CPSC 425: Assignment 5
Name: Terence Chen
Student number: 42602136

Claiming the "allotted two late days" given to every student as per the website/piazza
First time claiming any late days this term.

Q1) Nothing to hand in

Q2) Nothing to hand in

Q3) Nothing to hand in

Q4)
Code: util.py

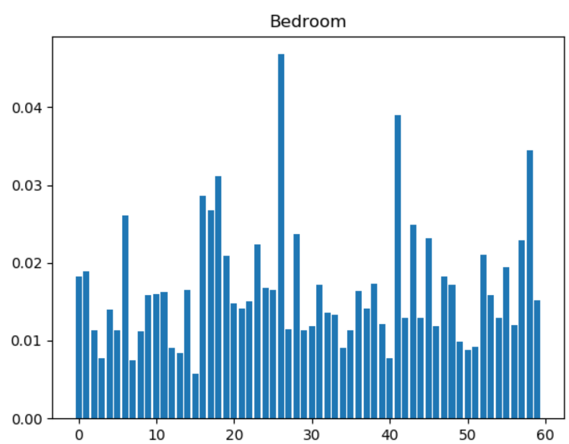
```python
def build_vocabulary(image_paths, vocab_size):
    """ Sample SIFT descriptors, cluster them using k-means, and return the fitted k-means model.
    NOTE: We don't necessarily need to use the entire training dataset. You can use the function
    sample_images() to sample a subset of images, and pass them into this function.

    Parameters
    ----------
    image_paths: an (n_image, 1) array of image paths.
    vocab_size: the number of clusters desired.

    Returns
    -------
    kmeans: the fitted k-means clustering model.
    """
    n_image = len(image_paths)

    # Since want to sample tens of thousands of SIFT descriptors from different images, we
    # calculate the number of SIFT descriptors we need to sample from each image.
    n_each = int(np.ceil(10000 / n_image))  # You can adjust 10000 if more is desired

    # Initialize an array of features, which will store the sampled descriptors
    features = np.zeros((0, 128)) # changed how array was initialized from given code

    for i, path in enumerate(image_paths):
        # Load SIFT features from path
        descriptors = np.loadtxt(path, delimiter=',',dtype=float)

        # TODO: Randomly sample n_each features from descriptors, and store them in features
        # Randomly sample from descriptors using with n_each features
        random_sample = descriptors[(np.random.choice(descriptors.shape[0], min(n_each, descriptors.shape[0]), replace = True)),:]
        features = np.concatenate((features, random_sample), axis = 0) # store in features

    # TODO: pefrom k-means clustering to cluster sampled SIFT features into vocab_size regions.
    # You can use KMeans from sci-kit learn.
    # Reference: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

    kmeans = KMeans(n_clusters = vocab_size).fit(features) # computing k-means clustering
    return kmeans
```

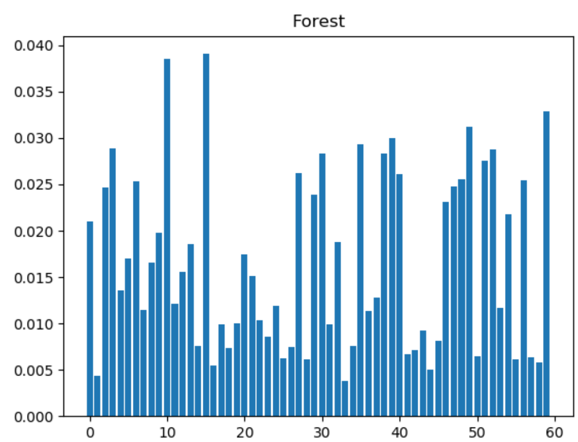
```python
def get_bags_of_sifts(image_paths, kmeans):
    """ Represent each image as bags of SIFT features histogram.

    Parameters
    ----------
    image_paths: an (n_image, 1) array of image paths.
    kmeans: k-means clustering model with vocab_size centroids.

    Returns
    -------
    image_feats: an (n_image, vocab_size) matrix, where each row is a histogram.
    """
    n_image = len(image_paths)
    vocab_size = kmeans.cluster_centers_.shape[0]
    scene_label = []

    image_feats = np.zeros((n_image, vocab_size))

    for i, path in enumerate(image_paths):
        # Load SIFT descriptors
        descriptors = np.loadtxt(path, delimiter=',',dtype=float)

        # TODO: Assign each descriptor to the closest cluster center
        center = kmeans.cluster_centers_ # cluster center computed with KMeans
        closest_cluster = pairwise_distances_argmin(descriptors, center) # closest cluster



        for k in closest_cluster:
            image_feats[i][k] += 1

        # TODO: Build a histogram normalized by the number of descriptors
        image_feats[i] /= descriptors.shape[0] # normalize descriptors
        scene_label.append(path.split('/')[2]) # get the scene label and store it in the list

    #return the scene labels to dispay histogram in main
    return image_feats, scene_label
```

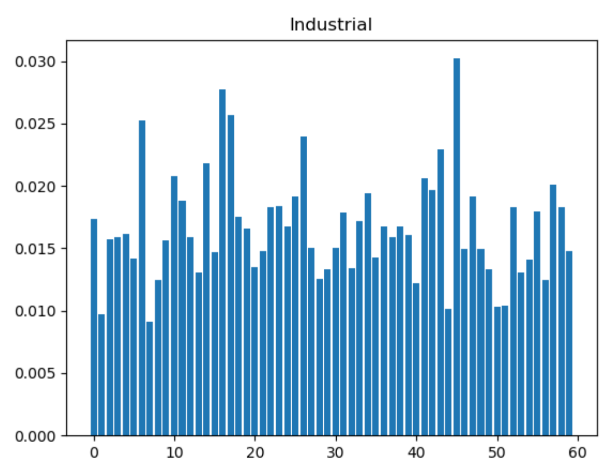
main.py:

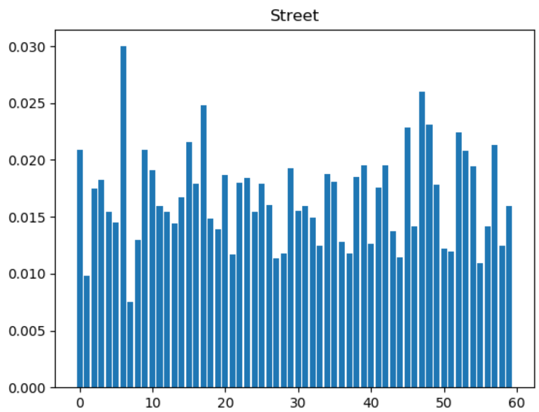
```python
print('Extracting SIFT features\n')
#TODO: You code build_vocabulary function in util.py
# kmeans = build_vocabulary(train_image_paths, vocab_size=200)
v_size = 60
kmeans = build_vocabulary(train_image_paths, vocab_size = v_size)

#TODO: You code get_bags_of_sifts function in util.py
# train scene label is the scene label of the images
train_image_feats, train_scene_labels = get_bags_of_sifts(train_image_paths, kmeans)
test_image_feats = get_bags_of_sifts(test_image_paths, kmeans)

#If you want to avoid recomputing the features while debugging the
#classifiers, you can either 'save' and 'load' the extracted features
#to/from a file.

# Bulid train histograms

matching_hist = {} # initialize dictionary to pair scenelabel and histogram
x_dim = np.arange(train_image_feats.shape[1]) # arrange the number of rows t consturt histograph
for i , label in enumerate(train_labels):
        # match up correspoding key value, key is scene label and value being sum of values with same key
        temp = matching_hist.get(train_scene_labels[i], (np.zeros((1, train_image_feats.shape[1])), 0))
        matching_hist[train_scene_labels[i]] = (np.add(temp[0], train_image_feats[i]), temp[1] + 1)

for key, (value, count) in matching_hist.iteritems():
        # take average
        average = np.divide(value, count)

        # plot histogram
        plt.figure()
        plt.bar(x_dim,average[0])
        plt.title(key)
        plt.savefig(key+'.png')
        plt.close()

```

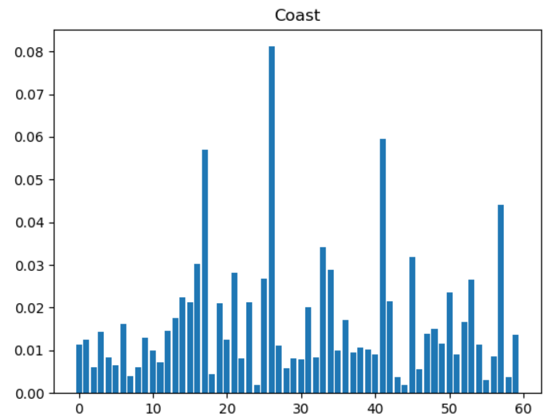
Histograms: vocab_size = 60

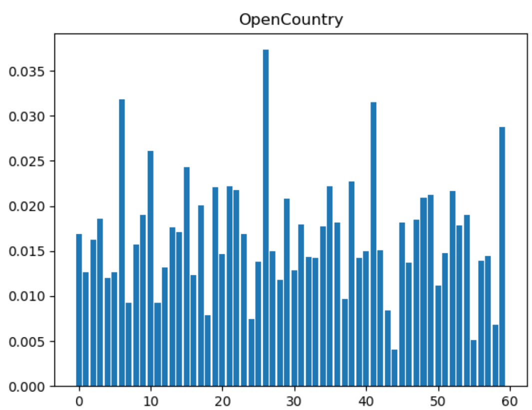


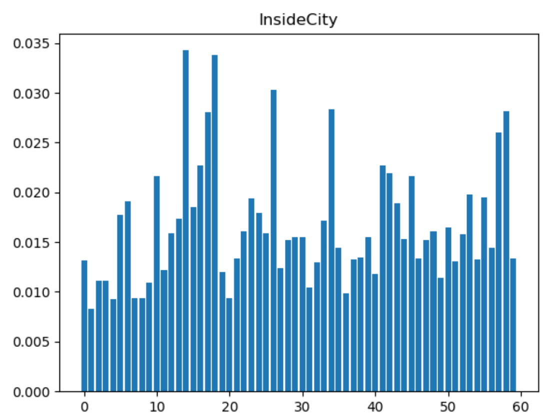


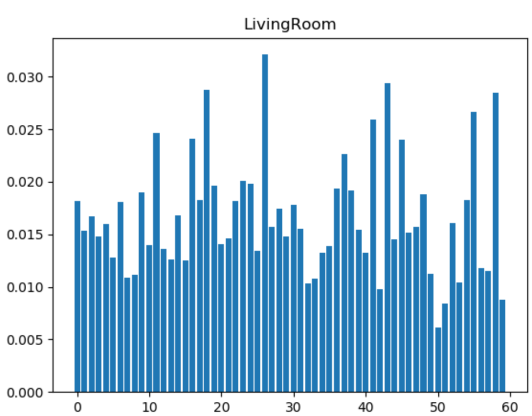


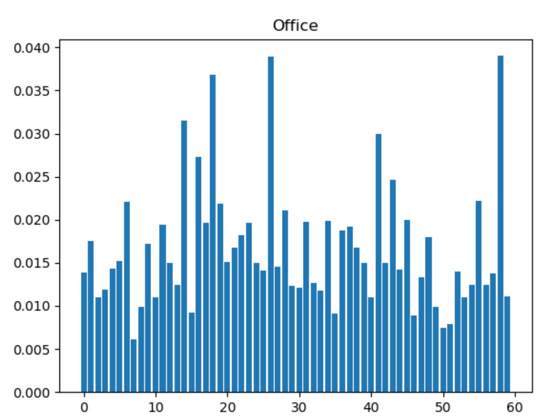

Street
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0 10 20 30 40 50 60

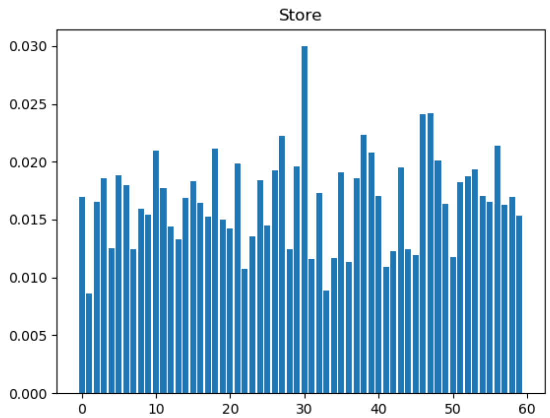

Coast
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0 10 20 30 40 50 60

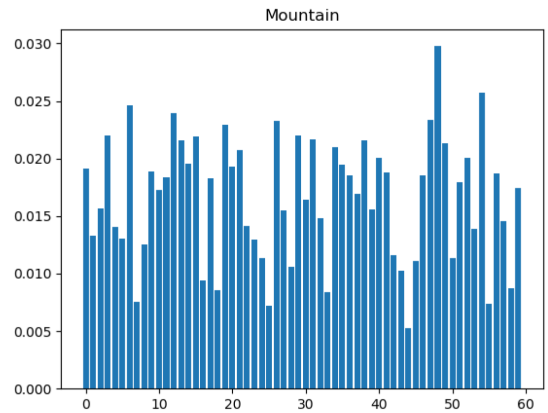

OpenCountry
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0 10 20 30 40 50 60

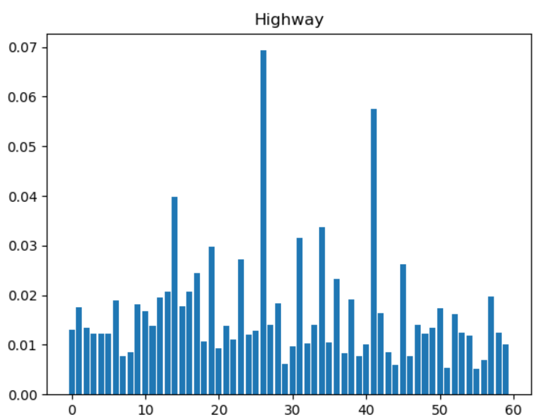

InsideCity
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0 10 20 30 40 50 60


LivingRoom
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0 10 20 30 40 50 60


Office
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0 10 20 30 40 50 60







By looking at these graphs, we can observe that there are clear peaks for some classes while other have two or more bars where they are pretty close to the top. Meaning that the difference between them smaller. Images with less deviation in terms of the bar heights are more difficult to differentiate as there are multiple candidates. Classes with clear peaks are easier to separate as there is a somewhat unanimous decision. While hard to separate classes would be something like InsideCity and Industrial.

Q5)

Code:

classify.py:

```python
def nearest_neighbor_classify(train_image_feats, train_labels, test_image_feats):

    '''
    Parameters
        ----------
        train_image_feats:  is an N x d matrix, where d is the dimensionality of the feature representation.
        train_labels: is an N x l cell array, where each entry is a string
                          indicating the ground truth one-hot vector for each training image.
        test_image_feats: is an M x d matrix, where d is the dimensionality of the
                                  feature representation. You can assume M = N unless you've modified the starter code.

    Returns
        -------
        is an M x l cell array, where each row is a one-hot vector
        indicating the predicted category for each test image.

    Usefull funtion:

        # You can use knn from sci-kit learn.
        # Reference: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

    '''
     # perform kmn as according to scikit example
    neighbor = KNeighborsClassifier(n_neighbors=20)
    neighbor.fit(train_image_feats, train_labels)
    predicted_labels = neighbor.predict(test_image_feats[0])
    return predicted_labels

```

main.py:

```python
print('Using nearest neighbor classifier to predict test set categories\n')
#TODO: YOU CODE nearest_neighbor_classify function from classifers.py
pred_labels_knn = nearest_neighbor_classify(train_image_feats, train_labels, test_image_feats)
knn_true = np.sum(pred_labels_knn == test_labels) # true positives where test labels match with the prediction


print('Using support vector machine to predict test set categories\n')
#TODO: YOU CODE svm_classify function from classifers.py
pred_labels_svm = svm_classify(train_image_feats, train_labels, test_image_feats)
svm_true = np.sum(pred_labels_svm == test_labels) # true positives where test labels match with the prediction

print('---Evaluation---\n')
all_test_label = len(test_labels) # all labels
# Step 3: Build a confusion matrix and score the recognition system for
#         each of the classifiers.
# TODO: In this step you will be doing evaluation.
# 1) Calculate the total accuracy of your model by counting number
#    of true positives and true negatives over all.
# 2) Build a Confusion matrix and visualize it.
#    You will need to convert the one-hot format labels back
#    to their category name format.

# 1) calculate total accuracy

print('KNN Total Accuracy: ')
print(float(knn_true)/all_test_label)

print('SVM Total Accuracy: ')
print(float(svm_true)/all_test_label)

# 2) Confusion matrix

print('KNN Confusion Matrix')
print(confusion_matrix(test_labels,pred_labels_knn))

print('SVM Confusion Matrix')
print(confusion_matrix(test_labels,pred_labels_svm))

```

result: k =20

```
KNN Total Accuracy:
0.34
```

```
KNN Confusion Matrix
[[7 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 2 0 0 0 0 0 0]
 [0 0 4 0 0 0 1 1 0 0 0 0 1 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 2 1 0 0 1 0 2 0 1 0 0 0 0]
 [0 2 0 0 0 2 0 1 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0 0 0 1 0 0 0 1]
 [0 0 0 0 0 0 0 4 3 0 1 0 0 0 1]
 [0 0 1 0 0 0 0 1 6 0 0 0 0 0 0]
 [1 0 3 0 0 1 0 1 0 0 2 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 5 0 0 0 0]
 [2 1 0 0 0 0 0 1 1 0 3 2 0 0 0]
 [0 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [4 0 0 0 0 0 0 0 3 0 1 3 0 0 0]
 [3 0 1 0 0 0 0 1 1 0 2 0 0 0 2]]
```

Unfortunately for me, due to busy schedule I wasn't able to play around with the value k in a lot of detail. For the classifier portion I referenced the scikit-learn.org site example shown in the assignment descriptions. However, from my testing I found that for knn, if I went to low (< 10) or to high the accuracy drops. I found that around k =18-25 yielded me somewhat ok results.

Q6)

classify.py:

```python
def svm_classify(train_image_feats, train_labels, test_image_feats):

    '''
    Parameters
        ----------
        train_image_feats:  is an N x d matrix, where d is the dimensionality of the feature representation.
        train_labels: is an N x l cell array, where each entry is a string
                                indicating the ground truth one-hot vector for each training image.
        test_image_feats: is an M x d matrix, where d is the dimensionality of the
                                        feature representation. You can assume M = N unless you've modified the starter code.

    Returns
        -------
        is an M x l cell array, where each row is a one-hot vector
        indicating the predicted category for each test image.

    Usefull funtion:

        # You can use svm from sci-kit learn.
        # Reference: https://scikit-learn.org/stable/modules/svm.html

    '''
    # perform svm as according to scikit example
    clf = svm.SVC(kernel = 'linear', C = 158 )
    clf.fit(train_image_feats, train_labels)
    predicted_labels = clf.predict(test_image_feats[0])
    return predicted_labels
```

main.py code same as q5)

Result: C =158

```
SVM Total Accuracy:
0.5
```

```
SVM Confusion Matrix
[[7 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 1 4 0 0 0 1 0 0 0 0 0 1 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 2 6 0 0 0 1 0 0 0 0 0 0]
 [0 0 1 1 0 3 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 2 0 0 0 0 0 0 0 1]
 [0 1 0 1 0 0 0 3 2 0 0 0 0 0 2]
 [0 1 0 0 0 0 0 0 6 0 0 0 0 0 1]
 [0 3 1 0 0 0 1 2 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 5 0 0 0 0]
 [2 1 0 0 0 0 0 1 0 0 2 4 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 1 0 0]
 [2 0 0 0 1 0 0 0 3 0 0 1 0 4 0]
 [1 0 0 0 0 0 0 2 2 0 2 0 1 0 2]]
```

Again same with the knn, except this time I chose C =158. If I take C too low < 100 or too high, I lose some of the accuracy. In the end I found that around the 150-165 range to be somewhat ok in terms of the accuracy. I was able to achieve a better result with svm than KNN.