Object recognition and computer vision, Master MVA, 2017/2018

Assignment 1: Instance-level recognition

Rémi Lespinet

remi.lespinet@ens-paris-saclay.fr

I Sparse features for matching specific objects in images

I.A SIFT features detections

QIA.1: (i) Why is it important to have a similarity co-variant feature detector? (ii) How does this affect the descriptors computed at these detections? (iii) How does this affect the matching process?

(i) For a wide range of application, it's really important to have a feature detector match the same features when there's rotation translation or scale change. In our case, we want our ability to recognize objects to be independant from their position in the scene, in particular we want to find the same features for an object regardless of

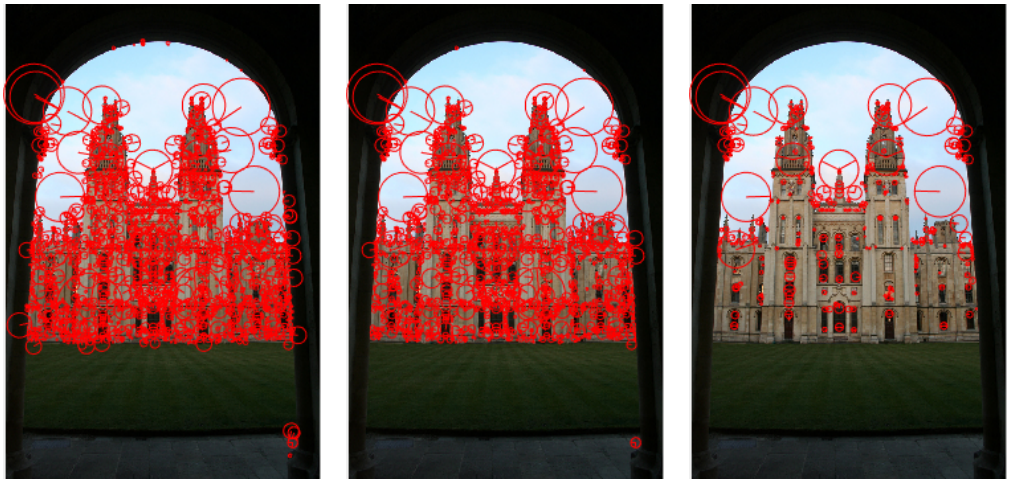
- if its position in the image space has been translated : translation invariance
- if it's close or far from the viewpoint : scale invariance
- if the object or the image has been rotated in the image space : rotation invariance

(ii) In order for the co-variant feature detector to make sense, the descriptors computed at these detections must also be invariant by rotation, scale and translation.

(iii) The matching process is simplified, since the descriptors are invariant by rotation scale and translation, we just have to match descriptors that are close to each other (for example in the euclidan distance sense). This would yield a match even if one image is rotated, scaled or translated compared to the other.

QIA.2: Show the detected features in the two images for three different values of the peakThreshold option

The detected features for three values of the *peakThreshold* parameters (0.0005, 0.001 and 0.005) are represented on figure 1 for the image 1 (*all_souls_000002.jpg*) and on figure 2 for the image 2 (*all_souls_000015.jpg*)



(a) $peakThreshold = 0.0005$ (b) $peakThreshold = 0.001$ (c) $peakThreshold = 0.005$

Figure 1: Feature detectors with three different values of $peakThreshold$ for image1

(a) $peakThreshold = 0.0005$ (b) $peakThreshold = 0.001$ (c) $peakThreshold = 0.005$

Figure 2: Feature detectors with three different values of $peakThreshold$ for image2

QIA.3: Note the change in spatial density of detections across images, for a given value of peakThreshold. (i) Is the density uniform? If not, why? (ii) Which implications for image matching can this have? (iii) How can it be avoided ?

(i) The spatial density is not uniform between images, the value of the parameter "peakThreshold" determines the minimum value of the difference of gaussian that is accepted. Zones of the image that have more details will have more keypoints, moreover, this is not invariant by non linear contrast change, so zones that are more contrasted will also have a higher number of contrast points.

(ii) This will have an impact of matching, if the images that we want to match have very different contrasts, the same value of the peakThreshold can lead to a very different

number of points, which is an additionnal difficulty in the matching process

(iii) There are two way to avoid this issue, the first is to process the image before finding the keypoints (for example histogram equalization), and the second is to filter the keypoints after this operation.

I.B SIFT features descriptors and matching between images

QIB.1: Note the descriptors are computed over a much larger region (shown in blue) than the detection (shown in green). Why is this a good strategy?

The detection part of the SIFT algorithm gives us a point, and a scale, the idea then is to compute the descriptors over a region around the keypoints found. This is a good strategy, because it makes the the descriptors robust against small changes in the local geometry around the detected keypoint.

QIB.2: Examine carefully the mismatches and try to understand the different causes of mismatch. (i) In your report, present at least 3 of them and explain why the mistakes are being made. For example, is the change in lighting a problem? (ii) What additional constraints can be applied to remove the mismatches?

(i) The figure 3 presents 50 matches given by the nearest neighbor methods for the two images *all_souls_000002.jpg* and *all_souls_000015.jpg* for *peakThreshold = 0.005* The first thing we can notice is that there's a lot of mismatch

- In the right image of figure 3 we see that half of the building is covered with shade, and we see that none of the 3D points at this location in image 1 have been matched at the right place, this is because the whole shaded region has a lower contrast, which implies that there are no keypoints in this region as explained in the question QIA.3. This is a first source of mistake.
- We also see that a window in the image2 reflects all the sun which results in saturation, and this window appears completely white. There's a keypoint in the image 1 which would normally have a correspondance on this location in the image 2, but due to this light effect, a mistake is made for this point. The light is a huge problem, since it induces effects that are highly not linear (specularities, shades, ...).
- We see that the building has repeated patterns, windows, ornements are all very similars, this means, that for the descriptors that are located in a window will be really close to each other (they will be located in a small ball in $\mathbb{R}^{128}$), moreover taking a picture is a process that is noisy in essence (not even talking about the light conditions and the jpg compression which induces more noise). this makes the nearest neighbor algorithm retrieve a wrong descriptor (for example if the descriptor is located on a window, it retrieves another window).

(ii) We could apply constraints that impose all the matches to be coherent with each other (geometric constraints), e.g. find a subset of matches that are coherent in the geometrical sense. We could also drop the matches if there's too much uncertainty in this match in some sense (this is what we are going to do in the next section)

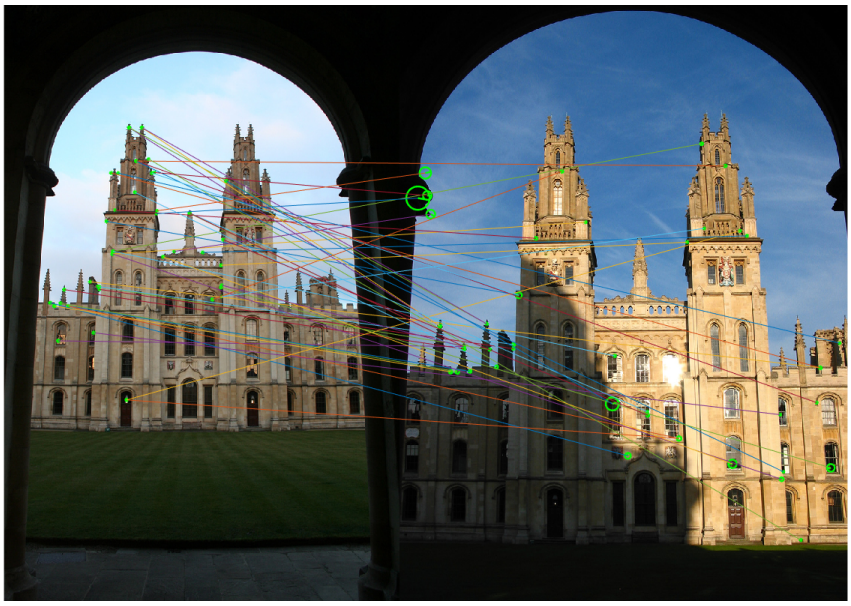


Figure 3: Representation of 50 matches given by the nearest neighbor methods for the two images *all_souls_000002.jpg* and *all_souls_000015.jpg* for $peakThreshold = 0.005$.

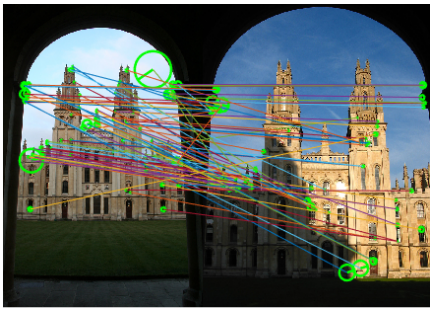
I.C Improving SIFT matching (i) using Lowe's second nearest neighbour test

QIC.1: Illustrate and comment on improvements that you can achieve with this step. Include in your report figures showing the varying number of tentative matches when you change the nnThreshold parameter.

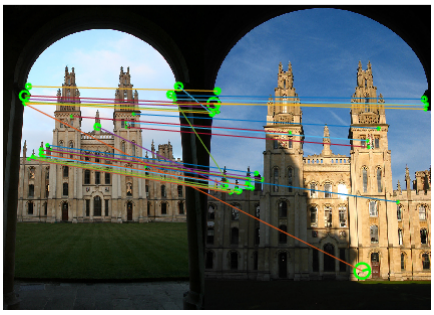
The first thing we can notice, is that it reduces drastically the number of points

nnThreshold	points	removed (%)
1 (unchanged)	458	0%
0.8	96	79.04%
0.6	35	92.36%
0.4	14	96.94%

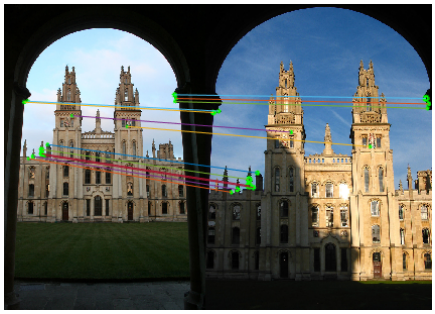
The figure 4 shows the different matches kept for 3 values of $nnThreshold$, we can see that for $nnThreshold = 0.6$ there's 35 points which are almost all good matches. For $nnThreshold = 0.4$, there's only 14 points left but they all are good matches.



(a) $nnThreshold = 0.8$ (96 points kept over 458)



(b) $nnThreshold = 0.6$ (35 points kept over 458)



(c) $nnThreshold = 0.4$ (14 points kept over 458)

Figure 4: Lowe's second nearest neighbour test for different values of $nnThreshold$

The figure 5 shows the 42 points removed when we take $nnThreshold = 0.99$, these are the 42 points whose nearest neighbor ratio are closest to 1. We can see that a lot of mismatch are removed, but some good matches are also discarded with this criterion.

Figure 5: Representation of the 42 points removed when taking $nnThreshold = 0.99$

We see in these examples that the method seems to work very well, at the cost of discarding a lot of points so there's a tradeoff between the number of points and the quality of the match.

I.D Improving SIFT matching (ii) using a geometric transformation

QID.1: Work out how to compute this transformation from a single correspondence. Hint: recall from Stage I.A that a SIFT feature frame is an oriented circle and map one onto the other.

Recall the QI.A, each SIFT frame consists of a position (t_x, t_y), a scale s, and a rotation θ. If we have a match, we have 2 SIFT frame $(t_x^{(1)}, t_y^{(1)}, s_1, \theta_1)$ and $(t_x^{(2)}, t_y^{(2)}, s_2, \theta_2)$

The relation is given by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{s_2}{s_1} \begin{bmatrix} cos(\theta_2 - \theta_1) & -sin(\theta_2 - \theta_1) \\ sin(\theta_2 - \theta_1) & cos(\theta_2 - \theta_1) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x^{(2)} - t_x^{(1)} \\ t_y^{(2)} - t_y^{(1)} \end{bmatrix}$$

QID.2: Illustrate improvements that you can achieve with this step.

The figure 6 presents the matches obtained after geometric verification, we can verify that all matches are correct. This method is a great improvement, it allows to keep the largest number of matches that are coherent with the hypothesis of similarity change between the two images. This is a good approximation in our case since the angles between the viewpoint and the planes of the scene doesn't change much between the two images.

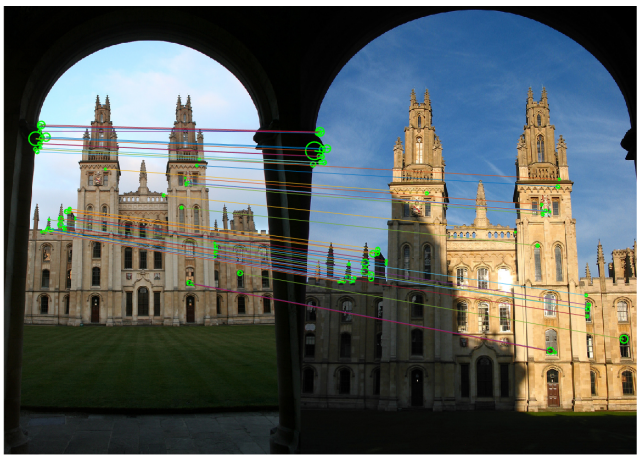
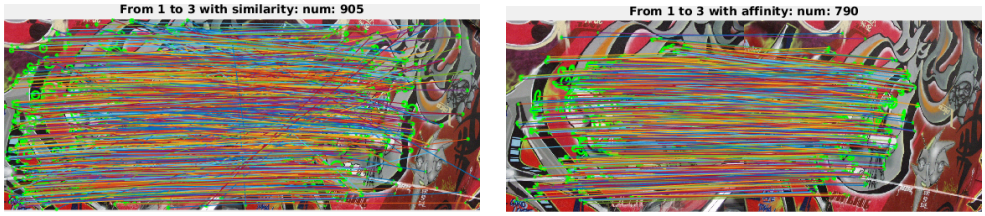


Figure 6: Representation of the matches obtained by geometric verification ($peakThreshold = 0.005$ and Lowe's test is not used)

II Affine co-variant detectors

QII.1: Include in your report images and a graph showing the number of verified matches with changing viewpoint. At first there are more similarity detector matches than affine. Why?

We see that the similarity co-variant detector (SIFT) behaves well between images 1 and 2 and between images 1 and 3, this is because the perspective transformation is not extreme, and it is well approximated by a similarity. The figure 7 compares the number of matches between similiarity and affine co-variant detectors.



(a) similarity co-variant detector (b) affine co-variant detector

Figure 7: Comparison of the number of matches between similarity and affine co-variant detectors for low perspective (images 1 and 3)

When the perspective between the two images starts to be more aggressive, the descriptors of the similarity co-variant detector fails to represent the local geometry accurately, which leads to a very low number of good matches after geometric verification. (In the exterme case of image *graf/img6.png* there's only 6 matches left, and only 3 of them are correct). The affine co-variant detector on the other hand, still provides a good number of matches even in extreme perspective conditions (see figure 8).

(a) similarity co-variant detector (b) affine co-variant detector

Figure 8: Comparison of the number of matches between similarity and affine co-variant detectors for high perspective (images 1 and 5)

The figure 9 shows the evolution of the number of matches as the change in perspective increases for the similarity co-variant and affine co-variant descriptor. We see that the number of matches in the similarity co-variant detector decrease more rapidly than the affine co-variant detector as the perspective becomes more and more extreme.

We notice that the similarity co-variant detector produces more matches when the perspective is small, the first thing that we can see is that the *getFeatures* function produces

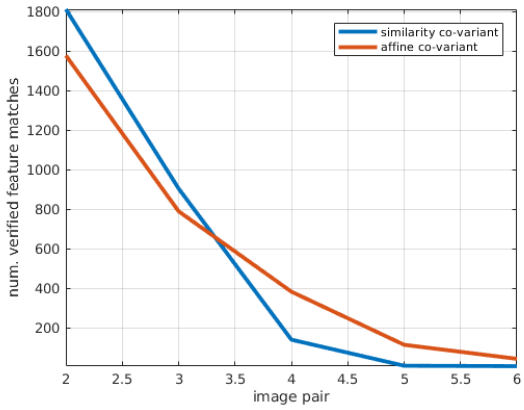


Figure 9: Representation of the number of matches as the function of the perspective (abscissa i corresponds to the number of matches between image 1 and image i)

less keypoints (see table below) but this cannot explain this behavior because the number of extra points found is negligeable (see the table below).

Image	Similarity co-variant	Affine co-variant
1	4474	4097
2	4837	4805
3	5158	5064
4	5222	5046
5	5803	5177
6	5660	4929

Table 1: Comparison of the number of keypoint produced with each detector for each image in our persepective set

By looking closely at the matches produced by the similarity co-variant detector after the geometric verification, we see that there's a lot of mismatches, this is due to the fact that we try to find a similarity between those two images, but the real underlying transformation is not a similarity, so the $RANSAC$ algorithm finds the best affinity that could best explain the matches, and it's parameters (t_x, t_y, s, θ) are erroneous. When computing the parameters for each match in the second part of the $RANSAC$ algorithm, we would have points whose θ and s are really close to the original, but whose t_x and t_y are really far, and these points would not be removed by the threshold.

III Towards large scale retrieval

III.A Accelerating descriptor matching with visual words

QIIIA.1: In the above procedure the time required to convert the descriptors into visual words was not accounted for. Why the speed of this step is less important in practice?

In practice this step is less important because we want to match an image against a database, and the step of converting thee descriptors to visual words can be done as a preprocessing step, e.g we only need to do it once for all images at the beginning, we can then use the visual words for every query without having to recompute them (except when we need to add images)

QIIIA.2: What is the speedup in seconds in searching a large, fixed database of 10, 100, 1000 images? Measure and report the speedup in seconds for 10, 100 and 1000 images.

Suppose that we have extracted n keypoints from image 1 and m keypoints from image 2, and suppose $m < n$

With this method, we only need to compute the word indices for the 2 images, this gives us a vector consisting of the id of each word associated with a descriptor. The matches can then be computed very efficiently in $O(n)$ by computing an histogram ($O(m)$) and iterating over the words indices of image 1 and lookup in the computed histogram ($O(n)$).

In the previous method, we had to find the 2 nearest neighbor for each descriptor (This is difficult in high dimension (128 for sift), using approximate nearest neighbor with LSH, the query complexity is about $O(dm^{1/(1+\epsilon)^2})$ with d being the dimension (128 for sift) and $1+\epsilon$ the factor between the real nearest neighbour and an accepted answer. We then have to iterate over all descriptors to have all matches, this lead to $O(dnm^{1/(1+\epsilon)^2})$ (There's also a preprocessing time that I dont put in the equation)

The following table report the time in seconds for the nearest neighbor method and for the visual words method

database size	ANN	Visual words
1	0.0253	0.000602
10	0.2492	0.004719
100	2.2120	0.042452
1000	22.1203	0.424516

Table 2: Comparison of the time (in second) needed to match descriptors from an image to images in the database with the 2 methods

To compute this table, I only measure the cumulated time spent on the *findNeighbor* for the ANN method and *matchWords* for the visual words. For the ANN method, this suppose that we have precomputed the features for all images in the database and that they fit in memory. For the Visual words method, this suppose that the kdtree has already been built (which is the usual case). each cells is the mean of 5 run of the algorithm on the corresponding database. For the 1000 size database, I added artificially pictures by duplication (because the database was not big enough)

III.B Searching with an inverted index

QIIIB.1: Why does the top image have a score of 1?

The top image has a score of 1 because it is exactly the image that we have queried (e.g we query an image that is in the database)

QIIIB.2: Show the first 25 matching results, indicating the correct and incorrect matches. How many erroneously matched images do you count in the top results?

The figure 10 shows the 25 images with the highest number of visual words in common with the query image representation. As explained in the previous question, the queried image is the the one in the top-left corner (with score 1)

Figure 10: top 25 images with the highest number of visual words in common with the query image (top-left image) in the database *oxbuild_lite*

We see that 9 images over the 25 images returned are correct (they represent the same building as the queried image) There are 16 images that are incorrect. We also notice that there are images of this building that do not appear in these 25 images, and that the best match (excluding the image itself) is incorrect.

However we can notice that the mismatched images, look very similar to the queried image (in particular, the Radcliffe Camera that is present 11 times has a very similar structure with pillars, statues, molding, and the materieals are very similars (same colors, some parts have bricks texture,...))

III.C Geometric rescoring

QIIIC.1: Why is the top score much larger than 1 now?

The idea is to use the 25 images obtained via the precedent algorithm, and add a geometric verification step. The score that we see in figure 11 is simply the number of matches after this geometrical verification step, this explains why the top score is much larger than 1, it corresponds to the number of geometrically verified matches between the image and itself (here 1241).

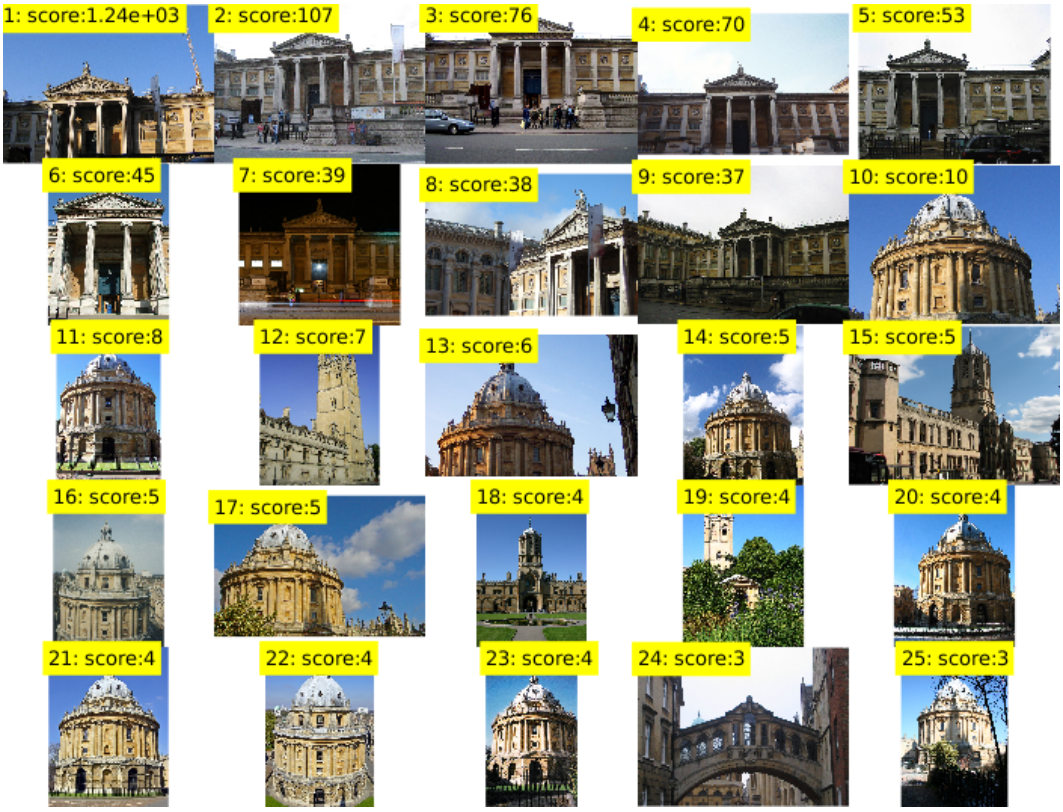


Figure 11: top 25 images with the highest number of visual words in common with the query image (top-left image) reordered according to their number of geometrically verified matches with the query image

QIIIC.2: Illustrate improvements of retrieval results after geometric verification.

The figure 11 presents the 25 images with the highest number of visual words in common with the query image, with their new score (the number of geometrically verified matches). The images have been re-ordered according to this new score.

Re-ordering according to this new score is a good improvement. The 9 good images are propulsed to the top of the classement. We can also notice there's a gap between the last good image (score 37 and the first wrong image 10), and all mismatches have a very low score (less than 10), this could be used to remove this images from the search.

We notice that there are images representing the same building that do not appear in this set of 25 images, which would have obtained a higher number of geometrically verified matches than some of the images in this set (namely the ones that are wrong), but the whole point of doing the search this way is to avoid the computationally expensive cost of geometric verification for all images. So this is a good approximation, altough it could discard the best matching image in the dataset if there are many other images that have a huge number of local patches in common with the query image, it's unlikely that it happens.

IV Large scale retrieval

QIV.1: How many features are there in the painting database?

There's 100000 visual words in this database.

QIV.2: How much memory does the image database take?

The file takes 191 906 193 bytes on disk.

QIV.3: What are the stages of the search? And how long does each of the stages take for one of the query images?

1) **Images features**
    1) Compute images descriptors
    2) Quantify descriptors using the database and get a visual words index per descriptor
    3) Create the histogram with visual words indices
2) **Inverted index**
    1) For all images in the database compute a similarity score between this image's histogram and the histogram of the queried image
    2) Sort the images according to their similarity score in descending order

3) **Geometric verification**

1) For the first N images (here $N = 100$)
   (i) Search for a good subset of matches that are geometrically coherent with the queried image (RANSAC) (Geometric verification)
   (ii) Assign this image the number of geometrically verified matches as geometric score
2) Sort the first N images according to their geometric score (descending order)
3) Return the first $M < N$ images

I executed the algorithm 10 times on each mystery painting. The following table presents the mean time spent on the different parts. The difference in execution time is due to the images having very different sizes.

Images	Images features	Inverted index	Geometric verification
mistery-painting1.jpg (150x182)	0.181	0.0206	0.242
mistery-painting2.jpg (286x232)	0.326	0.0207	0.383
mistery-painting3.jpg (690x540)	1.443	0.0221	1.361

Table 3: Comparison of the time spent in the different part of the algorithm (in second)