HACETTEPE UNIVERSITY

Department of Computer Engineering
SPRING 2019

BBM 418 Computer Vision Laboratory Problem Set-1

Authors:	Dr. Nazlı Ikizler Cinbis
	TA Ozge Yalcinkaya
Name:	Ege Berke Balseven
Number:	21590776
Subject:	Image Retrieval Basics
Due Date:	13.04.2019

Contents

1 Introduction

In this experiment, i learned basic image representation and retrieval methods. First, I experimented with simple image representations and used the k-NN classification method to obtain categories of the query images. I compared the results in terms of classification accuracy. Then in the second part I used a more advanced method to represent the images and repeated the classification part. In the end, I compared all the results and commented on them. I used a subset of the Caltech 256 record. Image Retrieval is generally the introduction of pictures and text-based searches that are closest to what is searched for. For example; Google Image search and Google Images are examples of this. What i do here is to train the machine according to certain classes and to be able to give the closest results when given the relevant picture.

2 Implementation Details

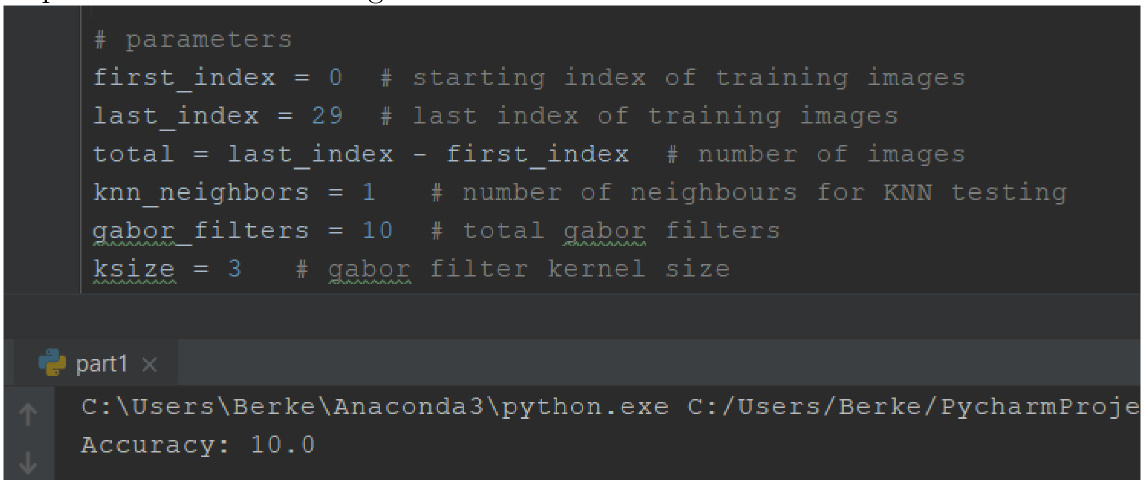
I've had too many problems. Some of these are: cv2.xfeatures2d.SIFT_create(), it is about opencv version. I decreased the version. kmeans.fit(descriptor_list). query data dimension must match training data dimension.Invalid index to scalar variable. object of type 'numpy.float64' has no len(). kNN implementation was also difficult. I've done all the methods in a single py file for each part so it can be a little complicated. I did not have enough time to classify and make it more organized. But the only thing that needs to be done to run is to set the query and train paths.

3 Experimental Results

3.1 PART 1: Simple Image Representations

Gabor Filter:
Gabor Filter is used to detect the edges that extend in certain directions on the image. A Gabor Filter Bank is created for this. Using more Gabor filters can determine more edge types. In the homework, i created the features by passing through the image i made through the Gabor filters. Then i took average and made a distance calculation with the query image and i found the nearest objects for each images. There are two features i can change in gabor filter. Lambda and theta. Theta for angle and lambda for frequency. I only change theta in below results, but end i also

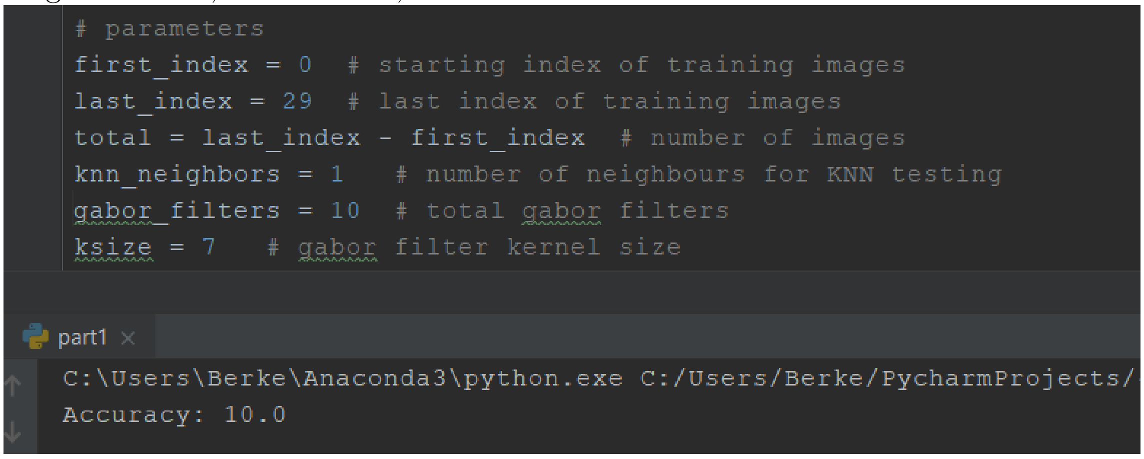
try lambda with different thetas for see changes.
Experimental results with gabor filter bank:



```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
knn_neighbors = 1   # number of neighbours for KNN testing
gabor_filters = 10  # total gabor filters
ksize = 3   # gabor filter kernel size
```

```
part1
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmProje
Accuracy: 10.0
```

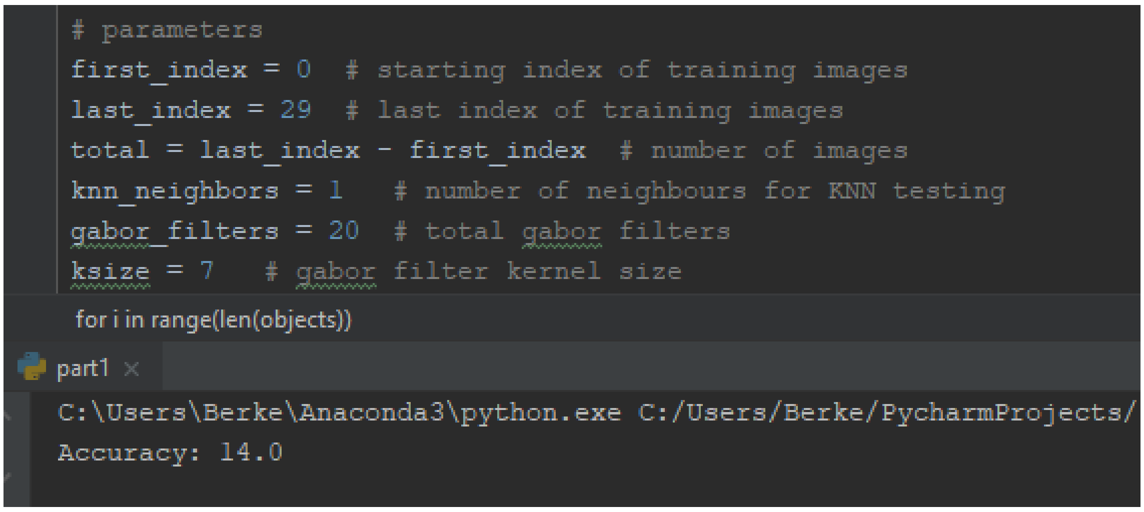
10 gabor filters, 3 kernel size, result bad



```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
knn_neighbors = 1   # number of neighbours for KNN testing
gabor_filters = 10  # total gabor filters
ksize = 7   # gabor filter kernel size
```

```
part1
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmProjects/
Accuracy: 10.0
```

10 gabor filters, 7 kernel size, result bad at 10 filters



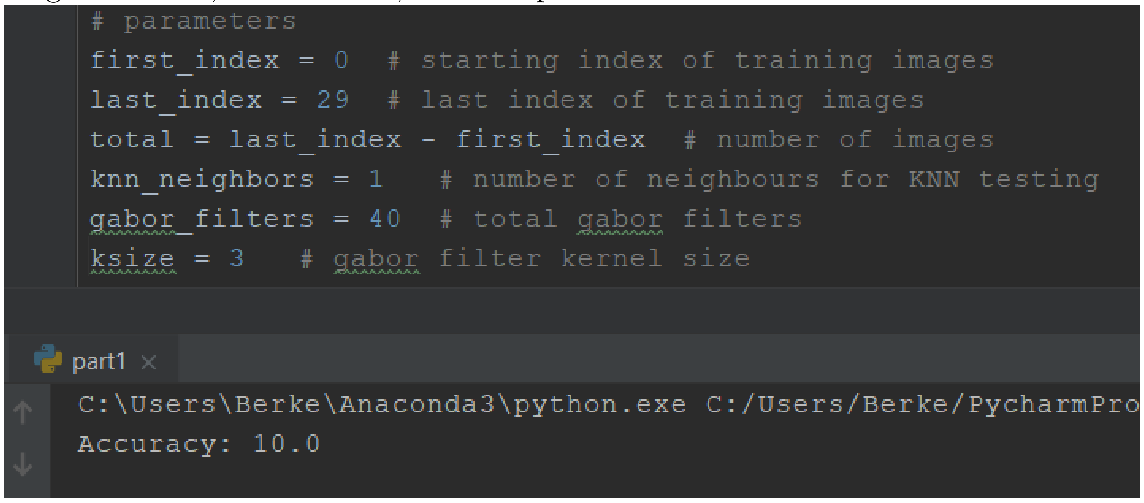
```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
knn_neighbors = 1   # number of neighbours for KNN testing
gabor_filters = 20  # total gabor filters
ksize = 7   # gabor filter kernel size
```

for i in range(len(objects))

part1

```
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmProjects/
Accuracy: 14.0
```

20 gabor filters, 7 kernel size, result improved

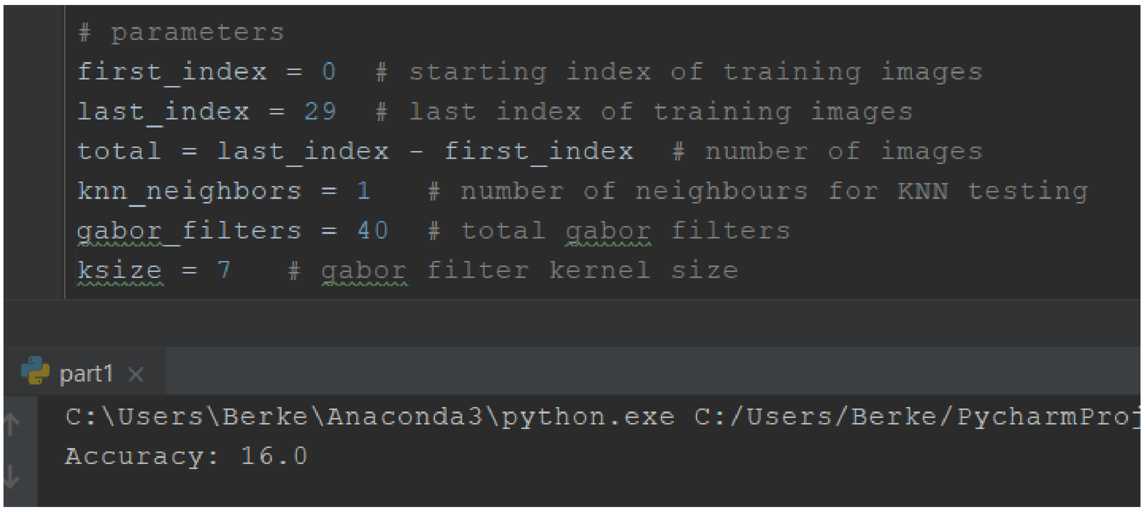


```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
knn_neighbors = 1   # number of neighbours for KNN testing
gabor_filters = 40  # total gabor filters
ksize = 3   # gabor filter kernel size
```

part1

```
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmPro
Accuracy: 10.0
```

40 gabor filters, 3 kernel size, kernel size too small, bad result

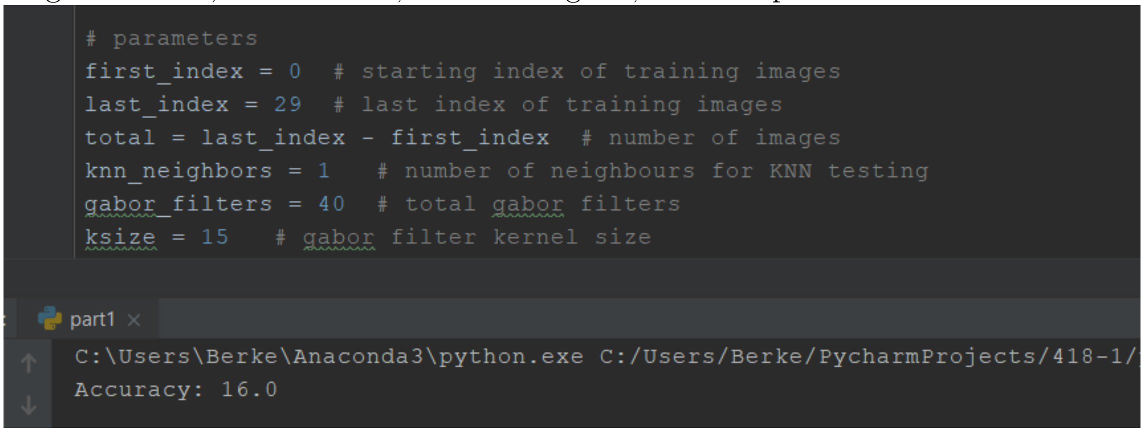


```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
knn_neighbors = 1   # number of neighbours for KNN testing
gabor_filters = 40  # total gabor filters
ksize = 7   # gabor filter kernel size
```

part1

```
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmPro
Accuracy: 16.0
```

40 gabor filters, 7 kernel size, kernel size good, result improved

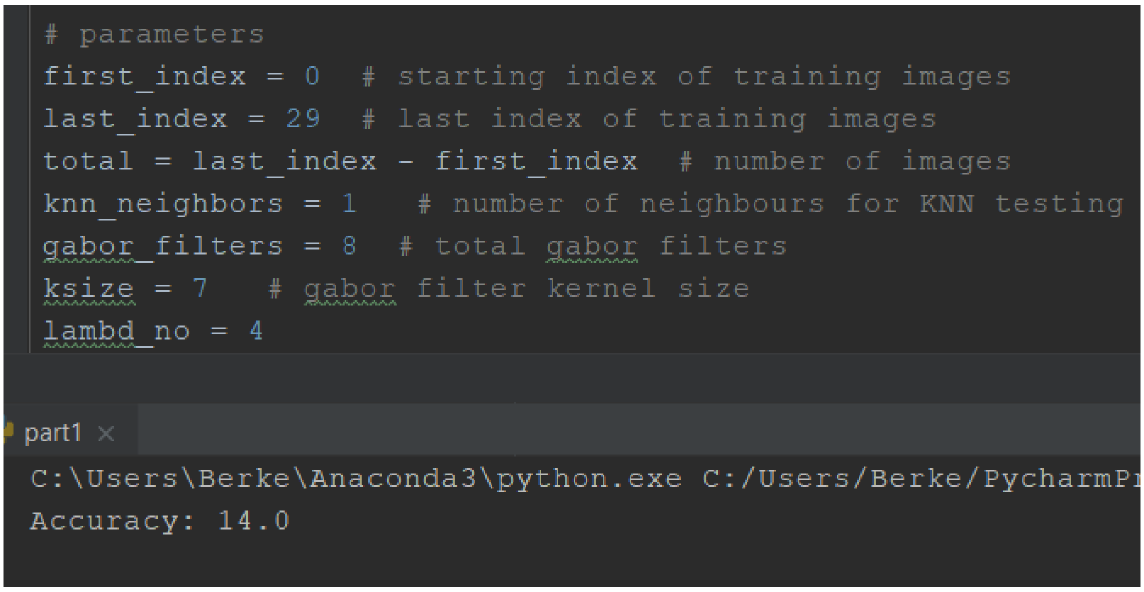


```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
knn_neighbors = 1   # number of neighbours for KNN testing
gabor_filters = 40  # total gabor filters
ksize = 15   # gabor filter kernel size
```

part1

```
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmProjects/418-1/
Accuracy: 16.0
```

40 gabor filters, 15 kernel size, kernel size much but does not affect after a certain level.



```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
knn_neighbors = 1   # number of neighbours for KNN testing
gabor_filters = 8  # total gabor filters
ksize = 7   # gabor filter kernel size
lambd_no = 4
```

part1

```
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmPr
Accuracy: 14.0
```

In addition i also change lambda value with theta value. There are also 40 different gabor filters but these are 5 different scales and of 8 different angles instead of 40 angles. So we have less different angles. Scales are not effects too much, angles more important to detect more edges or features. The result also shows this, 14 less then 16.

Average SIFT

SIFT:

Scale Invariant Feature Transform (SIFT) is an image descriptor for image-based matching and recognition developed by David Lowe. SIFT isn't just scale invariant. However we can get good results in this features: rotation, illumination and view-point.

SIFT is a comprehensive algorithm. Entire algorithm in some parts:

Constructing a scale space

LoG Approximation

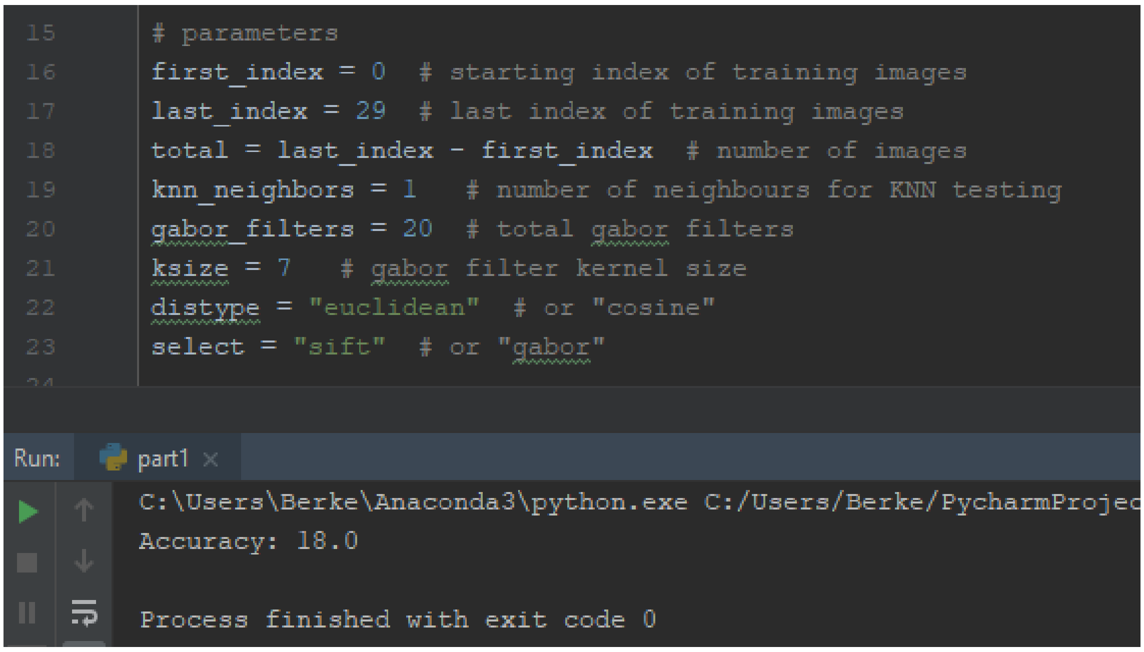
Finding keypoints

Get rid of bad key points

Assigning an orientation to the keypoints

Generate SIFT features

The SIFT descriptor has been proven to be very useful in practice for image matching and object recognition under real world conditions.



```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
knn_neighbors = 1   # number of neighbours for KNN testing
gabor_filters = 20  # total gabor filters
ksize = 7   # gabor filter kernel size
distype = "euclidean"  # or "cosine"
select = "sift"  # or "gabor"
```

```
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmProjec
Accuracy: 18.0

Process finished with exit code 0
```

Result is not bad and better then gabor as expected.

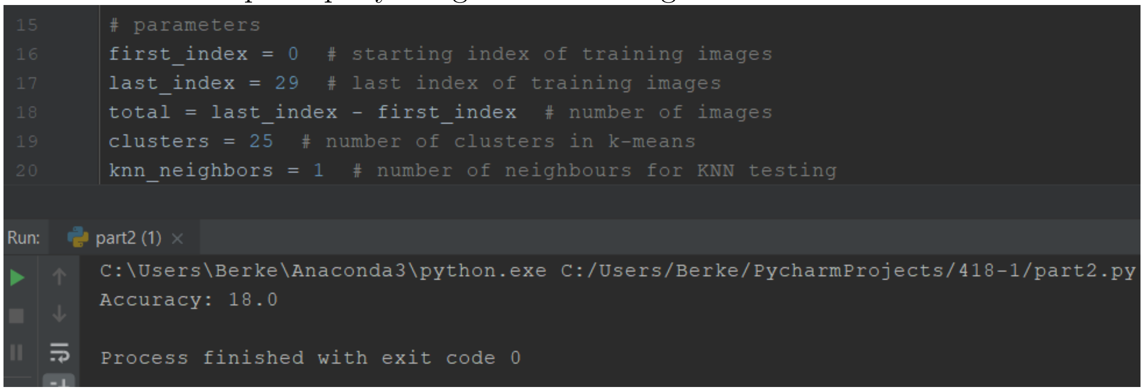
3.2 PART 2: Bag of Visual Words

The main idea of bag of visual words (BOVW) is to represent an image as a set of features. Features includes keypoints and descriptors. Keypoints are the interest points in an image, so no matter the image is rotated, shrink, or expand, its keypoints will always be the same. And descriptor is the description of the keypoint. We use the keypoints and descriptors to create vocabularies and represent each image as a frequency histogram of features that are in the image. From the frequency histogram, after we can find another similar images or predict the category of the image.

Some words needed to know
Codevector = codeword = visual word
Codebook = codeword = visual dictionary
Codebook = Set of visual words. Which means unordered set of words bag.
Bag of features = bag of visual words = bag of words
Quantization = indexing

Bag of word steps:
Extract features: with SIFT
Learn “visual vocabulary” : clustering (with k means)

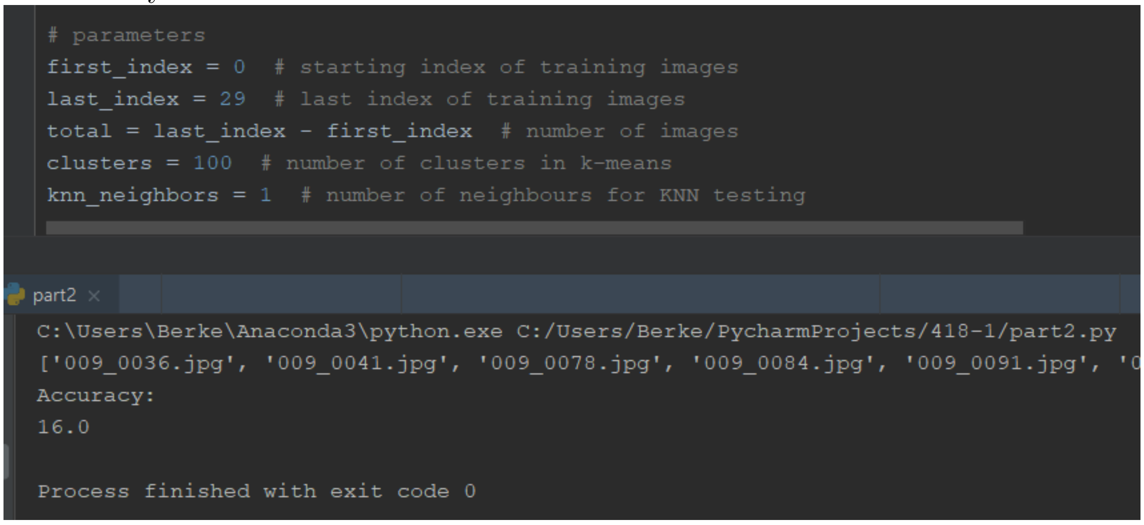
Quantize features using visual vocabulary : n-dimension to 1 dimension
Represent images by frequencies of "visual words" : Histogram
Then we can compute query image's nearest neighbors.



```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
clusters = 25  # number of clusters in k-means
knn_neighbors = 1  # number of neighbours for KNN testing
```

```
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmProjects/418-1/part2.py
Accuracy: 18.0

Process finished with exit code 0
```

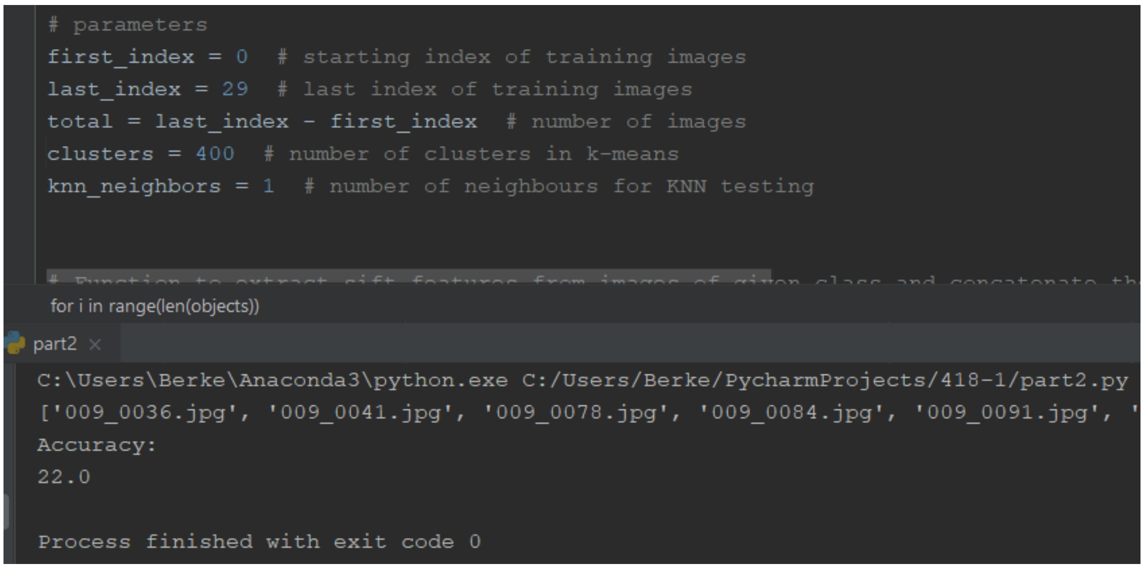
25 K Means clusters, 1 NN neighbours
18 accuracy



```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
clusters = 100  # number of clusters in k-means
knn_neighbors = 1  # number of neighbours for KNN testing
```

```
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmProjects/418-1/part2.py
['009_0036.jpg', '009_0041.jpg', '009_0078.jpg', '009_0084.jpg', '009_0091.jpg', '0
Accuracy:
16.0

Process finished with exit code 0
```

100 K Means clusters, 1 NN neighbours
16 accuracy



```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
clusters = 400  # number of clusters in k-means
knn_neighbors = 1  # number of neighbours for KNN testing
```

```
for i in range(len(objects))
```

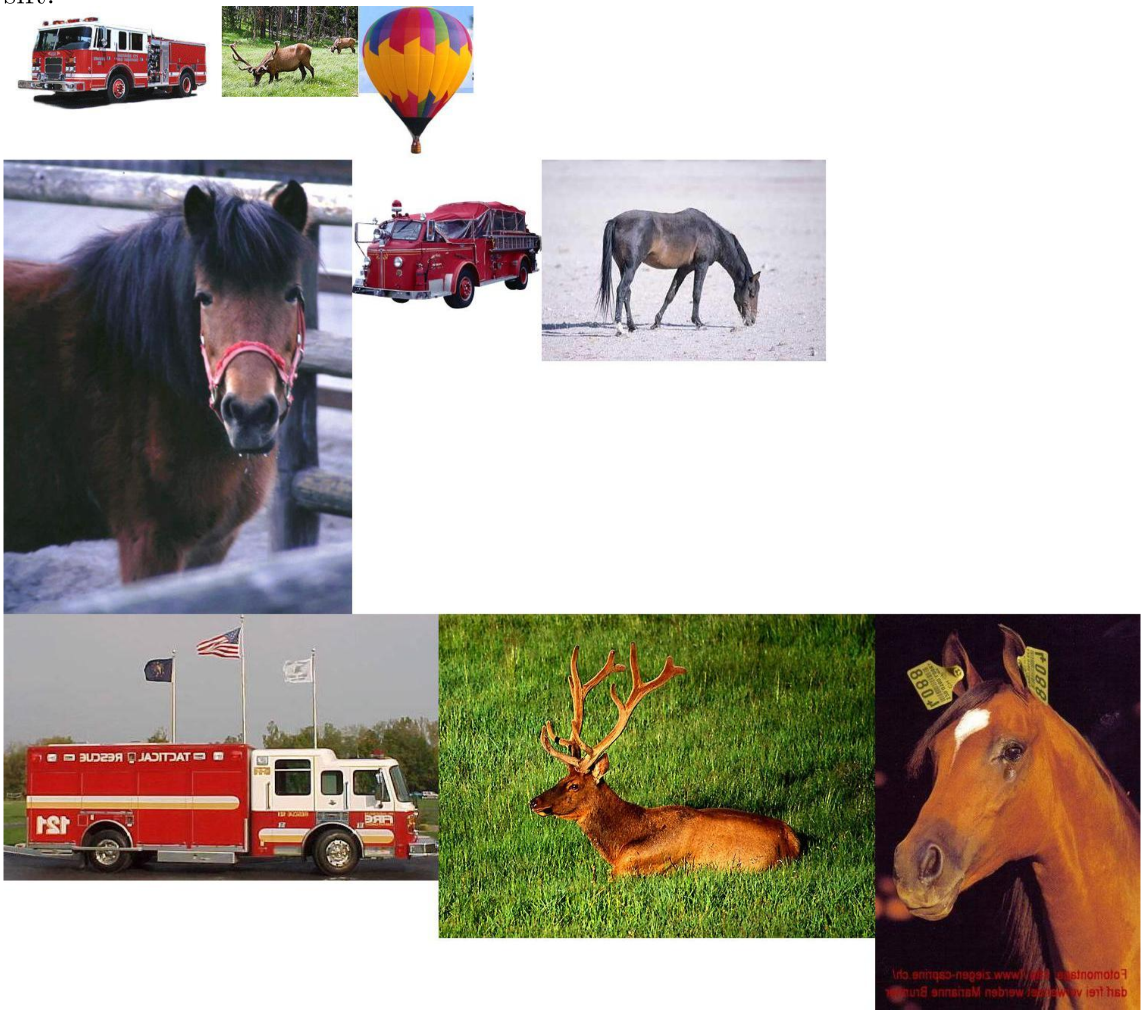
```
part2
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmProjects/418-1/part2.py
['009_0036.jpg', '009_0041.jpg', '009_0078.jpg', '009_0084.jpg', '009_0091.jpg', '
Accuracy:
22.0

Process finished with exit code 0
```

400 K Means clusters, 1 NN neighbours
22 accuracy
Accuracies increases with respect to cluster numbers.

Comperation for all cases:
I could not do BoW with spatial tiling.
Here are the true predicted results in all queries for each part.
Gabor:

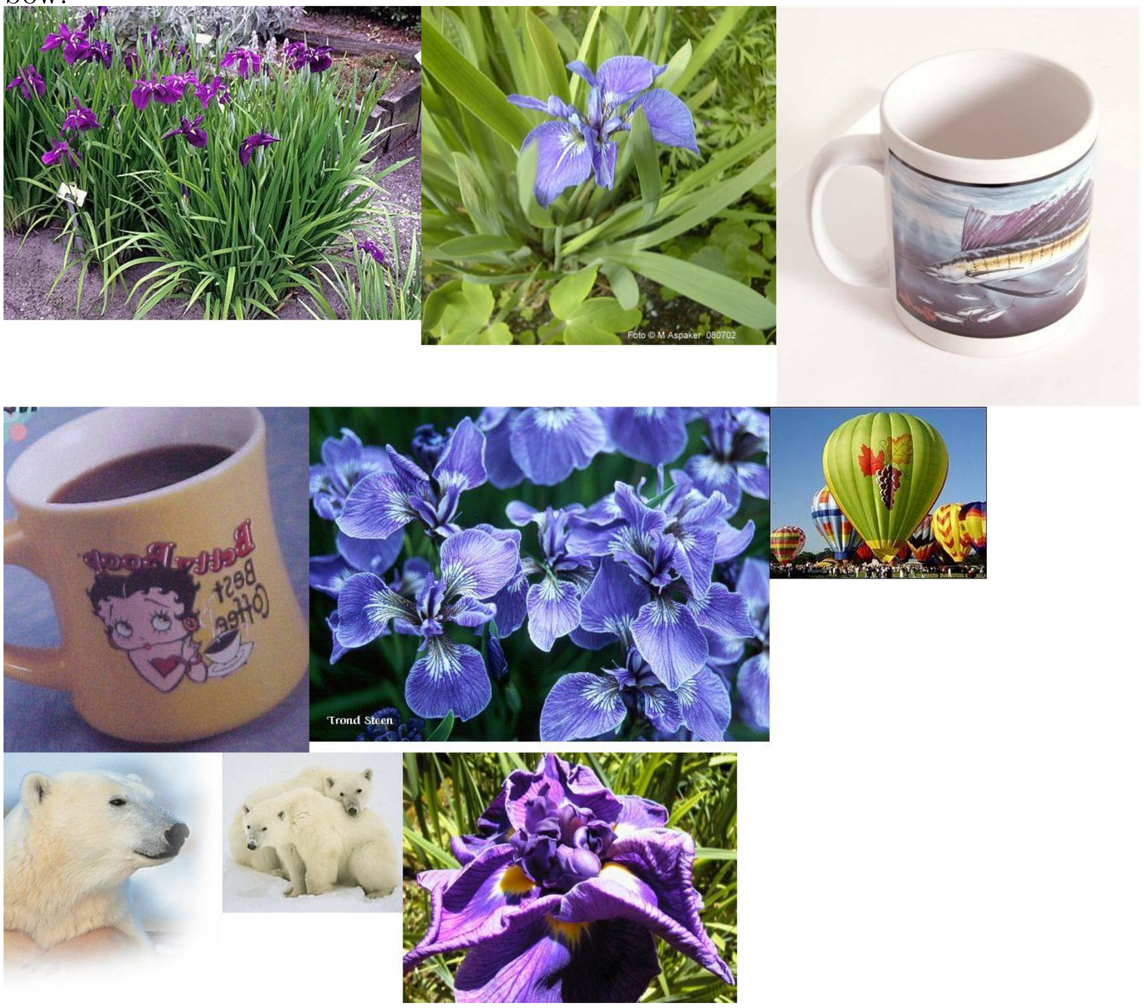
Here, we can say that it successfully predicted images by looking edges and similar backgrounds. Because they easier to distinguish.

sift:



In this case by looking instead of only edges and similar backgrounds it also took keypoints, colors and shapes into account.

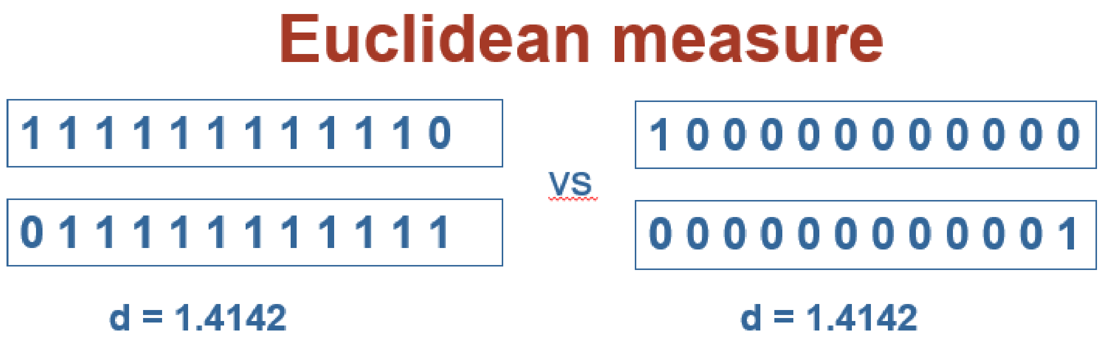
bow:



Here, keypoints are circular shape, colors, flowers, white bears. So by looking these images we can say that these keypoints are criterion for predicting.

3.3 Bonus:

Instead of just euclidean distance i added cosine similarity option. Euclidean Distance measures how far apart two points in a n-dimensional space are, i.e. it measures the length of a straight line from point A to point B. Furthermore cosine Similarity measures their similarity in orientation, i.e. the angle between two points A and B with vertex at zero.



Euclidean measure

1 1 1 1 1 1 1 1 1 1 1 0

0 1 1 1 1 1 1 1 1 1 1 1

d = 1.4142

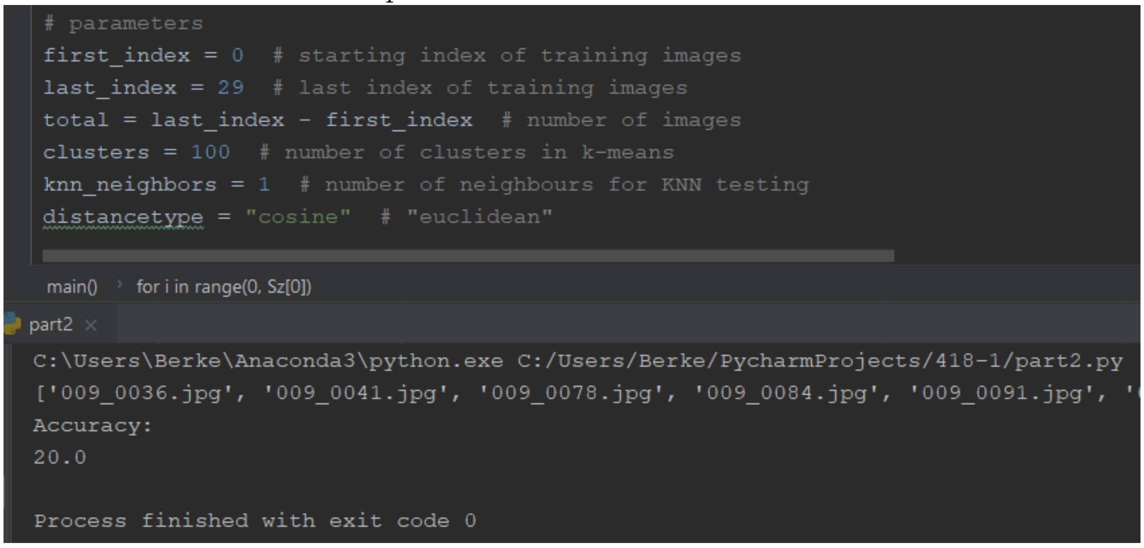
VS

1 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1

d = 1.4142

Cosine similarity is better then euclidean distance. Above image shows us one of the reason. In this case the difference between two points are equal. But on the other case real similarities of this points are too much.



```
# parameters
first_index = 0  # starting index of training images
last_index = 29  # last index of training images
total = last_index - first_index  # number of images
clusters = 100  # number of clusters in k-means
knn_neighbors = 1  # number of neighbours for KNN testing
distancetype = "cosine"  # "euclidean"
```

```
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/PycharmProjects/418-1/part2.py
['009_0036.jpg', '009_0041.jpg', '009_0078.jpg', '009_0084.jpg', '009_0091.jpg', '
Accuracy:
20.0

Process finished with exit code 0
```

100 cosine 20 accuracy is better then euclidean

4 Conclusion

Results are not satisfactory as well. Because of the K Means cluster size, number of iterations, distance type, KNN nearest neighbours and most importantly absence of the training images. The experiment was difficult, i failed too much nearly everywhere but i learned a lot from the experiment.