CSE 515 MULTIMEDIA AND WEB DATABASES

Project Phase 3 Report Document

Video Retrieval

Group 1

Rubinder Singh
Lei Guo
Mihir Thakkar
Srujan Kanteti
Yiqian Zhang
Yu-Hsuan Chuang

Date: 2-Dec-2016

Abstract - The motivation for this project is to better understand video object search and retrieval using PageRank, personalized PageRank algorithm and further understand Locality Sensitive Hashing (LSH) for efficient nearest neighbour search. There are three specific parts in this project that were performed over six tasks. In the first part of the project we used PCA to reduce the dimensionality of the SIFT vectors and then populate a graph using the reduced dimensions. In the second part of this project we implement classic and personalized PageRank and ASCOS++ algorithms to find the most significant and most relevant frames. For the last part of the project we implement layered-LSH and Nearest Neighbor search. For performing these tasks, Python libraries were used such as, networkx, scikit-learn, numpy, nearpy and scipy.

Keywords - Euclidean Distance, Principle Component Analysis, Dimensionality Reduction, PageRank, ASCOS++, Locality Sensitive Hashing, Graph, Object Search, Nearest Neighbor search.

1. Introduction

1.1 Terminology

1.1.1 SIFT

This feature extraction method is finding distinctive features of an image using its scalar invariant keypoints [8]. The scalar invariant features or SIFT feature or keypoint is a circular region of an image. The SIFT vectors can be computed using SIFT detectors and SIFT descriptors [9] [8]. SIFT detector is 4xK matrix containing 4 values in each column for all K SIFT vectors and SIFT descriptor is 128xK matrix containing 128 descriptor values in each column for all K SIFT vectors [7]. SIFT descriptor contains a 3D histogram of 128 bins [9].

1.1.2 Euclidean Distance

A distance function that is good for comparing vectors. This was used in tasks 2 to calculate similarity.

***Formula for the Euclidean Distance*:**

The distance between two vectors x = (a, b) and y = (c, d) is given by

$$Euclidean\_Distance((x, y)) = \sqrt{(a - c)^2 + (b - d)^2}$$

1.1.3 PageRank

PageRank is a link analysis algorithm and it is a way to measure the significance of website pages. It uses a series of weighting numbers to each element in a hyperlinked set of documentation (such as world wide web)[6]. It assumes that the distribution is evenly divided among all documents. When doing the computation, it uses "iteration" to make sure the approximate PageRank values of each node are get close to the theoretical true value.[6]

1.1.4 ASCOS++

ASCOS++ is an improvement of ASCOS. It not only consider the network but also the edge weights. Experimental results show that ASCOS++ has a better score than SimRank and several famous similarity measures because it can identify the hierarchical relationship between nodes.

1.1.5 Locality Sensitive Hashing (LSH)

Locality sensitive hashing is a hashing technique which leverages binary random projection and is done basically to reduce dimensions of the input data and also to prune the search space for nearest neighbour search. The vectors are classified in *L* layers and *2^k* buckets based on random hash functions. If any two vectors are in the same bucket, they are considered a near neighbours.

1.2 Problem Specification

1.2.1 Task 1

Objective: The objective of this task is to apply PCA and reduce the dimensionality of the features to a specified amount (d). The reduction is performed on the a SIFT database (file_name.sift) and then outputted onto a new file (file_name_d.spc).

Inputs:

- **d (dimensionality) –** An integer that represents the preferred dimensionality of the feature space.
- **file_name.sift (database) –** A database that contain all the SIFT points in the given videos.

Outputs:

- **file_name_d.spc (database) –** A new database that is created by the program. This database contains the reduced feature set.
- **file_name_d.score (scores)** – A file that contains the scores of the PCA matrix.

1.2.2 Task 2

Objective: The objective of this task is to implement a similarity graph using the reduced feature space (file_name_d.spc). Each node is in the graph is represented by a key point and each node will have a specified amount (k) of neighbors.

Inputs:

- **k (neighbors) –** An integer that represents the amount of neighbors for each node in the graph.
- **file_name_d.spc (database) –** A database that contain all the SIFT points in task one.

Outputs:

- **file_name_d_k.gspc (graph) –** A graph that is created by the program.

1.2.3 Task 3 (A)

Objective: The objective of this task is to identifies the most significant m frames in the reduced feature similarity graph (filename_d_k.gspc) by using PageRank.

Inputs:

- **file_name_d_k.gspc (graph) –** A graph database that contain the similarity between frames.
- **m –** An integer that represents the amount of most significant frames we need to find out.

Outputs:

- **Output_rank<RANK_NUM>_index_<VIDEO_NUM>_<FRAME_NUM>.png**
  'm' frames of above specified format are produced in Output/Frames directory.

1.2.4 Task 3 (B)

Objective: The objective of this task is to identifies the most significant m frames in the reduced feature similarity graph (filename_d_k.gspc) by using ASCOS++.

Inputs:

- **file_name_d_k.gspc (graph) –** A graph database that contain the similarity between frames.
- **m –** An integer that represents the amount of most significant frames we need to find out.

Outputs:

- **Output_rank<RANK_NUM>_index_<VIDEO_NUM>_<FRAME_NUM>.png**
  'm' frames of above specified format are produced in Output/Frames directory.

1.2.5 Task 4 (A)

Objective: The objective of this task is to identifies the most relevant m frames relative to the three input frames by using PageRank.

Inputs:

- **file_name_d_k.gspc (graph) –** A graph database that contain the similarity between frames.
- **m –** An integer that represents the amount of most significant frames we need to find out.
- **(vi, fi) –** Three pairs of input video/frame ids

Outputs:

- **Output_rank<RANK_NUM>_index_<VIDEO_NUM>_<FRAME_NUM>.png**
  'm' frames of above specified format are produced in Output/Frames directory.

1.2.6 Task 4 (B)

Objective: The objective of this task is to identifies the most relevant m frames relative to the three input frames by using ASCOS++.

Inputs:

- **file_name_d_k.gspc (graph) –** A graph database that contain the similarity between frames.
- **m –** An integer that represents the amount of most significant frames we need to find out.
- **(vi, fi) –** Three pairs of input video/frame ids

Outputs:

- **Output_rank<RANK_NUM>_index_<VIDEO_NUM>_<FRAME_NUM>.png**
  'm' frames of above specified format are produced in Output/Frames directory.

1.2.7 Task 5

Objective: The objective of this task is to finish a Locality Sensitive Hashing (LSH) tool by mapping keypoints into a hash bucket for each layer.

Inputs:

- **file_name_d.spc –** a file contains all the keypoint.
- **L –** the number of layers
- The number **K** for (2^k) of buckets per layer

Outputs:

- **file_name_d.lsh –** {layer num, bucket num, ⟨index,i,j,l, x, y⟩}

1.2.8 Task 6

Objective: The objective of this task is to finish a similarity-based video object search tool.

Inputs:

- **file_name_d.lsh –** LSH index file
- **N –** An integer that represents the amount of most significant frames we need to find out.
- **⟨i j x1 y1 x2 y2⟩–** i: video numbers j: frame numbers ⟨x1,y1⟩;⟨x2,y2⟩ is a rectangle which has the object.

Outputs:

- **N frames –** not in the same video of the query, containing the most similar frames with objects are visualized in Output/n_frames directory.
- The amount of unique SIFT vectors
- The amount of all SIFT vectors
- The amount of data bytes to implement the query

1.3 Assumptions

1.3.1 Parameters for sample input and output

- d =10 and 60
- k = 2 and 10
- K = 5 and 10
- L = 3 and 6

1.3.2 Folder structure

- The file structure is a dependency for execution of all programs.
- Input: ./Input/
- Output: ./Output/
- Video files: ./Input/Videos/

1.3.3 Original index used in LSH output for fast lookup in video object search

- In the output of LSH, we have used a column for original index which is the index of vector in original database. This is used for keypoint query-object comparison after pruning the search space using locality sensitive hashing.

1.3.4 Corresponding .spc file present in /Input/ for LSH given input parameters

If task 5 (LSH) is executed using some parameters other than given sample, we assume that we have corresponding .spc file is present in the Input directory. If not, we can copy the task 1 output file from /Output/ directory and rename it as required for LSH.

2. Implementation

2.1 Task 1 - Principle Component Analysis

The Principal Component Analysis can be done by leveraging the PCA class provided by scikit-learn in python. The function *pca.transform()* does principal component analysis on the original object-feature matrix and returns the new object-principal components matrix after reducing dimensions of low variance. [12]

The scores of the principal components can also be computed by leveraging the PCA class provided by scikit-learn. The function "pca.transform()" does principal component analysis on the original object-feature matrix and then the scores can be acquired by getting the attribute *pca.components_*.

The main steps of the program are:

1. Read the input database into the matrix M1.Extract the data of vectors excludes their original indices and put them in a new matrix M2.
2. Call the pca function with matrix M2 as the parameter and record the returned new object-principal components matrix.
3. Concatenate the rows of original indices with the corresponding rows of new object-principal components matrix respectively and form a new matrix M3.
4. Output M3 to the new database.
5. Extract the Scores (*pca.components_*) and save them into a new file.

2.2 Task 2 - Graph generation

The graph is created by leveraging the transformed database. The graph algorithm takes a total of $O(n^2)$ steps where n is number of frames in the whole database. Below is the pseudocode for the algorithm to populate the graph:

```
VIDEOS = Number of videos in the database

FOR X = 0 to VIDEOS
        FRAMES = Number of Frames in the current video X
        FOR X2 = 0 to FRAMES
                x2-similarity-values = list()

                FOR Y = 0 to VIDEOS
                        IF X != Y
                                FRAMES = Number of Frames in the current video Y
                                FOR Y2 = 0 to FRAMES
                                        similarity=ComputeSimilarity(KEYPOINTS[X], KEYPOINTS[Y])
                                                ADD similarity to the list x2-similarity-values
                                 END
                        END
                END
                PRINT the k most x2-similarity-values.
        END
END
```

The similarity between two frames is calculated using Euclidean distance. Euclidean distance gave us the best results for phase two of the project. All the query keys-points in one frame are compared with the object key-points in another frame. The highest possible similarity is extracted from the keys points.

Formula for the Similarity using Euclidean Distance:

The distance between two points (x, y) and (a, b) is given by

$$Euclidean\_Distance((x, y), (a, b)) = 1 - \sqrt{(x - a)^2 + (y - b)^2}$$

2.3 Task 3 - Most Significant Frame Selection

2.3.1 Implementation for Task 3(a) - PageRank

The PageRank can be implemented by leveraging the pagerank function provided by networkx in python. It computes a ranking of the nodes in the graph G based on the structure of the incoming links and returns the PageRank order of the nodes in the graph.[4]

Formula for the Similarity Score using PageRank:

$$\vec{P} = T * \vec{P}$$

where T is a $node\_size \times node\_size$ transition matrix noted the possibility from node i to node j

The main steps of the program are:

1. Read the database file, extract the data of vectors and transform it into an intermediary file. Then read the intermediary file into the matrix M_1.

2. Build a directed graph G whose nodes represent the video frames and edges between nodes represent the similarity between the two video frames.
3. Call the pagerank function with graph G as the input parameter and record the returned dictionary D_1 which contains nodes and their PageRank scores.
4. Sort the dictionary D_1 based on the pagerank score and form a new dictionary D_2.
5. Output the m most significant frames according to their Pagerank scores in D_2.

2.3.2 Implementation for Task 3(b) - ASCOS++

The ASCOS++ can be implemented by leveraging the ascos function provided by graphsim in python. This function computes an asymmetric similarity for weighted networks of the graph G, and it can only measure the undirected graph. However, our graph is a directed graph. So, we wrote a new function that can measure directed graph based on the original ASCOS++ function. To measure directed graph, we need to consider the out-neighbor edge weight instead of undirected edge weight. The function will return the new similarity scores between nodes in the graph.[5]

Formula for the Similarity Score using ASCOS++ [5]:

$$s_{ij} := \begin{cases} c \cdot \sum_{\forall k \in N(i)} \frac{w_{ik}}{w_{i*}} (1 - \exp(-w_{ik})) s_{kj}, & \text{if } i \neq j \\ 1 & \text{if } i = j, \end{cases} \quad (4)$$

where w_{ik} is the weight of edge *e (i, k)*, and $w_{i*} = \Sigma_{\forall k \in N(i)} \; w_{ik}$.

The following pseudocode is for computing ascos++ similarity score **[5]**,

Input: A: an adjacency matrix of size n by n; c: the discounted parameter
Output: S = [s_{ij}]: ASCOS++ similarity score matrix
S ← initial guessing matrix of size n by n;
WHILE S not converge **DO**
    FOR i ← 1 to n **DO**
        FOR j ← 1 to n **DO**
            Update s_{ij} by Equation (4);
        END
    END
END ;

Use ASCOS++ similarity score to calculate pagerank score [5]:

PageRank(j) 1/N $\Sigma_{\forall i}$ *(sij)*

where s_{ij} is the ASCOS similarity score from i to j .

The main steps of the program are:

1. Read the database file, extract the data of vectors and transform it into an intermediary file. Then read the intermediary file into the matrix M_1.

2. Build a directed graph G whose nodes represent the video frames and edge between nodes represent the similarity between the two video frames.
3. Call our ascos function with graph G as the input parameter and record the returned matrix M_2 of new similarity score.
4. Compute rank score using the equation: *PageRank(j)* $1/N \Sigma_{\forall i}$ (sij), and record it in a new dictionary D_1.
   The ASCOS score from *i* to *j* is the same as measuring how soon a random surfer starting at *i* arrives at *j*, whereas the PageRank score of *j* is proportional to how soon a surfer starting at a random node *i* is expected to arrive at *j* [5].
5. Sort the dictionary D_1 based on the ascos rank score and form a new dictionary D_2.
6. Output the *m* most significant frames according to their ascos scores in D_2.

2.4 Task 4 - Most Relevant Frame Selection

2.4.1 Personalized PageRank

The personalized PageRank can also be implemented by leveraging the pagerank function provided by networkx in python. [11] It computes a ranking of the nodes in the graph G based on the structure of the incoming links and a personalized matrix which emphasize the three relevant frames and returns the PageRank order of the nodes in the graph.

The main steps of the program are:

1. Read the database file, extract the data of vectors and transform it into an intermediary file. Then read the intermediary file into the matrix M_1.
2. Build a directed graph G whose nodes represent the video frames and edges between nodes represent the similarity between the two video frames.
3. Create a dictionary and set personalized degree to each node.
4. Call the pagerank function with graph G and personalization dictionary from step 3 as the input parameters and record the returned dictionary D_1 which contains nodes and their PageRank scores.
5. Sort the dictionary D_1 based on the pagerank score and form a new dictionary D_2.
6. Output the *m* most relevant frames according to their Pagerank scores in D_2.

Formula for the Similarity Score using personalized PageRank:

$$\vec{p} = \alpha T.\vec{p} + (1-\alpha)\begin{bmatrix} 0 \\ 0 \\ 1/3 \\ 0 \\ 1/3 \\ 0 \\ 0 \\ 1/3 \\ \vdots \\ 0 \end{bmatrix}$$

T: the original adjacency matrix

p: the PageRank value of each node

α: the damping factor

Seed vector: the dictionary of the personalized degree of each node (according to the requirement, the three reference nodes get $\frac{1}{3}$ respectively, and the rest of them get 0)

2.4.2 ASCOS++

In the personalized ASCOS++ method, if a node is considered more important, the similarity between it and its predecessors get an extra score. This is reasonable because during the random walk, if a node is more important, it will have a larger probability to be walked at [13], which means the similarity between it and its predecessors will increase. In the program, we can achieve this by adding a personalized matrix to the similarity matrix in each iteration.

Formula for the Similarity Score using personalized ASCOS++:

$$S' = S + M \qquad (5)$$

Where S is the original similarity matrix that we obtained at each iteration with ASCOS++, M is the personalized matrix and S' is the new similarity matrix that will be used in the next iteration. An example of M is as following.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/4 & 0 & 0 \end{bmatrix}$$

In the matrix, if node j is one of the referential node, a total power of 1 will be shared by all the elements which have a non-zero similarity value at jth column.

The following pseudocode is for computing personalized ASCOS++ similarity score [5],

Input: A: an adjacency matrix of size n by n; c: the discounted parameter
Output: S = [s_{ij}]: personalized ASCOS++ similarity score matrix
S ← initial guessing matrix of size n by n;
WHILE S not converge **DO**
    FOR i ← 1 to n **DO**
        FOR j ← 1 to n **DO**
            Update s_{ij} by Equation (4);
        END
    END
    Add personalized matrix to S_{ij} by Equation (5);
END

The way to calculate the infer Pagerank score with ASCOS++ similarity score is the same as the original ASCOS++ method [5].

The main steps of the program are also very similar to the original ASCOS++ method, except that we add a personalized matrix to the similarity matrix during each iteration [5].

2.5 Task 5 - Locality Sensitive Hashing

The nearest neighbour search becomes difficult and inefficient in high dimensional space. So locality sensitive hashing is used which is efficient data structure for lookup purposes. Any hashed data makes lookup easier and efficient in terms of time and space. LSH basically reduces the dimensionality of input data by mapping it to objects which are called buckets. Each bucket contains several input objects which are "most" probably similar to each other. LSH leverages the concept of probability of similarity between input objects (or SIFT vectors according to this particular project).

2.5.1 Mathematical definition of Locality Sensitive Hashing

An LSH family of hash functions is used to map input objects to buckets is defined as follows:

$d(P,Q) < T$, then $h(P)$ and $h(Q)$ collides with at least probability P_1
$d(P,Q) > cT$, then $h(P)$ and $h(Q)$ collides with at most probability P_2

where;

- *P, Q : some SIFT vectors*
- *d(P,Q) : distance between P and Q based on some distance measure, say Euclidean or Cosine*
- *h(x) : h is a hash function*

The hash family is good if $P_1 > P_2$.

2.5.2 Binary Random Projection

As nearest neighbour search is inefficient at high dimensional data [3], we have to reduce dimensionality. So we classify the dataset in several buckets based on binary random projection. This classification of buckets happen *L* times for each of the *L* layers. For each layer algorithm for binary random projection is as follows:

1. Generate k random vectors to divide the input vector space.
2. Based on cosine similarity function *sim( )*, append 0 or 1 value to bucket of a vector.
3. Repeat step 2 for k random vectors with all input vector objects.
4. This will generate one binary sequence (bucket number) for each input vector.
5. Repeat above algorithm for *L* layers.

2.5.3 Algorithm

Create list of k random vectors for bucket assignment.

1. For all vectors(input) as V:
   If vector lies in one side of the random vector, value is 0 or else 1.
   Append this 0 or 1 to the bucket number for vector V.
   Store video, frame, cell, layer and bucket number for this vector in the result set.
2. Output the generated result set in an output file.

2.6 Task 6 - Video Object Search

The main aim of this task is to retrieve *n* frames containing same (or perhaps similar) object from videos other than query video. The query object is defined by two points which are diagonally opposite points of a rectangular part in the query video frame. The algorithm used for object search is as follows:

1. Get all the SIFT keypoints from rectangular portion of query frame.
2. Retrieve all key points from the same bucket as the query key points in each of the locality sensitive hash layers.
3. From the pruned search space containing nearest neighbour keypoints, search for a set of keypoints which are most similar (or least distant, based on Euclidean distance) to the set of query set of keypoints.
4. Searching *n* such most similar sets (or objects), we can visualize the frame using frame number from LSH output file.

LSH is useful for applications like object recognition [13]. There can be millions of keypoints in the database object videos where we are searching for objects, making search space very large. So, using locality sensitive hashing, we have reduced number of keypoints which are similar to query keypoints by considering only the keypoints which are present in same bucket as query keypoints. After retrieving all similar keypoints, we can retrieve video number and frame number (which is represented by original index in output of LSH). Using Euclidean distance, we

compute most relevant frames and lookup original database using original_index and visualize it in form of an image in the output directory as stated earlier.

3. Sample Output

3.1 Task 1

- **Input:**

```
Enter the input file name(File should exist in Input directory): in_file.sift
Loading and Preprocessing database......
Enter the target dimensionality: 10
```

- **Output Reduced dimensions (in_file_10.spc):**

```
1,1,3,41.2504,317.7509,-2.4748,-0.6240,0.4299,0.1849,0.2140,-0.1227,0.0657,-0.0929,-0.0442,-0.0037
1,1,3,41.2504,317.7509,-3.7408,0.0803,-0.4216,0.1836,-0.1210,-0.1002,0.0599,-0.0954,-0.0473,0.0011
1,1,3,22.1905,315.4990,-0.5807,-0.7096,0.4093,-0.1611,-0.0938,-0.0850,0.2484,0.1419,-0.0356,-0.0608
1,1,3,36.9405,315.5559,1.9394,-2.2538,-0.4343,-0.1550,0.0418,-0.0963,0.1690,0.0710,0.0559,-0.0489
1,1,3,43.4531,322.0541,-2.0192,-0.0602,0.4243,-0.1511,-0.1032,-0.1113,0.1689,0.0983,0.0302,-0.0325
```

  - **Format:** <Video Number, Video Frame, Cell number, X, Y, d-reduced vectors>

- **Output Score (in_file_10.score):**

```
1,1,0.483612
1,89,0.011646
1,49,0.011629
1,47,0.010008
1,87,0.009704
1,57,0.009572
1,69,0.009046
1,15,0.008905
1,61,0.008554
1,81,0.008415
```

  - **Format:** <New Dimension, Original Dimension, Score>

3.2 Task 2

- **Input:**

```
Enter the input file name(File should exist in Input directory): in_file_10.spc
Enter k, for the k most similar frames: 2
```

- **Output:**

```
1,1,4,1,1.000000
1,1,7,1,1.000000
1,2,22,7,0.999788
1,2,16,5,0.999740
1,3,16,7,0.999677
1,3,19,7,0.999677
1,4,16,11,0.999464
1,4,19,11,0.999464
1,5,22,7,0.999408
1,5,10,9,0.999352
```

- **Format:** <Video Number, Frame Number, Similar Video, Similar Frame, Similarity Value >

3.3 Task 3

3.3.1 Task 3 (a)

Input:

```
Enter file name (default: in_file.gspc), the path is 'Input/', don't contain path:in_file_10_10.gspc
Loading database......
Database loaded......
Time consumed: 1.35454297066 seconds.
Enter m, for the m most significant frames:10
```

Output:

Format: Output_rank<RANK_NUM>_index_<VIDEO_NUM>_<FRAME_NUM>.png

49_22 **49_20** **49_21** **49_23** **49_24**

49_17 **50_25** **50_24** **50_22** **50_21**

3.3.2 Task 3 (b)

Input:

```
Enter file name (default: in_file.gspc), the path is 'Input/', don't contain path:in_file_10_10.gspc
Enter m, for the m most significant frames:5
Output_rank0_index_49_22.png
Output_rank1_index_49_20.png
Output_rank2_index_49_21.png
Output_rank3_index_49_23.png
Output_rank4_index_49_24.png
```

Output:

Format: Output_rank<RANK_NUM>_index_<VIDEO_NUM>_<FRAME_NUM>.png

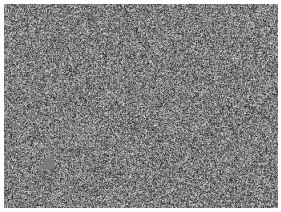
49_22 **49_20** **49_21** **49_23** **49_24**

3.4 Task 4

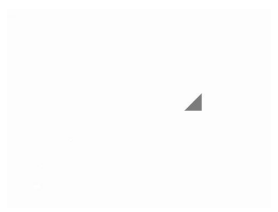
3.4.1 Task 4 (a)

Input:

```
Enter file name (default: in_file.gspc), the path is 'Input/', don't contain path:in_file_10_10.gspc
Enter first seed pair VIDEO_NUMBER,FRAME_NUMBER:50, 23
Enter second seed pair VIDEO_NUMBER,FRAME_NUMBER:35, 4
Enter third seed pair VIDEO_NUMBER,FRAME_NUMBER:36, 23
```



50_23



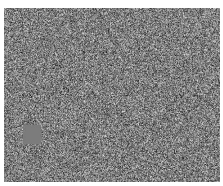
35_4

36_23

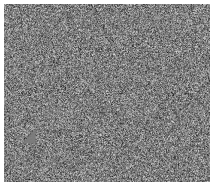
Output:

Format: Output_rank<RANK_NUM>_index_<VIDEO_NUM>_<FRAME_NUM>.png

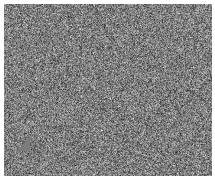
```
Time consumed: 3.30729198456 seconds.
Enter m, for the m most significant frames:10
Input_50_23.png
Input_35_4.png
Input_36_23.png
Output_rank0_index_5_3.png
Output_rank1_index_29_3.png
Output_rank2_index_29_2.png
Output_rank3_index_11_3.png
Output_rank4_index_29_4.png
Output_rank5_index_35_3.png
Output_rank6_index_35_2.png
Output_rank7_index_11_4.png
Output_rank8_index_35_11.png
Output_rank9_index_49_22.png
```



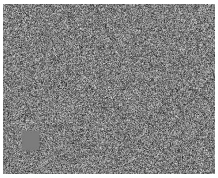
5_3



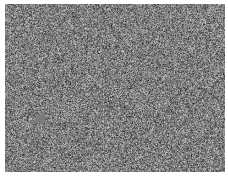
29_3



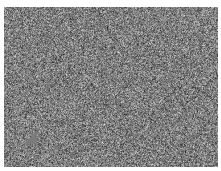
29_2



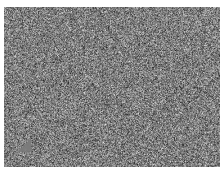
11_3



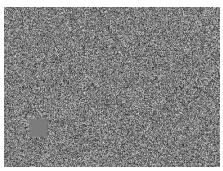
29_4



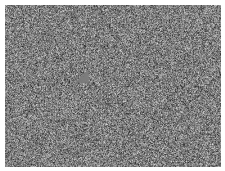
35_3



35_2



11_4



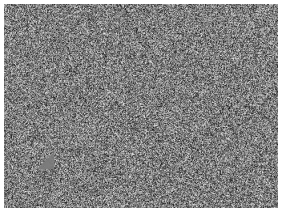
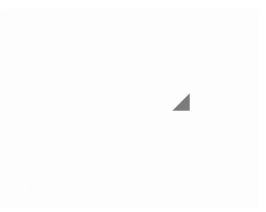
35_11

49_22

3.4.2 Task 4 (b)

Input:

```
Enter file name (default: in_file.gspc), the path is 'Input/', don't contain path:in_file_10_10.gspc
Enter first seed pair VIDEO_NUMBER,FRAME_NUMBER:50, 23
Enter second seed pair VIDEO_NUMBER,FRAME_NUMBER:35, 4
Enter third seed pair VIDEO_NUMBER,FRAME_NUMBER:36, 23
```


50_23 **35_4** **36_23**

Output:

Format: Output_rank<RANK_NUM>_index_<VIDEO_NUM>_<FRAME_NUM>.png

```
Enter m, for the m most significant frames: 10
Input_50_23.png
Input_35_4.png
Input_36_23.png
Output_rank0_index_49_22.png
Output_rank1_index_49_20.png
Output_rank2_index_49_21.png
Output_rank3_index_49_23.png
Output_rank4_index_49_24.png
Output_rank5_index_50_25.png
Output_rank6_index_49_17.png
Output_rank7_index_50_24.png
Output_rank8_index_50_22.png
Output_rank9_index_50_21.png
```





49_22 **49_20** **49_21** **49_23** **49_24**





50_25 **49_17** **50_24** **50_22** **50_21**

3.5 Task 5

- **Input:**

```
Enter the input file name(File should exist in Input directory): in_file_10.spc
Enter L, for L layers of LSH: 3
Enter K, for 2^K buckets in each hash layer: 5
```

- **Output:**

```
1, 16, 402657,9, 22, 2, 285.9622, 153.9246
1, 16, 2164030,49, 1, 2, 338.8908, 159.7361
1, 16, 2167624,49, 3, 1, 204.8818, 57.8134
1, 16, 2174456,49, 7, 4, 328.7052, 194.0258
1, 16, 2176230,49, 8, 3, 84.4194, 208.2084
1, 16, 2176246,49, 8, 1, 203.7462, 58.135
1, 16, 2176273,49, 8, 4, 328.0973, 206.7306
1, 16, 2178096,49, 9, 3, 84.4194, 208.2084
1, 16, 2182375,49, 11, 4, 328.4848, 193.1847
1, 16, 2184471,49, 12, 2, 338.9551, 160.5109
```

- **Format:** <Layer Number, Bucket Number, Index Number (spc database), Video Number, Frame Number, Cell Number, X, Y >

3.6 Task 6

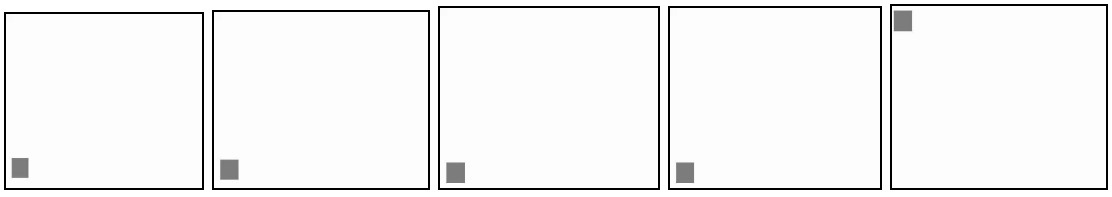
- **Input:**

  - **The Red box is the frame where we get the image**

```
Enter the input file name(File should exist in Input directory): in_file_10.lsh
Loading database......
Database loaded......
Enter n, for number of frames: 5
Enter Video, Frame No and 2 points seperated by space (V F X1 Y1 X2 Y2): 3 1 10 275 57 323
```

- **Output:**

```
Overall Sift Vectors Considered from all buckets: 2035729
Unique Sift Vectors Considered from all buckets: 568035
Original Database access Size: 68164440
Video No: 12.0 Frame No: 1.0
Video No: 24.0 Frame No: 1.0
Video No: 21.0 Frame No: 1.0
Video No: 9.0 Frame No: 1.0
Video No: 18.0 Frame No: 1.0
```



v9_f1.jpg **v12_f1.jpg** **v18_f1.jpg** **v21_f1.jpg** **v24_f1.jpg**

4. Interface Specification

4.1 Task 1

Execution instructions:

1. go to terminal or PyCharm console
2. Go inside cse515group1/ directory.
3. Run following command in python console:
   python task_filename.py

Example:

python task1.py

Note: Enter inputs as prompted on the console

d = 10 or 60

Expected output:

Each task outputs new files with reduced dimensions using PCA clustering respectively and corresponding scores for the new dimensions.

Note: Video files will be inside ./Input/ directory.

Example format of the file name:

in_file_d.spc

4.2 Task 2

Execution instructions:

1. go to terminal or PyCharm console
2. Go inside cse515group1/ directory.
3. Run following command in python console:
   python task_filename.py

Example:

python task2.py

Note: Enter inputs as prompted on the console

Input file = in_file_10.gspc or in_file_60.gspc

k = 2 or 10

Expected output:

Task 2 : based on method in task 1, create a similarity graph that each video frame v_a, v_b is one of the k most similar frames to v_a that is not already in the same video file.

Example format of the file name:

in_file_d_k.gspc

4.3 Task 3

Execution instructions:

1. go to terminal or PyCharm console
2. Go inside cse515group1/ directory.
3. Run following command in python console:

   python task_filename.py

Example:

python task3_pagerank.py

python task3_ascos.py

Note: Enter inputs as prompted on the console

Input file = in_file_10_2.gspc, in_file_10_10.gspc, in_file_60_2.gspc or in_file_60_10.gspc

m = a random integer

Expected output:

Most significant m frames and corresponding rank score.

4.4 Task 4

Execution instructions:

1. Go to terminal or PyCharm console
2. Go inside cse515group1/ directory.
3. Run following command in python console:

   python task_filename.py

Example:

python task4_pagerank.py

python task4_ascos.py

Note: Enter inputs as prompted on the console

Input file = in_file_10_2.gspc, in_file_10_10.gspc, in_file_60_2.gspc or in_file_60_10.gspc

m = a random integer

Three input video/frames

Expected output:

Most relevant m frames relative to the three input frames and corresponding rank score

4.5 Task 5

Execution instructions:

1. Go to terminal or PyCharm console

2. Go inside cse515group1/ directory.
3. Run following command in python console:
python task_filename.py

Example:
python task5.py
Note: Enter inputs as prompted on the console
Input file = in_file_10.spc
L = 3
K = 5

Expected output:
Hashed vectors containing layer and bucker indexes for input SIFT vectors.
Example format of the file name:
in_file_10_3_5.lsh

4.6 Task 6

Execution instructions:
1. Go to terminal or PyCharm console
2. Go inside cse515group1/ directory.
3. Run following command in python console:
python task_filename.py

Example:
python task6.py
Note: Enter inputs as prompted on the console
Input file = in_file_10_3_5.lsh and in_file_10_3_5.spc
N = nearest N frames

Expected output:
Most relevant n frames containing the same object as query input object.
Example format of the file name:
Video frame images in the output directory

5. Requirements

5.1 System requirements

Python 2.7 latest version:

- Operating System: Mac OS X 10.5 and later, Windows 7 or later versions, Linux
- Any Intel x86-64 processor.
- Will not work on Intel Itanium Processors (formerly IA-64).

5.2 Installation

Python

1. Download and install Python 2.7 latest version of your PC.
(https://www.python.org/downloads/release/python-2712/)
2. Then install pip and following binaries using pip:

```
sudo apt-get install python-pip
sudo pip install numpy
sudo pip install scipy
sudo pip install matplotlib
sudo pip install pprocess
sudo pip install h5py
sudo pip install scikit-image
sudo pip install scikit-learn
sudo pip install videosequence
sudo pip install imageio
sudo pip install pandas
sudo pip install networkx
```

For Windows download the files from: http://www.lfd.uci.edu/~gohlke/pythonlibs/
Then use pip install on those files
After successful installation, follow the execution steps.

6. Related Work

6.1 Efficient ASCOS++ Computation - Distributed ASCOS++ computation

Motivated by Chen and Giles [2013], Chen and Giles rewrite the equations to compute ASCOS++, and propose an efficient distributed computation approach.
Given a graph G and its adjacency matrix A = $[a_{ij}]$, we can get P = $[p_{ij}]$ as the column-normalized matrix of A. Then we can rewrite the equation 4 into:

$$(I - cQ^T)S = (1 - c)\, I$$

where Q = $[q_{ij}]$ = $[p_{ij}(1 - \exp(-a_{ij}))]$,
And then split S into n column vectors S = $[S_1, S_2, \ldots, S_n]$ and the identity matrix I into n column vectors I = $[I_1, I_2, \ldots, I_n]$, the above equation will turn to be:

$$(I - cQ^T) = (1 - c)\, I_i,\ i = 1, 2, \ldots., n$$

The original problem was divided by solving the matrix S of dimension n by n into n independent tasks. Since the matrix $(I - cQ^T)$ is diagonally dominant, we can determine the solutions by applying the Jacobi iterative method [Saad 2003]. The time complexity of obtaining the similarity score between one pair of nodes is O (n) because we only need to compute 1/n of the entire network.

6.2 The Anatomy of a Large-Scale Hypertextual Web Search Engine

The pagerank algorithm is one of the technology which Google uses for its web searching. Sergey Brin and Lawrence Page talk about each part of Google search engine in their dissertation, including the pagerank algorithm.

The definition of the pagerank listed below:
We assume page A has pages T1...Tn which point to it (i.e., are citations). The parameter d is a damping factor which can be set between 0 and 1. We usually set d to 0.85. There are more details about d in the next section. Also C(A) is defined as the number of links going out of page A. The PageRank of a page A is given as follows:
PR(A) = (1-d) + d (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))
Note that the PageRanks form a probability distribution over web pages, so the sum of all web pages' PageRanks will be one.[10]

Through counting links to a given page, Google can estimate the importance of the page. Pagerank corresponds to the principal eigenvector of the normalized like matrix of the web.

7. Conclusion

As part of this project, we worked on video object search and retrieval. We got an opportunity to learn about the working of search algorithms like PageRank and ASCOS++ both classic and personalized which is useful in wide variety of Multimedia Information Retrieval applications like search engines. We also implemented LSH and Nearest Neighbor Search using different methods. Overall we got good results for all the tasks because they were able to get correct results when compared compared to the visualization test.

8. References

1. Alexandr Andoni and Piotr Indyk. “Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions”. Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122.
2. “Locality Sensitive Hashing”, https://en.wikipedia.org/wiki/Locality-sensitive_hashing
3. Donoho, "High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality”
4. “pagerank”, Networkx-PageRank, https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html#pagerank

5. Hung-Hsuan Chen and C. Lee Giles. 2015. ASCOS++: An Asymmetric Similarity Measure for Weighted Networks to Address the Problem of SimRank. ACM Trans. Knowl. Discov. Data 10, 2, Article 15 (October2015)
6. "PageRank", https://en.wikipedia.org/wiki/PageRank#Simplified_algorithm.
7. *Vedaldi, Andrea, Extract SIFT features* (http://vision.ucla.edu/~vedaldi/assets/sift/#sift)
8. http://www.inf.fu-berlin.de/lehre/SS09/CV/uebungen/uebung09/SIFT.pdf
9. http://www.vlfeat.org/api/sift.html.
10. The Anatomy of a Large-Scale Hypertextual Web Search Engine http://infolab.stanford.edu/~backrub/google.html.
11. Networkx-pagerank, https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html.
12. PageRank, http://nlp.stanford.edu/IR-book/html/htmledition/pagerank-1.html.
13. Malcolm Slaney and Michael Casey, "Locality-Sensitive Hashing for Finding Nearest Neighbors", http://www.slaney.org/malcolm/yahoo/Slaney2008-LSHTutorial.pdf.

Appendix

- Rubinder Singh
    - Worked equally for all tasks.
    - Performed testing.
    - Worked on Report.
- Lei Guo
    - Worked on task 2, 3 and 4 primarily.
    - Worked on Report.
- Mihir Thakkar
    - Worked on task 2 and 5 primarily.
    - Worked on Report.
- Srujan Kanteti
    - Worked on task 1, 2, 5 and 6 primarily.
    - Worked on Report.
- Yiqian Zhang
    - Worked on task 3 and 4 primarily.
    - Worked on Report.
- Yu-Hsuan Chuang
    - Worked on task 3 and 4 primarily.
    - Worked on Report.