CENG 483

Introduction to Computer Vision

Fall 2021-2022

Take Home Exam 2

Object Recognition

Student ID: 2310191

Important

Please note that in the implementation, OpenCV's Fast Feature Detector is used instead of SIFT detector, because not only it is fast regarding our experiments, but also it generates more points. Generating more points is not always necessarily better; however, note that SIFT generates less points, sometimes no points at all, leading to errors in the code. This is less likely with fast detector, but when you encounter such an error (Generally a dimension error resulting from `NoneType` descriptors), please restart the execution. Some images produce no keypoints. The implementation uses a subset of random test images due to memory concerns. In a few additional trials (usually at most one), you may achieve error-free execution.

1 Local Features (25 pts)

- Explain SIFT and Dense-SIFT in your own words. What is the main difference?

  Note that the information regarding this section comes from the original paper of Lowe (Distinctive Image Features from Scale-Invariant Keypoints, 2004). SIFT detector is comprised of several steps, namely scale space peak selection, keypoint localization and orientation assignment.

  In scale space peak selection step, the original image is blurred with multiples of a σ (sigma) value (the `sigma` parameter, 1.6 by default), hence producing an octave of images. Each octave is downsampled producing another octave. Along the scale and the spatial coordinates, a set of points are sampled. *nOctaveLayers* (3 by default) represents the frequency/number of samples along the scale within an octave. The samples are compared with their neighbors, accepted if they are local extrema.

  The keypoint localization is made up of two steps. Firstly, the low-contrast samples are eliminated. Some samples in the previous step accepted all local extrema; however, some of them are slightly greater/smaller than the neighboring points. In order to estimate such points, a score $|D(\hat{\mathbf{x}})|$ for each of them is calculated. If it is under some threshold T, then the point is eliminated. The authors use $T = 0.03$, and in OpenCV, *contrastThreshold* is divided by *nOctaveLayers* to obtain T, as per this webpage:

  `https://docs.opencv.org/3.4/d7/d60/classcv_1_1SIFT.html#ad337517bfdc068ae0ba0924ff1661131`

The second step is eliminating the points that are poorly localized along an edge. Remembering the eigenvalues of Harris corner detector, some points are more edge-like (the eigenvalues are further away from each other). Instead of calculating the eigenvalues directly as in Harris's paper, the authors come up with a ratio r, the ratio between the greater eigenvalue over the smaller one. Therefore, $r \geq 1$. If r value of the point is larger than a threshold r' (*edgeThreshold* as referred by OpenCV devs, 10 by default), the point is more edge-like, then rejected.

The orientation assignment step requires precomputed magnitudes and orientations of gradients of the images. For each keypoint, a region is specified around that point. An orientation histogram with 36 bins in 360 degree interval, over the orientation of those points in the region, weighted by their magnitudes and a gaussian function with $\sigma' = 1.5\sigma$ (remember $\sigma = 1.6$ by default) is formed. The peak of the histogram gives the orientation of the keypoint.

Given a set of keypoints with scale/size/radius and orientation obtained from a detection algorithm, SIFT descriptor could be used for matching the keypoints. The descriptor algorithm starts with a region around the keypoints similar to orientation assignment part. Authors used 16x16 grid of points. This grid is also divided into sub-grids (4x4 for each in the text, 16 in total). From each of these subgrids, their weighted orientation histogram with the same weights as in the orientation assignment (magnitude and 1.5σ) are obtained, with 8 bins. Concatenating these 8 bin-histograms, one has $16 \times 8 = 128$ dimensional vector as the descriptor of the keypoint.

Dense-SIFT uses a different kind of detection algorithm, as opposed to SIFT. Its inputs are a set of equally spaced keypoints with a fixed scale and orientation. In OpenCV, the default value for orientation is -1. In the implementation file `the2.py`, there is `DenseSift` class to obtain a set of keypoints of that sort (pay attention to the `detect` method).

- Put your quantitative results (classification accuracy) regarding 5 values of SIFT and 3 values of Dense-SIFT parameters here. In SIFT change each parameter once while keeping others same and in Dense-SIFT change size of feature extraction region. Discuss the effect of these parameters by using 128 clusters in k-means and 8 nearest neighbors for classification.

  In OpenCV implementation, the default values for SIFT are

  `nfeatures` = 0

  `nOctaveLayers` = 3

  `contrastThreshold` = 0.04

  `edgeThreshold` = 10

  `sigma` = 1.6

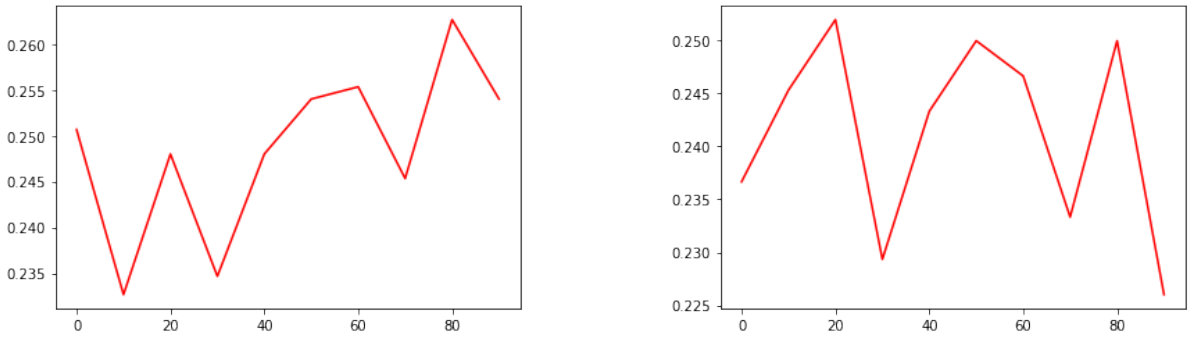
  The results are below

Parameters (others default)	Accuracy
All default	0.26666666666666666
`nfeatures` = 80	0.25333333333333335
`nOctaveLayers` = 10	0.26266666666666666
`contrastThreshold` = 0.08	0.26666666666666666
`edgeThreshold` = 5	0.24733333333333332
`sigma` = 0.5	0.2853333333333333

Table 1: Accuracy results with 128 clusters, 8 nearest neighbors

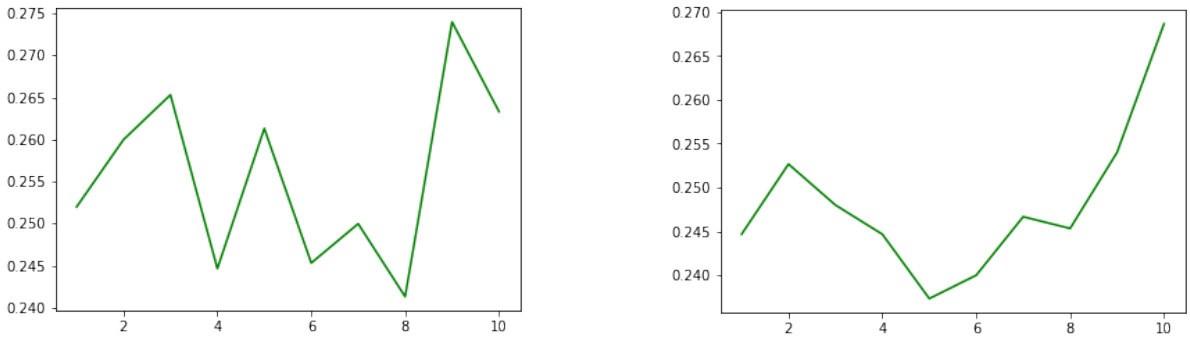
Note that the parameters are chosen regarding the local maxima of the accuracy graph from a set of experiments. In these experiments for each parameter, a graph is plotted over a set of values

in an interval, in a run. There are two runs for each parameter, in order to reduce the effect of random sampling of train sets (different set of training parameters at each run). Graphs for two runs for each parameters are given in figures 1-5 below.



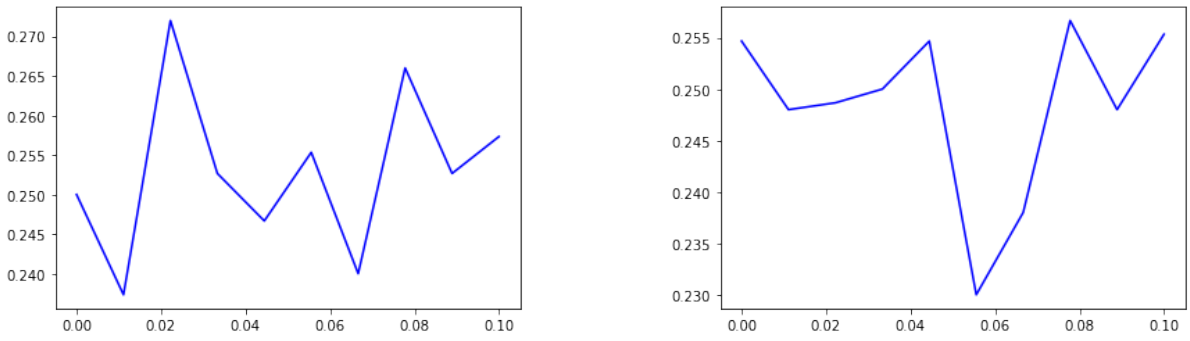
(a) (b)

Figure 1: `nfeatures`



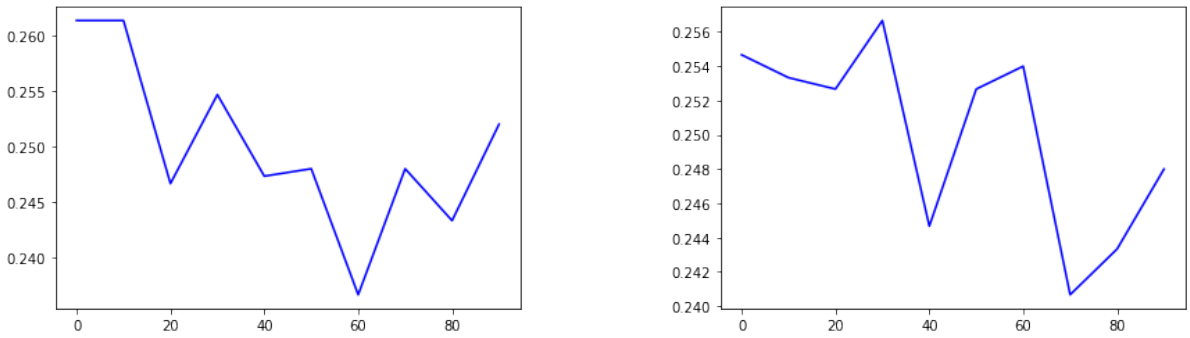
(a) (b)

Figure 2: `nOctaveLayers`



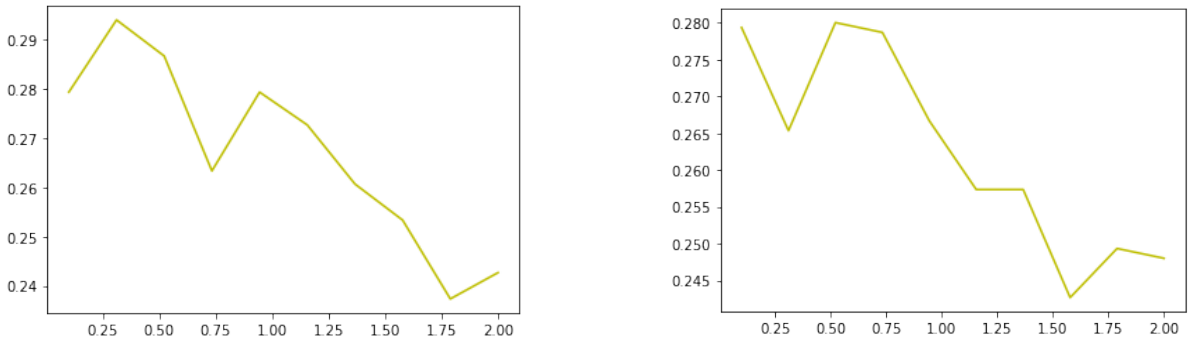
(a) (b)

Figure 3: `contrastThreshold`



(a) (b)

Figure 4: `edgeThreshold`



(a) (b)

Figure 5: `sigma`

Below are the results found in the original paper

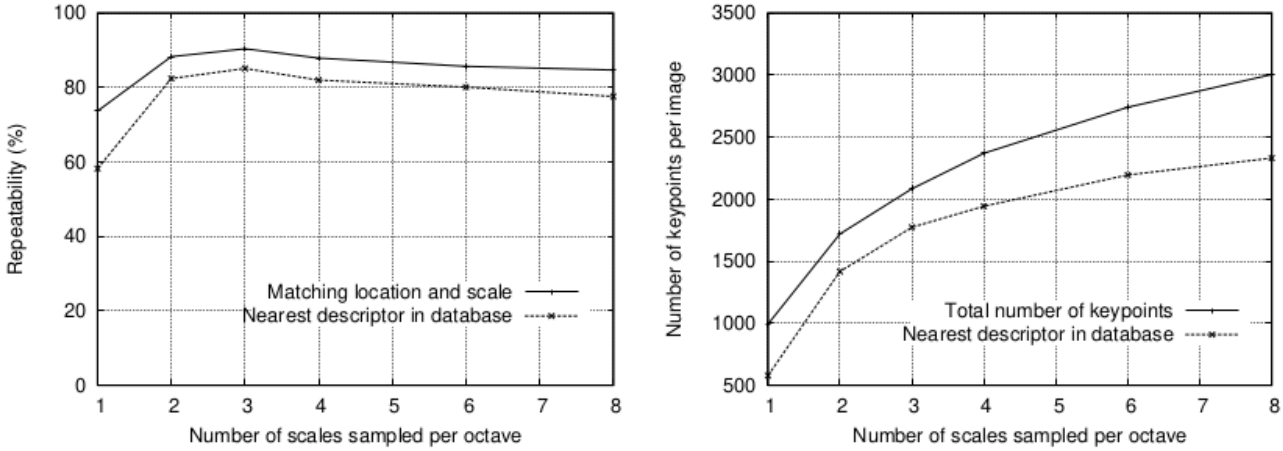


Figure 3: The top line of the first graph shows the percent of keypoints that are repeatably detected at the same location and scale in a transformed image as a function of the number of scales sampled per octave. The lower line shows the percent of keypoints that have their descriptors correctly matched to a large database. The second graph shows the total number of keypoints detected in a typical image as a function of the number of scale samples.

Figure 6: `nOctaveLayers`

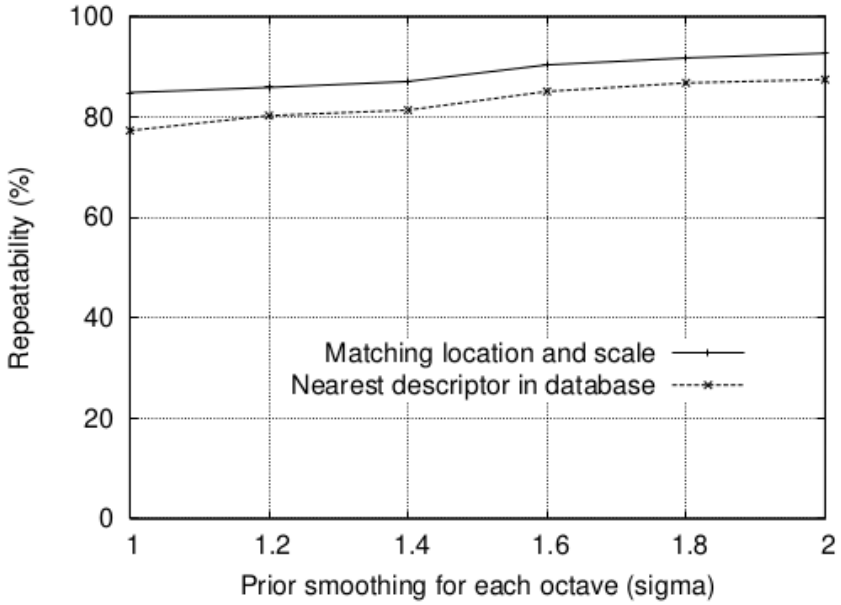


Figure 4: The top line in the graph shows the percent of keypoint locations that are repeatably detected in a transformed image as a function of the prior image smoothing for the first level of each octave. The lower line shows the percent of descriptors correctly matched against a large database.

Figure 7: `sigma`

The results in the original paper differ due to what they measure. Instead of the accuracy, they measure repeatibility; how often the keypoints are detected given transformed versions of images.

In my case, I observe the accuracy over random sets of images between runs. Furthermore, the detection algorithm is different. My implementation uses OpenCV's fast detector contrary to the SIFT detector.

For `nOctaveLayers`, in the original work, 3 is the optimal value in terms of repeatibility. However in my experiment, 3 was a local maxima, and the graph could be interpreted as increasing if we approximate it. Therefore, a large value is chosen in the interval (10). Both `edgeThreshold` and `sigma` show a downward trend contrary to upward trend of `sigma` value in the original paper. Small values are chosen for them (5 and 0.5). `nfeatures` attain its maximum value in both runs at 80, that is chosen. Similarly in `contrastThreshold`, 0.04 is chosen, giving the largest values for both runs.

The results for dSIFT is below

Grid size	Accuracy
4	0.27066666666666667
8	0.236
16	0.22466666666666665

Table 2: Accuracy results with 128 clusters, 8 nearest neighbors

It seems that with increasing grid size, the accuracy falls. I would like to add that for grid sizes 1 and 2, the implementation gave 0.2093333333333334 and 0.244 respectively. This could be due to small grid sizes creating many descriptors that include irrelevant details or misinformation, or there could arise more common words due to abundancy, such as sky or road (Overfitting). After some threshold, quantity of information is so small that it becomes insufficient to describe an object. This is the reverse-effect (Underfitting). Until 4, we are overfitting the data, after that we are underfitting the data.

For dense-SIFT and SIFT, among the observed values grid size of 4 and sigma of 0.5 worked the best. Please note that for the SIFT part, aside from the experiment results of two runs, I repeatedly tested some values multiple times in order to test whether there is a pattern. The values are more or less the same (no significant change). For instance, sigma values were between 0.28 and 0.30. 0.28 was never achieved with other parameters, but sigma. In the next sections, the SIFT implementation with sigma value of 0.5 will be used.

2 Bag of Features (45 pts)

- How did you implement BoF? Briefly explain.

  I will explain the whole pipeline for clarification. It is implemented as a Python Class. One first gives the path of `the2_data` to the initialized object. In initialization method, the important part is the random sampling that has been mentioned multiple times. With `np.random.choice` function, the file paths for the training set is sampled.

  Then there is the `configure` method where one gives the parameters that we have discussed in the previous section. `pipeline` method is the body of the pipeline. As the name implies, `detectAndDescribeAll` detects keypoints of and creates descriptions out of the images given by the file paths for the training set that were constructed in the initialization part. In `kmeansCluster` part, the descriptions are simply clustered together to form the words. I used sklearn's `MiniBatchKMeans` for efficiency. However, the algorithm is still slow for large k.

  `accuracy` method obtains the histograms for validation set out of these words and matches them with the histograms of the training set. Pay attention to the fact that both histogram and classification methods use `getDists` function, which returns the distance matrix. In this function, distance

of two groups are calculated simultaneously (if numpy allows). ith row and jth column gives the distance between ith vector in the first group (`a`), and jth vector in the second group (`b`). Choosing the minimum at each distance with an `argmin` or `argsort` gives the closest elements for each vector in the first group.

- Give pseudo-code for obtaining the dictionary.

  Define pipeline with parameters `kmeansk`, `knnk`

  Detect and describe keypoints of the images that are pointed by some filepaths

  Cluster the descriptions into `kmeansk` words. The procedure uses sklearn's `MiniBatchKMeans`, fits the values with the parameter `kmeansk`, returning the center values of these clusters.

  The algorithm goes on with image representation and classification.

- Give pseudo-code for obtaining BoF representation of an image once the dictionary is formed.

  Define histogram with parameter `image`

  Detect and describe keypoints of `image`

  Get distances from `getDists` function that was explained previously

  Get the indices of the closest (less distant) elements using `np.argmin`, forming a vector of words

  Convert the vector of words to a histogram using `np.histogram` returning the values (not bins).

- Put your quantitative results (classification accuracy) regarding 3 different parameter configurations for the BoF pipeline here. Discuss possible reasons for each one's relatively better/worse accuracy. You are suggested to keep $k \leq 1024$ in k-means to keep experiment durations managable. You need to use the best feature extractor you obtained in the previous part together with the same classifier.

k	Accuracy
32	0.256
64	0.2713333333333333
128	0.2886666666666667

Table 3: Accuracy results with SIFT, sigma value of 0.5, 8 nearest neighbors

It is observed that the accuracy increases with k due to underfitting. I could not execute the program with larger k values due to hardware limitations. However, I assume there will occur a peak point at some threshold k due to overfitting. I will keep the k value as default (128) in the next sections.

3 Classification (30 pts)

- Put your quantitative results regarding k-Nearest Neighbor Classifier for k values 16, 32 and 64 by using the best k-means representation and feature extractor. Discuss the effect of these briefly.

k	Accuracy
16	0.308
32	0.3353333333333333
64	0.30533333333333335

Table 4: Accuracy results with SIFT, sigma value of 0.5, 128 clusters

We could be underfitting when $k = 16$, and overfitting the data when $k = 64$. When k is climbing, the whole picture of nearest neighbors is becoming clearer, less underfitting occurs. However, after a certain threshold the nearest neighbors may carry misleading matches.

- What is the accuracy values, and how do you evaluate it? Briefly explain.

  The accuracy is defined as the ratio between the correctly predicted values to the total number of predictions. It could be calculated by definition, or given a confusion matrix C it could be calculated as $\frac{\text{trace(C)}}{\text{sum(C)}}$ where $sum(C)$ is the sum of all values of C.

- Give confusion matrices for classification results of these combinations.

Predictions / Actual results	orange	camel	elephant	crab	pear	skyscraper	aquarium_fish	woman	cup	lion	flatfish	mushroom	beetle	apple	road
orange	14	10	1	8	4	5	9	2	10	12	2	7	4	10	2
camel	5	27	6	8	3	3	7	2	4	7	3	8	10	4	3
elephant	3	24	24	5	4	7	0	8	2	3	2	3	9	3	3
crab	7	17	0	17	4	2	9	2	2	6	2	15	13	3	1
pear	13	12	3	4	11	6	7	2	7	8	4	2	5	15	1
skyscraper	5	9	2	1	6	61	1	1	6	1	0	0	0	2	5
aquarium_fish	13	3	2	6	3	2	34	1	7	10	3	11	1	3	1
woman	8	13	4	4	4	15	5	7	6	13	2	6	4	9	0
cup	8	7	1	0	5	12	4	0	49	1	2	1	8	1	1
lion	9	17	8	6	2	3	5	6	4	24	1	8	6	1	0
flatfish	10	15	1	1	2	3	10	4	8	8	16	3	4	14	1
mushroom	10	4	1	7	3	3	15	4	7	16	2	22	3	0	3
beetle	12	14	1	7	1	1	1	1	2	7	3	2	41	7	0
apple	19	6	3	0	3	2	8	1	7	4	2	1	3	41	0
road	3	4	1	1	0	2	0	0	7	3	2	0	3	0	74

Table 5: Confusion matrix for $k = 16$

Predictions / Actual results	orange	camel	elephant	crab	pear	skyscraper	aquarium_fish	woman	cup	lion	flatfish	mushroom	beetle	apple	road
orange	8	8	1	5	4	4	11	0	12	14	3	8	8	13	1
camel	4	25	2	6	1	6	7	2	9	10	1	5	14	3	5
elephant	1	20	20	6	2	10	1	9	4	4	3	0	14	2	4
crab	4	9	1	18	3	3	20	0	3	4	2	8	22	2	1
pear	7	10	0	2	11	12	11	1	11	13	2	2	7	11	0
skyscraper	3	3	1	0	0	63	2	1	10	4	2	1	1	2	7
aquarium_fish	6	6	2	5	0	2	47	0	10	5	3	10	3	0	1
woman	7	9	2	1	4	20	9	5	8	16	2	3	7	7	0
cup	2	2	1	1	2	8	10	0	61	0	0	0	7	3	3
lion	2	9	4	7	0	4	4	6	6	36	4	9	8	1	0
flatfish	5	10	1	4	5	7	13	5	7	11	6	5	3	15	3
mushroom	3	7	0	8	0	5	22	1	7	15	0	25	3	2	2
beetle	3	12	1	3	1	3	0	0	6	2	2	0	61	6	0
apple	13	3	2	3	1	3	12	2	11	2	2	2	2	42	0
road	1	4	2	0	1	3	4	0	7	0	0	0	1	2	75

Table 6: Confusion matrix for $k = 32$

Actual results \ Predictions	orange	camel	elephant	crab	pear	skyscraper	aquarium_fish	woman	cup	lion	flatfish	mushroom	beetle	apple	road
orange	1	8	0	4	3	11	11	1	16	13	0	10	8	12	2
camel	2	24	3	1	3	6	7	4	6	8	3	10	14	3	6
elephant	4	23	3	3	8	16	0	10	4	3	1	2	17	1	5
crab	1	14	0	4	4	4	23	0	3	2	2	14	18	9	2
pear	2	14	0	0	9	14	7	2	11	11	2	4	6	16	2
skyscraper	0	4	0	0	1	71	2	1	8	2	0	0	1	2	8
aquarium_fish	1	10	0	1	0	2	52	0	14	7	0	9	3	0	1
woman	3	10	1	0	4	18	6	5	12	16	1	4	10	9	1
cup	2	10	0	1	2	9	12	0	49	0	0	0	7	4	4
lion	1	15	1	0	3	4	5	5	6	38	1	9	7	5	0
flatfish	3	16	0	1	0	10	13	1	10	8	3	6	8	17	4
mushroom	2	6	0	3	0	5	18	0	7	20	1	31	0	3	4
beetle	1	22	2	2	2	3	0	0	5	3	1	1	50	7	1
apple	1	8	0	0	4	8	15	1	15	4	0	2	2	39	1
road	0	4	1	0	0	4	0	0	7	1	0	1	1	2	79

Table 7: Confusion matrix for $k = 64$