Hacettepe University
Department of Computer Engineering
BBM418 - Computer Vision Laboratory
Programming Assignment 1

Sevda Sayan
21527305
sevdasayan@cs.hacettepe.edu.tr

April 2019

Contents

1 Introduction

In this assignment we are expected to implement basic image representation and retrieval methods. In first part, a gabor filter bank is constructed by creating forty gabor filter and each filter assigned to each image in our data-set which is taken from Caltech 256[1]. In second part, sift feature vectors are extracted and at the end, a vector with length 1x128 for each image is obtained. Also in last part, bag of visual word decriptors are extracted.
Main purpose underlying these parts is comparing accuracy of the recognition results obtained each part by using "k-NN Classification".
In this report, some implementation details like used parameters, functions and libraries are given. And also experimental results that obtained for each part are placed with comments about them.

2 Implementation Details

```
query_file_names = []
for (dirpath, dirnames, filenames) in os.walk(query_path):
    for f in filenames:
        query_file_names.append(os.path.join(os.path.join(dirpath, f)))

cv_query_images = []
for filename in query_file_names:
    cv_query_images.append(cv2.imread( filename))
```

Figure 1: Reading Dataset

For importing train and query images, I assemblage all of their file names into an array after that by using build-in funtion cv2.imread(), I imported them and appended into image array. Hereby I can easily reach both filenames and images of query and train data.

2.1 Gabor filter bank

Gabor filters are orientation-sensitive filters, used for edge and texture analysis. For generating gabor filter bank, there is a function named buildFilters(). This function returns an array that has forty gabor filter inside it.

After building filters, I applied my gabor filters to each image to get different filter responses. Below process() function achieves this purpose. It applies all filters that taken from its second parameter into an image that taken from its first parameter. By doing this convolution I used build-in function named ndimage.convolve(). For each response, it calculates mean value of each filtered image and returns normalized 1x40 array that has these mean values.

By the help of these two functions, I got response of both query and train images. Each response of query image is compared with responses of train

```python
def build_filters():
    filters = []
    ksize = 31
    for theta in np.arange(0, np.pi, np.pi / 40):
        kern = cv2.getGaborKernel((ksize, ksize), 4.0, theta, 10.0, 0.5, 0, ktype=cv2.CV_32F)
        kern /= 1.5*kern.sum()
        filters.append(kern)
    return filters
```

Figure 2: Building filters

```python
def process(img, filters):
    accum = np.zeros(shape=(1,40))
    i = 0
    for kern in filters:
        fimg = ndimage.convolve(img, kern, mode='constant', cval=1.0)
        mean_val = np.mean(fimg)
        np.put(accum, i, mean_val)
        i = i+1
    return normalize(accum)
```

Figure 3: Processing image with filters

images by using Euclidean distance. For calculating Euclidean distance, I used build-in function named distance.euclidean() from scipy library. We can say that our query image's class is the same as train image's class that has smallest distance with query image.

```python
    for k in range(len(train_responces)):
        dist_in = distance.euclidean(response , train_responces[k])
        if(dist_in < dist ):
            dist = dist_in
            max_dist_idx = k
```

Figure 4: Calculating smallest distance

2.2 Average SIFT

The scale-invariant feature transform (SIFT) is an algorithm used to detect and describe local features in digital images. It locates certain key points and then furnishes them with quantitative information (so-called descriptors) which can for example be used for object recognition.
For extracting SIFT feature vectors, firstly I translated requested image into black and white using build-in function named cvtColor() and then used build-in function named xfeatures2d.SIFTcreate() from opencv library.

```python
def to_gray(color_img):
    gray = cv2.cvtColor(color_img, cv2.COLOR_BGR2GRAY)
    return gray

def gen_sift_features(gray_img):
    sift = cv2.xfeatures2d.SIFT_create()
    kp, desc = sift.detectAndCompute(gray_img, None)
    return (kp, desc)
```

Figure 5: Extracting SIFT vectors

By the help of these two functions, I got response of both query and train images as gabor filters. And again Euclidean distance is used among descriptor vectors that extracted from images. We again can say that our query image's class is the same as train image's class that has smallest distance among average of the train's descriptors and query image's descriptor vector.

2.3 BoW

To classify image category creating a bag of visual words is one of the good solution. At the end of this process a histogram of visual word occurrences will be generated and this histogram represents an image. These histograms are used to train an image category classifier. The steps below describe how did I create the bag of visual words, and then train and apply an image category classifier.

i) Codebook

Functions that used for extracting SIFT features are the same as part 2.2 and Figure5. After procuring Nx128 SIFT descriptors which are extracted from training set, I assembled each of training image's descriptors by using make-desc-arr() function given below. This function takes training images as parameter and returns descriptors as an array.

```python
def make_desc_arr(cv_train_images):
    desc_array = []
    for k in range(5):
        train_gray = to_gray(cv_train_images[k])
        for d in range(128):
            desc_array.append(gen_sift_features(train_gray)[1][d])
    return desc_array
```

Figure 6: Creating descriptor list

Lastly for extracting codebook, build-in KMeans() function from sklearn.cluster library is usedand composed descriptor list fitted into this clustering algorithm. Eventually different accuracy test results are observed according to various cluster numbers.

```
kmeans = KMeans(n_clusters = 500)
kmeans.fit(descriptor_list)
```

Figure 7: k-means usage

ii) Quantization For attaining visual word representation of each image obtained codebook from previous part is used. Lastly each image is represented as 1xN where each descriptor in the image is described by a cluster centroid id.

iii) Histogram Histograms summarizes entire image based on its distribution of word occurrences. For constructing histogram of each image build-histogram() function is used. The function then increments histogram bins based on the proximity of the descriptor to a particular cluster center. The histogram length corresponds to the number of visual words that the bagOfFeatures object constructed. The histogram becomes a feature vector for the image. At the end, a histogram that has length 1x(number of cluster) constructed for each image.

```
def build_histogram(descriptor_list, cluster_alg):
    histogram = np.zeros( len(cluster_alg.cluster_centers_) )
    cluster_result =  cluster_alg.predict(descriptor_list)
    for i in cluster_result:
        histogram[i] += 1.0
    return histogram
```

Figure 8: Histogram building

By the helps of these built histograms for both query and train images, distances among specified images are calculated by using euclidean and average accuracies are obtained and observed by using k-nn algorithm.

Histograms of each train images are gathered into preprocessed-trains array. This function first generates SIFT vector of intended image and then builds histogram of that image according to it's descriptor vector.

2.4 BoW with spatial tiling

Now, we are splitting image NxN tiles and make our histograms from each part separately. Again length of our histogram will be identical with length of our histogram. After that all the images will be represented by the concatenated

```
preprocessed_trains =[]
for i in range(len(cv_train_images)):
    train_gray = to_gray(cv_train_images[i])
    train_kp, train_desc = gen_sift_features(train_gray)
    histogram = build_histogram(train_desc, kmeans)
    preprocessed_trains.append(histogram)
```

Figure 9: Array of train histograms

BoW histograms from all the tiles. After obtain nx(number of cluster) histogram matrix we can make our comparision by using them and can obtain our accuracies.

For splitting images according to their sizes, getting pixel size of them was necessary. To achive this, I used a build-in function named shape from opencv library. After that, dividing them with N gave me pixel size of each tile and septated image can be extracted from original image by taking just values in this interval.

```
for ih in range(N):
    for iw in range(N):
        x = int(width / N * iw)
        y = int(height / N * ih)
        h = int((height / N))
        w = int((width / N))

        img_in = cv_query_images[i][y:y + h, x:x + w]
```

Figure 10: Tiling an image

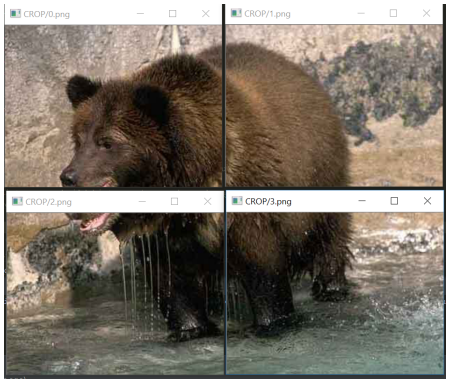

Figure 11: Tilled parts of an image

3 Experimental Results

3.1 Results of gabor filter bank

When I apply gabor filters that formed when theta is evenly increase among zero and pi, obtained average class based Average accuracy (ratio of correctly predicted query images) and class-based accuracy (ratio of correctly predicted query images) for each category separately are shown in below.

However when I apply gabor filters that formed when theta is evenly increase among zero and pi/4, obtained average accuracy and some of class based accuracies decreased. This downfall might be caused by range of theta in different gabor filters.

By looking results obtained, best fixed gabor filter bank is found when theta change in interval zero and pi.

Gabor Filter Bank / Theta	**(0, pi)**	**(0, pi/4)**	**(0, pi/8)**
Average Class Based Accuracy For 009	0%	0%	0%
Average Class Based Accuracy For 024	20%	0%	0%
Average Class Based Accuracy For 041	20%	20%	20%
Average Class Based Accuracy For 065	20%	20%	20%
Average Class Based Accuracy For 072	40%	0%	0%
Average Class Based Accuracy For 105	0%	20%	20%
Average Class Based Accuracy For 107	0%	20%	0%
Average Class Based Accuracy For 118	0%	0%	20%
Average Class Based Accuracy For 152	20%	20%	20%
Average Class Based Accuracy For 212	20%	0%	60%
Average Accuracy	**14%**	**10%**	**16%**

Figure 12: Accuracies when theta (0,pi), (0,pi/4) and (0,pi/8)

3.2 Results of average SIFT

When I extract SIFT feature vector and take the average of the descriptors, I got a vector with length 1x128 for each image. By using these vectors accuracies of query images are calculated. Results are given below.

SIFT	Accuracy
Average Class Based Accuracy For 009	20%
Average Class Based Accuracy For 024	0%
Average Class Based Accuracy For 041	20%
Average Class Based Accuracy For 065	20%
Average Class Based Accuracy For 072	40%
Average Class Based Accuracy For 105	40%
Average Class Based Accuracy For 107	20%
Average Class Based Accuracy For 118	20%
Average Class Based Accuracy For 152	20%
Average Class Based Accuracy For 212	20%
Average Accuracy	**22%**

Figure 13: Accuracies for SIFT

3.3 Results of BoW

Different conditions are tried for obtaining various results by changing the number of cluster K.

I performed a tests when number of cluster is 500, 300 and 200 and obtained results are given below.

BoW/ Number of Cluster	**K=100**	**K=200**	**K=300**	**K=400**	**K=500**
Class based accuracy for 009	40%	40%	40%	40%	40%
Class based accuracy for 024	0%	0%	0%	0%	0%
Class based accuracy for 041	20%	0%	0%	0%	0%
Class based accuracy for 065	0%	0%	0%	20%	0%
Class based accuracy for 072	60%	0%	20%	40%	20%
Class based accuracy for 105	60%	60%	60%	60%	100%
Class based accuracy for 107	40%	20%	0%	20%	40%
Class based accuracy for 118	0%	0%	0%	20%	0%
Class based accuracy for 152	20%	0%	20%	20%	20%
Class based accuracy for 212	0%	0%	40%	0%	0%
Average Accuracy	**24%**	**16%**	**18%**	**22%**	**22%**

Figure 14: Accuracies for BoW

We can clearly see that when number of cluster increase, average accuracies is increasing. But there is a exception about this increment when K is 100. This exception might be caused by closeness of selected hundreds of centroids. These centroids might be in equipoise with most close parts of two compared images.

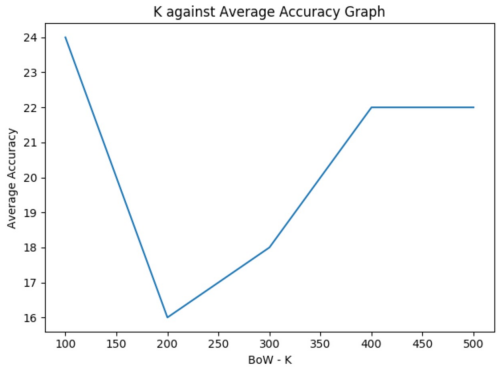

Figure 15: K against average accuracy

3.4 Results of BoW with spatial tiling

A straightforward way to find optimal representations is manually design representation candidates and verify the candidates by running the classifier. When I split query images for extracting bag of visual word for each image, surprisingly attained average accuracy decreased. However, when consider change of average accuracy against value of N, it effected my result linearly.

We can clearly speculate that different spatial representations can affects results considerably. While number of piece increase, obtained average accuracy is increasing according to implementation.

BoW with Spatial Tilling	**N=2**	**N=3**
Average Class Based Accuracy For 009	0%	0%
Average Class Based Accuracy For 024	0%	0%
Average Class Based Accuracy For 041	60%	80%
Average Class Based Accuracy For 065	0%	0%
Average Class Based Accuracy For 072	0%	20%
Average Class Based Accuracy For 105	0%	0%
Average Class Based Accuracy For 107	0%	20%
Average Class Based Accuracy For 118	40%	40%
Average Class Based Accuracy For 152	0%	0%
Average Class Based Accuracy For 212	0%	0%
Average Accuracy	**10%**	**16%**

Figure 16: Accuracies for BoW with spatial tiling

3.5 Comparing results of four different options

For observing performances of above-mentioned options (Gabor filter bank, Average SIFT, BoW, BoW with spatial tiling), three different images from different classes are selected and five most similar train images with them are estimated.

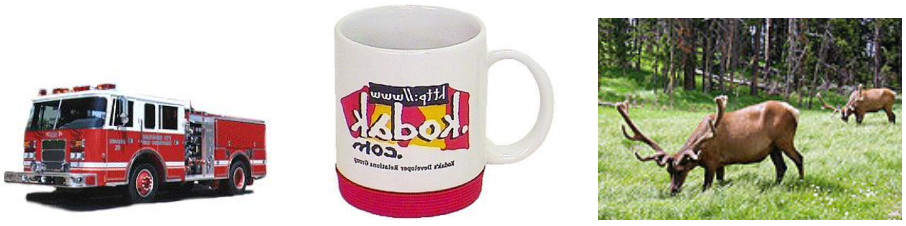

Figure 17: Selected three images for comparison

Most five similar images for these three images found and questions that what number of prediction is true? and how many times it guessed right? are visualized.

These visualized graphs are given below. Fist graph answers for "what number of prediction is true?". Likely, second graph answers for "how many times it guessed right?".

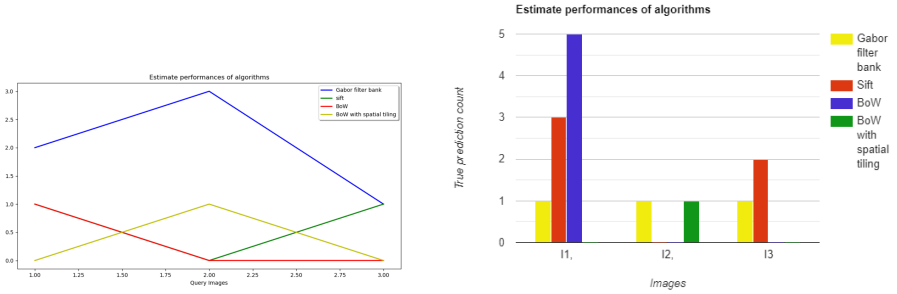

Figure 18: Successful prediction numbers and counts

We are seeing that first selected image has highest successful prediction when considering other three images. This result might be caused by closeness and right formed train data set. Additionally, successful prediction is usually found at the first try.

3.6 Drawbacks of euclidean

Euclidean distance performs well when deployed to datasets that include compact or isolated clusters (1) (2). Although Euclidean distance is very common in clustering, it has a drawback: if two data vectors have no attribute values in common, they may have a smaller distance than the other pair of data vectors containing the same attribute values (2) (3). Another problem with Euclidean distance as a family of the Minkowski metric is that the largest-scaled feature would dominate the others. Normalization of continuous features is a solution to this problem (4).

According to a research (5) that applied on textures for observing performances of different types of distance functions, performance of Square Chi and Squared Chord are found better than Euclidean and Manhattan distance, with classification accuracy is76

4 Conclusion

Gabor filter bank, Average SIFT, BoW and BoW with spatial tiling are applied and as a result, generally average accuracies are not sufficiently high. It is probably caused by poorness of training data set. Maybe our features doesn't fit enough to get better classification. Additionally number of classes might affect accuracies. It is possible that data set have less data to specify each class for our classifier to differentiate among them. Also if some input features are "noisy" and "non-infornative", it again affects our accuracy negatively. So, many factors might effect accuracy if data is not normalized for instance.

References

[1] J. A. Mao J, "A self-organizing network for hyperellipsoidal clustering (hec)," *IEEE Trans Neural Networks.*, 1996.

[2] F. P. Jain AK, Murty MN, "Data clustering: a review. acm computing surveys," *ACM*, 1999.

[3] L. L. Legendre P, "Numerical ecology.," *Elsevier*, 2012.

[4] A. S. Shirkhorshidi. `https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0144059`, Dec 2015.

[5] D. S. S. A. Alamri, "Satellite image classification by using distance metric," *IJCSIS*, 2016.