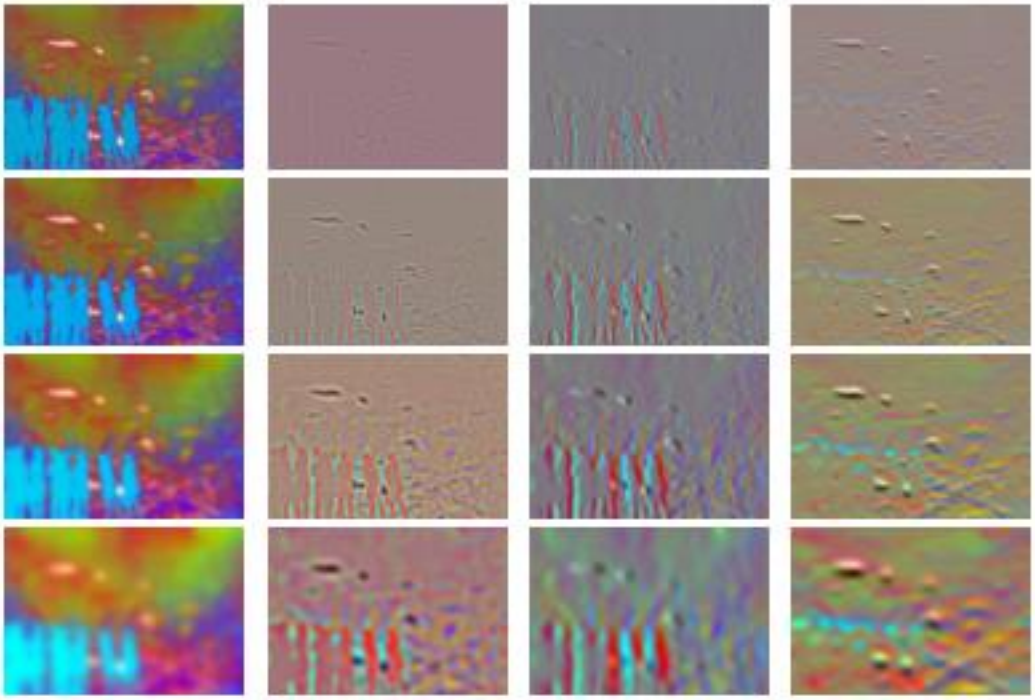
Q1.1.1: What properties do each of the filter functions pick up? Try to group the filters into broad categories (e.g. all the Gaussians). Why do we need multiple scales of filter responses?

- All the gaussians seem to be picking up edges in the image (like the outline of the people) though I think this is mainly seen in the Laplacian. The gaussians (x, y, regular) applied a lot of smoothing of the "roughness" of the image when compared with the Laplacian. The multiple scales allows for different "levels" of smoothness and edge detection and better control of the filtering processing.

Q1.1.2 (Filter scales = [1,2,4,8])

```
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 = [1,2,4,8]

    #Gray-scale images output as 3F channel image
    dimensions = img.shape
    height = img.shape[0]
    width = img.shape[1]
    # channels = img.shape[2]

    if len(dimensions) < 3:
        # What does it mean to duplicate into three channels?
        gray_img = np.stack((img,)*3,axis =2)
        img = gray_img
    else:
        pass

    img = skimage.color.rgb2lab(img)

    #How am I supposed to add to the filter responses if it is not an array?
    # I don't understand what filter responses should initially be?
    # filter_responses = np.zeros(shape = (height,width,3*4))
    filter_scales = [1,2,4,8]

    responses = []
    list_filter = ["Gaussian", "Laplacian", "Dogx", "Dogy"]

    for i in filter_scales:
        #Gaussian Filter
        #How to implement zero padding in the image?
        for filter in list_filter:
            if filter == "Gaussian":
                for rgb in range(0,3):
                    gaussian = scipy.ndimage.gaussian_filter(img[:,:,rgb], i, mode ="reflect")
                    responses.append(gaussian)

        # for rgb in range(0,3):
            #Laplacian filter
            if filter == "Laplacian":
                for rgb in range(0,3):
                    laplacian = scipy.ndimage.gaussian_laplace(img[:,:,rgb], i, mode ="reflect")
                    responses.append(laplacian)
```

```
                    responses.append(laplacian)
        # for rgb in range(0,3):
            #X derivative Gaussian
            if filter == "Dogx":
                for rgb in range(0,3):
                    gaussian_x = scipy.ndimage.gaussian_filter(img[:,:,rgb], i, order=[0,1], mode = "reflect")
                    responses.append(gaussian_x)
            # for rgb in range(0,3):
            #y derivative gaussian
            if filter == "Dogy":
                for rgb in range(0,3):
                    gaussian_y = scipy.ndimage.gaussian_filter(img[:,:,rgb], i, order=[1,0], mode = "reflect")
                    responses.append(gaussian_y)
    filter_responses = np.dstack(responses)

    return filter_responses
```



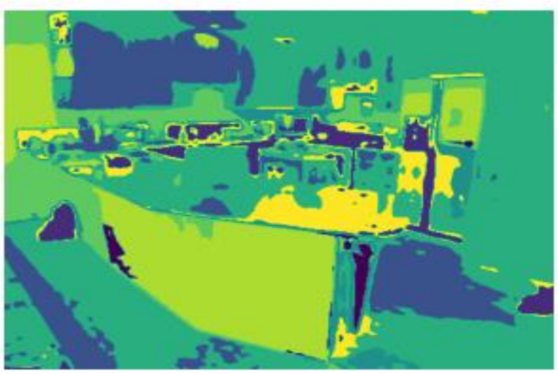
Q1.2

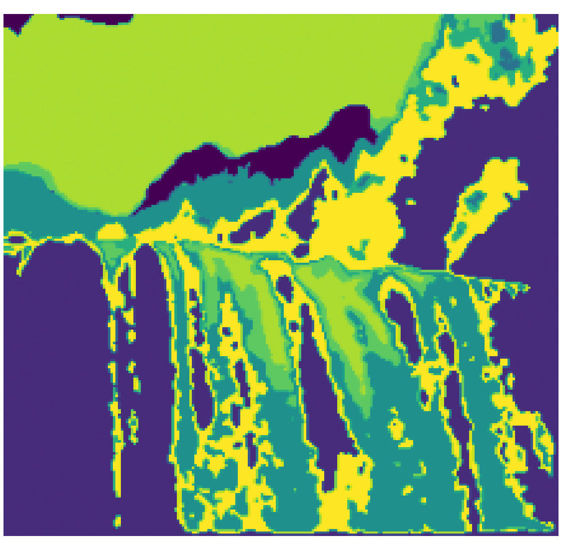
```python
def compute_dictionary_one_image(img_path):
    """
    Extracts a random subset of filter responses of an image and save it to disk
    This is a worker function called by compute_dictionary

    Your are free to make your own interface based on how you implement compute_dictionary
    """

    #Read one image
    img_path = '../data/' + str(img_path)
    img_path = img_path.replace('[','')
    img_path = img_path.replace(']','') #Adding directory to path
    img_path = img_path.replace("'",'')
    img = Image.open(img_path)
    img = np.array(img).astype(np.float32) / 255
    height = img.shape[0]
    width = img.shape[1]

    if len(img.shape) < 3:
        # What does it mean to duplicate into three channels?
        gray_img = np.stack((img,)*3, axis = 2)
        img = gray_img
    #Extracts the responses
    filter_responses = extract_filter_responses(opts, img)

    #Use alphaT to select the filter responses; opts.alpha is 25 (int)
    #Random pixels(How am I supposed to get the values at the alpha random pixels?)
    random_val2 = np.random.randint(0, width,25)
    random_val = np.random.randint(0, height,25)

    #Collect matrix of responses (size alphaTx3F)
    # matrix_of_responses = np.empty((opts.alpha*len(train_files),3*4))
    holder = filter_responses[random_val, random_val2,:]

    #Save to a temporary file
    # temp_file = tempfile.TemporaryFile()
    # temp_file.write(holder)

    # x = []
    # x.append(holder)

    return holder
```

```python
    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)
    """
#Load the training data
    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()


    #Iterate through paths to the image files to read the images
    img_path = []
    x = []

    pool = Pool(processes = (n_worker))

    # manager = multiprocessing.Manager()
    # return_dict = manager.dict()

    for i in range (0,1177):
        img_path.append(train_files[i])
    with multiprocessing.Pool(processes = 4) as pool:
        x = pool.map(compute_dictionary_one_image, img_path)

        #If I do the process this way then this means that there is a lot of processes trying to run in parallel
        # p = multiprocessing.Process(target = compute_dictionary_one_image, args=(img_path, return_dict))
        # x.append(p)
        # pool.apply_async(compute_dictionary_one_image, args=(img_path, return_dict))
        # x.append(p)
        # p.start()

    # pool.close()
    # pool.join()
    # x.append(q.get())
    # print(x)
    # Load the temporary files back

    # matrix_of_responses = np.empty((opts.alpha*len(train_files),3*4))
    matrix_of_responses = np.concatenate(x,axis=0)

    #call k-means
    kmeans = sklearn.cluster.KMeans(n_clusters = K).fit(matrix_of_responses)
    d = kmeans.cluster_centers_

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

Q1.3: Do the “word” boundaries make sense to you? The way the boundaries work is by showing the distribution of words (or better yet image features) in the image. The output displays the pattern of features in the given image.

```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)
    """
    height = img.shape[0]
    width = img.shape[1]
    #Each pixel in wordmap is assigned the closest visual response
    # at the respective pixel in img
    filter_responses = extract_filter_responses(opts,img)
    filter_responses = np.reshape(filter_responses,((height*width),48)) # Resahpe the filter responses format: (a - array, (new shape dimensions))

    # import pdb; pdb.set_trace()


    wordmap = np.zeros(img.shape[0:2])


    # responses_size = np.array([])
    # responses_size = np.reshape(responses_size,[filter_responses.shape[0],filter_responses.shape[1]])

    # import pdb; pdb.set_trace()
    smallest_distance_comp = scipy.spatial.distance.cdist(filter_responses,dictionary, metric = "euclidean")
    wordmap = np.argmin(smallest_distance_comp, 1)
    wordmap = np.reshape(wordmap,(height,width))


    return wordmap
```






Triple
Load

Q2.1

```
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
    # print(K)

    # np.histogram represents rectangles of the same horizontal size corresponding to class intervals called bins
    # Variable height corresponds to frequency

    size = 11 #Array is 1 size bigger than K as histogram size is returning K-1
    histogram = np.histogram(wordmap,range(0,K+1))[0] #This returns two arrays so we want the first

    # normalization

    row_sum = histogram.sum(axis=0)

    new_matrix = []
    for i in range(0,len(histogram)):
        new_matrix.append(histogram[i]/row_sum)
        # print(histogram[i])
        # print(row_sum)
    histogram = new_matrix

    if np.isfinite(histogram).all() == True:
        np.nan_to_num(histogram, copy = True, nan = 0.0, posinf = None, neginf = None)
    else:
        pass

    return histogram
```

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
    """

    # divide the image into a small number of cells, and concatenate the histogram of each
# of these cells to the histogram of the original image, with a suitable weight
    K = opts.K #K is the visual words
    L = opts.L #There are L+ 1 Layers

    # hist_all = np.ndarray(shape = ((K*4**(L+1)-1)/3))
    # Computation of histogram for the finest layers

    first_histogram = np.array([])
    weighted_gram = []

    hist_all = []

    # #First we need to make patchs of the image
    # #Computing first layer

    for i in range(0,2**L):
        for s in range(0,2**L): #If I only seperate based on columns then it returns entire column and not a patch
            patch_one_height = int(wordmap.shape[0]/(2**L))
            patch_two_width = int(wordmap.shape[1]/(2**L))
            first_histogram = np.append(first_histogram,
                                        get_feature_from_wordmap(opts,wordmap[patch_two_width*s:((s+1)*patch_two_width),i*patch_one_height:(i+1)*patch_one_height]))
    row_sum = first_histogram.sum(axis=0)

    # Need to look at first_histogram[0,0];[0,1];[1,0];[1,1]


    # #Histogram is now in array of first_histogram
```

```python
    #Normalizing

    new_matrix = []
    for i in range(0,len(first_histogram)):
        new_matrix.append(first_histogram[i]/row_sum)
    first_histogram = new_matrix

    #Weighting
    #Set the weight of 0 and 1 to 2^(-L); others 2^(l-L-1)
    for l in range(L): #l is layer number
        if l == 0 or l == 1:
            weight = 2**(-L)
        if l != 0 or l != 1:
            weight = 2**(l-L-1)
        for i in range(0,len(first_histogram)):
            weighted_gram.append(first_histogram[i]*weight)
    first_histogram = weighted_gram
    hist_all = np.append(first_histogram,hist_all) #Save first value

    first_histogram = np.reshape(first_histogram, [2,2,10]) #reshape into array to get [i,j]

    # print(first_histogram[0,0])

    #Remaining layers; now we are going through top and bottom of patch
    new_weight = []
    next_histogram = []
    for k in range(0,L): # To compute next layers
        c = np.array([1,1,K])
        for i in range(0,1):
            for j in range(0,1):
                row_z_c_z = first_histogram[i*(2**L),j*(2**L)]
                row_z_c_t = first_histogram[i*(2**L),j*(2**L)+1]
                row_t_c_z = first_histogram[i*(2**L)+1,j*(2**L)]
                row_t_c_t = first_histogram[i*(2**L)+1,j*(2**L)+1]
                c = row_z_c_z+row_z_c_t+row_t_c_z+row_t_c_t
                # y = np.concatenate((row_z_c_z+row_z_c_t+row_t_c_z,row_t_c_t))
                next_histogram.append(c) #Next patch to compute

                #Normalize again
                new_matrix = []
```

```
            for i in range(0,len(next_histogram)):
                new_matrix.append(next_histogram[i]/row_sum)
            next_histogram = new_matrix

            #     #Weight again
            for l in range(0,L): #l is layer number
                if l == 0 or l == 1:
                    weight = 2**(-L)
                if l != 0 or l != 1:
                    weight = 2**(l-L-1)
                for i in range(0,len(next_histogram)):
                    new_weight.append(next_histogram[i]*weight)
            next_histogram = new_weight
        first_histogram = np.reshape(next_histogram,[1,1,K])
        hist_all = np.append(next_histogram,hist_all)

        # first_histogram =
        # final_hist = next_histogram #Compute next layer
        # #After weighing again
        # hist_all = np.append(final_hist)

#     #Weight again
#     for l in range(0,L,-1): #l is layer number
#         if l == 0 or l == 1:
#             weight = 2**(-L)
#         if l != 0 or l != 1:
#             weight = 2**(l-L-1)
#         for i in range(0,len(next_histogram)):
#             new_weight.append(next_histogram[i]*weight)
#     next_histogram = new_weight
#     next_histogram_flat = next_histogram.flatten()
#     hist_all = np.append(next_histogram_flat,hist_all)
# print(hist_all.shape)
return hist_all
```

Q2.3

```
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)
    """

    # q = Computes histogram intersection similarity and each training sample as a vector of length T
    #returns 1 - q as a distance
    q = np.minimum(word_hist,histograms) #Compute distance between word_hist and histgram arrays
    sim = 1 - np.sum(q,1) #Add together for images; avoid for loops

    return sim
```

Q2.4

```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
    K = opts.K
    L = opts.L
    #Loading the training images
    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"))
    M = int((K*(4**(L+1)-1))/3)
    #Need to get dictionary (already loaded)
    #Get features (matrix with all of the histograms of training images)
    features = np.zeros([len(train_files),M])
    # features = np.zeros((len(train_files),M))
    # Labels with the labels of each training images features[i] = label_labels[i]
    argument_parameters = []

    for i in enumerate(train_files):
        #print(type(train_files)) - list
        img_path = str(i[1:])
        img_path = img_path.replace('(','')
        img_path = img_path.replace(')','') #Adding directory to path
        img_path = img_path.replace("'",'')
        img_path = img_path.replace(",",'')
        img_path = join(data_dir, img_path)
        argument_parameters.append((opts,img_path,dictionary))
```

```python
    with multiprocessing.Pool(processes = 4) as pool:
        # x = pool.map(get_image_feature, zip((opts,img_paths,dictionary)))
        x = pool.starmap(get_image_feature, argument_parameters)
        # x.close()
        # x.join()
    # for i in range(0,len(x)):
    #     features = np.append(features[i][0])

    for i in range(0,len(x)):
        features.append(x[i][0])
        # for j in range(0,len(x)):

    SPM_layer_num = opts.L
    # SPM_layer_num?

    # 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,
    )
```

Q2.5: Confusion Matrix and Accuracy

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

    [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"]

    test_files = open(join(data_dir, "test_files.txt")).read().splitlines()
    test_labels = np.loadtxt(join(data_dir, "test_labels.txt"), np.int32)
    ''''Compute the test image's distance to every image
    in the training set;
    2.Return the label of the closest training image
    3. Quantify accuracy with confusion matrix C(ij)
    '''
    C = np.zeros((8,8))
    img_paths =[]
    # Confusion matrix: C = 8x8
    for i in range(0,len(test_files)):
        features = get_image_feature(opts, img_paths[i],dictionary) #Get features
        similarities = similarity_to_set(features,trained_system['features']) #Get distances
        img_paths.append(join(data_dir,str(test_files[i]))) #Get image path

        # print(np.argmin(similarities))

        estimated = trained_system['labels']
        pro = estimated[np.argmin(similarities)]

        real_values = test_labels[i]

        C[real_values,pro] = C[real_values,pro] +1

    accuracy = np.trace(C)/np.sum(C)
    return C,accuracy
```

```
[[33.  1.  2.  4.  1.  2.  3.  4.]
 [ 0. 27.  6.  7.  5.  0.  2.  3.]
 [ 1.  4. 30.  1.  0.  1.  2. 11.]
 [ 4.  3.  2. 34.  5.  1.  0.  1.]
 [ 3.  2.  1. 12. 23.  6.  1.  2.]
 [ 2.  1.  6.  0.  3. 30.  4.  4.]
 [ 6.  0.  2.  1.  6. 11. 21.  3.]
 [ 3.  7.  6.  0.  1.  4.  7. 22.]]
0.55
```

Q2.6: List some hard classes/samples that are difficult to classify using the bags-of-words approach, and discuss why they are more difficult than the rest

Samples of images where the image may have objects in different positions and orientations. Or images that are in gray scale (will not return be able to accurately quantify color differences unlike human perception). For instance, in class an image of Einstein was used to try and detect where his eyes (sample) were located in the image which is easier to do when the image is in color but once it is converted to grayscale the sampling feature (what was used to detect the eyes in the image) can no longer accurately detect areas of “white (or lighter color)” to indicate whether the feature (word) “eyes” is in the image.