Object Recognition With Local Features

Alperen Bitirgen

1 Local Features

1.1 SIFT

The main idea here is that we can generate an inference about the classification of the object based on the similarities of the objects in the different pictures. We identify components that we can associate with the objects we see in the picture and which we call 'keypoints', while we make use of Lowe's algorithm. We make use of the DoG(Difference of Gaussian) method while deriving these components, and the most important advantage of this is that it can provide us with scale-invariant 'keypoints'. These 'keypoints' are used in the derivation of the components we call 'descriptors', which is the main finding for us. These components, in fact, enable us to make inferences about the structures that we can define as 'features', that is, the properties of the objects in the visual we examine.

1.1.1 *Parameter Configuration on SIFT*

During my implementation of this part, before trying some configurations for the SIFT function, I started with reading the OpenCV Reference[**4**] page to to help me analyze the potential effect of parameter configurations. Then I decided the change and amount of change I want to make to the parameters as below.

1.1.1.1 *nfeatures*

This parameter initially set to 0, which I think manages an internal process to obtain the number of features to extract without fixing the number to a some number. Initially, while using the training data, I obtained around 73039 descriptors. So, If I want to increase the number of features to examine I need to increase this number. Considering the number of images in train data-set, on the average each image helps to observe around 12 descriptors, and I decided to set the nfeatures to 16 to obtain a greater amount of descriptors which will probably then yield a better dictionary to classify images. This, parameter adjustment help me to increase my accuracy with about 0.022, which , I think a pretty good improvement.

1.1.1.2 *nOctaveLayers*

This parameter stated to be computed automatically but initially default value is 3. Since the each octave layer carries some portion of information about the image, I wanted to set it to a slightly greater value, that is 4. In a sense, I considered the count of octave layers as primitive version of CNN channels obtained with different kernels. However, in this case, by using SIFT, we use the Gaussian filters to be exact. After applying this change, my accuracy increased as I expected, about 0.043.

1.1.1.3 *contrastThreshold*

This parameter is mainly for adjusting the boundary for weak and strong features to be extracted from images. I wanted to assign a slightly larger value to this parameter, taking into account that the image quality of the pictures I have is seriously low and that features with values slightly above this threshold may actually be weak features. Therefore, I decided to set it to 0.045, which 0.04 is the default value, and obtained a better accuracy than the default values.

1.1.1.4 *edgeThreshold*

This parameter causes us to get rid of the edge-like features. In a normal sense, probably we would not want such a thing. However, since we have very distorted and low resolution images, algorithm might use those areas in favor of better accuracy. To achieve that, I wanted to increase this value and this way decrease the amount of filtered out features. Not surprisingly, I obtained a slightly better accuracy then the defaults, but it did not have a considerable impact.

1.1.1.5 *sigma*

This sigma value is actually the sigma value of the Gaussian filter applied. Normally, we use the Gaussian filter to blur the images. However, as we already have very insignificant shapes of objects in the images, decreasing this value a little might help us better examine while extracting features. Also, in the referenced page, it is suggested to decrease this value if we have low resolution images. I set a little smaller value, however it did have very vaguely effect on the performance.

Description/Change	nfeatures	nOctaveLayers	contrastThreshold	edgeThreshold	sigma	accuracy
default	0	3	0.04	10	1.6	0.156000
sigma	0	3	0.04	10	1.5	0.171333
edgeThreshold	0	3	0.04	12	1.6	0.174000
contrastThreshold	0	3	0.045	10	1.6	0.179333
nOctaveLayers	0	4	0.04	10	1.6	0.199333
nfeatures	16	3	0.04	10	1.6	0.178000
my preference	0	4	0.045	10	1.6	**0.200666**

Table 1: SIFT classification accuracy with adjusted parameters. (k = 128 for K-Means, k = 8 for K-NN)

1.1.2 Best configuration

For the best configuration, I used the values in the my preference row.

1.2 Dense-SIFT

While defining Dense-SIFT, I should mention the biggest difference from SIFT, which is that we don't use Lowe's algorithm when creating 'keypoint' components and creating 'descriptor' components on top of these components. What we do at this point is to build the 'descriptor' components on top of that by making use of the 'keypoint' components that we distribute as grids on a frame. In this way, Dense-SIFT helps us have more 'descriptor' components and therefore more 'feature' elements for the image we are examining.

1.2.1 *Parameter Configuration on Dense-SIFT*

For this case, I first fully understand the idea of Dense-SIFT by using the Dense-SIFT Visual Explanation[3]. Then examined the sample implementation on GitHub[2]. After that, I thought about the change and amount of change I want to make to the parameters, like this:

1.2.1.1 *bound*

This parameter allows us to assume a padding-like structure, as if there was a frame in the outermost part of our image. While using this parameter, on the one hand, we think that we cannot get the useful features we want at the extreme points, on the other hand, we can make better use of the part inside the bound. Therefore, decreasing this value a bit so that it can help us to extract more useful features would be the idea. However, I couldn't get the improvement I predicted on accuracy. Then, I realised that by just itself this parameter cannot help much; it needs to be adjusted with according to the others parameters as well.

1.2.1.2 *step_size*

In short, it allows us to determine how many pixels we will create a keypoint in the area within the bound. So, decreasing this parameter will allow us to sample more features. However, being too small can increase the running time of the algorithm excessively. Since we have images of 32x32 shape, I wanted to use a number that 32 still be divisible. As I interpret, his parameter set, helped to increase the accuracy.

1.2.1.3 *scale*

This parameter helps us to determine diameter of the meaningful keypoint neighborhood[5].Since we have a very low resolution image, decreasing the value of this parameter may help us.Such that, we do not need to interfere with a very large area and decide according to a more compact region which will probably determine a more characteristic feature. I also decreased this value o 4 and observed that my assumption has a supportable point.

Description/Change	**bound**	**step_size**	**scale**	**accuracy**
default	8	8	8	0.261333
bound	4	8	8	0.268000
step_size	8	4	8	0.285333
scale	8	8	4	0.272000
my preference	4	4	4	**0.321333**

Table 2: Dense-SIFT classification accuracy with adjusted parameters. (k = 128 for K-Means, k = 8 for K-NN)

Best configuration: For the best configuration, I used the values in the my preference row.

2 Bag of Features

2.1 BoF Implementation

At first, we create a kind of dictionary by using the features we get from the pictures we give as input to our algorithm. To create this dictionary structure, we create clusters with the descriptors we get from our inputs, and we detect which objects tend to feature distributions over these clusters and save them as histogram vectors. Then, we create the histogram vectors of these pictures by using the descriptors we have extracted from other pictures that we have not given as input to the algorithm before. In this way, we try to decide what the object in the picture is according to a certain number of features that we want to decide, by comparing the histogram vector of the picture we want to decide, according to their distances in a spatial structure with the features in the feature dictionary that we have previously obtained.

2.2 The pseudo-code for obtaining the dictionary

---

for each image in train data set:
local-variable <– read image in gray-scale
keypoints, descriptors <– obtain from gray-scale image using sift/dsift
descriptor-container <– append the descriptors
data-container <– append (class-label, descriptors) object

---

create a k-means cluster using descriptor-container

---

for each (class-label, descriptors) object of data-container:
feature-vector <– k-means cluster predictions of current data-container object
histogram <– calculate histogram by using feature-vector
instance-container <– append (class-label, histogram) object as a word
dictionary <– instance-object consists of (class-label, histogram) objects as words

---

2.3 The pseudo-code for obtaining BoF representation of an image once the dictionary is formed

---

local-variable <– read image in gray-scale
keypoints, descriptors <– obtain from gray-scale image using sift/dsift
cluster <– load the cluster previously structured
feature-vector <– using the cluster, predict the image by its descriptors
histogram <– sample the feature-vector to obtain BoF representation

---

2.4 Quantitative Results and Possible Reasons:

I wanted to try 256, 512, 768 values for K-Means clustering. This k, the cluster count of k-means, represents how many distinguishable word we have in our dictionary. We used the descriptors from the train data-set to generate these words. While creating the cluster, we created the words according to the k value given. So changing the value of k gives us a bigger or smaller dictionary. In this context, a decrease in the value of k may mean that the words that will describe some features lose their clarity, and therefore it will result in lowering the accuracy value. On the other hand, we can increase the number of words in the dictionary by increasing the k value, and the clearest benefit of this is that it makes it easier to separate the small differences between the words. Thanks to the separation of these small differences from each other, we can give more accurate results when deciding what objects are. Of course, one handicap is that, if we increase this k value to a very large extent that will cause algorithms to suffer from time complexity and will not do much help for classification since the words are so separated from each other and observing the relations between that got so much harder.

K in K-means	**Accuracy**
128	0.321333
256	0.328666
512	0.332666
768	**0.340666**

Table 3: Dense-SIFT classification accuracy with adjusted k values for K-Means. (k = 8 for K-NN)

Best configuration: For the best configuration, I used 768 for K-Means.

3 Classification

3.1 Effects of k value for k-nn

Using the features we obtained from the images in the previous sections, which we named words, we determined which object has what kind of word distribution and we called it a histogram. We can think of these histograms as relational vectors within a spatial structure. What we want to do when using K-NN is to select the k amount of vectors that are closest to the histogram of the new image we have from this spatial structure and select the object type with the highest number among these k amount of vectors as the type of the object in the new image. If this number is small, it will cause the objects to be confused with each other more and decrease the accuracy value, since enough similar vectors cannot be used. Therefore, by carefully increasing this value, we can more reliably exploit similarities within the spatial structure. However, increasing this value will not always give us a better result, because an increase in this value will cause a biased estimation to be made by taking these dissimilar object types into account, even though the similarity rate is low.

Best configuration: For the best configuration, I used 32 for K-NN.

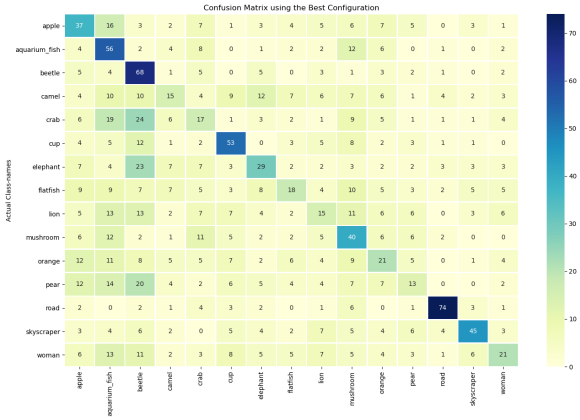
K in K-NN	Accuracy
8	0.340666
16	0.348000
32	**0.353333**
64	0.342000
24	**0.358000**

Table 4: Dense-SIFT classification accuracy with adjusted k values for K-NN.

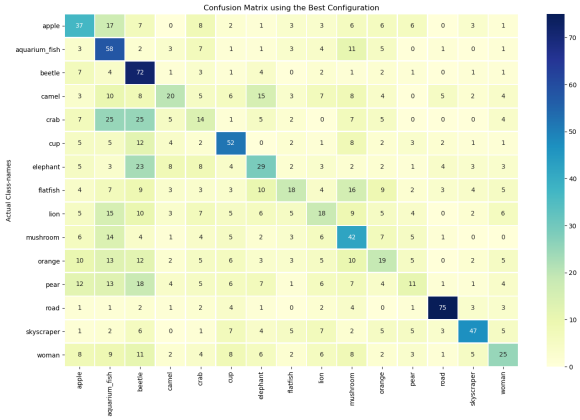
3.2 Accuracy values and how to evaluate them

The k-nn algorithm, which we use in the last stage of our operations, gives what kind of object the picture we give as input will be with the highest probability. Accuracy value is obtained by detecting what kind of objects the images in the validation data-set are, thanks to the dictionary structure we obtained by using the train data-set, and by calculating the consistency ratio of these determinations with the actual type of the objects.

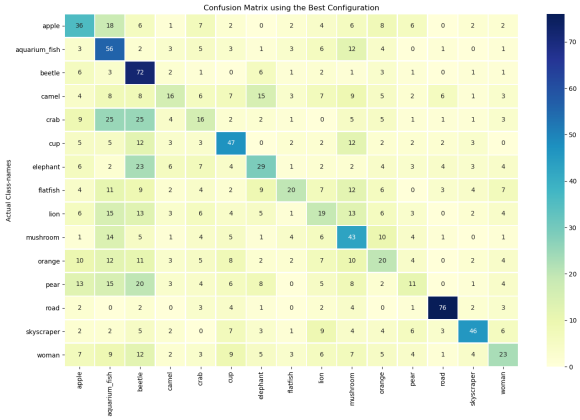
3.3 The confusion matrices for classification results of these combinations



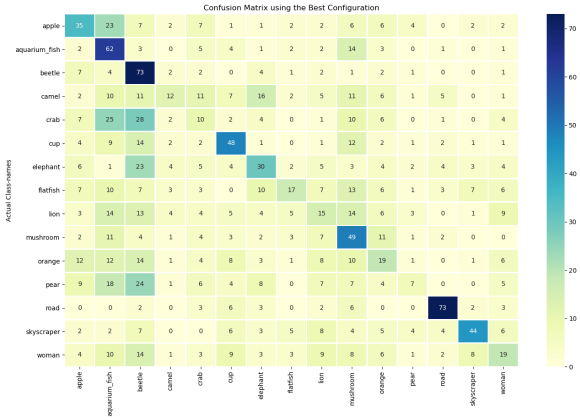
(a) Confusion Matrix for 16-NN



(b) Confusion Matrix for 24-NN



(c) Confusion Matrix for 32-NN



(d) Confusion Matrix for 64-NN

Figure 1: Confusion Matrices for k $\in [16, 24, 32, 64] Nearest Neighbors$.

4 Additional Comments and References

References:

[1] Working Example YouTube Tutorial
[2] Dense-SIFT Implementation GitHub
[3] Dense-SIFT Visual Explanation
[4] OpenCV :: Create SIFT Reference Page
[5] OpenCV :: SIFT Keypoint Reference Page