CENG 483

Introduction to Computer Vision

Fall 2021-2022

Take Home Exam 2

Object Recognition

Student ID: 2232320

1 Local Features (25 pts)

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

  The SIFT descriptor is invariant to translations, rotations and scaling. This invariant characteristic makes it stand out from the other descriptors. In SIFT, it divede picture to 4x4 grid and for each grid local orientation histograms is collected. This histograms is formed with 8 bins. Therefore, when we examine dimensions of descriptors we can see that 8x4x4=128 dimension. Descriptor dimensions is like (number of keypoint, 128). In dense sift, more features are collected. For, example, first image of the experiment dataset. I find 5 keypoint in SIFT and 49 keypoint in dense sift. In SIFT, keypoints are collected by Lowe's algorithm whereas in dense sift are collected from every locations I gave. I gave this locations and image to the sift detector in dense sift. This operation increase the accuracy.

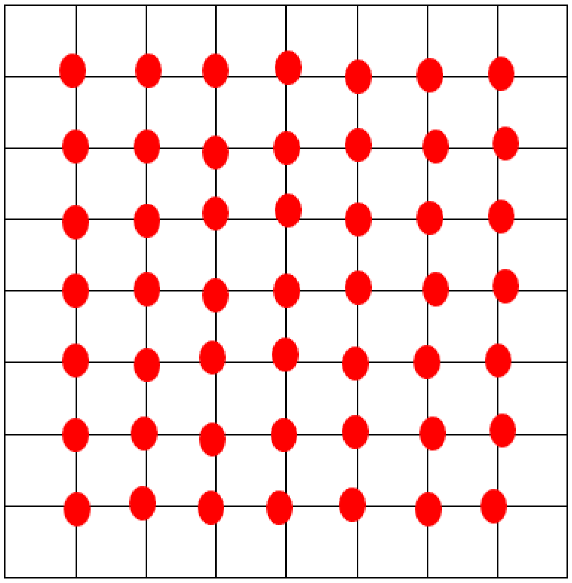


Figure1: Dense SIFT keypoints example

- 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.

  1. SIFT experiments and classification accuracies. Note that every parameter is changed one by one. Other parameters stay same while changing one parameter.

     (a) nfeatures: I try following values to find behavior and tendency of this paremeter and taking the best accuracy among them. [5,15,25,100,1000]. Here is my results

       – nfeatures: default
         k for KNN: 8
         k for kmeans: 128
         accuracy: 0.186

       – nfeatures: 5
         k for KNN: 8
         k for kmeans: 128
         accuracy: 0.14066666666666666

       – nfeatures: 15
         k for KNN: 8
         k for kmeans: 128
         accuracy: 0.18333333333333332

       – nfeatures: 25
         k for KNN: 8
         k for kmeans: 128
         accuracy: 0.18733333333333332

       – nfeatures: 100
         k for KNN: 8
         k for kmeans: 128
         accuracy: 0.186

       – nfeatures: 1000
         k for KNN: 8
         k for kmeans: 128
         accuracy: 0.186

         Comment for nfeatures:
         Accuracy decrease because in default number of feature for images changes. Lots of the images have features between 5-25 therefore limit in number of feature with 5 causes the loss of features. This leads to decrease in accuracy. In other words, less features leads to less accuracy. But at some value we see that even if number of features increase accuracy stay same. This means accuracy converge 0.186. I think, every image extract their features and when I look that in debug mode feature are between mostly 5-25. Therefore, nfeatures=100 or nfeatures=1000 doesn't mean anything because all features that can be extracted are already extracted. So, default value is a good choice for nfeatures. nfeatures 25 gives a 0.001 better result but I don't think it's appropriate to limit features with 25. May it depends on the validation set

images. Therefore, my choice is default settings. Default settings is 0 according to documentation but probably it means that how many features are found, so many features will be added.

(b) nOctaveLayers

- nOctaveLayers: default
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.186

- nOctaveLayers: 1
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.186

- nOctaveLayers: 5
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.19

- nOctaveLayers: 10
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.19533333333333333

- nOctaveLayers: 50
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.196

- nOctaveLayers: 100
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.20066666666666666

- nOctaveLayers: 200
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.06666666666666667

- nOctaveLayers: 1000
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.06666666666666667

  Comment for nOctaveLayer:
  Accuracy increase when number of OctaveLayers increase but after certain point model totally diverge. It increases until 100 but when it is equal to 200 model totally diver accuracy = 0.0666 which equals to random predict on for 15 class. Also, when number of OctaveLayers increase prediction time also increases. I want to granted that model totally diverge after certain point therefore I add 1000 for nOctaveLayer.

So, my best result is 100 for nOctaveLayers with 0.114 accuracy increase compared to default.

(c) contrastThreshold

- contrastThreshold = default
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.186

- contrastThreshold: 0.01
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.16533333333333333

- contrastThreshold: 0.02
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.17133333333333334

- contrastThreshold: 0.04
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.186

- contrastThreshold: 0.07
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.18266666666666667

- contrastThreshold: 0.09
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.17333333333333334

- contrastThreshold: 0.15
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.14733333333333334

- contrastThreshold: 0.2
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.09466666666666666

- contrastThreshold: 0.4
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.06666666666666667

- contrastThreshold: 0.7
  k for KNN: 8

k for kmeans: 128
accuracy: 0.06666666666666667

Comment for contrastThreshold:
Accuracy decrease when contrastThreshold decrease from the default value. When contrastThreshold increase from 0.01 to 0.04 which is default, accuracy increase. However, after 0.04 increase in contrastThreshold cause decrease in accuracy. Similar to OctaveLayers, after some point(in my experiment value between 0.2 to 0.4 which are very huge comparing to default 0.04) model totally diverge and give results same as random prediction So, default value is a good choice for nfeatures.

(d) edgeThreshold

- edgeThreshold = default
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.186

- edgeThreshold: 1
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.06666666666666667

- edgeThreshold: 3
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.16466666666666666

- edgeThreshold: 5
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.14466666666666667

- edgeThreshold: 10
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.186

- edgeThreshold: 12
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.18533333333333332

- edgeThreshold: 15
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.17066666666666666

- edgeThreshold: 35
  k for KNN: 8

k for kmeans: 128
accuracy: 0.182

- edgeThreshold: 100
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.16066666666666668

- edgeThreshold: 200
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.19066666666666668

- edgeThreshold: 300
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.17733333333333334

- edgeThreshold: 400
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.188

Comment for contrastThreshold:
Accuracy decrease when edgeThreshold decrease from the default value. When edgeThreshold increase from 1 (equal to random) to 10 which is default, accuracy increase. However, after 10 increase in edgeThreshold cause decrease in accuracy. Similar to OctaveLayers, after default value, it is observed that there is a decrease in accuracy. Actually, I expect increase in accuracy to a point and converge a certain value similar to nfeatures because documentation says that "The larger the edgeThreshold, the less features are filtered out (more features are retained)."

(e) sigma

- sigma: 0.001
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.06666666666666667

- sigma: 0.4
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.15266666666666667

- sigma: 0.8
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.15733333333333333

- sigma: 1.6
  k for KNN: 8
  k for kmeans: 128

accuracy: 0.186

- sigma: 2.4
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.16666666666666666

- sigma: 3.6
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.15933333333333333

- sigma: 7
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.11466666666666667

- sigma: 10
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.08533333333333333

Comment for contrastThreshold: Results of sigma value's behavior is actually expected. It increase some value 1.6(is also default) and decrease after that peak value. It uses in Gaussian formula and Gaussian behaves same. When sigma become very large and very small. Model totally diverges. In Gaussian graph, when sigma value become very high, graph spread out and all values approach 0 and when sigma value become very low, the graph narrows and becomes steeper in this narrow space. Both condition leads to diverge models. The best sigma value is 1.6, so I choose the default value.

2. Dense-SIFT experiments and classification accuracies

   (a) I adjust the size of dense sift 2,4 and 8. my results are as follows

   - scale: 4
     size: 2
     offset: 4
     k for KNN: 8
     k for kmeans: 128
     accuracy: 0.2966666666666667

   - scale: 4
     size: 4
     offset: 4
     k for KNN: 8
     k for kmeans: 128
     accuracy: 0.32133333333333336

   - scale: 4
     size: 8
     offset: 4
     k for KNN: 8

k for kmeans: 128
accuracy: 0.2866666666666667

Dense SIFT increases the accuracy considerable. It add new key points. These key points are extracted by dividing grids. It was explained in the previous question more detailed. In my experiment, offset is the first grid point (offset is 4 means first point is located (4,4)). Scale is the size of grid (scale 4 means size of grid is 4x4 pixel). Size is the diameter of the meaningful keypoint neighborhood for cv2.keypoint function. My examples for scale and offset change is in the 4. part Additional comment. The best size value is 4, so I choose the size value as 4.

2 Bag of Features (45 pts)

- How did you implement BoF? Briefly explain.

After obtaining dictionary which include object keeping all images' class, their own descriptor and an empty part. In BOF implementation, I loop the dictionary. For every image, I use cluster.predict(image's descriptor). This returns which cluster center the every image's features are assigned to. For example, there is 5 feature for an image, cluster.predict give 5 center of clusters. Then, these clusters are counted in a list which is the same length with k of kmeans and then this list is normalized. This operation enable us to obtain the histogram of cluster's center of that image. I add histogram to the empty part of my object in the dictionary because I don't lose the information of image class name and its histogram.

- Give pseudo-code for obtaining the dictionary.

Algorithm 1 Dictionary

```
1: descriptor = empty numpy array with size(0,128)
2: for $class = apple, aquariumfish, \ldots$ do
3:     for $classimages = 1, 2, \ldots, 400$ do
4:         descriptor=feature_extraction(image, SIFT_type, SIFT_parameters)
5:         obj = class name, descriptor, empty histogram
6:         append this obj to dictionary
7:         concatenate descriptor
8:     end for
9: end for
10: clusters = MiniBatchKMeans(n_clusters=k).fit(descriptor)
11: clusters_center = clusters.cluster_centers
```

Cluster is enough for dictionary. I just want to show clusters center to emphasize dictionary. //

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

Algorithm 2 BOF

```
for $j = 0, 1, \ldots$(len(catalog)) do
    feature_distribution = numpy.zero(kmenas_k)
    predictions = cluster.predict(dictionary[j].descriptor)
    for p in predictions do
        feature_distribution[p]++
        summation = sum(feature_distribution)
        normalize_feature_distribution = feature_distribution / summation
        dictionary[j].histogram = feature_distribution
    end for
end for
save(dictionary)
```

- 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.

  Up to now, my best configuration is that Dense SIFT detector with size 4 (accuracy = 0.321). Now, I obtain the best k for kmeans with best feature extractor configuration.

  1. k for kmeans: 32
     size: 4
     k for KNN: 8
     accuracy: 0.2853333333333333

  2. k for kmeans: 64
     size: 4
     k for KNN: 8
     accuracy: 0.29733333333333334

  3. k for kmeans: 128
     size: 4
     k for KNN: 8
     accuracy: 0.32133333333333336

  4. k for kmeans: 256
     size: 4
     k for KNN: 8
     accuracy: 0.3273333333333333

  5. k for kmeans: 512
     size: 4
     k for KNN: 8
     accuracy: 0.32666666666666666

Comment for Kmeans k
In kmeans, we try to divide our data to most appropriate cluster and find its cluster center. Therefore, the main point is how our data distribute in space. For example, in the following figure, clusters number and their center change data to data. In our experiment, we cluster the descriptor which we find from SIFT or dense SIFT. Therefore, we need to find most appropriate number of cluster for our data. According to experiments' result, Accuracy increase to 256 wit increase in k but after 256 accuracy is decrease. So, the best kmeans k is equal to 256 with accuracy 0.327.

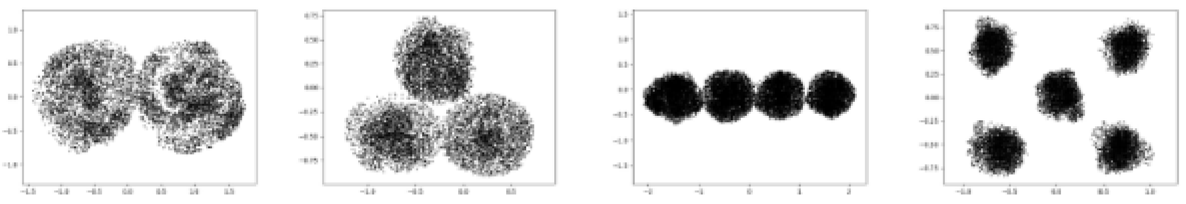

(a) clustering1 (b) clustering2 (c) clustering3 (d) clustering4

Figure2: Kmeans example

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.

  Up to now, my best configuration is that Dense SIFT detector with size 4 (accuracy = 0.321) and k for kmeans is 256. Now, I obtain the best k for knn with this best configuration.

  1. k for KNN: 8
     size: 4
     k for kmeans: 256
     accuracy: 0.327

  2. k for KNN: 16
     size: 4
     k for kmeans: 256
     accuracy: 0.348

  3. k for KNN: 32
     size: 4
     k for kmeans: 256
     accuracy: 0.344

  4. k for KNN: 64
     size: 4
     k for kmeans: 256
     accuracy: 0.3426666666666667

Comment for KNN k

KNN k specifies how many points we will look at closest to our point when decide our point class. Therefore, small number of k can mislead also large number can mislead. In my experiment, k=8 gives less result than k=16. k=32 and 64 give less result than k=16 as well. It is similar to the following figure. When k=3, prediction is green rectangle; on the other hand, when k=7 prediction is red star. Again, when k is very high, prediction is green. Therefore, we can't say big knn is almost good or small knn is almost bad. We need to tune knn value and in my case. The best result is for knn k = 16 with accuracy 0.348

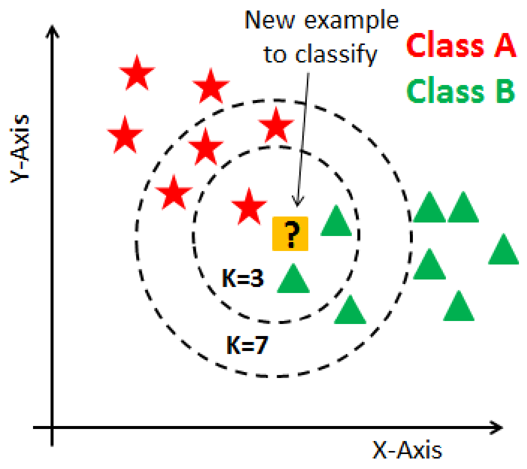
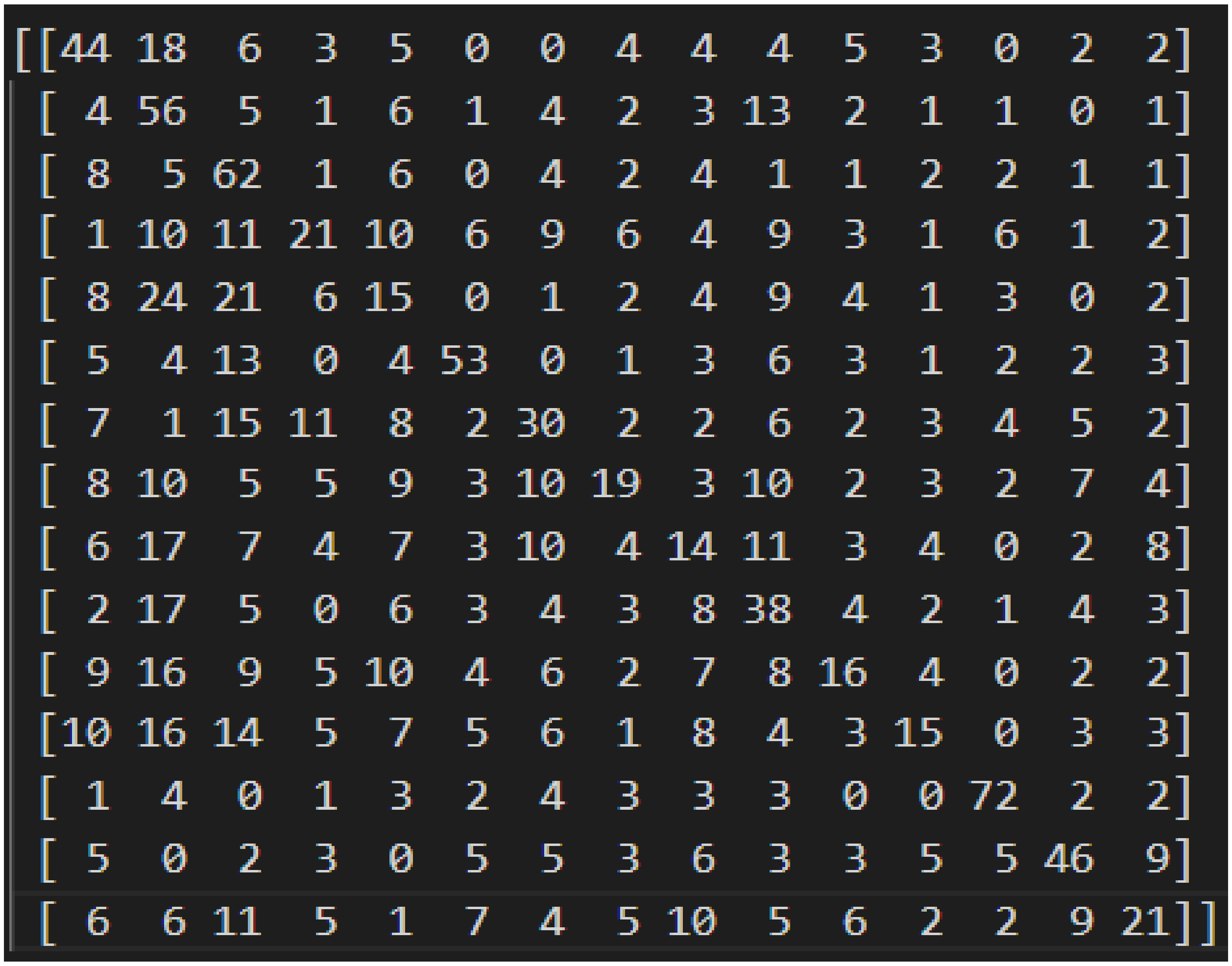


Figure3: KNN example

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

  I take best as 0.348. I write a KNN classifier. KNN classifier takes the histogram in the set and train set. Making a distance table from these histograms and choosing K nearest neighbor and predict class accordingly. After that if predicted class and actually class is same increment accuracy by 1. Then take average for test set.

- Give confusion matrices for classification results of these combinations.



```
[[44 18  6  3  5  0  0  4  4  4  5  3  0  2  2]
 [ 4 56  5  1  6  1  4  2  3 13  2  1  1  0  1]
 [ 8  5 62  1  6  0  4  2  4  1  1  2  2  1  1]
 [ 1 10 11 21 10  6  9  6  4  9  3  1  6  1  2]
 [ 8 24 21  6 15  0  1  2  4  9  4  1  3  0  2]
 [ 5  4 13  0  4 53  0  1  3  6  3  1  2  2  3]
 [ 7  1 15 11  8  2 30  2  2  6  2  3  4  5  2]
 [ 8 10  5  5  9  3 10 19  3 10  2  3  2  7  4]
 [ 6 17  7  4  7  3 10  4 14 11  3  4  0  2  8]
 [ 2 17  5  0  6  3  4  3  8 38  4  2  1  4  3]
 [ 9 16  9  5 10  4  6  2  7  8 16  4  0  2  2]
 [10 16 14  5  7  5  6  1  8  4  3 15  0  3  3]
 [ 1  4  0  1  3  2  4  3  3  3  0  0 72  2  2]
 [ 5  0  2  3  0  5  5  3  6  3  3  5  5 46  9]
 [ 6  6 11  5  1  7  4  5 10  5  6  2  2  9 21]]
```

from left to right and up to bottom

  - apple
  - aquarium_fish
  - beetle
  - camel
  - crab
  - cup elephant
  - flatfish
  - lion
  - mushroom
  - orange
  - pear
  - road
  - skyscraper
  - woman

4 Additional Comments and References

- scale: 4
  size: 4
  offset: 4
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.32133333333333336

  This is my default and the best configuration

- scale: 4
  step: 4
  offset: 2
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.312

  When offset 2 our locations for bot x and y 2,6,10,14,18,22,26,30. This is same number of locations with offset 4 in default. Therefore, result is very close with my default

- scale: 4
  size: 4
  offset: 8
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.30933333333333335
  When offset 8 our locations for bot x and y 8,12,16,20,24,28,32. This is less number of locations with offset 4 in default and it include edge of the image 32'nd pixels Therefore, result is very close with my default. Therefore, there is more drop.

- scale: 2
  size: 4
  offset: 4
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.31866666666666665

  When grid scale is 2 grid size is 2x2. This make a very small area but more keypoint. However, these keypoint may be irrelevant therefore, it cause some very small drop.

- scale: 8
  size: 4
  offset: 4
  k for KNN: 8
  k for kmeans: 128
  accuracy: 0.278

  When grid scale is 8 grid size is 8x8. This make a very small big but less keypoint. Less keypoint means less information therefore, there is a considerable drop in accuracy compared to default.