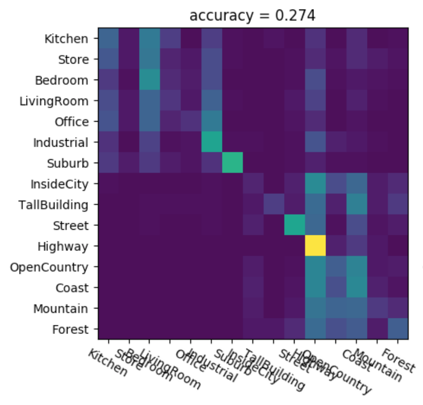
Scene Recognition

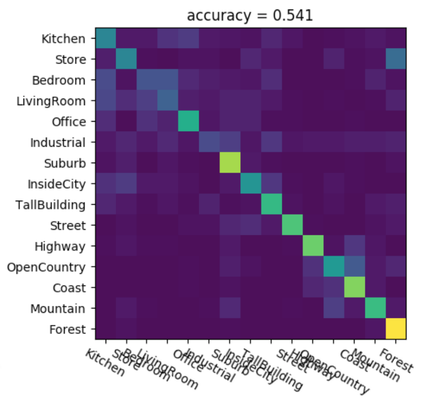
Dongha Kang

March 28, 2020

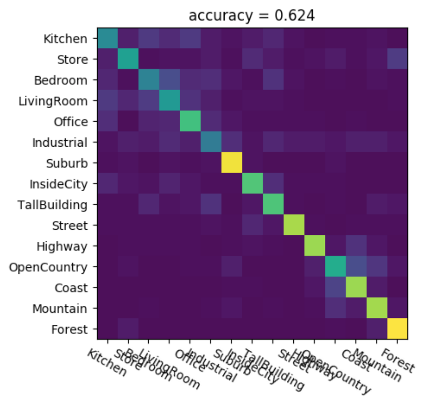
This is the visual recognition to classify the image in 15 different type of image scene including kitchen, office and so on. There are 1500 train images and 1500 test images to evaluate how accurate they match the scene.



(a) Tiny + KNN



(b) BoW + KNN



(c) BoW + SVM

1 Tiny Image

Tiny image KNN (k-nearest neighbor) classification changes all the train images to tiny train images and all the test images to tiny test images. After changing train images and test images to tiny train/test images, run k-nearest neighbor to define if they match similar to each other.

In this case, the tiny image size was created in pixel size of 16 X 16 in the function `get_tiny_image`. In `predict_knn`, I used `sklearn.neighbors`'s `KNeighborClassifier` to calculate K-nearest neighbor. For tiny image, I found that $k = 10$ works the best. With 1500 tiny images, by using `fit` and `predict` functions, `predict_knn` could generate predicted label class. If the predicted label ([0, 14], label class), matches with test image's label classes, the accuracy grows. This approach generated 27.4% accuracy (Figure 1a).

2 Bag-of-word Visual Vocabulary

BoW (Bag-of-words) classification compute sift descriptor on a dense set of locations on image and make a vocabulary with all the dense set of locations on image. With the given vocabulary, it computes the bow with each feature of train images and test images.

`compute_dsift`, computes descriptor points with specific stride and size of the image and creates dense feature. For this particular instance, stride and size was 25. With all the dense features, in `build_visual_dictionary`, visual dictionary was built using `sklearn.cluster.KMeans`, in this particular instance, `n_cluster` (the number of clusters to form) was 200 (saved as dic_size), `n_init` (number of time the k-means algorithm ran) was 5 and `max_iter` (max iteration) was 300. With kmeans that was calculated, dictionary was built by calling `clusters_centers_` to find the mean center of the dense features. With obtained vocabulary, in `compute_bow`, feature and dictionary will be used in `sklearn.neighbors.KNeighborClassifier` to find the closest cluster center. In this case I used 1 for the k value. This `compute_bow` will be used for all the train/test images. to calculate the estimation between train/test images.

All this calculation was done in the function called `load_classify_bow`.

2.1 Using k-nearest neighbor

After `load_classify_bow`, `feature_train, feature_test` will be obtained. Given `feature_train` and `feature_test`, by using `predict_knn` (same as Tiny + KNN), one can estimate similarity between predicted feature test and test images. This approach will provide around 54.1% accuracy (Figure 1b).

2.2 Using SVM

Instead of using KNN algorithm, SVM (Support Vector Machine) can be used. In this particular instance, `sklearn.svm.LinearSVC` was used. `LinearSVC` takes regularization parameter (C), which in this case regularization parameter C = 0.58. I also used method function called `decision_function` that predict confidence scores for samples. With predicted confidence scores, the one with the highest score 'wins' and act as a predicted test label. Same procedure, one can estimate similarity between predicted feature test (SVM) and test images. With this approach, accuracy becomes around 62.4% (Figure 1c).