CENG 483

Introduction to Computer Vision

Fall 2021-2022

Take Home Exam 2

Object Recognition

Student Random ID: 2375574

Please fill in the sections below only with the requested information. If you have additional things to mention, you can use the last section. Please note that all of the results in this report should be given for the **validation set**. Also, when you are expected to comment on the effect of a parameter, please make sure to fix other parameters.

1 Local Features (25 pts)

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

Sift is a keypoint detector that is designed to capture features that are indepent from the scale of the image. Its scale invariance property stems from difference of gaussian operations with different kernel size resolutions which captures keypoints at different scales. Then we can quantize an orientation for each keypoint and when orientations of all features are described wrt the keypoint orientation, then it'll additionally have rotation invariance alongside scale invariance. These are all desired invariances that are needed to select interesting or descriptive points in images in order to identify, match, classify etc. Dense-Sift is a kind of modification on top of conventional Sift method. In Sift locations of keypoint descriptors are determined by the default algorithm. On the contrary, in Dense-Sift center-positions (x,y) can be fed to Sift algorithm to create descriptors around that point, or equivalently one can partition the image into grid like portions and feed them individually through default Sift which in a way has similar behaviour, resulting in specific sift vectors around that spatial region.

Sift: Parameters vs Accuracy					
Accuracy (1.0)	nfeatures	nOctaveLayers	contrastThreshold	edgeThreshold	sigma
0.17	0	3	0.04	10	1.6
0.10	1	3	0.04	10	1.6
0.18	20	3	0.04	10	1.6
0.16	100	3	0.04	10	1.6
0.19	0	1	0.04	10	1.6
0.16	0	5	0.04	10	1.6
0.17	0	7	0.04	10	1.6
0.0	0	3	1	10	1.6
0.16	0	3	0.1	10	1.6
0.17	0	3	0.01	10	1.6
0.0	0	3	0.04	1	1.6
0.19	0	3	0.04	50	1.6
0.19	0	3	0.04	100	1.6
0.0	0	3	0.04	10	0.1
0.15	0	3	0.04	10	1
0.17	0	3	0.04	10	2
0.17	0	3	0.04	10	3
0.0	0	3	0.04	10	10

In this experiment, k-value in kmeans is set to 128 cluster and k value of k_nn neearest neighbor is set to 8 as required. By changing the parameters of SIFT, I've obtained significant changes on the accuracy of our classifier.

Firstly, nfeatures determines the number of top features in terms of local contrast scores to keep in the sift descriptor. Default nfeatures value takes all of the features in the descriptor. When nfeatures is set very low like 1, I've observed that accuracy drops drastically which makes sense because we are dropping discriminative sift vectors and this information loss has a cost on the overall accuracy. However, nfeatures=20 or 100, more or less yields the same accuracy which probably means that top 20 sift vectors were discriminative enough to make classification equivalent to the default param that is used in sift.

Secondly, nOctaveLayers are number of difference of gaussian layers which helps capturing features at varying scales. If we have more nOctaveLayers then that would be more features at varying scales but with exponentially (with order of k^2) more blurring. This creates new features at very high scale due to higher blurring as a result of increased nOctaveLayers value. As far as I understand, in sift they have probably chosen the elbow value which should be somewhere around 3, because after/smaller than 3 it starts to decline very sharply. In addition, having less nOctaveLayers might be disadvantageous too depending on the problem and image properties, hence I think 3 would be a better or less risky choice when compared to 1,2 which results in less scale invariant descriptors.

Thirdly, contrastThreshold parameter filters weak feature below a threshold value measured in contrast. When it's set higher, we would get less features and when it's set lower, we would get more features. However, more features don't always imply more accuracy, some redundant features can hurt our performance in classification as we'll see. I've observed that increasing contrastThreshold, results in less sift descriptors. For instance, contrastThreshold=1 results no sift vector, because it's too high and

contrastThreshold=0.001 results in most number of sift descriptors. ContrastThreshold is a hyperparameter that we should tune for our problem and in our case 0.04 has resulted in the best accuracy, because it has discarded less important features and emphasized on more important features.

Fourtly, edgeThreshold is a hyperparameter that basically filters edge like features, its meaning is slightly different than contrastThreshold in the sense that, higher edgeThreshold value results in more features, because it basically eleminates features above that threshold not below. So, higher threshold results in more features, but once again that doesn't imply more accuracy, we still need to tune our param to give priority to important edges. As it's clear from our experiments, low edgeThreshold values yields less features, edgeThreshold=1 is an extreme case which results in no sift descriptors. However, after some point accuracy doesn't improve as threshold gets larger.

Finally, sigma parameter is the sigma of the gaussians applied at each octave layers in sift. Higher sigma implies more blurring, which would encourage sift to capture features at higher scales and lower sigma would encourage sift to capture features at smaller scale. As sigma increases, number of sift descriptors decrease, because blurring kills some details. Also, low sigma values have low accuracies, because we capture features at lower scales more frequently. That's why we need to form a balance for the problem by tuning the hyperparameter.

Dense-Sift: Parameters vs Accuracy (default sift parameters)			
Accuracy (1.0)	grid_size	keypoint_diameter	offset
0.27	8	4	4
0.20	16	4	4
0.12	32	4	4
0.07	8	0.1	4
0.14	8	1	4
0.20	8	10	4
0.25	8	4	0
0.27	8	4	2
0.25	8	4	8

In Dense-Sift, it's mentioned only grid-size parameter, but in order to extract key points at certain locations, we have to pass a keypoint diameter. Hence, I've parametrised over grid-size, keypoint-diameter, and also an offset value to skip edge pixels and focus on inner part of the image as part of the experimentation.

Firstly, grid-size parameter adjusts spatiality of our sift descriptors, as expected as spatiality decreases we get less accuracy. Grid-size 8 has resulted in the best accuracy, whereas grid-size 32 with only one keypoint has very poor accuracy. However, if we have too little grid-size s like 1x1, I would expect not a very good accuracy, because too much unnecessary keypoints would overwhelm our classification leaving less emphasis on important keypoints.

Secondaly, keypoint-diameter adjusts the radius of the keypoint at that (x,y) location. In other words, it corresponds to the diameter of the keypoint circles. As it's illustrated in our experiment that too low and too high keypoint-diameters result in less accuracy. This parameter is also in a way correlated with grid-size, because as our grid-sizes decrease our key-point diameter should also be decreased, because otherwise it wouldn't be meaning to have large circles overflowing smaller grids.

Thirdly, offset value is how much we skip from edges of an image. It is a reasonable assumption that

some pixels from edge are not that interesting and we focus more on inner part of the image. Our observations make sense, when we increase offset too much we get less accuracy due to information loss, however by tuning the parameter we can increase accuracy with around 1-2 points.

2 Bag of Features (45 pts)

- How did you implement BoF? Briefly explain.
- Give pseudo-code for obtaining the dictionary.
- Give pseudo-code for obtaining BoF representation of an image once the dictionary is formed.
- 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.

I've implemented BoF through several steps. Firstly, I've iterated over all images and extracted their sift vectors/descriptors as in the from (m, 128) where m is the number of detected keypoints and 128 is the length of the sift vector. If there was no sift-vector which happened rarely, I've added a null vector with all zeros in the form of (1,128) to describe that case, in order to comply with the general BoF model. Then, I've saved all of these sift descriptors for each image which we now the class due to training set. Then I've stacked all of these vectors of images all together, and applied kmeans with k=128 and iters=15. That gives me a codebook of size (128,128) where we have 128 many centroids of sift vectors of size 128. I've implemented my own knn algorithm to get k-nearest neighbors of sift descriptors. Also, I used my own knn implementation to get 1-nn for bag of words representation calculation, where I assign sift descriptors of an image to its neares centroid in the codebook and I keep a statistics or histogram of these codebooks/bins for each image. Then I normalize these histogram such that each image codebook histogram sums to 1, aka l-1 norm. In the second phase, I calculate accuracy over test-set and I apply the same stages as I did in training namely extracting sift vectors, assigning vectors to centroids, and calculation normalized bow representations for each test image. Then, I run knn over normalized test image bow representation wrt normalized training image bow representation with k=8 as required to get 8 neares neighbor and I take the dominant class out of these votes to make a prediction. Finally, I calculate my test accuracy since I've the labels.

Pseudo-code for dictionary:
1. Obtain sift descriptor vectors for each image in the set.
2. Stack sift descriptor vectors all-together.
3. Run kmeans over the descriptor stack with parameters k and iters.
4. Returns codebook/dictionary of shape (k,128) including k-centroids of 128-d sift vectors.

Pseudo-code for BoF representation:
1. Apply for each image in the set/
2. Calculate knn with k=1 (same as 1-nearest neighbor) between codebook (pre-calculated dictionary of (kmeans_k,128)) and image sift descriptors.
3. Assign each image descriptor vector to its closest centroid in codebook
4. Count the number of occurances of each codebook vector for that image (histogram)
5. Normalize histogram representation such that it sums to 1.
6. Returns normalized BoF representation for that image

Dense-Sift: Parameters vs Accuracy (default sift parameters)		
Accuracy (1.0)	kmeans_k	kmeans_iters
0.27	8	15

The feature extractor for these results are from Dense-Sift with default sift parameters and grid-size=8, keypoint-diamater=4, offset=4 which is used a baseline comparison model.

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.
- What is the accuracy values, and how do you evaluate it? Briefly explain.
- Give confusion matrices for classification results of these combinations.

4 Additional Comments and References

(if there any)