

Local Image Descriptors

One can use raw RGB data of images directly for image matching or other applications. Unfortunately, those raw RGB data can vary significantly depending on various factors such as illumination changes. In general, we would like to identify robust features (e.g., points or image regions) that are invariant under various transformation including illumination and camera viewpoint changes. Once we identify such robust features, we can search similar images based on those features. Some of well known transformation include occlusion with other objects, articulation of parts of objects, intra-category variations, geometric transformation (e.g., projective transformation), and photometric transformations.

Image descriptors are classified into global or local ones. Global descriptors represent the overall information contained in the image, while local ones encode local information of the image. In this chapter, we mainly discuss local descriptors that are manually designed for achieving our goal of identifying robust features. The main reason why we talk about such manually designed ones is to understand various factors that are important for constructing robust features. These factors will be utilized for deep learning based features that will be discussed later. Also, note that techniques for computing local descriptors can be applied even to global descriptors.

Image descriptors should be repeatable under those various transformation and satisfy the following characteristics:

1. **Locality.** Features should be extracted by using local information for gaining robustness against occlusion and clutters of the environment.
2. **Quantity.** There should be a sufficient number of features to cover interesting regions of an object. By having many features, we can more robustly detect similar objects.

Identifying robust features under various changes is the key component of image search and many computer vision tasks.

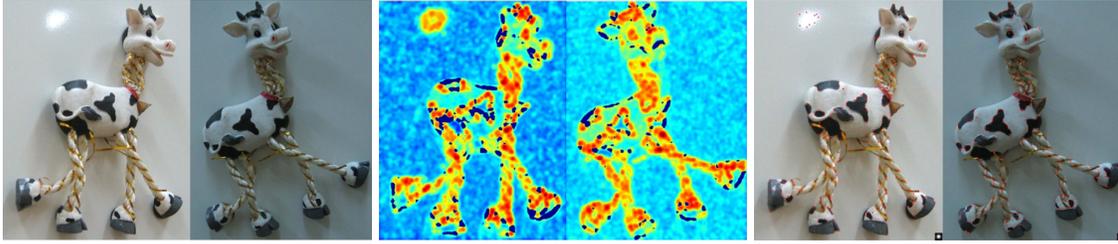


Figure 2.1: From left, input images, response values of the Harris corner detector, and computed corner points are shown. Courtesy of Frolova's slides.

3. **Efficiency.** These features should be extracted in a real-time for various applications.

Many techniques have been developed as candidates for those features, and serve as basic building blocks for various computer vision applications as well as image search.

2.1 Corners as Keypoints

One can easily recognize that corners are interesting and robust features that exist even after various transformations. Furthermore, when people are asked to identify corners of objects in an image, there is a high probability that those identified corners are covered by many different people. This intuitively suggests that corners can be repeatable and robust features that can be used for image matching. Fig. 2.1 shows detected corner points of two images that are rotated each other with different illumination levels. In this section, we discuss corners as useful keypoints of images.

Corners can be identified by looking at signal changes in the 2D image space among many other alternatives. In this section, we discuss the Harris corner detector¹ that detects corners reasonably well. The Harris corner detector is very popular thanks to its strong invariance to rotation and illumination changes.

To detect corners as features, we look at changes of intensity of an image along a shift, (u, v) , in X and Y directions. Its change, $E(u, v)$, is then defined as the following:

$$E(u, v) = \sum_{(x, y) \in P} w(x, y) (I(x + u, y + v) - I(x, y))^2, \quad (2.1)$$

where P indicates a set of all the possible image patches, each of which is associated with a weight map, $w(x, y)$. The weight can be set as a binary function (e.g, 1 within the window and 0 otherwise) or set as a Gaussian function. $E(u, v)$ is also known as the auto-correlation function.

The image function $I(x + u, y + v)$ within a small window size can be approximated as $I(x, y) + \nabla_I(x, y)(u, v)$ based on the Taylor

¹ Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50, 1988

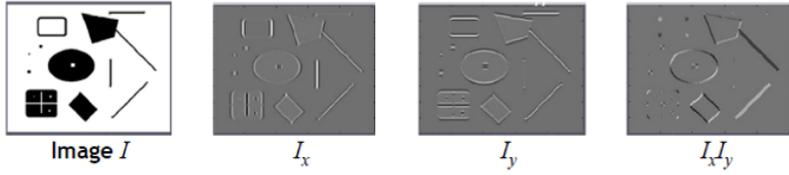


Figure 2.2: $I_x, I_y, I_x I_y$ are shown given an example image. Excerpted from Szeliski's slides.

expansion, where $\nabla I(x, y) = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$. After applying the Taylor expansion to Eq. 2.1, we get the following approximation in a matrix format:

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}, \quad (2.2)$$

where the matrix M is defined as:

$$M = \sum_{(x,y) \in P} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, \quad (2.3)$$

where I_x and I_y are partial gradients along X and Y image directions, respectively. The matrix M is also known as the second moment matrix.

What information does the matrix tell us? Different quantities of the matrix M given an image is shown in Fig. 2.2. For an axis-aligned corner, either one of I_x or I_y is computed as zero along their vertical and horizontal edges. As a result, M has the following values for the axis-aligned corner:

$$M = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}. \quad (2.4)$$

As a result, to detect corners, we look for regions that have large values for λ_1 and λ_2 . This procedure works for the axis-aligned corners, but how about rotated corners? For those rotated corners, we use the same matrix representation with a rotation matrix, R :

$$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R. \quad (2.5)$$

Still, the matrix of rotated corners has the same form to that of axis-aligned corners, when we take out the rotation part. As a result, analyzing this form with λ_1 and λ_2 is useful for identifying corners. In general, these λ_1 and λ_2 correspond to principal curvatures, i.e., eigenvalues, of image gradients.

Given this information, the following three cases are possible (Fig. 2.3):

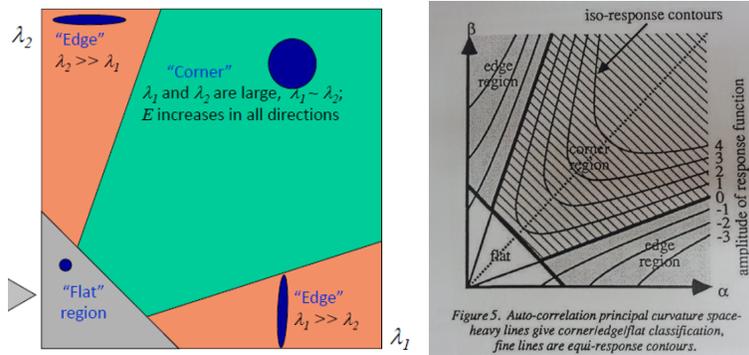


Figure 2.3: The left figure (Courtesy of Grauman's slides) shows different regions depending on eigenvalues of the second moment matrix, while the right one shows the response function of the Harris corner detector; Courtesy of the original harris corner detection paper.

1. **Edges.** Either one of eigenvalues are much bigger than the others. This indicates that the region in the window has an edge.
2. **Corners.** Both eigenvalues are big. As a result, any changes in this region drastically decrease the autocorrelation function. This indicates a corner in the region.
3. **Flat regions.** When both eigenvalues are small, this region is flat and does not cause much changes irrespective of any shift directions (u, v) .

Computing eigenvalues of the matrix M is expensive. Fortunately, the sum and product of eigenvalues can be efficiently computed based on the trace and determinant of the matrix M (Ch. 26):

$$\text{tr}(M) = \sum_i \lambda_i, \quad (2.6)$$

$$\text{det}(M) = \prod_i \lambda_i. \quad (2.7)$$

By utilizing this fact, we define the following response function, R , for identifying corners:

$$R = \text{det}(M) - \alpha \text{tr}(M)^2 = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2, \quad (2.8)$$

where α is shown empirically as 0.04 to 0.06. The response function R then has positive isovalues roughly corresponding to those corner regions (Fig. 2.3). On the other hand, the response function R gets to negative values on flat and edge regions. In other words, this response function is designed to behave like this. Another approach for designing alternative response function is a data-drive approach by using recent deep learning approaches. Note that the response function R is evaluated with a few arithmetic operations and thus it is efficient.

The next step is to compute corner points that have highest response values. We can pick a pixel with the highest response value,

Corner points are defined as pixels with locally maximum response values. This is also known as non-maximum suppression.

but it can detect only a corner. The common approach for detecting many corner points is to find pixels that have locally maximum response values. To do that, we typically introduce a local window and identify a pixel with the highest response value within that local window. This is also known as non-maximum suppression, i.e., filtering out pixels that do not have non-maximum values. In summary, given the response values over the image, we identify pixels with locally maximum values, and treat them as corner points. Identified corners in an example image is shown in Fig. 2.1.

The Harris corner detector is rotation-invariant, but it is not scale invariant, since depending on the resolution of a corner, it can be seen as a smooth curve at a zoom-in view.

2.1.1 Commonly Asked Questions

When we detect corners, we consider intensity of images. I think that color or other information of images should be also important. How can we consider them? The Harris corner detector looks at gradients of intensity of images. We can naively apply the same concept (gradients) to colors of images. However, there is a better way of considering colors or other information of images.

Why do we consider the auto-correlation equation, when we detect corners? We would like to detect key points that are robust to many different configurations for image matching. Since we would like to extract key points independently in each image, we look at the autocorrelation function. For the case of corners, the autocorrelation function can have big changes, when we have small shifts in a local window. In order to look at those changes in all the possible directions in an efficient manner, we apply the Taylor expansion with two orthogonal directions.

2.2 Scale-Invariant Region Detection

It is important to find features that are also invariant to scales. Additionally, a feature point itself does not provide ample information for defining a robust descriptor, which serves as a representation for image search or related applications. In general, by looking at a region, say 7 by 7 image pixels, around the feature point, we can collect useful information and construct robust features. To define such regions, we also need to know a scale or window size around the feature.

A naive approach for defining such a region in a scale invariant manner is to compute features with different scales, and use all

By looking at a region around a key-point and aggregating the information from the region, we can construct a useful image descriptor.

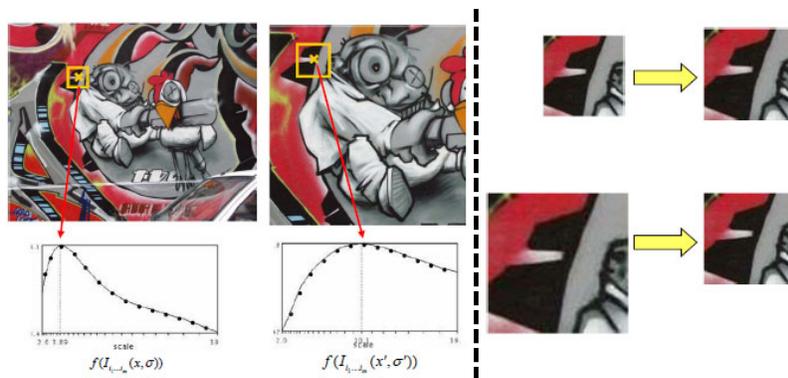


Figure 2.4: On the left, the same image with two different resolutions are shown. On the bottom, chosen characteristic scales, scales that have local maximum values according to a function, $f(\cdot)$, are shown; in this example, we use the average color intensity within each image region for the function. On the right, we remap those computed regions with the scales into a canonical image patch. The picture is excerpted from slides of Tyutelaars.

of them for later operations such as matching. Unfortunately, this approach is prohibitively expensive in terms of computation and memory requirement, especially for large-scale image databases.

To address this problem, many efforts have been put into. A general solution to this is to identify a *characteristic scale* of image regions by searching a local maximum in the scale space of the image ². At a high level, we build a series of lower resolutions of an input image as the scale space of the image. The X and Y spaces as well as the scale space of the image defines the 3D space of the image. We then detect the local maximum in the 3D space of the image including its scale space. The chosen scale of a pixel or its image region with the local maximum in the 3D space is called characteristic scale of the image region.

Note that as we used the local maximum values for detecting corners in the Harris corner detection (Ch. 2.1), we also use such pixels that have the local maximum values in the 3D space of the image. Finally, we re-map the original region with the chosen scale into a normalized path size (Fig. 2.4).

Convolution operation. For computing the scale-space of an input image, we perform convolution operation to the input image. The convolution operator is commonly used in many image processing. Thanks to its wide usage in various application, the convolution operator is also adopted in many deep learning architectures for processing images and videos. Therefore, we explain its concept here, before explaining the scale space of the image.

Given an image input $f(x, y)$ where x and y indicates pixel indices of the image, the convolution operation is defined as the following:

$$g(x, y) = w * f(x, y) = \sum_{dx=-a}^a \sum_{dy=-b}^b w(dx, dy) f(x + dx, y + dy), \quad (2.9)$$

² Tony Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):79–116, 1998

Operation	Filter Kernel w	Filter Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Approximated Gaussian blur (3 by 3)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Table 2.1: This table shows different operations by changing the filter kernel used in Eq. 2.9. This table is inspired by wiki and credits to the images are given to Michael Plotke.

where $w(x, y)$ is filter kernel with $2a$ by $2b$ kernel size, and $g(\cdot, \cdot)$ filtered image. By changing the filter kernel, we can implement various image processing operations. Table 2.1 shows examples of such operations.

In our context for computing the scale space of the image, the filter kernel is set with the Gaussian function like the one shown in Table 2.1. When we perform the convolution operation with the Gaussian function to an image, the filtered image looks blurry, as shown in Fig. 2.5.

Scale-space. For various operations, we build a scale space of an image for representing its multi-resolution structure. Given an image, I , we construct its lower resolution, $I(\sigma)$, which is computed by performing the convolution operator with the Gaussian function with a standard deviation, σ , as the filter weight to the input image I . By using wider σ values, we can successively build a series of lower resolutions of the image.

The Gaussian function is preferred to build the scale-space of the image, since it does not create any additional structures from the fine to the coarse scale. For example, a local minimum in a scale increases its value in its coarser scale and may not be a local minimum in that space. On the other hand, other points cannot become a local minimum in its coarser space.

Signature function for the scale. We need to use a function for identifying the characteristic scale in the scale-space of an image. Interestingly, neurophysical studies point out that visual response



Figure 2.5: This shows two Gaussian blurred images with different standard deviation values to the original image. Image credit to IkamusumeFan.

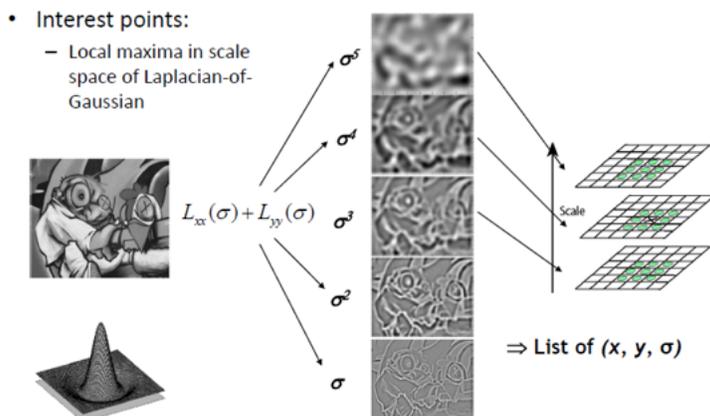


Figure 2.6: To compute the characteristic scale, we compute the scale-space of an image and then find the local maxima in the 3D scale-space consisting of the 2D image and the 1D scale spaces. This figure is excerpted from slides of Mikolajczyk.



Figure 2.7: Extracted blobs from a sunflower image. We use codes (Codes/blob_lapacian) available at <http://www.cs.utah.edu/~jfishbau/advimproc/project1/>. These codes are also available in the book homepage, <https://sgvr.kaist.ac.kr/~sungeui/search/>.

of many mammalian retina and visual cortex can be described by Gaussian derivatives. As a result, it is natural to consider Gaussian functions and their derivatives to identify the characteristic scale of an image. In particular, the Laplacian of Gaussian, also known as a blob detector, has been used for the purpose and identified to work well. The Laplacian of Gaussian can be denoted as $L_{xx}(\sigma) + L_{yy}(\sigma)$, where $L_{xx}(\sigma)$ and $L_{yy}(\sigma)$ indicate the second derivatives of the Gaussian function along X and Y directions.

Many differential functions including Laplacian and Harris functions have been tested for identifying the characteristic scale, but Laplacian shows the best result in an independent work ³ in terms of correctly detecting characteristic scales in images.

Laplacian-of-Gaussian (LoG) and its fast approximation. To compute the characteristic scale with LoG, we first compute the scale-space of the image with varying standard deviations for the Gaussian function, and find the local maxima in the 3D space consisting of 2D image space and the one dimensional scale space. A schematic view

³ K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *the International Conference on Computer Vision*, pages 525–531 vol.1, 2001

$$\text{LoG} = \sigma^2 (L_{xx}(x, y, \sigma) + L_{yy}(x, y, \sigma))$$

$$\text{DoG} = (G(x, y, k\sigma) - G(x, y, \sigma))$$

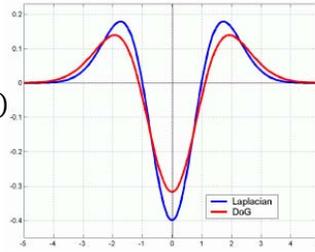


Figure 2.8: Laplacian-of-Gaussian (LoG) can be efficiently approximated by the Difference-of-Gaussian (DoG). k is a constant and when $k = \sqrt{2}$, there is not much difference between two functions. The figure is excerpted from slides of Leibe.

of computing the scale with LoG is shown in Fig. 2.6. Fig. 2.7 shows extracted blobs based on LoG that are local maxima of the 3D space from an input image.

Computing LoG for an image takes non-negligible time by evaluating LoG from the image. Interestingly, Difference-of-Gaussian (DoG), $G(x, y, k\sigma) - G(x, y, \sigma)$, shows a similar shape to LoG, as shown in Fig. 2.8. Since we need to compute the Gaussian filtered images for the scale-space, their difference can be computed efficiently. Thanks to this benefit, DoG is adopted for the well-known SIFT feature extraction (Ch. 2.3). We therefore use DoG for computing the characteristic scale instead of LoG.

Harris-Laplace. In Ch. 2.1, we discussed the Harris corner detection, which is invariant to many transformation, but not for scales. By adopting the characteristic scale in the Harris corner detector, we can also design scale-invariant Harris corner detection, known as Harris-Laplace ⁴, which uses LoG for computing the scale. Specifically, it computes corners, i.e., computes its response function, and perform non-local suppression to compute local minima. We also compute the local maxima in its 3D scale-space of the input image.

2.3 SIFT

So far, we talked about how to identify keypoints that are robust under various changes including different scales of images. Once we identify such keypoints, the next step is to represent the information on the keypoint pixel and its neighboring pixels as the descriptor. Again, this descriptor should be invariant under various changes. Once we have such robust descriptors, we can then match well between two descriptors that are extracted from the same point of two different images.

To achieve the aforementioned goal of constructing robust descriptors, many different techniques have been proposed. Among

⁴ K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *the International Conference on Computer Vision*, pages 525–531 vol.1, 2001

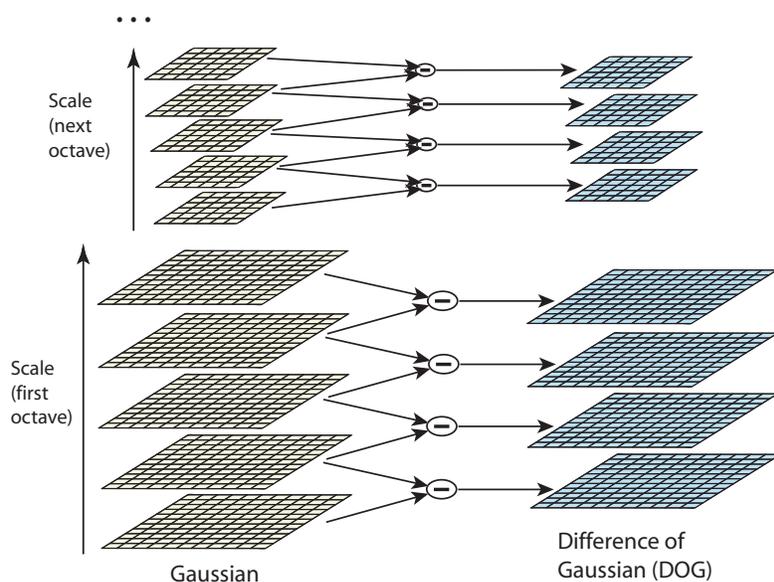


Figure 2.9: This figure illustrates how SIFT efficiently computes a scale-space of image by approximating LoG by DoG, and gradually reduces image resolutions. The figure is excerpted from the paper of Lowe.

many manually created image features, SIFT (Scale Invariant Feature Transform) is a seminal technique as a robust image feature for image matching and search ⁵. Overall, SIFT uses DoG for detecting the characteristic scale of features (Fig. 2.9).

Especially, as the scale-space representation, it uses a concept of octave for creating a scale-space efficiently. Note that identifying keypoints and computing descriptors should be performed efficiently, since users expect performing image search to be done in a few seconds. As shown in Fig. 2.9, we compute a scale-space of an input image that are filtered by Gaussian in the first octave, and compute a series of DoG by using two neighboring images in the scale space.

For the next octave, we can also do the same operation, but we reduce down the image resolution for lowering down the computational overhead of the future operations. Note that the image has smoothed out much and thus we can reduce down its resolution without losing much information. We then continue the process. This way we can accelerate the process of computing the scale space, while constructing high-quality image descriptors.

For identifying interest points, we look at 3 by 3 by 3 local window space in the scale space and find local maximum in that local window space (Ch. 2.2). This process is also shown in Fig. 2.10.

Those key points can include edge responses, since DoG responses to strong edges as well as blobs. For higher stability, we identify such edge cases and remove them. To detect such strong edge cases, we can use additional edge detector method, but can compute the

⁵ D.G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60 (2):91–110, 2004

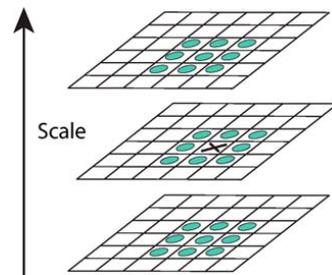


Figure 2.10: This shows a local region in the 3D space. We treat the center pixel (marked as X) to be a keypoint, once it has the local maximum value. The figure is excerpted from the paper of Lowe.

Pixels with local maxima can include edge responses. We thus need to filter those edge cases.

Hessian matrix that has the second derivatives of the Gaussian functions:

$$H = \sum_{(x,y) \in P} \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}, \quad (2.10)$$

where I_{xy} is the second partial derivatives along x followed by y directions. Note that the trace of the Hessian matrix is the Laplacian operator, and the second moment matrix (Eq. 2.3) used for the Harris corner detector is based on the first-order partial derivatives, while we use the second-order partial derivatives here.

Inspired by the response function of the Harris corner detector (Ch. 2.1), we consider its trace and determinant. In this case, we do not need to compute exact eigenvalues, but rather the ratio between them. Suppose that λ_1 and λ_2 are two eigenvalues of the Hessian matrix, H , while $\lambda_1 \geq \lambda_2$. Also, let r to be a ratio between them and thus $\lambda_1 = r\lambda_2$, where $r \geq 1$. We then have the following equation:

$$\frac{\text{Tr}(H)^2}{\text{Det}H} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1\lambda_2} = \frac{(r+1)^2}{r}. \quad (2.11)$$

This function has the minimum when $r = 1$, and increases as we have a larger ratio between two eigenvalues. In practice, we reject interest points whose ratio is bigger than 10.

Once we filter out strong edge cases, pixels with those left local maxima are treated as keypoints. At each keypoint, we look at a local window, say 8 by 8 pixels, in the image space. Our goal is to construct a robust image descriptor from pixels in the local window, also called image patch. As we designed a rotation-invariant keypoint, we also design a rotation-invariant image descriptor. As the first step for computing the rotation-invariant descriptor, we rotate the patch into a canonical orientation based on the dominant gradient of the patch; i.e., we rotate the patch such that its dominant gradient is aligned into a canonical orientation such as 90° . This process is also called orientation normalization.

For the normalized patch, we then divide them into 4 by 4 image patch and compute a histogram of orientations with eight bins for each sub-region. Note that the orientations are simply computed by image difference operations, and those computed gradients are much robust than using RGB values directly, especially for various illumination changes. As a result, SIFT use the orientation as the basic descriptor and use a histogram of orientations as a descriptor of each image region.

The aforementioned setting (e.g., 4 by 4 image sub-patches) was chosen to show the best result based on empirical tests (Fig. 2.11). The dimensionality of SIFT then has 128 ($= 4 \times 4 \times 8$). In practice, we

A patch containing the local maxima may have an arbitrary orientation. We thus normalize its orientation into a canonical one.

Gradients of images are more robust, i.e., does not vary much, than RGB values for various illumination changes.

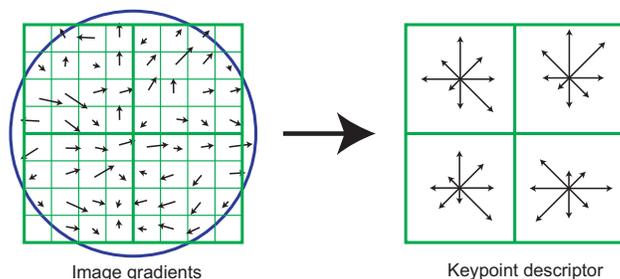


Figure 2.11: The right shows image gradients on a local image patch, which are weighted by a Gaussian function shown in the circle. By aggregating image gradients on a 4 by 4 image patch into a gradient histogram consisting of 8 bins shown on the left, we compute a SIFT image descriptor. The image is excerpted from the paper of Lowe.

get about two thousands of stable interest points for 500×500 image resolutions. This number of features is very important for identifying objects with severe transformations (e.g., occlusions) on images.

2.3.1 Commonly Asked Questions

Why Laplacian-of-Gaussian (LoG) is better than the average intensity for the automatic scale detection? LoG has been known to be an excellent detector for blobs as well as edges. Also, many Gaussian kernels have been known not to present any new information in coarser representations. Human visual responses can be modeled as Gaussian derivatives. Overall many studies suggest that LoG is one of the best functions that robustly detect the characteristic scale. For example, if an image has illumination change, the function of average intensity may not find good characteristic scales for images that shows the same object.

What are the differences between covariance and invariance? Invariant features are ones that do not change even if there are illumination changes and so on. On the other hand, covariant features increases or decreases depending on other factors. We would like to design invariant or at least covariant features for various types of image retrieval.

How can we eliminate the edge responses from the LoG function used in SIFT? We discussed how to detect edges and corners from the second moment matrix constructed by the first derivatives. Since the LoG function used the second derivatives, we compute Hessian matrix and do a similar process on the matrix to identify edge responses, as we did for the second moment matrix used for the Harris detector.