University "POLITEHNICA" of Bucharest

Automatic Control and Computers Faculty,
Computer Science and Engineering Department

A scalable algorithm for similar image detection

Scientific Advisers:

Prof. Dr. Ing. Nicolae Ţăpuş
Ing. Ştefan-Teodor Crăciun

Author:

Andrei-Bogdan Pârvu

Abstract

In this paper, I propose an algorithm for detecting similar images in a large scale dataset, with the goal of obtaining a method of determining the occurrence of copyright infringement. I use the Harris corner detection and the SIFT keypoints and descriptors for analyzing the properties of an image, and then maintaining the data in a KD-tree structure for fast querying. I also describe the application architecture and the way the data should be distributed so that an efficient use of multiple machines can speed up the algorithm.

Contents

Chapter 1

Introduction

Image processing has been a very important research domain the last years, the ever increasing number of images present on the Internet becoming more and more challenging to index, store and analyze.
Privacy and copyright have also been big issues which have been discussed and assessed for a long time, major users in the photography industry wanting to know at all times when someone is using or modifying one of their photos.

1.1 Requirements

We want to design a scalable algorithm which can maintain a large set of images, and perform queries of finding a highly similar image with a given input one.
The algorithm should be focused on copyright detection, so it should be able to determine if the input image is one of the images in the dataset, with possible transformations applied upon it:

- watermarks
- scaling
- cropping
- various filters

As said above, the granularity of the algorithm should be able to distinguish between two lightly similar images (e.g. two different pictures of the Eiffel Tower, made by two different persons) and two images, one of which is obtained from the other.

The running time of the algorithm is also a major factor which should be seriously taken into consideration. It should be able to handle a large number of queries and provide a small response-time per query. Of course, there is a close connection between the running time of the algorithm and its precision, connection which should be closely analyzed in order to create a balance between the two.

In Figure 1.1, we can see an example usage of the algorithm. The large database is queried with the image in the bottom-left corner, and the algorithm is capable is returning the original image (even if the query image has been cropped, rotated and a gray filtered applied).

Figure 1.1: Basic usage of our algorithm

Chapter 2

Design of the Algorithm

2.1 Analysis of an image

2.1.1 Using Harris corner detector

The analysis of an image begins with applying the Harris detector on it. It will compute a given score for each pixel of the image, the higher the score, the greater the chance that pixel represents a corner.
Using these values we will have to determine a subset of pixels that will represent the Harris mask of the image. Experimentally, I have established that all the points which have a value greater than $0.01 * max\_image\_value$ are to be part of the subset.
As an example, Figure 2.1 shows a normal image of a woman's face, while Figure 2.2 shows the corresponding pixels that form the corner mask.

Figure 2.1: Sample image

Figure 2.2: Harris corner mask applied

As it can be seen from the image, the corner mask centers around the interest points in the image, the eyes and the eyelashes, while leaving the smooth surfaces (the skin) unmarked.

Another observation is that possible watermarks can be present in the image (as seen above), which, of course, the mask detects (they are center pieces of the image, and the corner algorithm cannot determine that they have no real connection with the image).

Furthermore, in some cases the watermark might contain all (or at least a vast majority of) the points in the mask (as seen in Figure 2.3). To avoid such a behavior, I have split the image into 9 sub images (three rows and three columns of identical dimensions) and the Harris mask is applied to each of these.

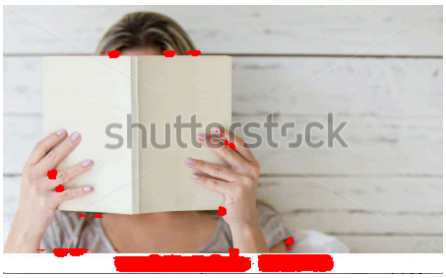

Figure 2.3: Single image

Figure 2.4: 9 subimages

This determines the corner mask to also contain pixels from the center of the image (some of which, unfortunately, are also a watermark).

2.1.2 Using SIFT keypoints and descriptors

As we have seen, the SIFT algorithm computes the keypoints of an image, and then the descriptors of these keypoints. Due to the high similarity nature of our problem, we do not want to compute the keypoints for the entire image, but filter them based on the corner mask determined in the previous section (as observed in Figure 2.5 and Figure 2.6)

Figure 2.5: SIFT keypoints for the entire image

Figure 2.6: SIFT keypoints for corner mask

The SIFT descriptors are then computed for each keypoint located in the corner mask and this will be the information stored for a certain image.

2.2 Analysis of a pair of images

In order to analyze a pair of images, we shall use the SIFT descriptors determined in the previous section. The two sets of descriptors are compared in order to obtain the best matches between pairs of keypoints. A distance is computed between each pair of keypoint descriptors, which is the Euclidian norm between the SIFT descriptors of the keypoints. Experimentally, I

have concluded that a match between two keypoints has a high similarity if the Euclidian norm is less than 100.
For a pair of images, we first find the set of corner-mask keypoints and then compute the best matches between these keypoints. We shall keep only the best 10 matches, and compute the arithmetic mean between the distances of these matches. As stated before, if this mean is smaller than 100, the two images are considered similar. We shall name this algorithm the *pair similarity algorithm*.
Figure 2.7 shows the corresponding matches between two images with two different watermarks, one of which is rotated 90^o clockwise.

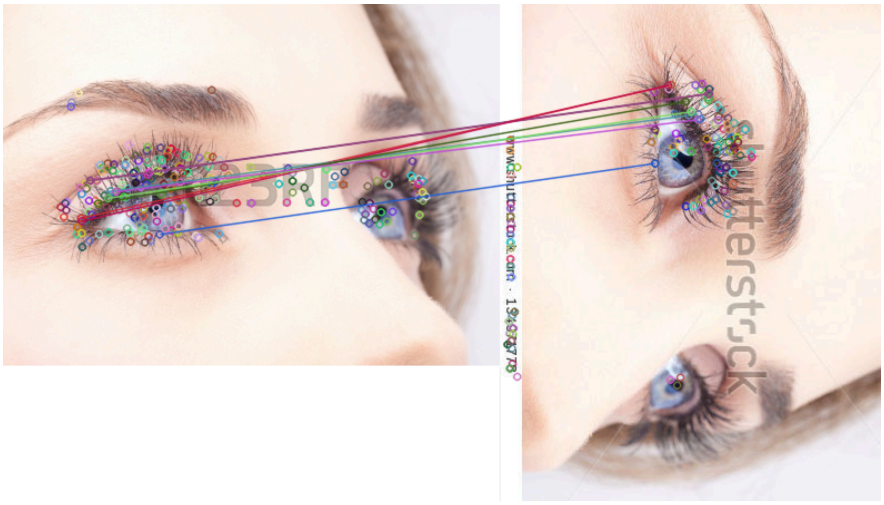

Figure 2.7: Comparison between two images

2.3 Analysis of a set of images

Suppose that we have a set of images, and we want to compare a test image with the set and detect whether we have a similar image within the set. Of course, the first possibility is the brut force one: we iterate through all the images and apply the *pair similarity algorithm* described in the previous section. Although this provides a correct result, it has a complexity of $O(number\_of\_images * image\_match\_time)$. We shall name this basic algorithm as the *linear algorithm*. Although this algorithm is very straightforward, its complexity is undesirable if the number of images becomes large. Moreover, most of the images in our set will likely have a big similarity distance with our searched image, so maybe we don't want to apply the full *pair similarity algorithm*.
Thus, we need to determine an efficient algorithm which can filter the initial set of images to a smaller set which contains very likely matching candidates with our test image.
The filtering algorithm is implemented as follows: we will maintain a maximum number of M descriptors for each image in the initial set and create a KD-tree with the set of descriptors of the initial images. When a query for a test image arrives, we will compute its descriptors and then perform a N nearest-neighbor search on our KD-tree. Then we will select the top T images with the most descriptors returned by the KD-tree and perform the *linear algorithm*. Experimentally, I have determined that the optimal values for M, the number of descriptors, N, the number of neighbors and T, the number of images is 40, 5 and 10. We shall name this algorithm the *kdtree algorithm*.

Chapter 3

Architecture

3.1 Basic structure

To illustrate the functionality of the described algorithm, I have designed a basic application which maintains a data base of images and, for a given query image, can retrieve the best match between the query and the image set.
The basic structure of the application is shown in Figure 3.1: all the queries are received by the Front End Processor (FEP) and distributed to several image servers which will determine the most similar image, return it to the FEP which will in turn return it to the querying instance. There are several advantages of separating the image servers from the FEP. First of all, it allows load balancing: the FEP can maintain a queue of requests and the server with the least work to do can pick it up and process it. Secondly, it allows different types of image servers to be tested, by easily inserting and removing the image server module.
The querying instance type is not particularly important, it can be either a simple web page or a browser extension (e.g. Google Chrome extension).

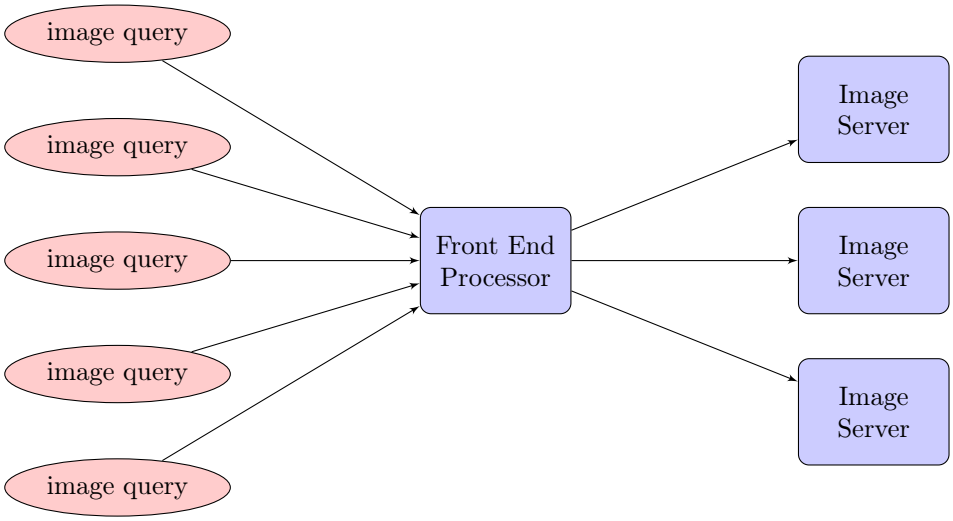

Figure 3.1: Basic structure

3.2 Front End Processor

The FEP acts as a classical server. It uses socket communication in order to receive the queries and send them to the designated image server. When a response is received from the image server the FEP sends it back to the entity that has made the given query.

3.3 Image Server

3.3.1 Linear Server

The first type of server uses the *linear algorithm* described in Design of the Algorithm. It stores the list of images that form our database and applies the above mentioned algorithm when a given query arrives from the FEP.

3.3.2 KD-tree Server

The second type of server uses the *kdtree algorithm* described in Design of the Algorithm. At initialization time the server reads the list of images that comprise our database and applies the above mentioned algorithm. The resulting KD-tree is maintained in memory throughout the incoming queries and it it used to find the most similar image and return it to the FEP.

Chapter 4

Results

I have tested the two algorithms described in Design of the Algorithm using the structure given in Architecture.
There are two major parts which should be analyzed: the correctness of the algorithm and it's running time.

4.1 Correctness

For testing purposes, I have used a database of 500 images, containing groups of the same image with different watermarks and sizes. I ran the *linear algorithm* in order to form groups of matching images which could be easily checked for similarity.

4.2 Running time

To be able to determine the speed of the two presented algorithms I have run them on sets of $5, 10, 20, 50, 200, 350$ and 500 images. The corresponding running times are shown in Figure 4.1. The red line shows the corresponding running times for the *linear algorithm*, while the green one show the times for the *kdtree algorithm*. As it can be seen, the *kdtree algorithm* outperformes the linear one, with the same returned image (so it does not give different results). The non-ascending running times of the *kdtree algorithm* can be explained by the fact that the KD-tree data structure is traversed heuristically, so depending on the given input a certain query can execute with varying running times.
Of course. the *kdtree algorithm* does require an initialization time, which is the price that has to be paid in order to perform fast queries. Figure 4.2 shows the initialization time of an image server which constructs a KD-tree server. As it can be seen, this time grows linearly, but this gets compensated by the small query time. For a 500-size image set, the initialization of the KD-tree takes 35.476 seconds, which is less than a query on the same image set using the *linear algorithm*, which takes 77.331 seconds.

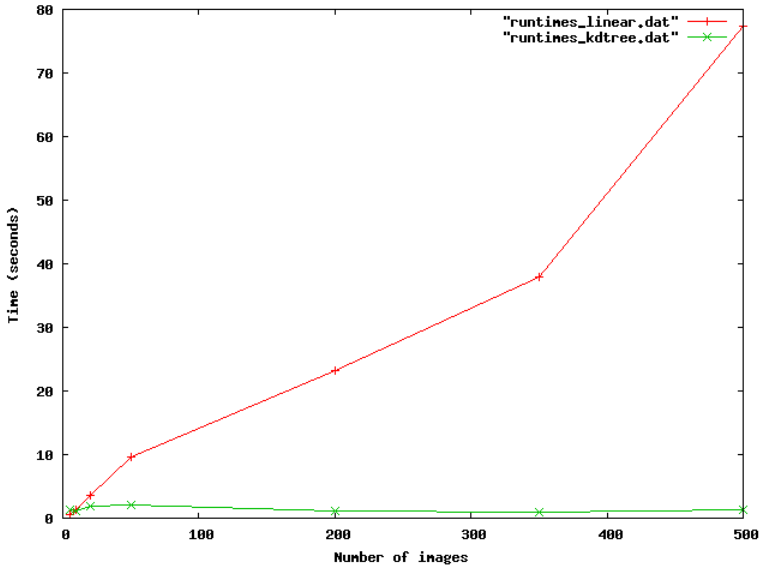

Figure 4.1: Runtime of the two algorithms

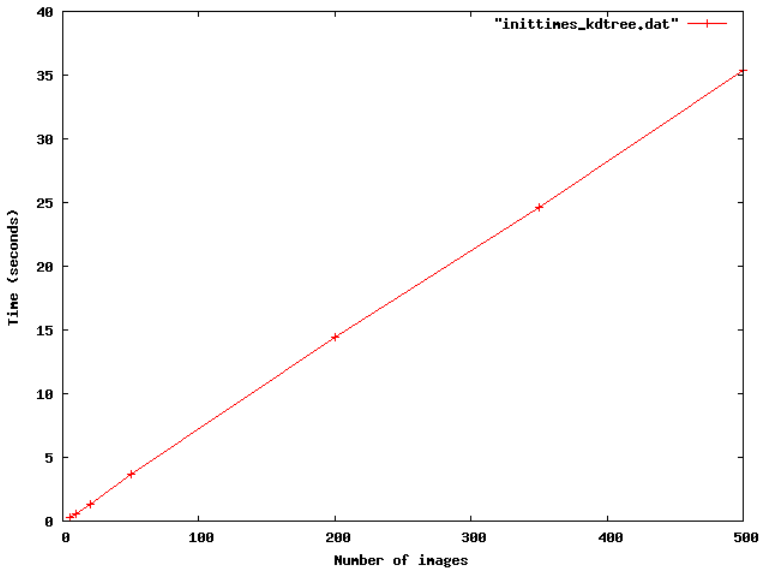

Figure 4.2: Initialization time of a KD-tree server

Chapter 5

Conclusions

In this paper I have designed a scalable algorithm that is capable of detecting a highly similar images in a large database.
I have succeeded in obtaining a small response time for querying the database in finding a highly similar image with a given one, maintaining at the same time a good quality of the responses.

5.1 Further Work

This algorithm can be further extended for handling a dynamic database, in which operations as adding and removing an image can be present along with the query operations previously described.
Also, the structure and size of the KD-tree data structure can be further analyzed, in order to determine the moment in which new data should be used to create a new KD-tree.
The management and distribution of the queries between the FEP and the image server should be considered as a possible improvement, possibly establishing several filtering layers instead of a single one (as described in Architecture).