UNIVERSITY OF MINNESOTA, TWIN CITIES
COLLEGE OF SCIENCE AND ENGINEERING

Assignment #4

Student name: *Hardik Gupta*

Course: *Computer Vision (CSCI 5561)* – Professor: *Dr. Volkan Isler*
Due date: *November 20th, 2023*

Tiny Image kNN Classification

```
def get_tiny_image(img, tiny_size):
    ...
    return feature
```

Input: **img** is an gray scale image, $tiny\_size = (w, h)$ is the tiny image size.
Output: **feature** is a vector of size $wh1$ representing the tiny image.

```
def predict_kNN(feature_train, label_train, feature_test, n_neighbors):
    ...
    return pred_test
```

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, e.g., 256 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_{te} × d matrix that contains the testing features where n_{te} is the number of testing data samples. n_neighbors is the number of neighbors for label prediction.
Output: pred_test is a n_{te} vector that specifies the predicted label for the testing data.

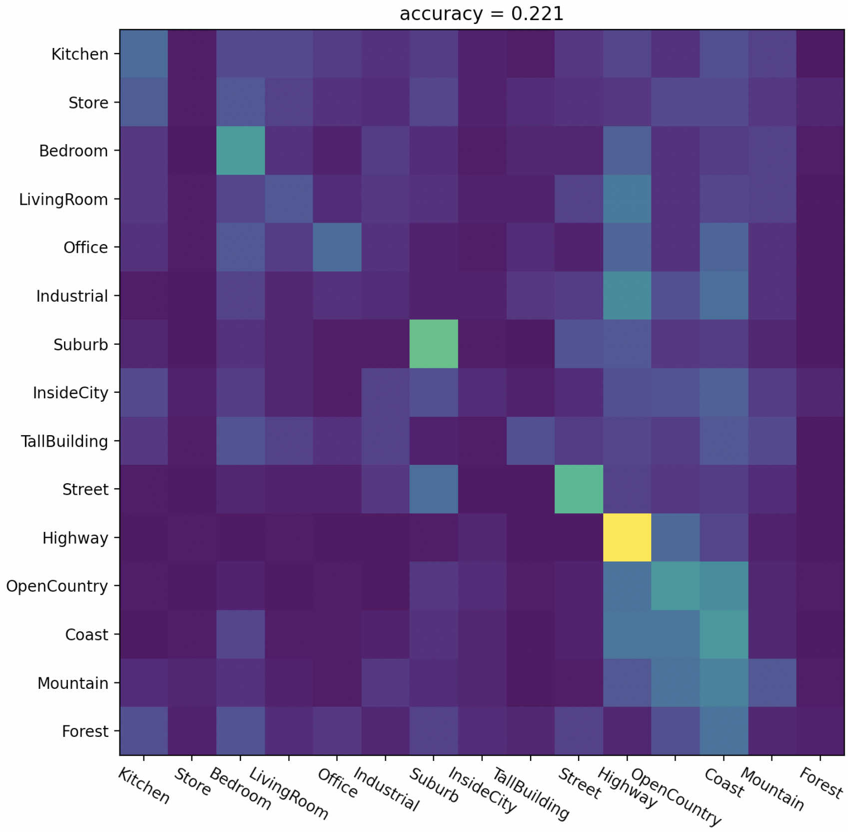
```
def compute_confusion_matrix_and_accuracy(pred_test, label_test, n_classes):
    ...
    return confusion, accuracy
```

Input: *pred_test* is a n_{te} vector that specifies the predicted label for the testing data. $label_test \in [1, 15]$ is a n_{te} vector that specifies the label of the test data. *n_classes* is the number of categories, i.e. 15 in this case.
Output: confusion is a 15 × 15 confusion matrix and accuracy is the prediction accuracy on testing data.

Solution. Resizing of the image in tiny_size. Then vectorization and and normalization of the image to have 0 mean and unit length.

We create an instance of the *KNeighborsClassifier* and fitting the label and features of the samples and produce predictions on test samples.

Then we produce confusion_matrix and accuracy score using the predictions.


accuracy = 0.221
Kitchen
Store
Bedroom
LivingRoom
Office
Industrial
Suburb
InsideCity
TallBuilding
Street
Highway
OpenCountry
Coast
Mountain
Forest

Bag-of-words Visual Vocabulary

```
def compute_dsift(img, stride, size):
    ...
    return dsift
```

Input: *img* is a gray scale image. *stride* and *size* are both integers controls locations on image to compute SIFT features and diameter of the meaningful keypoint neighborhood.
Output: *dsift* is a collection of SIFT features whose size is $n \times 128$. n is total number of locations to compute SIFT features on *img*.

```
def build_visual_dictionary(features, d_size):
    ...
    return vocab
```

Input: features is a collection of SIFT features of all training images ($N \times 128$) and *d_size* is the size of the dictionary (the number of visual words).
Output: vocab lists the quantized visual words whose size is d_size×128.

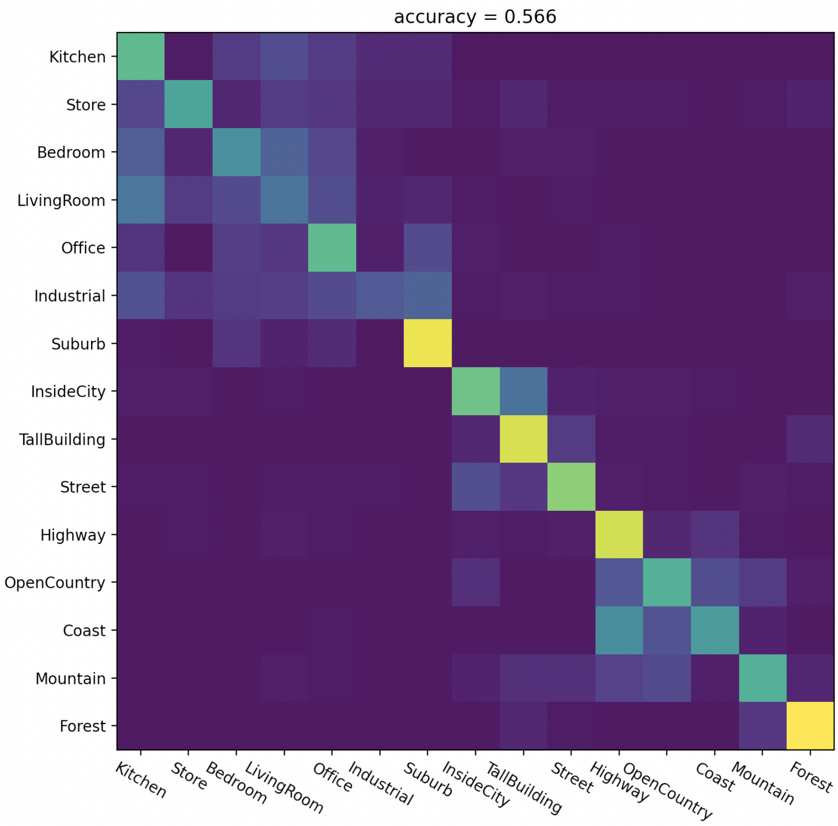
```
def compute_bow(dsift, vocab):
    ...
    return bow_feature
```

Input: *dsift* 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 *d_size*.

Solution. **compute_dsift** function computes dense SIFT (Scale-Invariant Feature Transform) features for an input image with a specified stride and size.

build_visual_dictionary function builds a visual dictionary using K-means clustering (KMeans). It takes a set of features (features) and the desired dictionary size (dict_size). The resulting dictionary is returned.

compute_bow function computes a Bag of Visual Words (BoW) feature representation for a set of dense SIFT features (dsift) using a pre-built visual dictionary (vocab).

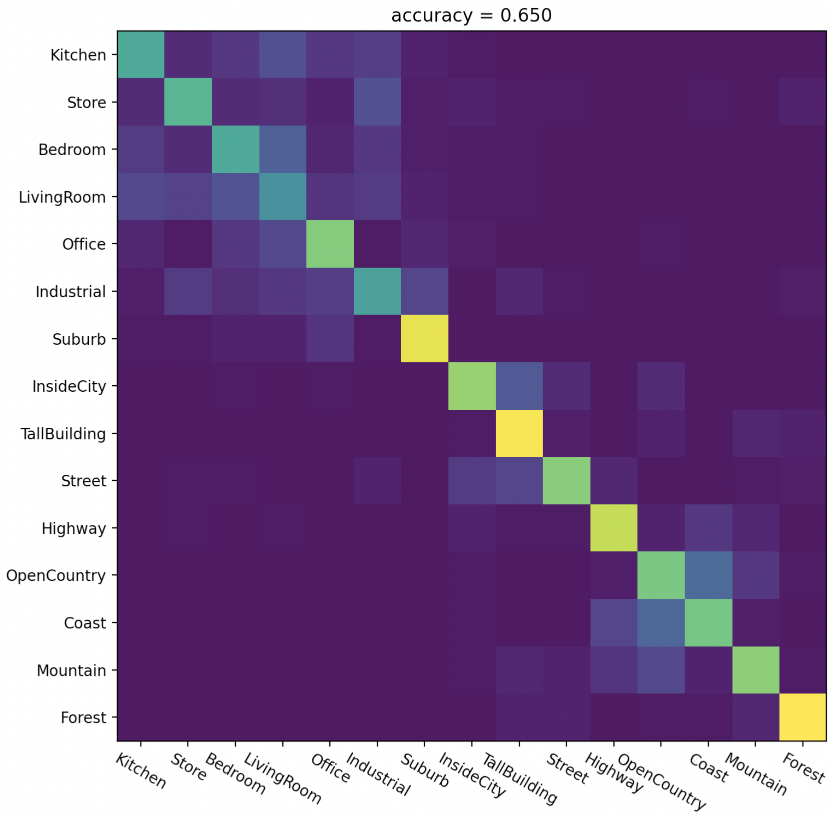


BoW+SVM

```
def predict_svm(feature_train, label_train, feature_test, n_classes):
    ...
    return pred_test
```

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 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 $n_{te} \times d$ matrix that contains the testing features where n_{te} is the number of testing data samples. *n_classes* is the number of categories, i.e., 15 in this case.
Output: *pred_test* is a *n_tr* vector that specifies the predicted label for the testing data.

Solution. **predict_svm** function performs Support Vector Machine (SVM) classification. It takes training features (feature_train) and labels (label_train), as well as test features (feature_test) and the number of classes (n_classes). It returns the predicted labels for the test features using an SVM classifier.


accuracy = 0.650
Kitchen
Store
Bedroom
LivingRoom
Office
Industrial
Suburb
InsideCity
TallBuilding
Street
Highway
OpenCountry
Coast
Mountain
Forest