Sridevi Kaza
16-720 Computer Vision A: Homework 1

1.1.1

A Gaussian filter is used to blur images and remove finer details or noise in an image. The Laplacian of a gaussian can help find edges of larger features in an image by first applying a gaussian to remove finer details then applying a Laplacian to pick up edges. The derivative of gaussian in the x direction picks up vertical lines while the derivative of gaussian in the y direction picks up horizontal lines. Using multiple scales for each filter response is useful since different scales allow for different levels of filtering. For example, if we wanted to remove more of the finer details, we could increase the scale of the gaussian. However, if we wanted to preserve more of those features, we could reduce the scale.

1.1.2

Code from visual_words.py:

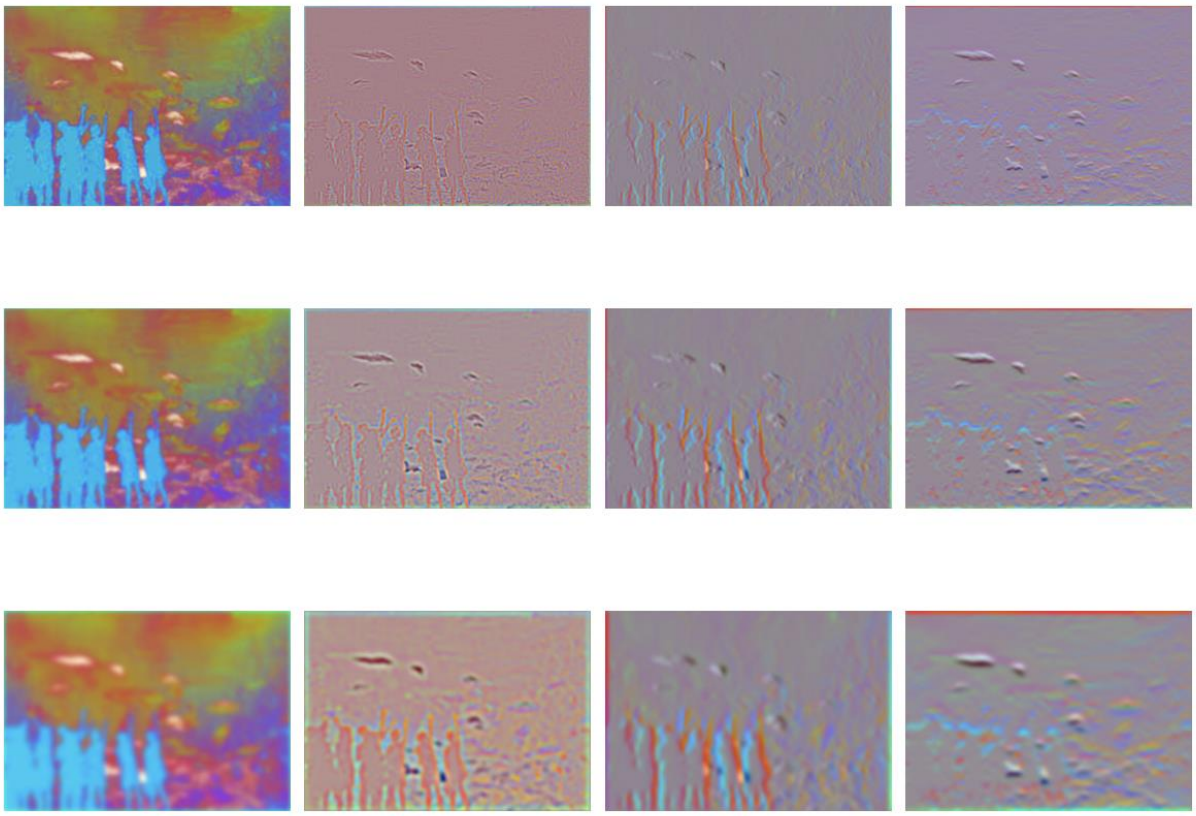
```python
def extract_filter_responses(opts, img):
    """
    Extracts the filter responses for the given image.

    [input]
    * opts    : options
    * img     : numpy.ndarray of shape (H,W) or (H,W,3)
    [output]
    * filter_responses: numpy.ndarray of shape (H,W,3F)
    """
    # make sure image has 3 channels
    if(len(img.shape)<3):
        img = np.dstack((img,img,img))
    if(img.shape[2] == 4):
        img = np.delete(img, 3, 2)
    if(img.shape[2] == 1):
        img = np.dstack((img,img,img))
    # convert to lab
    img = skimage.color.rgb2lab(img)
    filter_scales = opts.filter_scales
    filter_responses = ()
    c1 = img[:,:,0]
    c2 = img[:,:,1]
    c3 = img[:,:,2]

    for scale in filter_scales: # loop through each scale

        gaussian_1 = scipy.ndimage.gaussian_filter(input=c1,sigma=scale,mode='constant')
        gaussian_2 = scipy.ndimage.gaussian_filter(input=c2,sigma=scale,mode='constant')
        gaussian_3 = scipy.ndimage.gaussian_filter(input=c3,sigma=scale,mode='constant')
        gaussian = np.dstack((gaussian_1,gaussian_2,gaussian_3))

        laplacian_1 = scipy.ndimage.gaussian_laplace(input=c1,sigma=scale,mode='constant')
        laplacian_2 = scipy.ndimage.gaussian_laplace(input=c2,sigma=scale,mode='constant')
        laplacian_3 = scipy.ndimage.gaussian_laplace(input=c3,sigma=scale,mode='constant')
        laplacian = np.dstack((laplacian_1,laplacian_2,laplacian_3))

        x_deriv_1 = scipy.ndimage.gaussian_filter(input=c1,sigma=scale,order=(0,1),mode='constant')
        x_deriv_2 = scipy.ndimage.gaussian_filter(input=c2,sigma=scale,order=(0,1),mode='constant')
        x_deriv_3 = scipy.ndimage.gaussian_filter(input=c3,sigma=scale,order=(0,1),mode='constant')
        x_deriv = np.dstack((x_deriv_1,x_deriv_2,x_deriv_3))

        y_deriv_1 = scipy.ndimage.gaussian_filter(input=c1,sigma=scale,order=(1,0),mode='constant')
        y_deriv_2 = scipy.ndimage.gaussian_filter(input=c2,sigma=scale,order=(1,0),mode='constant')
        y_deriv_3 = scipy.ndimage.gaussian_filter(input=c3,sigma=scale,order=(1,0),mode='constant')
        y_deriv = np.dstack((y_deriv_1,y_deriv_2,y_deriv_3))

        filter_responses += (gaussian,laplacian,x_deriv,y_deriv)

    filter_responses = np.dstack(filter_responses)
    return filter_responses
```

Sridevi Kaza
16-720 Computer Vision A: Homework 1

Code from main.py:

```python
def main():
    opts = get_opts()

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

Results:



Sridevi Kaza
16-720 Computer Vision A: Homework 1

1.2:

Code from visual_words.py:

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

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

    [saved]
    * dictionary : numpy.ndarray of shape (K,3F)
    """
    data_dir = opts.data_dir
    feat_dir = opts.feat_dir
    out_dir = opts.out_dir
    alpha = opts.alpha
    K = opts.K
    F = len(opts.filter_scales)*4
    train_files = open(join(data_dir, "train_files.txt")).read().splitlines()
    T = len(train_files)
    filter_responses = ()

    for f in range(len(train_files)): # loop through training images
        file = train_files[f]
        img_responses = compute_dictionary_one_image(opts, file)
        filter_responses += (img_responses,)

    filter_responses = np.vstack(filter_responses) #alphaT x 3F
    kmeans = KMeans(n_clusters=K).fit(filter_responses)
    dictionary = kmeans.cluster_centers_
    # example code snippet to save the dictionary
    np.save(join(out_dir, 'dictionary.npy'), dictionary)
```

```python
def compute_dictionary_one_image(opts, file):
    """
    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
    """
    out_dir = opts.out_dir
    alpha = opts.alpha
    img_path = join(opts.data_dir, file)
    img = Image.open(img_path)
    img = np.array(img).astype(np.float32) / 255
    filter_responses = extract_filter_responses(opts, img) # (H,W,3F)
    H = filter_responses.shape[0]
    W = filter_responses.shape[1]
    three_F = filter_responses.shape[2]
    alpha_responses = np.ones((alpha,three_F))

    for i in range(alpha): # loop through pixels
        h_rand = np.random.randint(0,H-1)
        w_rand = np.random.randint(0,W-1)
        one_pixel_response = filter_responses[h_rand,w_rand,:]
        alpha_responses[i,:] = one_pixel_response
    return alpha_responses
    # np.save(join(out_dir, file), alpha_responses)
```

Code from main.py:

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

Sridevi Kaza
16-720 Computer Vision A: Homework 1

1.3

Code from visual_words.py

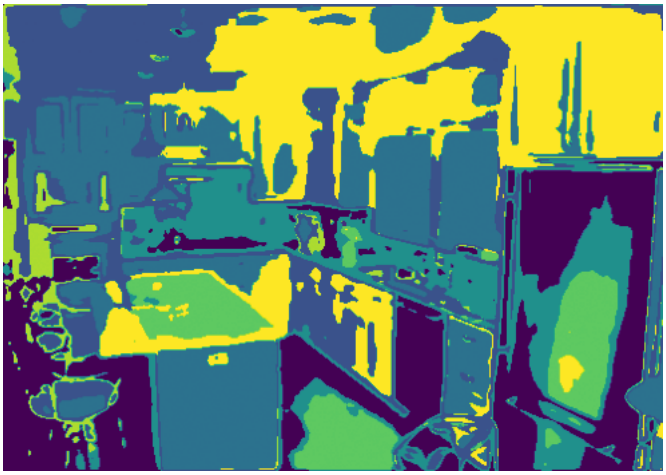
```python
def get_visual_words(opts, img, dictionary):
    """
    Compute visual words mapping for the given img using the dictionary of visual words.

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

    [output]
    * wordmap: numpy.ndarray of shape (H,W)
    """
    filter_size = len(opts.filter_scales)*4*3
    H = img.shape[0]
    W = img.shape[1]
    wordmap = np.ones((H,W))
    filter_responses = extract_filter_responses(opts, img)
    for h in range(H):
        for w in range(W):
            pixel_values = filter_responses[h,w,:]
            pixel_values = np.reshape(pixel_values,(1,filter_size))
            distances = scipy.spatial.distance.cdist(pixel_values, dictionary, metric='euclidean')
            wordmap[h,w] = np.argmin(distances)
    return wordmap
```

Code from main.py:

```python
    # Q1.3
    # img_path = join(opts.data_dir, 'kitchen/sun_afniacozbemafjwf.jpg')
    img_path = join(opts.data_dir, 'kitchen/sun_aasmevtpkslccptd.jpg')
    img = Image.open(img_path)
    img = np.array(img).astype(np.float32)/255
    dictionary = np.load(join(opts.out_dir, 'dictionary.npy'))
    wordmap = visual_words.get_visual_words(opts, img, dictionary)
    util.visualize_wordmap(wordmap)
```

Wordmap Results:

Sridevi Kaza
16-720 Computer Vision A: Homework 1

Above are images of wordmaps with their original RGB images. The “word” boundaries in the wordmaps do make sense. We can see the distinctions between different features or colors through the boundaries of the “words.” For example, in the third image of the park, most of the grass in the same “word” but the trees in the background have different “words.” There are two main “words” for the tree region since the trees change color toward the right side of the image, which is clearly shown by the boundaries. Also, we can see the boundaries corresponding to different “words” where the children are. The two kitchen images also have boundaries that make sense. The boundaries change when the colors or features in the image change. It’s also interesting to see that the two kitchen images mostly share the same “words.” The predominant “words” in the two images are the same, which is a good sign since those should share many features since they’re both kitchen scenes.

Sridevi Kaza
16-720 Computer Vision A: Homework 1

2.1

Code from visual_recog.py:

```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
    hist = np.histogram(wordmap,range(0,K+1))[0]
    return hist/np.sum(hist)
```

Code from main.py:

```python
    # Q2.1
    histog = visual_recog.get_feature_from_wordmap(opts, wordmap)
    hist_plot = plt.bar(x,height=histog)
    plt.show()
```

Sridevi Kaza
16-720 Computer Vision A: Homework 1

2.2

```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
    """
    final = []
    K = opts.K
    L = opts.L
    height = wordmap.shape[0]
    width = wordmap.shape[1]
    # resize wordmap to be divisible by 4
    if (height % 4) != 0:
        height = height - (height % 4)
    if (width % 4) != 0:
        width = width - (width % 4)
    wordmap = wordmap[0:height,0:width]
    l = L
    # loop through layers (starting at finest layer)
    while (l >= 0):
        n = 2**l
        cell_h = height/n
        cell_w = width/n
        # set weight based on layer
        if (l==0 or l==1):
            weight = 2**(-L)
        else:
            weight = 2**(l-L-1)
        #finest layer
        if l == L:
            hist_array = np.ones((n,n,K))
            for w in range(n):
                for h in range(n):
                    starth = int(h*cell_h)
                    endh = int((h+1)*cell_h)
                    startw = int(w*cell_w)
                    endw = int((w+1)*cell_w)
                    cell_image = wordmap[starth:endh,startw:endw]
                    hist = get_feature_from_wordmap(opts,cell_image) #hist for one cell
                    hist_array[h,w,:] = hist
        # all other layers
        else:
            hist_array = np.ones((n,n,K))
            for w in range(n):
                for h in range(n):
                    startw = int(2*w)
                    endw = int(2*w+1)
                    starth = int(2*h)
                    endh = int(2*h+1)
                    add_hists = np.ones((4,K))
                    add_hists[0,:] = prev_hist_array[starth,startw,:]
                    add_hists[1,:] = prev_hist_array[starth,endw,:]
                    add_hists[2,:] = prev_hist_array[endh,startw,:]
                    add_hists[3,:] = prev_hist_array[endh,endw,:]
                    add_hists = np.sum(add_hists,axis=0) # combine 4 hists (add together)
                    normalized_add_hists = add_hists/np.sum(add_hists)
                    hist_array[h,w,:] = normalized_add_hists
        hist_array = hist_array/np.sum(hist_array) #normalize at layer level
        final.append(hist_array.flatten()*weight) #flatten layer level of hists and append to final
        prev_hist_array = hist_array
        l -= 1
    # flatten 3 arrays in final to 1 array
    flat_list = []
    for sublist in final:
        for item in sublist:
            flat_list.append(item)
    return flat_list
```

Sridevi Kaza
16-720 Computer Vision A: Homework 1

2.3

Code from visual_recog.py:

```
159 def similarity_to_set(word_hist, histograms):
160     """
161     Compute similarity between a histogram of visual words with all training image histograms.
162 
163     [input]
164     * word_hist: numpy.ndarray of shape (K)
165     * histograms: numpy.ndarray of shape (N,K)
166 
167     [output]
168     * sim: numpy.ndarray of shape (N)
169     """
170     sim = np.sum( np.minimum(word_hist,histograms)  , axis=1)
171     dist = np.ones(len(sim)) - sim
172     return dist
```

Sridevi Kaza
16-720 Computer Vision A: Homework 1

2.4

Code from visual_recog.py:

```python
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
    hist_size = int(opts.K * (4**(SPM_layer_num+1)-1) / 3)

    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"))

    # get list of SPM histograms for all training images
    N = len(train_files)
    features = np.ones((N,hist_size))
    for i in range(N):
        img_path = join(opts.data_dir, train_files[i])
        features[i,:] = get_image_feature(opts, img_path, dictionary)

    # 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,
    )
```

```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)
    """
    img = Image.open(img_path)
    img = np.array(img).astype(np.float32)/255
    wordmap = visual_words.get_visual_words(opts, img, dictionary)
    return get_feature_from_wordmap_SPM(opts, wordmap)
```

Code from main.py:

```python
    # Q2.3-2.4
    visual_recog.build_recognition_system(opts, n_worker=n_cpu)
```

Sridevi Kaza
16-720 Computer Vision A: Homework 1

2.5

Code from visual_recog.py:

```
176 def evaluate_recognition_system(opts, n_worker=1):
177     """
178     Evaluates the recognition system for all test images and returns the confusion matrix.
179 
180     [input]
181     * opts        : options
182     * n_worker  : number of workers to process in parallel
183 
184     [output]
185     * conf: numpy.ndarray of shape (8,8)
186     * accuracy: accuracy of the evaluated system
187     """
188 
189     data_dir = opts.data_dir
190     out_dir = opts.out_dir
191     trained_system = np.load(join(out_dir, "trained_system.npz"))
192     dictionary = trained_system["dictionary"]
193     features = trained_system["features"]
194     labels = trained_system["labels"]
195 
196     # using the stored options in the trained system instead of opts.py
197     test_opts = copy(opts)
198     test_opts.K = dictionary.shape[0]
199     test_opts.L = trained_system["SPM_layer_num"]
200     test_files = open(join(data_dir, "test_files.txt")).read().splitlines()
201     test_labels = np.loadtxt(join(data_dir, "test_labels.txt"), np.int32)
202     confusion = np.ones((8,8))
203 
204     N = len(test_files)
205     predicted_labels = np.zeros((N))
206     for i in range(N):
207         img_path = join(opts.data_dir,test_files[i])
208         word_hist = get_image_feature(opts, img_path, dictionary)
209         distances = similarity_to_set(word_hist, features)
210         min_index = np.argmin(distances)
211         predicted_labels[i] = int(labels[min_index])
212         print(i)
213         confusion[ int(predicted_labels[i]),test_labels[i] ] += 1
214     accuracy = confusion.trace() / np.sum(confusion)
215     return confusion, accuracy
```

Code from main.py:

```
49     # Q2.5
50     conf, accuracy = visual_recog.evaluate_recognition_system(opts, n_worker=n_cpu)
51 
52     print(conf)
53     print(accuracy)
```

Sridevi Kaza
16-720 Computer Vision A: Homework 1

Default parameters from opts.py:

```
26      # Visual words (requires tuning)
27      parser.add_argument('--filter-scales', nargs='+', type=float,
28                          # default = [1,2,4],
29                          default=[1, 2],
30                          help='a list of scales for all the filters')
31      parser.add_argument('--K', type=int, default=10,
32                          help='# of words')
33      parser.add_argument('--alpha', type=int, default=25,
34                          help='Using only a subset of alpha pixels in each image')
35
36      # Recognition system (requires tuning)
37      parser.add_argument('--L', type=int, default=1,
38                          help='# of layers in spatial pyramid matching (SPM)')
```

Accuracy with default parameters: 50.43%
Confusion matrix:

```
[[33.  1.  2.  3.  3.  3.  8.  5.]
 [ 2. 30.  5.  3.  2.  2.  1.  8.]
 [ 3.  5. 34.  3.  1.  5.  2.  5.]
 [ 4.  7.  1. 35. 14.  2.  2.  1.]
 [ 2.  5.  2.  7. 27.  3.  8.  3.]
 [ 4.  1.  2.  2.  4. 33. 13.  8.]
 [ 5.  3.  2.  3.  4.  5. 20.  6.]
 [ 5.  6. 10.  2.  3.  5.  4. 22.]]
0.5043103448275862
```

Sridevi Kaza
16-720 Computer Vision A: Homework 1

2.6) Based on the results of the confusion matrix, it appears that many of the kitchen scenes are being incorrectly classified as laundromat scenes. A kitchen could easily be misclassified with a laundromat because they have many features in common. For example, the features of the kitchen counter could be easily mistaken with the row of washing machines. The scenes have many similar colors as well, so our algorithm is struggling to differentiate them. Our approach focuses on detecting lower-level features which can be similar across images of different labels. Applying a gaussian to laundromats and to a kitchen counter may produce very similar looking results. Also, our approach lacks the ability to look at a combination of features which could be useful in differentiating between scenes.

Sridevi Kaza
16-720 Computer Vision A: Homework 1

3.1) Table of ablation:

	Changing K	filter scales	L	alpha	K	accuracy
DEFAULT	K = 10	1,2	1	25	10	0.504310345
	K = 30	1,2	1	25	30	0.551724138
HIGHEST	K = 50	1,2	1	25	50	0.592672414
	K = 60	1,2	1	25	60	0.543103448
	K = 100	1,2	1	25	100	0.547413793
	Changing alpha	filter scales	L	alpha	K	accuracy
	alpha = 25	1,2	1	25	50	0.592672414
	alpha = 50	1,2	1	50	50	0.584051724
	alpha = 100	1,2	1	100	50	0.549568966
	Changing L	filter scales	L	alpha	K	accuracy
	L = 1	1,2	1	25	50	0.592672414
	L = 2	1,2	2	25	50	0.577586207
	L = 1	1,2	1	25	100	0.547413793
	L = 2	1,2	2	25	100	0.55387931
	L = 2	1,2,4,8,16	2	150	150	0.581896552
	L = 3	1,2,4,8,16	3	150	150	0.584051724
	L = 3	1,2,4,8,16	3	100	100	0.590517241
	L = 4	1,2,4,8,16	4	100	100	0.568965517
	Changing filter scales	filter scales	L	alpha	K	accuracy
	scales = 1,2	1,2	1	25	50	0.592672414
	scales = 1,2,4	1,2,4	1	25	50	0.558189655
	scales = 1,2,4,8	1,2,4,8	1	25	50	0.560344828
	Testing Higher Input Values					
		filter scales	L	alpha	K	accuracy
		1,2,4,8	2	400	400	0.586206897
		1,2,4,8,8*sqrt(2)	2	200	300	0.581896552
		1,2,4,8,16	3	100	200	0.584051724
		1,2,4,8,16	2	100	100	0.5625
		1,2,4,8,16	2	50	50	0.55387931
		1,2,4,8	2	50	100	0.588362069

L: L represents the number of spatial pyramid layers used in feature extraction. Dividing the image into a smaller number of cells and then concatenating the histograms of each of the cells with appropriate weights allows us to take into account the spatial structure of an image. Increasing L should improve our final accuracy since it increases our ability to associate features with their spatial structure. In my results above, each color represents a grouping of parameters in which I increased L. I found that changing L surprisingly did not have much of an impact on my accuracy. This result may be caused by the weights for the smaller cells not being much higher than the weights for the larger cells. Changing the weighting or structure of my SPM implementation could have possibly led to better results here.

Alpha: Alpha refers to the number of randomly selected pixels from each set of filter banks to use for the K-means clustering and generating a visual words dictionary. Increasing alpha should increase our accuracy since it provides more data to use to detect and predict features. Increasing alpha increases the amount of information we are getting from each image. However, my experimental results did not show much improvement from increasing alpha.

Sridevi Kaza
16-720 Computer Vision A: Homework 1

Possibly, the default value of 25 pixels may have been an adequate number of pixels to gather features from each set of filter banks to create meaningful K-means clusters.

K: Increasing K creates more K-means clusters. From the table above, we can see that increasing K improved the accuracy until a certain point. Increasing the value of K allows for fewer points within each cluster and more total clusters. Having fewer points in each cluster means that the points are closer to each centroid. Therefore, increasing K leads to a higher accuracy since the centroids are better defined. Increasing K from 10 to 50 led to almost a 10% increase in accuracy. However, at a certain point there may be too many clusters which leads to overfitting. From the results, it seems that K=50 is the optimal K value since higher K values led to slightly lower accuracies.

Filter scales: Changing the filter scales used to extract filter responses allows us to scale up or scale down the intensity of our filters. Adding more filter scales of larger values helps us detect higher level features by blurring the images more and filtering out finer details. Utilizing different filter levels can be very useful to add filter responses which capture both lower-level and higher-level features. Using larger filter scales allows us to bring information from a larger region into each individual pixel value. Therefore, adding more filter scales should increase the final accuracy.

Higher Input Values: After studying the effects of varying each of the parameters, I was able to achieve results close to 60%, but the highest result I got was still only 59.2%. I decided to greatly increase my parameter values to see if I could increase the accuracy any more. The higher values showed good results, but most of the values fell around ~58% as shown in the table above. Increasing the values even more after a certain point did not seem to have much of an impact on the accuracy.