

Chapter 13

Image preprocessing and feature extraction

This chapter provides a brief overview of modern preprocessing methods for computer vision. In section 13.1, we introduce methods in which we replace each pixel in the image with a new value. Section 13.2 considers the problem of finding and characterizing edges, corners and interest points in images. In section 13.3 we discuss visual descriptors; these are low dimensional vectors that attempt to characterize the interesting aspects of an image region in a compact way. Finally, in section 13.4, we discuss methods for dimensionality reduction.

13.1 Per-pixel transformations

We start our discussion of preprocessing with *per-pixel operations*: these methods return a single value corresponding to each pixel of the input image. We denote the original 2D array of pixel data as \mathbf{P} , where p_{ij} is the element at the i^{th} of I rows and the j^{th} of J columns. The element p_{ij} is a scalar representing the grayscale intensity. Per-pixel operations return a new 2D array \mathbf{X} of the same size as \mathbf{P} , containing elements x_{ij} .

13.1.1 Whitening

The goal of whitening (figure 13.1) is to provide invariance to fluctuations in the mean intensity level and contrast of the image. Such variation may arise because of a change in ambient lighting intensity, the object reflectance, or the camera gain. To compensate for these factors, the image is transformed so that the resulting pixel values have zero mean and unit variance. To this end, we compute the mean μ and variance σ^2 of the original grayscale image \mathbf{P} :



Figure 13.1 Whitening and histogram equalization. a) A number of faces which have been captured with widely varying contrasts and mean levels b) After whitening the images have the same mean and variance. c) After histogram equalization the distribution of gray values is approximately uniform. Both of these transformations reduce the amount of variation due to contrast and intensity changes.

$$\begin{aligned}\mu &= \frac{\sum_{i=1}^I \sum_{j=1}^J p_{ij}}{IJ} \\ \sigma^2 &= \frac{\sum_{i=1}^I \sum_{j=1}^J (p_{ij} - \mu)^2}{IJ}.\end{aligned}\quad (13.1)$$

These statistics are used to transform each pixel value separately so that,

$$x_{ij} = \frac{p_{ij} - \mu}{\sigma}. \quad (13.2)$$

For color images, this operation may be carried out by computing the statistics μ and σ^2 from all three channels or by separately transforming each of the RGB channels based on their own statistics.

Note that even this simple transform has the potential to hamper subsequent inference about the scene: depending on the task, the absolute intensities may or may not contain critical information. Even the simplest preprocessing methods must be applied with care.

13.1.2 Histogram equalization

The goal of histogram equalization (figure 13.1c) is to modify the statistics of the intensity values so that *all* of their moments take predefined values. To this end, a nonlinear transformation is applied that forces the distribution of pixel intensities to be flat.

We first compute the histogram of the original intensities \mathbf{h} where the k^{th} of K entries is given by

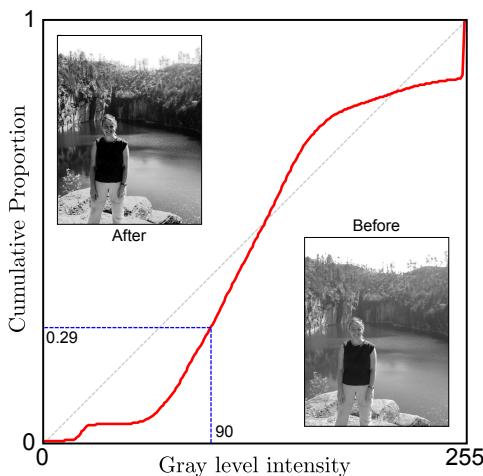


Figure 13.2 Histogram equalization. The abscissa indicates the pixel intensity. The ordinate indicates the proportion of intensities that were less than or equal to this value. This plot can be used as a look up table for histogram equalizing the intensities. For a given intensity value on the abscissa, we choose the new intensity to be the maximum output intensity K times the value on the ordinate. After this transformation, the intensities are equally distributed. In the example image, many of the pixels are bright. Histogram equalization spreads these bright values out over a larger intensity range, and so has the effect of increasing the contrast in the brighter regions.

$$h_k = \sum_{i=1}^I \sum_{j=1}^J \delta[p_{ij} - k], \quad (13.3)$$

where the operation $\delta[\bullet]$ returns one if the argument is zero and zero otherwise. We then cumulatively sum this histogram and normalize by the total number of pixels to compute the cumulative proportion c of pixels that are less than or equal to each intensity level:

$$c_k = \frac{\sum_{l=1}^k h_l}{IJ}. \quad (13.4)$$

Finally, we use the cumulative histogram as a look up table to compute the transformed value so that

$$x_{ij} = K c_{p_{ij}}. \quad (13.5)$$

For example, in figure 13.2 the value 90 will be mapped to $K \times 0.29$ where K is the maximum intensity (usually 255). The result is a continuous number rather than a discretized pixel intensity, but is in the same range as the original data. The result can be rounded to the nearest integer if subsequent processing demands.

Problem 13.1

13.1.3 Linear filtering

After filtering an image, the new pixel value x_{ij} consists of a weighted sum of the intensities of pixels in the surrounding area of the original image \mathbf{P} . The weights are stored in a filter kernel \mathbf{F} , which has entries $f_{m,n}$, where $m \in \{-M \dots M\}$ and $n \in \{-N \dots N\}$.

More formally, when we apply a filter, we *convolve* the \mathbf{P} with the filter \mathbf{F} , where two-dimensional convolution is defined as

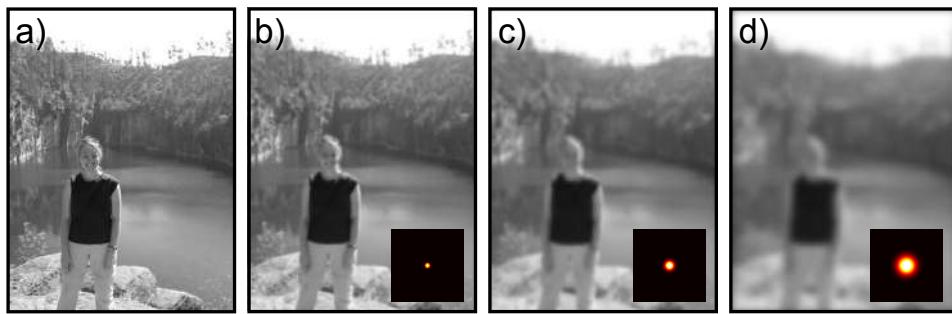


Figure 13.3 Image blurring. a) Original image. b) Result of convolving with a Gaussian filter (filter shown in bottom right of image). Each pixel in this image is a weighted sum of the surrounding pixels in the original image, where the weights are given by the filter. The result is that the image is slightly blurred. c-e) Convolving with a filter of increasing standard deviation causes the resulting image to be increasingly blurred.

Problem 13.2

$$x_{ij} = \sum_{m=-M}^M \sum_{n=-N}^N p_{i-m,j-n} f_{m,n}. \quad (13.6)$$

Notice that by convention, the filter is flipped in both directions so the top left of the filter $f_{-M,-N}$ weights the pixel $p_{i+M,j+N}$ to the right and below the current point in \mathbf{P} . Many filters used in vision are symmetric in such a way that this flipping makes no practical difference.

Without further modification, this formulation will run into problems near the borders of the image: it needs to access points that are outside the image. One way to deal with this is to use *zero padding* in which it is assumed that the value of P is 0 outside the defined image region.

We now consider a number of common types of filter.

Gaussian (blurring) filter

To blur an image, we convolve it with a 2D Gaussian,

$$f(m, n) = \frac{1}{2\pi\sigma^2} \exp \left[-\frac{m^2 + n^2}{2\sigma^2} \right]. \quad (13.7)$$

Problem 13.4

Each pixel in the resulting image is a weighted sum of the surrounding pixels, where the weights depend on the Gaussian profile: nearer pixels contribute relatively more to the final output. This process blurs the image, where the degree of blurring is dependent on the standard deviation σ of the Gaussian filter (figure 13.3). This is a simple method to reduce noise in images taken at very low light levels.

First derivative filters and edge filters

A second use for image filtering is to locate places in the image where the intensity changes abruptly. Consider taking the first derivative of the image along the rows. We could approximate this operation by simply computing the difference between two offset pixels along the row. This operation can be accomplished by filtering with the operator $\mathbf{F} = [-1 \ 0 \ 1]$. This filter gives zero response when the image is flat in the horizontal direction: it is hence invariant to constant additive luminance changes. It gives a negative response when the image pixel values are increasing as we move in the horizontal direction and a positive response when they are decreasing (recall that convolution flips the filter by 180°). As such, it is selective for edges in the image.

The response to the filter $\mathbf{F} = [-1 \ 0 \ 1]$ is noisy because of its limited spatial extent. Consequently, slightly more sophisticated filters are used to find edges in practice. Examples include the Prewitt operators (figures 13.4a-b)

$$\mathbf{F}_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, \quad \mathbf{F}_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \quad (13.8)$$

and the Sobel operators

$$\mathbf{F}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad \mathbf{F}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \quad (13.9)$$

where, in each case, the filter \mathbf{F}_x is a filter selective for edges in the horizontal direction and \mathbf{F}_y is a filter selective for edges in the vertical direction.

Laplacian filters

The Laplacian filter is the discrete two dimensional approximation to the Laplacian operator ∇^2 , and is given by

$$\mathbf{F} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \quad (13.10)$$

Applying the discretized filter \mathbf{F} to an image results in a response of high magnitude where the image is changing, regardless of the direction of that change (figure 13.4c): the response is zero in regions that are flat and significant where edges occur in the image. It is hence invariant to constant additive changes in luminance, and useful for identifying interesting regions of the image.

Laplacian of Gaussian filters

In practice, the Laplacian operator produces noisy results. A superior approach is to first smooth the image with a Gaussian filter and then apply the Laplacian. Due to the associative property of convolution, we can equivalently convolve the Laplacian filter by a Gaussian and apply the resulting *Laplacian of Gaussian* filter to the image (figure 13.4d). This Laplacian of Gaussian has the advantage that it

Problem 13.5
Problem 13.6

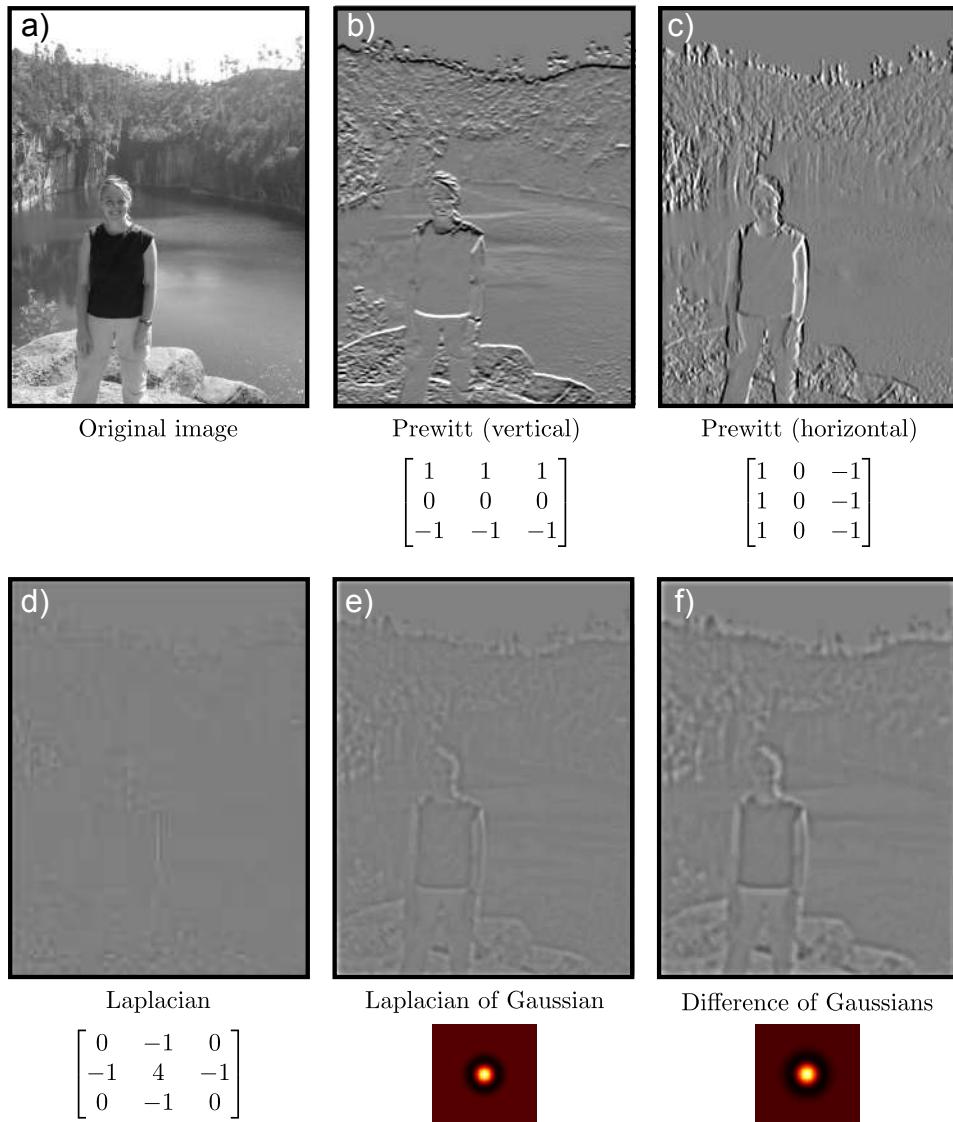


Figure 13.4 Image filtering with first and second derivative operators. The original image is shown in figure 13.3a. a) Convolving with the vertical Prewitt filter produces a response that is proportional to the size and polarity of edges in the vertical direction. b) The horizontal Prewitt filter produces a response to edges in the horizontal direction. c) The Laplacian filter gives a significant response where the image changes rapidly regardless of direction. d) The Laplacian of Gaussian filter produces similar results but the output is smoothed and hence less noisy. e) The difference of Gaussians filter is a common approximation to the Laplacian of Gaussian.

can be tuned to be selective for changes at different scales, depending on the scale of the Gaussian component.

Difference of Gaussians

The Laplacian of Gaussian filter is very well approximated by the difference of Gaussians filter (compare figures 13.4d and 13.4e). As the name implies, this filter is created by taking the difference of two Gaussians at nearby scales. The same result can be achieved by filtering the image with the two Gaussians separately and taking the difference between the results. Again, this filter responds strongly in regions of the image that are changing at a predetermined scale.

Gabor filters

Gabor filters are selective for both scale and orientation. The 2D Gabor function is the product of a 2D Gaussian with a 2D sinusoid. It is parameterized by the covariance of the Gaussian and the phase ϕ , orientation ω , and wavelength λ of the sine wave. If the Gaussian component is spherical, it is defined by

$$f_{mn} = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{m^2 + n^2}{2\sigma^2}\right] \sin\left[\frac{2\pi(\cos[\omega]m + \sin[\omega]n)}{\lambda} + \phi\right], \quad (13.11)$$

where σ controls the scale of the spherical Gaussian. It is typical to make the wavelength proportional to the scale σ of the Gaussian so a constant number of cycles is visible.

The Gabor filter is selective for elements within the image at a certain frequency and orientation band, and with a certain phase (figure 13.5). It is invariant to constant additive changes in luminance when the sinusoidal component is asymmetric. This is also nearly true for symmetric Gabor functions, as long as several cycles of the sinusoid are visible. A response that is independent of phase can easily be generated by squaring and summing the responses of two Gabor features with the same frequency, orientation, and scale, but with phases that are $\pi/2$ radians apart. The resulting quantity is termed the *Gabor energy* and is somewhat invariant to small displacements of the image.

Filtering with Gabor functions is motivated by mammalian visual perception: this is one of the first processing operations applied to visual data in the brain. Moreover, it is known from psychological studies that certain tasks (e.g., face detection) are predominantly dependent on information at intermediate frequencies. This may be because high-frequency filters see only a small image region and are hence noisy and relatively uninformative, and low-frequency filters act over a large region and respond disproportionately to slow changes due to lighting.

Haar-like filters

Haar-like filters consist of adjacent rectangular regions that are balanced so that the average filter value is zero, and they are invariant to constant luminance changes. Depending on the configuration of these regions, they may be similar to derivative or Gabor filters (figure 13.6).

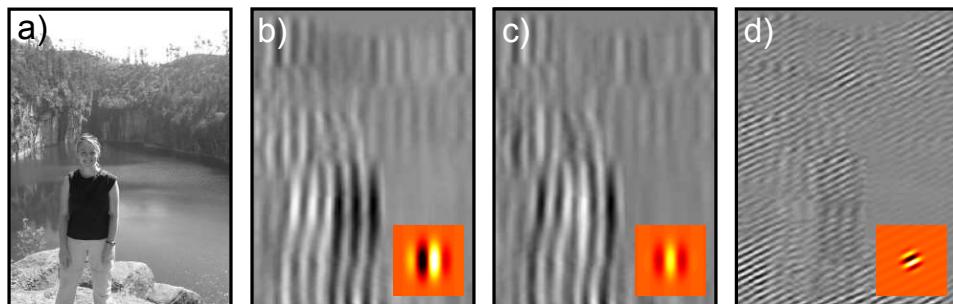


Figure 13.5 Filtering with Gabor functions. a) Original image. b) After filtering with horizontal asymmetric Gabor function at a large scale (filter shown bottom right). c) Result of filtering with horizontal symmetric Gabor function at a large scale. d) Filtering with vertical Gabor filter (responds to vertical changes). e) Response to diagonal Gabor function.

However, Haar-like filters are noisier than the filters they approximate: they have sharp edges between positive and negative regions, and so moving by a single pixel near an edge may change the response significantly. This drawback is compensated for by the relative speed with which Haar functions can be computed.

To compute Haar functions rapidly, we first form the *integral image* (figure 13.6g). This is an intermediate representation in which each pixel contains the sum of all of the intensity values above and to the left of the current position. So the value in the top-left corner is the original pixel value at that position and the value in the bottom right corner is the sum of all of the pixel values in the image. The values in the other parts of the integral image are between these extremes.

Given the integral image \mathbf{I} , it is possible to compute the sum of the intensities in any rectangular region with just four operations, regardless of how large this region is. Consider the region is defined by the range $[i_0, i_1]$ down the columns and $[j_0, j_1]$ along the rows. The sum S of the internal pixel intensities is

$$S = \mathbf{I}_{i_1, j_1} + \mathbf{I}_{i_0, j_0} - \mathbf{I}_{i_1, j_0} - \mathbf{I}_{i_0, j_1}. \quad (13.12)$$

The logic behind this calculation is illustrated in figure 13.6f-i.

Problem 13.3

Since Haar-like filters are composed of rectangular regions, they can be computed using a similar trick. For a filter with two adjacent rectangular regions, six operations are required. With three adjacent rectangular regions, eight operations are required. When the filter dimensions M and N are large, this approach compares very favorably to a naïve implementation of conventional filtering which requires $O(MN)$ operations to compute the filter response due to a $M \times N$ kernel. Haar-like filters are often used in real-time applications such as face detection because of the speed with which they can be computed.

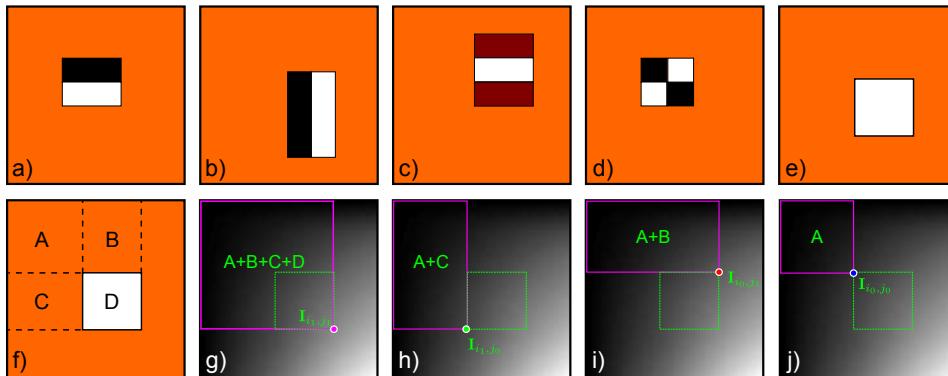


Figure 13.6 Haar-like filters. a-d) Haar-like filters consist of rectangular regions. Convolution with Haar-like filters can be done in constant time. e) To see why, consider the problem of filtering with this single rectangular region. f) We denote the sum of the pixel values in these four regions as A, B, C and D . Our goal is to compute D . g) The integral image has a value that is the sum of the intensities of the pixels above and to the left of the current position. The integral image at position (i_1, j_1) hence has value $A + B + C + D$. h) The integral image at (i_1, j_0) has value $A + C$. i) The integral image at (i_0, j_1) has value $A + B$. j) The integral image at (i_0, j_0) has value A . The sum of the pixels in region D can now be computed as $\mathbf{I}_{i_1, j_1} + \mathbf{I}_{i_0, j_0} - \mathbf{I}_{i_1, j_0} - \mathbf{I}_{i_0, j_1} = (A + B + C + D) + A - (A + C) - (A + B) = D$. This requires just four operations regardless of the size of the original square region.

13.1.4 Local binary patterns

The local binary patterns (LBP) operator returns a discrete value at each pixel that characterizes the local texture in a way that is partially invariant to luminance changes. For this reason, features based on local binary patterns are commonly used as a substrate for face recognition algorithms.

The basic LBP operator compares the eight neighboring pixel intensities to the center pixel intensity, assigning a 0 or a 1 to each neighbor depending on whether they are less than or greater than the center value. These binary values are then concatenated in a predetermined order and converted to a single decimal number that represents the ‘type’ of local image structure (figure 13.7).

With further processing, the LBP operator can be made orientation invariant: the binary representation is repeatedly subjected to bit-wise shifts to create eight new binary values, and the minimum of these values is chosen. This reduces the number of possible LBP values to 36. In practice, it has been found that the distribution over these 36 LBP values is dominated by those that are relatively *uniform*. In other words, binary strings where transitions are absent (e.g., 00000000, 11111111) or infrequent (e.g., 00001111, 00111111) occur most frequently. The number of texture classes can be further reduced by aggregating all of the non-uniform LPBs into a single class. Now the local image structure is categorized into

Problem 13.7

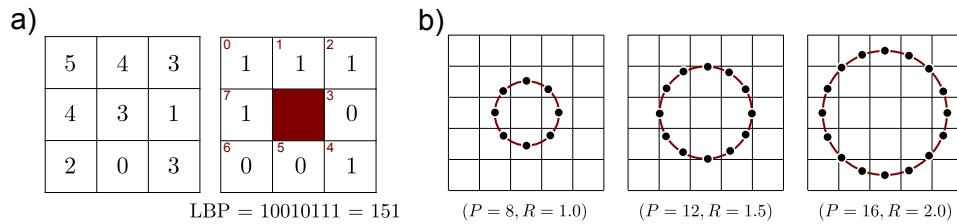


Figure 13.7 Local binary patterns. a) The local binary pattern (LBP) is computed by comparing the central pixel to each of its eight neighbors. The binary value associated with each position is set to one if that neighbor is greater than or equal to the central pixel. The eight binary values can be read out and combined to make a single 8-bit number. b) Local binary patterns can be computed over larger areas by comparing the current pixels to the (interpolated) image at positions on a circle. This type of LBP is characterized by the number of samples P and the radius of the circle R .

nine LBP types (eight rotationally invariant uniform patterns and one non uniform class).

The LBP operator can be extended to use neighborhoods of different sizes: the central pixel is compared to positions in a circular pattern (figure 13.7b). In general, these positions do not exactly coincide with the pixel grid, and the intensity at these positions must be estimated using bilinear interpolation. This extended LBP operator can capture texture at different scales in the image.

13.1.5 Texton maps

The term ‘texton’ stems from the study of human perception and refers to a primitive perceptual element of texture. In other words, it roughly occupies the role that a phoneme takes in speech recognition. In a machine vision context a texton is a discrete variable that designates which one of a finite number of possible texture classes is present in a region surrounding the current pixel. A texton map is an image in which the texton is computed at every pixel (figure 13.8).

Texton assignment depends on training data. A bank of N filters is convolved with a set of training images. The responses are concatenated to form one $N \times 1$ vector for each pixel position in each training image. These vectors are then clustered into K classes using the K-means algorithm (section 13.4.4). Textons are computed for a new image by convolving it with the same filter bank. For each pixel, the texton is assigned by noting which cluster mean is closest to the $N \times 1$ filter output vector associated with the current position.

The choice of filter bank seems to be relatively unimportant. One approach has been to use Gaussians at scales σ , 2σ , and 4σ to filter all three color channels, and derivatives of Gaussians at scales 2σ and 4σ and Laplacians of Gaussians at scales σ , 2σ , 4σ , and 8σ to filter the luminance (figure 13.9a). In this way, both color and

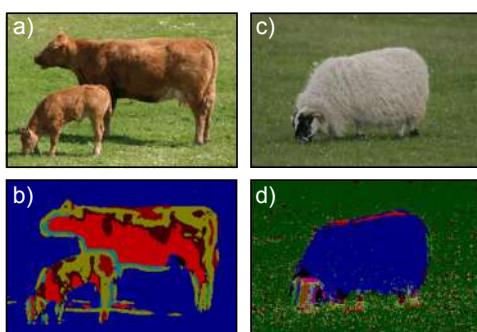


Figure 13.8 Texton maps. In a texton map each pixel is replaced by the texton index. This index characterizes the texture in the surrounding region. a) Original image. b) Associated texton map. Note how similar regions are assigned the same texton index (indicated by color). c) Original image. d) Associated texton map (using different filter bank from (b)). Texton maps are often used in semantic image segmentation. Adapted from Shotton *et al.* (2009). ©2009 Springer.

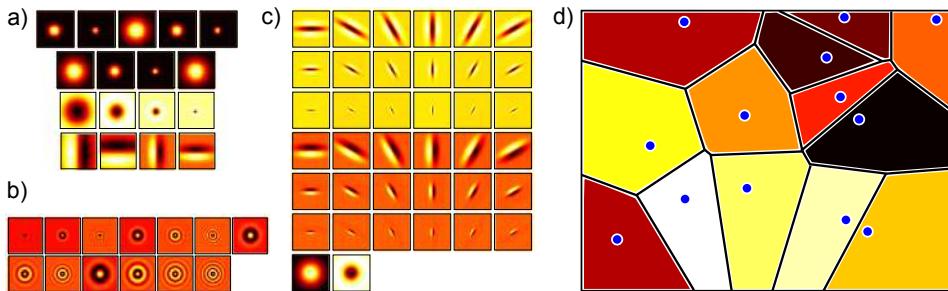


Figure 13.9 Textons. The image is convolved with a filter bank to yield an $N \times 1$ vector of filter responses at each position. Possible choices for the filter bank include a) a combination of Gaussians, derivatives of Gaussians, and Laplacians of Gaussians; b) rotationally invariant filters; and c) the maximum response (MR8) database. d) In training, the $N \times 1$ filter response vectors are clustered using K-means. For new data, the texton index is assigned based on the nearest of these clusters. Thus, the filter space is effectively partitioned into Voronoi regions.

texture information is captured.

It may be desirable to compute textons that are invariant to orientation. One way of achieving this is to choose rotationally invariant filters to form the filter bank (figure 13.9b). However, these have the undesirable property of not responding at all to oriented structures in the image. The Maximum Response (MR8) filter bank is designed to provide a rotationally invariant measure of local texture which does not discard this information. The MR8 filter bank (figure 13.9c) consists of a Gaussian and a Laplacian of Gaussian filter, an edge filter at three scales and a bar filter (a symmetric oriented filter) at the same three scales. The edge and bar filter are replicated at 6 orientations at each scale, giving a total of 38 filters. To induce rotational invariance only the *maximum* filter response over orientation is used. Hence the final vector of filter responses consists of eight numbers, corresponding to the Gaussian and Laplacian filters (already invariant) and the maximum responses over orientation of the edge and bar filters at each of the three scales.



Figure 13.10 Reconstruction from edges. a) Original image. b) Edge map. Each edge pixel has associated scale and orientation information as well as a record of the luminance levels at either side. c) The image can be reconstructed almost perfectly from the edges and their associated information. Adapted from Elder (1999). ©1999 Springer.

13.2 Edges, corners, and interest points

In this section, we consider methods that aim to identify informative parts of the image. In *edge detection* the goal is to return a binary image where a non-zero value denotes the presence of an edge in the image. Edge detectors optionally also return other information such as the orientation and scale associated with the edge. Edge maps are a highly compact representation of an image, and it has been shown that it is possible to reconstruct an image very accurately with just information about the edges in the scene (figure 13.10).

Corners are positions in the image that contain rich visual information and can be found reproducibly in different images of the same object (figure 13.12). There are many schemes to find corners, but they all aim to identify points that are locally unique. Corner detection algorithms were originally developed for geometric computer vision problems such as wide baseline image matching; here we see the same scene from two different angles and wish to identify which points correspond to which. In recent years, corners have also been used in object recognition algorithms (where they are usually referred to as *interest points*). The idea here is that the regions surrounding interest points contain information about which object class is present.

13.2.1 Canny edge detector

To compute edges with the Canny edge detector (figure 13.11), the image \mathbf{P} is first blurred and then convolved with a pair of orthogonal derivative filters such as Pre-witt filters to create images \mathbf{H} and \mathbf{V} containing derivatives in the horizontal and vertical directions, respectively. For pixel (i, j) , the orientation θ_{ij} and magnitude a_{ij} of the gradient is computed using

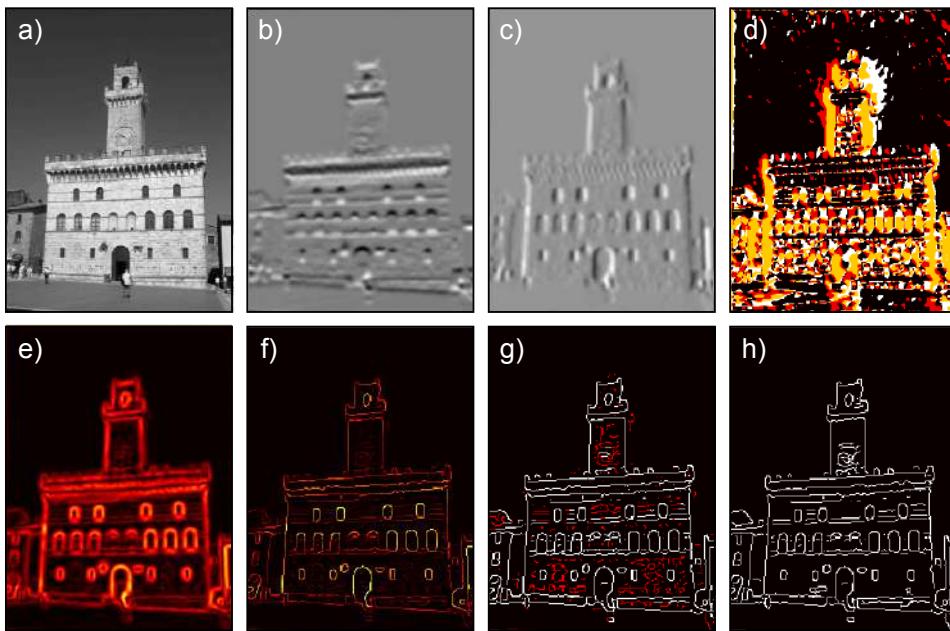


Figure 13.11 Canny edge detection. a) Original image. b) Result of vertical Prewitt filter. c) Results of horizontal Prewitt filter. d) Quantized orientation map. e) Gradient amplitude map. f) Amplitudes after non-maximal suppression. g) Thresholding at two levels: the white pixels are above the higher threshold. The red pixels are above the lower threshold but below the higher one. h) Final edge map after hysteresis thresholding contains all of the white pixels from (g) and those red pixels that connect to them.

$$\begin{aligned}\theta_{ij} &= \arctan[v_{ij}/h_{ij}] \\ a_{ij} &= \sqrt{h_{ij}^2 + v_{ij}^2}.\end{aligned}\quad (13.13)$$

A simple approach would be to assign an edge to position (i, j) if the amplitude there exceeds a critical value. This is termed *thresholding*. Unfortunately, it produces poor results: the amplitude map takes high values on the edge, but also at adjacent positions. The Canny edge detector eliminates these unwanted responses using a method known as *non-maximum suppression*.

In non-maximum suppression the gradient orientation is quantized into one of four angles $\{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$, where angles 180° apart are treated as equivalent. The pixels associated with each angle are now treated separately. For each pixel the amplitude is set to zero if either of the neighboring two pixels *perpendicular* to the gradient have higher values. For example, for a pixel where the gradient orientation is vertical (the image is changing in the horizontal direction), the pixels to the left and right are examined and the amplitude is set to zero if either of these

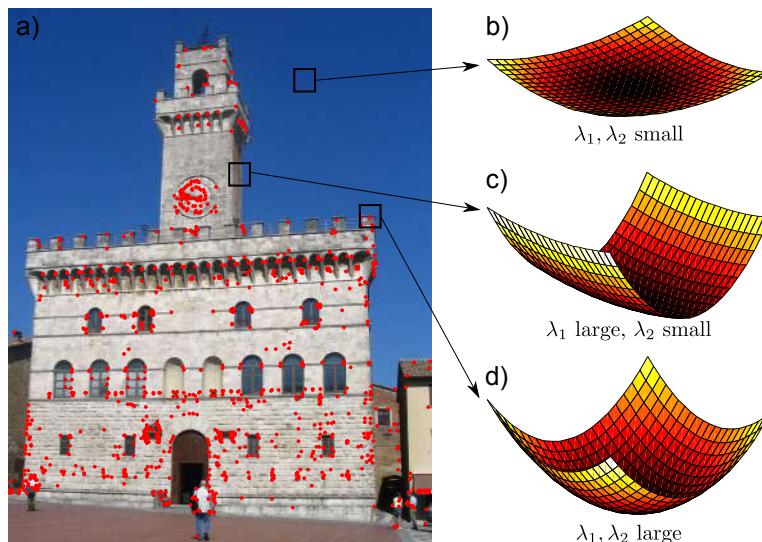


Figure 13.12 Harris corner detector. a) Image with detected corners. The corner detection algorithm is based on the image structure tensor, which captures information about the distribution of gradients around the point. b) In flat regions, both singular values of the image structure tensor are small. c) On edges, one is small and the other large. d) At corners, both are large indicating that the image is changing quickly in both directions.

are greater than the current value. In this way, the gradients at the maximum of the edge amplitude profile are retained, and those away from this maximum are suppressed.

A binary edge map can now be computed by comparing the remaining non-zero amplitudes to a fixed threshold. However, for any given threshold there will be misses (places where there are real edges, but their amplitude falls below the threshold) and false positives (pixels labeled as edges where none exist in the original image). To decrease these undesirable phenomena, knowledge about the continuity of real-world edges is exploited. Two thresholds are defined. All of the pixels whose amplitude is above the higher threshold are labeled as edges, and this threshold is chosen so that there are few false positives. To try to decrease the number of misses, pixels that are above the lower amplitude threshold *and* are connected to an existing edge pixel are also labeled as edges. By iterating this last step, it is possible to trace along weaker parts of strong contours. This technique is known as *hysteresis thresholding*.

13.2.2 Harris corner detector

The Harris corner detector (figure 13.12) considers the local gradients in the horizontal and vertical directions around each point. The goal is to find points in the image where the image intensity is varying in both directions (a corner) rather than in one direction (an edge), or neither (a flat region). The Harris corner detector bases this decision on the *image structure tensor*

$$\mathbf{S}_{ij} = \sum_{m=i-D}^{i+D} \sum_{n=j-D}^{j+D} w_{mn} \begin{bmatrix} h_{mn}^2 & h_{mn}v_{mn} \\ h_{mn}v_{mn} & v_{mn}^2 \end{bmatrix}, \quad (13.14)$$

where \mathbf{S}_{ij} is the image structure tensor at position (i, j) , which is computed over a square region of size $(2D+1) \times (2D+1)$ around the current position. The term h_{mn} denotes the response of a horizontal derivative filter (such as the Sobel) at position (m, n) and the term v_{mn} denotes the response of a vertical derivative filter. The term w_{mn} is a weight that diminishes the contribution of positions that are far from the central pixel (i, j) .

To identify whether a corner is present, the Harris corner detector considers the singular values λ_1, λ_2 of the image structure tensor. If both singular values are small, then the region around the point is smooth and this position is not chosen. If one singular value is large but the other small, then the image is changing in one direction but not the other, and point lies on or near an edge. However, if both singular values are large, then this image is changing rapidly in both directions in this region and the position is deemed to be a corner.

In fact the Harris detector does not directly compute the singular values, but evaluates a criterion which accomplishes the same thing more efficiently:

$$c_{ij} = \lambda_1 \lambda_2 - \kappa(\lambda_1^2 + \lambda_2^2) = \det[\mathbf{S}_{ij}] - \kappa \cdot \text{trace}[\mathbf{S}_{ij}], \quad (13.15)$$

where κ is a constant (values between 0.04 and 0.15 are sensible). If the value of c_{ij} is greater than a predetermined threshold, then a corner may be assigned. There is usually an additional non-maximal suppression stage similar to that in the Canny edge detector to ensure that only peaks in the function c_{ij} are retained.

13.2.3 SIFT detector

The scale invariant feature transform (SIFT) detector is a second method for identifying interest points. Unlike the Harris corner detector, it associates a scale and orientation to each of the resulting interest points. To find the interest points a number of operations are performed in turn.

The intensity image is filtered with a difference of Gaussian kernel at a series of K increasingly coarse scales (figure 13.13). Then the filtered images are stacked to make a 3D volume of size $I \times J \times K$ where I and J are the vertical and horizontal size of the image. *Extrema* are identified within this volume: these are positions where the 26 3D voxel neighbors (from a $3 \times 3 \times 3$ block) are either all greater than or all less than the current value.

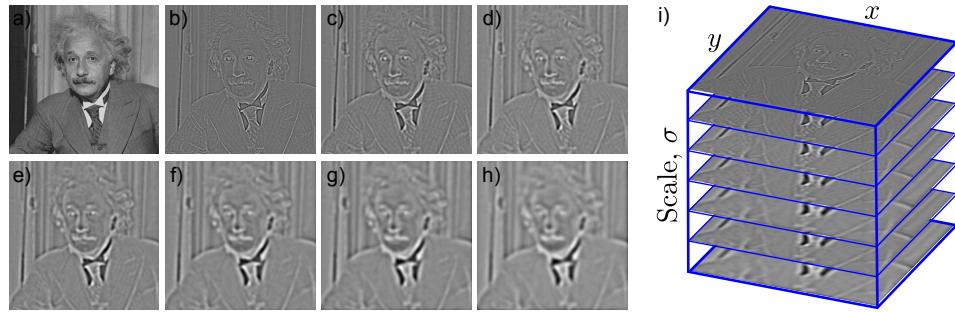


Figure 13.13 The SIFT detector. a) Original image. b-h) The image is filtered with difference of Gaussian kernels at a range of increasing scales. i) The resulting images are stacked to create a 3D volume. Points that are local extrema in the filtered image volume (i.e., are either greater than or less than all 26 3D neighbors) are considered to be candidates for interest points.

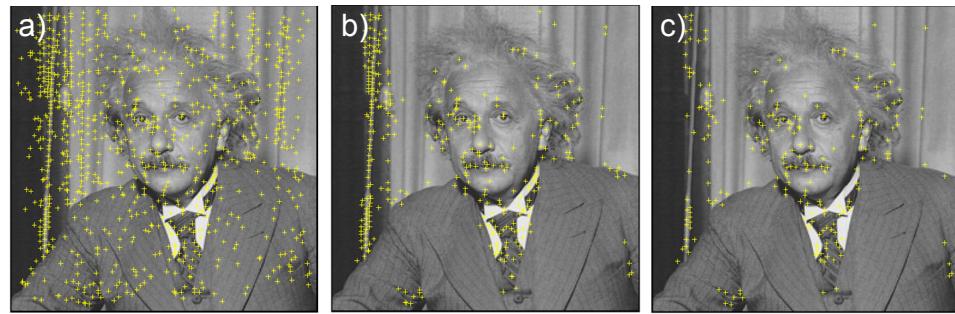


Figure 13.14 Refinement of SIFT detector candidates. a) Positions of extrema in the filtered image volume (figure 13.13i). Note that the scale is not shown. These are considered candidates to be interest points. b) Remaining candidates after eliminating those in smooth regions. c) Remaining candidate points after removing those on edges using the image structure tensor.

These extrema are localized to sub-voxel accuracy, by applying a local quadratic approximation and returning the position of the peak or trough. The quadratic approximation is made by taking a Taylor expansion about the current point. This provides a position estimate that has sub-pixel resolution and an estimate of the scale that is more accurate than the resolution of the scale sampling. Finally, the image structure tensor \mathbf{S}_{ij} (equation 13.14) is computed at the location and scale of each point. Candidate points in smooth regions and on edges are removed by considering the singular values of \mathbf{S}_{ij} as in the Harris corner detector (figure 13.14).

This procedure returns a set of interest points that are localized to sub-pixel

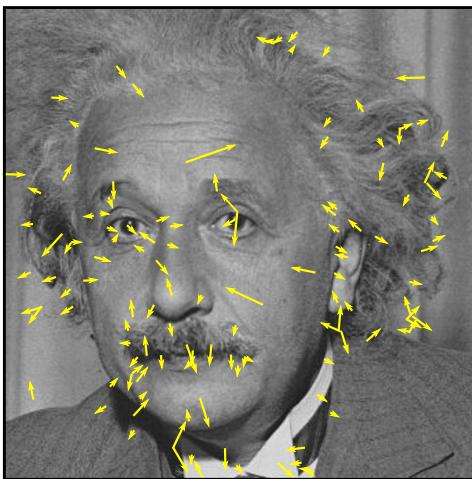


Figure 13.15 Results of SIFT detector. Each final interest point is indicated using an arrow. The length of the arrow indicates the scale with which the interest point is identified and the angle of the arrow indicates the associated orientation. Notice that there are some positions in the image where the orientation was not unique and here two interest points are used, one associated with each orientation. An example of this is on the right shirt collar. Subsequent descriptors that characterize the structure of the image around the interest points are computed relative to this scale and orientation and hence inherit some invariance to these factors.

accuracy and associated accurately with a particular scale. Finally, a unique orientation is also assigned to each interest point. To this end, the amplitude and orientation of the local gradients are computed (equations 13.13) in a region surrounding the interest point whose size is proportional to the identified scale. An orientation histogram is then computed over this region with 36 bins covering all 360° of orientation. The contribution to the histogram depends on the gradient amplitude and is weighted by a Gaussian profile centered at the location of the interest point, so that nearby regions contribute more. The orientation of the interest point is assigned to be the peak of this histogram. If there is a second peak within 80% of the maximum, we may choose to compute descriptors at two orientations at this point. The final detected points are hence associated with a particular orientation and scale (figure 13.15).

13.3 Descriptors

In this section, we consider *descriptors*. These are compact representations that summarize the contents of an image region.

13.3.1 Histograms

The simplest approach to aggregating information over a large image region is to compute a histogram of the responses in this area. For example, we might collate RGB pixel intensities, filter responses, local binary patterns, or textons into a histogram depending on the application. The histogram entries can be treated as discrete and modeled with a categorical distribution, or treated as a continuous vector quantity.

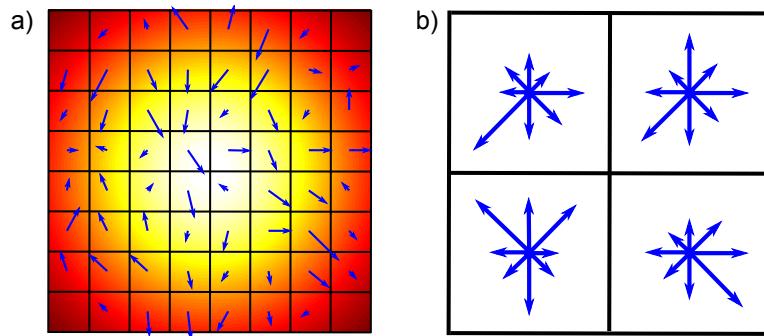


Figure 13.16 SIFT descriptor a) Gradients are computed for every pixel within a region around the interest point. b) This region is subdivided into cells. Information is pooled within these cells to form an 8D histogram. These histograms are concatenated to provide a final descriptor that pools locally to provide invariance to small deformations but also retains some spatial information about the image gradients. In this figure, information from an 8×8 pixel patch has been divided to make a 2×2 grid of cells. In the original implementation of the SIFT detector, a 16×16 patch was divided into a 4×4 grid of cells.

For continuous quantities such as filter responses, the level of quantization is critical. Quantizing the responses into many bins potentially allows fine discrimination between responses. However, if data are scarce then many of these bins will be empty, and it is harder to reliably determine the statistics of the descriptor. One approach is to use an adaptive clustering method such as K-means (section 13.4.4) to automatically determine the bin sizes and shapes.

Histogramming is a useful approach for tasks where spatial resolution is not paramount: for example, to classify a large region of texture, it makes sense to pool information. However, this approach is largely unsuitable for characterizing structured objects: the spatial layout of the object is important for identification. We now introduce two representations for image regions that retain some spatial information but also pool information locally and thus provide invariance to small displacements and warps of the image. Both the SIFT descriptor (section 13.3.2) and the HOG descriptor (section 13.3.3) concatenate several histograms that were computed over spatially distinct blocks.

13.3.2 SIFT descriptors

The scale invariant feature transform (SIFT) descriptor (figure 13.16) characterizes the image region around a given point. It is usually used in conjunction with interest points that were found using the SIFT detector. These interest points are associated with a particular scale and rotation, and the SIFT descriptor would typically be computed over a square region that is transformed by these values.

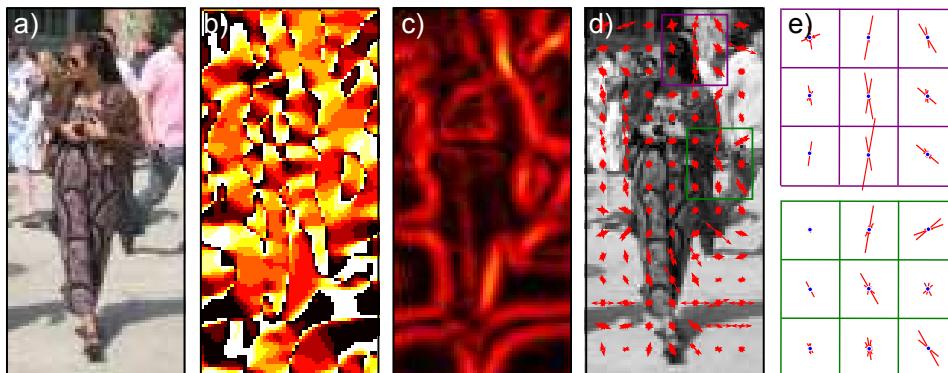


Figure 13.17 HOG descriptor. a) Original image. b) Gradient orientation, quantized into nine bins from 0 to 180° . c) Gradient magnitude. d) Cell descriptors are 9D orientation histograms that are computed within 6×6 pixel regions. e) Block descriptors are computed by concatenating 3×3 blocks of cell descriptors. The block descriptors are normalized. The final HOG descriptor consists of the concatenated block descriptors.

The goal is to characterize the image region in a way that is partially invariant to intensity and contrast changes and small geometric deformations.

To compute the SIFT descriptor, we first compute gradient orientation and amplitude maps (equation 13.13) as for the Canny edge detector over a 16×16 pixel region around the interest point. The resulting orientation is quantized into eight bins spread over the range $0 - 360^\circ$. Then the 16×16 detector region is divided into a regular grid of non-overlapping 4×4 cells. Within each of these cells an eight dimensional histogram of the image orientations is computed. Each contribution to the histogram is weighted by the associated gradient amplitude and by distance so that positions further from the interest point contribute less. The $4 \times 4 = 16$ histograms are concatenated to make a single 128×1 vector, which is then normalized.

The descriptor is invariant to constant intensity changes as it is based on gradients. The final normalization provides some invariance to contrast. Small deformations do not affect the descriptor too much as it pools information within each cell. However, by keeping the information from each cell separate, some spatial information is retained.

13.3.3 Histogram of oriented gradients

The Histogram of Oriented Gradients (HOG) descriptor attempts to construct a more detailed characterization of the spatial structure with a small image window. It is a useful preprocessing step for algorithms that detect objects with quasi-regular structure such as pedestrians. Like the SIFT descriptor, the HOG

descriptor consists of a collection of normalized histograms computed over spatially offset patches; the result is a descriptor that captures coarse spatial structure, but is invariant to small local deformations.

The process of computing a HOG descriptor suitable for pedestrian detection consists of the following stages. First, the orientation and amplitude of the image gradients are computed at every pixel in a 64×128 window using equation 13.13. The orientation is quantized into nine bins spread over the range $0 - 180^\circ$. The 64×128 detector region is divided into a regular grid of overlapping 6×6 cells. A 9D orientation histogram is computed within each cell, where the contribution to the histogram is weighted by the gradient amplitude and the distance from the center of the cell so that more central pixels contribute more. For each 3×3 block of cells, the descriptors are concatenated and normalized to form a block descriptor. All of the block descriptors are concatenated to form the final HOG descriptor.

The final descriptor contains spatially pooled information about local gradients (within each cell), but maintains some spatial resolution (as there are many cells). It creates invariance to contrast polarity by only using the gradient magnitudes. It creates invariance to local contrast strength by normalizing relative to each block. The HOG descriptor is similar in spirit to the SIFT descriptor but is distinguished by being invariant to contrast polarity, having a higher spatial resolution of computed histograms and performing normalization more locally.

13.3.4 Bag of words descriptor

The descriptors discussed thus far have been intended to characterize small regions of images. Often these regions have been connected to interest points. The bag of words representation attempts to characterize a larger region or an entire image by summarizing the statistics of the descriptors (e.g., SIFT) associated with all of the interest points in a region.

Each observed descriptor is considered to be one of a finite vocabulary of possible descriptors (termed *visual words*). Collectively, this vocabulary is known as a *dictionary*. The bag of words descriptor is simply a histogram describing the frequency of observing these words giving no regard to their position. To compute the dictionary, interest points are found in a large number of images and the associated descriptor is computed. These descriptors are clustered using K-means (section 13.4.4). To compute the bag of words representation, each descriptor is assigned to the nearest word in this dictionary.

The bag of words representation is a remarkably good substrate for object recognition. This is somewhat surprising given that it surrenders any knowledge about the spatial configuration about the object. Of course, the drawback of approaches based on the bag of words is that it is very hard to localize the object after we have identified its presence or to decide how many instances are present using the same model.

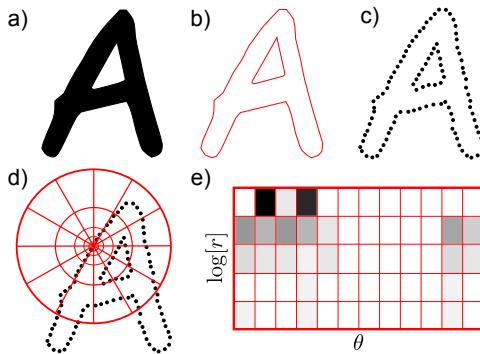


Figure 13.18 Shape context descriptor. a) Object silhouette. b) Contour of silhouette. c) Points are placed at equally spaced intervals around the silhouette. d) A log polar sampling array is centered at each point. e) The shape of the object relative to this point is captured by the histogram over the bins of the log polar array. The final descriptor would consist of a concatenation of the values from histograms from multiple points around the edge of the object.

13.3.5 Shape context descriptor

For certain vision tasks, the silhouette of the object contains much more information than the RGB values themselves. Consider, for example, the problem of body pose estimation: given an image of a human being, the goal is to estimate the 3D joint angles of the body. Unfortunately, the RGB values of the image depend on the person's clothing and are relatively uninformative. In such situations, it is wiser to attempt to characterize the shape of the object.

The shape context descriptor is a fixed length vector that characterizes the object contour. Essentially, it encodes the relative position of points on the contour. In common with the SIFT and HOG descriptors, it pools information locally over space to provide a representation that can capture the overall structure of the object but is not affected too much by small spatial variations.

To compute the shape context descriptor (figure 13.18), a discrete set of points is sampled along the contour of the object. A fixed length vector is associated with each point that characterizes the relative position of the other points. To this end, a log polar sampling array is centered on the current point. A histogram is then computed where each bin contains the number of the other points on the silhouette that fell into each bin of the log-polar array. The choice of the log-polar scheme means that the descriptor is very sensitive to local changes in the shape, but only captures the approximate configuration of distant parts.

The collection of histograms for all of the points on this image captures the shape. However, to directly match to another shape, the point correspondence must be established. It is possible to make this descriptor invariant to orientation by evaluating the orientation of the contour at each point and rotating the log polar sampling scheme so that it is aligned with this orientation.

13.4 Dimensionality reduction

It is often desirable to reduce the dimensionality of either the original or pre-processed image data. If we can do this without losing too much information, then the resulting models will require fewer parameters and be faster to learn and to

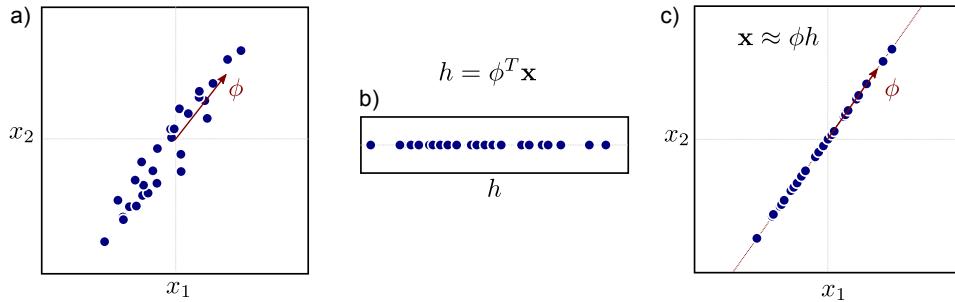


Figure 13.19 Reduction to a single dimension. a) Original data and direction ϕ of maximum variance. b) The data are projected onto ϕ to produce a one dimensional representation. c) To reconstruct the data, we re-multiply by ϕ . Most of the original variation is retained. PCA extends this model to project high dimensional data onto the K orthogonal dimensions with the most variance, to produce a K dimensional representation.

use for inference.

Dimensionality reduction is possible because a given type of image data (e.g., RGB values from face images) usually lie in a tiny subset of the possible data space; not all sets of RGB values look like real images, and not all real images look like faces. We refer to the subset of the space occupied by a given dataset as a *manifold*. Dimensionality reduction can hence be thought of as a change of variables: we move from the original coordinate system to the (reduced) coordinate system within the manifold.

Our goal is hence to find a low-dimensional (or hidden) representation \mathbf{h} which can approximately explain the data \mathbf{x} , so that

$$\mathbf{x} \approx f[\mathbf{h}, \boldsymbol{\theta}], \quad (13.16)$$

where $f[\bullet, \bullet]$ is a function that takes the hidden variable and a set of parameters $\boldsymbol{\theta}$. We would like the lower-dimensional representation to capture all of the relevant variation in the original data. Hence, one possible criterion for choosing the parameters is to minimize the least squares *reconstruction error*, so that

$$\hat{\boldsymbol{\theta}}, \hat{\mathbf{h}}_{1\dots I} = \underset{\boldsymbol{\theta}, \mathbf{h}_1\dots I}{\operatorname{argmin}} \left[\sum_{i=1}^I (\mathbf{x}_i - f[\mathbf{h}_i, \boldsymbol{\theta}])^T (\mathbf{x}_i - f[\mathbf{h}_i, \boldsymbol{\theta}]) \right], \quad (13.17)$$

where \mathbf{x}_i is the i^{th} of I training examples. In other words, we aim to find a set of low dimensional variables $\{\mathbf{h}_i\}_{i=1}^I$ and a mapping from \mathbf{h} to \mathbf{x} so that it reconstructs the original data as closely as possible in a least squares sense.

13.4.1 Approximation with a single number

Let us first consider a very simple model in which we attempt to represent each observed datum with a single number (figure 13.19), so that

$$\mathbf{x}_i \approx \phi h_i + \mu, \quad (13.18)$$

where the parameter μ is the mean of the data set and the parameter ϕ is a basis vector mapping the low dimensional representation \mathbf{h} back to the original data space \mathbf{x} . For simplicity, we will assume from now on that the mean of the dataset is zero and $\mathbf{x}_i \approx \phi h_i$. This can be achieved by computing the empirical mean μ and subtracting it from every example \mathbf{x}_i .

The learning algorithm optimizes the criterion

$$\hat{\phi}, \hat{h}_{1\dots I} = \underset{\phi, h_{1\dots I}}{\operatorname{argmin}} [E] = \underset{\phi, h_{1\dots I}}{\operatorname{argmin}} \left[\sum_{i=1}^I (\mathbf{x}_i - \phi h_i)^T (\mathbf{x}_i - \phi h_i) \right]. \quad (13.19)$$

Careful consideration of the cost function (equation 13.19) reveals an immediate problem: the solution is ambiguous as we can multiply the basis function ϕ by any constant k and divide each of the hidden variables $\{h_i\}_{i=1}^I$ by the same number to yield exactly the same cost. To resolve this problem, we force the vector ϕ to have unit length. This is accomplished by adding in a Lagrange multiplier λ so that the cost function becomes

$$\begin{aligned} E &= \sum_{i=1}^I (\mathbf{x}_i - \phi h_i)^T (\mathbf{x}_i - \phi h_i) + \lambda(\phi^T \phi - 1) \\ &= \sum_{i=1}^I \mathbf{x}_i^T \mathbf{x}_i - 2h_i \phi^T \mathbf{x}_i + h_i^2 + \lambda(\phi^T \phi - 1). \end{aligned} \quad (13.20)$$

To minimize the function, we first take the derivative with respect to h_i , and then equate the resulting expression to zero to yield

$$\hat{h}_i = \hat{\phi}^T \mathbf{x}_i. \quad (13.21)$$

In other words, to find the reduced dimension representation h_i we simply project the observed data onto the vector ϕ .

We now take the derivative of equation 13.20 with respect to ϕ , substitute in the solution for h_i , equate the result to zero, and re-arrange to get

$$\sum_{i=1}^I \mathbf{x}_i \mathbf{x}_i^T \hat{\phi} = \lambda \hat{\phi}, \quad (13.22)$$

or in matrix form

$$\mathbf{X} \mathbf{X}^T \hat{\phi} = \lambda \hat{\phi}, \quad (13.23)$$

where the matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_I]$ contains the data examples in its columns. This is an eigenvalue problem. To find the optimal vector, we compute the SVD $\mathbf{ULV}^T = \mathbf{XX}^T$ and choose the first column of \mathbf{U} .

The scatter matrix \mathbf{XX}^T is a constant multiple of the covariance matrix, and so this has a simple geometric interpretation. The optimal vector ϕ to project onto corresponds to the principal direction of the covariance ellipse. This makes intuitive sense; we retain information from the direction in space where the data vary most.

13.4.2 Principal component analysis

Principal component analysis (PCA) generalizes the above model. Instead of finding a scalar variable h_i that represents the i^{th} data example \mathbf{x}_i , we now seek a K dimensional vector \mathbf{h}_i . The relation between the hidden and observed spaces is

$$\mathbf{x}_i \approx \Phi \mathbf{h}_i, \quad (13.24)$$

where the matrix $\Phi = [\phi_1, \phi_2, \dots, \phi_K]$ contains K basis functions or *principal components*; the observed data are modeled as a weighted sum of the principal components, where the k^{th} dimension of \mathbf{h}_i weights the k^{th} component.

The solution for the unknowns Φ and $\mathbf{h}_{1\dots I}$ can now be written as

$$\hat{\Phi}, \hat{\mathbf{h}}_{1\dots I} = \underset{\Phi, \mathbf{h}_{1\dots I}}{\operatorname{argmin}} [E] = \underset{\Phi, \mathbf{h}_{1\dots I}}{\operatorname{argmin}} \left[\sum_{i=1}^I (\mathbf{x}_i - \Phi \mathbf{h}_i)^T (\mathbf{x}_i - \Phi \mathbf{h}_i) \right]. \quad (13.25)$$

Once more, the solution to this is non-unique as we can post-multiply Φ by any matrix \mathbf{A} and pre-multiply each hidden variable \mathbf{h}_i by the inverse \mathbf{A}^{-1} and still get the same cost. To (partially) resolve this problem, we add the extra constraint that $\Phi^T \Phi = \mathbf{I}$. In other words, we force the principal components to be orthogonal and length one. This gives a modified cost function of

$$E = \sum_{i=1}^I (\mathbf{x}_i - \Phi \mathbf{h}_i)^T (\mathbf{x}_i - \Phi \mathbf{h}_i) + \lambda (\Phi^T \Phi - \mathbf{I}), \quad (13.26)$$

where λ is a Lagrange multiplier. We now minimize this expression with respect to $\Phi, \mathbf{h}_{1\dots I}$ and λ . The expression for the hidden variables becomes

$$\mathbf{h}_i = \Phi^T \mathbf{x}_i. \quad (13.27)$$

The K principal components $\Phi = [\phi_1, \phi_2, \dots, \phi_K]$ are now found by computing the singular value decomposition $\mathbf{ULV}^T = \mathbf{XX}^T$ and taking the first K columns of \mathbf{U} . In other words, to reduce the dimensionality, we project the data \mathbf{x}_i onto a hyperplane defined by the K largest axes of the covariance ellipsoid.

This algorithm is very closely related to probabilistic principal component analysis (section 17.5.1). Probabilistic PCA additionally models the noise that accounts for the inexact approximation in equation 13.24. Factor analysis (section 7.6) is also very similar, but constructs a more sophisticated model of this noise.

Algorithm 13.1

13.4.3 Dual principal component analysis

The preceding method requires us to compute the SVD of the scatter matrix \mathbf{XX}^T . Unfortunately, if the data had dimension D , then this is a $D \times D$ matrix, which may be very large. We can sidestep this problem by using dual variables. We define Φ as a weighted sum of the original data points so that

$$\Phi = \mathbf{X}\Psi, \quad (13.28)$$

where $\Psi = [\psi_1, \psi_2, \dots, \psi_K]$ is a $I \times K$ matrix representing these weights. The associated cost function now becomes

$$E = \sum_{i=1}^I (\mathbf{x}_i - \mathbf{X}\Psi\mathbf{h}_i)^T (\mathbf{x}_i - \mathbf{X}\Psi\mathbf{h}_i) + \lambda(\Psi^T \mathbf{X}^T \mathbf{X}\Psi - \mathbf{I}). \quad (13.29)$$

The solution for the hidden variables becomes

$$\mathbf{h}_i = \Psi^T \mathbf{X}^T \mathbf{x}_i = \Phi^T \mathbf{x}_i, \quad (13.30)$$

and the K dual principal components $\Psi = [\psi_1, \psi_2, \dots, \psi_K]$ are extracted from the matrix \mathbf{U} in the SVD $\mathbf{ULV}^T = \mathbf{X}^T \mathbf{X}$. This is a smaller problem of size $I \times I$, and is more efficient when the number of data examples I is less than the dimensionality of the observed space D .

Notice that this algorithm does not require the original data points: it only requires the inner products between them and so it is amenable to kernelization. This resulting method is known as kernel PCA.

13.4.4 The K-means algorithm

A second common approach to dimensionality reduction is to abandon a continuous representation altogether and represent each data point using one of a limited set of prototype vectors. In this model, the data are approximated as

$$\mathbf{x}_i \approx \boldsymbol{\mu}_{h_i}, \quad (13.31)$$

Algorithm 13.2

where $h_i \in \{1, 2, \dots, K\}$ is an index that identifies which of the K prototype vectors $\{\boldsymbol{\mu}_k\}_{k=1}^K$ approximates the i^{th} example.

To find the assignment indices and the prototype vectors (figure 13.20) we optimize

$$\hat{\boldsymbol{\mu}}_{1\dots K}, \hat{h}_{1\dots I} = \operatorname{argmin}_{\boldsymbol{\mu}, h} \left[\sum_{i=1}^I (\mathbf{x}_i - \boldsymbol{\mu}_{h_i})^T (\mathbf{x}_i - \boldsymbol{\mu}_{h_i}) \right]. \quad (13.32)$$

In the K-means algorithm this cost function is minimized using an alternating strategy in which we first assign each datapoint to the nearest prototype

$$\hat{h}_i = \operatorname{argmin}_{h_i} \left[(\mathbf{x}_i - \boldsymbol{\mu}_{h_i})^T (\mathbf{x}_i - \boldsymbol{\mu}_{h_i}) \right], \quad (13.33)$$

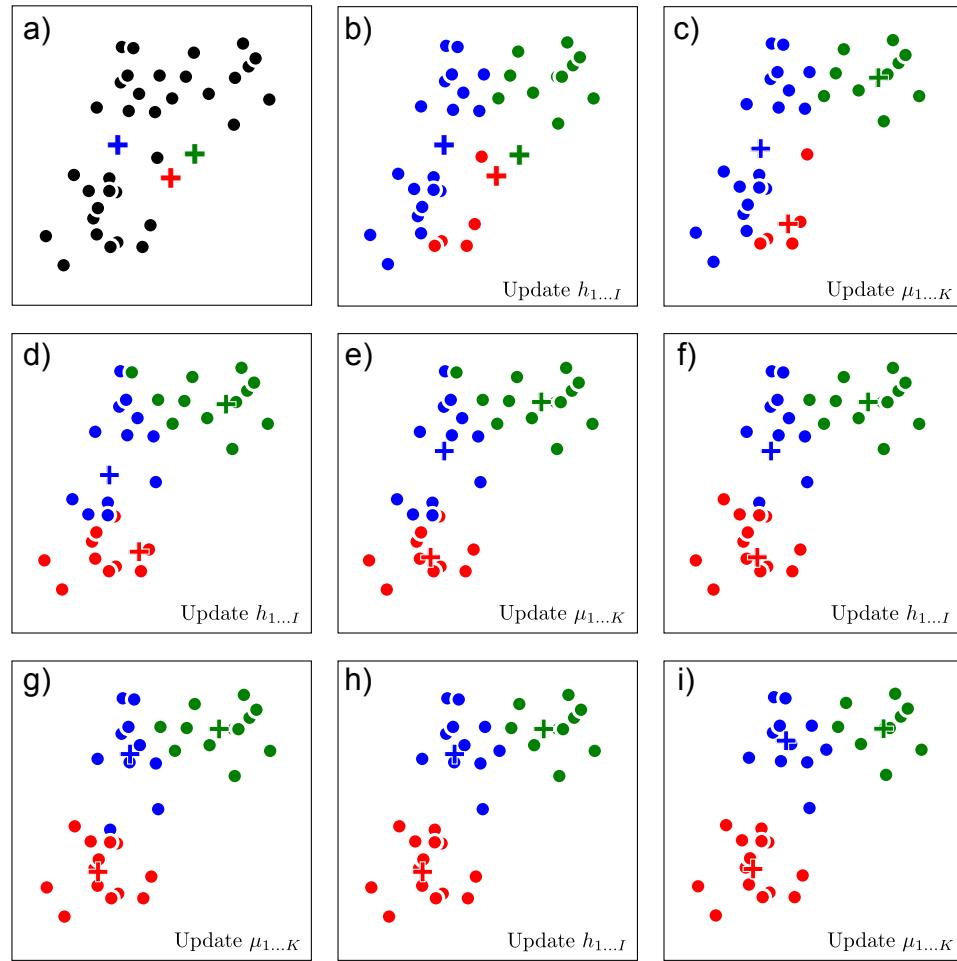


Figure 13.20 K-means algorithm for $K = 3$ clusters. a) We initialize the three prototype vectors (crosses) to random positions. We alternately b) assign the data to the nearest prototype vector and c) update the prototype vectors to be equal to the mean of the points assigned to them. d-i) We repeat these steps until there is no further change.

and then update the prototypes

$$\begin{aligned}\hat{\boldsymbol{\mu}}_k &= \underset{\boldsymbol{\mu}_k}{\operatorname{argmin}} \left[\sum_{i=1}^I \left[(\mathbf{x}_i - \boldsymbol{\mu}_{h_i})^T (\mathbf{x}_i - \boldsymbol{\mu}_{h_i}) \right] \right] \\ &= \frac{\sum_{i=1}^I \mathbf{x}_i \delta[h_i - k]}{\sum_{i=1}^I \delta[h_i - k]},\end{aligned}\quad (13.34)$$

where $\delta[\bullet]$ is a function that returns one when its argument is zero and zero otherwise.

wise. In other words, the new prototype $\hat{\mu}_k$ is simply the average of the data points that are assigned to this cluster. This algorithm is not guaranteed to converge to the global minimum and so it requires sensible starting conditions.

The K-means algorithm is very closely related to the mixtures of Gaussians model (section 7.4). The main differences are that the mixtures of Gaussians model is probabilistic and defines a density over the data space. It also assigns weights to the clusters and describes their covariance.

Problem 13.8
Problem 13.9
Problem 13.10

Conclusion

Careful reading of the information in this chapter should convince you that there are certain recurring ideas in image preprocessing. To make a descriptor invariant to intensity changes we *filter* the image and *normalize* the filter responses over the region. A unique descriptor orientation and scale can be computed by *maximizing* over responses at different orientations and scales. To create invariance to small spatial changes, local responses are *pooled*. Despite the simplicity of these ideas, it is remarkable how much impact they have on the performance of real systems.

Notes

Image Processing: There are numerous texts on image processing which contain far more information than I could include in this chapter. I would particularly recommend the books by O' Gorman *et al.* (2008), Gonzalez & Woods (2002), Pratt (2007), and Nixon & Aguado (2008). A comprehensive recent summary of local image features can be found in Li & Allinson (2008).

Edge and corner detection: The Canny edge detector was first described in Canny (1986). Elder (1999) investigated whether it was possible to reconstruct an image based on edge information alone. Nowadays, it is common to use machine learning methods to identify object boundaries in images (e.g., Dollár *et al.* 2006)

Early work in corner detection (interest point detection) includes that of Moravec (1983), Förstner (1986) and the Harris corner detector (Harris & Stephens 1988) which we described in this chapter. Other more recent efforts to identify stable points and regions include the SUSAN corner detector (Smith & Brady 1997), a saliency-based descriptor (Kadir & Brady 2001), maximally stable extremal regions (Matas *et al.* 2002), the SIFT detector (Lowe 2004) and the FAST detector (Rosten & Drummond 2006). There has been considerable recent interest in *affine invariant* interest point detection, which aims to find features that are stable under affine transforms of the image (e.g., Schaffalitzky & Zisserman 2002; Mikolajczyk & Schmid 2002; Mikolajczyk & Schmid 2004). Mikolajczyk *et al.* (2005) present a quantitative comparison of different affine region detectors. A recent review of this area can be found in Tuytelaars & Mikolajczyk (2007).

Image descriptors: For robust object recognition and image matching it is crucial to characterize the region around the detected interest point in a way that is compact and stable to changes in the image. To this end, Lowe (2004) developed the SIFT descriptor, Dalal & Triggs (2005) developed the HOG descriptor, and Forssén & Lowe (2007) developed a descriptor for use with maximally stable extremal regions. Bay *et al.* (2008) developed a very efficient version of SIFT features known as SURF. Mikolajczyk & Schmid (2005) present a quantitative comparison of region descriptors. Recent work on image descriptors has applied machine learning techniques to optimize their performance (Brown *et al.* 2011; Philbin *et al.* 2010).

More information about local binary patterns can be found in Ojala *et al.* (2002). More information about the shape context descriptor can be found in Belongie *et al.* (2002).

Dimensionality reduction: Principal components analysis is a linear dimensionality reduction method. However, there are also many nonlinear approaches that describe a manifold of images in high dimensions with fewer parameters. Notable methods include kernel PCA (Schölkopf *et al.* 1997), ISOMAP (Tenenbaum *et al.* 2000), local linear embedding Roweis & Saul (2000), charting (Brand 2002), the Gaussian process latent variable model (Lawrence 2004), and Laplacian eigenmaps (Belkin & Niyogi 2001). Recent reviews of dimensionality reduction can be found in Burgess (2010) and De La Torre (2011).

Problems

Problem 13.1 Consider an 8-bit image in which the pixel values are evenly distributed in the range 0 to 127, with no pixels taking a value of 128 or larger. Draw the cumulative histogram for this image (see figure 13.2). What will the histogram of pixel intensities

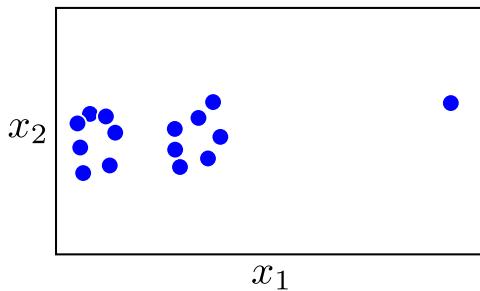


Figure 13.21 Clustering with the K-means algorithm in the presence of outliers (problem 13.9). This data set contains two clusters and a single outlier (the point on the right hand side). The outlier causes problems for the K-means algorithm when $K = 2$ clusters are used due to the implicit assumption that the clusters can be modeled as normal distributions with spherical covariance.

look like after applying histogram equalization?

Problem 13.2 Consider a continuous image $p[i, j]$ and a continuous filter $f[n, m]$. In the continuous domain, the operation $f \otimes p$ of convolving an image with the filter is defined as:

$$f \otimes p = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p[i - m, j - n] f[m, n] dm dn.$$

Now consider two filters f and g . Prove that convolving the image first with f and then with g has the same effect as convolving f with g and then convolving the image with the result. In other words:

$$g \otimes (f \otimes p) = (g \otimes f) \otimes p.$$

Does this result extend to discrete images?

Problem 13.3 Describe the series of operations that would be required to compute the Haar-like filters in figures 13.6a-d from an integral image. How many points from the integral image are needed to compute each?

Problem 13.4 Consider a blurring filter where each pixel in an image is replaced by a weighted average of local intensity values, but the weights decrease if these intensity values differ markedly from the central pixel. What effect would this *bilateral filter* have when applied to an image?

Problem 13.5 Define a 3×3 filter that is specialized to detecting luminance changes at a 45° angle and gives a positive response where the image intensity increases from the bottom left to the bottom right of the image.

Problem 13.6 Define a 3×3 filter that responds to the second derivative in the horizontal direction, but is invariant to the gradient and absolute intensity in the horizontal direction and invariant to all changes in the vertical direction.

Problem 13.7 Why are most local binary patterns in a natural image typically uniform or near-uniform?

Problem 13.8 Give one example of a 2D data set where the mixtures of Gaussians model will succeed in clustering the data, but the K-means algorithm will fail.

Problem 13.9 Consider the data in figure 13.21. What do you expect to happen if we run the K-means algorithm with two clusters on this data set? Suggest a way to resolve this problem.

Problem 13.10 An alternative approach to clustering the data would be to find modes (peaks) in the density of the points. This potentially has the advantage of also automatically selecting the number of clusters. Propose an algorithm to find these modes.

Part V

Models for geometry

Part V: Models for geometry

In part V, we finally acknowledge the process by which real-world images are formed. Light is emitted from one or more sources and travels through the scene interacting with the materials via physical processes such as reflection, refraction, and scattering. Some of this light enters the camera and is measured. We have a very good understanding of this forward model. Given known geometry, light sources, and material properties, computer graphics techniques can simulate what will be seen by the camera very accurately.

The ultimate goal for a vision algorithm would be a complete *reconstruction*, in which we aim to invert this forward model, and estimate the light sources, materials, and geometry from the image. Here, we aim to capture a structural description of the world: we seek an understanding of where things are, and to measure their optical properties, rather than a semantic understanding. Such a structural description can be exploited to navigate around the environment or build 3D models for computer graphics.

Unfortunately, full visual reconstruction is very challenging. For one thing, the solution is non-unique. For example, if the light source intensity increases, but the object reflectance decreases commensurately, the image will remain unchanged. Of course, we could make the problem unique by imposing prior knowledge, but even then reconstruction remains difficult; it is hard to effectively parameterize the scene, and the problem is highly non-convex.

In this part of the book, we consider a family of models that approximate both the 3D scene and the observed image with sparse sets of visual primitives (points). The forward model that maps the proxy representation of the world (3D points) to the proxy representation of the image (2D points) is much simpler than the full light transport model, and is called the *projective pinhole camera*. We investigate the properties of this model in chapter 14.

In chapter 15, we consider the situation where the pinhole camera views a plane in the world; there is now a one-to-one mapping between points on the plane and points in the image, and we characterize this mapping with a family of 2D transformations. In chapter 16, we will further exploit the pinhole camera model to recover a sparse geometric model of the scene.

