Homework 2 Writeup

1 Implementation Detail

1.1 Feature extraction

1.1.1 HOG - Histogram of Oriented Gradient

The HOG feature extraction method divide the images into small cells. In this case, the size of cell is 16*16. Then it calculate the gradient of each pixel and get the gradient histogram of each cell. Each block is composed of cells, so the feature descriptors of all cells in a block are connected in series to obtain the HOG feature descriptor of the block. Finally we can get the descriptor of all the blocks. The output of hog.compute is a vector of 1 row, therefore, we need to reshape it in order to obtain the vector which has the same dimension as each descriptor.

```
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)

dsize = hog.getDescriptorSize()
descriptors = hog.compute(img,winStride=(32,32),padding=(0,0))
descriptors = descriptors.reshape(-1,dsize)
```

1.1.2 SIFT - Scale invariant feature transform

SIFT feature extraction method returns the keypoint of each image. We split the original image to 20 * 20 grid and extract the feature for each sub-image. The dimension of SIFT feature is 128.

```
sift = cv2.xfeatures2d.SIFT_create()
descripters = []
height= img.shape[0]
width = img.shape[1]
split1 = np.array_split(img, width/20, axis=1)
for split in split1:
    split2 =np.array_split(split, height/20, axis=0)
    for ig in split2:
        keypoints, descripter = sift.detectAndCompute(ig,None)
        if descripter is not None:
            descripters.append(descripter)
descripters = np.vstack(descripters)
```

1.2 K-means Clustering

After obtaining features, we are going to use the unsupervised classification algorithm to separate features in 200 clusters and get the center of each cluster.
The initial centers are chose randomly. For each iteration, we calculate the distance between each data from each center and classify each point to the nearest center. The result are saved in the vector ”clusters”. Then, we calculate the new centers by take the average of all data of one cluster and save the difference of new centers and old centers in variable ”error”.
The iteration possess two stopping condition, when the error is smaller than a threshold which means the calculation of centers converge to a stable result, or the number of iteration is over maximum.
We have to pay attention to the empty centers which has no data belongs to it. Those empty centers are deleted from the list of centers.

```
n = all_features.shape[0]
dim = all_features.shape[1]

mean = np.mean(all_features, axis = 0)
std = np.std(all_features, axis = 0)
centers = np.random.randn(vocab_size,dim)*std + mean #
                                    initialisation randomly

centers_old = np.zeros(centers.shape)
centers_new = centers

error = np.linalg.norm(centers_new - centers_old)

distances = np.zeros((n,vocab_size))
clusters = np.zeros(n)

iteration = 0
while error >= epsilon and iteration < max_iter :
    for i in range(vocab_size):
        distances[:,i] = np.linalg.norm(all_features - centers_new
                                    [i], axis = 1)
    clusters = np.argmin(distances,axis = 1)
    centers_old = centers_new
    for i in range(vocab_size):
        centers_new[i] = np.mean(all_features[clusters == i],axis
                                    = 0)

    error = np.linalg.norm(centers_new - centers_old)
    iteration = iteration + 1

#delete empty centers
nans = []
for i in range(vocab_size):
    if np.isnan(centers_new[i,0]):
        nans.append(i)
centers_new = np.delete(centers_new, nans, axis=0)
```

1.3 Get features in PCA

This PCA algorithm reduces the dimension of data. Firstly, the data should be normalized. Then, find the two(or other feature number) largest eigenvector of covariance matrix of data. np.argsort() function sorts data in increase order. Finally, we can get the dimension reduction vocabulary by multiply original vocabulary and eigenvectors. Therefore, the input size of vocabulary is (200 * 36) for HOG feature extraction and the output would be (200 * 2). The visualization of PCA is based on HOG feature extraction.

```
    avr = np.mean(vocab,axis = 0)
    vocab_norm = vocab - avr

    covMat = np.cov(vocab_norm,rowvar = 0)

    eigVals,eigVects = np.linalg.eig(np.mat(covMat))

    i = np.argsort(eigVals)
    feat_num_indice = i[-1:-(feat_num+1):-1]
    feat_num_eigVect = eigVects[:,feat_num_indice]
    reduction_vocab = vocab_norm * feat_num_eigVect
```

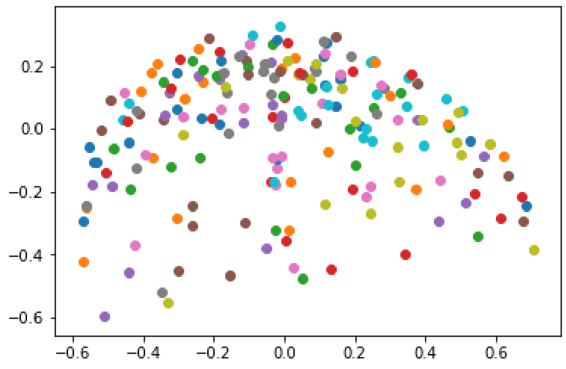

Figure 1: PCA visuliazation.

1.4 Get bag of words

The bags of word array saves the count of the number of feature descriptors which has the most similarity of each cluster in our word vocabulary by calculating the distance. The input of pidst() function must have same number of column. For example pdist(A,B) for A of size (x,y) and B of size(z,y), so the output size will be (z,x). Thus, the size of variable ”distances” should be (1, vocab size).Then we store the cluster id of smallest distance age and add 1 to the vocabulary corresponding. Therefore, the sum of each row of matrix bag of word should be the number of feature.Considering the size of images are different, we have to normalize the bag of words matrix in the end.

```
    bag_of_words = np.zeros((1500,vocab_size))
    n = 0 # number of image
    for path in image_paths:
        img = cv2.imread(path)[:, :, ::-1]
        features = feature_extraction(img, feature)
        distances = []
        for i in range(features.shape[0]):
            distances = pdist(np.mat(features[i]),vocab)
                        # size of distance = (1,vocabsize)
            indice = np.argsort(distances)[0,0]
            bag_of_words[n,indice] = bag_of_words[n,indice] + 1
        n = n + 1
    bag_of_words = bag_of_words / bag_of_words.max(axis=0)
    return bag_of_words
```

We can also use pyramid representation which contains more spatial information. Instead of feed only one image in feature extraction function, we divide image into smaller subimages and do the feature extraction for all of them. In the first level, we divide image into 4 parts, so the size of each image will be (0.5*height,0.5*width). In the second level, we divide image into 16 parts, so the subimage size will be(0.25*height, 0.25*width). However, considering the smaller subimage is, the less feature points it has, so we have to multiply weights when accumulate the histrogram of bag of words.

```
    imgs_n = int((1 / 3) * (4 ** (max_level + 1) - 1))
    sp = np.zeros((1500,vocab_size*imgs_n)).astype(np.float)
    n = 0
    for path in image_paths:
        img_origin = cv2.imread(path)
        height = img_origin.shape[0]
        width = img_origin.shape[1]
        idx = 0 #the number of subimage of one image
        #cut image
        for l in range(0,max_level+1):
            item_width = int(width / (2**l))
            item_height = int(height / (2**l))
            for i in range(0,2**l):
                for j in range(0,2**l):
                    subimg = img_origin[j*item_height:
                            (j+1)*item_height,
                            i*item_width:
                            (i+1)*item_width,:]
                    features = feature_extraction(subimg, feature)
                    distances = []
                    for k in range(features.shape[0]):
                        distances = pdist(np.mat(features[k]),vocab)
                        indice = np.argsort(distances)[0,0]
                        sp[n,idx * vocab_size + indice] =
                                sp[n,idx * vocab_size + indice]
                                + 2**(l-max_level) #weight
                    idx = idx + 1
        n = n + 1
    sp = sp / sp.max(axis=0)
```

1.5 SVM classify

We choose one vs rest SVM as classification model. The decision function of classifier returns a matrix of (number of images * number of classes).If the value is positive, it means the point is in the True Positive area, when it's zero, it means the point is on the decision boundary. If the value is negative, the point is in the wrong side of decision boundary. So we picked the categories with largest value of decision function and saved it in the prediction variable. Two kind of kernels are applied, which is linear kernel and RBF.

```
    categories = np.unique(train_labels)

    if kernel_type == 'RBF':

        prediction = []
        rbf_clf = svm.SVC(kernel = 'rbf')
        rbf_clf.fit(train_image_feats,train_labels)
        df = rbf_clf.decision_function(test_image_feats)
        for i in range(df.shape[0]):
            max_index = list(df[i]).index(max(df[i]))
            prediction.append(categories[max_index])

    elif kernel_type == 'linear':

        prediction = []
        lin_clf = svm.LinearSVC()
        lin_clf.fit(train_image_feats,train_labels)
        df = lin_clf.decision_function(test_image_feats)
        print(df[0])
        for i in range(df.shape[0]):
            max_index = list(df[i]).index(max(df[i]))
            prediction.append(categories[max_index])

    prediction = np.array(prediction)
    return prediction
```

2 Result

2.1 Comparison representation

The feature extracion method is fixed as HOG and the SVM kernel is fixed as RBF. The accuracy of two representation are similar but for each class, see Table1. However,the accuracy for each classes are different, see Table2. For example, the accuracy has an obvious increase for Store and an obvious decrease for Bedroom. When we segment the image to smaller sub-images, the large space of white wall in bedroom might be more confusing. But for the picture of Store which has more complicated layout of room and more spatial information, the accuracy might increase with spatial pyramid.

Representation	Accuracy
get bag of words	0.535
get spatial pyramid	0.537

Table 1: comparison representation

class	bag of words acc	spatial pyramid acc
Kitchen	0.500	0.400
Store	0.380	0.690
Bedroom	0.280	0.150
LivingRoom	0.270	0.280
Office	0.610	0.580
Industrial	0.410	0.480
Suburb	0.700	0.800
InsideCity	0.540	0.450
TallBuilding	0.500	0.450
Street	0.450	0.410
Highway	0.670	0.730
OpenCountry	0.420	0.440
Coast	0.680	0.640
Mountain	0.740	0.710
Forest	0.870	0.840

Table 2: comparison representation

2.2 Comparison SVM kernel

The feature extracion method is fixed as HOG and the representation is fixed as get bag of words.See Table3

SVM Kernel	Accuracy
linear	0.562
RBF	0.537

Table 3: Comparison SVM kernel

2.3 Comparison feature extraction

The representation is fixed as bag of words and the SVM kernel is fixed as RBF. See Table4

feature extraction	Accuracy
HOG	0.537
SIFT	0.304

Table 4: comparison representation

2.4 Confusion Matrix

The attached file index.html shows the confusion matrix of the best model