Q-1.1.1

The gaussian filter smoothens the image and allows to pick the overall behavior of the pixels in a region.
The laplacian of gaussian picks up the edges in the image.
Derivative of gaussian in x picks up vertical lines in the image.
Derivative of gaussian in y picks up horizontal lines on the image.

Multiple filter scales allows us to pick features from a small or large area, hence providing a better way to comprehend the image.

Q-1.1.2

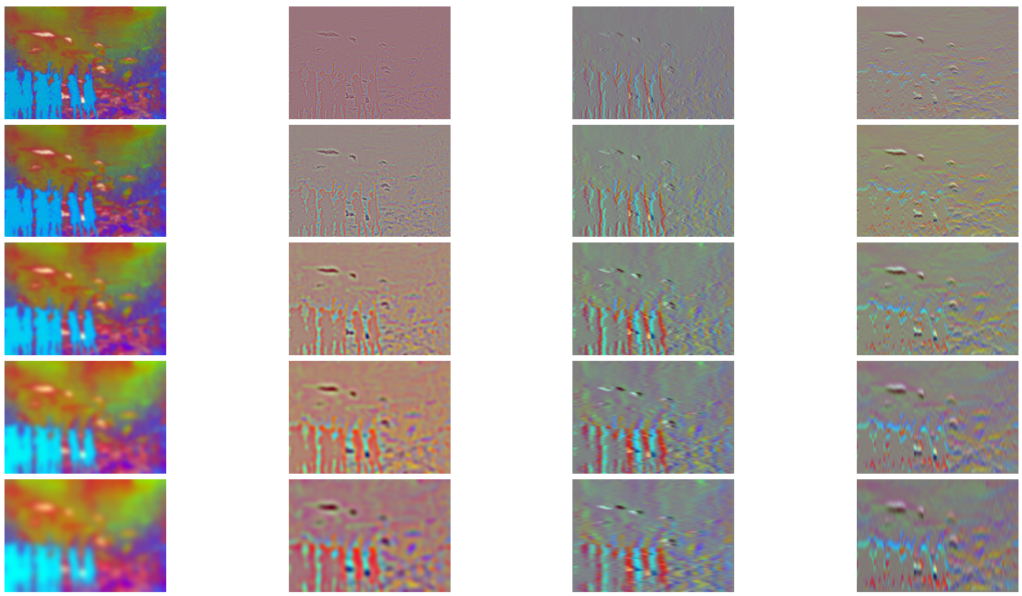


Image: aquarium/sun_aztvjgubyrgvirup.jpg
Scale: [1,2,4,6,8]

Q-1.2

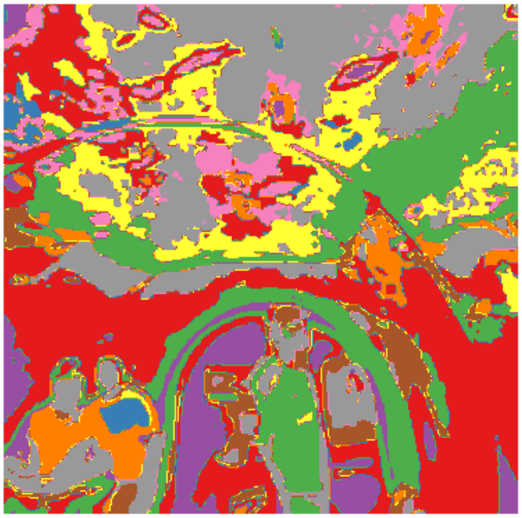
```
def compute_dictionary_one_image(args):
    '''
    Extracts a random subset of filter responses of an image and save it
to disk
    This is a worker function called by compute_dictionary

    Your are free to make your own interface based on how you implement
compute_dictionary
    '''
    image_path, alpha, opts, idx = args
    # print(args)
    img = Image.open(image_path)
    img = np.array(img).astype(np.float32)/255
    filter_response = extract_filter_responses(opts, img)
    x = np.random.choice(filter_response.shape[0], alpha, replace=False)
    y = np.random.choice(filter_response.shape[1], alpha, replace=False)
    pixel_response = filter_response[x,y,:]
    print("(~ ",idx," ~)")
    tmp_dir = '../tmp'
    if not os.path.exists(tmp_dir):
        os.makedirs(tmp_dir)
    np.savetxt(os.path.join(tmp_dir, str(idx) + ".csv"), pixel_response,
delimiter="," )
```

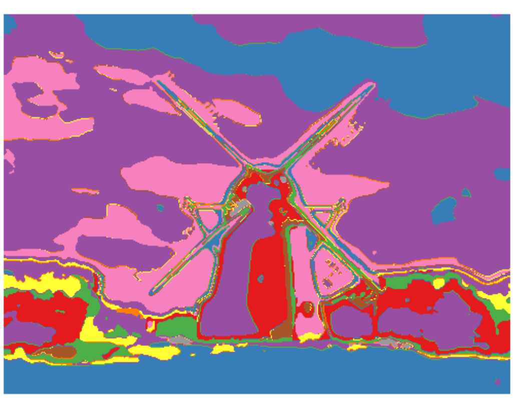
```
def compute_dictionary(opts, n_worker):
    '''
    Creates the dictionary of visual words by clustering using k-means.

    [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
    alpha = opts.alpha

    # For testing purpose, you can create a train_files_small.txt to only
load a few images.
    # train_files = open(join(data_dir,
'train_files.txt')).read().splitlines()
    train_files = open(join(data_dir,
'train_files.txt')).read().splitlines()
    image_path = ["../data/" + x for x in train_files]
    # num_images=len(image_path)

    pool = multiprocessing.Pool(processes=n_worker)
    args = zip(image_path, [alpha]*len(image_path),
[opts]*len(image_path), range(len(image_path)))
    pool.map(compute_dictionary_one_image, args)

    # Starting k-means

    tmp_files = listdir("../tmp")

    pixel_response_path = ["../tmp/" + x for x in tmp_files]
    file0=pixel_response_path[0]
    arr1=np.loadtxt(file0, delimiter=",")
    channel = arr1.shape[1]


    pixel_response_master = np.empty([0,channel])
    for file in pixel_response_path:
        pixel_response = np.loadtxt(file, delimiter=",")
        pixel_response_master=np.concatenate((pixel_response_master,
pixel_response), axis=0)
    # print("pixel_respose_master: ",pixel_response_master.shape)
```

```
    start = timer()
    kmeans =
sklearn.cluster.KMeans(n_clusters=opts.K).fit(pixel_response_master)
    centroids = kmeans.cluster_centers_

    np.save("dictionary.npy", centroids)
```

Q-1.3





The “word” boundaries do make sense. The regions with the same color mean that they map to the same word in the dictionary. Also, regions with the same color mean that they have similar properties, like color(actual), texture, etc.

Q-2.1

```python
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

    wordmap_1d = wordmap.reshape(-1)
    histogram = np.bincount(wordmap_1d, minlength = K)
    histogram = histogram/np.sum(histogram)

    return histogram
```

Q-2.2

```
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




    hist_all = []
    for l in range(L):
        num_tile=2**l
        cols = np.array_split(wordmap, num_tile, axis=1)
        for i in range(len(cols)):
            rows=np.array_split(cols[i], num_tile, axis=0)
            for j in range(len(cols)):
                tile = rows[j]
                hist = get_feature_from_wordmap(opts, tile)
                if l==0 | l==1:
                    weight =1
                else:
                    weight = 2**(-l)
                hist_weighted = hist*weight

                hist_all = np.append(hist_all, hist_weighted, axis=0)

    hist_all = hist_all/np.sum(hist_all)

    return hist_all
```

Q-2.3

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

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

    [output]
    * sim: numpy.ndarray of shape (N)
    '''

    similarity_matrix=np.minimum(word_hist, histograms)
    similarity_score=np.sum(similarity_matrix, axis=1)
    sim=similarity_score

    return sim
```

Q-2.4

```
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
    '''

    data_dir = opts.data_dir
    out_dir = opts.out_dir
    SPM_layer_num = opts.L
    K=opts.K
    L=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'))

    features=np.empty([0,int(K*((4**L)-1)/3)])
    for i in range(len(train_files)):
        img_path=join(opts.data_dir, train_files[i])
        img_features = get_image_feature(opts, img_path, dictionary)
        img_features = np.asarray(img_features)
        img_features = np.reshape(img_features, (1, int(K*((4**L)-1)/3)))


        features = np.concatenate((features, img_features), axis=0)
        print("Train File: ", i)
```

```python
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)/3)
    '''

    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)
    feature_SPM = get_feature_from_wordmap_SPM(opts,wordmap)
    feature = feature_SPM

    return feature
```

Q-2.5

Overall accuracy = 66.25%

Filter scale: [1,2,4]

K: 50

Alpha: 25

L: 3

Confusion matrix:

```
[[42.  0.  0.  2.  2.  0.  1.  3.]
 [ 1. 28.  6.  5.  4.  0.  2.  4.]
 [ 2.  5. 26.  0.  0.  4.  3. 10.]
 [ 0.  0.  0. 39. 10.  0.  0.  1.]
 [ 1.  2.  2. 10. 30.  2.  3.  0.]
 [ 2.  0.  3.  1.  1. 38.  1.  4.]
 [ 6.  1.  1.  1.  5.  4. 30.  2.]
 [ 1.  4.  4.  2.  1.  4.  2. 32.]]
```

Q-2.6

We see a great error on columns/rows 3 and 4. These columns correspond to kitchen and laundromat respectively. They are misclassified because both the categories have similar features. They have similar furniture, indoor, and floor texture.
Another instance of high error is seen at row 2 and column 7, these correspond to highway and windmill images. They too have similar features, such as open areas, trees, sky.

These similarities make the classification difficult.

Q-3.1

Filter scale: [1,2,4,6,8]
K: 50
Alpha: 25
L: 4
Accuracy: 65.5%

Filter scale: [1,2,4]
K: 50
Alpha: 25
L: 4
Accuracy: 65.5%

Filter scale: [1,2,4]
K: 100
Alpha: 25
L: 4
Accuracy: 64.0%