In [7]:
```python
import math
import multiprocessing
import os
from copy import copy
from os.path import join

import numpy as np
import pandas as pd
import scipy.ndimage
import skimage.color
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.cluster import KMeans
from tqdm.autonotebook import tqdm

import imageio
from skimage import io
```

16-720 Computer Vision: Homework 1 (Spring 2022)

Spatial Pyramid Matching for Scene Classification

In [8]:

```
class Opts(object):
    def __init__(
        self,
        data_dir="../data",
        feat_dir="../feat",
        out_dir=".",
        filter_scales=(1, 2, 4, 8, 8*np.sqrt(2)),
        K=10,
        alpha=25,
        L=1,
    ):
        '''
        Manage tunable hyperparameters.

        You can also add your own additional hyperparameters.

        [input]
        * data_dir: Data directory.
        * feat_dir: Feature directory.
        * out_dir: Output directory.
        * filter_scales: A list of scales for all the filters.
        * K: Number of words.
        * alpha: Subset of alpha pixels in each image.
        * L: Number of layers in spatial pyramid matching (SPM).

        '''
        self.data_dir = data_dir
        self.feat_dir = feat_dir
        self.out_dir = out_dir
        self.filter_scales = list(filter_scales)
        self.K = K
        self.alpha = alpha
        self.L = L

opts = Opts()
```

In [9]:

```python
# utils

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()

    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):
    fig = plt.figure(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)

```

Question 1

Q1.1.1

The filters in these filter banks are Gaussian filter, Laplacian of Gaussian filter, the derivative of

Gaussian filter in x-direction and derivative of Gaussian filter in the y-direction.

Gaussian Filter: It is a smoothening low-pass filter. It removes high-frequencies from the image and allows for the lower frequencies to pass through it.

Laplacian of Gaussian Filter: It is a second derivative filter which gives zero response for uniform regions in an image and positive/negative response in regions where there are changes. These filters are commonly used to detect blob-like features in images.

Gaussian x-derivative: Captures the vertical edges in the image.

Gaussian y-derivative: Captures the horizontal edges in the image.

We need multiple scales of filter responses because we don't know the scale of the point of interest. A line in one scale can be represented as a point on another scale. So, with multiple scales of filter, we will be able to better represent the features.

Q1.1.2

In [ ]:

```

```

In [ ]:

```

```

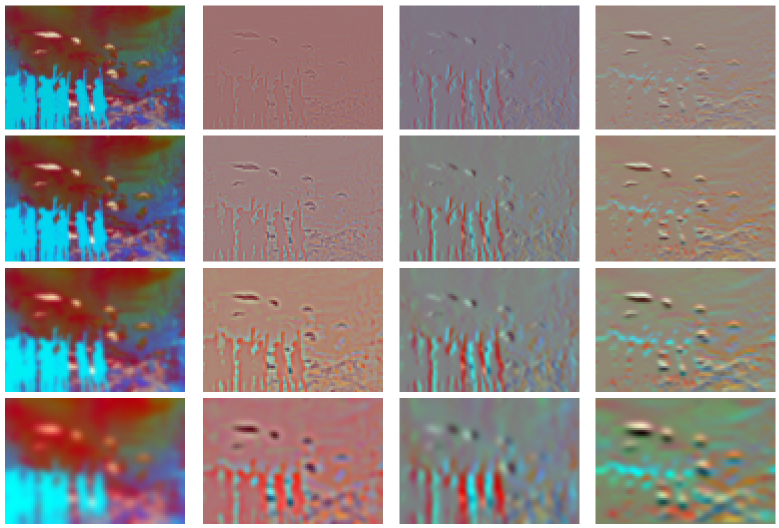
In [10]:

```
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)
    '''

    filter_scales = opts.filter_scales
    # ----- TODO -----

    F = 4

    if len(img.shape) < 3:
        C = 1
        H, W = img.shape
    else:
        H, W, C = img.shape

    if C == 1:
        img = np.expand_dims(img, axis = 2)
        img = np.tile(img, (1,1,3))
        H, W, C = img.shape

    if C == 4:
        img = img[:,:,0:3]
        H, W, C = img.shape

    img = img.astype('float')/255.
    img = skimage.color.rgb2lab(img)
    filter_responses = np.zeros((H, W, C*F*len(filter_scales)))


    for i in range(3):
        count = i
        for j in range(4):
            filter_responses[:,:,count] = scipy.ndimage.filters.gaussian_fil
            filter_responses[:,:,count + 3] = scipy.ndimage.filters.gaussian
            filter_responses[:,:,count + 6] = scipy.ndimage.filters.gaussian
            filter_responses[:,:,count + 9] = scipy.ndimage.filters.gaussian
            count += C*F
        count = i+1
#     filter_responses = np.dstack(filter_responses)

    return filter_responses
```

In [11]:

```python

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)
    '''

    filter_scales = opts.filter_scales
    # ----- TODO -----

    F = 4

    if len(img.shape) < 3:
        C = 1
        H, W = img.shape
    else:
        H, W, C = img.shape

    if C == 1:
        img = np.expand_dims(img, axis = 2)
        img = np.tile(img, (1,1,3))
        H, W, C = img.shape

    if C == 4:
        img = img[:,:,0:3]
        H, W, C = img.shape

    img = img.astype('float')/255.
    img = skimage.color.rgb2lab(img)
    filter_responses = []

    for sigma in filter_scales:
        for i in range(3):
            filter_responses.append(scipy.ndimage.filters.gaussian_filter(im

        for i in range(3):
            filter_responses.append(scipy.ndimage.filters.gaussian_laplace(i

        for i in range(3):
            filter_responses.append(scipy.ndimage.filters.gaussian_filter(im

        for i in range(3):
            filter_responses.append(scipy.ndimage.filters.gaussian_filter(im

    filter_responses = np.dstack(filter_responses)

    return filter_responses
```

In [12]:
```
# Should have filters for at least 3 scales.

opts.filter_scales = [1, 2, 4, 8]
img_path = join(opts.data_dir, 'aquarium/sun_aztvjgubyrgvirup.jpg')
img = plt.imread(img_path) / 255
filter_responses = extract_filter_responses(opts, img)
display_filter_responses(opts, filter_responses)
```



Q1.2

In [13]:

```
from numpy.random import default_rng

def compute_dictionary_one_image(img_path, opts):
    """
    Extracts a random subset of filter responses of an image and save it to
    This is a worker function called by compute_dictionary

    Your are free to make your own interface based on how you implement comp
    """
#     opts, idx, img_path = args
    # ----- TODO -----

    alpha = opts.alpha

    img = imageio.imread('../data/' + img_path)
    img = img.astype('float')/255

    filter_responses = extract_filter_responses(opts, img)
    H, W, C = filter_responses.shape

    responses = np.reshape(filter_responses, (H*W, C))

#     random = default_rng()
    joints = np.random.randint(H*W, size=alpha)

    responses = responses[joints, :]

    return responses

    np.save('%s%d'%(sample_response_path, i), np.asarray(responses))


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)
    """

    data_dir = opts.data_dir
    feat_dir = opts.feat_dir
    out_dir = opts.out_dir
    K = opts.K

    train_files = open(join(data_dir, "train_files.txt")).read().splitlines(
    # ----- TODO -----

    img_response=compute_dictionary_one_image(os.path.join(opts.data_dir,(tr
    img_stack=np.zeros((0,img_response.shape[1]))
    for i in range(len(train_files)):
        img_response=compute_dictionary_one_image(os.path.join(opts.data_dir
```

```
        img_stack=np.vstack((img_stack,img_response))
#         print(img_stack.shape)

    kmeans = KMeans(n_clusters=K,n_jobs=-1).fit(img_stack)
    dictionary = kmeans.cluster_centers_
    np.save('dictionary.npy',dictionary)

    return dictionary
```

In [14]:

```
print('abc')
n_cpu = get_num_CPU()
print(n_cpu)
compute_dictionary(opts, n_worker=n_cpu)
print('abc')
```

```
abc
12

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:792: Futu
reWarning: 'n_jobs' was deprecated in version 0.23 and will be removed in 1.0
(renaming of 0.25).
  warnings.warn("'n_jobs' was deprecated in version 0.23 and will be"

abc
```

Q1.3

The wordmap shows the contours in each image. Words change along the edges and tend to stay the same for homogenous regions.

In [15]:

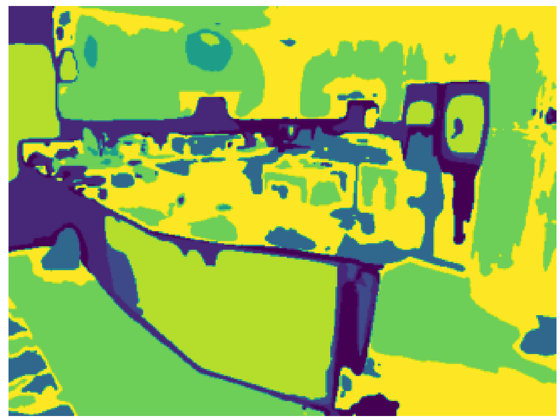
```
def get_visual_words(opts, img, dictionary):
    """
    Compute visual words mapping for the given img using the dictionary of v

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

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

    # ----- TODO -----
    filter_response = extract_filter_responses(opts, img)

    H, W, C = filter_response.shape
    filter_response = filter_response.reshape(H*W, C)

    distance = scipy.spatial.distance.cdist(filter_response, dictionary, met
    wordmap = np.argmin(distance, axis = 1)
    wordmap = wordmap.reshape(H, W)

    return wordmap

```

In [16]:

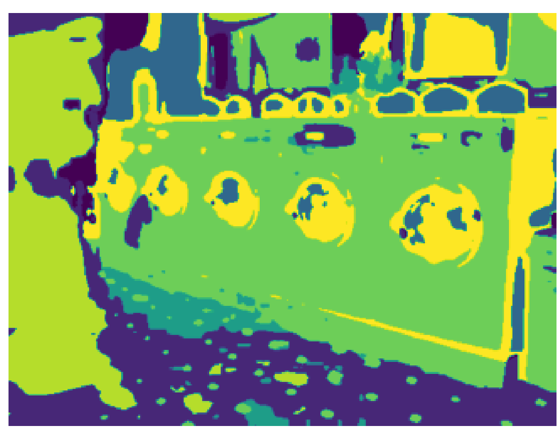
```
dictionary = np.load(join(opts.out_dir, 'dictionary.npy'))
```

In [17]:

```python
img_path = join(opts.data_dir, 'kitchen/sun_aasmevtpkslccptd.jpg')
img = plt.imread(img_path) / 255.
wordmap = get_visual_words(opts, img, dictionary)
visualize_wordmap(img, wordmap)

img_path = join(opts.data_dir, 'highway/sun_ailjxpgyepocjdos.jpg')
img = plt.imread(img_path) / 255.
wordmap = get_visual_words(opts, img, dictionary)
visualize_wordmap(img, wordmap)

img_path = join(opts.data_dir, 'laundromat/sun_aabvooxzwmzzvwds.jpg')
img = plt.imread(img_path) / 255.
wordmap = get_visual_words(opts, img, dictionary)
visualize_wordmap(img, wordmap)

```





Here we can see the images of kitchen, highway and the laundromat. The visualization does make sense but it varies from scene to scene. Consider the highway image, it has less defined features relative to the laundromat image, making the highway image harder to make sense wigthout the actual reference image next to it.

Q2.1

In [18]:
```
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)
    '''

    K = opts.K
    # ----- TODO -----
    H, W = wordmap.shape
    wordmap = np.reshape(wordmap, (1, H*W))
    hist, edges = np.histogram(wordmap, np.linspace(0, K, K+1, endpoint = Tr
    hist = hist/np.linalg.norm(hist, ord=1)
    hist = np.reshape(hist, (1, K))

    return hist
```

In [19]:
```
get_feature_from_wordmap(opts, wordmap)
```

Out[19]:
```
array([[2.48213333e-02, 2.05957333e-01, 1.49333333e-04, 6.73546667e-02,
        0.00000000e+00, 3.59200000e-02, 0.00000000e+00, 3.74906667e-01,
        1.35264000e-01, 1.55626667e-01]])
```

Q2.2

In [101]:

```
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)/3)
    """

    K = opts.K
    L = opts.L
    # ----- TODO -----
    H, W = wordmap.shape

    hist_all = []
    norm_factor = H*W

    for i in range(L+1):

        if i == 0 or i == 1:
            weight = 2**(-L)
        else:
            weight = 2**(L-i-1)

        cell_num = 2**i

        x = np.array_split(wordmap, cell_num, axis=0)
        for r in x:
            y = np.array_split(r, cell_num, axis=1)
            for c in y:
                hist, bin_edges = np.histogram(c, bins=K)
                hist_all = np.append(hist_all, hist / norm_factor * weight)



    return hist_all
```

In [21]:
```
img_path = join(opts.data_dir, 'kitchen/sun_aasmevtpkslccptd.jpg')
img = plt.imread(img_path) / 255.
wordmap = get_visual_words(opts, img, dictionary)
hist = get_feature_from_wordmap_SPM(opts, wordmap)
print(hist)
```

```
[0.02398133 0.04710933 0.005872   0.03305333 0.         0.004328
 0.         0.16242933 0.07177867 0.151448   0.00380533 0.022856
 0.         0.005672   0.         0.004328   0.         0.04239467
 0.01204267 0.03423467 0.00819733 0.008128   0.00028267 0.01006133
 0.         0.         0.         0.03782667 0.00543467 0.05540267
 0.00256533 0.01257867 0.00271467 0.00686667 0.         0.
 0.         0.04837067 0.03905867 0.012512   0.00941333 0.00354667
 0.00287467 0.01045333 0.         0.         0.         0.03383733
 0.01524267 0.04929867]
```

Q2.3

In [22]:
```
def distance_to_set(word_hist, histograms):
    """
    Compute the distance between a histogram of visual words with all traini

    [input]
    * word_hist: numpy.ndarray of shape (K)
    * histograms: numpy.ndarray of shape (N,K)

    [output]
    * dists: numpy.ndarray of shape (N)
    """

    # ----- TODO -----
    dists = None

    d = np.minimum(word_hist, histograms)
    dists = np.sum(d, axis = 1)

    return dists
```

In [23]:
```
 word_hist = [1, 2]
histograms = [[1, 2],[0,4]]
distance_to_set(word_hist, histograms)
```

Out[23]:
```
array([3, 2])
```

Q2.4

In [24]:

```
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)
    """

    # ----- TODO -----
#     feature = None
    image = io.imread(img_path)
    image = image.astype('float')/255
    wordmap = get_visual_words(opts, image, dictionary)
    feature = get_feature_from_wordmap_SPM(opts, wordmap)
#     print(np.shape(wordmap), wordmap)
    return feature
```

In [ ]:

```

```

In [76]:

```
def build_recognition_system(opts, n_worker=1):
    """
    Creates a trained recognition system by generating training features fro

    [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
    """

    data_dir = opts.data_dir
    out_dir = opts.out_dir
    SPM_layer_num = opts.L

    train_files = open(join(data_dir, "train_files.txt")).read().splitlines(
    train_labels = np.loadtxt(join(data_dir, "train_labels.txt"), np.int32)
    dictionary = np.load(join(out_dir, "dictionary.npy"))

    # ----- TODO -----
    K = opts.K
    L = opts.L
#     labels = []
    train_data = np.asarray(train_files)
    length_train = train_data.shape[0]

    hist_features = []

    for i in range(0, length_train ):
        image_path = os.path.join(data_dir, train_files[i])
        features = get_image_feature(opts, image_path, dictionary)
#         print(features)
#         print("Shape of Feature: ", np.shape(hist_features))
        hist_features.append(features)


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

```

In [26]:
```
build_recognition_system(opts, n_worker=n_cpu)

```

```
Shape of Feature:  (1114, 50)
Shape of Feature:  (1115, 50)
Shape of Feature:  (1116, 50)
Shape of Feature:  (1117, 50)
Shape of Feature:  (1118, 50)
Shape of Feature:  (1119, 50)
Shape of Feature:  (1120, 50)
Shape of Feature:  (1121, 50)
Shape of Feature:  (1122, 50)
Shape of Feature:  (1123, 50)
Shape of Feature:  (1124, 50)
Shape of Feature:  (1125, 50)
Shape of Feature:  (1126, 50)
Shape of Feature:  (1127, 50)
Shape of Feature:  (1128, 50)
Shape of Feature:  (1129, 50)
Shape of Feature:  (1130, 50)
```

In [ ]:
```

```

Q2.5

In [79]:

```
def evaluate_recognition_system(opts, n_worker=1):
    """
    Evaluates the recognition system for all test images and returns the con

    [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
    """

    data_dir = opts.data_dir
    out_dir = opts.out_dir

    trained_system = np.load(join(out_dir, "trained_system.npz"))
    dictionary = trained_system["dictionary"]

    # using 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"]
#     print(test_opts.K, test_opts.L)

    test_files = open(join(data_dir, "test_files.txt")).read().splitlines()
    test_labels = np.loadtxt(join(data_dir, "test_labels.txt"), np.int32)

    # ----- TODO -----
    conf, accuracy = None, None

    trained_features = trained_system['features']
    train_labels = trained_system['labels']
    test_labels = np.asarray(test_labels)
#     print(np.shape(trained_features), np.shape(test_labels))
    conf = np.zeros((8,8))
    pred_label=list()
#     count = 0
    for i in range(len(test_files)):
        image_path = os.path.join(data_dir, test_files[i])
        hist_all = get_image_feature(opts, image_path, dictionary)
#         print(np.shape(hist_all))
#         print(trained_features)
        distance = distance_to_set(hist_all, trained_features)
        pred_index = np.argmax(distance)
#         p = train_labels[pred_index]
#         pred_label.append(p)
#         print(pred_index)
        pred_label = train_labels[pred_index]
        conf[test_labels[i], pred_label] += 1
#         if(pred_label[i] == test_labels[i]):
#             count+=1
#     accuracy = count/len(test_labels)
        accuracy = np.trace(conf)/np.sum(conf)
        print("{} accuracy: {}".format(i, accuracy))

```

```
    return conf, accuracy
```

In [ ]:

```

```

In [ ]:

```

```

Initial Case

filter_scales = [1, 2, 4, 8]

K = 10

alpha = 25

L = 1

Accuracy achieved: 49.25%

Confusion Matrix:

[[34. 0. 1. 2. 2. 3. 5. 3.]

[0. 23. 8. 8. 5. 1. 1. 4.]

[0. 6. 26. 3. 3. 3. 2. 7.]

[4. 3. 0. 25. 12. 4. 2. 0.]

[1. 4. 0. 11. 16. 8. 6. 4.]

[2. 0. 7. 2. 4. 27. 6. 2.]

[3. 1. 1. 5. 8. 5. 22. 5.]

[0. 3. 7. 4. 1. 7. 4. 24.]]

In [28]:

```
conf, accuracy = evaluate_recognition_system(opts, n_worker=n_cpu)

print("Accuracy:", accuracy)
classes = [
    "aquarium", "desert", "highway", "kitchen",
    "laundromat", "park", "waterfall", "windmill",
]
df = pd.DataFrame(conf, columns=classes)
df.insert(0, "", classes)
df
```

```
397 accuracy: 0.49246231155778897
398 accuracy: 0.49122807017543857
399 accuracy: 0.4925
Accuracy: 0.4925
```

Out[28]:

		aquarium	desert	highway	kitchen	laundromat	park	waterfall	windmill
0	aquarium	34.0	0.0	1.0	2.0	2.0	3.0	5.0	3.0
1	desert	0.0	23.0	8.0	8.0	5.0	1.0	1.0	4.0
2	highway	0.0	6.0	26.0	3.0	3.0	3.0	2.0	7.0
3	kitchen	4.0	3.0	0.0	25.0	12.0	4.0	2.0	0.0
4	laundromat	1.0	4.0	0.0	11.0	16.0	8.0	6.0	4.0
5	park	2.0	0.0	7.0	2.0	4.0	27.0	6.0	2.0
6	waterfall	3.0	1.0	1.0	5.0	8.0	5.0	22.0	5.0
7	windmill	0.0	3.0	7.0	4.0	1.0	7.0	4.0	24.0

In [29]:

```
print(conf)
```

```
[[34.  0.  1.  2.  2.  3.  5.  3.]
 [ 0. 23.  8.  8.  5.  1.  1.  4.]
 [ 0.  6. 26.  3.  3.  3.  2.  7.]
 [ 4.  3.  0. 25. 12.  4.  2.  0.]
 [ 1.  4.  0. 11. 16.  8.  6.  4.]
 [ 2.  0.  7.  2.  4. 27.  6.  2.]
 [ 3.  1.  1.  5.  8.  5. 22.  5.]
 [ 0.  3.  7.  4.  1.  7.  4. 24.]]
```

Q2.6

We can see that there are some pairs which have more intersection than others, signifying that these are some of the hard classes to classify.

From the confusion matrix, we can infer some of these pairs as: (laundromat, kitchen); (waterfall, park); (windmill, highway)

The reason for this can be more than one. Some of which include: Small number of clusters, alpha number for the patches which might result in the classifier not having enough information to classify it to any class, etc

This is a laundromat image that was classified as kitchen. We can see how the model machine might have gotten confused between the two because the washing machines resemble a counter table of the kitchen and there is not much to differentiate with. Similar images of waterfall and park might have been deteched as well since both the scenes have vast amount of similar structure, park has a grass scene and waterfall is an image filled with water, again making it hard for the system to actually find features to differentiate with.

Q3.1

Filter Scales	K	Alpha	L	Accuracy
[1, 2, 4, 8, 16]	50	100	2	55
[1, 2, 4, 8, 16]	100	150	2	59.75
[1, 2, 4, 8, 12]	**75**	**125**	**2**	**60.75**
[1, 2,4, 8, 10]	100	200	3	59.75
[1, 2, 3, 5, 10]	200	200	2	56.25

The above table shows the multiple different test cases that were tried to improve the accuracy. The bold row is the one that receieved the maximum accuracy. I had tried more test cases with higher L values and different scales and alpha values, but the accuracy seemed to be stuck between 59-62 for some reason.

As the value of K increases the elements in a cluster reduces. So average distortion should decrease due to which classification accuracy increases.

Whnen Layers (L) are increased, more details can be captured, making the classification easier and thus increasing accuracy.

Alpha increment provides more data and filter scales help you pick all the multi sized features, making it flexible.

Case1

filter_scales = [1, 2, 4, 8, 16]

K = 50

alpha = 100

L = 2

Accuracy achieved: 55%

Confusion Matrix:

[[33. 5. 0. 3. 1. 1. 1. 6.]

[0. 31. 5. 7. 4. 1. 0. 2.]

[2. 4. 25. 3. 0. 1. 7. 8.]

[4. 0. 1. 35. 9. 0. 1. 0.]

[1. 1. 1. 13. 25. 3. 6. 0.]

[1. 0. 2. 4. 2. 31. 6. 4.]

[3. 2. 4. 4. 1. 14. 19. 3.]

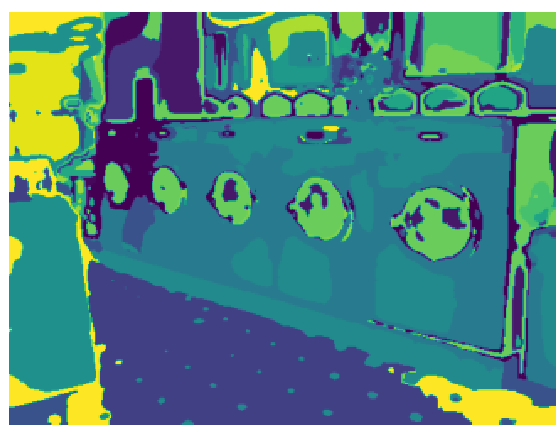
[1. 4. 9. 2. 1. 6. 6. 21.]]

In [30]:
```
opts.filter_scales = [1, 2, 4, 8, 16]
opts.K = 50
opts.alpha = 100
opts.L = 2


compute_dictionary(opts, n_worker = n_cpu)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:792: FutureWarning: 'n_jobs' was deprecated in version 0.23 and will be removed in 1.0 (renaming of 0.25).
  warnings.warn("'n_jobs' was deprecated in version 0.23 and will be"
```

Out[30]:
```
array([[ 1.15876212e-01,  4.36782268e-03,  4.05798567e-02, ...,
        -5.21518768e-05,  1.86628810e-05,  4.52797306e-05],
       [ 1.03455566e-01,  1.27282049e-03,  2.84666342e-03, ...,
        -1.79602358e-04,  2.37297312e-05,  6.75507649e-05],
       [ 2.28347684e-01, -3.85973657e-02, -4.56490844e-02, ...,
        -1.20623188e-04,  4.59700828e-05,  3.01844266e-04],
       ...,
       [ 5.75928270e-02, -8.56209810e-03, -7.30004015e-02, ...,
        -1.87743934e-05,  7.62965416e-06,  1.34216587e-04],
       [ 2.00365441e-01,  1.84316568e-03,  4.05263572e-02, ...,
        -1.17066480e-04,  1.81136066e-05,  1.11786048e-04],
       [ 9.29853627e-02,  1.10812091e-02, -2.14113370e-01, ...,
        -9.38671153e-05,  2.16766730e-05,  2.72513921e-04]])
```

In [31]:
```
dictionary = np.load(join(opts.out_dir, 'dictionary.npy'))
# print(np.shape(dictionary))

img_path = join(opts.data_dir, 'laundromat/sun_aabvooxzwmzzvwds.jpg')
img = plt.imread(img_path) / 255.
wordmap = get_visual_words(opts, img, dictionary)
visualize_wordmap(img, wordmap)

```



In [32]:

```
build_recognition_system(opts, n_worker = n_cpu)
```

```
Shape of Feature:  (1158, 1050)
Shape of Feature:  (1159, 1050)
Shape of Feature:  (1160, 1050)
Shape of Feature:  (1161, 1050)
Shape of Feature:  (1162, 1050)
Shape of Feature:  (1163, 1050)
Shape of Feature:  (1164, 1050)
Shape of Feature:  (1165, 1050)
Shape of Feature:  (1166, 1050)
Shape of Feature:  (1167, 1050)
Shape of Feature:  (1168, 1050)
Shape of Feature:  (1169, 1050)
Shape of Feature:  (1170, 1050)
Shape of Feature:  (1171, 1050)
Shape of Feature:  (1172, 1050)
Shape of Feature:  (1173, 1050)
Shape of Feature:  (1174, 1050)

Shape of Feature:  (1175, 1050)
Shape of Feature:  (1176, 1050)
```

In [33]:

```
conf, accuracy = evaluate_recognition_system(opts, n_worker=n_cpu)

print("Accuracy:", accuracy)
```

```
382 accuracy: 0.556135770234987
383 accuracy: 0.5546875
384 accuracy: 0.5532467532467532
385 accuracy: 0.5544041450777202
386 accuracy: 0.5529715762273901
387 accuracy: 0.5541237113402062
388 accuracy: 0.5526992287917738
389 accuracy: 0.5512820512820513
390 accuracy: 0.5498721227621484
391 accuracy: 0.548469387755102
392 accuracy: 0.549618320610687
393 accuracy: 0.5482233502538071
394 accuracy: 0.549367088607595
395 accuracy: 0.547979797979798
396 accuracy: 0.5465994962216625
397 accuracy: 0.5477386934673367
398 accuracy: 0.5488721804511278
399 accuracy: 0.55
Accuracy: 0.55
```

In [34]:
```
classes = [
    "aquarium", "desert", "highway", "kitchen",
    "laundromat", "park", "waterfall", "windmill",
]
df = pd.DataFrame(conf, columns=classes)
df.insert(0, "", classes)
df
```

Out[34]:

		aquarium	desert	highway	kitchen	laundromat	park	waterfall	windmill
0	aquarium	33.0	5.0	0.0	3.0	1.0	1.0	1.0	6.0
1	desert	0.0	31.0	5.0	7.0	4.0	1.0	0.0	2.0
2	highway	2.0	4.0	25.0	3.0	0.0	1.0	7.0	8.0
3	kitchen	4.0	0.0	1.0	35.0	9.0	0.0	1.0	0.0
4	laundromat	1.0	1.0	1.0	13.0	25.0	3.0	6.0	0.0
5	park	1.0	0.0	2.0	4.0	2.0	31.0	6.0	4.0
6	waterfall	3.0	2.0	4.0	4.0	1.0	14.0	19.0	3.0
7	windmill	1.0	4.0	9.0	2.0	1.0	6.0	6.0	21.0

In [35]:
```
print(conf)
```

```
[[33.  5.  0.  3.  1.  1.  1.  6.]
 [ 0. 31.  5.  7.  4.  1.  0.  2.]
 [ 2.  4. 25.  3.  0.  1.  7.  8.]
 [ 4.  0.  1. 35.  9.  0.  1.  0.]
 [ 1.  1.  1. 13. 25.  3.  6.  0.]
 [ 1.  0.  2.  4.  2. 31.  6.  4.]
 [ 3.  2.  4.  4.  1. 14. 19.  3.]
 [ 1.  4.  9.  2.  1.  6.  6. 21.]]
```

In [ ]:
```

```

In [ ]:
```

```

Case2

filter_scales = [1, 2, 4, 8, 16]

K = 100

alpha = 150

L = 2

Accuracy achieved: 59.75%

Confusion Matrix:

[[33. 3. 5. 1. 2. 2. 3. 1.]

[1. 29. 6. 2. 4. 1. 1. 6.]

[2. 3. 25. 0. 3. 7. 4. 6.]

[4. 1. 1. 33. 6. 1. 2. 2.]

[1. 1. 2. 15. 26. 3. 2. 0.]

[3. 0. 3. 1. 2. 39. 2. 0.]

[1. 2. 4. 1. 3. 8. 28. 3.]

[0. 3. 7. 2. 0. 6. 6. 26.]]

In [38]:

```
opts.filter_scales = [1, 2, 4, 8, 16]
opts.K = 100
opts.alpha = 150
opts.L = 2


compute_dictionary(opts, n_worker = n_cpu)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:792: Futu
reWarning: 'n_jobs' was deprecated in version 0.23 and will be removed in 1.0
(renaming of 0.25).
  warnings.warn("'n_jobs' was deprecated in version 0.23 and will be"
```

Out[38]:

```
array([[ 9.51995574e-02, -3.02588995e-02,  3.89448314e-02, ...,
        -9.82660421e-05, -2.41316505e-06,  1.13328688e-04],
       [ 2.40745624e-01, -6.68988718e-03, -2.31454853e-02, ...,
        -2.29451210e-04,  3.53342495e-05,  1.70943453e-04],
       [ 1.34470733e-01, -1.08000023e-02, -7.34082026e-02, ...,
         4.12280392e-05,  8.24376616e-06,  2.82038431e-04],
       ...,
       [ 1.46432668e-01, -8.96352498e-02,  1.14805161e-01, ...,
        -5.12143211e-05, -2.21977612e-04,  2.11997921e-04],
       [ 1.23446784e-01,  1.90365096e-02,  1.17470404e-01, ...,
        -5.90639366e-05,  4.69625512e-06,  1.53943161e-05],
       [ 1.87571906e-01, -2.51547342e-03,  4.91168370e-03, ...,
        -1.88244007e-04, -1.66824016e-06,  4.72441652e-05]])
```

In [39]:

```
dictionary = np.load(join(opts.out_dir, 'dictionary.npy'))
# print(np.shape(dictionary))

img_path = join(opts.data_dir, 'laundromat/sun_aabvooxzwmzzvwds.jpg')
img = plt.imread(img_path) / 255.
wordmap = get_visual_words(opts, img, dictionary)
visualize_wordmap(img, wordmap)

```

In [40]:

```
build_recognition_system(opts, n_worker = n_cpu)
```

```
Shape of Feature:  (1158, 2100)
Shape of Feature:  (1159, 2100)
Shape of Feature:  (1160, 2100)
Shape of Feature:  (1161, 2100)
Shape of Feature:  (1162, 2100)
Shape of Feature:  (1163, 2100)
Shape of Feature:  (1164, 2100)
Shape of Feature:  (1165, 2100)
Shape of Feature:  (1166, 2100)
Shape of Feature:  (1167, 2100)
Shape of Feature:  (1168, 2100)
Shape of Feature:  (1169, 2100)
Shape of Feature:  (1170, 2100)
Shape of Feature:  (1171, 2100)
Shape of Feature:  (1172, 2100)
Shape of Feature:  (1173, 2100)
Shape of Feature:  (1174, 2100)

Shape of Feature:  (1175, 2100)
Shape of Feature:  (1176, 2100)
```

In [41]:

```python
conf, accuracy = evaluate_recognition_system(opts, n_worker=n_cpu)

print("Accuracy:", accuracy)
```

```
382 accuracy: 0.6057441253263708
383 accuracy: 0.6067708333333334
384 accuracy: 0.6051948051948052
385 accuracy: 0.6036269430051814
386 accuracy: 0.6020671834625323
387 accuracy: 0.6030927835051546
388 accuracy: 0.6015424164524421
389 accuracy: 0.6025641025641025
390 accuracy: 0.6035805626598465
391 accuracy: 0.6020408163265306
392 accuracy: 0.6030534351145038
393 accuracy: 0.6015228426395939
394 accuracy: 0.6
395 accuracy: 0.601010101010101
396 accuracy: 0.5994962216624685
397 accuracy: 0.5979899497487438
398 accuracy: 0.5989974937343359
399 accuracy: 0.5975
Accuracy: 0.5975
```

In [45]:

```python
classes = [
    "aquarium", "desert", "highway", "kitchen",
    "laundromat", "park", "waterfall", "windmill",
]
df = pd.DataFrame(conf, columns=classes)
df.insert(0, "", classes)
df
```

Out[45]:

		aquarium	desert	highway	kitchen	laundromat	park	waterfall	windmill
0	aquarium	33.0	3.0	5.0	1.0	2.0	2.0	3.0	1.0
1	desert	1.0	29.0	6.0	2.0	4.0	1.0	1.0	6.0
2	highway	2.0	3.0	25.0	0.0	3.0	7.0	4.0	6.0
3	kitchen	4.0	1.0	1.0	33.0	6.0	1.0	2.0	2.0
4	laundromat	1.0	1.0	2.0	15.0	26.0	3.0	2.0	0.0
5	park	3.0	0.0	3.0	1.0	2.0	39.0	2.0	0.0
6	waterfall	1.0	2.0	4.0	1.0	3.0	8.0	28.0	3.0
7	windmill	0.0	3.0	7.0	2.0	0.0	6.0	6.0	26.0

In [46]:
```
print(conf)
```

```
[[33.  3.  5.  1.  2.  2.  3.  1.]
 [ 1. 29.  6.  2.  4.  1.  1.  6.]
 [ 2.  3. 25.  0.  3.  7.  4.  6.]
 [ 4.  1.  1. 33.  6.  1.  2.  2.]
 [ 1.  1.  2. 15. 26.  3.  2.  0.]
 [ 3.  0.  3.  1.  2. 39.  2.  0.]
 [ 1.  2.  4.  1.  3.  8. 28.  3.]
 [ 0.  3.  7.  2.  0.  6.  6. 26.]]
```

In [ ]:
```

```

In [ ]:
```

```

Case3

filter_scales = [1, 2, 4, 8, 12]

K = 75

alpha = 125

L = 2

Accuracy achieved: 60.75%

Confusion Matrix:

[[31. 1. 2. 2. 6. 2. 3. 3.]

[1. 30. 3. 6. 4. 3. 3. 0.]

[1. 3. 30. 2. 5. 1. 3. 5.]

[2. 0. 1. 31. 15. 0. 1. 0.]

[0. 0. 0. 12. 30. 5. 2. 1.]

[3. 0. 4. 2. 2. 36. 3. 0.]

[2. 0. 1. 2. 5. 7. 30. 3.]

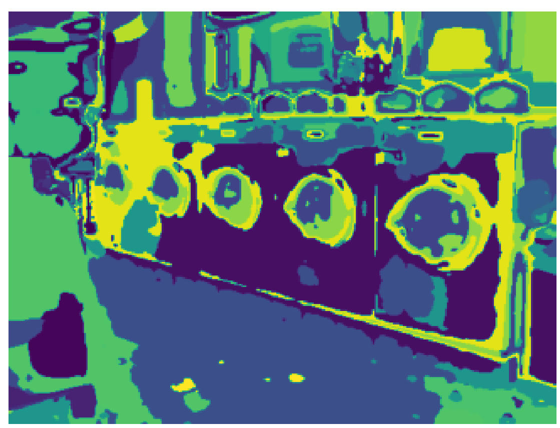
[1. 3. 5. 3. 4. 5. 4. 25.]]

In [47]:
```
opts.filter_scales = [1, 2, 4, 8, 12]
opts.K = 75
opts.alpha = 125
opts.L = 2


compute_dictionary(opts, n_worker = n_cpu)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:792: Futu
reWarning: 'n_jobs' was deprecated in version 0.23 and will be removed in 1.0
(renaming of 0.25).
  warnings.warn("'n_jobs' was deprecated in version 0.23 and will be"
```

Out[47]:
```
array([[ 5.95452137e-02,  6.96601560e-04,  3.76379443e-03, ...,
        -3.08905703e-04,  3.31365833e-05,  1.23726408e-04],
       [ 2.38887560e-01, -8.26069175e-04,  2.88026701e-03, ...,
        -2.54312634e-04, -1.02548407e-05,  8.29625924e-05],
       [ 1.36621767e-01, -2.59319951e-03, -3.56313909e-02, ...,
        -2.86191823e-04,  1.74122237e-05,  2.45691422e-04],
       ...,
       [ 1.74387705e-01, -1.67885017e-03,  4.37426562e-03, ...,
        -1.31298074e-04,  1.58452824e-05,  6.76558440e-05],
       [ 1.29279498e-01, -5.68756333e-04,  2.11790326e-02, ...,
        -8.65162840e-05,  2.07977186e-05,  3.27904258e-05],
       [ 2.27241325e-01, -4.47448141e-02,  9.42580553e-02, ...,
        -2.09557101e-05, -1.19034273e-04,  1.81845101e-04]])
```

In [48]:
```
dictionary = np.load(join(opts.out_dir, 'dictionary.npy'))
# print(np.shape(dictionary))

img_path = join(opts.data_dir, 'laundromat/sun_aabvooxzwmzzvwds.jpg')
img = plt.imread(img_path) / 255.
wordmap = get_visual_words(opts, img, dictionary)
visualize_wordmap(img, wordmap)

```



In [49]:
```
build_recognition_system(opts, n_worker = n_cpu)
```

```
Shape of Feature:  (1158, 1575)
Shape of Feature:  (1159, 1575)
Shape of Feature:  (1160, 1575)
Shape of Feature:  (1161, 1575)
Shape of Feature:  (1162, 1575)
Shape of Feature:  (1163, 1575)
Shape of Feature:  (1164, 1575)
Shape of Feature:  (1165, 1575)
Shape of Feature:  (1166, 1575)
Shape of Feature:  (1167, 1575)
Shape of Feature:  (1168, 1575)
Shape of Feature:  (1169, 1575)
Shape of Feature:  (1170, 1575)
Shape of Feature:  (1171, 1575)
Shape of Feature:  (1172, 1575)
Shape of Feature:  (1173, 1575)
Shape of Feature:  (1174, 1575)

Shape of Feature:  (1175, 1575)
Shape of Feature:  (1176, 1575)
```

In [50]:
```
conf, accuracy = evaluate_recognition_system(opts, n_worker=n_cpu)

print("Accuracy:", accuracy)
```

```
382 accuracy: 0.6109660574412533
383 accuracy: 0.6119791666666666
384 accuracy: 0.6103896103896104
385 accuracy: 0.6088082901554405
386 accuracy: 0.6098191214470284
387 accuracy: 0.6108247422680413
388 accuracy: 0.609254498714653
389 accuracy: 0.6102564102564103
390 accuracy: 0.6086956521739131
391 accuracy: 0.6071428571428571
392 accuracy: 0.6081424936386769
393 accuracy: 0.6065989847715736
394 accuracy: 0.6050632911392405
395 accuracy: 0.6035353535353535
396 accuracy: 0.6045340050377834
397 accuracy: 0.6055276381909548
398 accuracy: 0.606516290726817
399 accuracy: 0.6075
Accuracy: 0.6075
```

In [51]:
```
classes = [
    "aquarium", "desert", "highway", "kitchen",
    "laundromat", "park", "waterfall", "windmill",
]
df = pd.DataFrame(conf, columns=classes)
df.insert(0, "", classes)
df
```

Out[51]:

		aquarium	desert	highway	kitchen	laundromat	park	waterfall	windmill
0	aquarium	31.0	1.0	2.0	2.0	6.0	2.0	3.0	3.0
1	desert	1.0	30.0	3.0	6.0	4.0	3.0	3.0	0.0
2	highway	1.0	3.0	30.0	2.0	5.0	1.0	3.0	5.0
3	kitchen	2.0	0.0	1.0	31.0	15.0	0.0	1.0	0.0
4	laundromat	0.0	0.0	0.0	12.0	30.0	5.0	2.0	1.0
5	park	3.0	0.0	4.0	2.0	2.0	36.0	3.0	0.0
6	waterfall	2.0	0.0	1.0	2.0	5.0	7.0	30.0	3.0
7	windmill	1.0	3.0	5.0	3.0	4.0	5.0	4.0	25.0

In [52]:
```
print(conf)
```

```
[[31.  1.  2.  2.  6.  2.  3.  3.]
 [ 1. 30.  3.  6.  4.  3.  3.  0.]
 [ 1.  3. 30.  2.  5.  1.  3.  5.]
 [ 2.  0.  1. 31. 15.  0.  1.  0.]
 [ 0.  0.  0. 12. 30.  5.  2.  1.]
 [ 3.  0.  4.  2.  2. 36.  3.  0.]
 [ 2.  0.  1.  2.  5.  7. 30.  3.]
 [ 1.  3.  5.  3.  4.  5.  4. 25.]]
```

In [ ]:
```

```

In [ ]:
```

```

Case4

filter_scales = [1, 2, 4, 8, 10]

K = 100

alpha = 200

L = 3

Accuracy achieved: 59.75%

Confusion Matrix:

[[31. 7. 0. 2. 3. 3. 2. 2.]

[1. 34. 8. 3. 1. 0. 1. 2.]

[3. 6. 26. 4. 1. 3. 0. 7.]

[2. 4. 2. 26. 14. 1. 1. 0.]

[0. 2. 0. 9. 31. 5. 2. 1.]

[1. 1. 1. 2. 3. 36. 4. 2.]

[1. 5. 2. 1. 4. 8. 28. 1.]

[0. 4. 6. 0. 1. 8. 4. 27.]]

In [53]:

```
opts.filter_scales = [1, 2, 4, 8, 10]
opts.K = 100
opts.alpha = 200
opts.L = 3


compute_dictionary(opts, n_worker = n_cpu)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:792: Futu
reWarning: 'n_jobs' was deprecated in version 0.23 and will be removed in 1.0
(renaming of 0.25).
  warnings.warn("'n_jobs' was deprecated in version 0.23 and will be"
```

Out[53]:

```
array([[ 8.21060057e-02, -5.67046124e-04,  3.47494867e-03, ...,
        -4.19657783e-04,  3.42027444e-07,  1.47033474e-04],
       [ 5.34965995e-02, -1.22170931e-03,  2.49588810e-02, ...,
        -7.21907379e-05,  1.74770369e-05,  1.18251574e-05],
       [ 1.29752981e-01,  1.82200336e-02,  8.61959020e-02, ...,
        -5.96709962e-05,  2.94634095e-05,  7.73447819e-05],
       ...,
       [ 1.29471524e-01, -2.84972263e-02, -1.08635313e-01, ...,
         1.86785199e-04, -3.52658144e-05,  2.05732741e-04],
       [ 1.59056442e-01, -3.59041763e-03,  2.36466965e-02, ...,
        -1.58189172e-04,  1.67860724e-05,  5.21496198e-05],
       [ 1.10161698e-01, -9.89226734e-02, -6.83617294e-02, ...,
        -7.24628781e-05,  7.66037647e-05,  1.94375326e-04]])
```

In [54]:

```
dictionary = np.load(join(opts.out_dir, 'dictionary.npy'))
# print(np.shape(dictionary))

img_path = join(opts.data_dir, 'laundromat/sun_aabvooxzwmzzvwds.jpg')
img = plt.imread(img_path) / 255.
wordmap = get_visual_words(opts, img, dictionary)
visualize_wordmap(img, wordmap)
```

In [55]:

```
build_recognition_system(opts, n_worker = n_cpu)
```

```
Shape of Feature:  (1158, 8500)
Shape of Feature:  (1159, 8500)
Shape of Feature:  (1160, 8500)
Shape of Feature:  (1161, 8500)
Shape of Feature:  (1162, 8500)
Shape of Feature:  (1163, 8500)
Shape of Feature:  (1164, 8500)
Shape of Feature:  (1165, 8500)
Shape of Feature:  (1166, 8500)
Shape of Feature:  (1167, 8500)
Shape of Feature:  (1168, 8500)
Shape of Feature:  (1169, 8500)
Shape of Feature:  (1170, 8500)
Shape of Feature:  (1171, 8500)
Shape of Feature:  (1172, 8500)
Shape of Feature:  (1173, 8500)
Shape of Feature:  (1174, 8500)

Shape of Feature:  (1175, 8500)
Shape of Feature:  (1176, 8500)
```

In [56]:

```
conf, accuracy = evaluate_recognition_system(opts, n_worker=n_cpu)

print("Accuracy:", accuracy)
```

```
381 accuracy: 0.5942408376963351
382 accuracy: 0.5926892950391645
383 accuracy: 0.59375
384 accuracy: 0.5922077922077922
385 accuracy: 0.5932642487046632
386 accuracy: 0.5917312661498708
387 accuracy: 0.5927835051546392
388 accuracy: 0.5938303341902313
389 accuracy: 0.5948717948717949
390 accuracy: 0.5959079283887468
391 accuracy: 0.5969387755102041
392 accuracy: 0.5979643765903307
393 accuracy: 0.5964467005076142
394 accuracy: 0.5949367088607594
395 accuracy: 0.5959595959595959
396 accuracy: 0.5944584382871536
397 accuracy: 0.5954773869346733
398 accuracy: 0.5964912280701754
399 accuracy: 0.5975
Accuracy: 0.5975
```

In [57]:

```
classes = [
    "aquarium", "desert", "highway", "kitchen",
    "laundromat", "park", "waterfall", "windmill",
]
df = pd.DataFrame(conf, columns=classes)
df.insert(0, "", classes)
df
```

Out[57]:

		aquarium	desert	highway	kitchen	laundromat	park	waterfall	windmill
0	aquarium	31.0	7.0	0.0	2.0	3.0	3.0	2.0	2.0
1	desert	1.0	34.0	8.0	3.0	1.0	0.0	1.0	2.0
2	highway	3.0	6.0	26.0	4.0	1.0	3.0	0.0	7.0
3	kitchen	2.0	4.0	2.0	26.0	14.0	1.0	1.0	0.0
4	laundromat	0.0	2.0	0.0	9.0	31.0	5.0	2.0	1.0
5	park	1.0	1.0	1.0	2.0	3.0	36.0	4.0	2.0
6	waterfall	1.0	5.0	2.0	1.0	4.0	8.0	28.0	1.0
7	windmill	0.0	4.0	6.0	0.0	1.0	8.0	4.0	27.0

In [58]:
```
print(conf)
```

```
[[31.  7.  0.  2.  3.  3.  2.  2.]
 [ 1. 34.  8.  3.  1.  0.  1.  2.]
 [ 3.  6. 26.  4.  1.  3.  0.  7.]
 [ 2.  4.  2. 26. 14.  1.  1.  0.]
 [ 0.  2.  0.  9. 31.  5.  2.  1.]
 [ 1.  1.  1.  2.  3. 36.  4.  2.]
 [ 1.  5.  2.  1.  4.  8. 28.  1.]
 [ 0.  4.  6.  0.  1.  8.  4. 27.]]
```

In [ ]:
```

```

In [ ]:
```

```

Case5

filter_scales = [1, 2, 3, 5, 10]

K = 200

alpha = 200

L = 2

Accuracy achieved: 56.25%

Confusion Matrix:

[[37. 0. 2. 0. 3. 0. 5. 3.]

[1. 22. 8. 8. 4. 2. 2. 3.]

[1. 4. 22. 3. 2. 3. 4. 11.]

[3. 2. 1. 29. 13. 1. 1. 0.]

[1. 1. 0. 15. 25. 4. 4. 0.]

[0. 0. 3. 2. 2. 42. 1. 0.]

[2. 1. 3. 1. 3. 14. 25. 1.]

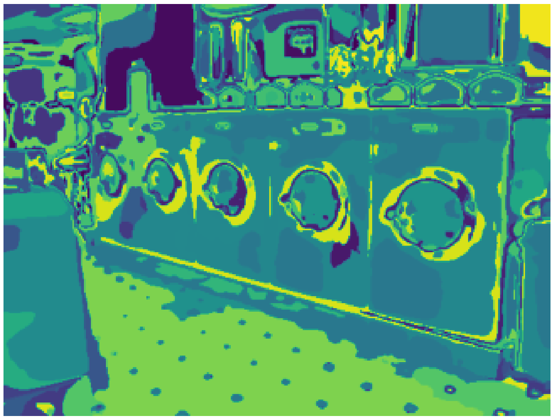
[1. 1. 6. 2. 4. 10. 3. 23.]]

In [63]:
```
opts.filter_scales = [1, 2, 3, 5, 10]
opts.K = 200
opts.alpha = 200
opts.L = 2


compute_dictionary(opts, n_worker = n_cpu)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:792: Futu
reWarning: 'n_jobs' was deprecated in version 0.23 and will be removed in 1.0
(renaming of 0.25).
  warnings.warn("'n_jobs' was deprecated in version 0.23 and will be"
```

Out[63]:
```
array([[ 1.29377155e-01, -5.64801798e-03,  3.10983500e-02, ...,
        -1.31864401e-04,  6.11324617e-06,  6.16579329e-05],
       [ 6.31473490e-02, -1.34501179e-03, -1.62445011e-03, ...,
        -3.17486765e-04,  2.90724553e-05,  1.04276530e-04],
       [ 1.94192204e-01, -2.32278274e-02, -9.90309405e-02, ...,
         1.22239310e-04,  1.18466882e-05,  4.80632956e-04],
       ...,
       [ 1.86110835e-01, -1.04997311e-02,  2.15088356e-01, ...,
        -8.51147174e-05, -3.70065402e-06,  2.69004298e-04],
       [ 1.79280385e-01, -2.30258002e-02,  1.43002145e-02, ...,
        -3.87870244e-04, -3.86769338e-05,  1.62616362e-04],
       [ 1.41869249e-01, -7.27733162e-03, -1.17055719e-01, ...,
         7.62737765e-05, -1.35167612e-05,  2.50842828e-04]])
```

In [64]:
```
dictionary = np.load(join(opts.out_dir, 'dictionary.npy'))
# print(np.shape(dictionary))

img_path = join(opts.data_dir, 'laundromat/sun_aabvooxzwmzzvwds.jpg')
img = plt.imread(img_path) / 255.
wordmap = get_visual_words(opts, img, dictionary)
visualize_wordmap(img, wordmap)
```



In [65]:

```
build_recognition_system(opts, n_worker = n_cpu)
```

```
Shape of Feature:  (1158, 4200)
Shape of Feature:  (1159, 4200)
Shape of Feature:  (1160, 4200)
Shape of Feature:  (1161, 4200)
Shape of Feature:  (1162, 4200)
Shape of Feature:  (1163, 4200)
Shape of Feature:  (1164, 4200)
Shape of Feature:  (1165, 4200)
Shape of Feature:  (1166, 4200)
Shape of Feature:  (1167, 4200)
Shape of Feature:  (1168, 4200)
Shape of Feature:  (1169, 4200)
Shape of Feature:  (1170, 4200)
Shape of Feature:  (1171, 4200)
Shape of Feature:  (1172, 4200)
Shape of Feature:  (1173, 4200)
Shape of Feature:  (1174, 4200)

Shape of Feature:  (1175, 4200)
Shape of Feature:  (1176, 4200)
```

In [66]:

```
conf, accuracy = evaluate_recognition_system(opts, n_worker=n_cpu)

print("Accuracy:", accuracy)
```

```
382 accuracy: 0.5639686684073107
383 accuracy: 0.5651041666666666
384 accuracy: 0.5636363636363636
385 accuracy: 0.5621761658031088
386 accuracy: 0.5633074935400517
387 accuracy: 0.5644329896907216
388 accuracy: 0.5629820051413882
389 accuracy: 0.5641025641025641
390 accuracy: 0.5626598465473146
391 accuracy: 0.5612244897959183
392 accuracy: 0.5623409669211196
393 accuracy: 0.5609137055837563
394 accuracy: 0.5620253164556962
395 accuracy: 0.5606060606060606
396 accuracy: 0.5591939546599496
397 accuracy: 0.5603015075376885
398 accuracy: 0.5614035087719298
399 accuracy: 0.5625
Accuracy: 0.5625
```

In [67]:

```
classes = [
    "aquarium", "desert", "highway", "kitchen",
    "laundromat", "park", "waterfall", "windmill",
]
df = pd.DataFrame(conf, columns=classes)
df.insert(0, "", classes)
df
```

Out[67]:

		aquarium	desert	highway	kitchen	laundromat	park	waterfall	windmill
0	aquarium	37.0	0.0	2.0	0.0	3.0	0.0	5.0	3.0
1	desert	1.0	22.0	8.0	8.0	4.0	2.0	2.0	3.0
2	highway	1.0	4.0	22.0	3.0	2.0	3.0	4.0	11.0
3	kitchen	3.0	2.0	1.0	29.0	13.0	1.0	1.0	0.0
4	laundromat	1.0	1.0	0.0	15.0	25.0	4.0	4.0	0.0
5	park	0.0	0.0	3.0	2.0	2.0	42.0	1.0	0.0
6	waterfall	2.0	1.0	3.0	1.0	3.0	14.0	25.0	1.0
7	windmill	1.0	1.0	6.0	2.0	4.0	10.0	3.0	23.0

In [68]:

```
print(conf)
```

```
[[37.  0.  2.  0.  3.  0.  5.  3.]
 [ 1. 22.  8.  8.  4.  2.  2.  3.]
 [ 1.  4. 22.  3.  2.  3.  4. 11.]
 [ 3.  2.  1. 29. 13.  1.  1.  0.]
 [ 1.  1.  0. 15. 25.  4.  4.  0.]
 [ 0.  0.  3.  2.  2. 42.  1.  0.]
 [ 2.  1.  3.  1.  3. 14. 25.  1.]
 [ 1.  1.  6.  2.  4. 10.  3. 23.]]
```

Q3.2

I did not get the time to implement this but my plan was to pass the images through the function resize_image() which will resize every image to a 256x256 size image, prior to making the dictionary and formulating trained_system. This method is proposed in multiple published papers. I took my inspiration from the following paper:

Ref: Torralba, Antonio, Rob Fergus, and William T. Freeman. "80 million tiny images: A large data set for nonparametric object and scene recognition." IEEE transactions on pattern analysis and machine intelligence 30.11 (2008): 1958-1970.

Prior to this, I also tried to reduce the weights for the images. I kept the hyperparameters fixed as to when I got the maximum accuracy and scaled down the weights assigned. Right now when in layer 3, weights are 1/4(layer 0), 1/4(layer 1) and 1/2(layer 2), I changed it down to 1/8, 1/8 and 1/4 respectively for layers 0, 1 and 2. Since I am scaling down all of the layers by the same factor, there should not be a massive jump in accuracy, but I expected a little bump since it follows a similar idea as the IDF-TF suggested in 3.3. However, what I noticed was my accuracy had dipped to close to 56%. I think this happened because of improper assignment of weights to the 0th and 1st level, or maybe I should have had a combination of weights to adapt to every layer rather than just scaling it down bya single factor all throughout.

In [107]:
```
def resize_image():
    data_dir="../data"
    train_files = open(join(data_dir, "train_files.txt")).read().splitlines(
    imageFile = os.path.join(opts.data_dir,(train_files[0]))
    im1 = Image.open(imageFile)
    width = 256
    height = 256
    # use one of these filter options to resize the image
    im2 = im1.resize((width, height), Image.NEAREST)      # use nearest neig
    im3 = im1.resize((width, height), Image.BILINEAR)     # linear interpola
    im4 = im1.resize((width, height), Image.BICUBIC)      # cubic spline int
    im5 = im1.resize((width, height), Image.ANTIALIAS)

    return im2
```

Alternatively, to increase the speed of the computation, we can implement multiprocessing - process based parallelism (using pool.map)

It's implementation will look somewhat similar to the following:

In [111]:
```
import multiprocesssing

pool = multiprocessing.Pool(num_workers)
pool.map(compute_dictionary_one_image, arguments)
pool.close()
pool.join()

#Similarily for the other functions as well, where num_workers = n_cpu
```

In [ ]:
```

```

Q3.3 (Extra Credit)

The basic idea of this approach is to assign less weights to the patches of images that are very frequent and do not provide a lot of information about how to classify them into different classes. So when we weigh down these common patches and scale up the rare patches, we will have a better classification process, thus improving the accuracy.

In [ ]:

```
def compute_IDF(opts, n_worker=1):
    # YOUR CODE HERE

    K value

    if bins !=  0, then new bin + 1
    np.log(1177/list) = weights[]
    pass

def evaluate_recognition_System_IDF(opts, n_worker=1):
    # YOUR CODE HERE
    pass
```