Homework 2 Writeup(20196055 Aashish Waikar)

Instructions

- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- There is no page limit.

In the beginning...

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. See Equation 1.

$$a = b + c \tag{1}$$

My code and explanations

1)My code for HoG feature descriptor in feature_extraction.py. I have used the existing functions for HoG descriptor and then reshaped h so that the horizontal dimension becomes 36.

```
1 # Your code here. You should also change the return value
    .
2         hog = cv2.HOGDescriptor(win_size,block_size,
             block_stride,cell_size,nbins,deriv_aperture,
             win_sigma,histogram_norm_type,l2_hys_threshold
             ,gamma_correction,nlevels)
3         h = hog.compute(img)
4
5         h=np.reshape(h, (-1,36))
6         return h
7         #return np.zeros((1500, 36))
```

2)My code for SIFT feature descriptor in feature_extraction.py. I have used the existing functions for SIFT descriptor with grid size 20 x 20.

```
# Your code here. You should also change the return value.
        sift = cv2.xfeatures2d.SIFT_create()
        grid = 20
        kp = [cv2.KeyPoint(x, y, grid) for x in range(0, img.shape[0], grid) for y in range(0, img.shape[1], grid)]
        feat = sift.compute(img, kp)
        return feat[1]
        #return np.zeros((1500, 128))
```

3)My code for Kmeans in kmeans_clustering.py. I have first initialised the vocabulary randomly with vectors from all_features. Then, in each loop, I first cluster the features using existing vocabulary(assign each feature to the cluster with closest centroid) and then recompute the centroids and update those in the vocabulary. The vocabulary is returned either when the max_iterations are complete or when the distance b/w centroids in consecutively computed vocabularies becomes less than epsilon.

```
# Your code here. You should also change the return value.
    #choosing random centroids
    #print(all_features.shape)

    total_pts=len(all_features[:,0])
    rand_arr=np.random.choice(total_pts, vocab_size, replace=False)
    rand_arr.sort()
    vocab_matrix=np.zeros((vocab_size, all_features.shape[1]))
    class_arr=np.zeros(total_pts)

    # #initialised vocab matrix
    ###########
    for i in range(vocab_size):
        vocab_matrix[i]=all_features[rand_arr[i]]

    #print(vocab_matrix)

    for r in range(max_iter):
        #assign each pt to a cluster
        dist_mat = pdist(all_features, vocab_matrix)
        c1=0
        for vec in dist_mat:
            index = np.argmin(vec)
            class_arr[c1]=index
            c1+=1
```

```
26
27          #finding new centroids
28          class_sum=np.zeros((vocab_size,all_features.shape
                [1]))
29          class_count=np.zeros(vocab_size)
30          new_vocab=np.zeros((vocab_size,all_features.shape
                [1]))
31
32          for i in range(total_pts):
33              cur_class=class_arr[i]
34              #print(int(cur_class))
35              class_sum[int(cur_class)]+=all_features[i]
36              class_count[int(cur_class)]+=1
37
38          for i in range(vocab_size):
39              if class_count[i]==0:
40                  new_vocab[i]=vocab_matrix[i]
41              else:
42                  new_vocab[i]=class_sum[i]/class_count[i]
43          #print(new_vocab)
44          dist_mat1 = pdist(vocab_matrix,new_vocab)
45
46          for c3 in range(vocab_size):
47              if dist_mat1[c3][c3]>epsilon :
48                  break
49
50          vocab_matrix=new_vocab
51          #print(vocab_matrix)
52          if c3==vocab_size-1:
53              break
54
55      return vocab_matrix
```

4)My code for PCA in get_features_from_pca.py. I have calculated PCA using the technique of Eigen-value decomposition. First, compute the mean and centre the vectors with respect to it. Then compute the covariance matrix and its eigen value decomposition. Then choose 2 or 3(whichever final dimensionality chosen) eignvectors with the maximum variance and compute reprojections.

```
1  # Your code here. You should also change the return value
       .
2
3       m=np.mean(vocab.T,axis=1)
4       cent_m=vocab-m
5       covar=np.cov(cent_m.T)
6       val, vec=np.linalg.eig(covar)
7       l=val.tolist()
```

```
    tup=[l.index(x) for x in sorted(l, reverse=True)[:feat_num]]
    red_vec=np.zeros((vocab.shape[1],feat_num))
    for i in range(feat_num):
        red_vec[:,i]=vec[:,tup[i]]

    P=red_vec.T.dot(cent_m.T)
    return P.T
    #return np.zeros((vocab.shape[0],2))
```

5)My code for Bag of words in get_bag_of_words.py. I iterate over all images using a for loop. Then, for each image, I compute features using feature_extraction.py and then cluster them using the existing trained vocabulary. I compute a new array whose each element represents the frequency of that corresponding vocabulary vector in that image. Finally, these arrays computed in each iteration are stacked with axis = 0 to result in the final matrix which is returned.

```
    if feature == 'HoG':
        vocab = np.load('vocab_hog.npy')
    elif feature == 'SIFT':
        vocab = np.load('vocab_sift.npy')

    vocab_size = vocab.shape[0]
    ft_size = vocab.shape[1]

    output_mat=np.zeros((image_paths.shape[0],vocab_size))

    # # Your code here. You should also change the return value.
    i=0
    for path in image_paths:
        #dealing with one image
        img = cv2.imread(path)[:, :, ::-1]
        #total_ft*featurevect_length
        features = feature_extraction(img, feature)
        distance_mat = pdist(features,vocab)
        for vec in distance_mat:
            index = np.argmin(vec)
            output_mat[i][index]+=1
        output_mat[i] = output_mat[i] / linalg.norm(output_mat[i])
        i=i+1

    return output_mat
```

6)My code for Spatial Pyramid in get_spatial_pyramid_feats.py. Here, the change is that in each iteration, I compute the histograms for each image and its set of subimages according to max_level. All these histograms are appended to result in 1 single array for each iteration. This array represents the features for the current image(we hope that it has some spatial information now). These arrays are stacked vertically to result in the final matrix.

```
    # Your code here. You should also change the return value.
    ft_size = vocab.shape[1]

    output_mat=np.zeros((1,vocab_size))

    # # Your code here. You should also change the return value.
    i=0
    for path in image_paths:
        #dealing with one image
        img = cv2.imread(path)[:, :, ::-1]
        hor = img.shape[1]
        ver = img.shape[0]

        f_image_mat=np.zeros((1,1))
        for l in range(max_level+1):
            hstep = math.floor(hor/(2**l))
            vstep = math.floor(ver/(2**l))
            x, y = 0, 0
            for c1 in range(1,2**l + 1):
                x = 0
                for c2 in range(1, 2**l + 1):
                    features = feature_extraction(img[y:y+vstep, x:x+hstep], feature)
                    #print("type:",desc is None, "x:",x,"y:",y, "desc_size:",desc is None)
                    distance_mat = pdist(features,vocab)
                    row = np.zeros(vocab_size)
                    for vec in distance_mat:
                        index = np.argmin(vec)
                        row[index]+=1
                    weight = 2**(l-max_level)
                    f_row_mat = np.zeros((1,vocab_size))
                    f_row_mat[0] = weight*row
                    f_image_mat = np.append(f_image_mat, f_row_mat, axis=1)

                    x = x + hstep
                y = y + vstep
```

```
        if i==0:
            output_mat = np.append(output_mat,
               f_image_mat[:,1:], axis=1)
            output_mat = output_mat[:,vocab_size:]
        else:
            output_mat = np.append(output_mat,
               f_image_mat[:,1:], axis=0)
        i+=1
    #output_mat = output_mat[1:,:]
    output_mat = (1 / 3) * (4 ^ (max_level + 1) - 1) *
       output_mat

    #print(output_mat)
    return output_mat
```

7)My code for svm in svm_classify.py. Here, I loop over all the categories and then train svm classifier for the current category vs all other othegories. I initialise the classifier with kernel=kernel_type which is either 'linear' or 'rbf' and use C=0.025. I varied C and chose this as it gave good performance. Then, I train the svm with training data and compute the confidence score using classifier.decision_function(). Finally, the category which gets the highest confidence for the image is the predicted label.

```
    # Your code here. You should also change the return
       value.

    pred_labels = train_labels
    cfd_matrix = np.zeros((test_image_feats.shape[0], len
       (categories)))
    i=0

    for categ in categories:
        new_train_labels1 = np.zeros(len(train_labels))
        new_train_labels1 = np.where(train_labels==categ
           ,1,new_train_labels1)
        classifier = svm.SVC(random_state=0,C=0.025,
           kernel=kernel_type)
        classifier.fit(train_image_feats,
           new_train_labels1)
        cfd_arr = classifier.decision_function(
           test_image_feats)
        cfd_matrix[:,i] = cfd_arr
        i+=1

    j=0
    for vec in cfd_matrix:
        index = np.argmax(vec)
        pred_labels[j] = categories[index]
```

```
        j+=1


    return pred_labels
```

PCA result

1. Result 1(HOG) (left)
2. Result 2(SIFT) (right)

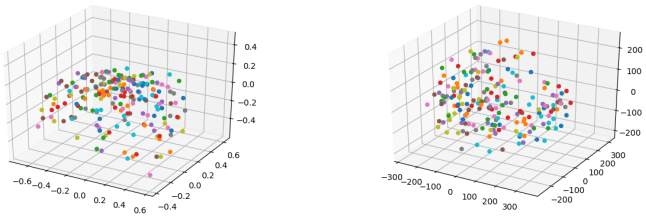


Figure 1: *Left:* HOG *Right:* SIFT

Recognition accuracy of SIFT vs HOG(linear kernel and spatial pyramid) 1.

Condition	accuracy
SIFT	0.725
HOG	0.691

Table 1: SIFT vs HOG

Recognition accuracy of SIFT vs HOG(linear kernel and bag of words) 2.

Condition	accuracy
SIFT	0.635
HOG	0.559

Table 2: SIFT vs HOG

Recognition accuracy of linear vs rbf(using sift and bag of words with spatial pyramid) 3.

Recognition accuracy of bag of words without and with spatial pyramid (using sift and linear kernel) 4.

Condition	accuracy
linear	0.725
rbf	0.707

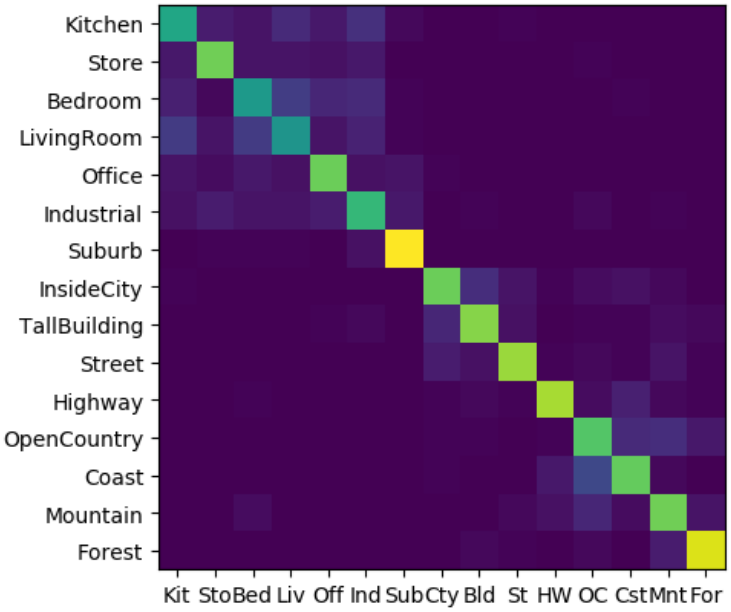
Table 3: linear vs rbf

Condition	accuracy
without_spatial_pyramid	0.635
with_spatial_pyramid	0.725

Table 4: with andd without spatial pyramid

Confusion matrix(for spatial pyramid with sift features and rbf kernel)

1. Confusion matrix with accuracy 0.707



The result in stored in file index.html