional)

1. Feature representation extra credit

   - **Sampling features from different levels of a Gaussian pyramid**: the implementation detail can be found in `gaussian\_pyramid\_feature()`. Simply speaking, `pyramid_gaussian` funtion in `sklearn` is used to constructed the image pyramid. Then images are resized back the original image size and features are constructed based upon these image patches. This method does improve the accuracy by about 4%. Therefore, I used it combined with hog as my final submission.
   - **Add gist descriptors and consider with other existing descriptors together for vocabulary**:

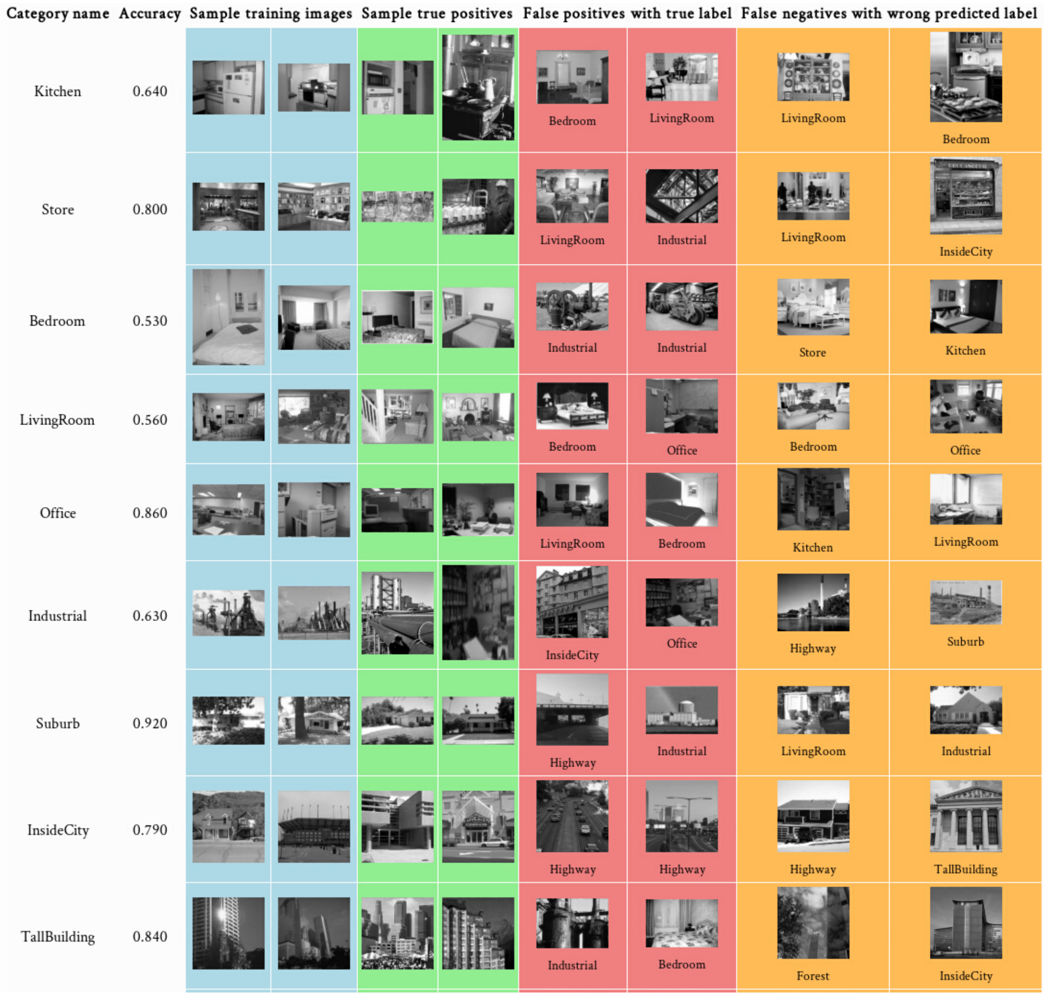
     the implementation detail can be found in `gist_feature()`. Intuitively, GIST summarizes the gradient information (scales and orientations) for different parts of an image, which provides a rough description (the gist) of the scene.Given an input image, the gist descriptor is computed by

     (a) Convolve the image with 32 Gabor filters at 4 scales, 8 orientations, producing 32 feature maps of the same size of the input image.

     (b) Divide each feature map into 16 regions (by a 4x4 grid), and then average the feature values within each region.

     (c) Concatenate the 16 averaged values of all 32 feature maps, resulting in a 16x32=512 GIST descriptor.

     As the resulting feature for the image is (512,), to combine with hog features, I reshape the gist feature to (-1, 36) after throwing away the final 8 dimension in original vector as $512\%36 = 8$. Here is the result for combining hog, gaussian-pyramid and gist descriptors. The parameters are the same as provided in Result section.



Category name	Accuracy	Sample training images		Sample true positives		False positives with true label		False negatives with wrong predicted label	
Kitchen	0.640					Bedroom	LivingRoom	LivingRoom	Bedroom
Store	0.800					LivingRoom	Industrial	LivingRoom	InsideCity
Bedroom	0.530					Industrial	Industrial	Store	Kitchen
LivingRoom	0.560					Bedroom	Office	Bedroom	Office
Office	0.860					LivingRoom	Bedroom	Kitchen	LivingRoom
Industrial	0.630					InsideCity	Office	Highway	Suburb
Suburb	0.920					Highway	Industrial	LivingRoom	Industrial
InsideCity	0.790					Highway	Highway	Highway	TallBuilding
TallBuilding	0.840					Industrial	Bedroom	Forest	InsideCity

Figure 1: For more scences details, please check the index.html file

Figure 2: For more scences details, please check the index.html file

Models	Accuracy(%)
bag_of_words/SVM	78.267
bag_of_words/NN	63.333
tiny_image/SVM	24.333
tiny_image/NN	22.467

Table 2: Performance of different methods using gist in provided dataset

2. Feature quantization and bag of words extra credit

   - **Use ”soft assignment” to assign visual words to histogram bins**: below the corresponding code for soft assignment part. The distances between image features and vocabulary are filtered with gaussian formula. Each visual word will cast a distance-weighted vote to multiple bins. Thus, the further distance between the feature and the word, the less contribution the word to the final bincount number. This will improve the accuracy by about 3%.

```
sigma = 0.05
distances = np.exp(-(distances ** 2) / (sigma ** 2))
distances = distances / np.sum(distances, axis=1)[:, np.
                                             newaxis]
feats = np.sum(distances, axis=0)
```

3. Classifier extra credit

   - **Train the SVM with different kernels**:
     the implementation detail can be found in `svm_classify()`. Different SVM models have been tested here using `SVC` with different kernels. The performance comparison is presented below.

Models	Accuracy(%)
Linear	75.800
poly	78.267
sigmoid	59.933
rbf	72.733
anova/chi2	57.867

Table 3: Performance of different methods using different SVM method on the same vocabulary

4. **Spatial Pyramid representation and classifier**:

   the implementation detail can be found in `build_spatial_pyramid()` and `spatial_pyramid_matching()`.

   This feature is not fully implemented as I haven’t successfully configured how to trace back the location of feature generated by hog. In the paper, the dense

SIFT descriptor is used and thus the location of each feature in the image is the center point location. However, for hog descriptor, the block will overlap with each other and the returned feature vectors are flatten and reshaped, which makes it difficult to figure out how to assign the location of each feature back to the image. Thus, in my implementation, I assume that there exists one function `locate_descriptors()` that can locate the corresponding feature vector in the image and return the positions.

The principle of spatial pyramid representation is that counting the feature label in different scales, and adding these count together with weigting levels' count to a final feature for the image (See `spatial_pyramid_matching()` for weights assignment). Once we could locate the positions of the features and counting them by dividing the original image to different sections. The final feature can be obtained by summing them up.

5. Experimental design extra credit

- **experiment with many different vocabulary sizes**: the performance of model with different vocabulary size is presented below. However, the performance is not stable as the results will fluctuate about 2% due to the randomalization of Kmeans and SVM. We could see that increase the vocabulary size could increase accuracy at first. However, larger vocabulary size doesn't guarantee better performance.

Vocab Size	10	20	50	100	200	400	1000	10000
bag_of_words/SVM	54	63.533	71.933	74.267	76.867	74.933	81.267	77.0
bag_of_words/NN	49	54.4	59.467	61.267	63.133	60.4	65.133	53.6

Table 4: Performance with different vocabulary size

- **report performance on the 397-category SUN database**.