16-720A — Spring 2021 — Homework 1

Jen-Hung Ho
jenhungh@andrew.cmu.edu

February 19, 2021

Collaborators: NONE

Question 1

Q1.1.1

The properties of each filter functions:
(1) Gaussian: Gaussian filter is a low-pass filter that reduces high-frequency noises. The effects of Gaussian filter are smoothing and blurring the images by taking the weighted average among central pixel and pixels around.
(2) Laplacian of Gaussian (LoG): LoG is an edge detection filter. The effect of LoG is picking up the edges in the image by taking second derivative to detect sudden changes.
(3) Derivative of Gaussian in the x-direction: Taking derivative in the x-direction will pick up the vertical edges in the image.
(4) Derivative of Gaussian in the y-direction: Taking derivative in the y-direction will pick up the horizontal edges in the image.

We can split the filters into two group: Gaussian filter belongs to the smoothing filters and the other three belong to the edge detection filters.

The reason that we need multiple scales of filter responses is that we don't exactly know the size of the features we want to extract. Thus, we need different amounts of details of the image to collect all features.

Q1.1.2

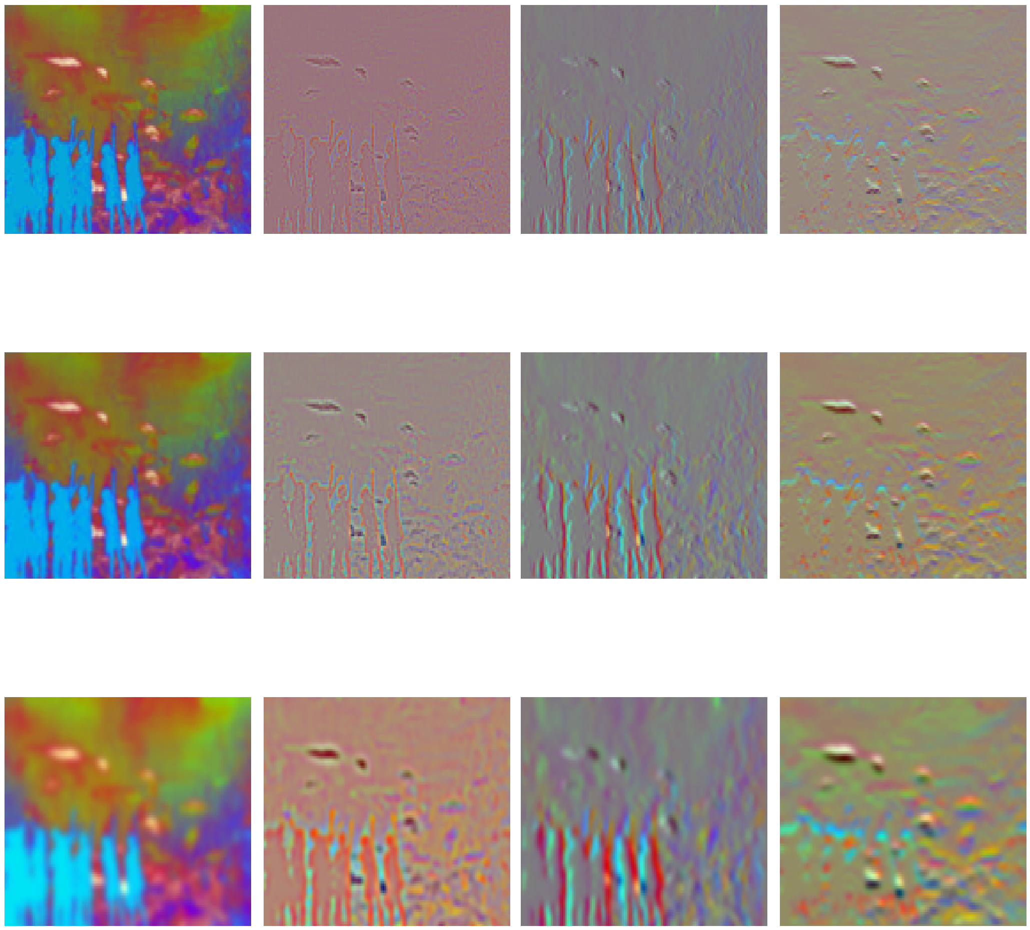

Figure 1: Filter responses of each filter on aquarium/sun_aztvjgubyrgvirup.jpg

Q1.3

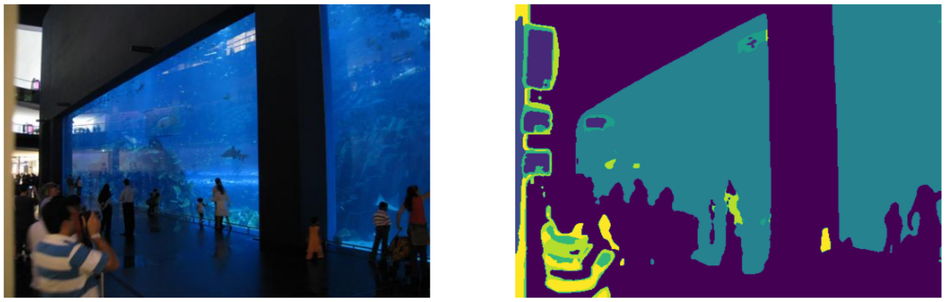

Figure 2: Original image and wordmaps of aquarium/sun_aadolwejqiytvyne.jpg

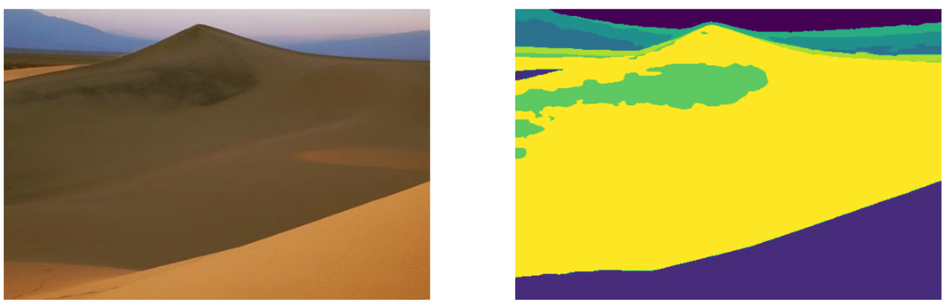

Figure 3: Original image and wordmaps of desert/sun_aaqyzvrweabdxjzo.jpg

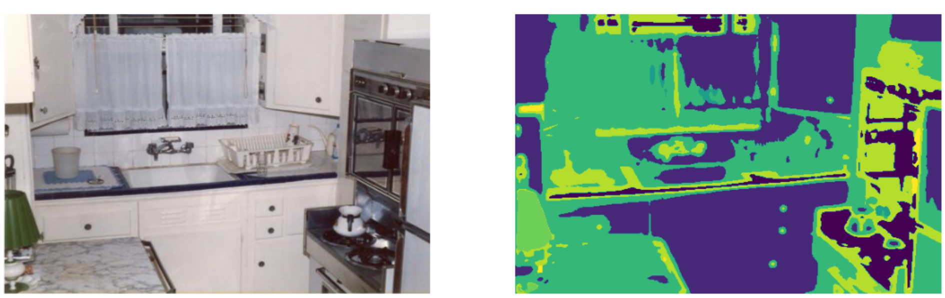

Figure 4: Original image and wordmaps of kitchen/sun_aaebjpeispxohmfv.jpg

The word boundaries make sense to me in these three images. Image of aquarium and kitchen have more features and thus there are more visual words (different colors). For image like desert, there are few visual words since most regions in the image are uniform.

Question 2

Q2.5

The default hyperparameter setting: filter scales = [1, 2], K = 10, alpha = 25, L = 1.
The confusion matrix:

```
[[32.  1.  2.  4.  1.  2.  4.  4.]
 [ 0. 28.  4.  6.  5.  0.  2.  5.]
 [ 1.  4. 33.  1.  1.  1.  1.  8.]
 [ 3.  2.  3. 33.  6.  1.  1.  1.]
 [ 3.  2.  1. 11. 26.  3.  3.  1.]
 [ 2.  1.  5.  1.  3. 30.  4.  4.]
 [ 6.  1.  1.  1.  6. 12. 20.  3.]
 [ 4.  7.  4.  0.  2.  7.  5. 21.]]
```

The accuracy = 55.75%

Q2.6

The correct ratio of each class:
aquarium (label 0): correct ratio = 64%
desert (label 1): correct ratio = 56%
highway (label 2): correct ratio = 66%
kitchen (label 3): correct ratio = 66%
laundromat (label 4): correct ratio = 52%
park (label 5): correct ratio = 60%
waterfall (label 6): correct ratio = 40%
windmill (label 7): correct ratio = 42%

From the correct ratio, we can know that **waterfall** and **windmill** are hard classes that are more difficult to classify. According to the confusion matrix, the **waterfall** class is mostly misclassified into **park**. It is probably because of the massive uniform regions (waterfall) and various terrains around the waterfall, which makes it hard to classify. The **windmill** class is mostly misclassified into **park** and **desert**. This is probably because many **windmill** images contain other objects (ex. trees) that might increase the possibility of misclassification.

Question 3

Q3.1

Change	Filter Scales	K	Alpha	L	Accuracy
default	[1, 2]	10	25	1	55.75%
K	[1, 2]	100	25	1	62%
K	[1, 2]	150	25	1	60.75%
L	[1, 2]	10	25	0	45.25%
L	[1, 2]	10	25	2	55.75%
L	[1, 2]	10	25	3	53.75%
alpha	[1, 2]	10	50	1	55.25%
filter scales	[1, 2, 5]	10	25	1	54%
K and L	[1, 2]	100	25	0	58.25%
K and L	[1, 2]	100	25	2	64.75%
K and L	[1, 2]	100	25	3	63.5%
K and L	[1, 2]	150	25	0	58%
K and L	[1, 2]	150	25	2	64.5%
K and L	[1, 2]	150	25	3	64.75%

Table 1: Table of ablation study of hyperparameters tuning

I have tried to increase K, L, alpha, and filter scales. The maximum accuracy I could reach is **64.75%** with **K = 100, L = 2, alpha = 25, and filter scales = [1,2]**. K is the number of visual words in the dictionary, so normally higher K means more references in the dictionary. However, since we randomly select alpha pixels in a image, K should have a upper boundary. Increasing L means there are more layers in the Spatial Pyramid, which splits the image into more cells. This makes sense that L should have a proper values (not too big) since single cell might not contain enough information if its too small. Adding and changing filter scales depend on the size of the features. (In this dataset, filter scales = [1, 2] is enough to use.)

Extra Credit

To improve the performance, change the relative parameters (weights/cells of SPM, similarity score of two histograms) in custom_visual_recog.py and run custom.py.

1. Change the structure of weights of the Spatial Pyramid

Changing the base of the weights and cells (chop the image into 3^l x 3^l) can increase the number of cells in each layer. This will improve the performance if L is small. (Table 2 shows that the accuracy increases slightly when L = 1.)

Change	Filter Scales	K	Alpha	L	weights and cells	Accuracy
default	[1, 2]	10	25	1	base = 2	55.75%
SPM weights and cells	[1, 2]	10	25	1	base = 3	56.5%
default	[1, 2]	100	25	1	base = 2	62%
SPM weights and cells	[1, 2]	100	25	1	base = 3	67%

Table 2: Performance Improvement: Change of structure of weights of the Spatial Pyramid

2. Replace the histogram intersection with some other similarity scores

Change	Filter Scales	K	Alpha	L	weights and cells	Accuracy
default	[1, 2]	10	25	1	base = 2	55.75%
Euclidean Distance	[1, 2]	10	25	1	base = 2	52.75%
Kolmogorov-Smirnov Distance	[1, 2]	10	25	1	base = 2	47.25%
Match Distance	[1, 2]	10	25	1	base = 2	55.75%

Table 3: Performance Improvement: Replace the histogram intersection with some other similarity scores

There are lots of similarity scores for two histograms.[1] I tried to use Euclidean distance, Kolmogorov-Smirnov distance, and matching distance as the similarity scores of SPM.

For Euclidean distance, the similarity score is

$$\sqrt{\sum_i [h_1(i) - h_2(i)]^2} \tag{1}$$

The Euclidean distance calculates the distance between corresponding bars of two histograms. Using Euclidean distance as similarity scores is expected to get a lower accuracy since the distance between corresponding bars might be small, which makes the similarity scores similar and thus misclassification will happen. As shown in Table 3, the accuracy really decreases.

For Kolmogorov-Smirnov distance, the similarity score is

$$max(|h_1(i) - h_2(i)|) \tag{2}$$

The Kolmogorov-Smirnov distance is expected to get a lower accuracy since it uses the difference between corresponding bars, which might not provide enough similarity information between two histograms. As shown in Table 3, the accuracy decreases even more than Euclidean distance.

For Match distance, the similarity score is

$$\sum_i |h_1(i) - h_2(i)| \tag{3}$$

The Match distance is expected to get the same accuracy as the default setting (intersection similarity) since they have the same similarity scores.(Match distance is two times bigger than intersection, but the ratio is the same.) As shown in Table 3, the accuracy is the same as the default setting.

From my observation, **intersection similarity** seems to be the best similarity scores to use.

[1]Website (*https://stats.stackexchange.com/questions/7400/how-to-assess-the- similarity-of-two-histograms*) provides many methods to assess the similarity of two histograms.

Code Appendix

main.py

```
from os.path import join

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

import util
import visual_words
import visual_recog
from opts import get_opts


def main():
    opts = get_opts()

    # Q1.1
    img_path = join(opts.data_dir, 'kitchen/sun_aasmevtpkslccptd.jpg')
    img = Image.open(img_path)
    img = np.array(img).astype(np.float32) / 255
    filter_responses = visual_words.extract_filter_responses(opts, img)
    util.display_filter_responses(opts, filter_responses)

    # Q1.2
    # n_cpu = util.get_num_CPU()
    # visual_words.compute_dictionary(opts, n_worker=n_cpu)

    # Q1.3
    # Visualize wordmaps for three images (aquarium, desert, kitchen)
    # img_path = join(opts.data_dir, 'aquarium/sun_aadolwejqiytvyne.jpg')
    # img_path = join(opts.data_dir, 'desert/sun_aaqyzvrweabdxjzo.jpg')
    # img_path = join(opts.data_dir, 'kitchen/sun_aaebjpeispxohmfv.jpg')
    # img = Image.open(img_path)
    # img = np.array(img).astype(np.float32) / 255
    # dictionary = np.load(join(opts.out_dir, 'dictionary.npy'))
    # wordmap = visual_words.get_visual_words(opts, img, dictionary)
    # util.visualize_wordmap(img, wordmap)

    # Check Q2.1, Q2.2, Q2.4
    # Q2.1 Test histogram
    # hist = visual_recog.get_feature_from_wordmap(opts, wordmap)
    # Q2.2 Test histogram_all
    # hist_all = visual_recog.get_feature_from_wordmap_SPM(opts, wordmap)
    # Q2.4 Test get_image_feature
    # feature = visual_recog.get_image_feature(opts, img_path, dictionary)
```

```python

    # Q2.1-2.4
    # n_cpu = util.get_num_CPU()
    # visual_recog.build_recognition_system(opts, n_worker=n_cpu)

    # Q2.5
    # n_cpu = util.get_num_CPU()
    # conf, accuracy = visual_recog.evaluate_recognition_system(
    #     opts, n_worker=n_cpu)

    # print(conf)
    # print(accuracy)
    # np.savetxt(join(opts.out_dir, 'confmat.csv'),
    #            conf, fmt='%d', delimiter=',')
    # np.savetxt(join(opts.out_dir, 'accuracy.txt'), [accuracy], fmt='%g')


if __name__ == '__main__':
    main()
```

opts.py

```
'''
Hyperparameters wrapped in argparse
This file contains most of tunable parameters for this homework
You are asked to play around with them for Q3.1
It is recommended that you leave them as they are before getting to Q3.1

You can change the values by changing their default fields or by command-line
arguments. For example, "python main.py --filter-scales 2 5 --K 50"
'''

import argparse


def get_opts():
    parser = argparse.ArgumentParser(
        description='16-720 HW1: Scene Recognition')

    # Paths
    parser.add_argument('--data-dir', type=str, default='../data',
                        help='data folder')
    parser.add_argument('--feat-dir', type=str, default='../feat',
                        help='feature folder')
    parser.add_argument('--out-dir', type=str, default='.',
                        help='output folder')

    # Visual words (requires tuning)
    parser.add_argument('--filter-scales', nargs='+', type=float,
                        default=[1, 2],
                        help='a list of scales for all the filters')
    parser.add_argument('--K', type=int, default=10,
                        help='# of words')
    parser.add_argument(
        '--alpha', type=int, default=25,
        help='Using only a subset of alpha pixels in each image'
    )

    # Recognition system (requires tuning)
    parser.add_argument('--L', type=int, default=1,
                        help='# of layers in spatial pyramid matching (SPM) = L+1')

    # Additional options (add your own hyperparameters here)

    ##
    opts = parser.parse_args()
    return opts
```

visual_recog.py

```
import os
import math
import multiprocessing
from os.path import join
from copy import copy

import numpy as np
from PIL import Image

import visual_words
import matplotlib.pyplot as plt


def get_feature_from_wordmap(opts, wordmap):
    '''
    Compute histogram of visual words.

    [input]
    * opts: options
    * wordmap: numpy.ndarray of shape (H,W)

    [output]
    * hist: numpy.ndarray of shape (K)
    '''

    # Set up the parameters
    K = opts.K                                      # size of bins

    # Create the histogram from wordmap and Normalize
    hist, label = np.histogram(wordmap, bins = np.arange(K+1))
    hist = hist / np.sum(hist)                      # Normalize

    # Plot the histogram to check
    # plt.bar(range(K), hist)
    # plt.title(f"K = {K}, histogram of aquarium/sun_aadolwejqiytvyne.jpg")
    # plt.show()

    return hist


def get_feature_from_wordmap_SPM(opts, wordmap):
    '''
    Compute histogram of visual words using spatial pyramid matching.

    [input]
    * opts: options
```

```
    * wordmap: numpy.ndarray of shape (H,W)

    [output]
    * hist_all: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
    '''

    # Set up the parameters and Initialize hist_all
    K = opts.K
    L = opts.L
    row, col = wordmap.shape
    hist_all = np.array([], dtype = np.float32).reshape(1, 0)

    # Spatial Pyramid Matching : Slow Method (loop over each layers)
    # for layer_index in range(L):
    #     # Set up the weight of each layer
    #     if (layer_index == 0 or layer_index == 1):
    #         weight = 2 ** (-L)
    #     else:
    #         weight = 2 ** (L - layer_index - 1)

    #     # Chop the image into cells
    #     num_cell = 2 ** layer_index
    #     cell_row = int(row/num_cell)
    #     cell_col = int(col/num_cell)

    #     # Concatenate all histograms
    #     for row_index in range(num_cell):
    #         for col_index in range(num_cell):
    #             small_wordmap = wordmap[cell_row*row_index :
    #                                 cell_row*(row_index+1), cell_col*col_index :
    #                                 cell_col*(col_index+1)]
    #             single_hist = get_feature_from_wordmap(opts, small_wordmap)
    #             hist_all = np.append(hist_all, single_hist * weight)

    # Spatial Pyramid Matching : Fast Method
    # Start from the finest(top) layer
    # Set up the number of cells and size of each cell
    num_cell = 2 ** L
    cell_row = int(row/num_cell)
    cell_col = int(col/num_cell)

    # Initialize the finest layer and its weight
    finest_layer = np.zeros((num_cell, num_cell, K))
    if (L == 0 or L == 1):
        weight = 2 ** (-L)
    else:
        weight = 1/2

```

```
    # Compute the histograms of the finest layer
    for row_index in range(num_cell):
        for col_index in range(num_cell):
            small_wordmap = wordmap[cell_row*row_index : cell_row*(row_index+1),
                                    cell_col*col_index : cell_col*(col_index+1)]
            single_hist = get_feature_from_wordmap(opts, small_wordmap)
            finest_layer[row_index, col_index, :] = single_hist
    hist_all = np.append(finest_layer.reshape(1,-1)[0] * weight, hist_all)

    # Aggregate the remaining layers
    for layer_index in range(L-1, -1, -1):
        # Set up the weight of each layer
        if (layer_index == 0 or layer_index == 1):
            weight = 2 ** (-L)
        else:
            weight = 2 ** (layer_index - L - 1)

        # Aggregate the remaining layers from the finest layer
        num_cell = 2 ** layer_index
        single_layer = np.zeros((num_cell, num_cell, K))
        for row_index in range(num_cell):
            for col_index in range(num_cell):
                single_layer[row_index, col_index, :] =
                np.sum(finest_layer[row_index*2 : (row_index+1)*2, col_index*2
                                    : (col_index+1)*2, :], axis = (0, 1))
        hist_all = np.append(single_layer.reshape(1,-1)[0] * weight, hist_all)

    # Normalization
    hist_all = hist_all / np.sum(hist_all)

    # Plot the histogram_all to check
    # plt.bar(range(hist_all.shape[0]), hist_all)
    # plt.title(f"K = {K}, L ={L}, size = {hist_all.shape[0]}, histogram_all of
    #              aquarium/sun_aadolwejqiytvyne.jpg")
    # plt.show()

    return hist_all


def get_image_feature(opts, img_path, dictionary):
    '''
    Extracts the spatial pyramid matching feature.

    [input]
    * opts: options
    * img_path: path of image file to read
    * dictionary: numpy.ndarray of shape (K, 3F)

```

```

    [output]
    * feature: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
    '''

    # Load the image and check the data type and dimensions
    img = Image.open(img_path)
    img = np.array(img).astype(np.float32) / 255
    if len(img.shape) == 2:
        img = img[:, :, np.newaxis]
        img = np.tile(img, (1, 1, 3))

    # Extract the wordmap from the image (use dictionary)
    wordmap = visual_words.get_visual_words(opts, img, dictionary)

    # Compute the Spatial Pyramid Matching features (use wordmap)
    feature = get_feature_from_wordmap_SPM(opts, wordmap)

    # Plot the feature to check
    # plt.bar(range(feature.shape[0]), feature)
    # plt.title(f"size = {feature.shape[0]}, SPM feature of
    #               aquarium/sun_aadolwejqiytvyne.jpg")
    # plt.show()

    return feature


def build_recognition_system(opts, n_worker=1):
    '''
    Creates a trained recognition system by generating training features from
    all training images.

    [input]
    * opts: options
    * n_worker: number of workers to process in parallel

    [saved]
    * features: numpy.ndarray of shape (N,M)
    * labels: numpy.ndarray of shape (N)
    * dictionary: numpy.ndarray of shape (K,3F)
    * SPM_layer_num: number of spatial pyramid layers
    '''

    # Set up the file path and load the training files
    data_dir = opts.data_dir
    out_dir = opts.out_dir
    SPM_layer_num = opts.L

```

```
    # Load the trainig files and labels
    train_files = open(join(data_dir, 'train_files.txt')).read().splitlines()
    training_img_num = len(train_files)
    train_labels = np.loadtxt(join(data_dir, 'train_labels.txt'), np.int32)
    dictionary = np.load(join(out_dir, 'dictionary.npy'))

    # Multiprocessing to extract the traing features
    opts_list = [opts] * training_img_num
    img_path = [join(data_dir, img_name) for img_name in train_files]
    dictionary_list = [dictionary] * training_img_num
    args = zip(opts_list, img_path, dictionary_list)
    pool = multiprocessing.Pool(n_worker)
    features = pool.starmap(get_image_feature, args)

    # example code snippet to save the learned system
    np.savez_compressed(join(out_dir, 'trained_system.npz'), features =
    features, labels = train_labels, dictionary = dictionary,
    SPM_layer_num = SPM_layer_num)


def distance_to_set(word_hist, histograms):
    '''
    Compute distance between a histogram of visual words with all training
    image histograms.

    [input]
    * word_hist: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
    * histograms: numpy.ndarray of shape (T,K*(4^(L+1)-1)/3)

    [output]
    * dis: numpy.ndarray of shape (T)
    '''

    # Compute the intersection similarity bectween word_hist and histograms
    num_features, concantenated_size = histograms.shape
    intersection_similarity = np.minimum(word_hist, histograms)

    # Compute the distance (inverse of the intersection similarity)
    dis = np.full((num_features), 1) - np.sum(intersection_similarity, axis = 1)

    return dis


def evaluate_recognition_system(opts, n_worker=1):
    '''
    Evaluates the recognition system for all test images and returns the
    confusion matrix.

```

```
    [input]
    * opts: options
    * n_worker: number of workers to process in parallel

    [output]
    * conf: numpy.ndarray of shape (8,8)
    * accuracy: accuracy of the evaluated system
    '''

    # Set up file path and Load traind data
    data_dir = opts.data_dir
    out_dir = opts.out_dir
    trained_system = np.load(join(out_dir, 'trained_system.npz'))
    dictionary = trained_system['dictionary']
    trained_features = trained_system['features']
    trained_labels = trained_system['labels']

    # Use the stored options in the trained system instead of opts.py
    test_opts = copy(opts)
    test_opts.K = dictionary.shape[0]
    test_opts.L = trained_system['SPM_layer_num']

    # Load the test data
    test_files = open(join(data_dir, 'test_files.txt')).read().splitlines()
    test_img_num = len(test_files)
    test_labels = np.loadtxt(join(data_dir, 'test_labels.txt'), np.int32)

    # Extract the features from test data
    opts_list = [opts] * test_img_num
    img_path = [join(data_dir, img_name) for img_name in test_files]
    dictionary_list = [dictionary] * test_img_num
    args = zip(opts_list, img_path, dictionary_list)
    pool = multiprocessing.Pool(n_worker)
    test_features = np.asarray(pool.starmap(get_image_feature, args))
    # np.savez_compressed(join(out_dir, 'test_system.npz'), features =
      test_features)

    # Compute the predicted labels
    pred_labels = []
    for test_index in range(test_img_num):
        pred_index = np.argmin(distance_to_set(test_features[test_index, :],
                               trained_features))
        pred_labels.append(trained_labels[pred_index])
    pred_labels = np.asarray(pred_labels)

    # Compute the Confusion Matrix and Accuracy
    confusion_matrix = np.zeros((8, 8))
    for true_index, pred_index in zip(test_labels, pred_labels):
```

```
            confusion_matrix[true_index][pred_index] += 1
    accuracy = np.sum(np.diag(confusion_matrix)) / np.sum(confusion_matrix)

    return confusion_matrix, accuracy
```

visual_words.py

```
import os
import multiprocessing
from os.path import join, isfile

import numpy as np
import scipy.ndimage
import skimage.color
from skimage import io
from PIL import Image
from sklearn.cluster import KMeans

from opts import get_opts
opts = get_opts()

def extract_filter_responses(opts, img):
    '''
    Extracts the filter responses for the given image.

    [input]
    * opts: options
    * img: numpy.ndarray of shape (H,W) or (H,W,3)
    [output]
    * filter_responses: numpy.ndarray of shape (H,W,3F)
    '''

    # Check data type and range
    if (type(img[0, 0, 0]) != np.float32):
        img = img.astype(np.float32) / 255
    if (np.amax(img) > 1.0 or np.amin(img) < 0.0):
        img = img.astype(np.float32) / 255

    # Get the size of the image
    img_size = img.shape
    row, col, channel = img_size[0], img_size[1], img_size[2]

    # Make sure there are 3 channels (Duplicate gray-scale images)
    if len(img_size) == 2:
        img = img[:, :, np.newaxis]
        img = np.tile(img, (1, 1, 3))
    elif channel > 3:
        img = img[:, : , :3]

    # Convert image into lab color space
    lab_img = skimage.color.rgb2lab(img)

    # Set up filter scales and filter responses
```

```python
    filter_scales = opts.filter_scales
    filter_responses = np.zeros((row, col, 3*4*len(filter_scales)))

    # Update filter responses
    for s_index in range(len(filter_scales)):
        for c_index in range(3):
            filter_responses[:, :, 3*4*s_index + c_index] = scipy.ndimage.
            gaussian_filter(lab_img[:, :, c_index], filter_scales[s_index])
            filter_responses[:, :, 3*4*s_index + 3 + c_index] = scipy.ndimage.
            gaussian_laplace(lab_img[:, :, c_index], filter_scales[s_index])
            filter_responses[:, :, 3*4*s_index + 6 + c_index] = scipy.ndimage.
            gaussian_filter(lab_img[:, :, c_index], filter_scales[s_index],
            order = [0, 1])
            filter_responses[:, :, 3*4*s_index + 9 + c_index] = scipy.ndimage.
            gaussian_filter(lab_img[:, :, c_index], filter_scales[s_index],
            order = [1, 0])

    return filter_responses


def compute_dictionary_one_image(args):
    '''
    Extracts a random subset of filter responses of an image and save it to
    disk. This is a worker function called by compute_dictionary.

    Your are free to make your own interface based on how you implement
    compute_dictionary.
    '''

    # Set up the input information of args and read the image
    img_index, alpha, img_path = args
    img = io.imread(img_path)
    img = img.astype(np.float32) / 255

    # Extract the filter responses
    filter_responses = extract_filter_responses(opts, img)
    row, col, F = filter_responses.shape
    T = row * col

    # Randomly sampled the filter responses
    sampled_index = np.random.randint(T, size = alpha)
    sampled_filter_responses = filter_responses.reshape(T, F)
    sampled_filter_responses = sampled_filter_responses[sampled_index, :]

    # Save to a temporary file
    feat_dir = opts.feat_dir
    os.makedirs(feat_dir, exist_ok = True)
    np.save(join(feat_dir, f"img{img_index}.npy"), sampled_filter_responses)
```

```python


def compute_dictionary(opts, n_worker=1):
    '''
    Creates the dictionary of visual words by clustering using k-means.

    [input]
    * opts: options
    * n_worker: number of workers to process in parallel

    [saved]
    * dictionary: numpy.ndarray of shape (K,3F)
    '''

    # Set up file path and parameters
    data_dir = opts.data_dir
    feat_dir = opts.feat_dir
    out_dir = opts.out_dir
    K = opts.K
    alpha = opts.alpha
    filter_scales = opts.filter_scales
    # Set up the training files path
    train_files = open(join(data_dir, 'train_files.txt')).read().splitlines()
    img_path = [join(data_dir, img_name) for img_name in train_files]

    # Multiprocess the training data
    img_index = range(1, len(img_path)+1)
    alpha_list = [alpha] * len(img_path)
    args = zip(img_index, alpha_list, img_path)
    pool = multiprocessing.Pool(n_worker)
    pool.map(compute_dictionary_one_image, args)

    # Collect all subprocess to form the filter responses
    filter_responses = np.array([], dtype = np.float32).reshape(0,
                                            3*4*len(filter_scales))
    for img in os.listdir(feat_dir):
        subprocess_responses = np.load(join(feat_dir, img))
        filter_responses = np.append(filter_responses, subprocess_responses,
                                axis = 0)

    # Apply K-means to cluster the responses
    kmeans = KMeans(n_clusters = K).fit(filter_responses)
    dictionary = kmeans.cluster_centers_

    # Save the dictionary
    np.save(join(out_dir, 'dictionary.npy'), dictionary)


```

```python
def get_visual_words(opts, img, dictionary):
    '''
    Compute visual words mapping for the given img using the dictionary of
    visual words.

    [input]
    * opts: options
    * img: numpy.ndarray of shape (H,W) or (H,W,3)
    * dictionary: numpy.ndarray of shape (K,3F)

    [output]
    * wordmap: numpy.ndarray of shape (H,W)
    '''

    # Initialize the size of wordmap
    img_size = img.shape
    row, col = img_size[0], img_size[1]
    wordmap = np.zeros((row, col))

    # Compute every pixel of the wordmap : Slow Method (loop over row and col
    #                                                   for every pixel)
    # filter_responses = extract_filter_responses(opts, img)
    # for i in range(row):
    #     for j in range(col):
    #         pixel = np.array(filter_responses[i, j, :]).reshape(1,-1)
    #         distance = scipy.spatial.distance.cdist(pixel, dictionary,
    #                                                 metric = 'euclidean')
    #         wordmap[i, j] = np.argmin(distance)

    # Compute every pixel of the wordmap : Fast Method (reshape filter
                                                        responses)
    filter_responses = extract_filter_responses(opts, img)
    filter_responses = filter_responses.reshape((row * col), -1)
    distance = scipy.spatial.distance.cdist(filter_responses, dictionary,
                                            metric = 'euclidean')
    wordmap = np.argmin(distance, axis = 1).reshape(row, col)

    return wordmap
```

util.py

```
import numpy as np
import matplotlib.pyplot as plt
import multiprocessing


def get_num_CPU():
    '''
    Counts the number of CPUs available in the machine.
    '''
    return multiprocessing.cpu_count()


def display_filter_responses(opts, response_maps):
    '''
    Visualizes the filter response maps.

    [input]
    * response_maps: a numpy.ndarray of shape (H,W,3F).
    '''

    n_scale = len(opts.filter_scales)
    plt.figure(1)

    for i in range(n_scale * 4):
        plt.subplot(n_scale, 4, i + 1)
        resp = response_maps[:, :, i * 3:i * 3 + 3]
        resp_min = resp.min(axis=(0, 1), keepdims=True)
        resp_max = resp.max(axis=(0, 1), keepdims=True)
        resp = (resp - resp_min) / (resp_max - resp_min)
        plt.imshow(resp)
        plt.axis("off")

    plt.subplots_adjust(left=0.05, right=0.95, top=0.95,
                        bottom=0.05, wspace=0.05, hspace=0.05)
    plt.show()


def visualize_wordmap(original_image, wordmap, out_path=None):
    '''
    Visualizes the wordmap corresponding to an image.

    [input]
    * original_image: a numpy.ndarray of shape (H,W,3F).
    * wordmap: a numpy.ndarray of shape (H,W).
    * out_path (optional): Output path to save figure.

```

```
    '''
    fig = plt.figure(2, figsize=(12.8, 4.8))
    ax = fig.add_subplot(1, 2, 1)
    ax.imshow(original_image)
    plt.axis("off")
    ax = fig.add_subplot(1, 2, 2)
    ax.imshow(wordmap)
    plt.axis("off")
    plt.show()
    if out_path:
        plt.savefig(out_path, pad_inches=0)
```

custom.py

```
from os.path import join

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

import util
import visual_words
import custom_visual_recog
from opts import get_opts


def main():
    opts = get_opts()

    # Q1.2 Compute Dictionary
    n_cpu = util.get_num_CPU()
    visual_words.compute_dictionary(opts, n_worker=n_cpu)

    # Q2.1-2.4 Build the recognition system
    n_cpu = util.get_num_CPU()
    custom_visual_recog.build_recognition_system(opts, n_worker=n_cpu)

    # Q2.5 evaluate the recognition system
    n_cpu = util.get_num_CPU()
    conf, accuracy = custom_visual_recog.evaluate_recognition_system(opts,
                                    n_worker=n_cpu)

    print(f"K = {opts.K}, L = {opts.L}")
    print(conf)
    print(accuracy)
    # np.savetxt(join(opts.out_dir, 'custom_confmat.csv'),
    #            conf, fmt='%d', delimiter=',')
    # np.savetxt(join(opts.out_dir, 'custom_accuracy.txt'), [accuracy], fmt='%g')


if __name__ == '__main__':
    main()
```

custom_visual_recog.py

```
import os
import math
import multiprocessing
from os.path import join
from copy import copy

import numpy as np
from PIL import Image

import visual_words
import matplotlib.pyplot as plt


def get_feature_from_wordmap(opts, wordmap):
    '''
    Compute histogram of visual words.

    [input]
    * opts: options
    * wordmap: numpy.ndarray of shape (H,W)

    [output]
    * hist: numpy.ndarray of shape (K)
    '''

    # Set up the parameters
    K = opts.K                                          # size of bins

    # Create the histogram from wordmap and Normalize
    hist, label = np.histogram(wordmap, bins = np.arange(K+1))
    hist = hist / np.sum(hist)                          # Normalize

    # Plot the histogram to check
    # plt.bar(range(K), hist)
    # plt.title(f"K = {K}, histogram of aquarium/sun_aadolwejqiytvyne.jpg")
    # plt.show()

    return hist


def get_feature_from_wordmap_SPM(opts, wordmap):
    '''
    Compute histogram of visual words using spatial pyramid matching.

    [input]
    * opts: options
```

```
    * wordmap: numpy.ndarray of shape (H,W)

    [output]
    * hist_all: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
    '''

    # Set up the parameters and Initialize hist_all
    K = opts.K
    L = opts.L
    row, col = wordmap.shape
    hist_all = np.array([], dtype = np.float32).reshape(1, 0)

    # Spatial Pyramid Matching:Change the base of cells and weights from 2 to 3
    # Start from the finest(top) layer
    # Set up the number of cells and size of each cell
    num_cell = 2 ** L                   # increase the base to 3
    cell_row = int(row/num_cell)
    cell_col = int(col/num_cell)

    # Initialize the finest layer and its weight
    finest_layer = np.zeros((num_cell, num_cell, K))
    if (L == 0 or L == 1):
        weight = 2 ** (-L)              # increase the base to 3
    else:
        weight = 1/2                    # increase the base to 3

    # Compute the histograms of the finest layer
    for row_index in range(num_cell):
        for col_index in range(num_cell):
            small_wordmap = wordmap[cell_row*row_index : cell_row*(row_index+1),
                                    cell_col*col_index : cell_col*(col_index+1)]
            single_hist = get_feature_from_wordmap(opts, small_wordmap)
            finest_layer[row_index, col_index, :] = single_hist
    hist_all = np.append(finest_layer.reshape(1,-1)[0] * weight, hist_all)

    # Aggregate the remaining layers
    for layer_index in range(L-1, -1, -1):
        # Set up the weight of each layer
        if (layer_index == 0 or layer_index == 1):
            weight = 2 ** (-L)                      # increase the base to 3
        else:
            weight = 2 ** (layer_index - L - 1)     # increase the base to 3

        # Aggregate the remaining layers from the finest layer
        num_cell = 2 ** layer_index                 # increase the base to 3
        single_layer = np.zeros((num_cell, num_cell, K))
        for row_index in range(num_cell):
            for col_index in range(num_cell):
```

```
                single_layer[row_index, col_index, :] =
                np.sum(finest_layer[row_index*2 : (row_index+1)*2, col_index*2
                                    : (col_index+1)*2, :], axis = (0, 1))
        hist_all = np.append(single_layer.reshape(1,-1)[0] * weight, hist_all)

    # Normalization
    hist_all = hist_all / np.sum(hist_all)

    # Plot the histogram_all to check
    # plt.bar(range(hist_all.shape[0]), hist_all)
    # plt.title(f"K = {K}, L ={L}, size = {hist_all.shape[0]}, histogram_all of
    #               aquarium/sun_aadolwejqiytvyne.jpg")
    # plt.show()

    return hist_all


def get_image_feature(opts, img_path, dictionary):
    '''
    Extracts the spatial pyramid matching feature.

    [input]
    * opts: options
    * img_path: path of image file to read
    * dictionary: numpy.ndarray of shape (K, 3F)


    [output]
    * feature: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
    '''

    # Load the image and check the data type and dimensions
    img = Image.open(img_path)
    img = np.array(img).astype(np.float32) / 255
    if len(img.shape) == 2:
        img = img[:, :, np.newaxis]
        img = np.tile(img, (1, 1, 3))

    # Extract the wordmap from the image (use dictionary)
    wordmap = visual_words.get_visual_words(opts, img, dictionary)

    # Compute the Spatial Pyramid Matching features (use wordmap)
    feature = get_feature_from_wordmap_SPM(opts, wordmap)

    # Plot the feature to check
    # plt.bar(range(feature.shape[0]), feature)
    # plt.title(f"size = {feature.shape[0]}, SPM feature of
    #               aquarium/sun_aadolwejqiytvyne.jpg")
```

```
    # plt.show()

    return feature


def build_recognition_system(opts, n_worker=1):
    '''
    Creates a trained recognition system by generating training features from
    all training images.

    [input]
    * opts: options
    * n_worker: number of workers to process in parallel

    [saved]
    * features: numpy.ndarray of shape (N,M)
    * labels: numpy.ndarray of shape (N)
    * dictionary: numpy.ndarray of shape (K,3F)
    * SPM_layer_num: number of spatial pyramid layers
    '''

    # Set up the file path and load the training files
    data_dir = opts.data_dir
    out_dir = opts.out_dir
    SPM_layer_num = opts.L

    # Load the trainig files and labels
    train_files = open(join(data_dir, 'train_files.txt')).read().splitlines()
    training_img_num = len(train_files)
    train_labels = np.loadtxt(join(data_dir, 'train_labels.txt'), np.int32)
    dictionary = np.load(join(out_dir, 'dictionary.npy'))

    # Multiprocessing to extract the traing features
    opts_list = [opts] * training_img_num
    img_path = [join(data_dir, img_name) for img_name in train_files]
    dictionary_list = [dictionary] * training_img_num
    args = zip(opts_list, img_path, dictionary_list)
    pool = multiprocessing.Pool(n_worker)
    features = pool.starmap(get_image_feature, args)

    # example code snippet to save the learned system
    np.savez_compressed(join(out_dir, 'custom_trained_system.npz'), features =
    features, labels = train_labels, dictionary = dictionary,
    SPM_layer_num = SPM_layer_num)


# Use Intersection Similarity, Euclidean Distance, Kolmogorov-Smirnov Distance,
# and Match Distance as similarity scores
```

```python
def distance_to_set(word_hist, histograms):
    '''
    Compute distance between a histogram of visual words with all training
    image histograms.

    [input]
    * word_hist: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
    * histograms: numpy.ndarray of shape (T,K*(4^(L+1)-1)/3)

    [output]
    * dis: numpy.ndarray of shape (T)
    '''

    # Inrtersection Similarity
    # Compute the intersection similarity bectween word_hist and histograms
    # num_features, concantenated_size = histograms.shape
    # intersection_similarity = np.minimum(word_hist, histograms)
    # Compute the distance (inverse of the intersection similarity)
    # dis = np.full((num_features), 1) - np.sum(intersection_similarity, axis =
    1)

    # Euclidean Distance
    # Compute the L2 norm bectween word_hist and histograms
    T, K = histograms.shape
    word_hist_all = np.tile(word_hist, (T,1))
    similarity = np.square(word_hist_all - histograms)
    # Compute the distance
    dis = np.sum(similarity, axis = 1)
    dis = np.sqrt(dis)

    # Kolmogorov-Smirnov Divergance
    # Compute the difference bectween word_hist and histograms
    # T, K = histograms.shape
    # word_hist_all = np.tile(word_hist, (T,1))
    # diff = abs(word_hist_all - histograms)
    # Compute the distance
    # dis = np.amax(diff, axis = 1)

    # Match Distance
    # Compute the difference bectween word_hist and histograms
    # T, K = histograms.shape
    # word_hist_all = np.tile(word_hist, (T,1))
    # diff = abs(word_hist_all - histograms)
    # Compute the distance
    # dis = np.sum(diff, axis = 1)

    return dis

```

```python

def evaluate_recognition_system(opts, n_worker=1):
    '''
    Evaluates the recognition system for all test images and returns the
    confusion matrix.

    [input]
    * opts: options
    * n_worker: number of workers to process in parallel

    [output]
    * conf: numpy.ndarray of shape (8,8)
    * accuracy: accuracy of the evaluated system
    '''

    # Set up file path and Load traind data
    data_dir = opts.data_dir
    out_dir = opts.out_dir
    trained_system = np.load(join(out_dir, 'custom_trained_system.npz'))
    dictionary = trained_system['dictionary']
    trained_features = trained_system['features']
    trained_labels = trained_system['labels']

    # Use the stored options in the trained system instead of opts.py
    test_opts = copy(opts)
    test_opts.K = dictionary.shape[0]
    test_opts.L = trained_system['SPM_layer_num']

    # Load the test data
    test_files = open(join(data_dir, 'test_files.txt')).read().splitlines()
    test_img_num = len(test_files)
    test_labels = np.loadtxt(join(data_dir, 'test_labels.txt'), np.int32)

    # Extract the features from test data
    opts_list = [opts] * test_img_num
    img_path = [join(data_dir, img_name) for img_name in test_files]
    dictionary_list = [dictionary] * test_img_num
    args = zip(opts_list, img_path, dictionary_list)
    pool = multiprocessing.Pool(n_worker)
    test_features = np.asarray(pool.starmap(get_image_feature, args))
    # np.savez_compressed(join(out_dir, 'custom_test_system.npz'), features =
    #                      test_features)

    # Compute the predicted labels
    pred_labels = []
    for test_index in range(test_img_num):
        pred_index = np.argmin(distance_to_set(test_features[test_index, :],
                               trained_features))
```

```
            pred_labels.append(trained_labels[pred_index])
        pred_labels = np.asarray(pred_labels)

        # Compute the Confusion Matrix and Accuracy
        confusion_matrix = np.zeros((8, 8))
        for true_index, pred_index in zip(test_labels, pred_labels):
            confusion_matrix[true_index][pred_index] += 1
        accuracy = np.sum(np.diag(confusion_matrix)) / np.sum(confusion_matrix)

        return confusion_matrix, accuracy
```