

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

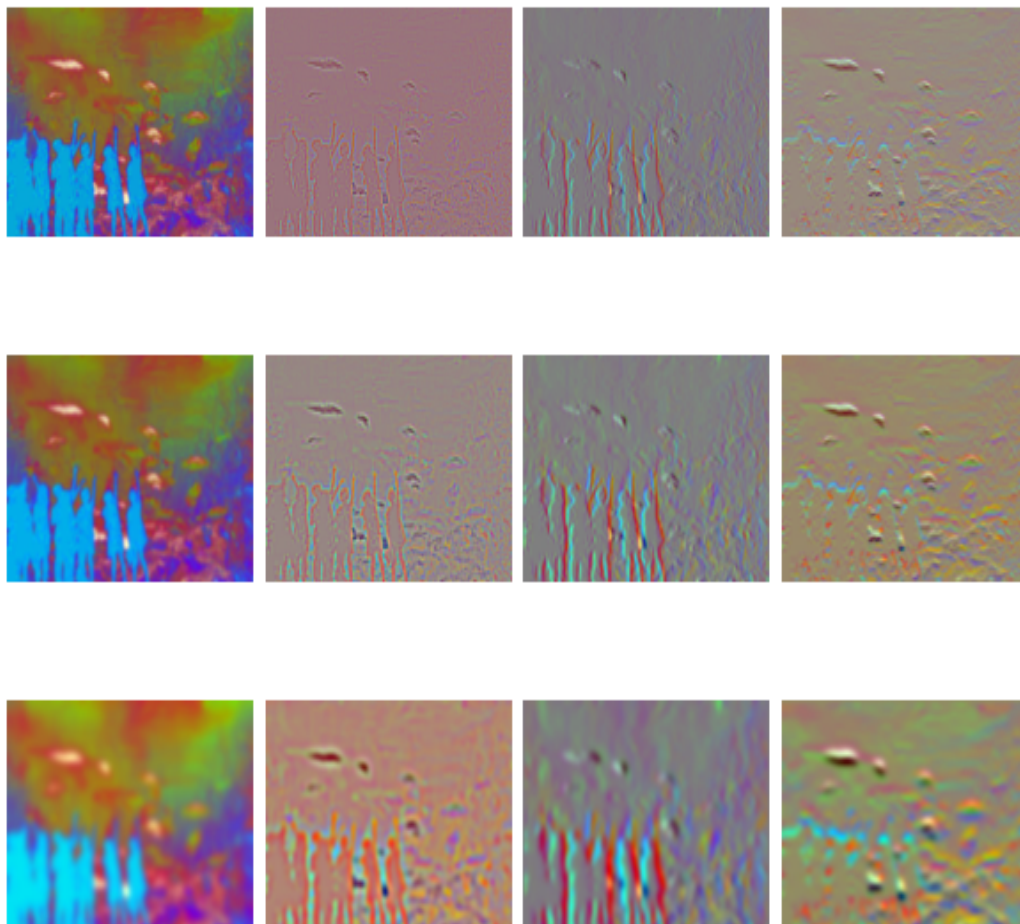


Figure 1: Filter responses of each filter on aquarium/sun_aztvjgubyrgrup.jpg

Q1.3



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

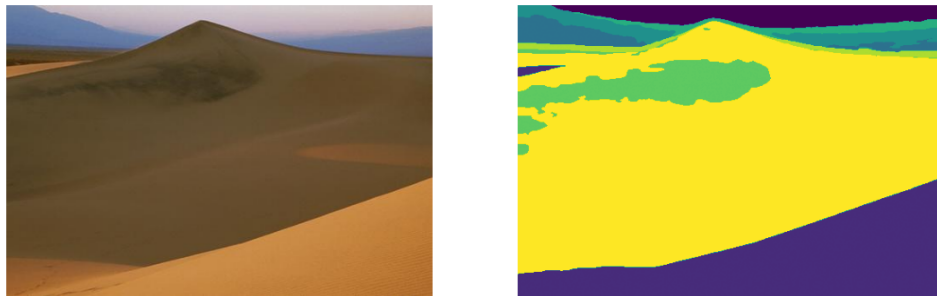


Figure 3: Original image and wordmaps of desert/sun_aaqyzvrweabdxjzo.jpg

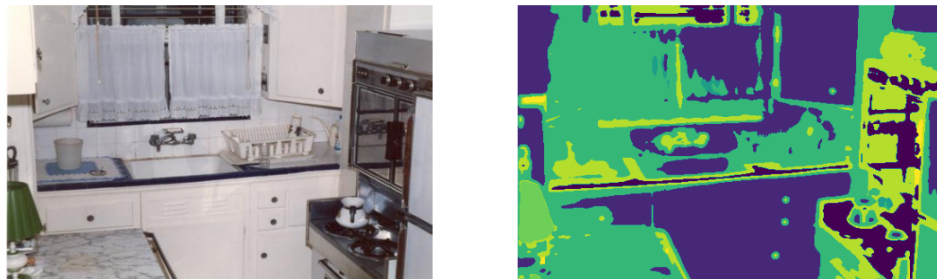


Figure 4: Original image and wordmaps of kitchen/sun_aaejbjeispxohmfv.jpg

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 $3^l \times 3^l$) 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

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

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

$$\max(|h_1(i) - h_2(i)|) \quad (2)$$

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

$$\sum_i |h_1(i) - h_2(i)| \quad (3)$$

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.

¹Website (<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

```

1  from os.path import join
2
3  import numpy as np
4  from PIL import Image
5  import matplotlib.pyplot as plt
6
7  import util
8  import visual_words
9  import visual_recog
10 from opts import get_opts
11
12
13 def main():
14     opts = get_opts()
15
16     # Q1.1
17     img_path = join(opts.data_dir, 'kitchen/sun_aasmevtpkslccptd.jpg')
18     img = Image.open(img_path)
19     img = np.array(img).astype(np.float32) / 255
20     filter_responses = visual_words.extract_filter_responses(opts, img)
21     util.display_filter_responses(opts, filter_responses)
22
23     # Q1.2
24     # n_cpu = util.get_num_CPU()
25     # visual_words.compute_dictionary(opts, n_worker=n_cpu)
26
27     # Q1.3
28     # Visualize wordmaps for three images (aquarium, desert, kitchen)
29     # img_path = join(opts.data_dir, 'aquarium/sun_aadolwejqiytyvne.jpg')
30     # img_path = join(opts.data_dir, 'desert/sun_aaqyzvrweabdæjzo.jpg')
31     # img_path = join(opts.data_dir, 'kitchen/sun_aaebjpeispæohmfv.jpg')
32     # img = Image.open(img_path)
33     # img = np.array(img).astype(np.float32) / 255
34     # dictionary = np.load(join(opts.out_dir, 'dictionary.npy'))
35     # wordmap = visual_words.get_visual_words(opts, img, dictionary)
36     # util.visualize_wordmap(img, wordmap)
37
38     # Check Q2.1, Q2.2, Q2.4
39     # Q2.1 Test histogram
40     # hist = visual_recog.get_feature_from_wordmap(opts, wordmap)
41     # Q2.2 Test histogram_all
42     # hist_all = visual_recog.get_feature_from_wordmap_SPM(opts, wordmap)
43     # Q2.4 Test get_image_feature
44     # feature = visual_recog.get_image_feature(opts, img_path, dictionary)

```



```
45
46     # Q2.1-2.4
47     # n_cpu = util.get_num_CPU()
48     # visual_recog.build_recognition_system(opts, n_worker=n_cpu)
49
50     # Q2.5
51     # n_cpu = util.get_num_CPU()
52     # conf, accuracy = visual_recog.evaluate_recognition_system(
53     #     opts, n_worker=n_cpu)
54
55     # print(conf)
56     # print(accuracy)
57     # np.savetxt(join(opts.out_dir, 'confmat.csv'),
58     #             conf, fmt='%d', delimiter=',')
59     # np.savetxt(join(opts.out_dir, 'accuracy.txt'), [accuracy], fmt='%g')
60
61
62 if __name__ == '__main__':
63     main()
```

opts.py

```

1  '''
2  Hyperparameters wrapped in argparse
3  This file contains most of tunable parameters for this homework
4  You are asked to play around with them for Q3.1
5  It is recommended that you leave them as they are before getting to Q3.1
6
7  You can change the values by changing their default fields or by command-line
8  arguments. For example, "python main.py --filter-scales 2 5 --K 50"
9  '''
10
11 import argparse
12
13
14 def get_opts():
15     parser = argparse.ArgumentParser(
16         description='16-720 HW1: Scene Recognition')
17
18     # Paths
19     parser.add_argument('--data-dir', type=str, default='../data',
20                         help='data folder')
21     parser.add_argument('--feat-dir', type=str, default='../feat',
22                         help='feature folder')
23     parser.add_argument('--out-dir', type=str, default='.',
24                         help='output folder')
25
26     # Visual words (requires tuning)
27     parser.add_argument('--filter-scales', nargs='+', type=float,
28                         default=[1, 2],
29                         help='a list of scales for all the filters')
30     parser.add_argument('--K', type=int, default=10,
31                         help='# of words')
32     parser.add_argument(
33         '--alpha', type=int, default=25,
34         help='Using only a subset of alpha pixels in each image'
35     )
36
37     # Recognition system (requires tuning)
38     parser.add_argument('--L', type=int, default=1,
39                         help='# of layers in spatial pyramid matching (SPM) = L+1')
40
41     # Additional options (add your own hyperparameters here)
42
43     ##
44     opts = parser.parse_args()
45     return opts

```

visual_recog.py

```

1  import os
2  import math
3  import multiprocessing
4  from os.path import join
5  from copy import copy
6
7  import numpy as np
8  from PIL import Image
9
10 import visual_words
11 import matplotlib.pyplot as plt
12
13
14 def get_feature_from_wordmap(opts, wordmap):
15     '''
16     Compute histogram of visual words.
17
18     [input]
19     * opts: options
20     * wordmap: numpy.ndarray of shape (H,W)
21
22     [output]
23     * hist: numpy.ndarray of shape (K)
24     '''
25
26     # Set up the parameters
27     K = opts.K # size of bins
28
29     # Create the histogram from wordmap and Normalize
30     hist, label = np.histogram(wordmap, bins = np.arange(K+1))
31     hist = hist / np.sum(hist) # Normalize
32
33     # Plot the histogram to check
34     # plt.bar(range(K), hist)
35     # plt.title(f"K = {K}, histogram of aquarium/sun_aadolwejqiyytvyne.jpg")
36     # plt.show()
37
38     return hist
39
40
41 def get_feature_from_wordmap_SPM(opts, wordmap):
42     '''
43     Compute histogram of visual words using spatial pyramid matching.
44
45     [input]
46     * opts: options

```

```

47  * wordmap: numpy.ndarray of shape (H,W)
48
49  [output]
50  * hist_all: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
51  '''
52
53  # Set up the parameters and Initialize hist_all
54  K = opts.K
55  L = opts.L
56  row, col = wordmap.shape
57  hist_all = np.array([], dtype = np.float32).reshape(1, 0)
58
59  # Spatial Pyramid Matching : Slow Method (loop over each layers)
60  # for layer_index in range(L):
61  #     # Set up the weight of each layer
62  #     if (layer_index == 0 or layer_index == 1):
63  #         weight = 2 ** (-L)
64  #     else:
65  #         weight = 2 ** (L - layer_index - 1)
66
67  #     # Chop the image into cells
68  #     num_cell = 2 ** layer_index
69  #     cell_row = int(row/num_cell)
70  #     cell_col = int(col/num_cell)
71
72  #     # Concatenate all histograms
73  #     for row_index in range(num_cell):
74  #         for col_index in range(num_cell):
75  #             small_wordmap = wordmap[cell_row*row_index :
76  #                                     cell_row*(row_index+1), cell_col*col_index :
77  #                                     cell_col*(col_index+1)]
78  #             single_hist = get_feature_from_wordmap(opts, small_wordmap)
79  #             hist_all = np.append(hist_all, single_hist * weight)
80
81  # Spatial Pyramid Matching : Fast Method
82  # Start from the finest(top) layer
83  # Set up the number of cells and size of each cell
84  num_cell = 2 ** L
85  cell_row = int(row/num_cell)
86  cell_col = int(col/num_cell)
87
88  # Initialize the finest layer and its weight
89  finest_layer = np.zeros((num_cell, num_cell, K))
90  if (L == 0 or L == 1):
91  #     weight = 2 ** (-L)
92  else:
93  #     weight = 1/2
94

```

```

95     # Compute the histograms of the finest layer
96     for row_index in range(num_cell):
97         for col_index in range(num_cell):
98             small_wordmap = wordmap[cell_row*row_index : cell_row*(row_index+1),
99                                     cell_col*col_index : cell_col*(col_index+1)]
100             single_hist = get_feature_from_wordmap(opts, small_wordmap)
101             finest_layer[row_index, col_index, :] = single_hist
102     hist_all = np.append(finest_layer.reshape(1,-1)[0] * weight, hist_all)
103
104     # Aggregate the remaining layers
105     for layer_index in range(L-1, -1, -1):
106         # Set up the weight of each layer
107         if (layer_index == 0 or layer_index == 1):
108             weight = 2 ** (-L)
109         else:
110             weight = 2 ** (layer_index - L - 1)
111
112         # Aggregate the remaining layers from the finest layer
113         num_cell = 2 ** layer_index
114         single_layer = np.zeros((num_cell, num_cell, K))
115         for row_index in range(num_cell):
116             for col_index in range(num_cell):
117                 single_layer[row_index, col_index, :] =
118                     np.sum(finest_layer[row_index*2 : (row_index+1)*2, col_index*2
119                                     : (col_index+1)*2, :], axis = (0, 1))
120             hist_all = np.append(single_layer.reshape(1,-1)[0] * weight, hist_all)
121
122     # Normalization
123     hist_all = hist_all / np.sum(hist_all)
124
125     # Plot the histogram_all to check
126     # plt.bar(range(hist_all.shape[0]), hist_all)
127     # plt.title(f"K = {K}, L = {L}, size = {hist_all.shape[0]}, histogram_all of
128     #               aquarium/sun_aadolwejqiyytvyne.jpg")
129     # plt.show()
130
131     return hist_all
132
133
134 def get_image_feature(opts, img_path, dictionary):
135     """
136     Extracts the spatial pyramid matching feature.
137
138     [input]
139     * opts: options
140     * img_path: path of image file to read
141     * dictionary: numpy.ndarray of shape (K, 3F)
142 
```

```

143
144     [output]
145     * feature: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
146     '''
147
148     # Load the image and check the data type and dimensions
149     img = Image.open(img_path)
150     img = np.array(img).astype(np.float32) / 255
151     if len(img.shape) == 2:
152         img = img[:, :, np.newaxis]
153         img = np.tile(img, (1, 1, 3))
154
155     # Extract the wordmap from the image (use dictionary)
156     wordmap = visual_words.get_visual_words(opts, img, dictionary)
157
158     # Compute the Spatial Pyramid Matching features (use wordmap)
159     feature = get_feature_from_wordmap_SPM(opts, wordmap)
160
161     # Plot the feature to check
162     # plt.bar(range(feature.shape[0]), feature)
163     # plt.title(f"size = {feature.shape[0]}, SPM feature of
164     #             aquarium/sun_aadolwejqiyytvyne.jpg")
165     # plt.show()
166
167     return feature
168
169
170 def build_recognition_system(opts, n_worker=1):
171     '''
172     Creates a trained recognition system by generating training features from
173     all training images.
174
175     [input]
176     * opts: options
177     * n_worker: number of workers to process in parallel
178
179     [saved]
180     * features: numpy.ndarray of shape (N,M)
181     * labels: numpy.ndarray of shape (N)
182     * dictionary: numpy.ndarray of shape (K,3F)
183     * SPM_layer_num: number of spatial pyramid layers
184     '''
185
186     # Set up the file path and load the training files
187     data_dir = opts.data_dir
188     out_dir = opts.out_dir
189     SPM_layer_num = opts.L
190

```

```

191     # Load the training files and labels
192     train_files = open(join(data_dir, 'train_files.txt')).read().splitlines()
193     training_img_num = len(train_files)
194     train_labels = np.loadtxt(join(data_dir, 'train_labels.txt'), np.int32)
195     dictionary = np.load(join(out_dir, 'dictionary.npy'))
196
197     # Multiprocessing to extract the training features
198     opts_list = [opts] * training_img_num
199     img_path = [join(data_dir, img_name) for img_name in train_files]
200     dictionary_list = [dictionary] * training_img_num
201     args = zip(opts_list, img_path, dictionary_list)
202     pool = multiprocessing.Pool(n_worker)
203     features = pool.starmap(get_image_feature, args)
204
205     # example code snippet to save the learned system
206     np.savez_compressed(join(out_dir, 'trained_system.npz'), features =
207     features, labels = train_labels, dictionary = dictionary,
208     SPM_layer_num = SPM_layer_num)
209
210
211 def distance_to_set(word_hist, histograms):
212     '''
213     Compute distance between a histogram of visual words with all training
214     image histograms.
215
216     [input]
217     * word_hist: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
218     * histograms: numpy.ndarray of shape (T,K*(4^(L+1)-1)/3)
219
220     [output]
221     * dis: numpy.ndarray of shape (T)
222     '''
223
224     # Compute the intersection similarity between word_hist and histograms
225     num_features, concatenated_size = histograms.shape
226     intersection_similarity = np.minimum(word_hist, histograms)
227
228     # Compute the distance (inverse of the intersection similarity)
229     dis = np.full((num_features), 1) - np.sum(intersection_similarity, axis = 1)
230
231     return dis
232
233
234 def evaluate_recognition_system(opts, n_worker=1):
235     '''
236     Evaluates the recognition system for all test images and returns the
237     confusion matrix.
238

```

```

239     [input]
240     * opts: options
241     * n_worker: number of workers to process in parallel
242
243     [output]
244     * conf: numpy.ndarray of shape (8,8)
245     * accuracy: accuracy of the evaluated system
246     '''
247
248     # Set up file path and Load traind data
249     data_dir = opts.data_dir
250     out_dir = opts.out_dir
251     trained_system = np.load(join(out_dir, 'trained_system.npz'))
252     dictionary = trained_system['dictionary']
253     trained_features = trained_system['features']
254     trained_labels = trained_system['labels']
255
256     # Use the stored options in the trained system instead of opts.py
257     test_opts = copy(opts)
258     test_opts.K = dictionary.shape[0]
259     test_opts.L = trained_system['SPM_layer_num']
260
261     # Load the test data
262     test_files = open(join(data_dir, 'test_files.txt')).read().splitlines()
263     test_img_num = len(test_files)
264     test_labels = np.loadtxt(join(data_dir, 'test_labels.txt'), np.int32)
265
266     # Extract the features from test data
267     opts_list = [opts] * test_img_num
268     img_path = [join(data_dir, img_name) for img_name in test_files]
269     dictionary_list = [dictionary] * test_img_num
270     args = zip(opts_list, img_path, dictionary_list)
271     pool = multiprocessing.Pool(n_worker)
272     test_features = np.asarray(pool.starmap(get_image_feature, args))
273     # np.savez_compressed(join(out_dir, 'test_system.npz'), features =
274     test_features)
275
276     # Compute the predicted labels
277     pred_labels = []
278     for test_index in range(test_img_num):
279         pred_index = np.argmin(distance_to_set(test_features[test_index, :],
280                                             trained_features))
281         pred_labels.append(trained_labels[pred_index])
282     pred_labels = np.asarray(pred_labels)
283
284     # Compute the Confusion Matrix and Accuracy
285     confusion_matrix = np.zeros((8, 8))
286     for true_index, pred_index in zip(test_labels, pred_labels):

```



```
287         confusion_matrix[true_index][pred_index] += 1
288     accuracy = np.sum(np.diag(confusion_matrix)) / np.sum(confusion_matrix)
289
290     return confusion_matrix, accuracy
```

visual_words.py

```

1  import os
2  import multiprocessing
3  from os.path import join, isfile
4
5  import numpy as np
6  import scipy.ndimage
7  import skimage.color
8  from skimage import io
9  from PIL import Image
10 from sklearn.cluster import KMeans
11
12 from opts import get_opts
13 opts = get_opts()
14
15 def extract_filter_responses(opts, img):
16     '''
17     Extracts the filter responses for the given image.
18
19     [input]
20     * opts: options
21     * img: numpy.ndarray of shape (H,W) or (H,W,3)
22     [output]
23     * filter_responses: numpy.ndarray of shape (H,W,3F)
24     '''
25
26     # Check data type and range
27     if (type(img[0, 0, 0]) != np.float32):
28         img = img.astype(np.float32) / 255
29     if (np.amax(img) > 1.0 or np.amin(img) < 0.0):
30         img = img.astype(np.float32) / 255
31
32     # Get the size of the image
33     img_size = img.shape
34     row, col, channel = img_size[0], img_size[1], img_size[2]
35
36     # Make sure there are 3 channels (Duplicate gray-scale images)
37     if len(img_size) == 2:
38         img = img[:, :, np.newaxis]
39         img = np.tile(img, (1, 1, 3))
40     elif channel > 3:
41         img = img[:, :, :3]
42
43     # Convert image into lab color space
44     lab_img = skimage.color.rgb2lab(img)
45
46     # Set up filter scales and filter responses

```

```

47     filter_scales = opts.filter_scales
48     filter_responses = np.zeros((row, col, 3*4*len(filter_scales)))
49
50     # Update filter responses
51     for s_index in range(len(filter_scales)):
52         for c_index in range(3):
53             filter_responses[:, :, 3*4*s_index + c_index] = scipy.ndimage.
54             gaussian_filter(lab_img[:, :, c_index], filter_scales[s_index])
55             filter_responses[:, :, 3*4*s_index + 3 + c_index] = scipy.ndimage.
56             gaussian_laplace(lab_img[:, :, c_index], filter_scales[s_index])
57             filter_responses[:, :, 3*4*s_index + 6 + c_index] = scipy.ndimage.
58             gaussian_filter(lab_img[:, :, c_index], filter_scales[s_index],
59             order = [0, 1])
60             filter_responses[:, :, 3*4*s_index + 9 + c_index] = scipy.ndimage.
61             gaussian_filter(lab_img[:, :, c_index], filter_scales[s_index],
62             order = [1, 0])
63
64     return filter_responses
65
66
67 def compute_dictionary_one_image(args):
68     '''
69     Extracts a random subset of filter responses of an image and save it to
70     disk. This is a worker function called by compute_dictionary.
71
72     You are free to make your own interface based on how you implement
73     compute_dictionary.
74     '''
75
76     # Set up the input information of args and read the image
77     img_index, alpha, img_path = args
78     img = io.imread(img_path)
79     img = img.astype(np.float32) / 255
80
81     # Extract the filter responses
82     filter_responses = extract_filter_responses(opts, img)
83     row, col, F = filter_responses.shape
84     T = row * col
85
86     # Randomly sampled the filter responses
87     sampled_index = np.random.randint(T, size = alpha)
88     sampled_filter_responses = filter_responses.reshape(T, F)
89     sampled_filter_responses = sampled_filter_responses[sampled_index, :]
90
91     # Save to a temporary file
92     feat_dir = opts.feat_dir
93     os.makedirs(feat_dir, exist_ok = True)
94     np.save(join(feat_dir, f"img{img_index}.npy"), sampled_filter_responses)

```

```

95
96
97 def compute_dictionary(opts, n_worker=1):
98     '''
99     Creates the dictionary of visual words by clustering using k-means.
100
101     [input]
102     * opts: options
103     * n_worker: number of workers to process in parallel
104
105     [saved]
106     * dictionary: numpy.ndarray of shape (K,3F)
107     '''
108
109     # Set up file path and parameters
110     data_dir = opts.data_dir
111     feat_dir = opts.feat_dir
112     out_dir = opts.out_dir
113     K = opts.K
114     alpha = opts.alpha
115     filter_scales = opts.filter_scales
116     # Set up the training files path
117     train_files = open(join(data_dir, 'train_files.txt')).read().splitlines()
118     img_path = [join(data_dir, img_name) for img_name in train_files]
119
120     # Multiprocess the training data
121     img_index = range(1, len(img_path)+1)
122     alpha_list = [alpha] * len(img_path)
123     args = zip(img_index, alpha_list, img_path)
124     pool = multiprocessing.Pool(n_worker)
125     pool.map(compute_dictionary_one_image, args)
126
127     # Collect all subprocess to form the filter responses
128     filter_responses = np.array([], dtype = np.float32).reshape(0,
129                                     3*4*len(filter_scales))
130
131     for img in os.listdir(feat_dir):
132         subprocess_responses = np.load(join(feat_dir, img))
133         filter_responses = np.append(filter_responses, subprocess_responses,
134                                     axis = 0)
135
136     # Apply K-means to cluster the responses
137     kmeans = KMeans(n_clusters = K).fit(filter_responses)
138     dictionary = kmeans.cluster_centers_
139
140     # Save the dictionary
141     np.save(join(out_dir, 'dictionary.npy'), dictionary)
142

```

```

143 def get_visual_words(opts, img, dictionary):
144     '''
145     Compute visual words mapping for the given img using the dictionary of
146     visual words.
147
148     [input]
149     * opts: options
150     * img: numpy.ndarray of shape (H,W) or (H,W,3)
151     * dictionary: numpy.ndarray of shape (K,3F)
152
153     [output]
154     * wordmap: numpy.ndarray of shape (H,W)
155     '''
156
157     # Initialize the size of wordmap
158     img_size = img.shape
159     row, col = img_size[0], img_size[1]
160     wordmap = np.zeros((row, col))
161
162     # Compute every pixel of the wordmap : Slow Method (loop over row and col
163     #                                           for every pixel)
164     # filter_responses = extract_filter_responses(opts, img)
165     # for i in range(row):
166     #     for j in range(col):
167     #         pixel = np.array(filter_responses[i, j, :]).reshape(1,-1)
168     #         distance = scipy.spatial.distance.cdist(pixel, dictionary,
169     #                                           metric = 'euclidean')
170     #         wordmap[i, j] = np.argmin(distance)
171
172     # Compute every pixel of the wordmap : Fast Method (reshape filter
173     #                                           responses)
174     filter_responses = extract_filter_responses(opts, img)
175     filter_responses = filter_responses.reshape((row * col), -1)
176     distance = scipy.spatial.distance.cdist(filter_responses, dictionary,
177                                           metric = 'euclidean')
178     wordmap = np.argmin(distance, axis = 1).reshape(row, col)
179
180     return wordmap

```

util.py

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  import multiprocessing
4
5
6  def get_num_CPU():
7      '''
8      Counts the number of CPUs available in the machine.
9      '''
10     return multiprocessing.cpu_count()
11
12
13  def display_filter_responses(opts, response_maps):
14      '''
15      Visualizes the filter response maps.
16
17      [input]
18      * response_maps: a numpy.ndarray of shape (H,W,3F).
19      '''
20
21     n_scale = len(opts.filter_scales)
22     plt.figure(1)
23
24     for i in range(n_scale * 4):
25         plt.subplot(n_scale, 4, i + 1)
26         resp = response_maps[:, :, i * 3:i * 3 + 3]
27         resp_min = resp.min(axis=(0, 1), keepdims=True)
28         resp_max = resp.max(axis=(0, 1), keepdims=True)
29         resp = (resp - resp_min) / (resp_max - resp_min)
30         plt.imshow(resp)
31         plt.axis("off")
32
33     plt.subplots_adjust(left=0.05, right=0.95, top=0.95,
34                        bottom=0.05, wspace=0.05, hspace=0.05)
35     plt.show()
36
37
38  def visualize_wordmap(original_image, wordmap, out_path=None):
39      '''
40      Visualizes the wordmap corresponding to an image.
41
42      [input]
43      * original_image: a numpy.ndarray of shape (H,W,3F).
44      * wordmap: a numpy.ndarray of shape (H,W).
45      * out_path (optional): Output path to save figure.
46

```

```
47     '''
48     fig = plt.figure(2, figsize=(12.8, 4.8))
49     ax = fig.add_subplot(1, 2, 1)
50     ax.imshow(original_image)
51     plt.axis("off")
52     ax = fig.add_subplot(1, 2, 2)
53     ax.imshow(wordmap)
54     plt.axis("off")
55     plt.show()
56     if out_path:
57         plt.savefig(out_path, pad_inches=0)
```

custom.py

```
1  from os.path import join
2
3  import numpy as np
4  from PIL import Image
5  import matplotlib.pyplot as plt
6
7  import util
8  import visual_words
9  import custom_visual_recog
10 from opts import get_opts
11
12
13 def main():
14     opts = get_opts()
15
16     # Q1.2 Compute Dictionary
17     n_cpu = util.get_num_CPU()
18     visual_words.compute_dictionary(opts, n_worker=n_cpu)
19
20     # Q2.1-2.4 Build the recognition system
21     n_cpu = util.get_num_CPU()
22     custom_visual_recog.build_recognition_system(opts, n_worker=n_cpu)
23
24     # Q2.5 evaluate the recognition system
25     n_cpu = util.get_num_CPU()
26     conf, accuracy = custom_visual_recog.evaluate_recognition_system(opts,
27                                                                    n_worker=n_cpu)
28
29     print(f"K = {opts.K}, L = {opts.L}")
30     print(conf)
31     print(accuracy)
32     # np.savetxt(join(opts.out_dir, 'custom_confmat.csv'),
33     #             conf, fmt='%d', delimiter=',')
34     # np.savetxt(join(opts.out_dir, 'custom_accuracy.txt'), [accuracy], fmt='%g')
35
36
37 if __name__ == '__main__':
38     main()
```


custom_visual_recog.py

```

1  import os
2  import math
3  import multiprocessing
4  from os.path import join
5  from copy import copy
6
7  import numpy as np
8  from PIL import Image
9
10 import visual_words
11 import matplotlib.pyplot as plt
12
13
14 def get_feature_from_wordmap(opts, wordmap):
15     '''
16     Compute histogram of visual words.
17
18     [input]
19     * opts: options
20     * wordmap: numpy.ndarray of shape (H,W)
21
22     [output]
23     * hist: numpy.ndarray of shape (K)
24     '''
25
26     # Set up the parameters
27     K = opts.K # size of bins
28
29     # Create the histogram from wordmap and Normalize
30     hist, label = np.histogram(wordmap, bins = np.arange(K+1))
31     hist = hist / np.sum(hist) # Normalize
32
33     # Plot the histogram to check
34     # plt.bar(range(K), hist)
35     # plt.title(f"K = {K}, histogram of aquarium/sun_aadolwejqiyytvyne.jpg")
36     # plt.show()
37
38     return hist
39
40
41 def get_feature_from_wordmap_SPM(opts, wordmap):
42     '''
43     Compute histogram of visual words using spatial pyramid matching.
44
45     [input]
46     * opts: options

```

```

47  * wordmap: numpy.ndarray of shape (H,W)
48
49  [output]
50  * hist_all: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
51  '''
52
53  # Set up the parameters and Initialize hist_all
54  K = opts.K
55  L = opts.L
56  row, col = wordmap.shape
57  hist_all = np.array([], dtype = np.float32).reshape(1, 0)
58
59  # Spatial Pyramid Matching: Change the base of cells and weights from 2 to 3
60  # Start from the finest(top) layer
61  # Set up the number of cells and size of each cell
62  num_cell = 2 ** L                # increase the base to 3
63  cell_row = int(row/num_cell)
64  cell_col = int(col/num_cell)
65
66  # Initialize the finest layer and its weight
67  finest_layer = np.zeros((num_cell, num_cell, K))
68  if (L == 0 or L == 1):
69      weight = 2 ** (-L)           # increase the base to 3
70  else:
71      weight = 1/2                 # increase the base to 3
72
73  # Compute the histograms of the finest layer
74  for row_index in range(num_cell):
75      for col_index in range(num_cell):
76          small_wordmap = wordmap[cell_row*row_index : cell_row*(row_index+1),
77                                   cell_col*col_index : cell_col*(col_index+1)]
78          single_hist = get_feature_from_wordmap(opts, small_wordmap)
79          finest_layer[row_index, col_index, :] = single_hist
80  hist_all = np.append(finest_layer.reshape(1,-1)[0] * weight, hist_all)
81
82  # Aggregate the remaining layers
83  for layer_index in range(L-1, -1, -1):
84      # Set up the weight of each layer
85      if (layer_index == 0 or layer_index == 1):
86          weight = 2 ** (-L)       # increase the base to 3
87      else:
88          weight = 2 ** (layer_index - L - 1) # increase the base to 3
89
90  # Aggregate the remaining layers from the finest layer
91  num_cell = 2 ** layer_index     # increase the base to 3
92  single_layer = np.zeros((num_cell, num_cell, K))
93  for row_index in range(num_cell):
94      for col_index in range(num_cell):

```

```

95         single_layer[row_index, col_index, :] =
96         np.sum(finest_layer[row_index*2 : (row_index+1)*2, col_index*2
97             : (col_index+1)*2, :], axis = (0, 1))
98         hist_all = np.append(single_layer.reshape(1,-1)[0] * weight, hist_all)
99
100     # Normalization
101     hist_all = hist_all / np.sum(hist_all)
102
103     # Plot the histogram_all to check
104     # plt.bar(range(hist_all.shape[0]), hist_all)
105     # plt.title(f"K = {K}, L = {L}, size = {hist_all.shape[0]}, histogram_all of
106     #         aquarium/sun_aadolwejqiyytvyne.jpg")
107     # plt.show()
108
109     return hist_all
110
111
112 def get_image_feature(opts, img_path, dictionary):
113     '''
114     Extracts the spatial pyramid matching feature.
115
116     [input]
117     * opts: options
118     * img_path: path of image file to read
119     * dictionary: numpy.ndarray of shape (K, 3F)
120
121
122     [output]
123     * feature: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
124     '''
125
126     # Load the image and check the data type and dimensions
127     img = Image.open(img_path)
128     img = np.array(img).astype(np.float32) / 255
129     if len(img.shape) == 2:
130         img = img[:, :, np.newaxis]
131         img = np.tile(img, (1, 1, 3))
132
133     # Extract the wordmap from the image (use dictionary)
134     wordmap = visual_words.get_visual_words(opts, img, dictionary)
135
136     # Compute the Spatial Pyramid Matching features (use wordmap)
137     feature = get_feature_from_wordmap_SPM(opts, wordmap)
138
139     # Plot the feature to check
140     # plt.bar(range(feature.shape[0]), feature)
141     # plt.title(f"size = {feature.shape[0]}, SPM feature of
142     #         aquarium/sun_aadolwejqiyytvyne.jpg")

```

```

143     # plt.show()
144
145     return feature
146
147
148 def build_recognition_system(opts, n_worker=1):
149     '''
150     Creates a trained recognition system by generating training features from
151     all training images.
152
153     [input]
154     * opts: options
155     * n_worker: number of workers to process in parallel
156
157     [saved]
158     * features: numpy.ndarray of shape (N,M)
159     * labels: numpy.ndarray of shape (N)
160     * dictionary: numpy.ndarray of shape (K,3F)
161     * SPM_layer_num: number of spatial pyramid layers
162     '''
163
164     # Set up the file path and load the training files
165     data_dir = opts.data_dir
166     out_dir = opts.out_dir
167     SPM_layer_num = opts.L
168
169     # Load the trainig files and labels
170     train_files = open(join(data_dir, 'train_files.txt')).read().splitlines()
171     training_img_num = len(train_files)
172     train_labels = np.loadtxt(join(data_dir, 'train_labels.txt'), np.int32)
173     dictionary = np.load(join(out_dir, 'dictionary.npy'))
174
175     # Multiprocessing to extract the traing features
176     opts_list = [opts] * training_img_num
177     img_path = [join(data_dir, img_name) for img_name in train_files]
178     dictionary_list = [dictionary] * training_img_num
179     args = zip(opts_list, img_path, dictionary_list)
180     pool = multiprocessing.Pool(n_worker)
181     features = pool.starmap(get_image_feature, args)
182
183     # example code snippet to save the learned system
184     np.savez_compressed(join(out_dir, 'custom_trained_system.npz'), features =
185     features, labels = train_labels, dictionary = dictionary,
186     SPM_layer_num = SPM_layer_num)
187
188
189 # Use Intersection Similarity, Euclidean Distance, Kolmogorov-Smirnov Distance,
190 # and Match Distance as similarity scores

```

```

191 def distance_to_set(word_hist, histograms):
192     '''
193     Compute distance between a histogram of visual words with all training
194     image histograms.
195
196     [input]
197     * word_hist: numpy.ndarray of shape (K*(4^(L+1)-1)/3)
198     * histograms: numpy.ndarray of shape (T,K*(4^(L+1)-1)/3)
199
200     [output]
201     * dis: numpy.ndarray of shape (T)
202     '''
203
204     # Intersection Similarity
205     # Compute the intersection similarity between word_hist and histograms
206     # num_features, concatenated_size = histograms.shape
207     # intersection_similarity = np.minimum(word_hist, histograms)
208     # Compute the distance (inverse of the intersection similarity)
209     # dis = np.full((num_features), 1) - np.sum(intersection_similarity, axis =
210     1)
211
212     # Euclidean Distance
213     # Compute the L2 norm between word_hist and histograms
214     T, K = histograms.shape
215     word_hist_all = np.tile(word_hist, (T,1))
216     similarity = np.square(word_hist_all - histograms)
217     # Compute the distance
218     dis = np.sum(similarity, axis = 1)
219     dis = np.sqrt(dis)
220
221     # Kolmogorov-Smirnov Divergence
222     # Compute the difference between word_hist and histograms
223     # T, K = histograms.shape
224     # word_hist_all = np.tile(word_hist, (T,1))
225     # diff = abs(word_hist_all - histograms)
226     # Compute the distance
227     # dis = np.amax(diff, axis = 1)
228
229     # Match Distance
230     # Compute the difference between word_hist and histograms
231     # T, K = histograms.shape
232     # word_hist_all = np.tile(word_hist, (T,1))
233     # diff = abs(word_hist_all - histograms)
234     # Compute the distance
235     # dis = np.sum(diff, axis = 1)
236
237     return dis
238

```

```

239
240 def evaluate_recognition_system(opts, n_worker=1):
241     '''
242     Evaluates the recognition system for all test images and returns the
243     confusion matrix.
244
245     [input]
246     * opts: options
247     * n_worker: number of workers to process in parallel
248
249     [output]
250     * conf: numpy.ndarray of shape (8,8)
251     * accuracy: accuracy of the evaluated system
252     '''
253
254     # Set up file path and Load trained data
255     data_dir = opts.data_dir
256     out_dir = opts.out_dir
257     trained_system = np.load(join(out_dir, 'custom_trained_system.npz'))
258     dictionary = trained_system['dictionary']
259     trained_features = trained_system['features']
260     trained_labels = trained_system['labels']
261
262     # Use the stored options in the trained system instead of opts.py
263     test_opts = copy(opts)
264     test_opts.K = dictionary.shape[0]
265     test_opts.L = trained_system['SPM_layer_num']
266
267     # Load the test data
268     test_files = open(join(data_dir, 'test_files.txt')).read().splitlines()
269     test_img_num = len(test_files)
270     test_labels = np.loadtxt(join(data_dir, 'test_labels.txt'), np.int32)
271
272     # Extract the features from test data
273     opts_list = [opts] * test_img_num
274     img_path = [join(data_dir, img_name) for img_name in test_files]
275     dictionary_list = [dictionary] * test_img_num
276     args = zip(opts_list, img_path, dictionary_list)
277     pool = multiprocessing.Pool(n_worker)
278     test_features = np.asarray(pool.starmap(get_image_feature, args))
279     # np.savez_compressed(join(out_dir, 'custom_test_system.npz'), features =
280     #                     test_features)
281
282     # Compute the predicted labels
283     pred_labels = []
284     for test_index in range(test_img_num):
285         pred_index = np.argmin(distance_to_set(test_features[test_index, :],
286         trained_features))

```

```
287         pred_labels.append(trained_labels[pred_index])
288     pred_labels = np.asarray(pred_labels)
289
290     # Compute the Confusion Matrix and Accuracy
291     confusion_matrix = np.zeros((8, 8))
292     for true_index, pred_index in zip(test_labels, pred_labels):
293         confusion_matrix[true_index][pred_index] += 1
294     accuracy = np.sum(np.diag(confusion_matrix)) / np.sum(confusion_matrix)
295
296     return confusion_matrix, accuracy
```