Homework 2 Writeup

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.

Interesting Implementation Detail

My code snippet highlights an interesting point.

Listing 1: HOG Feature extraction

```
1     if feature == 'HoG':
2         # HoG parameters
3         win_size = (32, 32)
4         block_size = (32, 32)
5         block_stride = (16, 16)
6         cell_size = (16, 16)
7         nbins = 9
8         deriv_aperture = 1
9         win_sigma = 4
10        histogram_norm_type = 0
11        l2_hys_threshold = 2.0000000000000001e-01
12        gamma_correction = 0
13        nlevels = 64
14
15
16        # Your code here. You should also change the
              return value.
17
18        # make HOG descriptor model with given parameter
19        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)
20
21        m = img.shape[0]
22        n = img.shape[1]
23
24        all_f = []
```

```
25
26          # divide original image with 16 X 16 grid and
                make subimages, so find HoG descriptor by
                grouped 4 cell (32 X 32) and 16 X 16 stride
27          for i in range(int(m / 16) - 1):
28              for j in range(int(n / 16) - 1):
29                  x = i * 16
30                  y = j * 16
31                  h = hog.compute(img[x:x+32, y:y+32])
32                  all_f.append(np.reshape(h, (1, 36)))
33
34          # combine Hog desciptor from sub images
35          all_f = np.concatenate(all_f, 0)
36          return all_f
```

Listing 2: SIFT Feature extraction

```
1           m = img.shape[0]
2           n = img.shape[1]
3
4           all_f = []
5           gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
6           sift = cv2.xfeatures2d.SIFT_create()
7
8           # divide original image with 20 X 20 grid and
                make subimages, so find SIFT descriptor by 20
                X 20 sub images.
9           for i in range(int(m / 20)):
10              for j in range(int(n / 20)):
11                  x = i * 20
12                  y = j * 20
13                  kp, des = sift.detectAndCompute(gray[x:x
                        +20, y:y+20], None)
14                  if len(kp) != 0:
15                      all_f.append(des)
16
17          #If sift is not detected, exception handling is
                done.
18          if len(all_f) != 0:
19              all_f = np.concatenate(all_f, 0)
20          else:
21              all_f = None
22
23          return all_f
```

In both HOG and SIFT, because there was a difference between applying the descriptor to the entire image at one time and dividing whole image and directly into the sub-image,

so I chose the latter. The details of the algorithm are annotated in the above code block.

Listing 3: K-means Clustering

```
    # Your code here. You should also change the return
        value.
    np.random.seed(0)
    n = all_features.shape[0]
    f_n = all_features.shape[1]
    center = np.zeros((vocab_size, f_n))

    # I choose start points as random sample of
        all_feature
    idx = np.random.choice(n, vocab_size, replace=False)
    for i in range(vocab_size):
        center[i, :] = all_features[idx[i], :]

    # In the worst case, iterations up to max_iter
    for k in range(max_iter):

        # back-up of previous center points, because of
            compare of epsilon
        prev_center = np.copy(center)

        # get distance between center points and feature
            points and labeling of feature using np.argmin
        d = pdist(all_features, center)
        lab = np.argmin(d, axis=1)

        # For empty cluster cases, align the points in
            the order that is farthest from the that point
            's center
        mind = np.min(d, axis=1)
        sort_arg = np.argsort(mind * -1)

        # Use the labels to update the center points to
            the average of the points in the cluster.
        center = np.zeros((vocab_size, f_n))
        num = np.zeros(vocab_size)
        for i in range(n):
            center[lab[i], :] = center[lab[i], :] +
                all_features[i, :]
            num[lab[i]] += 1

        # In empty case, the empty cluster is divided by
            0. Therefore, to prevent this, the point that
            is farthest from the center is adopted as a
```

```
            new center.
34          id = 0
35          for i in range(vocab_size):
36              if num[i] == 0:
37                  center[i, :] = all_features[sort_arg[id],
                        :]
38                  id += 1
39              else:
40                  center[i, :] = center[i, :] / num[i]
41
42          # The ending condition is checked by comparing
                the previous center point with the current
                point.
43          d2 = pdist(prev_center, center)
44          maxe = 0
45          for i in range(vocab_size):
46              if maxe < d2[i, i]:
47                  maxe = d2[i, i]
48
49          if maxe < epsilon:
50              break
51
52      return center
```

The special things I used to implement K-means are ,first, I choose start points as random sample of all feature and second, in the empty cluster case, to prevent this, the point that is farthest from the center is adopted as a new center. The details of the algorithm are annotated in the above code block.

Listing 4: PCA

```
1   n = vocab.shape[0]
2   f_n = vocab.shape[1]
3   f = np.copy(vocab)
4
5   # subtract average of each coordinate
6   for i in range(f_n):
7       f[:,i] = f[:,i] - np.average(f[:,i])
8
9   # make covariance matrix using matrix multiplication
        (X - X_bar)^T * (X - X_bar) / N
10  cov = np.matmul(np.transpose(f), f)
11  cov = cov / n
12
13  # The eigenvector corresponding to dominate
        eigenvalue are equal to the first,second,... column
        vector of V when a singular value decomposition.
```

```
14      u, s, vh = np.linalg.svd(cov, full_matrices=True)
15      v = np.transpose(vh)
16
17      # In order to obtain the principal component of each
            voca, I compute the inner product feature vector
            with eigenvector.
18      pca_f = np.zeros((n, feat_num))
19      for i in range(feat_num):
20          pca_f[:,i] = np.reshape(np.matmul(f, np.reshape(v
                [:,i], (-1,1))), -1)
21
22      return pca_f
```

First, I subtract average of each coordinate from vocabulary vectors. Then, get covariance matrix from (X - X bar)T * (X - X bar) / N . In additional, for finding eigenvector of cov matrix corresponding to dominate eigenvalue, I use SVD. The details of the algorithm are annotated in the above code block.

Listing 5: Bag of words

```
1       vocab_size = vocab.shape[0]
2       N = len(image_paths)
3       hist = np.zeros((N, vocab_size))
4       # Your code here. You should also change the return
            value.
5       k = 0
6       for path in image_paths:
7           img = cv2.imread(path)[:, :, ::-1]
8
9           # get features of image
10          features = feature_extraction(img, feature)
11
12          # get distance of features and codevectors, then
                get histogram
13          d = pdist(features, vocab);
14          lab = np.argmin(d, axis=1)
15          for l in lab:
16              hist[k, l] += 1
17
18          # normalizing
19          hist[k,:] = hist[k,:] / len(lab)
20          k += 1
21
22      return hist
```

First, get features of each image, and get distance between features and codevectors. Then I can construct histogram selecting a codevector that minimizes the distance for each feature using np.argmin.

Listing 6: Spatial pyramid representation

```
vocab_size = vocab.shape[0]

# number of kind of histogram
n2 = int((4 ** (max_level + 1) - 1) / 3)

N = len(image_paths)

# feature dimension is vocab_size * (1 / 3) * (4 ^ (max_level + 1) - 1)
hist = np.zeros((N, vocab_size * n2))

# Your code here. You should also change the return value.
k = 0
for path in image_paths:
    img = cv2.imread(path)[:, :, ::-1]
    n = img.shape[0]
    m = img.shape[1]

    id = 0
    for l in range(max_level + 1):

        # I choose the scale 1/4 when level 0, 1/4 when level 1, 1/2 when level 2, as in the paper.
        scale = 2 ** (l - max_level - 1)
        if l == 0:
            scale = 2 ** -2

        # split image to 2^l * 2^l subimages, and get feature and histogram each splitted images.
        div = 2 ** l
        dx = int(n / div)
        dy = int(m / div)
        for i in range(div):
            for j in range(div):
                x = i * dx
                y = j * dy
                features = feature_extraction(img[x:x+dx, y:y+dy], feature)
                if features is not None:
                    d = pdist(features, vocab);
                    lab = np.argmin(d, axis=1)
                    for l in lab:
```

```
39                                  hist[k, id + 1] += 1 * scale
40                      id += vocab_size
41
42          # normalizing
43          hist[k,:] = hist[k,:] / np.sum(hist[k,:])
44          k += 1
45
46      return hist
```

Not only the histogram of the whole image but also the histogram of the images divided into smaller sizes are separately obtained and used as a feature. I place a larger weight for small size image, to fill the spatial loss of the small divided image. The weights follow the formula written in the paper. The details of the algorithm are annotated in the above code block.

Listing 7: Multi-class SVM

```
1   categories = np.unique(train_labels)
2   N = train_image_feats.shape[0]
3   M = test_image_feats.shape[0]
4
5   c_n  =len(categories)
6
7   test_y = np.zeros((M, c_n))
8   if kernel_type == 'RBF':
9       kernel_type = 'rbf'
10
11  # Create 1 vs other svm detectors for the number of
        categories.
12  for k in range(c_n):
13
14      # '1' category to be label 1, 'other' category to
            be label -1
15      train_y = np.repeat(-1, N)
16      train_y[train_labels == categories[k]] = 1
17
18      # I use tuned paremater, C = 10, gamma = 'scale'
            and kernel_type
19      model = svm.SVC(C=10, kernel = kernel_type, gamma
            = 'scale').fit(train_image_feats, train_y)
20
21      # I can get score of test_image_feature of each
            svm detector
22      test_y[:,k] = np.reshape(model.decision_function(
            test_image_feats), -1)
23
24  # For choose label, I adopt svm detector with the
        most positive score.
```

```
25      test_lab = np.argmax(test_y, axis=1)
26      test_label = []
27      for i in range(M):
28          test_label.append(categories[test_lab[i]])
29
30      return np.array(test_label)
```

For execute Multi-class SVM, I create many 1 vs other svm detectors, with tuned parameter and kernel type, and adopt svm detector with the most positive score for choose label. The details of the algorithm are annotated in the above code block.

Recognition accuracy comparison between using SIFT features and HOG features (with fixing the SVM kernel as linear)

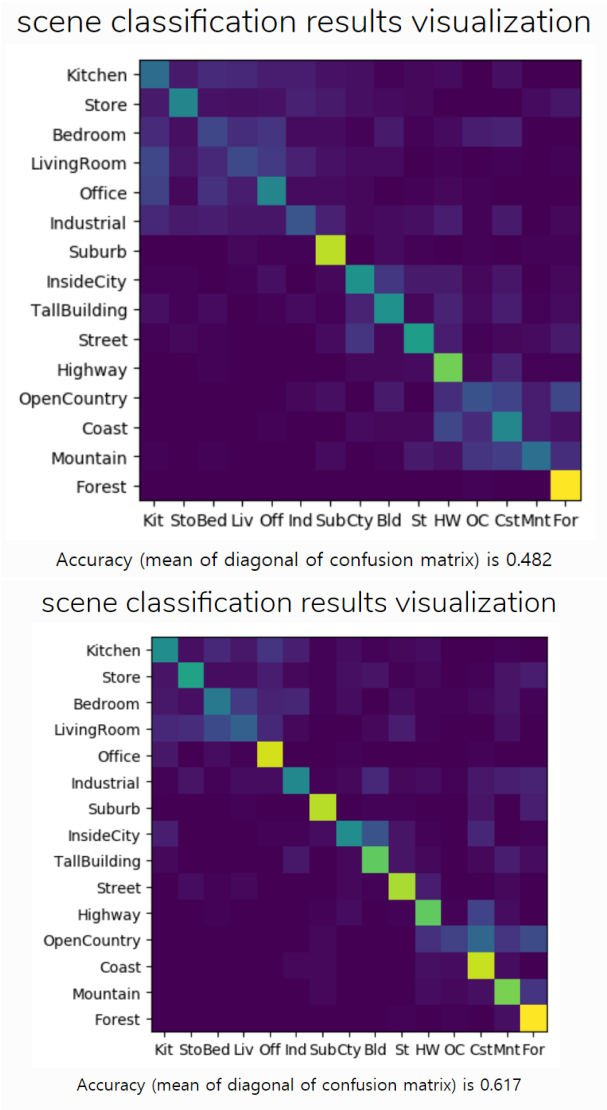

Figure 1: *Up:* Confusion matrix with SIFT features. *Down:* Confusion matrix with HOG features. (In the both cases, I fixed the spatial pyramid representation and the linear kernel)

HOG uses edge direction information, so it is suitable for identifying objects with unique outline,edge information, and SIFT can be matched irrespective of shape change, size change, rotation and so on because it is matched by the feature point unit of the model, so SIFT is a suitable method when the internal pattern is complicated , so the feature points are abundant. These results show that this data set is more robust to the HOG feature.

Plots which visualizes PCA of the vocabulary

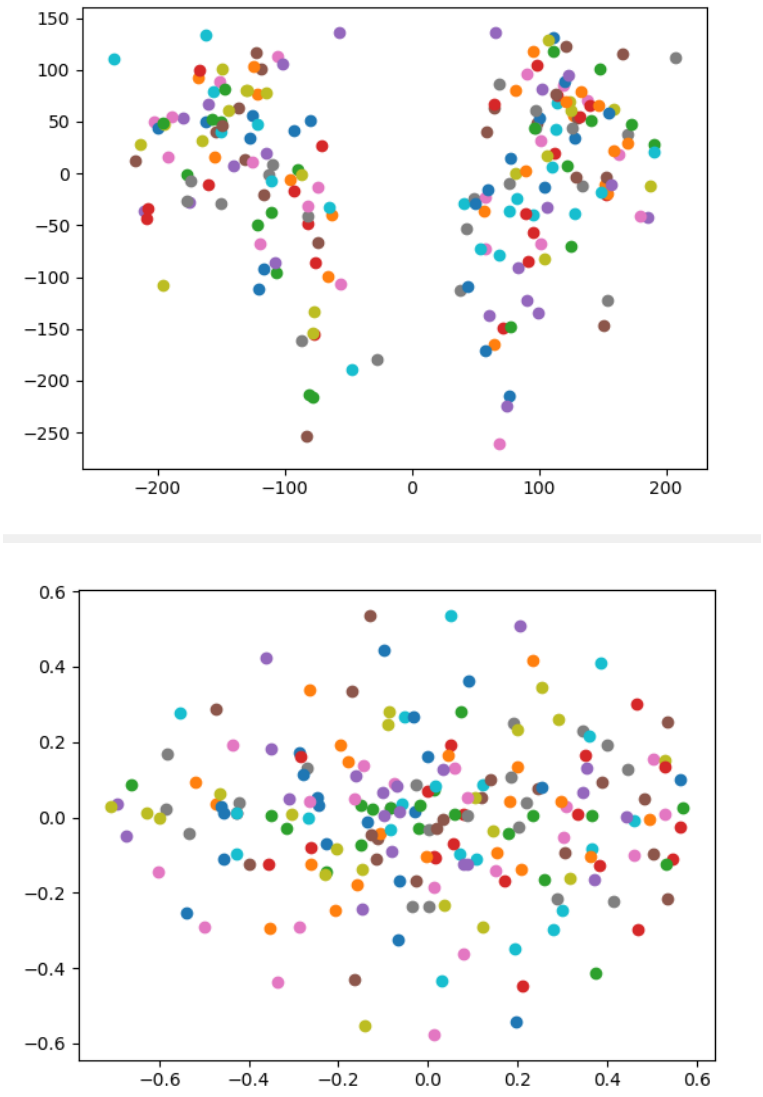

Figure 2: *Up:* PCA of SIFT vocabulary. *Down:* PCA of HOG vocabulary.

As the distribution of principal components of codevectors spreads over the PCA graph, the codebook describes the more generalized features, so the HOG feature is better suited for this dataset. This result is the same as the results on the previous page.

Recognition accuracy comparison between SVM kernels(linear vs. RBF)

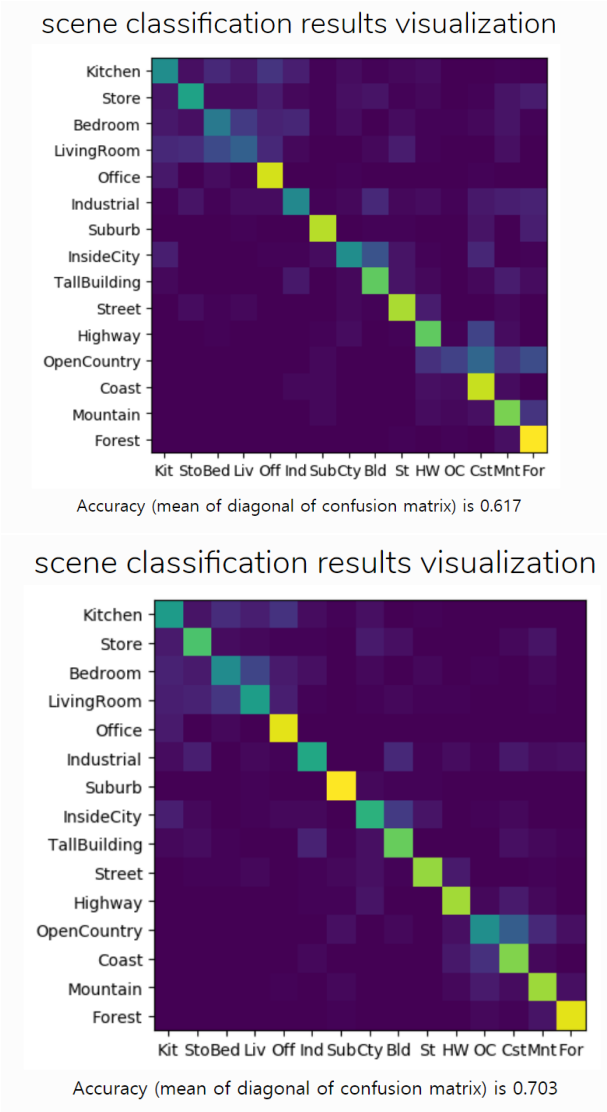

Figure 3: *Up:* Confusion matrix with linear kernel. *Down:* Confusion matrix with RBF. (In the both cases, I fixed the HOG feature and the spatial pyramid representation)

It can be seen that the higher accuracy is obtained when mapping the data to the higher dimensional using RBF. Therefore, it can not be said that the linear separability of feature is strong at present without kernel trick.

Recognition accuracy comparison between bag of words representation without spatial pyramid and that with spatial pyramid

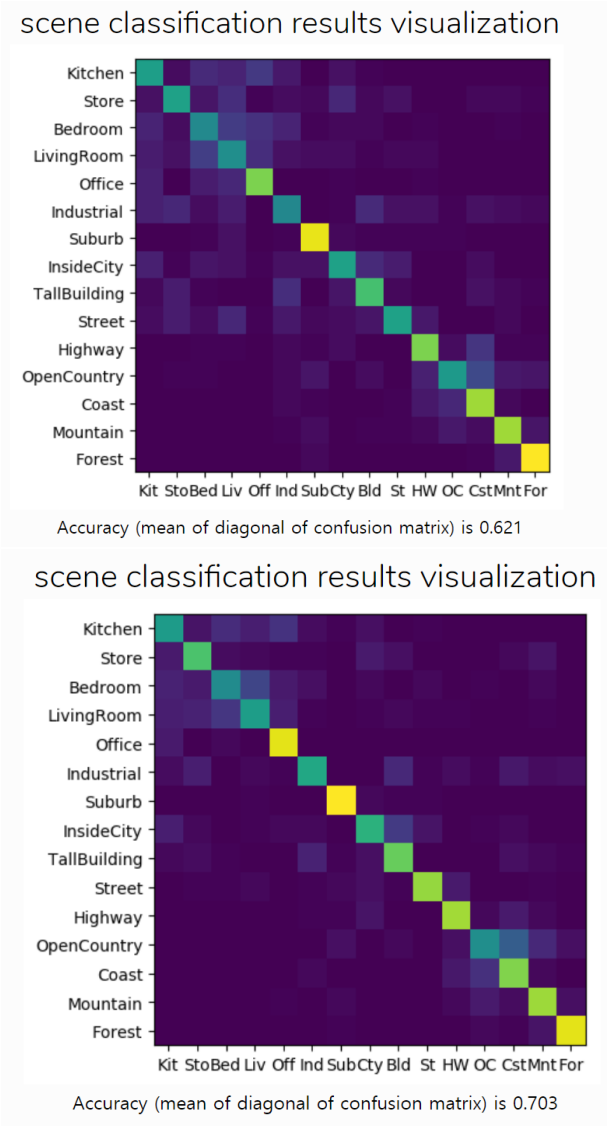

Figure 4: *Up:* Confusion matrix without spatial pyramid. *Down:* Confusion matrix with spatial pyramid. (In the both cases, I fixed the HOG feature and the RBF kernel)

When using spatial pyramid representation, it takes more time than just using bag of words without spatial pyramid, but it can improve the accuracy by recovering some lost spatial information