Lecture Note for SIFT Feature Descriptor

Zetong Li
School of Computer Science and Engineering
Sun Yet-Sen University
lizt9@mail2.sysu.edu.cn

Abstract

SIFT feature descriptor, a local feature descriptor with scale invariance, was first published in ICCV in 1999 by David Lowe [4], and published in IJCV in 2004 after being refined by David Lowe[5]. It is widely used in the field of image processing. However, the implementation of SIFT is not as simple as its use. Its inherent mathematical principles, parameter settings and source code implementations have aroused my strong interest. This note will record the complete process of learning SIFT and using python to manually implement SIFT, including the algorithm flow and the related mathematical derivation. The detailed information of the difference between DoG and LoG in Gaussian pyramid and DoG pyramid, the method to compute the approximate derivative and the Hessian matrix in extrema detection and detailed fitting, the parabolic interpolation in the orientation assignment and the trilinear interpolation in the keypoint descriptor will also be mentioned, which are rarely explained elsewhere, but are the important processes in the source code implementation.

1. Introduction

In the field of image recognintion, the SIFT algorithm was very popular in the past. Even in today with deep learning as the mainstream research direction, SIFT still has its unique advantages, and some researchers are still researching and improving the SIFT algorithm, which shows that learning and implementing SIFT are still of great significance.

Formally speaking, SIFT, the Scale Invariant Feature Transform, is said to be inspired by the response of primate visual inferior temporal cortical neurons. It is a method for image feature generation, which transforms an image into lots of local feature vectors. The SIFT features are invariant to image translation, scaling and rotation, and are partially invariant to affine distorction, change in 3D viewpoint, addition of noise and change in illumination. When appied in object recognition, the SIFT features can help improve the robustness of recognition in the presence of clutter and occlusion, while maintaining real-time performance. In addition, the SIFT features also play a great role in solving 3D structure from images, stereo correspondence and motion tracking.

In Lowe's paper[5], the SIFT features are generated by using a cascade filtering approach, of which the major stages are as following:

1. **Scale-space extrema detection:** Search over scales and image locations and identify poential points by looking for locations of extremas in a difference-of-Gaussian function to ensure the scale invariance of keypoints.

2. **Keypoint localization:** Perform a detailed fitting of all candidate points to determine the accurate scales and image locations, for the purpose of ensuring the stability of the keypoints.

3. **Orientation assignment:** Assign one or more orientations to each keypoint based on the local image gradient. Making good use of the orientations can achieve rotation invariance of the features.

4. **Keypoint descriptor:** Obtain feature vectors by weighted statistics of local image gradients at the selected scale in the region around each keypoint. And these feature vectors are the final SIFT features.

However, the major stages of my implementation of SIFT are slightly different from the above. In my implementation, it is more natural to combine scale-space extrema detection and keypoint localization with detailed fitting in the second step, and that the first step is to build a Gaussian pyramid and a difference-of-Gaussian pyramid.

Since Lowe rarely mentioned the details of SIFT implementation, my implementation refers to many other people's study notes and the implementation of SIFT in OpenCV. However, the correctness of my source code has not been fully verified, so my lecture note and source code can only provide a little immature reference for those who also want to learn more about SIFT.

2. Gaussian pyramid and DoG Pyramid

Scale space was first proposed by lijima in 1962, and then popularized by Witkin et al., which has attracted wide attention and use. It is a formal theory for handling image structures at different scales, by representing an image as a one-parameter family of smoothed images, parametrized by the size of the smoothing kernel used for suppressing fine-scale structures. In other words, a parameter that is regarded as scale is introduced into an image, and a family of images at multiple scales is obtained by continuously changing the scale parameter. It has been proved by Koenderink in 1984[2] and Lindeberg in 1994[3] that under some general assumptions the only possible kernel that is a linear kernel and implements scale transform is the Gaussian function.

Scale space theory makes it possible to find the keypoints of an image with scale invariance. For an image $I(r, c)$, its scale space $L(r, c, \sigma)$ can be obtained by the convolution of the variable-scale Gaussian kernel $G(r, c, \sigma)$ and the image:

$$L(r, c, \sigma) = G(r, c, \sigma) * I(r, c),$$

where

$$G(r, c, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{r^2+c^2}{2\sigma^2}}.$$

Candidate keypoints we are interested in are selected from the locations of extremas in difference-of-Gaussian function convolved with the image $D(r, c, \sigma)$:

$$\begin{aligned} D(r, c, \sigma) &= (G(r, c, k\sigma) - G(r, c, \sigma)) * I(r, c) \\ &= L(r, c, k\sigma) - L(r, c, \sigma). \end{aligned}$$

The reason why finding the extrema locations of $D(r, c, \sigma)$ as candidate points is effective is that the DoG function is a close approximation to the scale-normalized Laplacian of Gaussian $\sigma^2\nabla^2 G$, which has been shown to be required for true scale invariance by Lindeberg in 1994[3]. Further more, Mikolajczyk in 2002[6] found that the extremas of $\sigma^2\nabla^2 G$ produce the most stable image features compared to many other image functions. Note that

$$\frac{\partial G}{\partial \sigma} = \frac{r^2 + c^2 - 2\sigma^2}{2\pi\sigma^5} e^{-\frac{r^2+c^2}{2\sigma^2}},$$

$$\begin{aligned} \sigma\nabla^2 G &= \sigma\left(\frac{\partial^2 G}{\partial r^2} + \frac{\partial^2 G}{\partial c^2}\right) \\ &= \sigma \frac{r^2 + c^2 - \sigma^2}{2\pi\sigma^6} e^{-\frac{r^2+c^2}{2\sigma^2}} \\ &= \frac{\partial G}{\partial \sigma} \\ &\approx \frac{G(r, c, k\sigma) - G(r, c, \sigma)}{k\sigma - \sigma}. \end{aligned}$$

thus

$$\frac{1}{k-1}[G(r, c, k\sigma) - G(r, c, \sigma)] \approx \sigma^2\nabla^2 G.$$

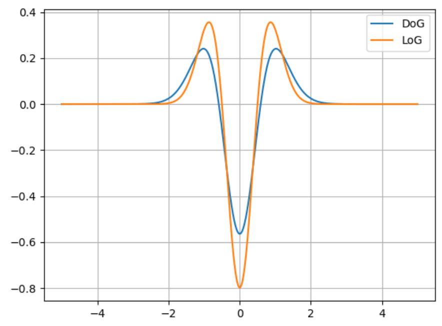

Figure 1. The approximation of DoG with factor $\frac{1}{k-1}$ to LoG in one-dimension case, where $\sigma = 0.5$, $k = \sqrt{2}$. It is obvious that the factor has no influence on the extrema locations.

When k is close enough to 1, $\frac{1}{k-1}[G(r, c, k\sigma) - G(r, c, \sigma)]$ will be also close enough to $\sigma^2\nabla^2 G$. Figure 1 shows the approximation of DoG with factor $\frac{1}{k-1}$ to LoG in one-dimension case. Obviously, the extrema location of DoG will remain the same after omitting the factor $\frac{1}{k-1}$, which means that it is quite reasonable to approximate LoG with DoG beacuse we only concerned with its extrema loacations. What is even more surprising is that, while it is possible to provide scale invariance, DoG convolved with the image can be efficiently computed by simply subtracting the corresponing smoothed images.

However, in actual implementation, building Gaussian pyramid is not simply computed according to the formula of $L(r, c, \sigma)$. Gaussian pyramid is based on image pyramid which is a multi-scale representation of an image, and image pyramid of an image is a series of image collections arranged in a pyramid shape with gradually reduced resolution. The difference between Gaussian pyramid and image pyramid is that each octave in image pyramid contains only a down-sampled image while in Gaussian pyramid contains several intervals each of which corresponds to a smoothed image with a scale. For an particular input image $I(r, c)$, it need to be doubled the size using linear interpolation before being used to build Gaussian pyramid, for the purpose of avoiding discarding the highest spatial frequencies when pre-smoothed. In the first octave of Gaussian pyramid, the doubled image $I_2(r, c)$ as input is incrementally convolved with Gaussian function $G(r, c, \sigma)$ to generate $L(r, c, \sigma)$ separated by a factor k in scale space. In order to double the scale between two adjacent octaves, we divide each octave into s intervals, thus factor $k = 2^{\frac{1}{s}}$. But actually we should produce $s + 3$ intervals in each octave so that there are $s + 2$ intervals in DoG pyramid and the extrema detection is complete in scale space. For improving the efficiency of calculation, in each octave, each interval except the first one can be obtained by smoothing the previous interval. For

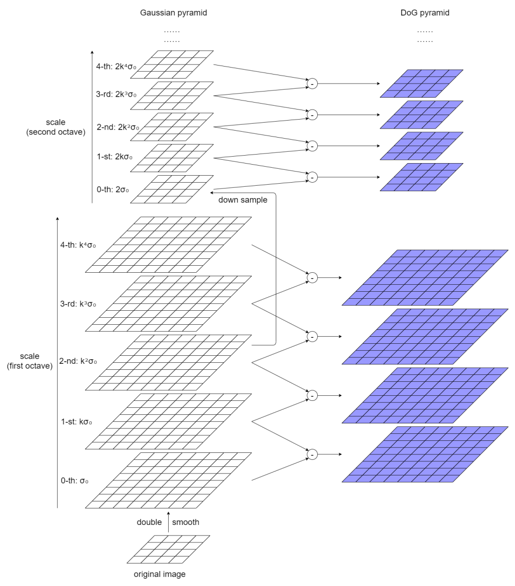

Figure 2. The complete building process of Gaussian pyramid and DoG pyramid, where $s = 2$, $k = 2^{\frac{1}{2}}$ as an example. It can be seen that the first interval of the second octave is obtained by down-sampling the s-th interval of the first octave, obviously $k^2\sigma = 2\sigma$.

each octave after the first octave, the first interval is obtained by down-sampling the s-th interval of the previous octave. Moreover, the 2D Gaussian function is separable, so it would be more efficient to use 1D Gaussian function in the horizontal and vertical directions. As for building DoG pyramid, we simply need to subtract adjacent images in Gaussian pyramid. The complete building process of Gaussian pyramid and DoG pyramid is shown in Figure 2.

The remaining problem is the settings of parameters s and σ, which involves the frequencies of sampling in scale and the spatial domain . Unfortunately, according to Lowe's paper, there is no minimum spacing of samples that will detect all extrema, as the extrema can be arbitrarily close. However, Lowe[5] pointed out through experiments that when $s = 3$, the highest repeatablity of keypoints is obtained, which means that we can detect subset of the most stable keypoints with coarse sampling of scales. In addition, in Lowe's paper[5], he assumed that the original image has a blur of at least $\sigma = 0.5$, thus the doubled image has $\sigma = 1.0$. After experimental comparison, Lowe chose to use $\sigma = 1.6$ as the initial scale in Gaussian pyramid.

Note that in implementation of Gaussian pyramid, each interval except the first one in each octave is computed according to the previous interval. Now assumed in the i-th octave, the j-th (j not equal to 0) interval has a blur $\sigma_{i,j} = 2^i k^j$, and it should be obtained as below:

$$L(r, c, \sigma_{i,j}) = G(r, c, \sqrt{\sigma_{i,j}^2 - \sigma_{i,j-1}^2}) * L(r, c, \sigma_{i,j-1}).$$

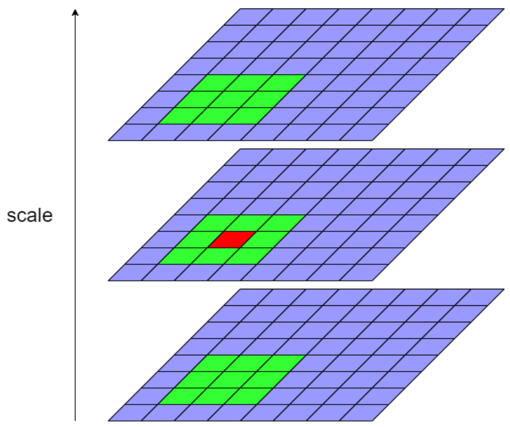

Figure 3. The process of extrema detection. Each sample point (red) need to compare to its 26 neighbors (green), and will be accepted as a candidate keypoint if it is larger or smaller than all of its neighbors.

Most people who reproduce SIFT did not pay attention to this computation method, and they may just compute:

$$L(r, c, \sigma_{i,j}) = G(r, c, \sigma_{i,j}) * L(r, c, \sigma_{i,j-1}),$$

which may be incorrect.

3. Extrema detection and detailed fitting

After obtaining DoG pyramid, it is easy to detect extremas (including maximas and minimas) through comparing each sample point to 8 neighbors in the current interval and 9 neighbors in the intervals above and below as shown in Figure 3.

It is worth noting that simply taking the locations obtained by the extrema detection as the keypoints does not guarantee good results, because there may be a little distance between the location of extrema in the discrete space and that in the continuous space, as shown in Figure 4. Therefore, it is needed to perform a detailed fitting to the locations of extremas. Fortunately, Brown and Lowe in 2002[1] developed a method for fitting a 3D quadratic function to the local sample point to determine the interpolated location of the extrema. To apply this method, assumed t as the offset from the sample point $x = (r, c, \sigma)$, we just need to use Taylor expansion of $D(r, c, \sigma)$, written as:

$$D(t) = D + \frac{\partial D}{\partial x}^T t + \frac{1}{2} t^T \frac{\partial^2 D}{\partial x^2} t,$$

where D, $\frac{\partial D}{\partial x}$ and $\frac{\partial^2 D}{\partial x^2}$ are computed at the sample point. The offset from the sample point to the interpolated location of the extrema can be derived by setting the derivative of

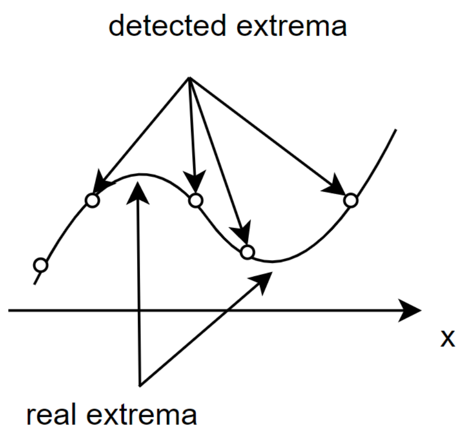

Figure 4. A simple example in which there is a little distance between the loacation of extrema in the discrete space and that in the continuous space.

$D(t)$ to zero, that is:

$$\frac{\partial D}{\partial x} + \frac{\partial^2 D}{\partial x^2} t = 0$$
$$\hat{t} = -\frac{\partial^2 D}{\partial x^2}^{-1} \frac{\partial D}{\partial x}.$$

There is no need to worry about the computation of the derivative and the Hessian matrix of D because they can be approximated by using differences of neighboring sample points, that is

$$\frac{\partial D}{\partial x} = \begin{bmatrix} D_r \\ D_c \\ D_s \end{bmatrix} \approx \begin{bmatrix} dr \\ dc \\ ds \end{bmatrix},$$
$$\frac{\partial^2 D}{\partial x^2} = \begin{bmatrix} D_{rr} & D_{rc} & D_{rs} \\ D_{cr} & D_{cc} & D_{cs} \\ D_{sr} & D_{sc} & D_{ss} \end{bmatrix} \approx \begin{bmatrix} drr & drc & drs \\ dcr & dcc & dcs \\ dsr & dsc & dss \end{bmatrix},$$

where

$$dr = \frac{1}{2}[L(r+1, c, \sigma) - L(r-1, c, \sigma)],$$
$$dc = \frac{1}{2}[L(r, c+1, \sigma) - L(r, c-1, \sigma)],$$
$$ds = \frac{1}{2}[L(r, c, k\sigma) - L(r, c, \frac{\sigma}{k})],$$

and

$$drr = L(r+1, c, \sigma) + L(r-1, c, \sigma) - 2L(r, c, \sigma),$$
$$dcc = L(r, c+1, \sigma) + L(r, c-1, \sigma) - 2L(r, c, \sigma),$$
$$dss = L(r, c, k\sigma) + L(r, c, \frac{\sigma}{k}) - 2L(r, c, \sigma),$$
$$drc = dcr = \frac{1}{4}[L(r+1, c+1, \sigma) - L(r-1, c+1, \sigma) - L(r+1, c-1, \sigma) + L(r-1, c-1, \sigma)],$$
$$drs = dsr = \frac{1}{4}[L(r+1, c, k\sigma) - L(r-1, c, k\sigma) - L(r+1, c, \frac{\sigma}{k}) + L(r-1, c, \frac{\sigma}{k})],$$
$$dcs = dsc = \frac{1}{4}[L(r, c+1, k\sigma) - L(r, c-1, k\sigma) - L(r, c+1, \frac{\sigma}{k}) + L(r, c-1, \frac{\sigma}{k})].$$

If the absolute value of $\hat{t}$ is larger than 0.5 in any dimension, that means the location of the extrema is closer to the adjacent sample point. Thus a new sample point is obtained by adding the offset to the sample point and the detailed fitting will be performed again until the absolute value of the offset is less than 0.5 in all dimensions. The final “new” sample point will replace the original one at the very beginning.

Once the slightly more accurate location of the extrema is obtained, we can use the value $D(\hat{t})$ to determine whether to reject it or not. In Lowe’s paper[5], all extrema with $|D(\hat{t})| < 0.03$ will be rejected, because points like this will have low contrast and thus be unstable. However, in actual implementation, the threshold 0.03 may be too strict for most cases. Therefore, in OpenCV, the contrast threshold is set to $\frac{0.04}{s}$ where s is the number of intervals ($s+2$ to be precise) in DoG pyramid.

In fact, some of the candidate keypoints are at the edge, which may not be a distinctive location for most cases. However, DoG function has a strong response along edges so we have to remove the candidate keypoints generated due to the edge response. It is said that points at the edge have large principals curvature across the edge while a small one in perpendicular direction, which can be seen by simply imagine a white line on a black background. It is convient to compute the Hessian matrix of D with respect to r and c, of which the eigenvalues are propotional to the principal curvatures of D. More fortunately, we are not concerned with the specific values of the eigenvalues but we just care about their ratio. Therefore, assumed the ratio η is larger than 1 (that is ratio of the larger eigenvalues α to the smaller one

β), then

$$H_{rc} = \begin{bmatrix} D_{rr} & D_{rc} \\ D_{cr} & D_{cc} \end{bmatrix} \approx \begin{bmatrix} drr & drc \\ dcr & dcc \end{bmatrix},$$
$$tr(H_{rc}) = D_{rr} + D_{cc} = \alpha + \beta \approx drr + dcc,$$
$$|H_{rc}| = D_{rr}D_{cc} - D_{rc}D_{cr} = \alpha\beta \approx drr\ dcc - drc\ dcr,$$
$$\frac{tr(H_{rc})^2}{|H_{rc}|} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(\eta+1)^2}{\eta}.$$

It is easy to prove that when $\eta > 1$, $\frac{(\eta+1)^2}{\eta}$ increases with η. Thus to set the threshold for η, we can simply set the threshold for $\frac{(\eta+1)^2}{\eta}$, that is $\frac{tr(H_{rc})}{|H_{rc}|}$, which can be computed efficiently. In Lowe's paper[5], he set the threshold for η to 10, which means that we require the keypoints satisfying $\frac{tr(H_{rc})}{|H_{rc}|} < \frac{(10+1)^2}{10}$.

Finally, the locations of the extremas that satisfying $D(\hat{t}) < 0.03\ or\ \frac{0.04}{s}$ and $\frac{tr(H_{rc})}{|H_{rc}|} < \frac{(10+1)^2}{10}$ will become the keypoints with strong stability and high repeatablity.

4. Orientation assignment

In order to achieve invariance to rotation, it is needed to assign one or more orientation for each keypoint based on the local image gradients. Assumed a particular pixel (r, c, σ) in an image, the magnitude $m(r, c, \sigma)$ and the orientation $\theta(r, c, \sigma)$ of the gradient can be computed as:

$$m(r, c, \sigma) = [(L(r+1, c, \sigma) - L(r-1, c, \sigma))^2 + (L(r, c+1, \sigma) - L(r, c-1, \sigma))^2]^{\frac{1}{2}},$$
$$\theta(r, c, \sigma) = arctan\frac{L(r, c+1, \sigma) - L(r, c-1, \sigma)}{L(r+1, c, \sigma) - L(r-1, c, \sigma)}.$$

It is reasonable to select one or more principal orientations for each keypoint according to the statistical results of the orientation histogram , which has 36 bins covering the 360 degree range. So, for a specific keypoint (r, c, σ), we are concerned with the gradients of the sample points within a region around the keypoint. Note that the size of the region is related to the scale of the keypoint within the octave to which the keypoint belongs. Thus, Lowe[5] advised to add the gradient of each sample point to the histogram weighted by the magnitude of the gradient multipled by a Gaussian weight with a σ that is 1.5 times the scale of the keypoint. To formally express the operation described here, we assume a sample point (r_s, c_s, σ) is around the keypoint, then the sample point will contribute w_s to the bin it belongs to, where

$$w_s = G(r_s - r, c_s - c, 1.5\sigma)m(r_s, c_s, \sigma)$$

The process is also clear in Figure 5. In actual implementation, according to the 3-sigma principle, using a Gaussian function with window size $2 \times 3 \times 1.5\sigma$ is enough for computing. Moreover, to avoid the influence of noise, the histogram should be smoothed before peak selection. Using a 1D Gaussian function with its window as $(1, 4, 6, 4, 1)$, we get:

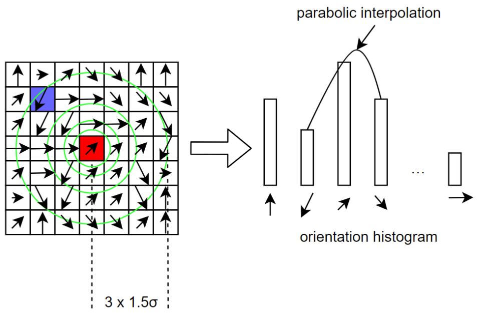

Figure 5. The process of statistically analyzing the gradients of the sample points around the keypoint. The green circles represent the corresponding Gaussian weights. The red one is the keypoint, while the blue one is a sample for example.

$$B(i) = \frac{b(i-2) + b(i+2)}{16} + \frac{4[b(i-1) + b(i+1)]}{16} + \frac{6b(i)}{16}$$

where $b(i)$ is the i-th bin of the histogram and $B(i)$ is the i-th bin in the smoothed histogram.

Noting that some keypoints may have several orientations with very large and close weights. It would be more fair to assign several orientations to a keypoint (or equivalently to create several keypoints with the same locations but different orientations) when there are local peaks which are within 80% of the highest peak in the smoothed orientation histogram, as suggested by Lowe[5].

Finally, for better accuracy, Lowe[5] pointed out that the parabolic interpolation is needed to fit the 3 histogram values closest to each peak to obtaine the interpolated peak position. Assumed the index of the peak is k and the value of the peak is $B(k)$, then to perform the parabolic interpolation means that points $(k-1, B(k-1))$, $(k, B(k))$ and $(k+1, B(k+1))$ are considered to be on the same parabola. What we want to do is to obtaine this parabola and determine its extrema as the accurate peak. Note that if $B(x) = ax^2 + bx + c$, then:

$$B(k-1) = a(k-1)^2 + b(k-1) + c,$$
$$B(k) = ak^2 + bk + c,$$
$$B(k+1) = a(k+1)^2 + b(k+1) + c,$$

gives

$$\begin{aligned}
a &= \frac{B(k-1)+B(k+1)}{2} - B(k), \\
b &= \frac{B(k-1)+B(k+1)}{2} \\
&\quad - k[B(k-1)+B(k+1)-2B(k)], \\
\hat{x} &= -\frac{b}{2a} = k + \frac{B(k-1)-B(k+1)}{2[B(k-1)+B(k+1)-2B(k)]}.
\end{aligned}$$

Therefore, the interpolated position of the peak with index k in the smoothed histogram can be obtained by simply adding $\frac{B(k-1)-B(k+1)}{2[B(k-1)+B(k+1)-2B(k)]}$ to k.

5. Keypoint descriptor

After the above steps, we have get enough information of the extracted keypoints including the locations, the scales and the orientations. With proper use, these keypoints will provide scale and rotation invariance. There is no doubt that computing a effective descriptor for each keypoint will bring great convience to actual use. So in Lowe's paper[5], to construct highly distinctive descriptors, the information such as the scale, the orientation and the local image region will be used.

According to Lowe[5], his method of constructing the descriptors is inspired by the model of biological vision and in particular of complex neurons in primary visual cortex. For a specific keypoint (r, c, σ), we will select the corresponding image in Gaussian pyramid as the object for operation. At first, we are supposed to determine the size of the region around the keypoint in which all sample points will contribute to the construction of the descriptor. Lowe[5] advised to divide the local image region into 4x4 subregions and to set a Gaussian function with σ equal to one half the width of the window of the descriptor for the purpose of weighting different sample points in different locations. However, Lowe did not mention the size of the subregions, which in OpenCV is set to 3σ where σ is the realitive scale of the keypoint within the octave it belongs to. Given that the sample points in the region should be rotated according to the orientation of the keypoint, as shown in Figure 6, it would be more approriate to set the size of the descriptor window to $(d+1) \times 3\sigma\sqrt{2}$ so that all the sample points in the square are included just as the case with no rotation, where d is the size of the subregions and $d = 4$ here. Also, in OpenCV, parameter σ of the Gaussian function is set to $\frac{d}{2}$, which is one half of the number of the subregions. It is slightly different from what Lowe advised, but setting in this way means that the sample points in the same subregion have the same contribution weights to the construction of the descriptor, which is also reasonable. In each subregion, the gradients of the sample points are computed, which are used to build the orientation histogram with 8 bins covering the 360 degree range. Note that it is the same as orientation assignment phase, the magnitude of the gradient multipled the corresponding Gaussian weight will be added to the bin to which the sample point belongs. It is worth noting that the rotation of the sample points can be achieved according to the following method:

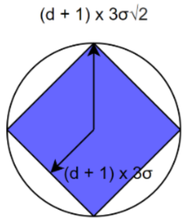

Figure 6. The descriptor window with size $(d+1) \times 3\sigma\sqrt{2}$. In this way, the sample points in the square will definitely contribute to the construction of the descriptor regardless of the orientation of the keypoint.

$$\begin{bmatrix} r' \\ c' \end{bmatrix} = \begin{bmatrix} cos\ \theta & -sin\ \theta \\ sin\ \theta & cos\ \theta \end{bmatrix} \begin{bmatrix} r \\ c \end{bmatrix},$$

where (r, c) is the original point and (r', c') is the point after rotated. The rotated points will be used in computing the corresponding Gaussian weight, and have no influence on the orientation of the original sample point, which however need to be rotated according to the orientation of the keypoint.

According to Lowe[5], when the descriptor changes slightly from one bin to another or from one histogram to another, the boundary affects will occur. So it is necessary to perform trilinear interpolation to distribute the contribution of each sample point into adjacent bins. In detail, this process is shown in Figure 7. Take the red point as an example. The value of that point will be distributed to 8 bins in total, which can be seen clearly in the lower left picture. Let us look at the detailed weights with which the red point distribute its value to its neighbors in the lower right picture in combination with Figure 8. Assumed the red point has value v, then the cubes numbered from 1 to 8 will get some of the value respectively:

$$\begin{aligned}
v_1 &= (1-r_t)(1-c_t)(1-\sigma_t)v, \\
v_2 &= r_t(1-c_t)(1-\sigma_t)v, \\
v_3 &= (1-r_t)c_t(1-\sigma_t)v, \\
v_4 &= r_t c_t(1-\sigma_t)v, \\
v_5 &= (1-r_t)(1-c_t)\sigma_t v, \\
v_6 &= r_t(1-c_t)\sigma_t v, \\
v_7 &= (1-r_t)c_t\sigma_t v, \\
v_8 &= r_t c_t \sigma_t v.
\end{aligned}$$

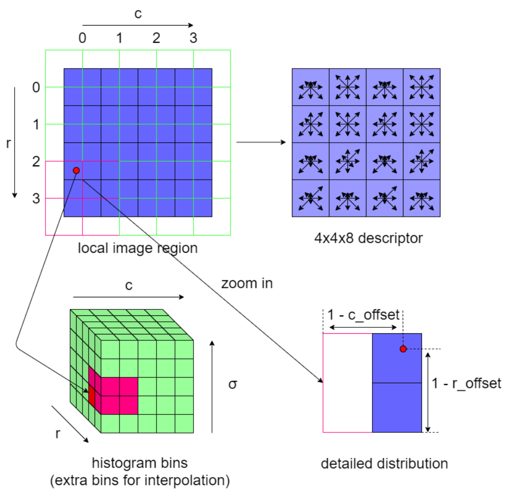

Figure 7. The process of trilinear interpolation on the histogram bins. The blue squares in the upper left are the local image region, while squares with green border are the histogram bins in 2D with extra bins for interpolation. Note that the red point in row 2 column 0 will distribute its value to squares with pink border. The cube in the lower left is the 3D representation of the histogram bins including variable σ. The picture in lower right shows the detailed weights with which the red point distribute its value to its neighbors in 2D. The picture in upper right is an example of 4x4x8 descriptor.

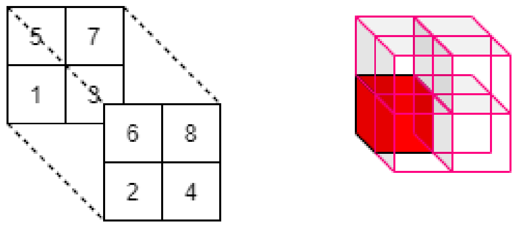

Figure 8. A picture that describes a simple way to number the corresponding cubes in Figure 7.

where r_t, c_t and σ_t are the offsets of r, c and σ respectively. After trilinear interpolation, we can get the 4x4x8 feature vectors of the keypoints like the upper right picture in Figure 7.

Finally, in order to improve the partially invariance of illumination change, the feature vectors still need to be slightly modified. In fact, there are several kinds of change of illumination. For example, change in contrast is to multiply a constant to the image pixels, which will multiply the gradients. It is easy to solve this problem by normalizing the feature vectors. Another example, change in brightness is to add a constant to all of the image pixels, which has no influence on gradients (we approximated it using pixel subtraction), thus the keypoint descriptors provide invariance of affine change in illumination. However, according to Lowe[5], non-linear illumination change does have a great impact on the magnitudes of the gradients, while little on the orientations. In Lowe's paper[5], he thresholded the values in all dimension of the normalized feature vectors to be smaller than 0.2 and this parameter was obtained by Lowe[5] through experiments. In this way, the descriptors will pay more attention to the orientation of the gradients and less to the magnitudes that are extremely large. After thresholding the magnitude, renormalize the feature vectors and we will obtained the final descriptors.

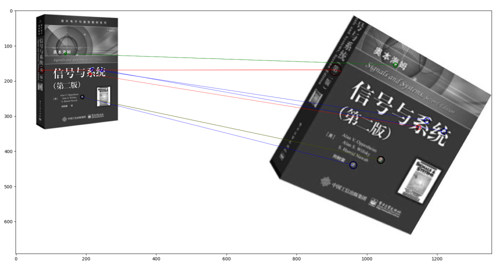

Figure 9. The matching result between two images with scaling and rotation.

6. Experiment

By referring to the parameter settings in Lowe's paper[5] and the concrete implementation of SIFT in OpenCV, I reproduce the SIFT algorithm using python.

The program I wrote requires two images to be matched as input, then it will use the SIFT algorithm to computed the SIFT features of each images and try to match the poential corresponding feature vectors (or say the corresponding keypoints) using K-nearest neighbors algorithm. The program will output a picture showing the top 10 similar keypoints with their scales represented by the size of circles and orientations represented by a straight line from the keypoint.

Figure 9 shows the matching result when applying scale and rotation to the images. Note that the resolution of the left picture from the Internet is 350x350, while that of the right one which is obtained by enlarging the left picture and rotate it 30 degrees clockwise is 678x692. The matching result looks pretty great except for a keypoint mismatched. The Euclidean distance between the matching points are [0.01320736 0.01756348 0.02107112 0.02970199 0.03262635 0.03483856 0.03536116 0.03643456 0.03862151 0.0429815], which means that matching points in top 10 are close enough.

Figure 10 shows the matching result when applying change in 3D viewpoint and change in illumination. Note that the resolution of the left picture from the Internet is the same as above, while that of the right one which is a photo actually taken with a mobile phone is 577x433. The matching result is amazing with all points correctly

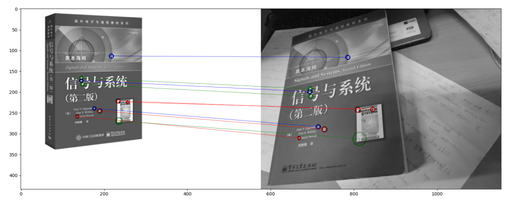

Figure 10. The matching result between two images with change in 3D viewpoint and change in illumination.

matched. The Euclidean distance between the matching points are [0.04425545 0.04444926 0.04460305 0.04462828 0.04583907 0.04636808 0.04834829 0.04872372 0.04974098 0.05026698], which is obviously larger than the result of the previous experiment. This is understandable because change in 3D viewpoint and change in illuminationare have a greater impact on the objects in the image than simple scaling and rotation.

However, one disadvantage of my reproducing results is that the running speed of my program is too slow. That is because in OpenCV, the SIFT algorithm is parallelized while in my implementation, it is implementated in the easiest but least efficient way.

Note that the source code and the three test pictures above can be found in `https://github.com/slippersss/SIFT`.

7. Related applications

In fact, the feature vectors, or the descriptors, generated by the SIFT algorithm play a grate role in many applications. The image features are borned to help match different images, which is a basis of object recognition. Due to the strong stability and highly distinctiveness, the SIFT features are very suitable for object recognition in cluttered real-world scenes. In Lowe's paper[5], he also proposed an approach to use the SIFT algorithm in combination with the Best-Bin-First algorithm and Hough transform to perform image matching and object identifying with pose verification. Even nowadays in the trend of deep learning, the SIFT features will still be used as the very effective artificial features.

8. Summary

This lecture note records the whole algorithm flow of SIFT, and the experiments of reproducing the SIFT algorithm and verifying some properties of the SIFT features. Reading Lowe's paper and the source code of the implementation of SIFT in OpenCV gives me a lot of new knowledge. Through manually implementing SIFT, I learned lots of details like the difference between DoG and LoG, the method to approximate the derivative and the Hessian matrix in an image, the process of performing parabolic interpolation and the trilinear interpolation. In addition, it is in this learning process that I know the concepts such as LoG, principal curvature , various interpolation methods and so on. All in all, I learned quiet a lot in the process of learning SIFT feature descriptor.

References

[1] M. Brown and D. Lowe. Invariant features from interest point groups. *Proc. BMVC*, pages 23.1–23.10, 2002. 3

[2] Jan J. Koenderink. The structure of images. *Biological Cybernetics*, 50(5):363–370, Aug 1984. 2

[3] T. Lindeberg. Scale-space theory : A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21:225–270, 1994. 2

[4] D.G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 2:1150–1157 vol.2, 1999. 1

[5] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004. 1, 3, 4, 5, 6, 7, 8

[6] Krystian Mikolajczyk. Detection of local features invariant to affine transformations. 01 2002. 2