Homework 5 Writeup

Table of Contents

Code Explanation: feature_extraction.py

HOG

First, create variable that makes HOG descriptors and compute descriptor for image.

```
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)
descriptors = hog.compute(img)
descriptors = np.array(descriptors)
```

The size of descriptor for each patch is 36, so column size is set to 36 and return.

```
descriptors = descriptors.reshape(-1,36)

return descriptors
```

SIFT

First, set grid size to 20 and create variable that makes SIFT descriptors. The width and height of image are also set.

```
grid_size = 20
sift = cv2.SIFT_create()
height= img.shape[0]
width = img.shape[1]
```

Create variables for movement for each grid.

```
h_g = height // grid_size
w_g = width // grid_size
```

Create point in the center of grid.

```
kps = []
for i in range(h_g):
  for j in range(w_g):
    kp_x = grid_size * j + grid_size / 2
    kp_y = grid_size * i + grid_size / 2
    kp = cv2.KeyPoint(kp_x, kp_y, grid_size)
    kps.append(kp)
```

Create descriptor for each point and return.

```
kps, descriptors = sift.compute(img, kps)
descriptors = np.array(descriptors)

return descriptors
```

Code Explanation: get_features_from_pca.py

First, make covariance matrix.

```
mean = np.mean(vocab, axis = 0)
shifted_vocab = vocab - mean
cov = np.cov(shifted_vocab.T)
```

Get eigenvectors with eigenvalues.

```
v, e=np.linalg.eig(cov)
```

Select feat_num eigenvectors with the largest variance (eigen value). And project each point with the eigenvectors.

```
i = np.argsort(-v)[:feat_num]
e = e[:,i]

return shifted_vocab @ e
```

Code Explanation: get_bags_of_words.py

First, set image length as a variable, and initialize all values in histogram to 0.

```
num_image = len(image_paths)
hist = np.zeros((num_image, vocab_size))
```

Get feature descriptors for each image. tmp_hist is also initialized to 0.

```
for i in range(num_image):
  img = cv2.imread(image_paths[i])[:, :, ::-1]
  features = feature_extraction(img, feature)
  tmp_hist = np.zeros(vocab_size)
```

Find distance between descriptors in vocab and descriptors in features. From each feature descriptor, select word with the smallest distance and add 1 to tmp_hist.

```
  dist = pdist(vocab, features)
  min_words = np.argmin(dist, axis=0)
  for min_word in min_words:
    tmp_hist[min_word] = tmp_hist[min_word] + 1
```

Normalize tmp_hist and put it in hist. Return the hist.

```
  tmp_hist = tmp_hist / np.linalg.norm(tmp_hist)
  hist[i,:] = tmp_hist

return hist
```

Code Explanation: get_spatial_pyramid_feats.py

First, set the number of images as a variable and initialize histogram to 0. The length of histogram was increased by 21 times (when max level = 2) compared to 'bag of words'.

```
num_image = len(image_paths)
feat_num = vocab_size * int((1 / 3) * (4 ** (max_level + 1) -
    1))
hist = np.zeros((num_image, feat_num))
```

Loaded image and set height and width as variables. concat_hist is equal to c(h_0_1, ..., h_lmax_4**lmax) in homework logistics. It is feature descriptor that represents each image.

```
for i in range(num_image):
  img = cv2.imread(image_paths[i])[:, :, ::-1]
  height = img.shape[0]
  width = img.shape[1]
  concat_hist = []
```

tmp_img represents a subimage according to each level and each location. The remaining variables were set to indicate the index of tmp_img.

```
  for j in range(max_level + 1):
    two_j = 2**j

    for m in range(two_j):
      for n in range(two_j):
        min_h = int(height/two_j*m)
        max_h = int(height/two_j*(m+1))
        min_w = int(width/two_j*n)
        max_w = int(width/two_j*(n+1))
        tmp_img = img[min_h:max_h, min_w:max_w, :]
```

Similar to 'get_bag_of_words.py', histogram (tmp_hist) of each subimage is created.

```
        features = feature_extraction(tmp_img, feature)
        tmp_hist = np.zeros(vocab_size)
        dist = pdist(vocab, features)
        min_words = np.argmin(dist, axis=0)
        for min_word in min_words:
          tmp_hist[min_word] = tmp_hist[min_word] + 1
```

Each sum of histogram may be different because the size of subimage(so, the number of feature descriptors) is different. So multiply by weight as shown in homework logistics. Then concatenate the histogram.

```
        if j == 0:
          tmp_hist * (2**(-1*max_level))
        else:
          tmp_hist * (2**(-1*max_level + j - 1))

        concat_hist.append(tmp_hist)
```

Flatten and normalize concat_hist. And put concat_hist in the place corresponding to the image in hist. Return hist.

```
  concat_hist = np.array(concat_hist).flatten()
  concat_hist = concat_hist / np.linalg.norm(concat_hist)
  hist[i,:] = concat_hist

return hist
```

Code Explanation: svm_classify.py

Put the length of category (15) as a variable and initialize confidence to 0.

```
len_category = len(categories)
confidence = np.zeros((test_image_feats.shape[0],
   len_category))
```

Will run svm for each category. First, make y_train that determines whether each category is or not.

```
for i in range(len_category):
  y_train = (train_labels == categories[i])
```

The svm model is different according to kernel type. Fit the model and calculate confidence for each train image. And put tmp_conf into the confidence matrix.

```
  if kernel_type == 'RBF':
    model = svm.SVC(kernel='rbf', C = 10, gamma = 0.1)
  if kernel_type == 'linear':
    model = svm.SVC(kernel='linear', C = 10)
  model.fit(train_image_feats, y_train)
  tmp_conf = model.decision_function(test_image_feats)

  confidence[:, i] = tmp_conf
```

For each image, category with the highest confidence is selected and return.

```
max_index = np.argmax(confidence, axis = 1)

return categories[max_index]
```

Result: (SIFT, Linear, Bag of Words)

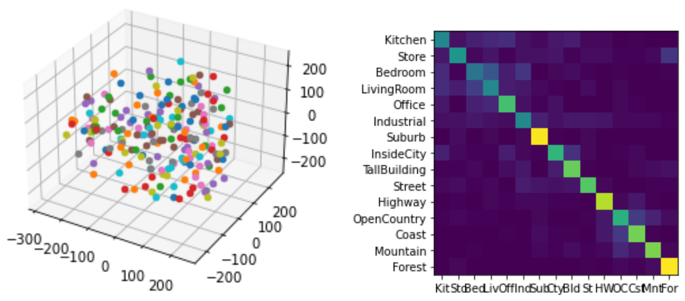
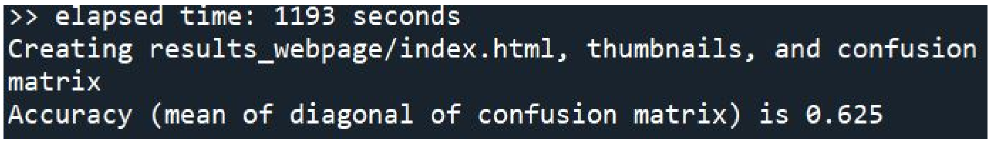


Figure 1: 3D Vocabulary descriptor and Confusion matrix



```
>> elapsed time: 1193 seconds
Creating results_webpage/index.html, thumbnails, and confusion
matrix
Accuracy (mean of diagonal of confusion matrix) is 0.625
```

Figure 2: Elapsed time and Accuracy

Result: (SIFT, Linear, Spatial Pyramid Features)

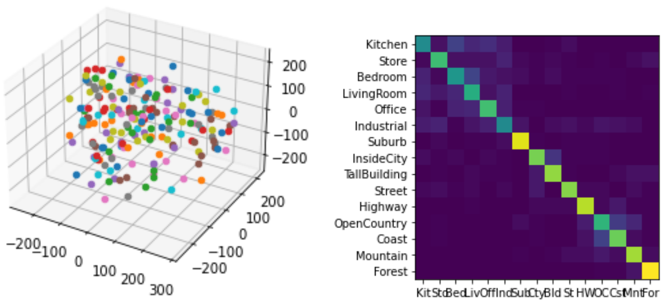
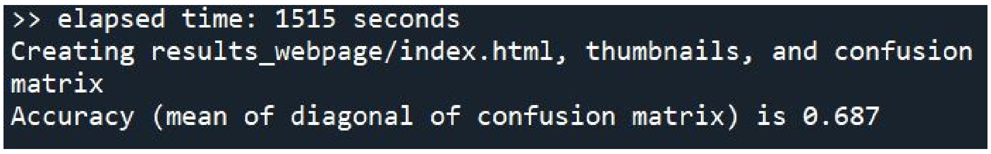


Figure 3: 3D Vocabulary descriptor and Confusion matrix



```
>> elapsed time: 1515 seconds
Creating results_webpage/index.html, thumbnails, and confusion
matrix
Accuracy (mean of diagonal of confusion matrix) is 0.687
```

Figure 4: Elapsed time and Accuracy

Result: (SIFT, RBF, Bag of Words)

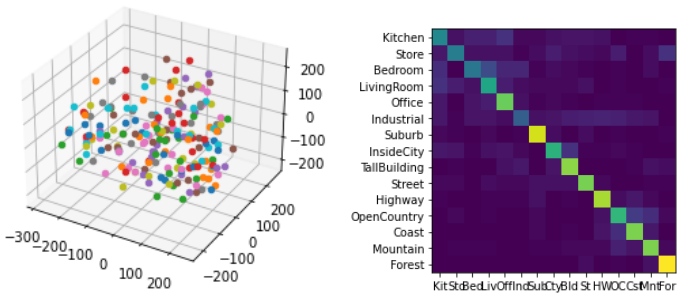
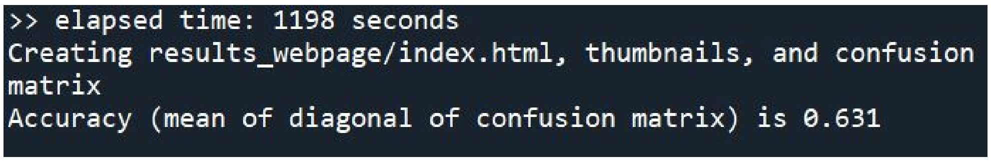


Figure 5: 3D Vocabulary descriptor and Confusion matrix



```
>> elapsed time: 1198 seconds
Creating results_webpage/index.html, thumbnails, and confusion
matrix
Accuracy (mean of diagonal of confusion matrix) is 0.631
```

Figure 6: Elapsed time and Accuracy

Result: (SIFT, RBF, Spatial Pyramid Features)

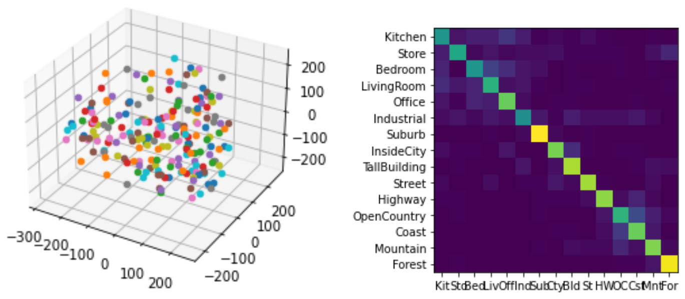
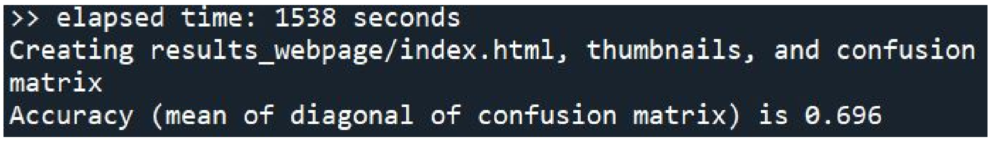


Figure 7: 3D Vocabulary descriptor and Confusion matrix



```
>> elapsed time: 1538 seconds
Creating results_webpage/index.html, thumbnails, and confusion
matrix
Accuracy (mean of diagonal of confusion matrix) is 0.696
```

Figure 8: Elapsed time and Accuracy

Result: (HOG, Linear, Bag of Words)

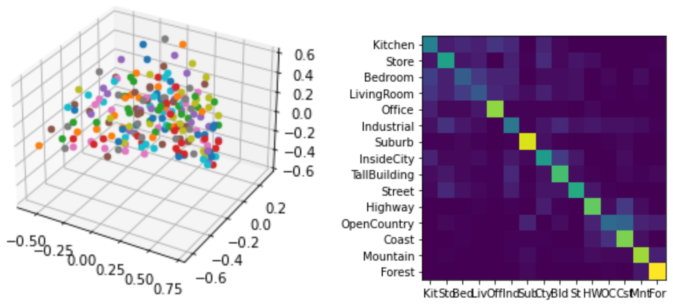
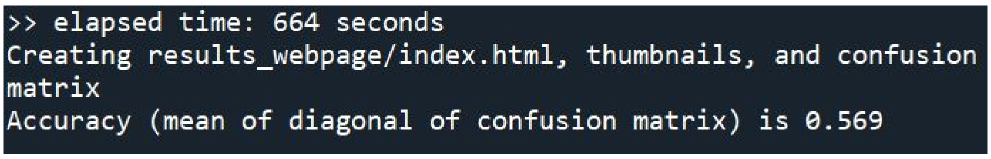


Figure 9: 3D Vocabulary descriptor and Confusion matrix



```
>> elapsed time: 664 seconds
Creating results_webpage/index.html, thumbnails, and confusion
matrix
Accuracy (mean of diagonal of confusion matrix) is 0.569
```

Figure 10: Elapsed time and Accuracy

Result: (HOG, Linear, Spatial Pyramid Features)

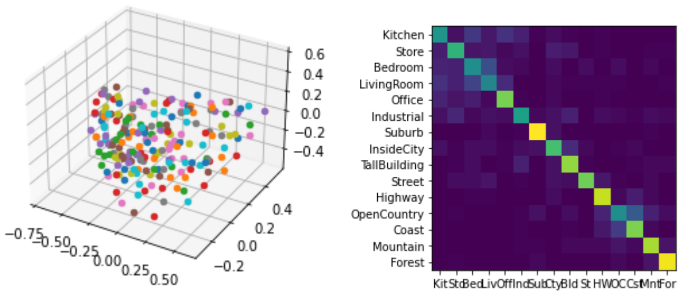
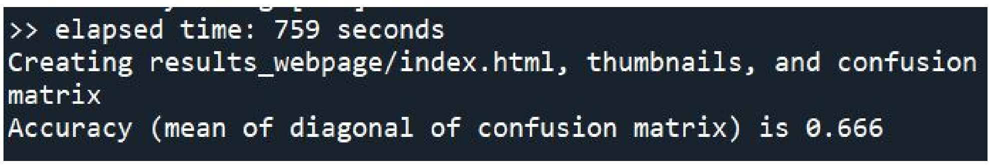


Figure 11: 3D Vocabulary descriptor and Confusion matrix



```
>> elapsed time: 759 seconds
Creating results_webpage/index.html, thumbnails, and confusion
matrix
Accuracy (mean of diagonal of confusion matrix) is 0.666
```

Figure 12: Elapsed time and Accuracy

Result: (HOG, RBF, Bag of Words)

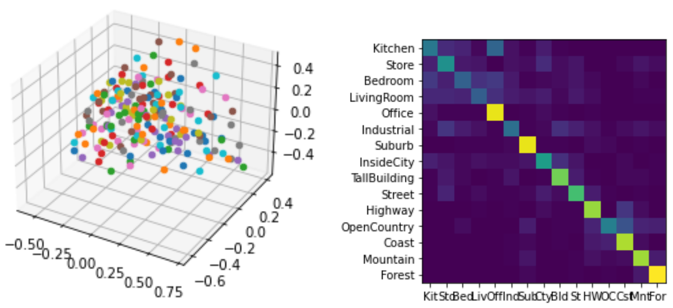
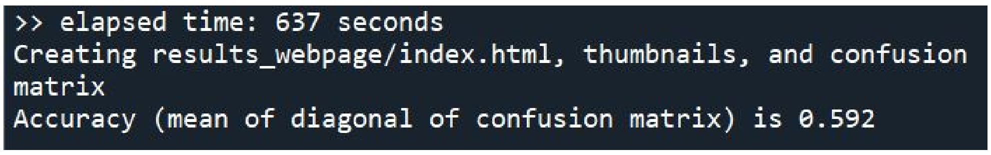


Figure 13: 3D Vocabulary descriptor and Confusion matrix



```
>> elapsed time: 637 seconds
Creating results_webpage/index.html, thumbnails, and confusion
matrix
Accuracy (mean of diagonal of confusion matrix) is 0.592
```

Figure 14: Elapsed time and Accuracy

Result: (HOG, RBF, Spatial Pyramid Features)

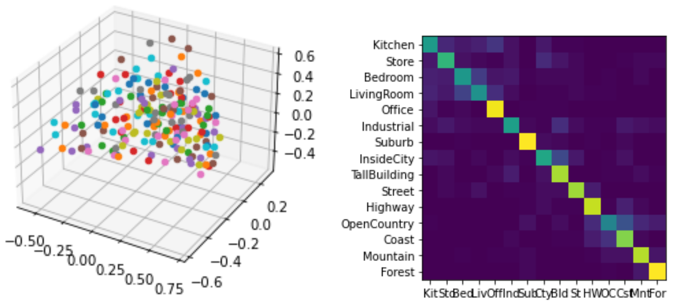
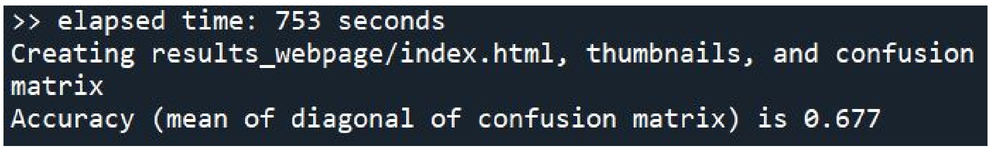


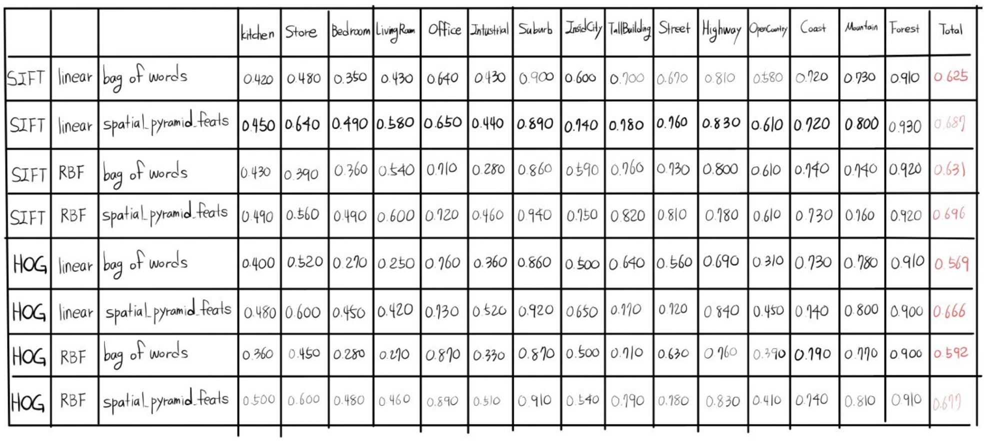
Figure 15: 3D Vocabulary descriptor and Confusion matrix



```
>> elapsed time: 753 seconds
Creating results_webpage/index.html, thumbnails, and confusion
matrix
Accuracy (mean of diagonal of confusion matrix) is 0.677
```

Figure 16: Elapsed time and Accuracy

Result: Summary of Accuracy



			kitchen	Store	Bedroom	LivingRoom	Office	Intustrial	Suburb	InsidCity	TallBuilding	Street	Highway	OpenCountry	Coast	Mountain	Forest	Total
SIFT	linear	bag of words	0.420	0.480	0.350	0.430	0.640	0.430	0.900	0.600	0.700	0.670	0.810	0.580	0.720	0.730	0.910	0.625
SIFT	linear	spatial_pyramid_feats	0.450	0.640	0.490	0.580	0.650	0.440	0.890	0.740	0.780	0.760	0.830	0.610	0.720	0.800	0.930	0.687
SIFT	RBF	bag of words	0.430	0.390	0.360	0.540	0.710	0.280	0.860	0.590	0.760	0.730	0.800	0.610	0.740	0.740	0.920	0.631
SIFT	RBF	spatial_pyramid_feats	0.490	0.560	0.490	0.600	0.720	0.460	0.940	0.750	0.820	0.810	0.780	0.610	0.730	0.760	0.920	0.696
HOG	linear	bag of words	0.400	0.520	0.270	0.250	0.760	0.360	0.860	0.500	0.640	0.560	0.690	0.310	0.730	0.780	0.910	0.569
HOG	linear	spatial_pyramid_feats	0.480	0.600	0.450	0.420	0.730	0.520	0.920	0.650	0.770	0.720	0.840	0.450	0.740	0.800	0.900	0.666
HOG	RBF	bag of words	0.360	0.450	0.280	0.270	0.870	0.330	0.870	0.500	0.710	0.630	0.760	0.390	0.790	0.770	0.900	0.592
HOG	RBF	spatial_pyramid_feats	0.500	0.600	0.480	0.460	0.890	0.510	0.910	0.540	0.790	0.780	0.830	0.410	0.740	0.810	0.910	0.677

Figure 17: Summary of Accuracy

Summary

- Compare SIFT and HOG: SHIFT was much longer time elapsed. However, accuracy was higher. The reason is that the length of feature descriptor of SIFT (128) is longer than that of HOG (36).

- Compare linear kernel and RBF kernel: RBF was more accurate. The reason is that it cannot be said to linearly classified when using SVM.

- Compare bag of words and spatial pyramid features: Spatial pyramid features has higher accuracy. The reason is that space information is also put into descriptors and trained.

- Elapsed time: Most of them run in less than 20 minutes. However, when using SIFT and spatial pyramid feature descriptor, it take more than 20 minutes. If I use a computer with better performance or reduce the number of descriptors, It will be able to reduce the time. The specifications of computer used are 2 cores, 4 threads, 1.6-2.3GHz.