Image Structure 2

Feature Detection and Matching
4F12: Computer Vision

Instructor: Samuel Albanie
Based on course material authored by Roberto Cipolla

VE Tag = Very Examinable
NE Tag = Non Examinable

Recap of last lecture

Summary

- How to represent images as matrices
- Nuisance factors in pixel intensity data
- Data reduction in computer vision and Marr's hierarchy
- Image structures: featureless regions, edges and corners
- Edge detection in 1D (and how to do it quickly)
- Edge detection in 2D (and how to do it quickly)
- Implementation details (truncated summations; convolution)

The Aperture Problem

The problem with edges

Suppose you are asked to look down through an opening and observe a grate moving below you. Which way is the grate moving?

Down and to the right? Straight down? Only to the right?

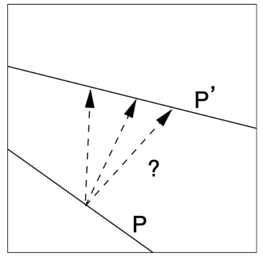
It is impossible to tell!

Can only measure motion normal to the edge

While edges are a powerful intermediate representation, they are sometimes insufficient.

This is especially the case when image motion is being analysed.

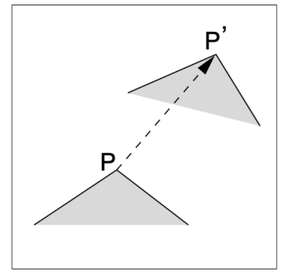
The motion of an edge is rendered ambiguous by the aperture problem: when viewing a moving edge, it is only possible to measure the *motion normal to the edge* locally.



Edge

Corners to the rescue

To measure image motion in 2D completely, we can look at corner features.



Corner

We saw earlier that a corner is characterised by an intensity discontinuity in two directions (this discontinuity can be detected using correlation).

Image credits: "Animated Example of the Aperture Problem", Bas Rokers. https://en.wikipedia.org/wiki/Motion_perception#/media/File:Aperture_problem_animated.gif

Cross-correlation - another important operator

Normalised cross-correlation

The normalised cross-correlation function measures how well an image patch $P(u,v)$ matches portions of an image, $I(x,y)$, that share the same size as the patch.

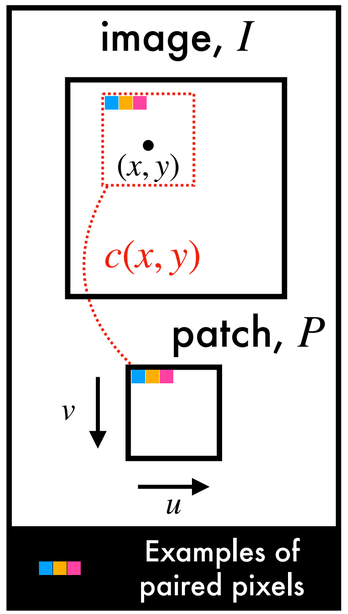
It entails *sliding the patch over the image*, computing the sum of the products of the pixels and normalising the result:

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

covariance

variance of patch

variance of image under patch



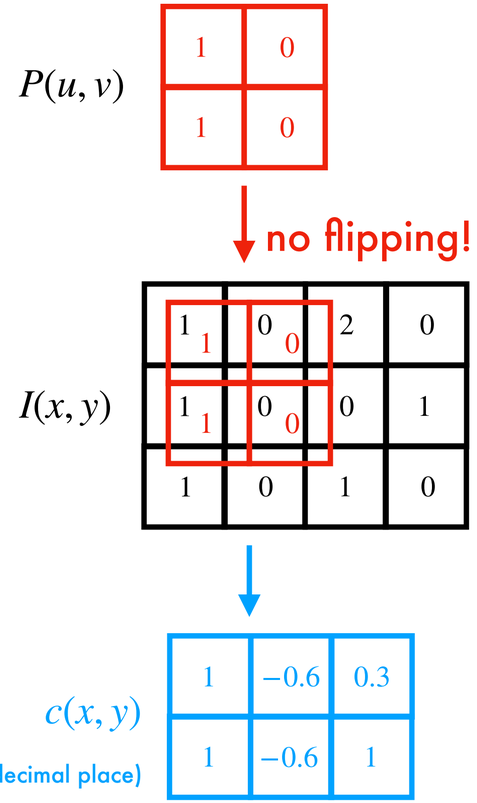
$\bar{P}$ is the mean pixel value of the patch

$\bar{I}_{x,y}$ is the mean pixel value of the image under the patch

If we multiple numerator & denominator of $c(x,y)$ by $1/n^2$, we can interpret terms as the covariance, variance of the patch and variance of the image under patch.

Note that the cross-correlation is normalised to $[-1,1]$ by computing it from the covariance and variances of the two signals/patches (adds robustness to *illumination* changes)

Normalised Cross-correlation: visualised



Note: it is common in software implementations to pad the edges of $I(x,y)$ with zeros so that $c(x,y)$ is the same shape as (unpadded) $I(x,y)$.

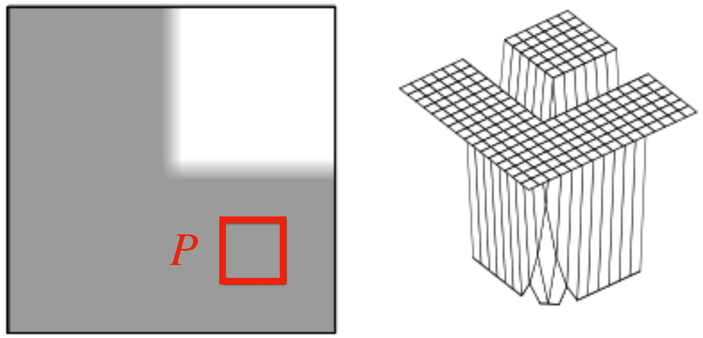
Cross-correlation - corners

cross-correlation peaks at corners

A patch which has a well-defined **peak** in its autocorrelation (self cross-correlation) function can be classified as a "corner".

Featureless

Auto-correlation with featureless patch

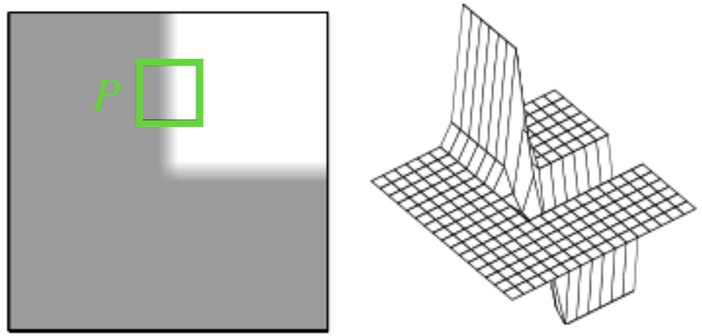


$I(x, y)$ $c(x, y)$

No clear peak in $c(x, y)$

Edge

Auto-correlation with an edge patch

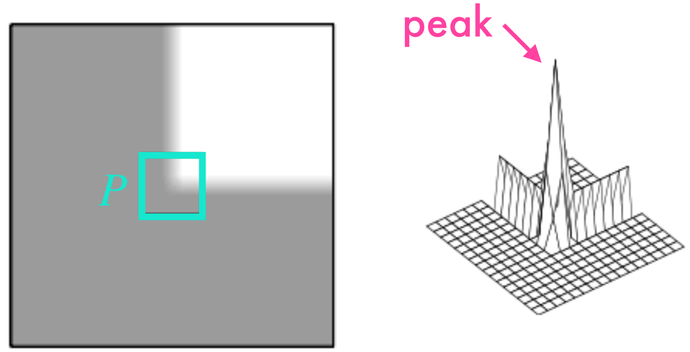


$I(x, y)$ $c(x, y)$

Produces ridge in $c(x, y)$

Corner

Auto-correlation with a corner patch



$I(x, y)$ $c(x, y)$

Produces clear **peak** in $c(x, y)$!

The link between sum of squared differences and cross-correlation

Definitions

The *sum-of-squared-differences (SSD)*, or *squared Euclidean distance*, is a popular metric for comparing patch similarity.

It is computed between a patch $P(u, v)$ containing $(2n + 1) \times (2n + 1)$ pixels, and another of the same size in an image $I(x, y)$ via:

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

The (*unnormalised*) *cross-correlation* (a simpler variant of the formula we met earlier) is given by:

$$UCC(x, y) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} P(u, v) I(x + u, y + v)$$

If we expand the expression for $SSD(x, y)$, we obtain:

$$SSD(x, y) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} \underbrace{P(u, v)^2}_{\text{constant}} - 2P(u, v)I(x + u, y + v) + \underbrace{I(x + u, y + v)^2}_{\text{approx. constant}}$$

The link

To see the link, note that:

1. The first patch term in $SSD(x, y)$, $P(u, v)^2$, is constant w.r.t x, y.
2. In *natural images* (captured from the real world), pixel values often vary *smoothly*, and so we can approximate the last term in $SSD(x, y)$, $I(x + u, y + v)^2$, by a constant (when summing across u, v, this term will have significant overlap for neighbouring x, y).

With these observations, we have that:

$$SSD(x, y) \approx -2 \cdot UCC(x, y) + constant$$

Thus, we see that *greater cross-correlation* implies *greater similarity* (a smaller distance) under the SSD metric.

Useful reference: http://www.cse.yorku.ca/~kosta/CompVis_Notes/ssd_cross-correlation.pdf

Corner Detection

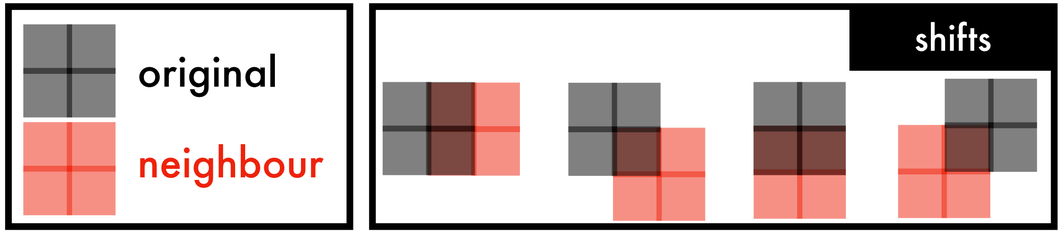
Corner detection challenge

Key challenge: A practical corner detection algorithm needs to do something more efficient than calculate full auto-correlation functions for every single pixel (cost is quadratic in the number of pixels).

However, we can compute correlations with "local" (nearby) patches (this approach motivates corner detector algorithms).

Moravec Corner Detector

An early corner detector proposed by Moravec computes the *sum-of-squared differences* between a patch and its immediate neighbours (with horizontal, vertical and diagonal shifts), keeping the minimum value.



Corners are identified as local maxima across the image.

NE

Harris Corner Detector: motivation

Harris and Stephens (1988) noted several areas with room for improvement in the Moravec corner detector:

- Only shifts at 45° are considered (what if the corner is at a different angle?)
- Detection was too noisy
- The detector was too sensitive to edges

References: Moravec, Hans Peter. Obstacle avoidance and navigation in the real world by a seeing robot rover. Dissertation. Stanford University, 1980.
Harris, Chris, and Mike Stephens. "A combined corner and edge detector." *Alvey vision conference*. 1988

Corner Detection (Harris)

Harris Corner Detector: steps 1 & 2

1. First (similarly to edge detection) we smooth the image $I(x, y)$ to obtain $S(x, y)$, to allow us to take gradients.

2. Calculate the change in intensity in an arbitrary direction $\mathbf{n}$:

$$S_{\mathbf{n}} \triangleq \frac{\nabla S(x, y) \cdot \mathbf{n}}{|\mathbf{n}|}$$

where $\nabla S = \begin{bmatrix} S_x \\ S_y \end{bmatrix}, \quad S_x = \frac{\partial S}{\partial x}, \quad S_y = \frac{\partial S}{\partial y}$

$$S_{\mathbf{n}}^2 = \frac{(\mathbf{n}^T \nabla S)(\nabla S^T \mathbf{n})}{\mathbf{n}^T \mathbf{n}} = \frac{\mathbf{n}^T \begin{bmatrix} S_x^2 & S_x S_y \\ S_x S_y & S_y^2 \end{bmatrix} \mathbf{n}}{\mathbf{n}^T \mathbf{n}}$$

Harris Corner Detector: step 3

3. To address the noise in the Moravec detector, smooth $S_{\mathbf{n}}^2$ by convolution with a Gaussian kernel of size σ_I:

$$C_{\mathbf{n}}(x, y) \triangleq G_{\sigma_I}(x, y) \circledast S_{\mathbf{n}}^2 = \frac{\mathbf{n}^T \begin{bmatrix} \langle S_x^2 \rangle & \langle S_x S_y \rangle \\ \langle S_x S_y \rangle & \langle S_y^2 \rangle \end{bmatrix} \mathbf{n}}{\mathbf{n}^T \mathbf{n}}$$

where $\langle \cdot \rangle$ is the smoothed value. This equivalent[1] to weighting the intensity differences squared, $S_{\mathbf{n}}^2$, in the local neighbourhood by Gaussian weights centred at (x, y).

The smoothed, squared change in intensity around (x, y) in direction $\mathbf{n}$ is therefore given by:

$$C_{\mathbf{n}}(x, y) = \frac{\mathbf{n}^T A \mathbf{n}}{\mathbf{n}^T \mathbf{n}} \qquad \text{where } A \triangleq \begin{bmatrix} \langle S_x^2 \rangle & \langle S_x S_y \rangle \\ \langle S_x S_y \rangle & \langle S_y^2 \rangle \end{bmatrix}$$

References: Harris, Chris, and Mike Stephens. "A combined corner and edge detector." *Alvey vision conference*. 1988

[1]More detailed derivation of equivalence: http://www.cse.yorku.ca/~kosta/CompVis_Notes/harris_detector.pdf

Corner Detection (Harris - eigenvalues)

Rayleigh Quotients

Reminder: a Rayleigh quotient[1] of a symmetric matrix A is a *normalised quadratic form*:

$$R(A, \mathbf{n}) = \frac{\mathbf{n}^T A \mathbf{n}}{\mathbf{n}^T \mathbf{n}}$$

with the useful property that $\lambda_{min} \leq R(A, \mathbf{n}) \leq \lambda_{max}$ where $\lambda_{min}, \lambda_{max}$ are the smallest and largest eigenvalues of A.

Interpreting $C_{\mathbf{n}}(x, y)$

Observation: since A is symmetric, we have cunningly defined $C_{\mathbf{n}}(x, y)$ as a Rayleigh quotient. Thus, we know that $\lambda_{min} \leq C_{\mathbf{n}}(x, y) \leq \lambda_{max}$.

Interpretation: if we try every possible orientation $\mathbf{n}$, the maximum smoothed change in squared intensity we will find is λ_{max}, and the minimum value is λ_{min}.

Finding corners from $C_{\mathbf{n}}(x, y)$

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

No structure: (smooth variation) $\lambda_{min} \approx \lambda_{max} \approx 0$

1D structure: (edge) $\lambda_{min} \approx 0$ (direction of edge), λ_{max} large (normal to edge)

2D structure: (corners) λ_{min} and λ_{max} both large

References: Rayleigh quotients come up a lot in machine learning and computer vision.
(some other applications are described here https://www.sjsu.edu/faculty/guangliang.chen/Math253S20/lec4RayleighQuotient.pdf NE)

Corner Detection (Harris - det/trace)

The det/trace trick

It is necessary to calculate A at every pixel by first computing three images of smoothed gradients ($\langle S_x^2 \rangle, \langle S_x S_y \rangle, \langle S_y^2 \rangle$).

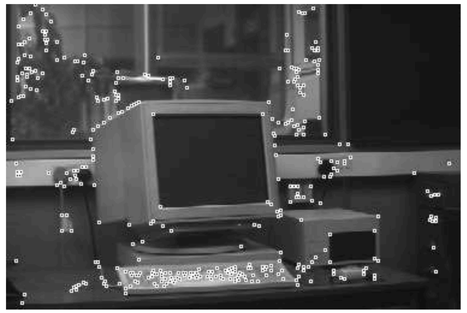
But, we can *avoid computing the eigenvalues* by evaluating the determinant and trace of A:

$$\text{trace } A = \lambda_{min} + \lambda_{max} = \langle S_x^2 \rangle + \langle S_y^2 \rangle$$
$$\det A = \lambda_{min}\lambda_{max} = \langle S_x^2 \rangle \langle S_y^2 \rangle - \langle S_x S_y \rangle^2$$

Only large if both λ_{max} and λ_{min} are large

Finally, 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").

Thresholding



Low threshold

High threshold

Applications of corner detection

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.

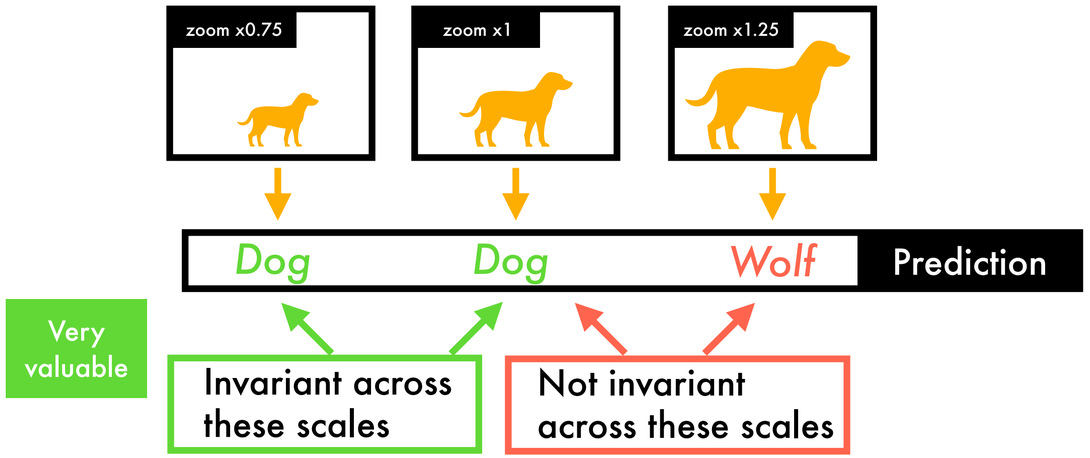
Current detectors are not too reliable, and higher-level visual routines must be designed to tolerate a significant number of outliers in the output of the corner detector.

The importance of scale invariance

Our dream: *invariance*

To work in the real world, we want our models to be **invariant** to certain properties of objects.

Example: Invariance to scale in object recognition: if our model gives the same output for different scales of input, it is said to be *"scale invariant"*.

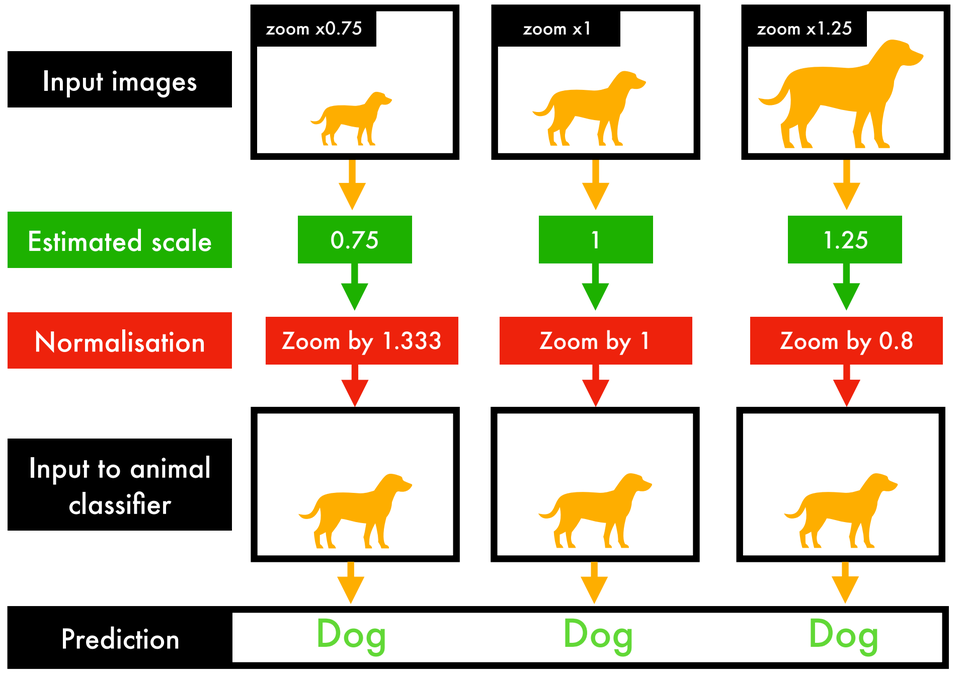


We want to be able to recognise tigers while they are still in the distance as well as close up...

How to achieve invariance?

Suppose we have an animal classifier that works at just **one** scale.

If we can estimate the scale of an object, we can **normalise** it to achieve scale invariance:



Scale is difficult to infer from corners

Observation

Corners and edges are useful for identifying points of interest, but they have a significant shortcoming:

It is difficult to infer the scale of edges and corners

Inferring scale

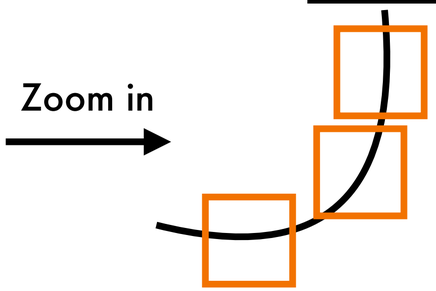
For a feature to be capable of predicting scale, it must itself **behave differently** at different scales (i.e. it must *not* be invariant).

Do corners behave differently at different scales?

Sometimes yes

A corner at one scale

Zoom in



...becomes edges at another scale!

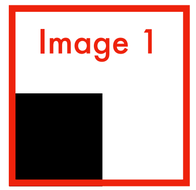
Sometimes no

A delivery robot with a camera needs to recognise two kennels for a dog food delivery:

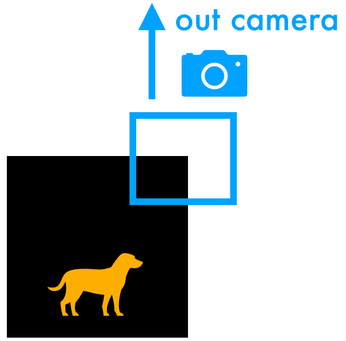
Big kennel

Little kennel

Which corner belongs to which kennel?



It is impossible to tell the scale from the corner image!



In practice....

It has been observed empirically that Harris corners alone *do not* reliably predict scale (Mikolajczyk and Schmid, 2001)

References: K. Mikolajczyk and C. Schmid. "Indexing based on scale invariant interest points." ICCV 2001; The corner-to-edges figure is based on a figure from Rob Fergus.
Excellent resource for further reading: Szeliski, Richard. "Computer Vision: Algorithms and Applications." 2nd Edition (2021).

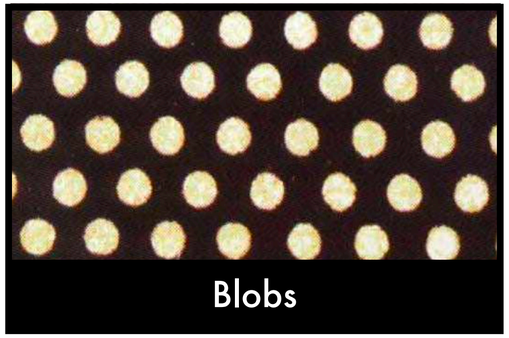
Blobs

Motivation

We'd like a feature that can be used to reliably predict scale. Blobs can help!

What is a blob?

A blob is an area of uniform/similar intensity in the image.



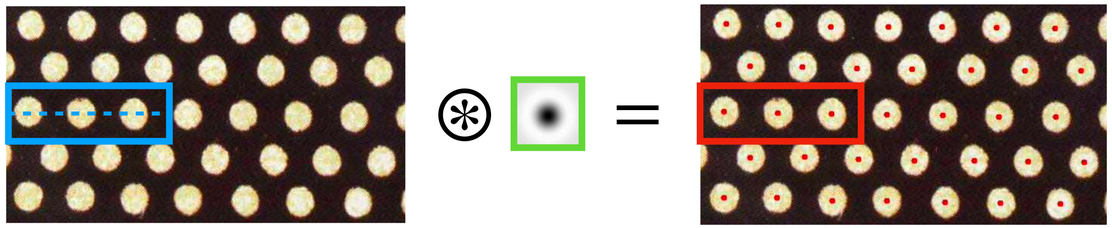
Blobs

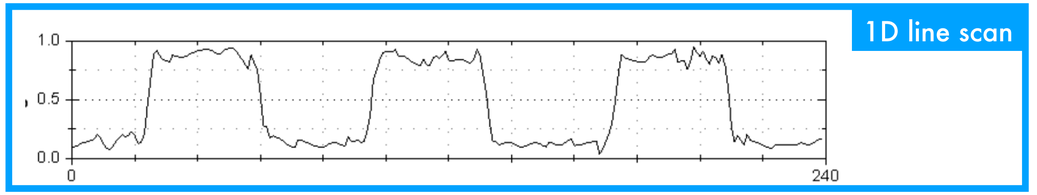
Whereas edges and corners are features which are found at discontinuities, blobs are localised in the middle of areas of similar intensity which are surrounded by pixels of a different intensity on their boundaries.

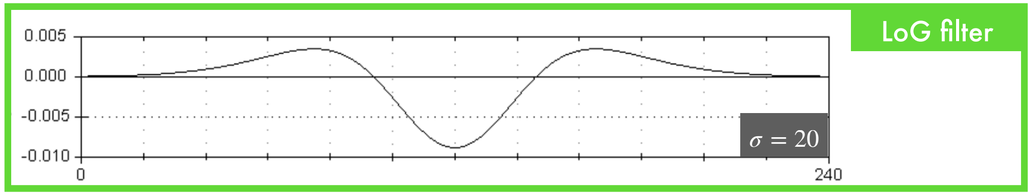
Reference: T. Lindeberg. "Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention." *IJCV, 1993*

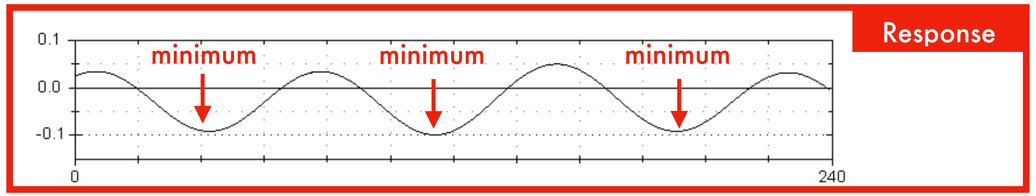
Detecting blobs

Blobs can be detected with the Laplacian of Gaussian filter.









Despite a noisy signal, the minima of the response from the scale-normalised Laplacian of Gaussian at the correct scale, σ, localise the centres of bright blobs on a **dark background** perfectly.

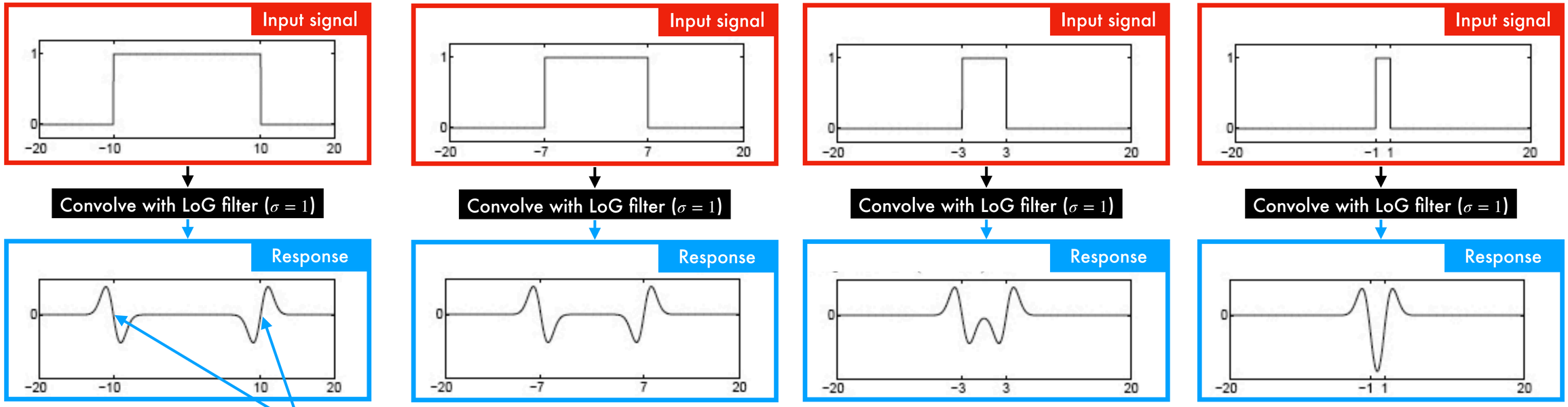
By contrast, **dark blobs** on a bright background will produce maxima in the response.

Blob centres and band-pass filtering

The role of σ

Why does the Laplacian of Gaussian filter give a strong negative response at the centre of a bright blob on a dark background (for the appropriate value of σ)?

To build intuition, we can apply a Laplacian of Gaussian with $\sigma = 1$ to a box function of different widths.



"Ripples" with zero-crossings at the position of edges (we saw this earlier for edge detection)

ripples get closer...

And closer...

When the scales match, we get superposition of the ripples to give a strong negative response.

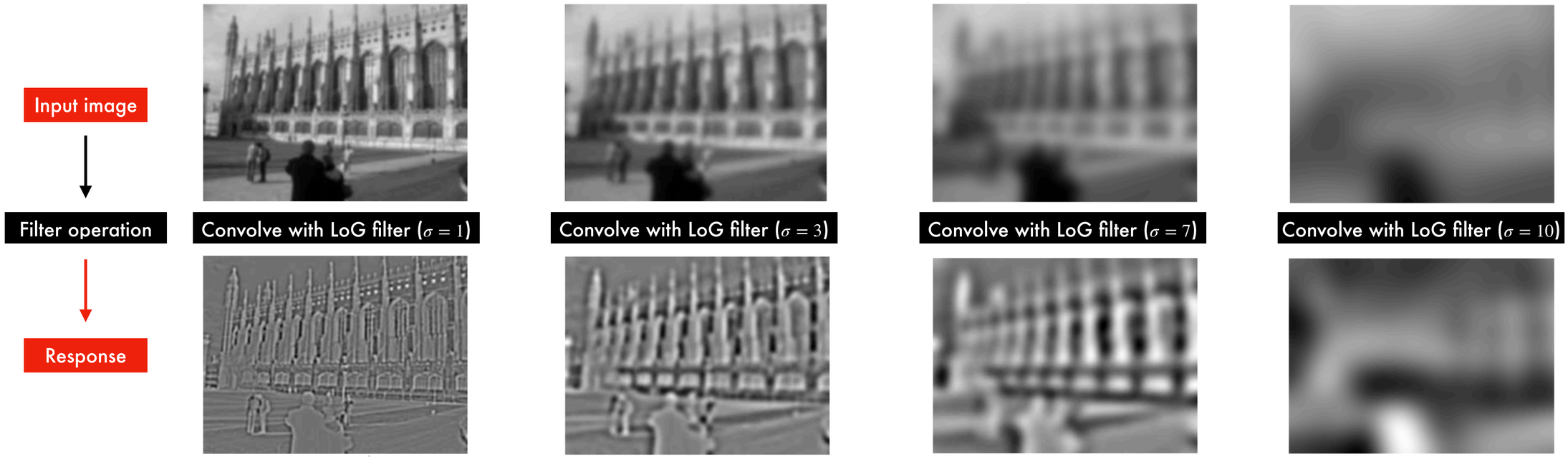
Figure credits: Svetlana Lazebnik

Blobs and band-pass filtering: example

The role of σ

The size of the blob detected depends on the σ value of the LoG filter used.

As the sigma is increased, *larger and larger* image features are detected, ranging from small boxes to entire buildings.



Responds to small structures

Responds to large structures

Each time the blob detector will fire on the centre 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.

Blobs and scales

Responses at different scales

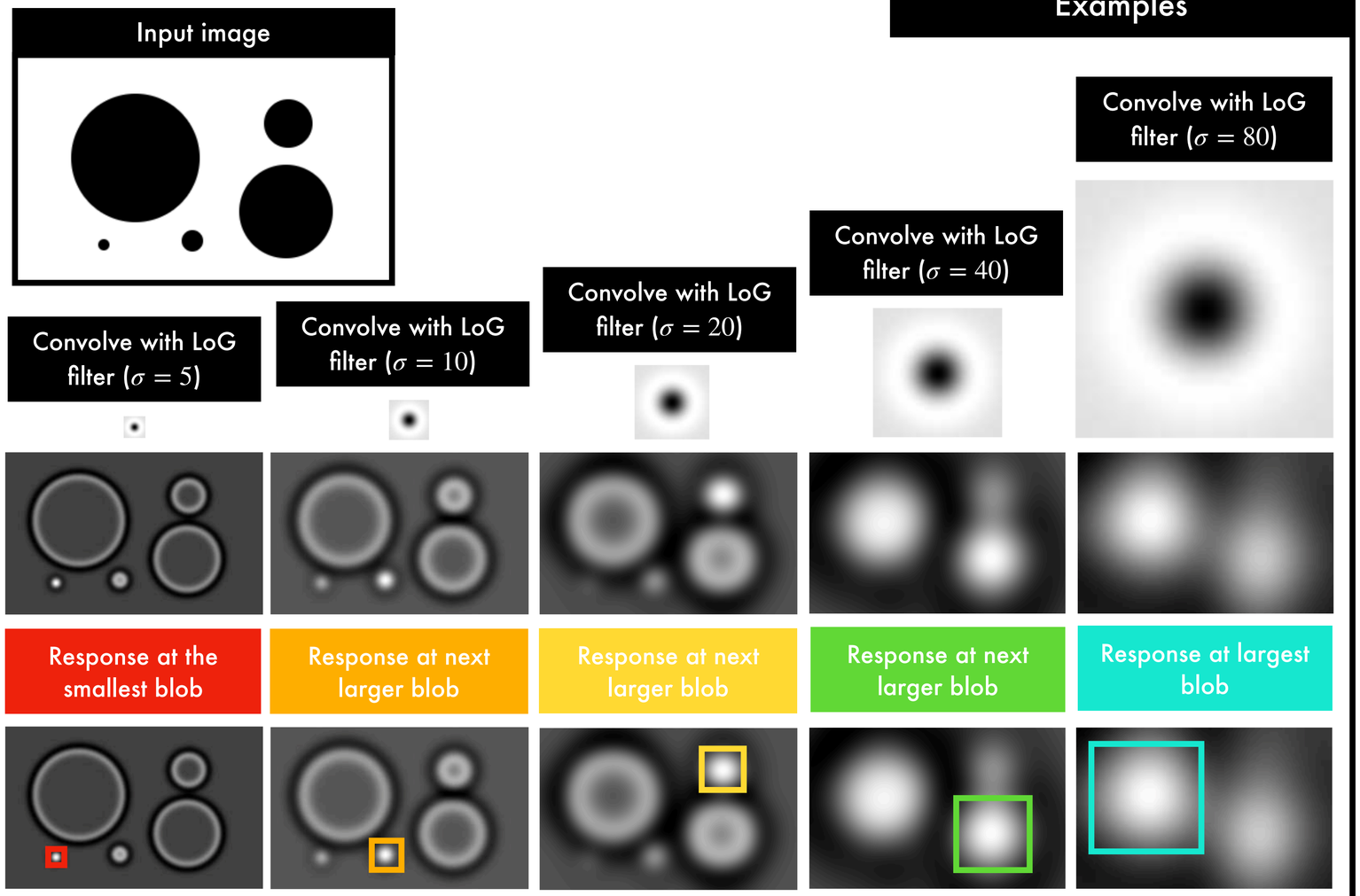
Blobs have a range of scales over which they will be detected.

The (scale-normalised) Laplacian of a Gaussian as recorded at a particular location is a smooth function over scale, with definite peaks or troughs.

These maxima and minima occur at the centre of blobs.

These are considered ideal places to examine the surroundings of the feature point for use in feature description.

Examples



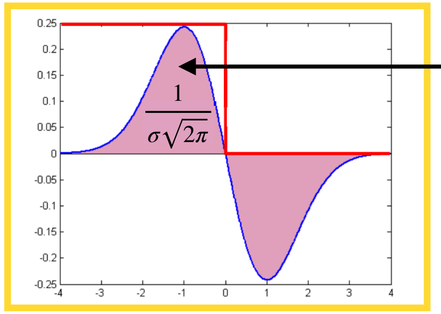
Key takeaway: different σ values can identify blobs at different scales.

A technical detail: the scale-normalised LoG filter

Why do we need to "scale-normalise" the LoG?

We mentioned that when detecting blobs, we use a *scale-normalised* LoG filter. What does this mean and why is it needed?

The response of a derivative of Gaussian filter to a perfect step edge decreases as σ increases.



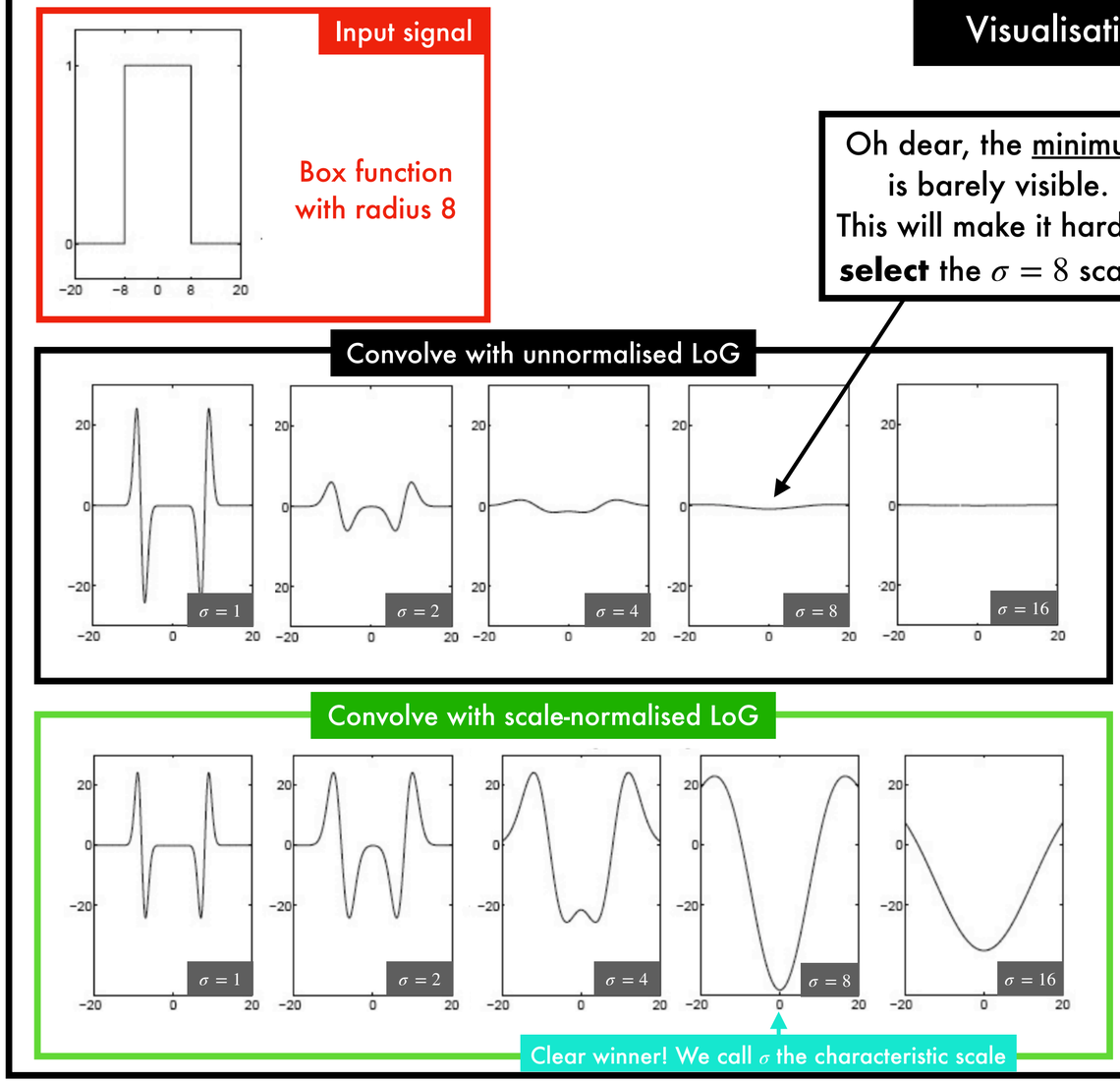
When the filter hits the edge, the response is the integral of the left peak

To produce the same response across different σ values we must multiply the Gaussian derivative by σ.

Since the Laplacian is the *second* derivative of the Gaussian, it must be multiplied by σ^2 to *scale-normalise*:

$$\nabla^2_{norm} G = \sigma^2 \nabla G$$

Visualisation



Slide content credits: Svetlana Lazebnik

Selecting the characteristic scale

Core idea

Different scales are ideal for interest points of different sizes.

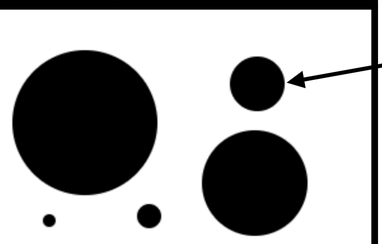
The ideal scale for a keypoint (the *characteristic scale*[1]) is the scale corresponding to the maximum of the detector response at that point.

For example, with a blob, we would want to find the maximum of the magnitude of the scale-normalised 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.

Scale space motivation

Input image

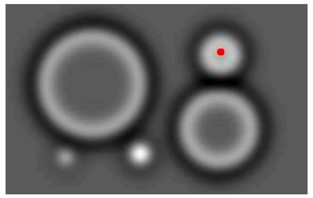


Suppose we want to find the scale of this blob

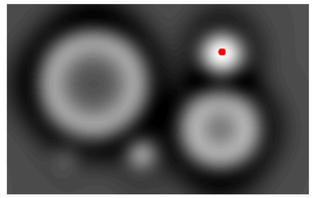
Convolve with scale-normalised LoG filter for different σ values

(Scale 1) $\sigma = 1$

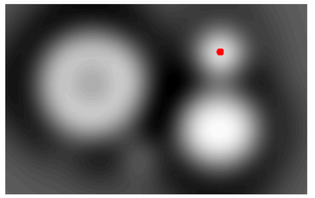
(Scale 2) $\sigma = 7$



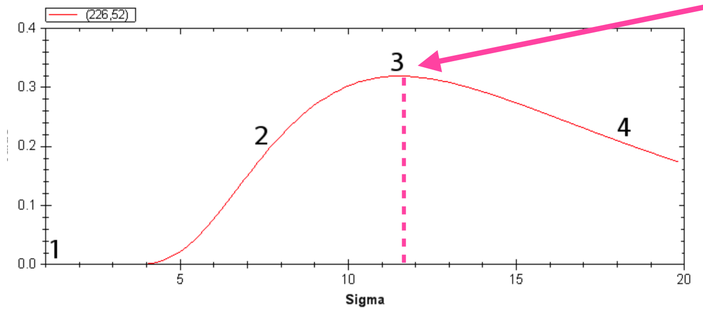
(Scale 3) $\sigma = 12$



(Scale 4) $\sigma = 18$



Scale-normalised LoG response at red dot



We see a clear maximum near scale 3 ($\sigma = 12$)

Note: this function is continuous. To find the exact point and scale of the blob, a set of discrete scales are sampled (via an Image Pyramid) and the maximum is found by *interpolation*.

Reference: [1](terminology for characteristic scale) T. Lindeberg, Feature detection with automatic scale selection. *International journal of computer vision, 1998*

Using scale space to achieve scale invariance

Achieving *scale invariance*

We saw earlier that can achieve scale invariance by accurately estimating the scale of a structure, then normalising.

We now have the tools we need: we can obtain scale independence by looking at the **different resolutions** (low-pass filtered at different scales) of an image, and selecting the scale that gives the strongest response.

There are an infinite number of possible resolutions for any image, which together form a three-dimensional function of intensity over location and scale.

This is what is technically known as the scale space of the image, denoted $S(x, y, \sigma)$.

Computing the scale space

We can calculate $S(x, y, \sigma)$ by convolving the original image $I(x, y)$ with Gaussians of different scale, σ, thus the scale space function can be written as:

$$S(x, y, \sigma) = G(x, y, \sigma) \circledast I(x, y)$$

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

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.

Does blurring need to be Gaussian? Yes! Other kernels can introduce new artefacts at coarser scales[1].

Samples from scale space $S(x, y, \sigma)$ at discrete values of σ

Reference: [1]J. J. Koenderink, "The structure of images". *Biological cybernetics, 1984*

Scale space: computational tricks

Challenge

Computing the full scale space of an image would be extremely expensive:

- Expensive in **computation** (many convolutions)
- Expensive in **memory** (many blurred images to store)

Trick 1: sparse sampling

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

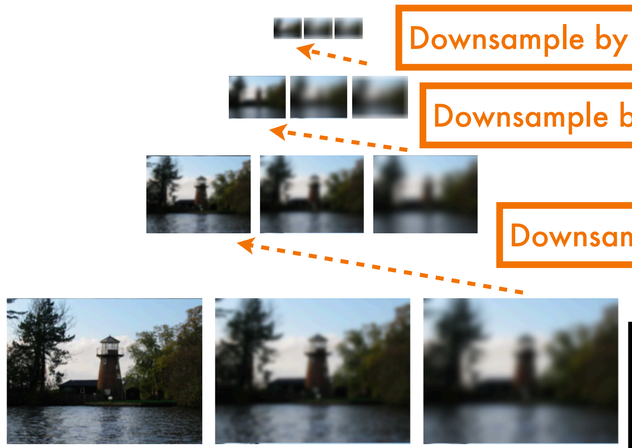
$$\sigma_i = 2^{\frac{i}{s}}\sigma_0$$

so that it doubles after s intervals[1] (each doubling is referred to as an octave). The s images in each octave are spaced logarithmically with the scale of neighbouring images satisfying $\sigma_{i+i} = 2^{\frac{1}{s}}\sigma_i$.

Trick 2: image pyramids

Recall: our image sampling rate should be $\geq 2 \times$ highest frequency (the Nyquist rate) to accurately capture the signal (avoid aliasing).

Each time the scale doubles (i.e. one full octave) in scale space, the blurring (a low-pass filter) has removed sufficient high frequency information that we can subsample the image by a factor of 2 without losing information!



Each layer of the pyramid corresponds to one octave.

Example image pyramid with four octaves, $s = 3$

Blurring smaller images is much cheaper because:

1. We process fewer pixels
2. We avoid the use of very large kernels to compute responses at large scales

Reference: [1](justification for logarithmic spacing NE) L. M. J. Florack, et al. "Scale and the differential structure of images." *Image and vision computing, 1992*

Scale space: more computational tricks

Trick 3: incremental blurs

Even within octaves, blurring with larger Gaussian kernels is expensive. How can avoid these costly convolutions?

The reproducing property of the Gaussian comes to the rescue:

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

Given $S(x, y, \sigma_i)$, where $\sigma_i = 2^{\frac{i}{s}}\sigma_0$, we want to compute $S(x, y, \sigma_{i+1})$, where $\sigma_{i+i} = 2^{\frac{1}{s}}\sigma_i$. From the reproducing property, we know that $G(\sigma_{i+1}) = G(\sigma_i) \circledast G(\sigma_{k_i})$ for some value of σ_{k_i} which we can solve for.

$$\sigma_{k_i} = \sqrt{\sigma_{i+1}^2 - \sigma_i^2} \text{ (reproducing property)}$$

$$\sigma_{i+1} = 2^{\frac{1}{s}}\sigma_i \text{ (by definition)}$$

$$\sigma_{k_i} = \sqrt{2^{\frac{2}{s}}\sigma_i^2 - \sigma_i^2} = \sigma_i\sqrt{2^{\frac{2}{s}} - 1}$$

Find incremental blur size

This gives s distinct and small incremental Gaussian (low-pass) filters, σ_{k_i}, need only be computed once!

They can be reused in each subsequent octave but on sub-sampled images to achieve the larger scales.

No large convolutions required!

Scale space: yet more computational tricks

Trick 4: DoG

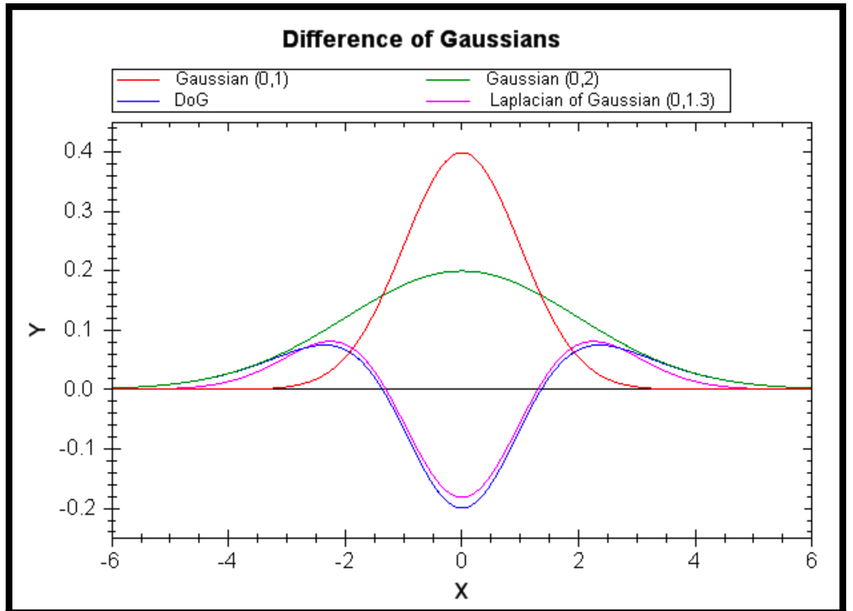
The Difference of Gaussians filter (or "DoG" as it is often called), is also a blob detector.

Blobs are found 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*.

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

The DoG approximation



Comparing the blue and magenta lines, we can see it's a pretty good approximation!

In a system which uses a scale space pyramid, DoG points are very useful entities, as a response can be computed simply subtracting one member of a pyramid level from the one directly above it!

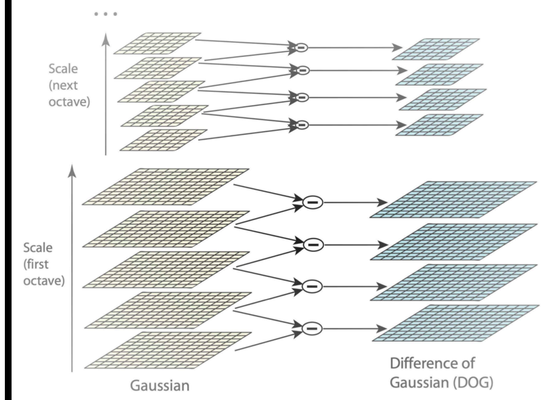
Putting it together: efficient scale-invariant keypoint detection

Finding keypoints efficiently across scales

Keypoint locations (the blob centres) are found by first computing an approximation for the Laplacian of the Gaussian pyramid by using Difference of Gaussians.

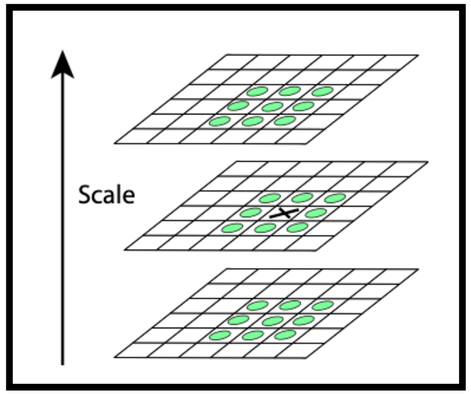
This is done efficiently by subtracting neighbouring images of same dimension in the Image Pyramid[1].

The DoG pyramid



The location of the local maximum/minimum of DoG response (in image position and over scale) gives the keypoint location and characteristic scale.

Finding local extrema



A local search of 26 neighbour responses is required to determine if a pixel is a blob-centre and to find the scale.

Summary

DoG pyramid allows us to estimate the position and scale of keypoints *efficiently*.

In the next lecture, we will see how we can use the estimated scale to perform scale normalisation to achieve **scale invariance**.

Reference: [1] Lowe, David G. "Distinctive image features from scale-invariant keypoints." IJCV (2004)

End of Image Structures Lecture 2

Appendix

Slide content credits

Much of the content of these slides is based on material by Roberto Cipolla. For other cases, I have tried to credit the figures where possible, but finding the original sources can be challenging (useful figures naturally propagate across many slide decks!) Please let me know if you spot a missing reference.

Appendix

More detailed derivation for Harris Corner Detector: step 3

The Gaussian-weighted $SSD(x, y)$ distance in the neighbourhood of the patch of the smoothed image $S(x, y)$ is given by:

$$SSD_{\text{Gaussian}}(x, y) = \sum_{u,v} G_{\sigma_I}(u, v) \cdot \big(S(x + u, y + v) - S(u, v)\big)^2$$

We can take a Taylor expansion at $S(u, v)$ to yield $S(u, v) = S(u, v) + \begin{bmatrix} x \\ y \end{bmatrix} \cdot \nabla S(u, v) + O(\nabla^2 S)$.

Ignoring higher order terms and substituting, we get that locally:

$$SSD_{\text{Gaussian}}(x, y) \approx \sum_{u,v} G_{\sigma_I}(u, v) \cdot \Big(\begin{bmatrix} x \\ y \end{bmatrix} \cdot \nabla S(u, v)\Big)^2 = \sum_{u,v} G_{\sigma_I}(u, v) \cdot \Big([x\ y]^T \begin{bmatrix} S_x^2 & S_{xy} \\ S_{xy} & S_y^2 \end{bmatrix} [x\ y]\Big)^2$$

References: Harris, Chris, and Mike Stephens. "A combined corner and edge detector." *Alvey vision conference*. 1988