Università di Pisa
Enterprise Information Management

ZeeMAGE

A visual similarity search system based on the Bag-of-Features approach using Lucene

Hello!

TEAM COMPOSITION

Alessio Apollonio - alessio.apo@gmail.com
Daniele Battista - daniele.bat90@gmail.com
Dario Ferrarotti - d.ferrarotti@gmail.com
Alessio Sanfratello - sanfra90@gmail.com
Tommaso Vongher - tom.vongher@gmail.com

INTRODUCTION

Objective

The objective of this project is to produce an online **Similarity Search Engine** based on images.

This engine will identify the similarity of the input image with a predefined **Dataset**, and will produce the images in the dataset that match the most with the given query.

The chosen approach is a version of the classical **bag of words** approach. In addition to that we improved the results using **RANSAC both with ORB and SIFT** features.

Dataset*

The Holidays dataset is a set of images which mainly contains some personal holidays photos. The remaining ones were taken on purpose to test the robustness to various attacks: **rotations**, **viewpoint** and **illumination changes**, **blurring**, etc...

The dataset contains **500 image groups**, each of which represents a distinct scene or object. **The first image of each group is the query** image and the correct retrieval results are the other images of the group.

The total size is around **1500 images.**

(*) Hamming embedding and weak geometric consistency for large scale image search Hervé Jégou, Matthijs Douze, Cordelia Schmid: http://lear.inrialpes.fr/pubs/2008/JDS08/

The computation takes place in two distinct phases:

OFFLINE PHASE

- **SIFT Feature Extraction**
  - Resize images
  - Features Reduction
- **K-Means clustering**
  - Parameters Tweaking
- **Textual representation & Indexing**

ONLINE PHASE

- **SIFT Feature Extraction from Query Image**
- **Textual Representation**
- **Similarity Search**
- **RANSAC ordering**

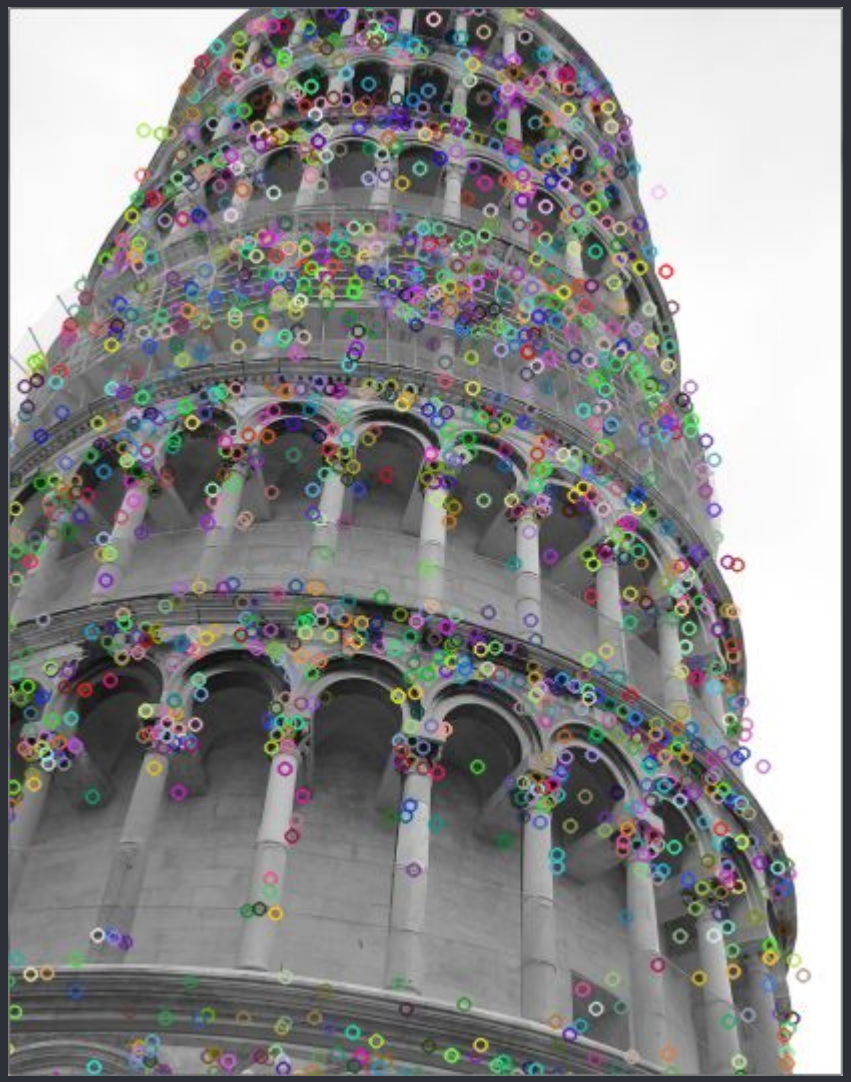
OFFLINE PHASE

1

Features Extraction

Using Scale Invariant Features Transform

FEATURES REDUCTION



We imposed a bound to the number of features that have been extracted from each image. This decision is due to the fact that most part of the images is characterized by 3000-4000 local features.

The reduction of local features, considering that each detected keypoint correspond to a local features, has been achieved by discarding the less significant in term of their "response" value.

The bound chosen after different trials is 1000.

FEATURES REDUCTION IN PRACTICE

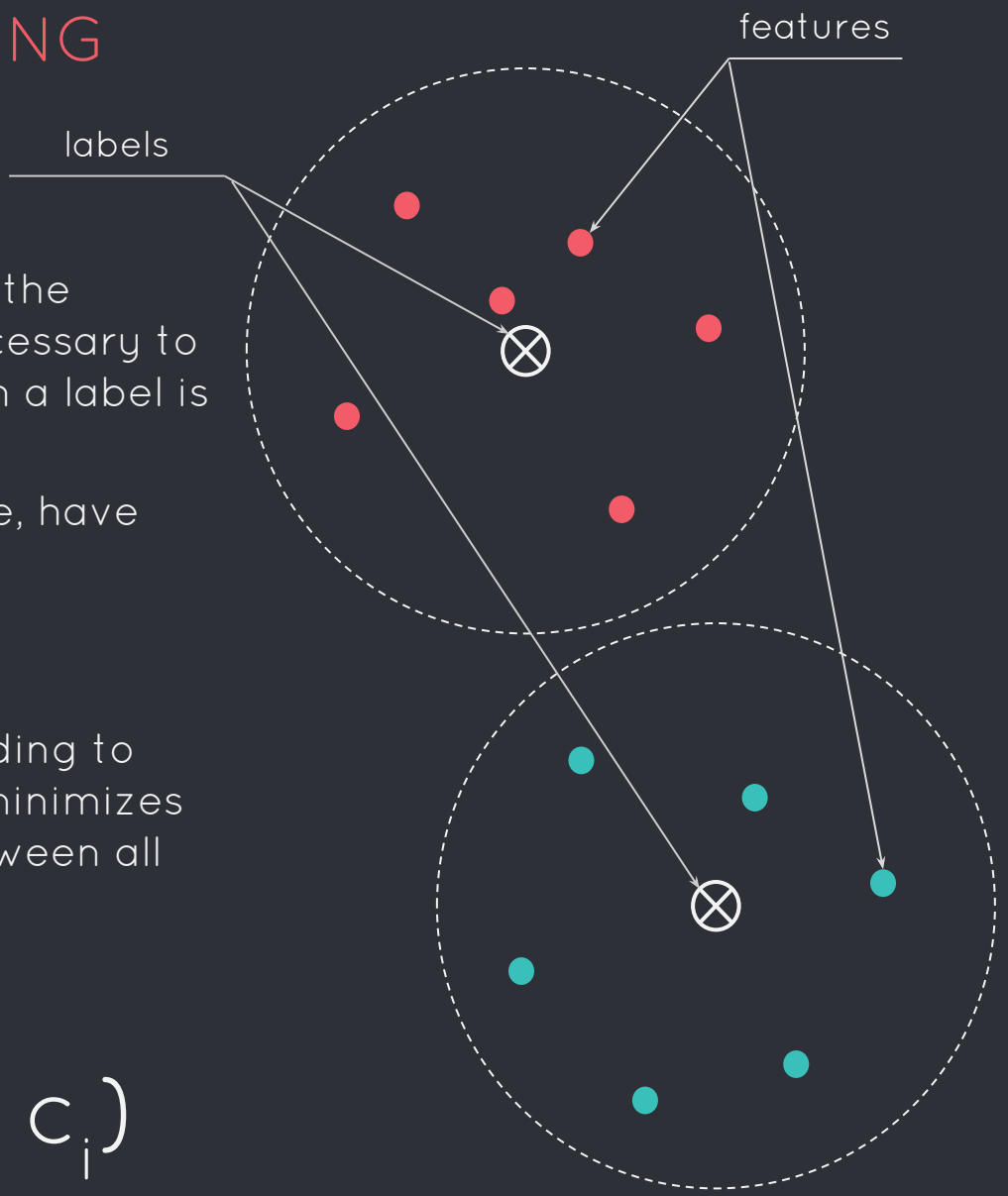
```
List<KeyPoint> listOfKeypoints = keypoints.toList();
Collections.sort(listOfKeypoints, new Comparator<KeyPoint>()
{
    @Override
    public int compare(KeyPoint keypnt1, KeyPoint keypnt2)
    {    if(keypnt1.response < keypnt2.response)
            return 1;
        else if(keypnt1.response > keypnt2.response)
            return -1;
        return 0;
    }
 });
List<KeyPoint> listOfBestKeypoints  =
    new ArrayList<KeyPoint>(listOfKeypoints.subList(0, listSize));
keypoints.fromList(listOfBestKeypoints);
```

2

K-Means Clustering

Subdividing the features into clusters

K-MEANS CLUSTERING



Once all the features from all the images are extracted it is necessary to create our dictionary in which a label is assigned to each feature.

Similar features will, therefore, have the same label.

All of this can be achieved by clustering the features according to the K-means algorithm that minimizes the sum of squared error between all the objects:

$$E=\sum_{i=1}^{k}\sum_{p\in C_i} \text{dist}(p, c_i)$$

K-MEANS CLUSTERING IN PRACTICE

The main issue was trying to reduce the computation time for the offline phase and the size of the center's matrix to be saved. To do so we selected just 1 attempt to execute the clustering algorithm and other parameters as follows.

OUR CONFIGURATION:

K = 30000 (number of clusters)

ε = 0.9 (the desired accuracy at which the iterative algorithm stops)

iterations = 10 (maximum number of iterations before algorithm stops)

attempts = 1 (repetitions of the whole process)

IN PRACTICE:

```
TermCriteria criteria = new TermCriteria(TermCriteria.MAX_ITER
+TermCriteria.EPS, 100, 0.9);
Core.kmeans(totalFeatures, 30000, clusteredHSV, criteria, 1, Core.
KMEANS_PP_CENTERS, centers);
```

3

Textual Representation & Indexing

The “bag of features” approach

TEXTUAL REPRESENTATION

Cluster3 Cluster1 Cluster2

Cluster1 Cluster3 Cluster1

...

Cluster3 Cluster2 Cluster3

...

...

...

Now we need to represent each image with a text file.

This representation simply contains the **list of all the cluster's ID** to which each of its features is assigned.

INDEXING

We can now use **Apache Lucene** in order to create an **index** that will allow to match a query to other similar images.

The index will maintain the information about the relation between a word (cluster ID) and a document(image) that contains it.

ONLINE PHASE

ONLINE PHASE DESCRIPTION

Objective

Objective of this phase is to respond to a query image given as input by the user with a collection of the **N images more similar in the available dataset**.

The dataset is composed by the roughly 1000 images on which we build the dictionary while the query image is chosen between a set of 500.

Preparation

The first thing to do is to load the matrix with the cluster centers computed by K-Means that was saved in GSON*.

This operation must be performed only at initialization, not for every query.

(*) Java library that can be used to convert Java Objects into their JSON representation
http://github.com/google/gson

1

Feature Extraction from Query Image

Using Scale Invariant Features Transform.

2

Textual Representation

As in the offline phase at this point we want to create a textual document to represent the query image.

In order to do so we insert in the document the ID of the cluster
to whom the feature was associated.

3 Similarity Search

Through the use of Apache Lucene we are able to establish a **list of N(10) similar images by means of a textual search** in the index produced during the Offline Phase.

4

Random Sample Consensus (RANSAC)

Based on ORB features

ORDERING THE RESULTS

At this point the system is capable of give back N results that are the most similar to the one in input. However there are no informations about how similar the single results with the query.

However the user might also be interested in this kind of information. In this case we can resort to an ordering method based on the Random Sample Consensus (RANSAC).

This method involves 4 steps:

1. **ORB Features extraction**
2. **Query matching and filtering**
3. **Execution of the RANSAC algorithm**
4. **Ordering**

FEATURES DETECTION AND MATCHING

We start detecting and extracting ORB (faster than SIFT) from both the query and Lucene results

For each of the results we want to find matches with the query.

Since the descriptors are binary we can use the Hamming Distance.

At the end we filtered the results discarding all the ones with a value of the Hamming Distance greater than 35*.

(*) experimental results show that this is the limit distance for an optimal filtering

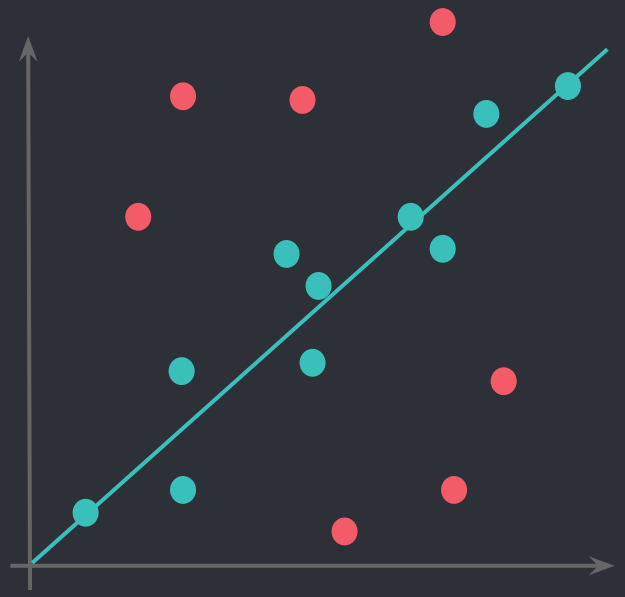
RANSAC

If the image has at least 4 good matches, we can execute RANSAC.

This algorithm, given the image keypoint and the good matches computes the number of inliers*.

The greater the number of inliers, the greater the similarity.

For each image is computed the number of inliers, then a list of couples (image, number of inliers) is produced.



Such list is then ordered by number of inliers

(*) data whose distribution can be explained by some set of model parameters, though may be subject to noise. Opposed to "outliers" which are data that do not fit the model.

5

Random Sample Consensus (RANSAC)

Based on SIFT features

USING SIFT FEATURES

ORB features are quick to be extracted but perform poorly in respect of SIFT features.

We can imagine to sacrifice some execution time in order to provide better results by using the SIFT features instead.

PROBLEM:
In this case we don't have an experimental result that provides an optimal distance to be used to filter the results.

SOLUTION: 2NN APPROACH

For each SIFT feature of the query image we use 2NN to find the two most similar SIFT features from one of the Lucene result images.

If:

$$\frac{\textit{distance}\ (\text{SIFT\_query, 1st SIFT\_train})}{\textit{distance}\ (\text{SIFT\_query, 2nd SIFT\_train})} < 0.64$$

then we can assume that the first NN result is a good match.

Now that we can find good matches we can repeat the process seen before and compute inliers with RANSAC.

RANSAC IN EXECUTION: query 106600.jpg

Bag of features:

106601.jpg

106602.jpg

106603.jpg

106604.jpg

145802.jpg

102701.jpg

132401.jpg

129202.jpg

106606.jpg

101401.jpg

Finding the inliers

Image	Good	Inliers
106601.jpg	157	82
106602.jpg	81	48
106603.jpg	66	27
106604.jpg	73	24
145802.jpg	21	5
102701.jpg	11	0
132401.jpg	17	6
129202.jpg	10	0
106606.jpg	47	8
101401.jpg	22	5

Ransac reorder

106601.jpg

106602.jpg

106603.jpg

106604.jpg

106606.jpg

132401.jpg

145802.jpg

101401.jpg

Thanks for your attention!