# Image Descriptor

14th Dec. 2021 Dongwon Kim

## Introduction

### Goal

The goal of the assignment is to find out image descriptors that represent similarities between images. Specifically

1. 1000 images and 1000 SIFT features are given.
2. SIFT features are given by binary files.
3. The number of features may be different among images.
4. The features are 128-dimension vectors.
5. Total number of images, dimension of image descriptor and descriptors of each image should be stored in binary file as output *'image_descriptor.des'* of the code.
6. The image descriptor should be the result of executing *'compute_descriptors.py'.*

### Related Theory [1]

For representing similarities between images, Bags of Words(BOW) vectors can be used. BOW has three parts: dictionary learning, encoding, and classification.

Visual dictionary is global visual word dictionary which means that features of all images should be used to learn visual words. Visual words can be computed with K-Mean clustering and the centroids of clusters become visual word.

Image features are mapped to proper visual words. When doing mapping, find the closest centroid and count the number of visual word occurrences. When counting the occurrences, weighted counting such as Term Frequency Inverse Document Frequency can be used.

In classification, KNN, SVM etc. can be used.

## Implementation

### Get Visual Dictionary

1. Read SIFT feature files and reshape the content of binary file to (-1, 128).
2. Vertically stack all descriptors. This step is to get global visual dictionary.
3. Using sikit learn KMeans(sklearn.cluster.KMeans [1]), get 128 cluster's centroids. (Set the number of clusters to 128 considering accuracy and computing time).

# Image Descriptor

14th Dec. 2021 Dongwon Kim

4. *'kmean_cv4.py'* is used to get the visual dictionary.
5. Google Colaboratory [2] is used to train KMeans. The file can be executed in Google Colaboratory. Otherwise, it will occur errors.(Some Google Colaboratory specific libraries and command are used such as drive.mount).
6. As result, get 128 visual words with 128 dimension vector for each word(128 x 128 np array).
7. Save the result as *'codeword_center.npy'*

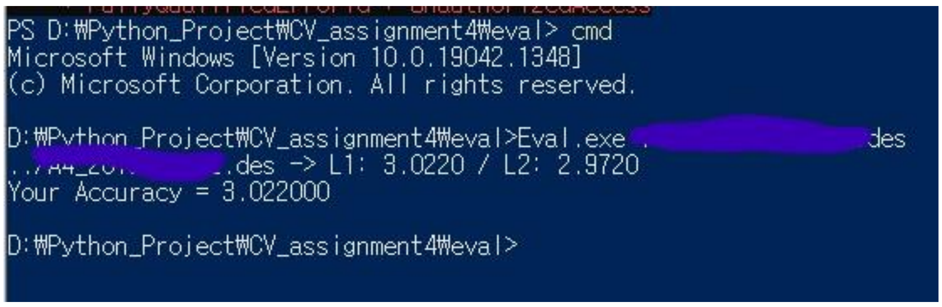
Encoding

1. Set N = 1000 and D = 128(defined when finding visual dictionary. The value of D is the same with dimension of the centroid vector).
2. For each SIFT descriptor file,
   a) Read file as unsigned char format.
   b) Reshape to (-1, 128) and store as np array.
   c) Get encoded descriptor using image_encode() function.
   d) Count the number of occurrences of each visual word and store the result in total_encoded.
      1) N x D np array
      2) Total_encoded[i][j] is the number of jth visual word in ith encoded image descriptor.
   e) To get weighted descriptor, use TF IDF by using tf_idf() function.
   f) Store the result as binary file *'image_descriptor.des'*.
3. Function image_encode(sift_descriptor, codeword)
   a) Sift_descriptor: n x 128 np array, sift descriptor of an image.
   b) Codeword: 128 x 128 np array, global visual dictionary.
   c) Calculate Euclidean distance between each descriptor and 128 visual words (stored in np array *'distance'*. The distance[j] contains distance of between ith feature of an image and jth visual word.).
   d) Get the index that has minimum distance using *np.argmin(distance)*
   e) Store found index to np array *encoded*.(encoded[i] is the closest visual word's index

in visual dictionary of ith feature of the image.)

4. Function tf_idf(encoded)
   a) Encoded: N x D total_encoded np array
   b) Count the number of occurrences(frequency) of each word in total descriptor.
   c) Compute weighted descriptor
      1) Weighted = total_encoded[i][j] * log10(D / frequency of jth word)

## Result



```
PS D:₩Python_Project₩CV_assignment4₩eval> cmd
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. All rights reserved.

D:₩Python_Project₩CV_assignment4₩eval>Eval.exe . [redacted] des
../A4_20 [redacted] .des -> L1: 3.0220 / L2: 2.9720
Your Accuracy = 3.022000

D:₩Python_Project₩CV_assignment4₩eval>
```

**Figure 1 Eval.exe result(accuracy)**

## Conclusion

The accuracy can be raised by increasing the number of visual words. However, increasing the number of visual words has limit because computation time for K-means clustering takes a lot of time when k is large. Plus, time to find the nearest centroids grow proportional to the number of visual words. Using approximating method such as tree data structure, hierarchical K-means and approximate K-means for finding the nearest centroid can reduce computational time. Although using approximating methods, it is hard to use BOW method for a large-scale data.

## References

[1] sklearn, "sklearn.cluster.KMeans," sklearn, [온라인]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html. [액세스: 14 12 2021].

[2] Google, *Colaboratory.*