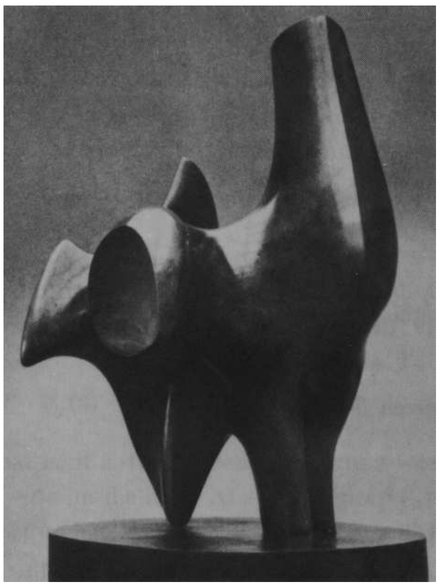
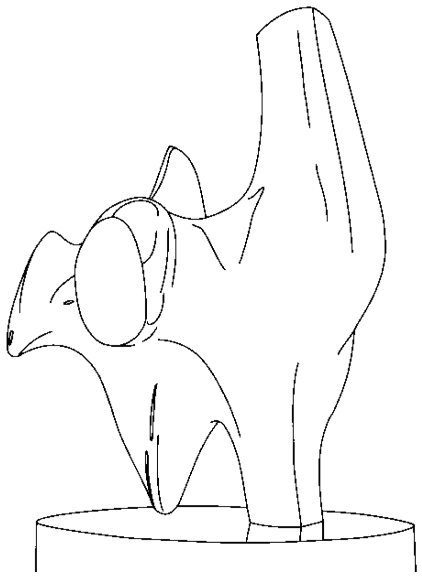
University of Cambridge
Engineering Part IIB

Module 4F12: Computer Vision

Image Structure: Feature Detection and Matching

Roberto Cipolla
October 2024

Image Intensities

We can represent a monochrome image as a matrix $I(x, y)$ of intensity values. The size of the matrix is typically 320×240 (QVGA) , 640×480 (VGA), 1280×720 (HD) or even as large as 4096×2160 (4K) and the intensity values are usually sampled to an accuracy of 8 bits for monochrome images (256 **grey levels**) or for each the 3 **RGB** colour channels.

If a point on an object is visible in view, the corresponding pixel intensity, $I(x, y)$ is a function of many geometric and photometric variables, including:

1. The position and orientation of the camera;
2. The geometry of the scene (3D shapes and layout);
3. The nature and distribution of light sources;
4. The reflectance properties of the surfaces: specular $\leftrightarrow$ Lambertian, albedo 0 (black) $\leftrightarrow$ 1 (white);
5. The properties of the lens and the CCD.

In practice the point may also only be partially visible or its appearance may be also be affected by occlusion.

Data reduction

With current computer technology, it is necessary to discard most of the data coming from the camera before any attempt is made at real-time image interpretation.

$$\begin{array}{rcl} \text{images} & \rightarrow & \text{generic salient features} \\ \text{100 MBytes/s} & & \text{100 KBytes/s} \\ \text{(mono CCD)} & & \end{array}$$

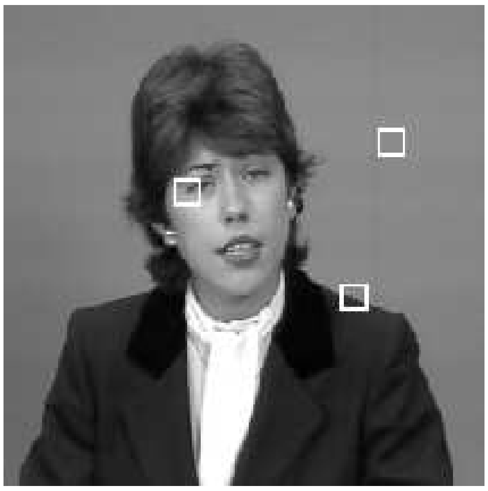
All subsequent interpretation is performed on the generic representation, not the original image. We aim to:

- Dramatically reduce the amount of data.
- Preserve the useful information in the images (such as the albedo changes and 2D shape of objects in the scene).
- Discard the redundant information in the images (such as the lighting conditions).

We would also like to arrive at a *generic* representation, so the same processing will be useful across a wide range of applications.

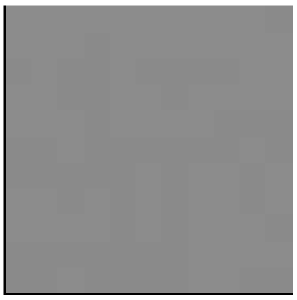
Image structure

The answer becomes apparent if we look at the structure of a typical image. In this photo of "Claire", we'll examine the pixel data around several patches: a featureless region, an edge and a corner.

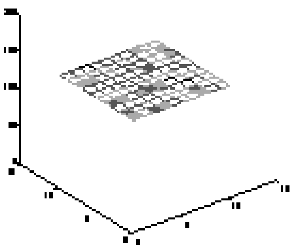


0D

The featureless region is characterized by a smooth variation of intensities.



140	141	141	140	141	141	140	141	141	141	139
140	141	141	139	140	140	140	140	140	140	140
139	140	139	139	140	139	139	138	139	140	140
140	140	139	139	140	140	138	140	140	140	140
140	140	141	139	141	141	140	140	139	139	138
139	139	141	139	139	139	139	139	139	140	139
139	139	139	139	139	140	139	140	140	139	141
140	140	139	140	139	141	139	140	140	139	140
140	140	140	140	139	140	139	140	131	131	139
139	139	138	139	139	139	139	140	141	140	140
139	139	140	139	139	139	139	140	140	139	139

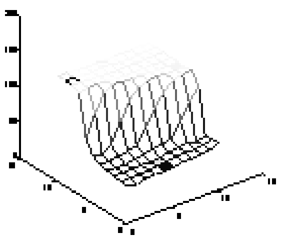


Edges and corners

1D

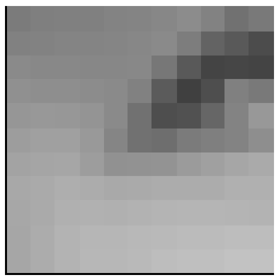
The patch containing the edge reveals an intensity discontinuity in one direction.

136	136	135	134	135	134	134	135	135	136	135
136	135	134	134	136	135	133	133	135	135	136
136	135	134	134	133	133	134	134	134	134	135
130	133	134	133	132	132	132	133	133	132	133
73	103	127	135	134	133	134	133	132	133	134
54	52	60	88	114	127	133	134	132	133	132
49	48	50	53	56	76	99	117	130	133	135
46	45	45	48	50	53	55	57	77	99	118
44	44	45	46	45	47	50	52	54	49	54
42	43	43	44	46	47	50	51	50	52	55
41	40	44	44	44	46	49	48	50	51	56

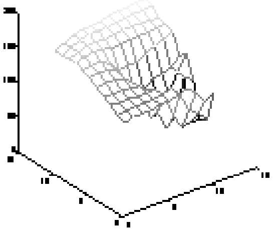


2D

The patch containing the corner reveals an intensity discontinuity in two directions.



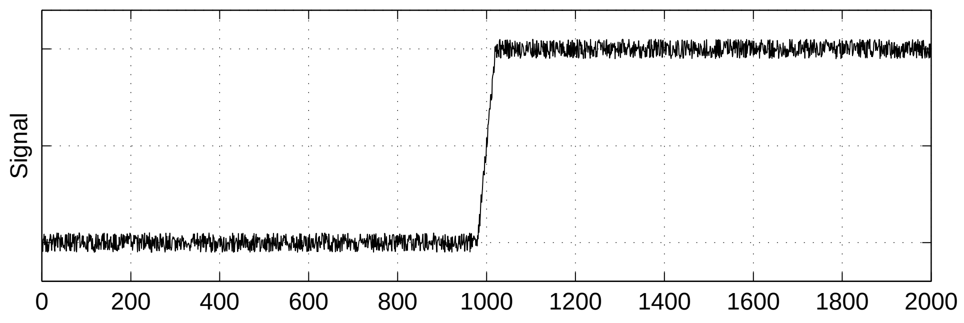
124	127	128	128	131	312	135	140	130	113	121
129	130	132	132	133	135	139	125	99	89	76
138	136	137	135	135	136	120	89	68	71	69
144	142	143	141	139	129	92	64	78	128	121
150	152	151	148	138	113	79	81	102	131	152
158	160	160	154	133	113	111	123	127	131	143
163	165	166	158	145	146	147	155	160	166	171
168	171	174	173	169	171	172	173	173	176	176
167	171	176	177	176	178	180	181	181	180	179
166	173	179	182	182	184	185	187	188	189	190
166	173	179	183	183	185	189	191	191	194	194



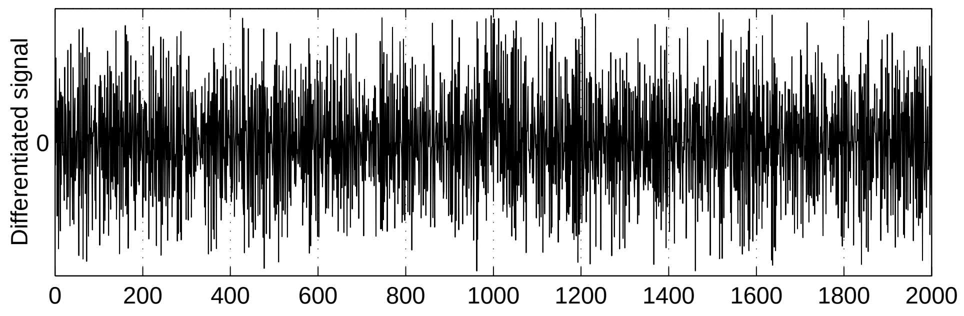
Note that an edge or corner representation imparts a desirable invariance to lighting: the intensity discontinuities are likely to be prominent, whatever the lighting conditions.

1D edge detection

We start with the simple case of edge detection in one dimension. When developing an edge detection algorithm, it is important to bear in mind the invariable presence of image noise. Consider this signal $I(x)$ with an obvious edge.



An intuitive approach to edge detection might be to look for maxima and minima in $I'(x)$.



This simple strategy is defeated by noise. For this reason, all edge detectors start by smoothing the signal to suppress noise. The most common approach is to use a Gaussian filter.

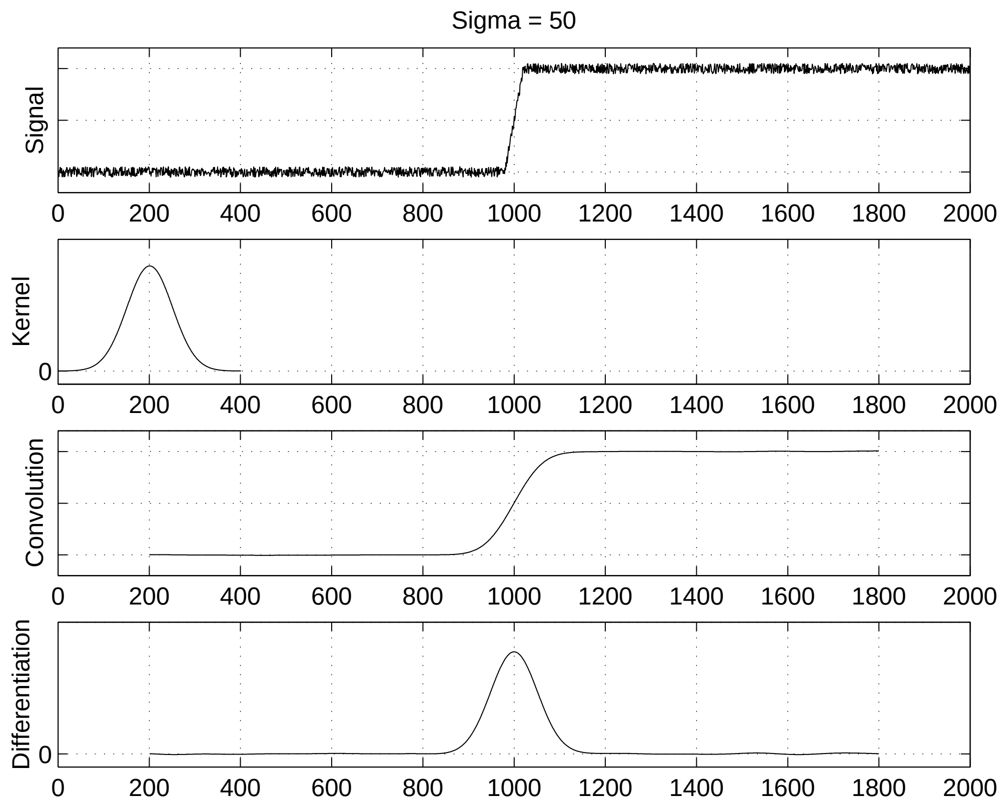
1D edge detection

A broad overview of 1D edge detection is:

1. Convolve the signal $I(x)$ with a Gaussian kernel $g_\sigma(x)$. Call the smoothed signal $s(x)$.

$$g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

2. Compute $s'(x)$, the derivative of $s(x)$.
3. Find maxima and minima of $s'(x)$.
4. Use thresholding on the magnitude of the extrema to mark edges.



1D edge detection

The smoothing in step (1) is performed by a 1D convolution:

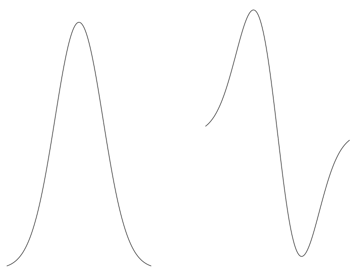
$$s(x) = g_\sigma(x) * I(x) = \int_{-\infty}^{+\infty} g_\sigma(u) I(x-u)\,du = \int_{-\infty}^{+\infty} g_\sigma(x-u) I(u)\,du$$

The differentiation in step (2) is also performed by a 1D convolution. Thus edge detection would appear to require two computationally expensive convolutions.

However, the derivative theorem of convolution tells us that

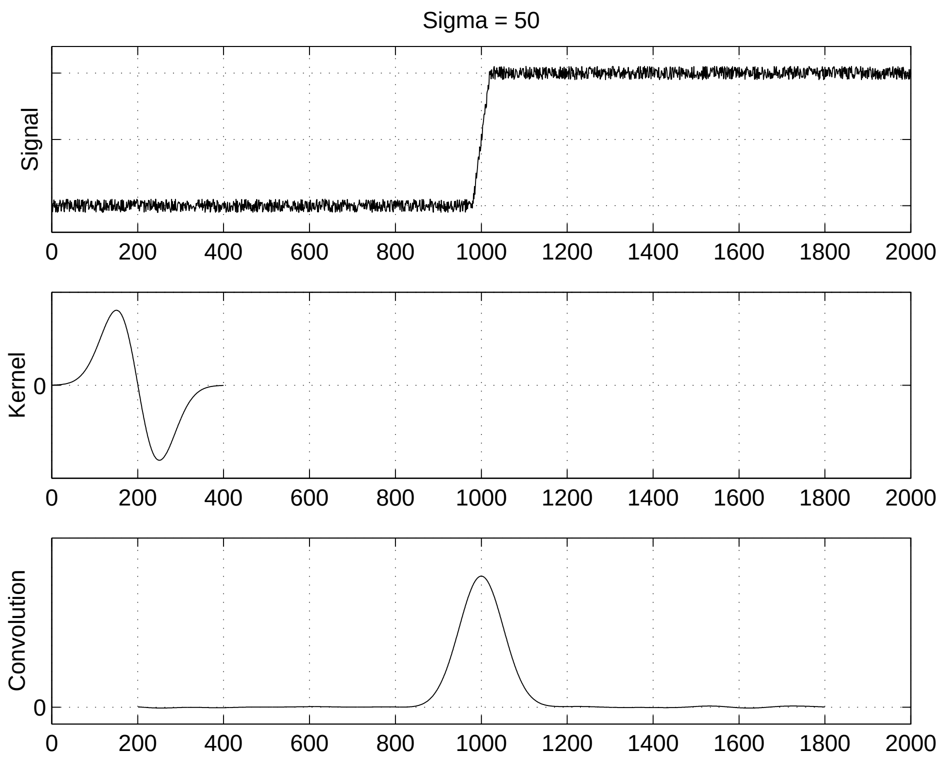
$$s'(x) = \frac{d}{dx}\left[g_\sigma(x) * I(x)\right] = g'_\sigma(x) * I(x)$$

so we can compute $s'(x)$ by convolving only once — a considerable computational saving.



$g_\sigma(x)$ $g'_\sigma(x)$

1D edge detection

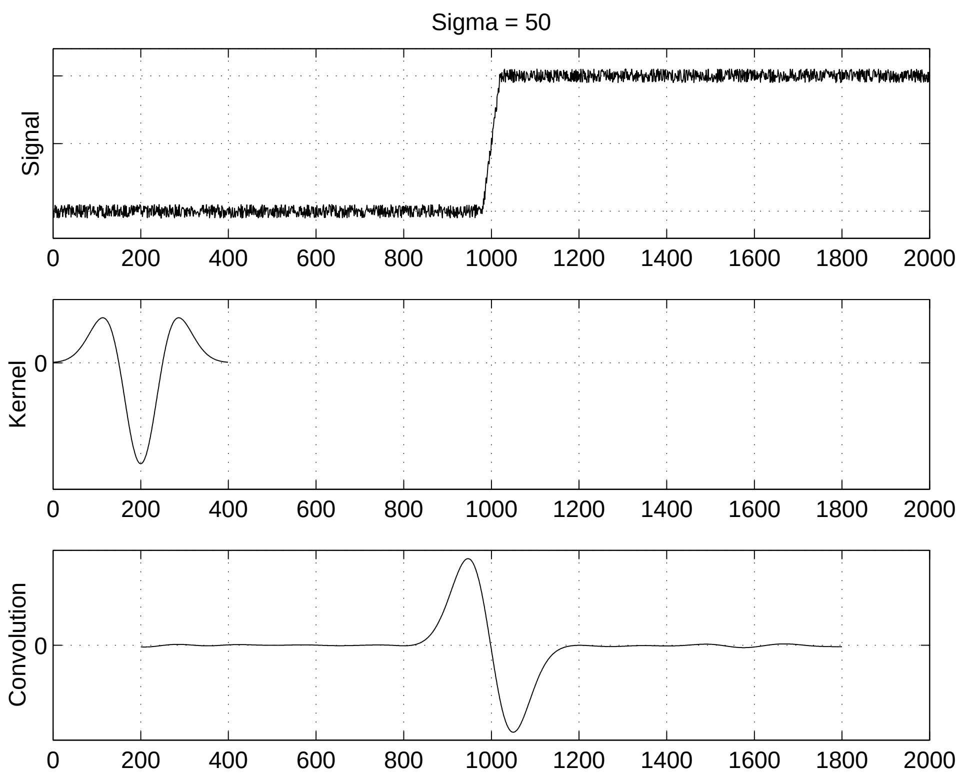


Having obtained the convolved signal $s'(x)$, interpolation can be used to locate any maxima or minima to sub-pixel accuracy. Finally, an edge is marked at each maximum or minimum whose magnitude exceeds some threshold.

Looking for maxima and minima of $s'(x)$ is the same as looking for zero-crossings of $s''(x)$. In many implementations of edge detection algorithms, the signal is convolved with the *Laplacian* of a Gaussian, $g''_\sigma(x)$:

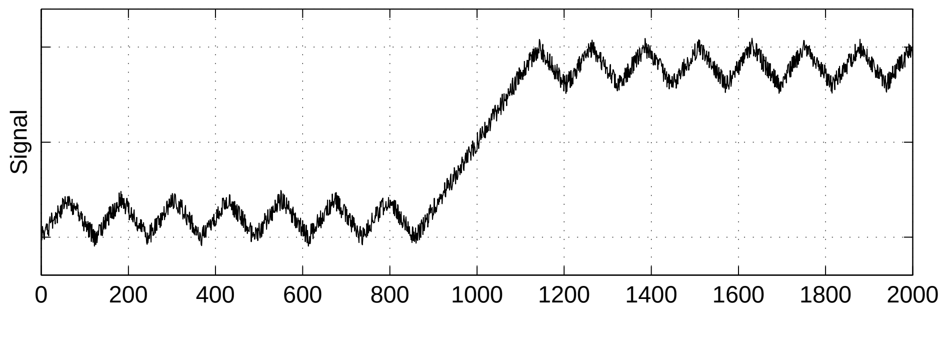
$$s''(x) = g''_\sigma(x) * I(x)$$

Zero-crossings



The zero-crossings of $s''(x)$ mark possible edges.

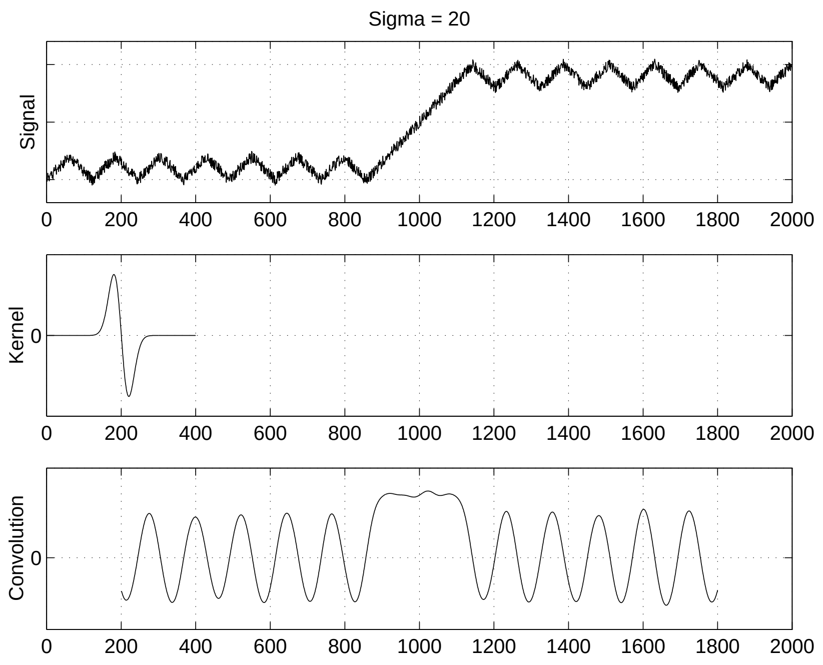
We have not yet addressed the issue of what value of σ to use. Consider this signal:



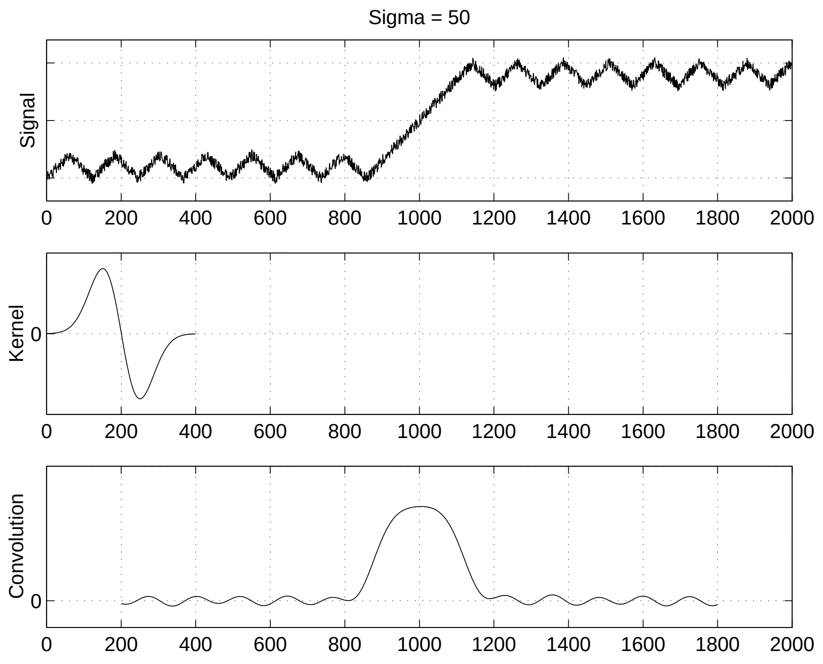
Does the signal have one positive edge or a number of positive and negative edges?

Multi-scale edge detection

Using a small σ brings out all the edges.



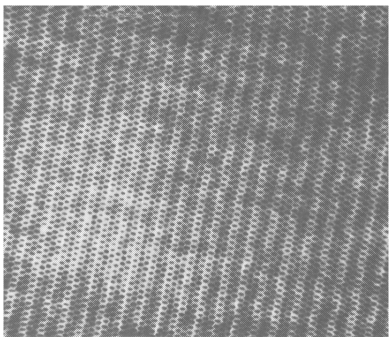
As σ increases, the signal is smoothed more and more, and only the central edge survives.



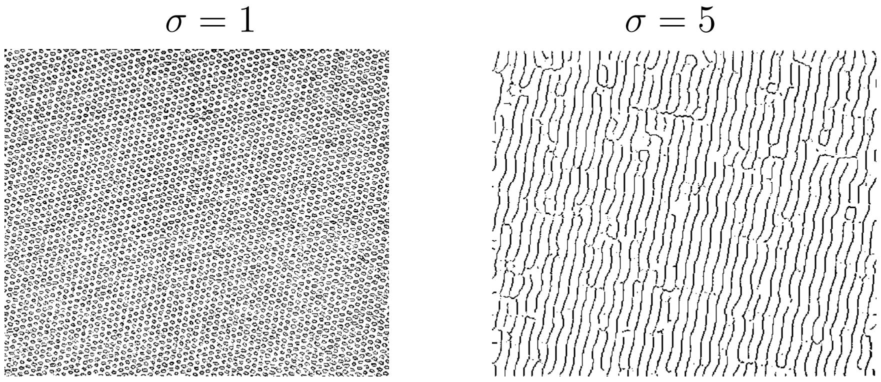
Multi-scale edge detection

The amount of smoothing controls the **scale** at which we analyse the image. There is no right or wrong size for the Gaussian kernel: it all depends on the scale we're interested in.

Modest smoothing (a Gaussian kernel with small σ) brings out edges at a fine scale. More smoothing (larger σ) identifies edges at larger scales, suppressing the finer detail.



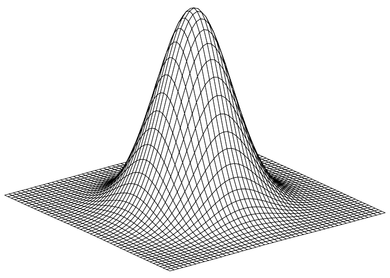
This is an image of a dish cloth. After edge detection, we see different features at different scales.



Fine scale edge detection is particularly sensitive to noise. This is less of an issue when analysing images at coarse scales.

2D edge detection

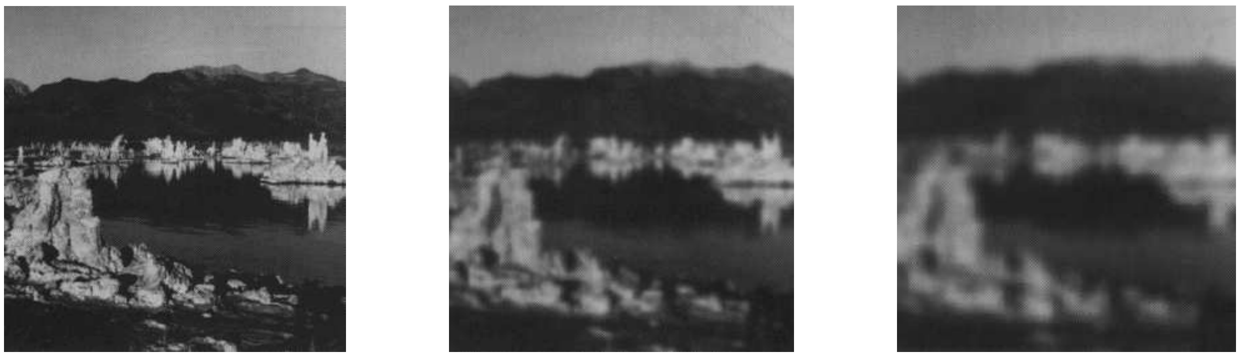
The 1D edge detection scheme can be extended to work in two dimensions. First we smooth the image $I(x, y)$ by convolving with a 2D Gaussian $G_\sigma(x, y)$:



$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2}\ \exp - \left(\frac{x^2 + y^2}{2\sigma^2}\right)$$

$$\begin{aligned} S(x, y) &= G_\sigma(x, y) * I(x, y) \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} G_\sigma(u, v) I(x - u, y - v)\, du\, dv \end{aligned}$$

The effects of this blurring on a typical image:



Unsmoothed $\sigma = 3$ pixels $\sigma = 4$ pixels

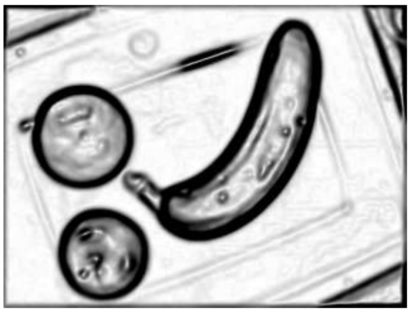
2D edge detection

The next step is to find the gradient of the smoothed image $S(x, y)$ at every pixel:

$$\begin{aligned}\nabla S &= \nabla(G_\sigma * I) \\ &= \begin{bmatrix} \frac{\partial(G_\sigma * I)}{\partial x} \\ \\ \frac{\partial(G_\sigma * I)}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial G_\sigma}{\partial x} * I \\ \\ \frac{\partial G_\sigma}{\partial y} * I \end{bmatrix}\end{aligned}$$

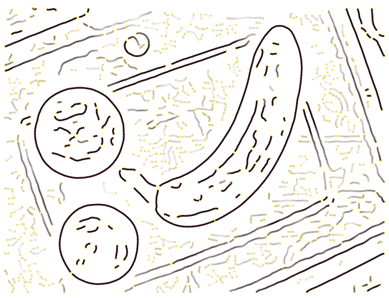
The following example shows $|\nabla S|$ for a fruity image:

(a) Original image

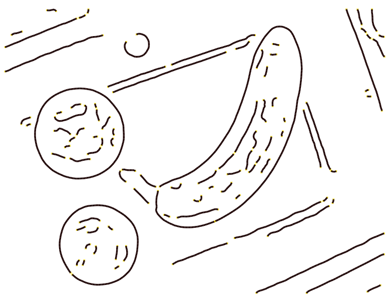

(b) Edge strength $|\nabla S|$

2D edge detection

The next stage of the edge detection algorithm is **non-maximal suppression**. Edge elements, or **edgels**, are placed at locations where $|\nabla S|$ is greater than local values of $|\nabla S|$ in the directions $\pm\nabla S$. This aims to ensure that all edgels are located at ridge-points of the surface $|\nabla S|$.


(c) Non-maximal suppression

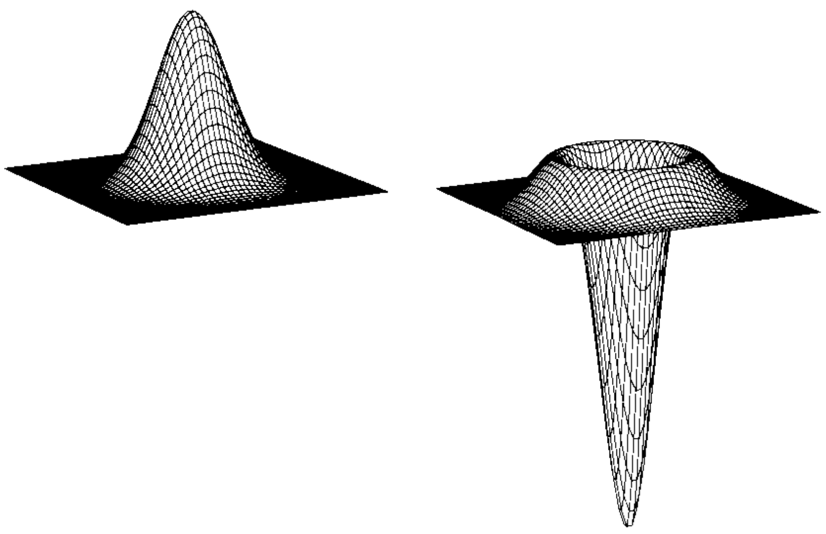
Next, the edgels are **thresholded**, so that only those with $|\nabla S|$ above a certain value are retained.


(d) Thresholding

2D edge detection

The edge detection algorithm we have been describing is due to Canny (1986). The output is a list of edgel positions, each with a strength $|\nabla S|$ and an orientation $\nabla S / |\nabla S|$.

An alternative approach to edge detection was developed by Marr and Hildreth (1980). While the Canny detector is a *directional* edge finder (both the gradient magnitude and direction are computed), the Marr-Hildreth operator is *isotropic*. It finds zero-crossings of $\nabla^2 G_\sigma * I$, where $\nabla^2 G_\sigma$ is the **Laplacian** of G_σ ($\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$).

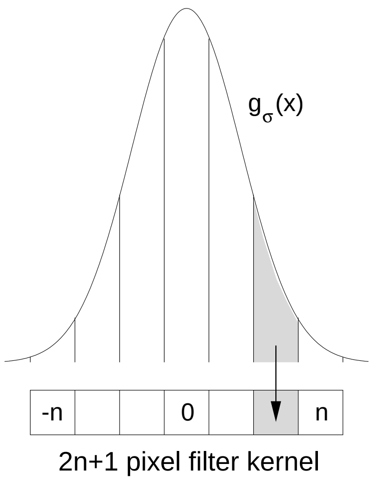


G_σ $\nabla^2 G_\sigma$

Implementation details

In practice, the image and filter kernels are discrete quantities and the convolutions are performed as truncated summations:

$$S(x,y) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} G_\sigma(u,v) I(x-u, y-v)$$



For acceptable accuracy, kernels are generally truncated so that the discarded samples are less than 1/1000 of the peak value.

σ	1.0	1.5	3	6
$2n+1$	7	11	23	45

The 2D convolutions would appear to be computationally expensive. However, they can be decomposed into two 1D convolutions:

$$G_\sigma(x,y) * I(x,y) = g_\sigma(x) * [g_\sigma(y) * I(x,y)]$$

The computational saving is $(2n+1)^2/2(2n+1)$.

Implementation details

Differentiation of the smoothed image is also implemented with a discrete convolution.

By considering the Taylor-series expansion of $S(x, y)$ it is easy to show that a simple finite-difference approximation to the first-order spatial derivative of $S(x, y)$ is given by:

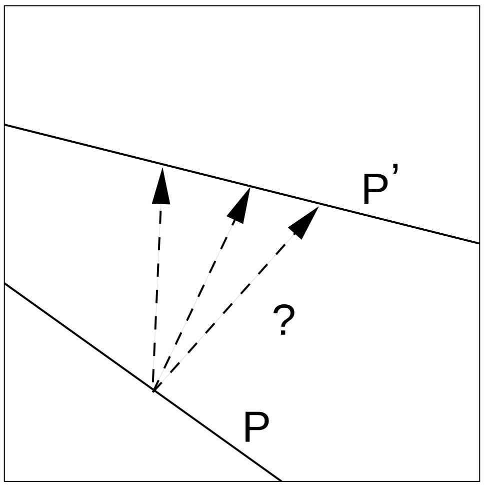
$$\frac{\partial S}{\partial x} \approx \frac{S(x+1, y) - S(x-1, y)}{2}$$

This is equivalent to convolving the rows of image samples, $S(x, y)$, with the kernel

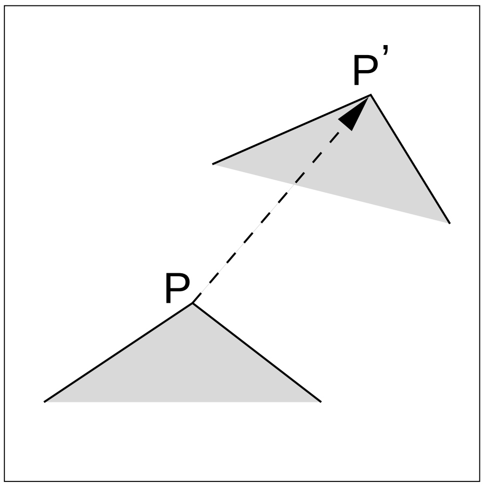
1/2	0	-1/2

Corners

While edges are a powerful intermediate representation, they are sometimes insufficient. This is especially the case when image *motion* is being analysed. The motion of an edge is rendered ambiguous by the **aperture problem**: when viewing a moving edge, it is only possible to measure the motion *normal* to the edge locally.



Edge



Corner

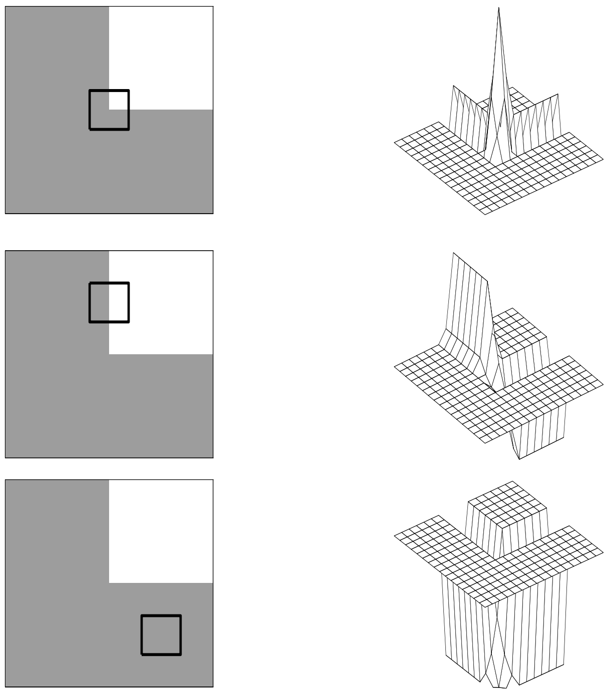
To measure image motion completely, we really need to look at **corner** features. We saw earlier that a corner is characterized by an intensity discontinuity in two directions. This discontinuity can be detected using **correlation**.

Correlation

The normalized **cross-correlation** function measures how well an image patch $P(x, y)$ matches other portions of the image, $I(x, y)$, as it is shifted from its original location. It entails sliding the patch over the image, computing the sum of the products of the pixels and normalizing the result:

$$c(x,y) = \frac{\displaystyle\sum_{u=-n}^{n}\sum_{v=-n}^{n} P(u,v)\, I(x+u, y+v)}{\sqrt{\displaystyle\sum_{u=-n}^{n}\sum_{v-n}^{n} P^2(u,v) \sum_{u=-n}^{n}\sum_{v-n}^{n} I^2(x+u, y+v)}}$$

A patch which has a well-defined peak in its correlation function can be classified as a "corner".



Image,I & patch,P Correlation $c(x, y)$

Note that the cross-correlation is fully normalised to $[-1, 1]$ by computed it from the **covariance and variances** of the two signals/patches (i.e. by subtracting means).

Corner detection

A practical corner detection algorithm needs to do something more efficient than calculate correlation functions for every pixel! We must begin with a smoothed image, $S(x,y)$, before differentiating.

1. Calculate change in intensity in direction $\mathbf{n}$:

$$\begin{aligned} S_n &\equiv \nabla S(x,y).\hat{\mathbf{n}} \equiv [\, S_x \quad S_y \,]^T .\hat{\mathbf{n}} \\ S_n^2 &= \frac{\mathbf{n}^T \nabla S \, \nabla S^T \, \mathbf{n}}{\mathbf{n}^T \mathbf{n}} \\ &= \frac{\mathbf{n}^T \begin{bmatrix} S_x^2 & S_x S_y \\ S_x S_y & S_y^2 \end{bmatrix} \mathbf{n}}{\mathbf{n}^T \mathbf{n}} \end{aligned}$$

   where $S_x \equiv \partial S/\partial x$ and $S_y \equiv \partial S/\partial y$.

2. Smooth S_n^2 by convolution with a Gaussian kernel of size σ_I:

$$\begin{aligned} C_n(x,y) &= G_{\sigma_I}(x,y) * S_n^2 \\ &= \frac{\mathbf{n}^T \begin{bmatrix} \langle S_x^2 \rangle & \langle S_x S_y \rangle \\ \langle S_x S_y \rangle & \langle S_y^2 \rangle \end{bmatrix} \mathbf{n}}{\mathbf{n}^T \mathbf{n}} \end{aligned}$$

   where $\langle \; \rangle$ is the smoothed value. This equivalent to weighting the intensity differences squared, S_n^2, in the local neighbourhood by Gaussian weights centred at (x,y).

Corner detection

The smoothed (weighted sum of) intensity changes around (x,y) in direction $\mathbf{n}$ is therefore given by

$$C_n(x,y) = \frac{\mathbf{n}^T \mathrm{A} \mathbf{n}}{\mathbf{n}^T \mathbf{n}}$$

where A is the 2×2 symmetric matrix

$$\begin{bmatrix} \left\langle S_x^2 \right\rangle & \langle S_x S_y \rangle \\ \langle S_x S_y \rangle & \left\langle S_y^2 \right\rangle \end{bmatrix}$$

Elementary eigenvector theory tells us that

$$\lambda_1 \leq C_n(x,y) \leq \lambda_2$$

where λ_1 and λ_2 are the eigenvalues of A. So, if we try every possible orientation $\mathbf{n}$, the maximum smoothed change in intensity we will find is λ_2, and the minimum value is λ_1.

We can therefore classify image structure around each pixel by looking at the eigenvalues of A:

No structure: (smooth variation) $\lambda_1 \approx \lambda_2 \approx 0$

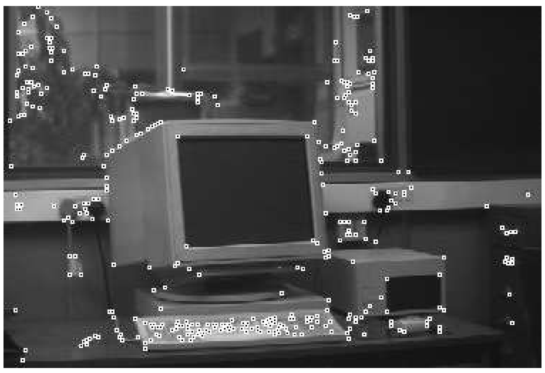
1D structure: (edge) $\lambda_1 \approx 0$ (direction of edge), λ_2 large (normal to edge)

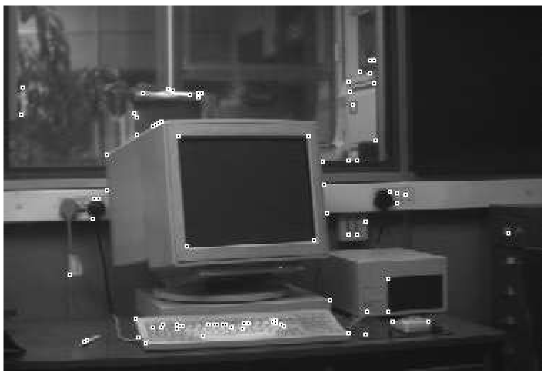
2D structure: (corner) λ_1 and λ_2 both large and distinct

Corner detection

It is necessary to calculate A at every pixel by first computing 3 images of smoothed gradients. We avoid computing the actual eigenvalues by evaluating the determinant ($\det A = \lambda_1\lambda_2$) and trace of the matrix A (trace A$= \lambda_1 + \lambda_2$) instead.

The corner detection algorithm we have been describing is due to Harris (1987). We mark corners where the quantity $\lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2$ exceeds some threshold ($\kappa \approx 0.04$ makes the detector a little "edge-phobic").

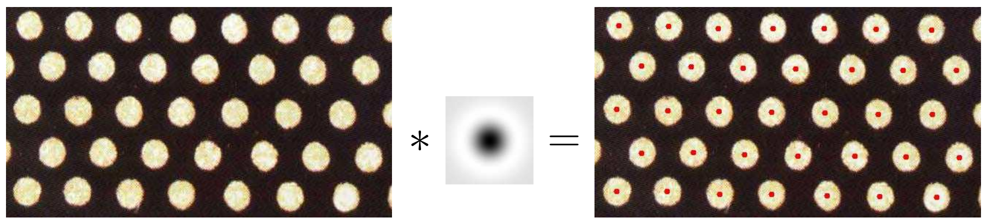

Low threshold


High threshold

Corners are most useful for **tracking** in image sequences or **matching** in stereo pairs. Unlike edges, the displacement of a corner is not ambiguous. Corner detectors must be judged on their ability to detect the *same* corners in similar images. Current detectors are not too reliable, and higher-level visual routines must be designed to tolerate a significant number of **outliers** in the output of the corner detector.

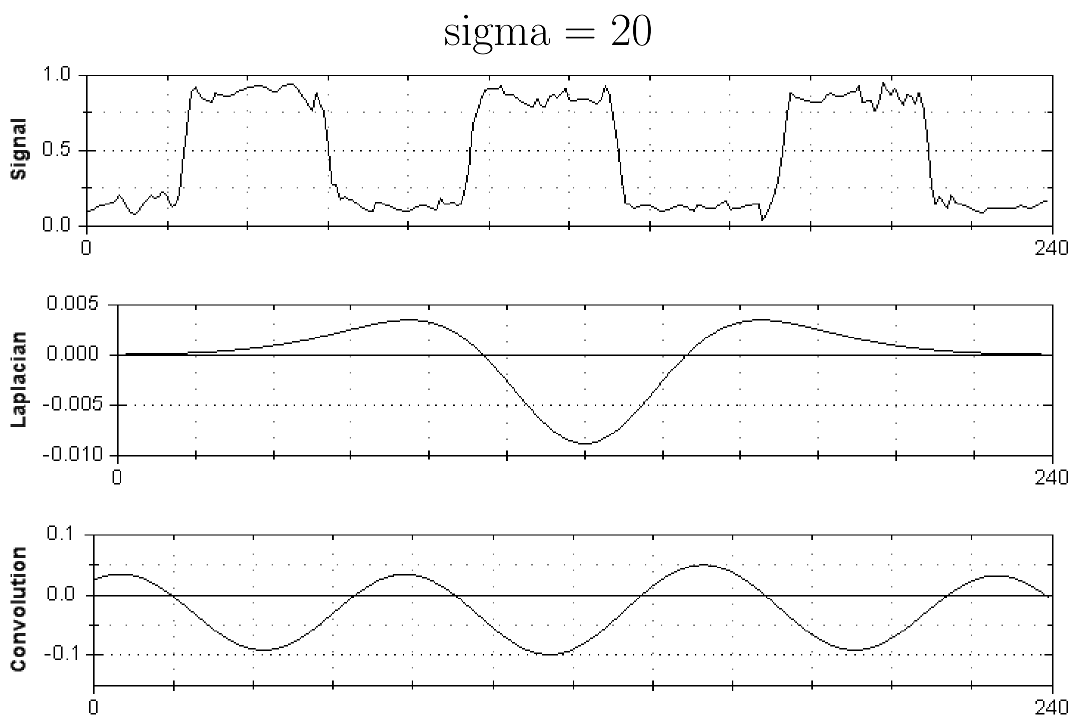
Blobs

A *blob* is an area of uniform intensity in the image. Whereas edges and corners are features which are found at discontinuities, blobs are localised in the middle of areas of similar intensity which are surrounded by pixels of a different intensity on their boundaries.



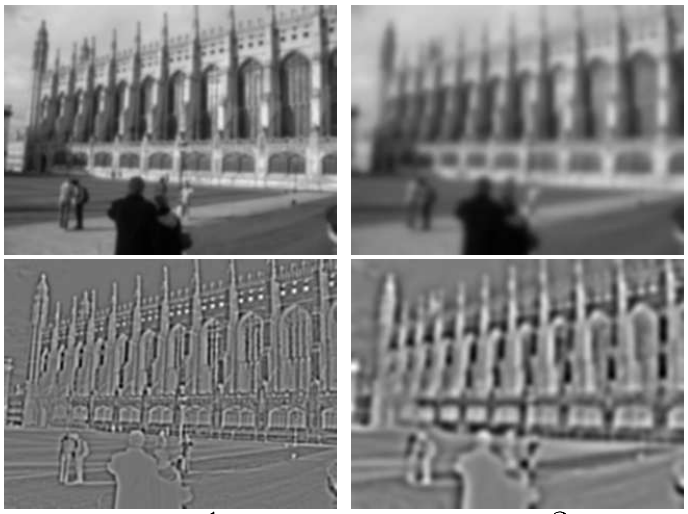
Polka Dots Detected Blobs

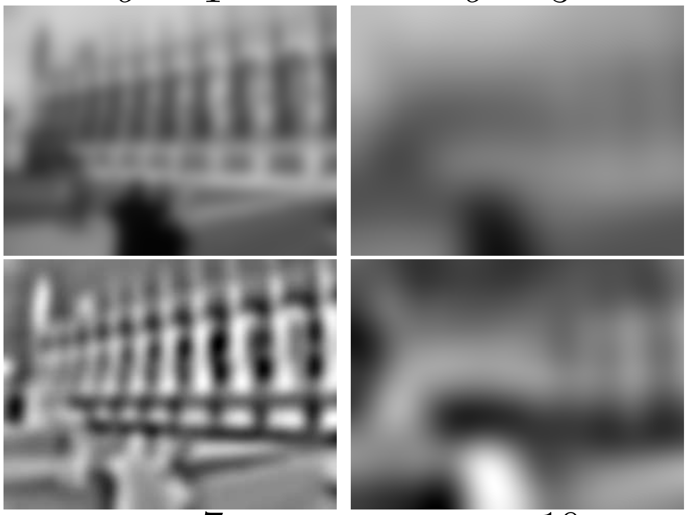
The 1D signal is a scan line running across one of the polka dots above. The result shows how, even though the signal is quite noisy, the local extrema of the convolution with **Laplacian of the Gaussian** at the correct scale, σ, localises the centre of the dots perfectly.



Blobs and Band-pass filtering

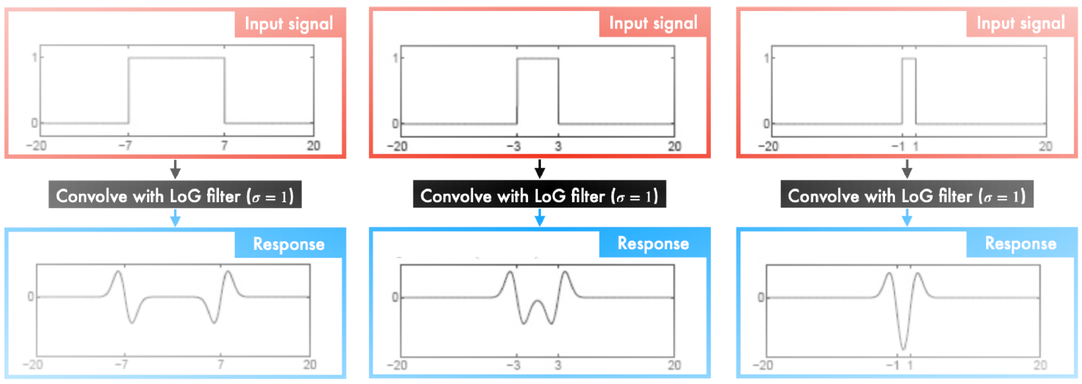
The size of the blob detected depends on the sigma of the detector used. As the sigma is increased, larger and larger image features are detected, ranging from small boxes to entire buildings. Each time the blob detector will fire on the center of the blob in question, making it ideal for extracting texture from the inside of an object or for fixing location of an object in the scene.


$\sigma = 1$ $\sigma = 3$

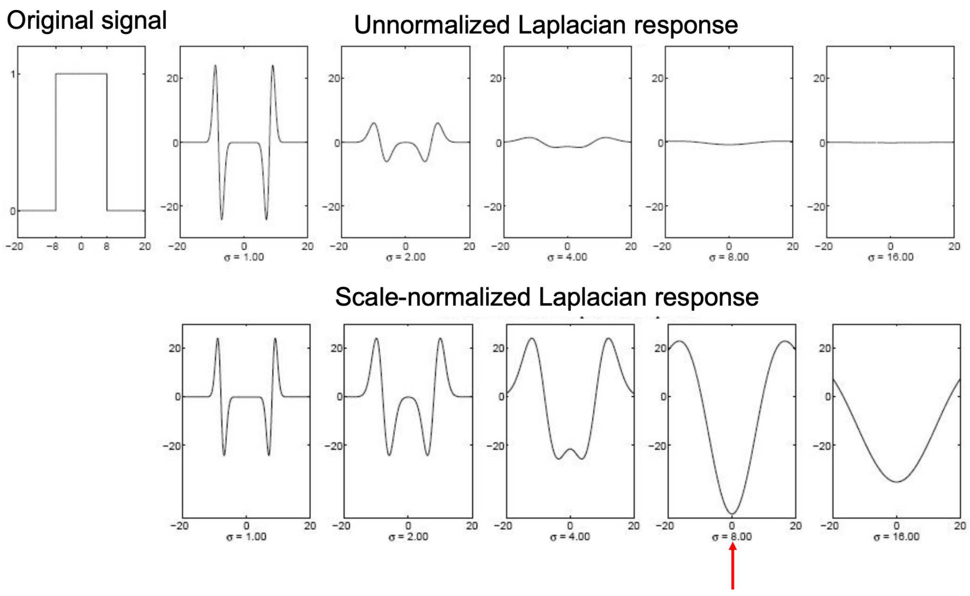

$\sigma = 7$ $\sigma = 19$

From edge to bar/blob detection

Convolution with the Laplacian of a Gaussian (LoG) kernel produces a **band-bass** filtered output with zero-crossings at intensity edges. When the scale, σ, is *matched* to the size of the pulse (bar) in 1D there is a strong response with an extrema at the centre of the pulse/bar.

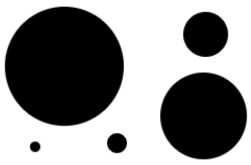


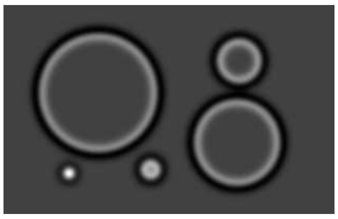
The response falls with increasing scale, σ, and needs to be scale-normalised (by convolution with $\sigma^2\nabla^2 G_\sigma$ instead) so that the strongest response corresponds to the centre of bars at different scales.



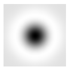
Finding blobs at different scales

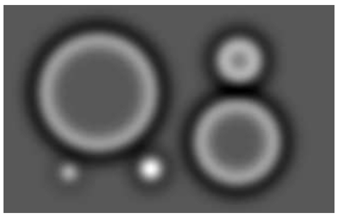
In 2D the maximum of the magnitude of the scaled Laplacian of Gaussian response localises the centre of a blob whilst the scale determines its size (radius of blob is $\sqrt{2}\sigma$).

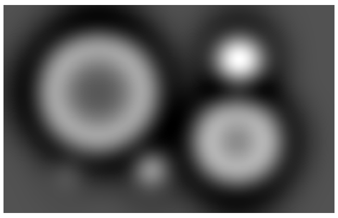

Original

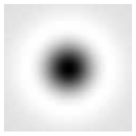

Level 0

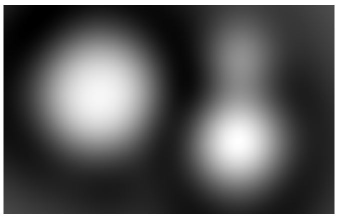
$\sigma = 5$

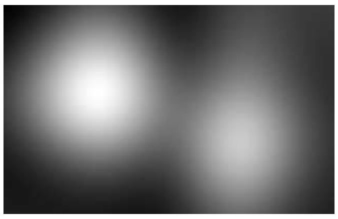

$\sigma = 10$

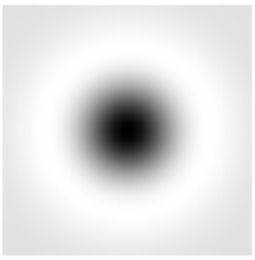

Level 1

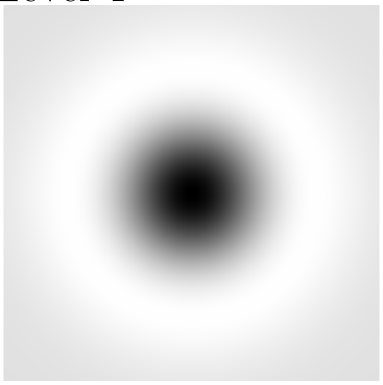

Level 2


$\sigma = 20$


Level 3

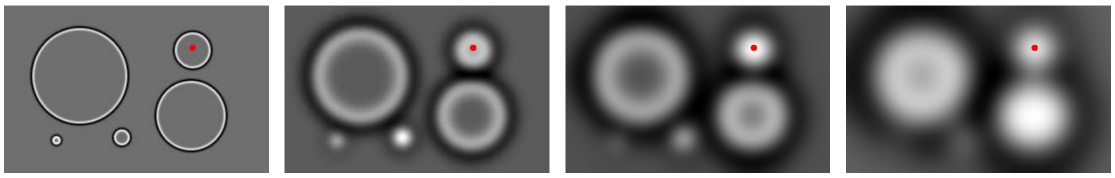

Level 4


$\sigma = 40$

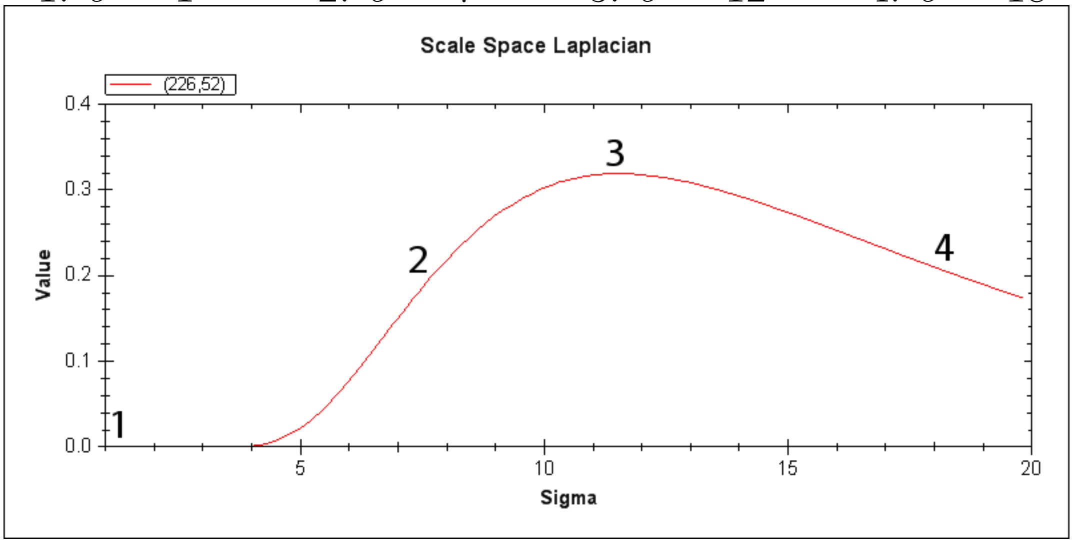

$\sigma = 80$

Finding the scale of a blob

The (scale-normalised) Laplacian of a Gaussian response as recorded at a particular location is a smooth function over scale. The ideal scale of a keypoint is the scale corresponding to the maximum of the scale space function at that point. For example, with a blob, we would want to find the maximum of the magnitude of the Laplacian of a Gaussian over scale. The image location of this local max response gives the blob centre position whilst the scale, σ, defines its size (radius of blob is $\sqrt{2}\sigma$).



1: $\sigma = 1$ 2: $\sigma = 7$ 3: $\sigma = 12$ 4: $\sigma = 18$



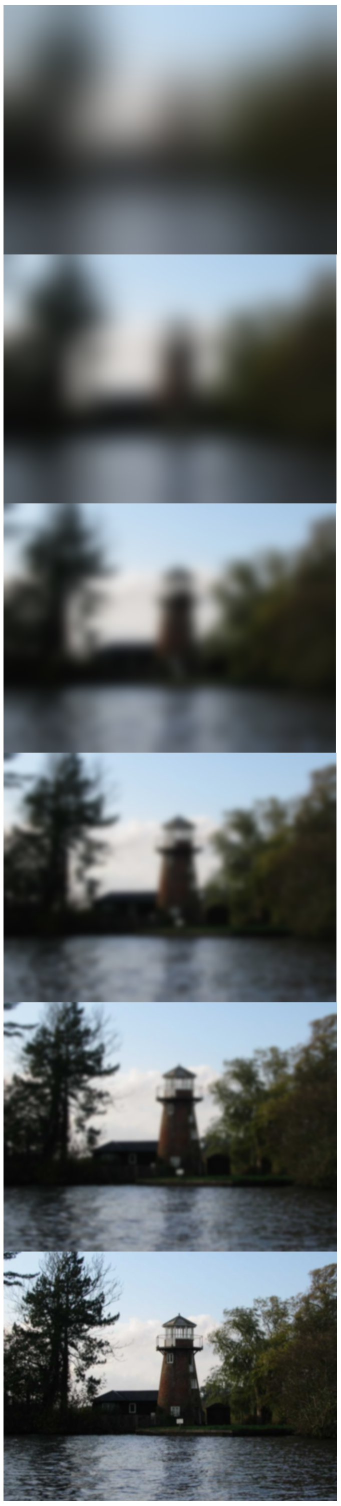
To help find the exact point and scale efficiently, a set of discrete scales are sampled (from an Image Pyramid) and the largest value is found by interpolation and then the exact point interpolated.

Scale Space

We achieve scale independence by looking at the different resolutions (low-pass filtered at different scales) of an image. There are an infinite number of possible resolutions for any image, a three-dimensional function of intensity over location and scale known as the **scale space** of the image, denoted $S(x, y, \sigma)$. This can be calculated by convolving the original image $I(x, y)$ with Gaussians of different scale σ, thus the scale space function can be written as

$$S(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$
$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

It is impractical to examine all possible resolutions, and indeed impossible to do so when we are restricted by digital image representation. Thus, we sample the space by choosing particular resolutions to examine.



Scale Space, cont.

We produce a discrete set of low-pass filtered images by smoothing with gaussians with a scale satisfying

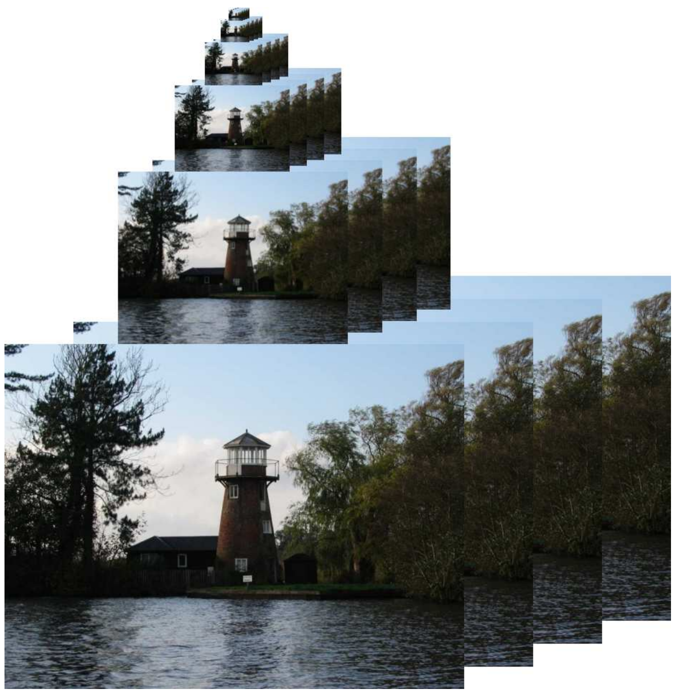
$$\sigma_i = 2^{\frac{i}{s}}\sigma_0$$

so that it doubles after s intervals. The s images in each octave are spaced logarithmically with the scale of neighbouring images satisfying

$$\sigma_{i+1} = 2^{\frac{1}{s}}\sigma_i$$

Blurring with large scales is avoided in 2 ways: **subsampling** the image after each octave (i.e. scale has doubled) and by using a finite set of **incremental Gaussian blurrs**.

The resulting sampling of spaces is called an **image pyramid**, for which we compute scales ranging from σ to 2σ (an octave), and then subsample the image for the next octave.



Scale Space, cont.

Within each octave, as we convolve images repeatedly with Gaussian filters, the resulting image has a σ calculated as

$$G(\sigma_1) * G(\sigma_2) = G\left(\sqrt{\sigma_1^2 + \sigma_2^2}\right)$$

At every interval i of the pyramid, we want $\sigma_i = 2^{\frac{i}{s}}\sigma_0$ (so that it doubles after s intervals). We need to achieve this incrementally, thus

$$G(\sigma_{i+1}) = G(\sigma_i) * G(\sigma_{k_i})$$

So, what is σ_{k_i}?

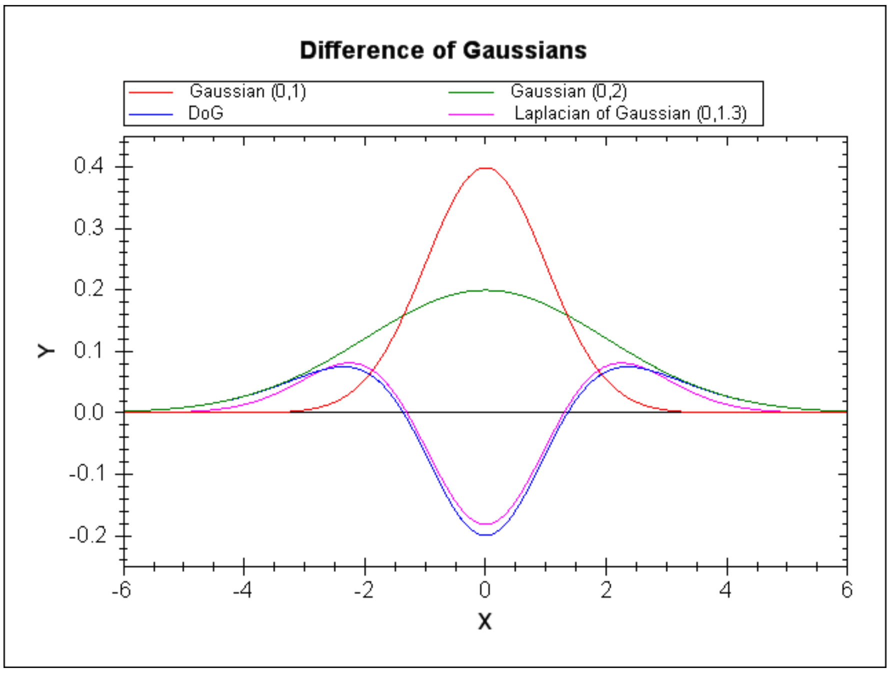
$$\begin{aligned}
\sigma_{k_i} &= \sqrt{\sigma_{i+1}^2 - \sigma_i^2} \\
\sigma_{i+1} &= 2^{\frac{1}{s}}\sigma_i \\
\sigma_{k_i} &= \sqrt{2^{\frac{2}{s}}\sigma_i^2 - \sigma_i^2} \\
\sigma_{k_i} &= \sigma_i\sqrt{2^{\frac{2}{s}} - 1}
\end{aligned}$$

The s distinct and small incremental Gaussian (low-pass) filters, σ_{k_i}, need only be computed once and can be reused in each subsequent octave but on sub-sampled images to achieve the larger scales.

Difference of Gaussian

The **difference of Gaussians** interest point, or DoG as it is often called, is a blob detector. The points are taken from the minima and maxima of the DoG response over an image. It takes its name from the fact that it is calculated as the difference of two Gaussians, which approximates the scale-normalised Laplacian of a Gaussian (see Lowe reference).

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G(x, y, \sigma)$$



In a system which utilizes a scale space pyramid (such as the one we will consider), the DoG point is a useful entity, as a response can be computed simply subtracting one member of a pyramid level from the one directly above it.

Keypoint detection and scale

Keypoint locations (the blob centres) are found by first computing an approximation for the Laplacian of the Gaussian pyramid by using Difference of Gaussians - i.e. subtracting neighbouring images of same dimension in the Image Pyramid. The image location of the local maximum/minimum of the this response (in image position and over scale) gives the feature/keypoint centre and characteristic scale. This will require a local search of 26 neighbour responses to determine if a pixel is a blob-centre and to find the scale.

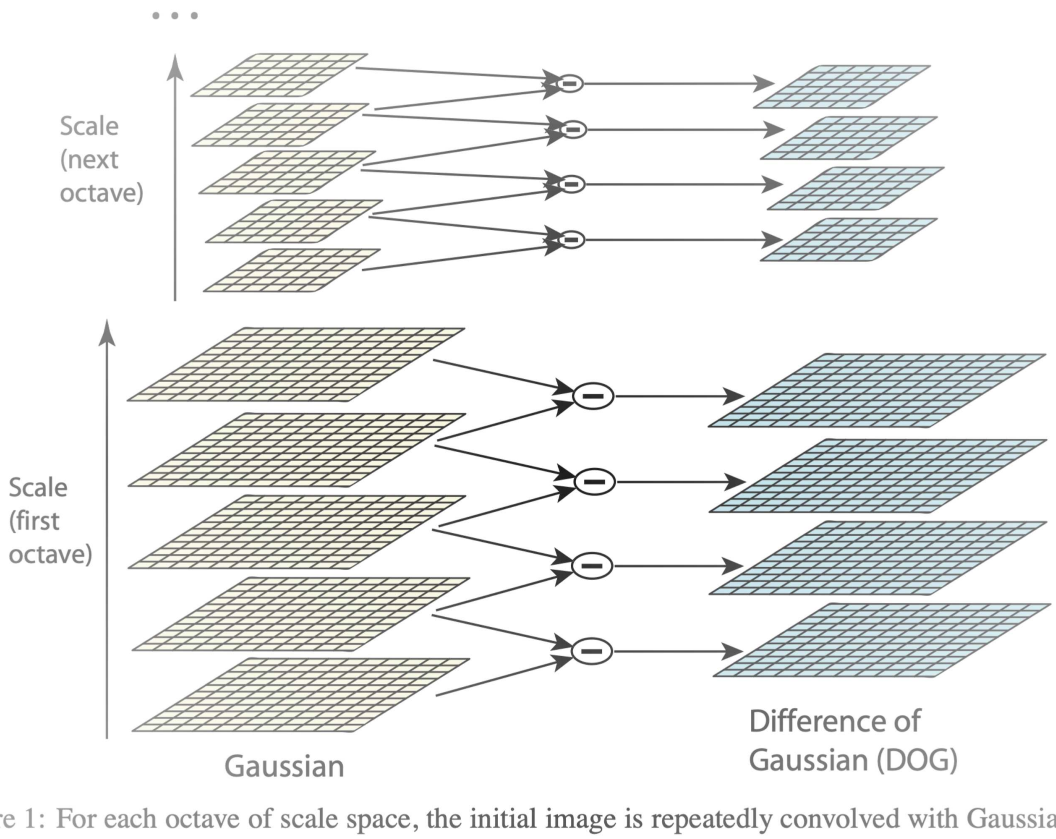
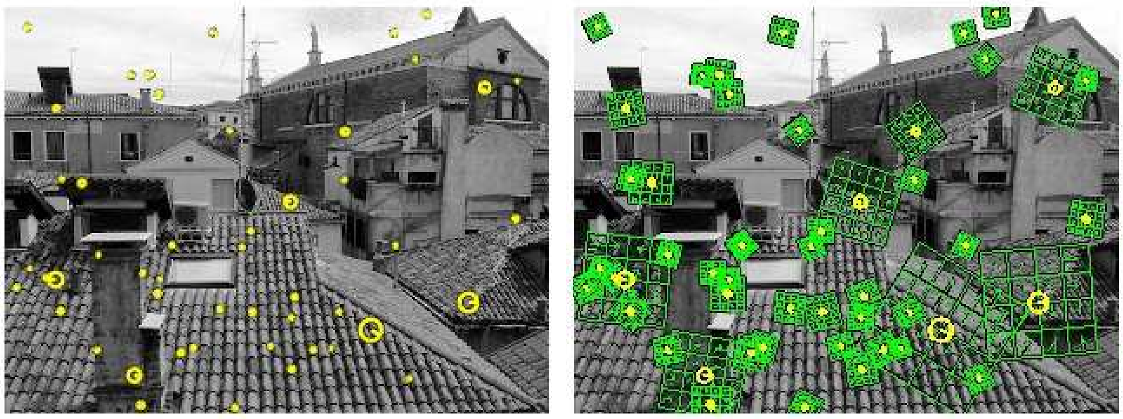


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

Keypoint detection - scale/orientation

For image matching, blob centres give keypoints which are particularly useful features to concentrate on, as they are usually found inside of objects as opposed to at their edges and thus are less likely to contain part of the background in queries, in addition to other properties such as stability, repeatability, and a definite optimal scale.

The blob centre gives the keypoint image location and the blob scale can be used to normalise for the size/scale of the image feature by sampling the pixels (typically 16×16) in the neighbourhood of the blob centre at the appropriate image of the Image Pyramid.

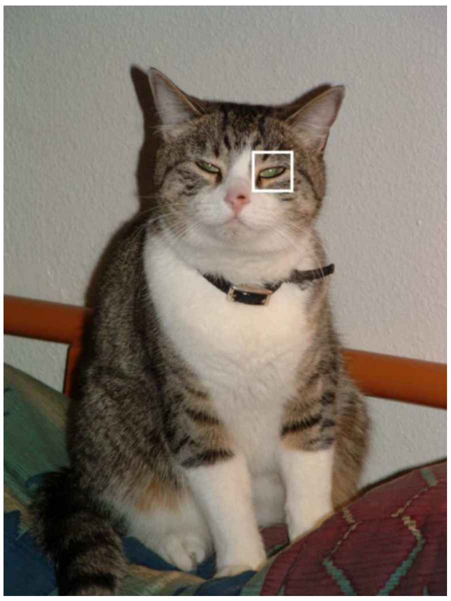


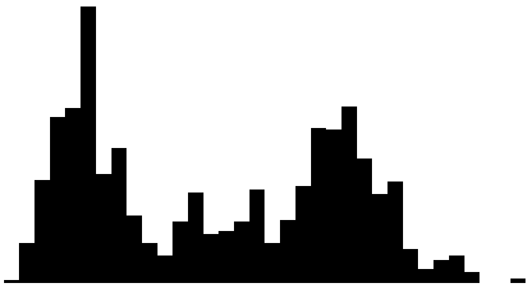
To compute the orientation of the keypoint we begin by computing the gradients (orientation and magnitude) for each pixel. We find the keypoint's **dominant orientation** by looking at the **histogram of oriented gradients** in the patch of pixels around the keypoint centre.

Keypoint dominant orientation

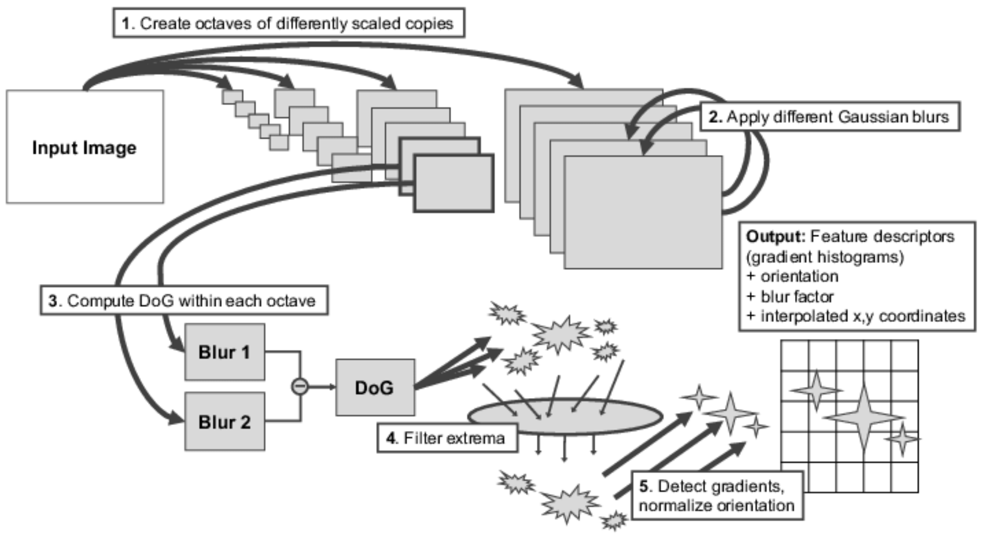
We can build a histogram (typically with 36 bins covering 360 degrees) of all of the edge orientations weighted by their gradient magnitudes in the neighbourhood (typically 16×16) of the keypoint. This needs to be smoothed (low-pass filtered with a 2D gaussian of size 1.5σ scale for the keypoint).

The highest peak in the histogram will approximate the dominant orientation - a better estimate can be found through interpolation (by fitting a parabola to the values of the bin and its two neighbours). If there is no clear maximum, then the keypoint/interest point is given several dominant orientations (i.e. several copies of the keypoint/interest point with different orientations are used).


Source Image


Orientation Histogram

Keypoint detection - scale/orientation


1. Create octaves of differently scaled copies
Input Image
2. Apply different Gaussian blurs
Output: Feature descriptors (gradient histograms)
+ orientation
+ blur factor
+ interpolated x,y coordinates
3. Compute DoG within each octave
Blur 1
Blur 2
DoG
4. Filter extrema
5. Detect gradients, normalize orientation

To produce a descriptor which is invariant to scale we first normalize for size/scale in the image (by sampling 16×16 patch of image intensities at the appropriate level of the image pyramid)

We can then normalize for image orientation (by sampling pixels after computing a characteristic reference/dominant orientation for the patch of pixels using a histogram of gradients).

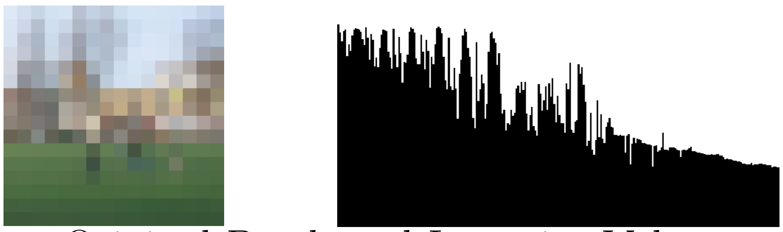
After this geometric size and orientation normalisation we can then compute an appropriate descriptor for each feature by either raw or normalised intensities, edges or the Scale Invariant Feature Transform (SIFT).

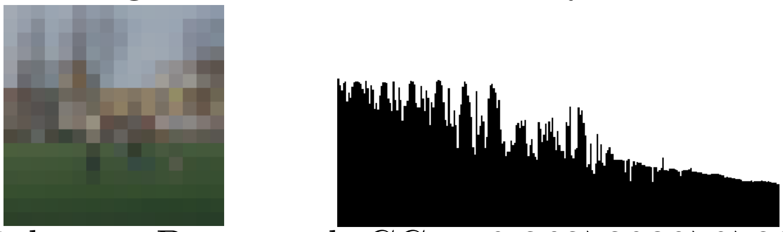
Matching intensity patches

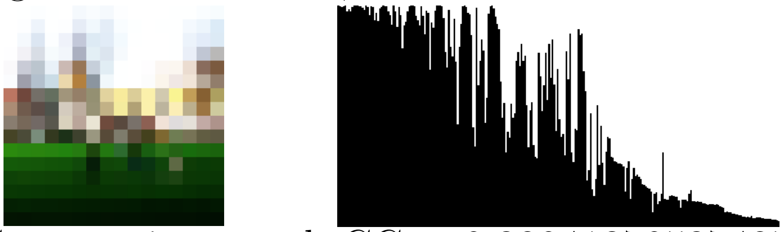
The simplest way to describe a patch of an image is just to store the N intensity values, $P[i]$. You can then compare patches directly using cross correlation to find a match.

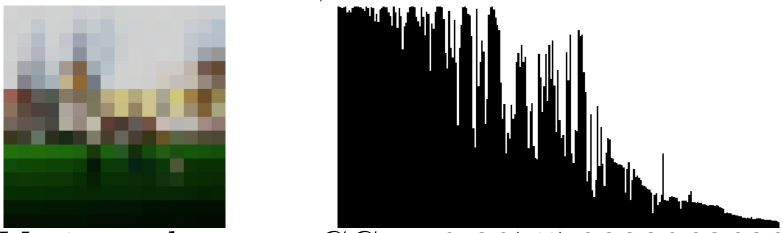
$$CC(P_1, P_2) = \sum_{i}^{N} P_1[i] P_2[i]. \tag{1}$$

In this raw form the description is not very robust to changes, however.


Original Patch and Intensity Values


Brightness Decreased, CC = 0.262720397078039


Contrast increased, CC = 0.380413705374859


Various changes, CC = 0.297579822063629

Zero Normalized Patches

Brightness changes are essentially changes in the mean brightness value. While the mean changes, the distribution of the intensity values around the mean stays the same. Thus, by giving the intensity values a zero mean, they become immune to brightness change.

$$\mu = \frac{1}{N}\sum_{x,y} I(x,y)$$
$$Z(x,y) = I(x,y) - \mu$$

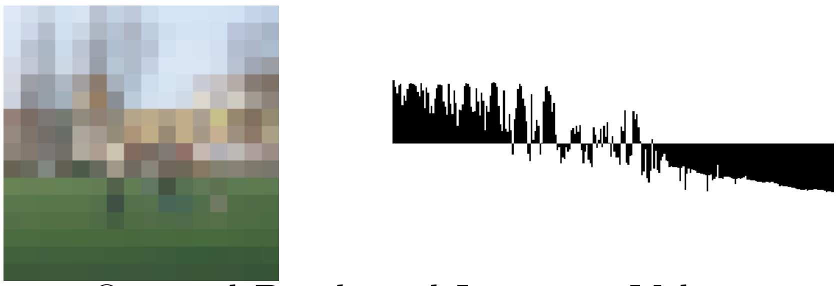
However, the intensity values are still affected by contrast changes. Contrast is essentially a change in the variance of the distribution of the intensity values around the mean.

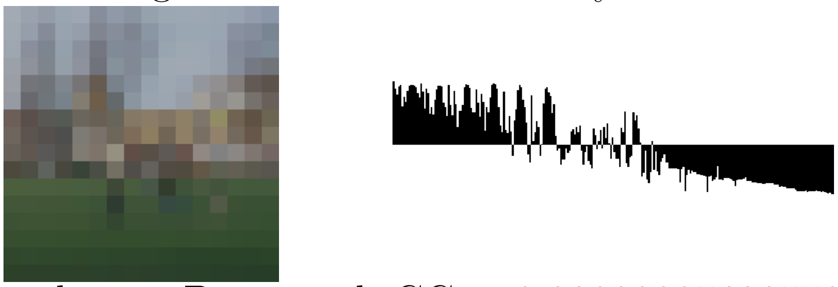
Thus, to deal with contrast all that is required is to divide each value by the standard deviation of the intensity value distribution.

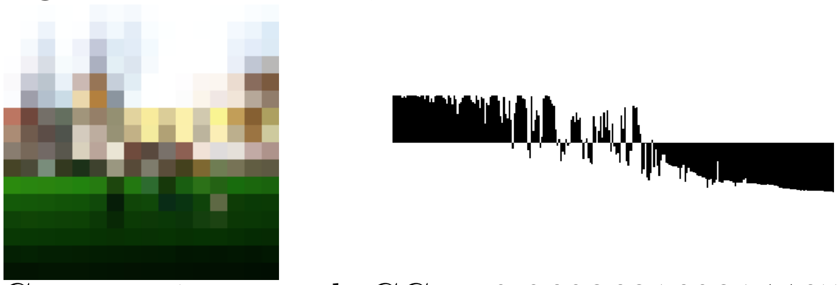
$$\sigma^2 = \frac{1}{N}\sum_{x,y} Z(x,y)^2$$
$$ZN(x,y) = \frac{Z(x,y)}{\sigma}$$

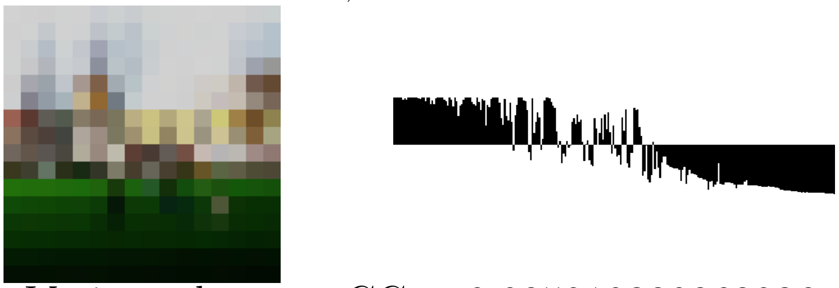
Zero Normalized Patches, cont.

The resulting collection of zero-mean, unit variance intensity values is known as a zero-normalized patch, and can be accurately matched using simple cross-correlation. The size of the descriptor grows with the size of the patch, and thus can be quite big. Even so, while not a data reduction, it is a useful way to represent these areas.

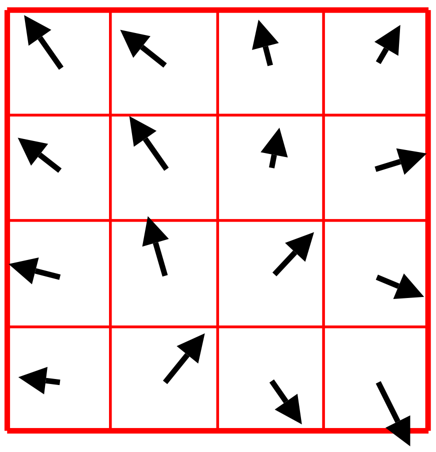

Original Patch and Intensity Values


Brightness Decreased, CC = 0.999988956295594

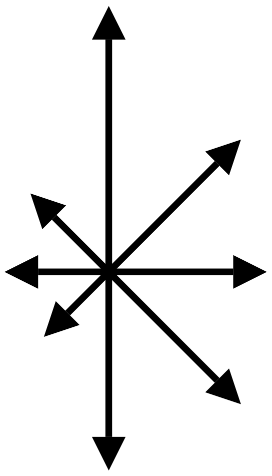

Contrast increased, CC = 0.969868160814465


Various changes, CC = 0.985010389868036

Histogram of Gradients


Gradient Grid

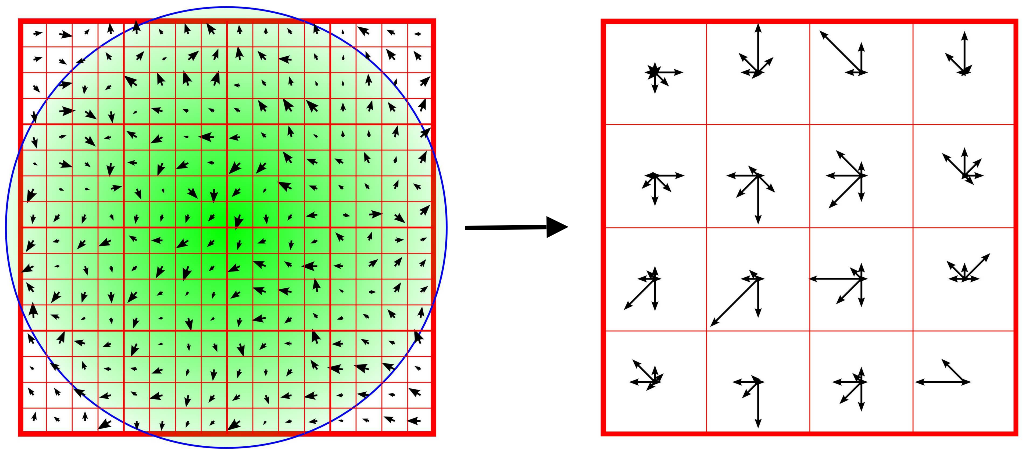
If you look at the gradient of each pixel in the patch, each will have its own distinct orientation/direction, or way that it is facing, and each will have a size/strength (gradient magnitude).



These can in turn be binned together into an orientation histogram, as shown on the right. Since this histogram is built using gradients/edges, which are robust to contrast and brightness changes and can be detected at different scales, and also incorporates orientation data (thus adding robustness to orientation) this makes them a very strong candidate for a descriptor.

The SIFT keypoint descriptor

SIFT stands for **S**cale-**I**nvariant **F**eature **T**ransform. It uses a collection of orientation histograms to create a robust and descriptive representation of a patch. This $N \times N$ patch (typically, $N = 16$) is extracted at the scale of the interest point. Thus, while the patch size never changes the area of the actual image it represents changes depending on scale.



The $N \times N$ patch is split into c cells, and within each cell the intensity gradient at every pixel is calculated and the directions binned into a histogram weighted by their magnitude and a Gaussian window with a σ of .5 times the scale of the feature centered on the patch. This weights the inner pixels (those closer to the interest point) to avoid possible occlusion problems.

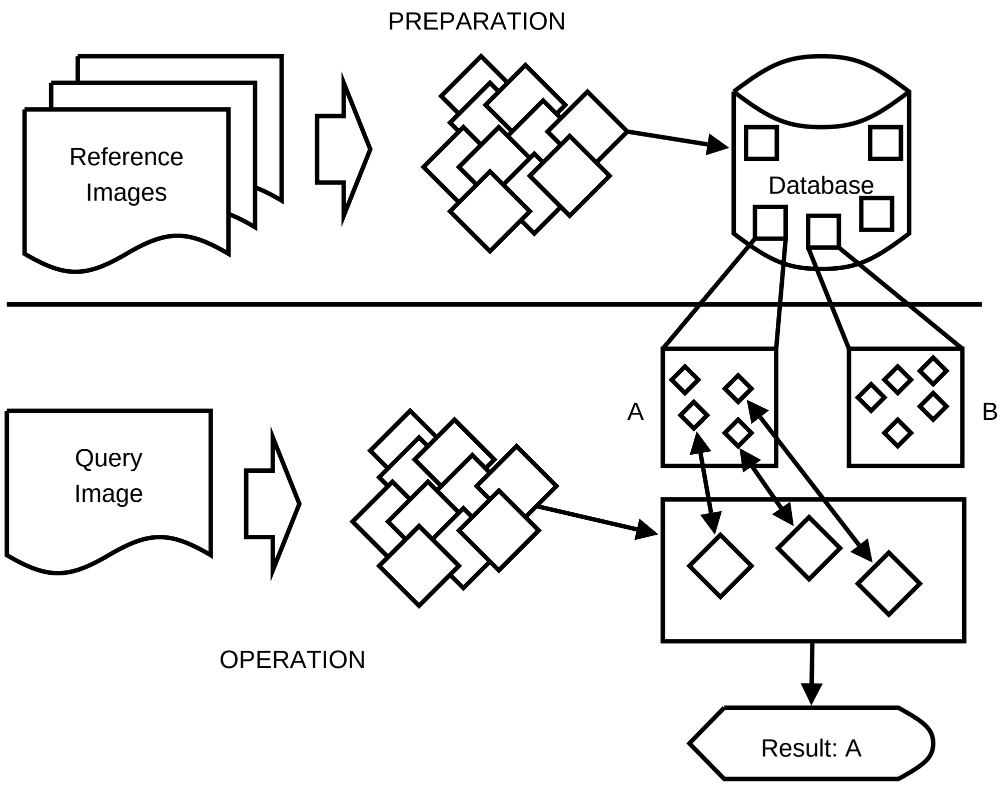
The SIFT keypoint descriptor

If the bins are centered on d directions (typically 8) in each of c cells (typically 16) , the resulting descriptor is a $d \times c$ vector (typically **128**D).

By dividing the patch into cells, a particular gradient can move around to some degree within the descriptor window and still contribute to the same directional histogram. Once the $d \times c$ vector has been extracted, it is normalized to provide invariance to gradient magnitude change. One final step is performed to help minimize the effects of non-affine lighting changes by thresholding so that all values in the unit vector are less than .2 (to reduce the effect of single elements such as those coming from very strong highlights) and then renormalizing.

Matching features over multiple views

We can use our design of keypoints and their descriptors to build a system to recognize a target object (specified by a reference image) in another viewpoint (query image):



So, one way of solving the correspondence problem is to search through all the feature points (keypoint descriptors) in the database images for the best match of a query feature.

A good match is usually defined as one which is small distance away in feature descriptor space (d=128 for SIFT) as measured by an Euclidean distance metric: following formula

$$E(\vec{x}, \vec{y}) = \sqrt{\sum_d (\vec{x}_d - \vec{y}_d)^2}$$

Finding correspondences - Search

This is called a **linear search**, and is prohibitively expensive to compute. Instead, we must find a way of storing the data to make searching faster.

A data structure organizes data such that it is more efficient to store, access and search. The simplest data structure is a list of items, such as an array of numbers. The most complex are almost impossible to visualize. One solution is to use tree-based data structures such as k-d trees to tackle the problem of nearest neighbour retrieval.

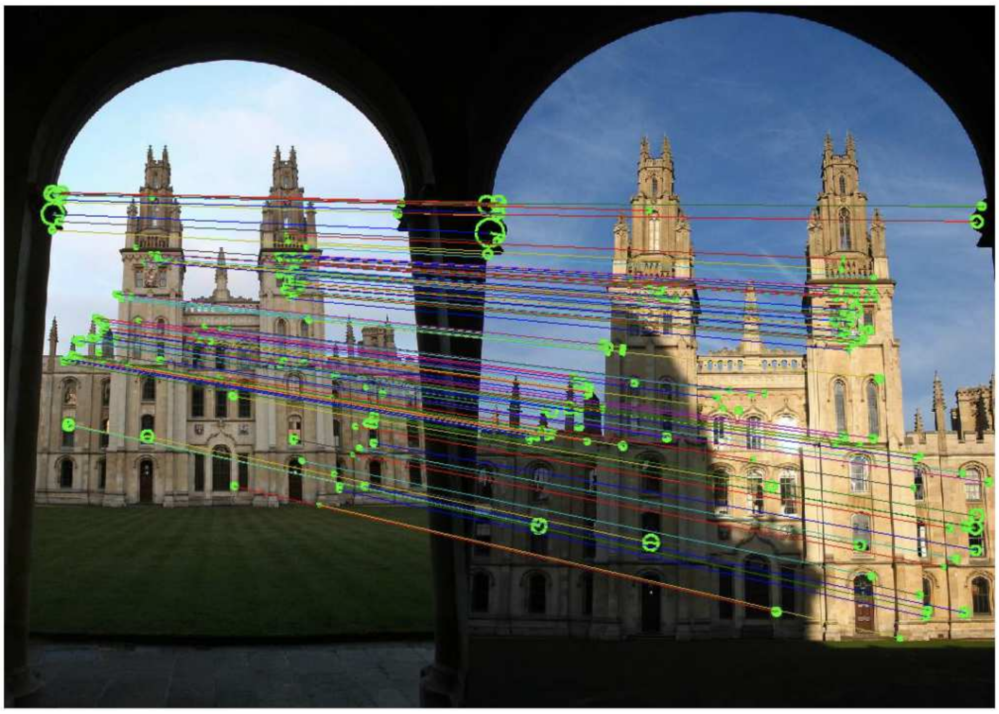
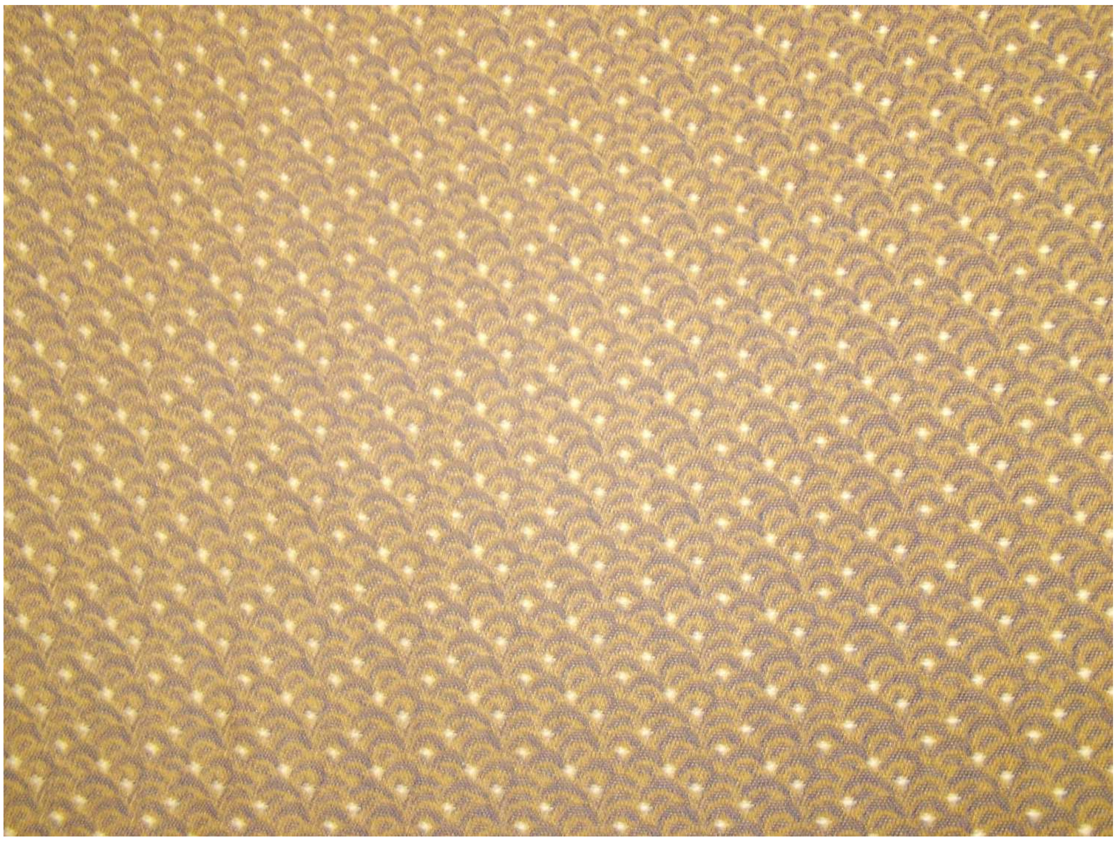


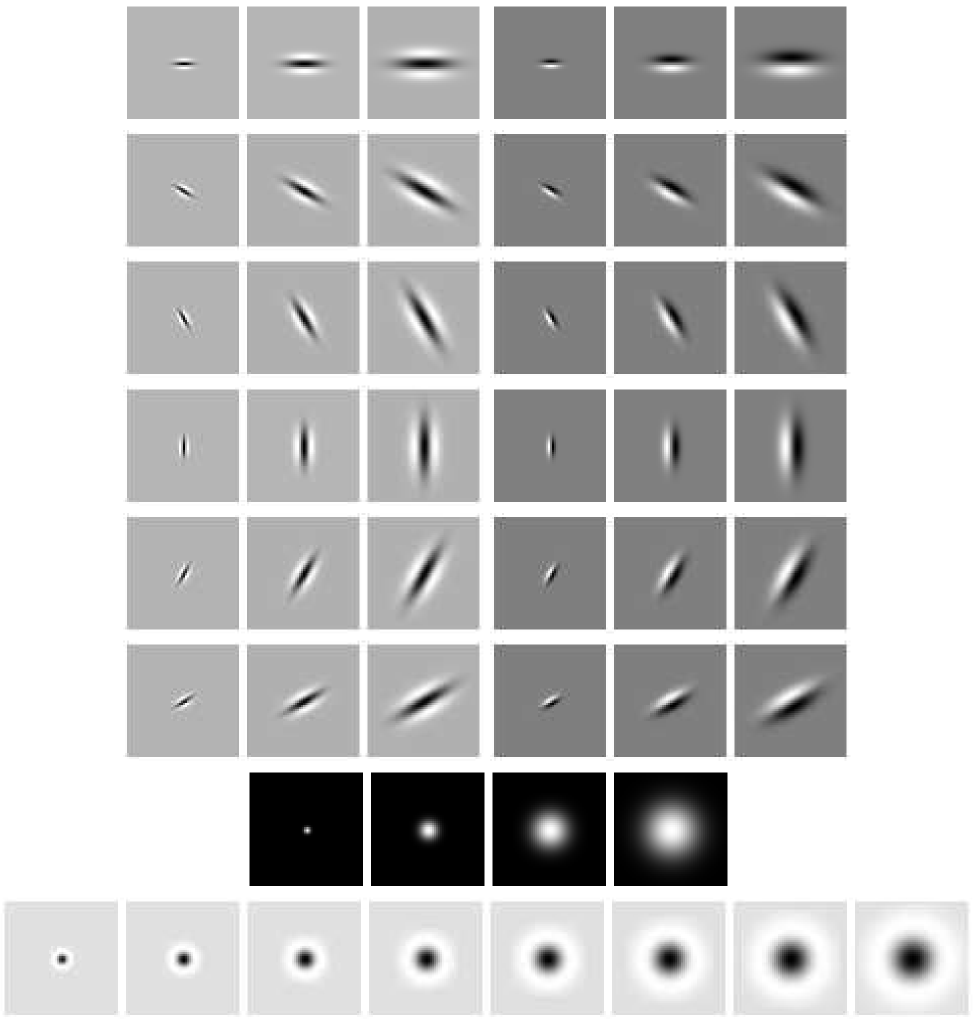
Image Structure: Texture

Image texture arises from large numbers of small objects such as grass, brush, pebbles and hair and surfaces with orderly and repetitive patterns such as the spots or stripes on animals, wood and skin. They typically consist of organised patterns of regular sub-elements called *textons*. A natural way to describe texture is to find these textons and describe how they are distributed.

Image textures can be described by their response to a collection of filters to represent the patterns of spots, bars etc.



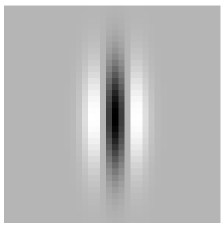
Characterising Texture

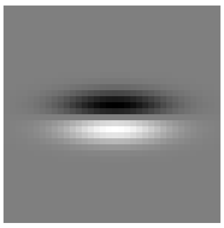


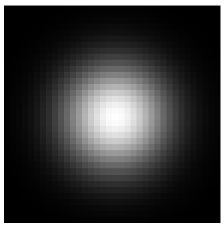
This is an example filter bank. It consists of 8 Laplacian of Gaussian filters and 4 Gaussian filters at different scales to provided non-oriented responses, and 36 oriented filters at 6 different angles, 3 different scales, and 2 different *phases*. The two phases of oriented filters are first and second derivatives of Gaussians on the minor axis and elongated Gaussians on the major axis, and thus detect edges or bars respectively along their major axes.

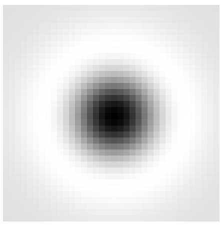
The descriptor is simply the concatenated responses of all of the filters in the filter bank at a pixel.

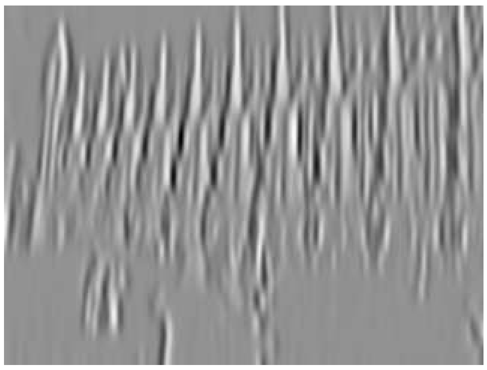
Filter Banks

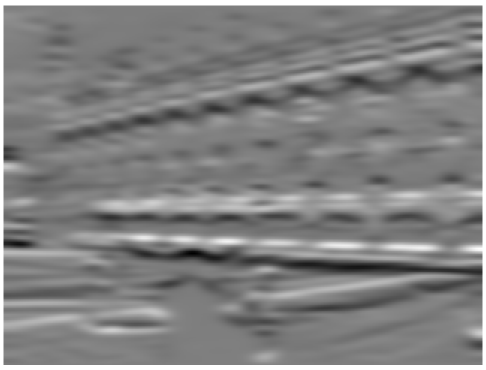

Bar

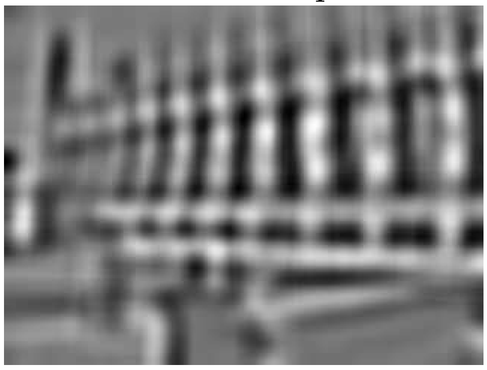

Edge

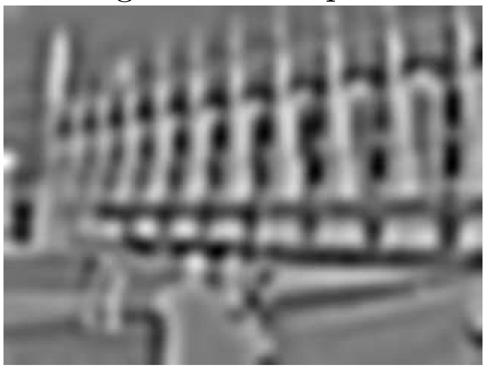

Brightness


Blob


Bar Filter Response


Edge Filter Response


Intensity Filter Response


Blob Filter Response

Learning feature hierarchies

In the previous sections the raw image has been pre-processed through hand-crafted feature extraction (edges, corners, textons). The features were not learned. In many recognition tasks in computer vision we will learn a hierarchy of features just by looking at examples - from low level to mid-level invariant representations to object identities. This is called **Deep Learning**.

Bibliography

Some of the illustrations in this handout were taken from the following publications, which also make good further reading.

Edge detection

J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

Corner detection

C. Harris. Geometry from visual motion. In A. Blake and A. Yuille, editors, *Active Vision*, pages 263–284. MIT Press, Cambridge MA, 1992.

Blobs and SIFT descriptor

D.G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.