HOMEWORK 1: SPATIAL PYRAMID MATCHING FOR SCENE CLASSIFICATION

16-720A Computer Vision (Spring 2023)
https://canvas.cmu.edu/courses/32966

OUT: Jan 25th, 2023
DUE: Feb 13th, 2023 11:59 PM
Instructor: Deva Ramanan
TAs: Kangle Deng, Vidhi Jain, Xiaofeng Guo, Chung Hee Kim, Ingrid Navarro Anaya

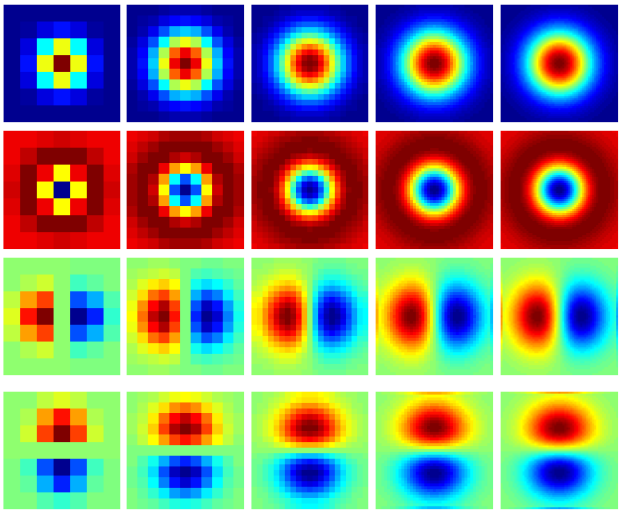


Figure 1.1: Multi-scale filter bank

1 Representing the World with Visual Words

1.1 Extracting Filter Responses

We want to run a filter bank on an image by convolving each filter in the bank with the image and concatenating all the responses into a vector for each pixel. In our case, we will be using 4 types of filters of multiple scales (`opts.filter_scales`). The filters are: (1) Gaussian, (2) Laplacian of Gaussian, (3) derivative of Gaussian in the x direction, and (4) derivative of Gaussian in the y direction.

Q1.1.1 (5 points): (a) What properties do each of the filter functions pick up? (See Fig 1.1) Try to group the filters into broad categories (*e.g.* all the Gaussians).

(b) Why do we need multiple scales of filter responses?

Q1.1.1 (a)(b)

(a) Gaussian filters blur images and remove noise and detail. It has a peak weight to the center pixel. Laplacian of Gaussian function is a second derivative function that highlights regions of rapid intensity change and is usually used for edge detection. Derivative of Gaussian in the x direction and y direction are used to detect vertical edges and horizontal edges respectively.
(b) With multiple scales of filter responses, one is more likely to achieve a better result. Since the scales of features can be different with different target images, using multiple scales of filter responses such as filter banks can pick up the features which fit the best, and thus help to present the feature better for various targets.

Q1.1.2 (10 points):

Q1.1.2

The code is packed in the .zip file. The collage of images is shown below (Fig 1.2).

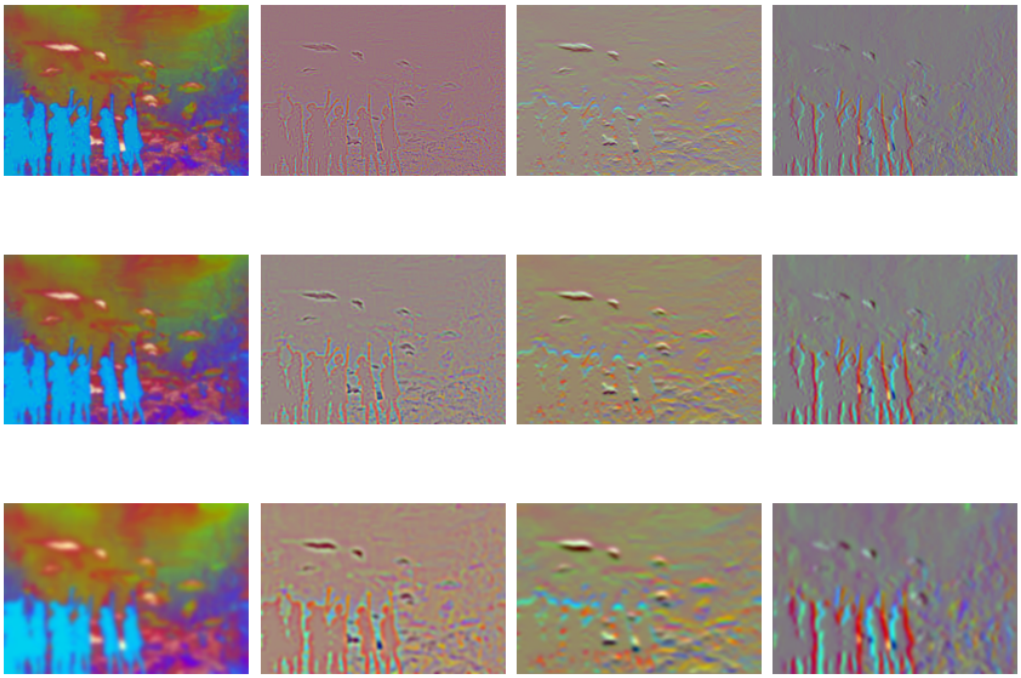


Figure 1.2: filter responses for all of the filters

1.2 Creating Visual Words

Q1.2 (10 points): Write the functions

`visual_words.compute_dictionary(opts, n_worker)`,
`visual_words.compute_dictionary_one_image(args)` (optional, multi-processing),

Include your implemented functions within the `minted` block below `compute_dictionary`, and optionally, `compute_dictionary_one_image` or other customized functions).

Q1.2

```python
from opts import get_opts
import imageio

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
    """

    # Here args is a collection of arguments passed into the function.
    opts = get_opts()
    i, alpha, path = args

    # Inside compute dictionary one image(), you should read an image,
    img = imageio.imread('../data/' + path)
    img = img.astype('float') / 255

    # extract the responses
    filter_responses = extract_filter_responses(opts, img)

    # Random sampling of responses:
    h = filter_responses.shape[0]
    w = filter_responses.shape[1]
    c = filter_responses.shape[2]
    random_sampling = np.reshape(filter_responses, (h * w, c))

    # generate alpha random pixel in the image
    i = np.random.randint(h * w, size=alpha)
    random_sampling = random_sampling[i, :]

    # print('complete one_img')
    return random_sampling

    # Saving the sampled responses to a temporary file
    np.save('%s%d' % (sample_response_path, i), np.asarray(random_sampling))
```

Q1.2 continued

```
def compute_dictionary(opts, n_worker=2):
    """
    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()#train files.txt

    train_data = np.asarray(train_files)  # ['aq/sun_as' 'aq/sun_atk',..]

    T = train_data.shape[0]
    alpha = opts.alpha

    pool = multiprocessing.Pool(processes=n_worker)
    responses = []
    for i in range(0, T):
        args = [(i, alpha, train_data[i])] # args:[(0,25,'aqua/sun_as.jpg')]
        # print()
        responses.append(pool.apply_async(compute_dictionary_one_image, args))

    features = []
    for res in responses:
        features.append(res.get())

    # collect a matrix filter_responses over all images that is alpha*T x 3F
    filter_responses = features[0]
    for i in range(1, len(features)):
        filter_responses = np.concatenate((filter_responses, features[i]),
        ↪  axis=0)

    # run k-means clustering
    kmeans = sklearn.cluster.KMeans(n_clusters=K).fit(filter_responses)
    dictionary = kmeans.cluster_centers_
    print(dictionary.shape)  # 10, 36

    # example code snippet to save the dictionary
    np.save(join(out_dir, 'dictionary.npy'), dictionary)
    return dictionary
```

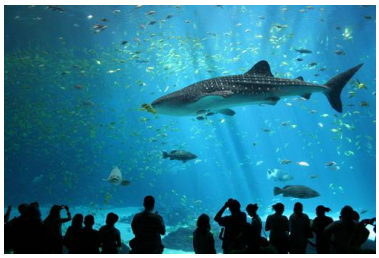
1.3 Computing Visual Words

Q1.3 (10 points):

Q1.3

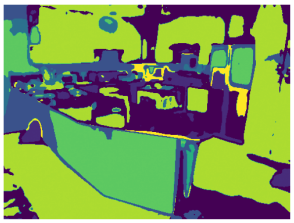
Figure 1.3 shows three wordmaps with their original images on top.

(a) kitchen

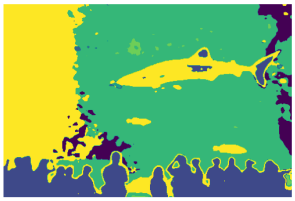


(b) aquarium

(c) laundry room



(d) wordmap of kitchen



(e) wordmap of aquarium

(f) wordmap of laundry room

Figure 1.3: plots of wordmap

2 Building a Recognition System

2.1 Extracting Features

We will first represent an image with a bag of words. In each image, we simply look at how often each word appears.

Q2.1 (10 points): Include your implemented functions within the `minted` block below.

Q2.1

```
# Copy and paste your code here.
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

    # histogram need to be computed over the flattened array.
    flat_wordmap = wordmap.flatten()
    hist, bin_edges = np.histogram(flat_wordmap, bins=K, density=True)
    hist = hist / np.sum(hist)
    # print("success")
    return hist
```

2.2 Multi-resolution: Spatial Pyramid Matching

Q2.2 (15 points):

Q2.2

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

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

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

    K = opts.K
    L = opts.L

    # calculate the weight for each layer
    # layer 0 and 1 is 2^(-L)
    weight = []
    for i in range(0, L + 1):
        if i == 0 or i == 1:
            weight.append(np.exp2(-L))
        else:
            weight.append(np.exp2(i - L - 1))

    hist_all = []
    # split layers to cells (2^l x 2^l)
    for i in range(0, L + 1):
        idx_num = np.exp2(i)
        row_div = np.array_split(wordmap, idx_num, axis=0)
        for r in row_div:
            small_cell = np.array_split(r, idx_num, axis=1)
            for idx in small_cell:
                hist_cell, bin_edges = np.histogram(idx, bins=K)
                # concatenate all list
                hist_all = np.append(hist_all, weight[i] * hist_cell)

    # normalize by all total feature
    hist_all = hist_all / np.sum(hist_all)

    return hist_all
```

2.3 Comparing images

Q2.3

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

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

    [output]
    * sim: numpy.ndarray of shape (N)
    """
    # similarity intersection is the sum mini value of each corresponding bins
    intersect = np.minimum(word_hist, histograms)
    sim = np.sum(intersect, axis=1)

    return sim
```

2.4 Building A Model of the Visual World

Q2.4 (15 points):

Q2.4

```
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)
    """
    # load an image
    img = imageio.imread('../data/' + img_path)
    img = img.astype('float') / 255

    # extract word map from the image
    wordmap = visual_words.get_visual_words(opts, img, dictionary)

    # compute the SPM
    feature = get_feature_from_wordmap_SPM(opts, wordmap)

    # return computed feature
    return feature


def build_recognition_system(opts, n_worker=3):
    """
    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
    """
```

Q2.4 continued

```
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"))

# training data as an array
train_data = np.asarray(train_files)  # ['aq/sun_as' 'aq/sun_atk',..]
N = train_data.shape[0]

K = opts.K

pool = multiprocessing.Pool(processes=n_worker)

response = []
for i in range(0, N):
    args = [opts, train_data[i], dictionary]  # args:[()]
    # print()
    response.append(pool.apply_async(get_image_feature, args))

feat_tmp = []
for res in response:
    feat_tmp.append(res.get())

# feature size M:
M = int(K * (4**(SPM_layer_num+1)-1) / 3)

# reshape features to be N*M
features = np.reshape(feat_tmp,(N, M))
# example code snippet to save the learned system
np.savez_compressed(join(out_dir, 'trained_system.npz'),
    features=features,
    labels=train_labels,
    dictionary=dictionary,
    SPM_layer_num=SPM_layer_num,
)
```

2.5 Quantitative Evaluation

Q2.5 (10 points):

Q2.5 (a) (b)

(a) The confusion matrix is shown in Fig 2.1.

```
C=[[24.  1.  4.  4.  3.  4.  4.  6.]
   [ 1. 23.  4.  5.  6.  2.  2.  7.]
   [ 1.  2. 25.  2.  1.  2.  2. 15.]
   [ 1.  2.  1. 26. 13.  3.  3.  1.]
   [ 0.  2.  1. 10. 27.  1.  7.  2.]
   [ 1.  0.  7.  3.  1. 33.  3.  2.]
   [ 8.  2.  2.  4.  3.  7. 20.  4.]
   [ 3.  3.  9.  2.  4.  5.  6. 18.]]
```

(b) The accuracy is 49%.

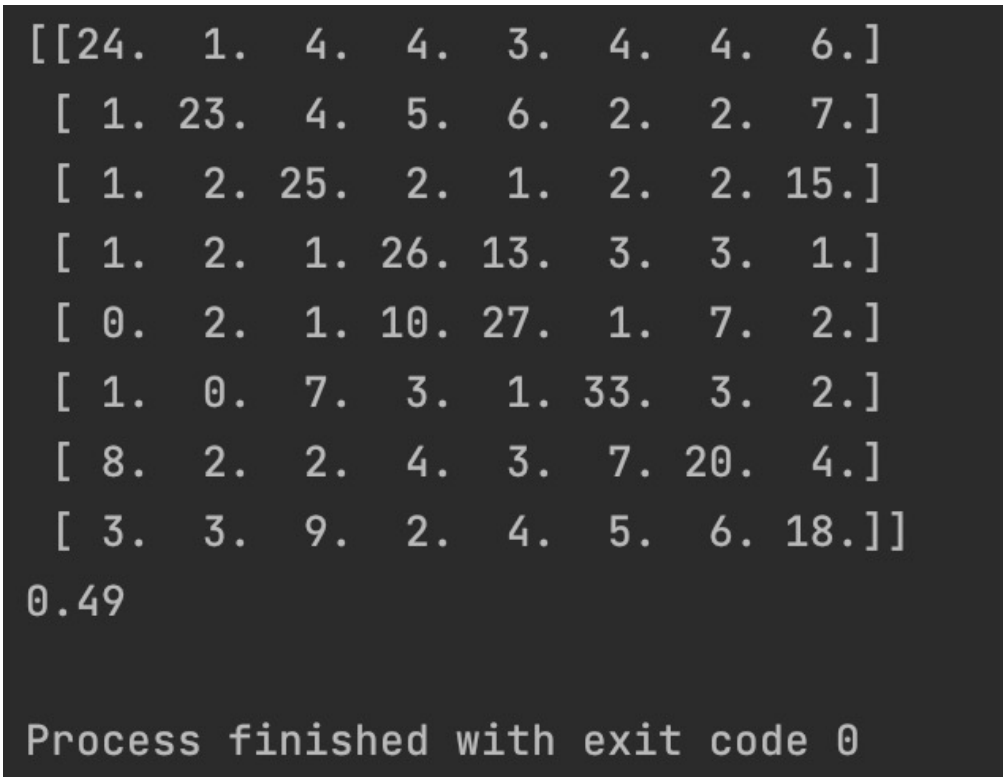


Figure 2.1: Q2.5:(a) Confusion matrix (up). (b) accuracy (bottom)

2.6 Find the failures

Q2.6 (5 points):

Q2.6

When they are mostly matched, the elements on the diagonal of the confusion matrix will be large, and the off-diagonal elements will be small. From the confusion matrix I have, I can clearly see that labels 3 and 4, which are kitchen and laundromat respectively, there will possibly contain misclassification images. Figure 2.2 shows an example of why they are more challenging to distinguish and classify than the other categories and images. Since items in these two categories are likely to have furniture that is squared, they sometimes all contain cabinets in the pictures. When the images have a further view, it is hard to distinguish the difference. The furnishing can also be confusing when classifying images into these two classes.

(a) kitchen

(b) laundromat

(c) wordmap of kitchen

(d) wordmap of laundromat

Figure 2.2: Two difficult classification between class ”kitchen” and class ”laundromat”

3 Improving performance

3.1 Hyperparameter tuning

Q3.1 (a) (b)

(a)

Parameter changed	Old Accuracy	New Accuracy
L(original: 1, new: 3)	49%	50.75%
K(original: 10, new: 30)	50.75%	62.5%
Alpha(original: 25, new: 50)	62.5%	63.5%

I first tune the number of spatial pyramid layers(L) used in feature extraction. Because the default value is set to 1, which is too small. When I increase the L to 3, the accuracy increased from 49% to 50.75%. Based on L=3, I then modified the number of visual words and also the size of the dictionary(K) value from the default of 10 to 20, the accuracy increased to 56.5%. After that, I increased the number of spatial pyramid layers in feature extraction(alpha) from the default 25 to 50, and the accuracy then reaches 63.5%. The final submission of my document has an accuracy of 61.25%.

(b) The number of spatial pyramid layers(L) matters because the default is 1 which only has 2 layers, thus not too much-discarded information about the spatial structure of the image is captured. If I increase L, more valuable and detailed information will be included.
The K value relates to the number of visual words and also the size of the dictionary, thus increasing the K also increases the size of the dictionary which can also contain more information than a lower K value.
Alpha, which is the number of randomized sample pixels we will choose in an image, has a proportional relationship to the accuracy. With more pixels we select from an image, more details will be captured and helps the model to classify the image into the correct class. Filter scales also matter, because increasing the size of filter scales will increase the size of filter bank F.
But there's also a tradeoff for increasing the value, if I increase the value too large, the result can be overfitting.

```
[[38.  2.  1.  1.  2.  1.  3.  2.]
 [ 2. 30.  6.  3.  2.  0.  3.  4.]
 [ 1.  6. 34.  0.  0.  4.  1.  4.]
 [ 3.  3.  1. 34.  6.  0.  3.  0.]
 [ 2.  0.  3. 18. 20.  4.  2.  1.]
 [ 1.  2.  4.  1.  2. 33.  6.  1.]
 [ 7.  1.  1.  2.  0.  8. 30.  1.]
 [ 0.  5. 12.  1.  1.  2.  3. 26.]]
0.6125
```

Figure 3.1: Q3.1 final accuracy

3.2 [Extra Credit] Further improvement

Q3.2 (10 points):

Q3.2

In terms of improving the speed, resizing the images while not discarding too much information might be helpful. Since it will have less information and from a mathematical perspective, the matrix will have fewer dimensions, thus increasing the processing speed. The result is it is hard to distinguish whether the speed is improving, and there's a tradeoff for resizing. We need to make sure when improving the speed of the process, the accuracy is not decreasing in the meantime.

4 HW1 Distribution Checklist

After unpacking `hw1.zip`, you should have a folder `hw1` containing one folder for the data (`data`), one for your code (`code`), and one for the report (`latex`). In the `code` folder, where you will primarily work, you will find:

- `visual_words.py`: function definitions for extracting visual words.
- `visual_recog.py`: function definitions for building a visual recognition system.
- `util.py`: some utility functions
- `main.py`: main function for running the system

The data folder contains:

- `data/`: a directory containing `.jpg` images from the SUN database.
- `data/train_files.txt`: a text file containing a list of training images.
- `data/train_labels.txt`: a text file containing a list of training labels.
- `data/test_files.txt`: a text file containing a list of testing images.
- `data/test_labels.txt`: a text file containing a list of testing labels.

5 HW1 submission checklist

Submit your write-up and code to Gradescope.

- **Writeup.** Please use this provided template for your writeup. The write-up should be a pdf file named <**AndrewId**>**_hw1.pdf**. **You must select the pages of the writeup that correspond to each question**.
- **Code.** The code should be submitted as a zip file named <**AndrewId**>**_hw1.zip**. By extracting the zip file, it should have the following files in the structure defined below.

  When you submit, remove the folder `data/` **and** `feat/` **if applicable, as well as any large temporary files that we did not ask you to create.**

  - <**andrew_id**>**/** # A directory inside .zip file
    - `code/`
      - `dictionary.npy`
      - `trained_system.npz`
      - <!– all of your .py files >
    - `<andrew_id>_hw1.pdf` make sure you upload this pdf file to Gradescope. Please assign the locations of answers to each question on Gradescope.

References