**CS4186 Assignment 1 Report**

Name: AIMAKOV Aslan

SID: 55984847

**Introduction**

For the given problem of instance search, there are many possible approaches available. One of the most straightforward ones is based on features matching, i.e. the main goal is to get the keypoints and their descriptors for each of the given 5000 images to say if one image is more likely to be similar to a given query image. After we get them, we can measure the similarity between each pair of query image and image to rank the images accordingly. Other methods may consider splitting the image piece by piece or some brute force methods like pixel-wise comparison, etc.

The methods I have implemented are SIFT+FLANN+Homography and Color Histogram which use quite different approaches to solve the problem. As it will be shown later, the first method shows much better performance (around 34% on 10 example queries), as opposed to around 4% by Color Histogram. In addition, SIFT+FLANN+Homography took me around 40-50 minutes to generate the rankList on 10 example queries to measure its performance, whereas Color Histogram took me around 3 hours. The hardware is i7-8700k, 32GB and GTX 1080 Ti (the computers in CS Lab).

On the other hand, SIFT+FLANN+Homography showed quite uneven performance among different query images which is discussed in the Results part of the report.

**Technical requirements and my setup**

* Windows 10
* Atom Text Editor (my own preference)
* Python >=3+
* Pandas, Numpy, Matplotlib, Pillow, OpenCV, OpenCV-Contrib, Scikit-image, Scipy (all can be installed using pip as in my case)
* Git (absolutely not essential, I just used it to track all the changes and to push to GitHub private repository)

**Methods**

The first method I used is called SIFT (Scale-invariant feature transform) combined with FLANN based matcher. The basic idea is to retrieve both the keypoints and their descriptions from each image once and store it, and when rankList is generated, compare each image with the query image to rank the image according to the similarity score.

*cv.SIFT\_create()* is used to initialize the class used for getting the keypoints and descriptors (using *.detectAndCompute()* method). Then I initialize a *FlannBasedMatcher()* class to call .*knnMatch()* method for feature matching. To finally create a mask for the method, I used *findHomography()* method.

Since the majority of the obtained features are not significant for instance search, David G. Lowe suggested to use 0.7 coefficient to determine if the feature is significant or not. This coefficient is necessary since we need to compare their distances for each pair of features. For the sake of this assignment, I assumed 5 good matches is enough to claim that the image is likely to be the same as the example query image. If the image got less than 5 good matches, this means it’s definitely not among the top 20 results for each query image. The core code for this method is shown on Figure 1 below:



**Figure 1**

The second method I used is called Color Histogram (manually implemented in my case) which basically creates a histogram for red, green, and blue colors indicating the proportion of all the pixels which have the same color (scaled from 0-255) and compares the histograms pairwise to get the similarity score.

I create a color histogram based on basic linear algebra, as illustrated in *create\_rgb\_hist()* function in the source code and compare the histograms using *cv.compareHist()* method to get the distance between them. I chose *cv.HISTCMP\_CORREL* flag which stands for correlation-based distance (the higher the value, the more likely the images are similar). The method is implemented as below on Figure 2:

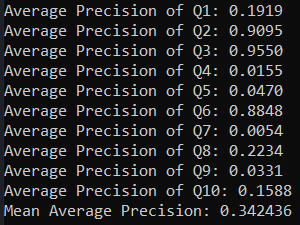


**Figure 2**

For both methods, I firstly store histograms/descriptors and keypoints for all 5000 images because I need to use each of them 10 times (20 when generating the rankList for 20 query images) to generate rankList for 10 example query images and to measure the performance. This trick saved me more than 30 minutes since it takes around 3-5 minutes to calculate all the descriptors and keypoints once (I have no idea how long it would have taken me in case of color histogram since it took me 3 hours to generate 5000 histograms).

**Results**

The rankList generated from 10 example queries shows the performance as demonstrated below on Figure 3:



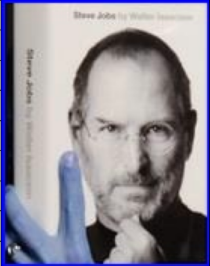
**Figure 3**

It can be seen that for some queries like Q2, Q3, and Q6 the precision is very good (around 88-95% each), whereas it poorly performed on some queries like Q4, Q5, Q7, and Q9 (around 0.5-5% each). On the other hand, MAP (Mean Average Precision) is around 34% which is quite good for the purpose of this assignment. The below examples illustrate the first five query examples using actual images (big image on the left is the example query image, and small 10 images are top 10 results for each query):

A picture containing text

Description automatically generated

**Figure 4 (Q1)**

A picture containing text

Description automatically generated

**Figure 5 (Q2, Top performer)**

****

**Figure 6 (Q3, Top performer)**

****

**Figure 7 (Q4, Poor performer)**

**A picture containing text

Description automatically generated**

**Figure 8 (Q5, Poor performer)**

After playing with Lowe coefficient (0.7) and minimum number of good matches, I didn’t get any change in MAP more than 1%, therefore, I decided to keep these parameters for feature description+matching method.