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)

VE Tag = Very Examinable
NE Tag = Non Examinable

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?

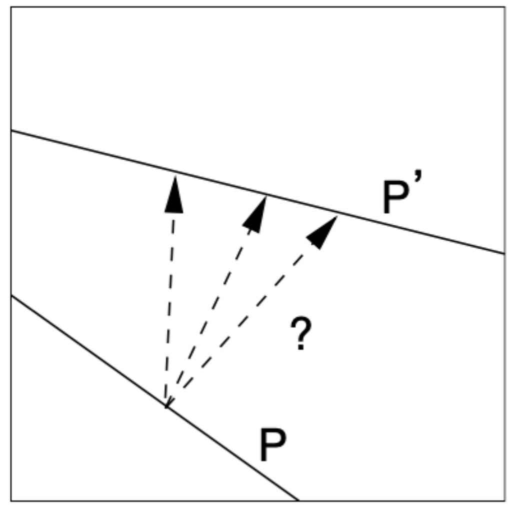
It is impossible to tell!

Can only measure motion normal to the edge

Edges are a powerful intermediate representation but they are sometimes insufficient

This is especially the case when image motion is being analysed

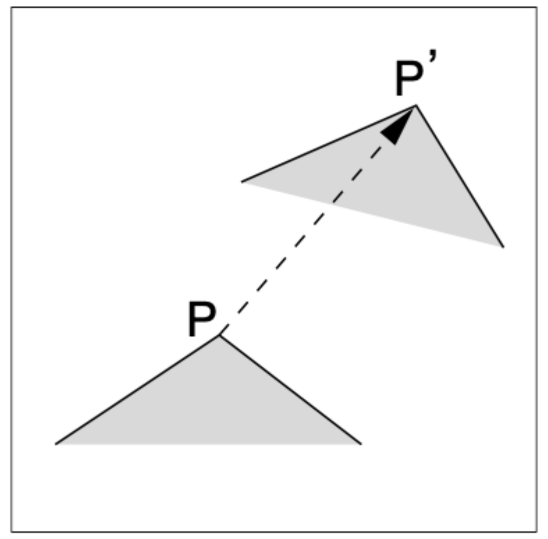
The motion of an edge is rendered ambiguous by the aperture problem: when viewing a moving edge, it is only possible to measure the *motion normal to the edge* locally



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

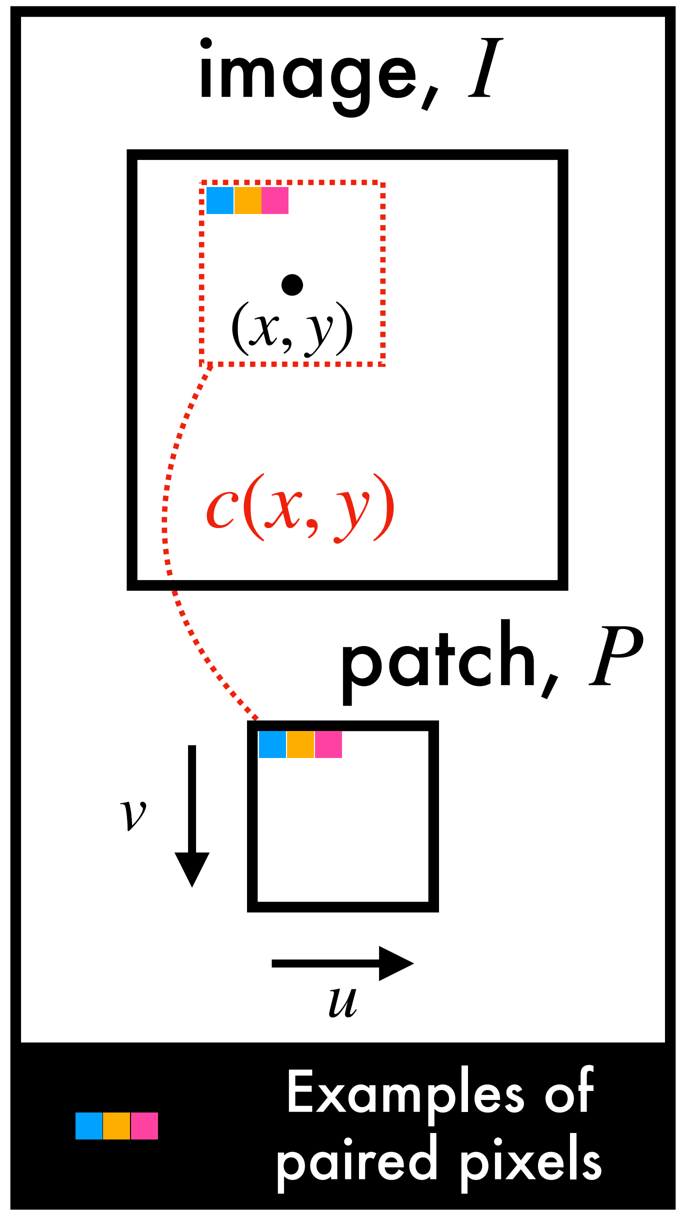
Cross-correlation - another important operator

Normalised cross-correlation

Normalised cross-correlation 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:

$$c(x,y) = \frac{\boxed{\sum_{u=-n}^{n}\sum_{v=-n}^{n}(P(u,v)-\bar{P})(I(x+u,y+v)-\bar{I}_{x,y})}\ \text{covariance}}{\sqrt{\underbrace{\left(\sum_{u=-n}^{n}\sum_{v=-n}^{n}\left(P(u,v)-\bar{P}\right)^2\right)}_{\text{variance of patch}}}\sqrt{\underbrace{\left(\sum_{u=-n}^{n}\sum_{v=-n}^{n}\left(I(x+u,y+v)-\bar{I}_{x,y}\right)^2\right)}_{\text{variance of image under patch}}}}$$



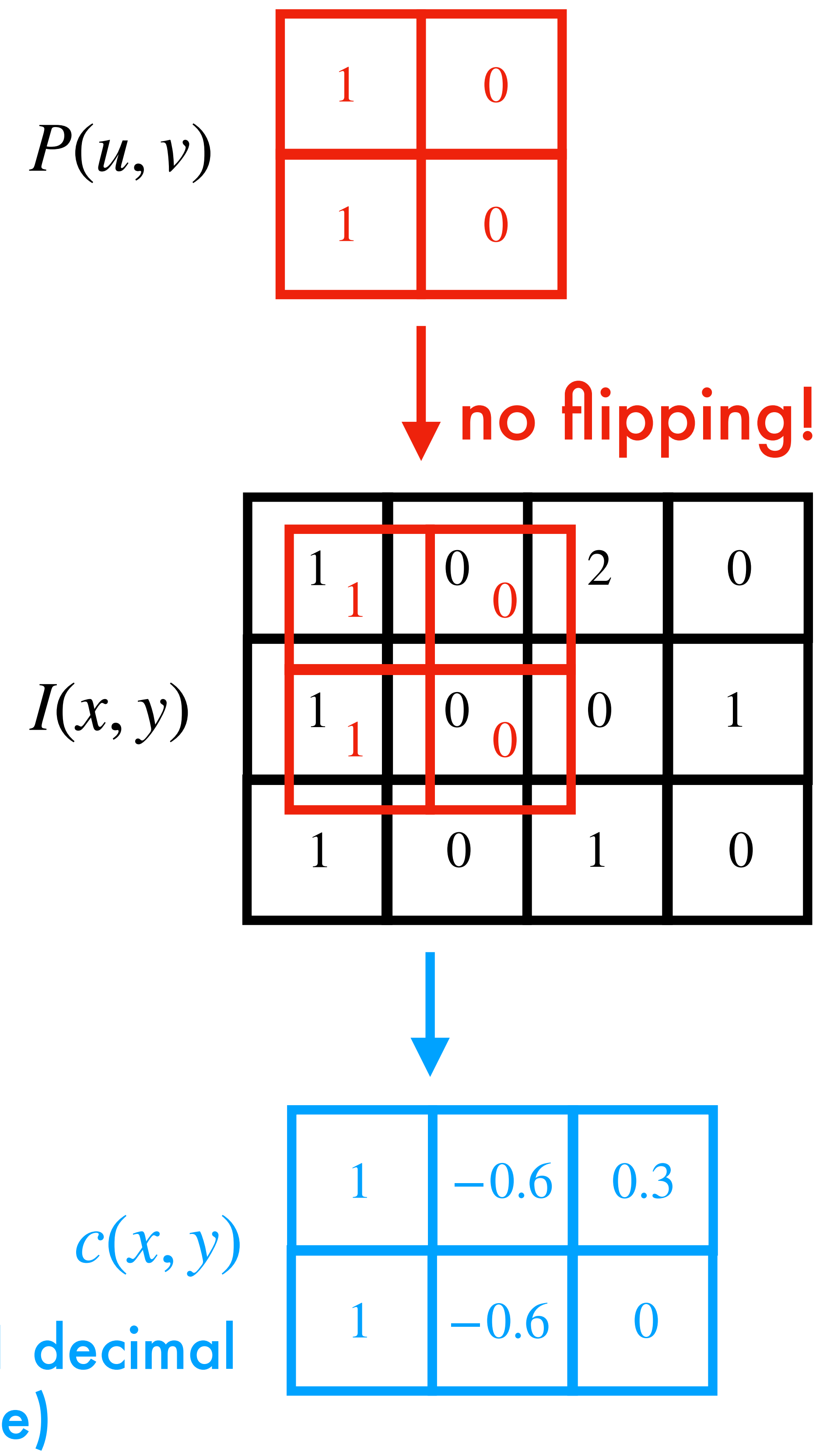
$\bar{P}$ is the mean pixel value of the patch

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

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

Note: 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)

Visualisation



$P(u, v)$

1	0
1	0

$I(x, y)$

1 1	0 0	2	0
1 1	0 0	0	1
1	0	1	0

$c(x, y)$ (to 1 decimal place)

1	−0.6	0.3
1	−0.6	0

Note: software often pads edges of $I(x, y)$ with zeros so that $c(x, y)$ is same shape as (unpadded) $I(x, y)$

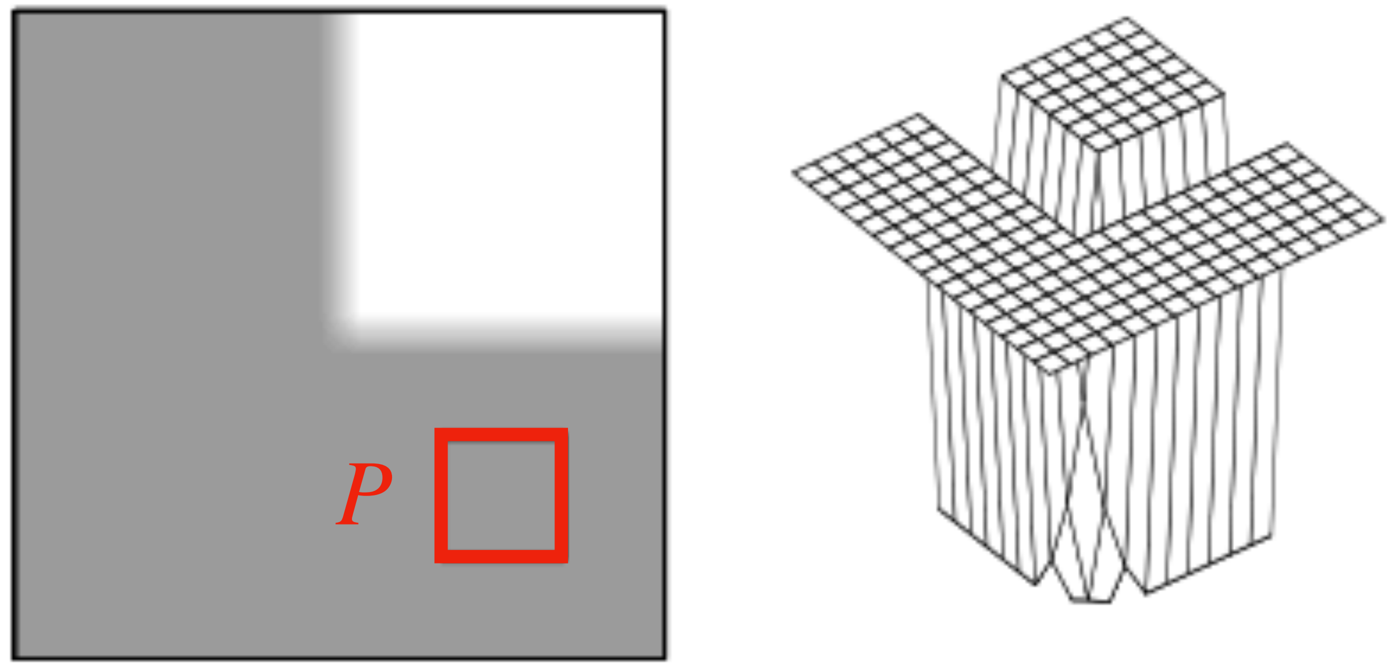
Cross-correlation - corners

cross-correlation peaks at corners

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

Featureless

Auto-correlation with featureless patch

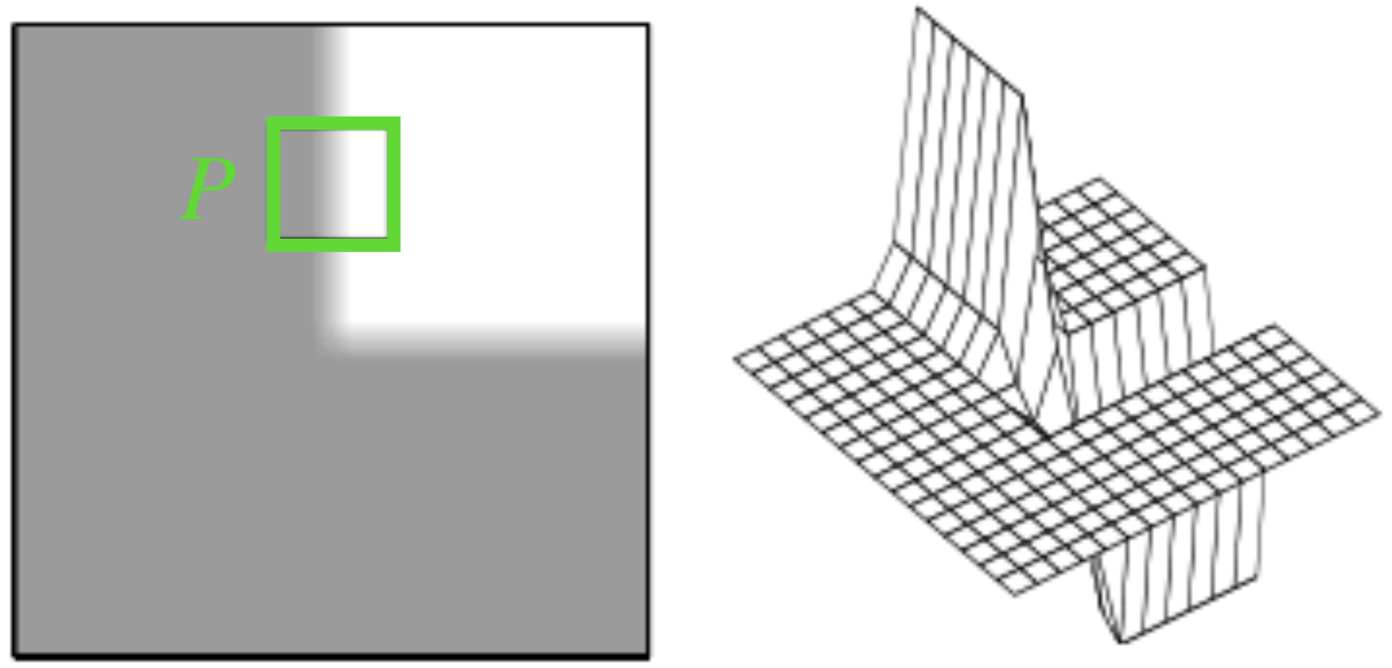


$I(x, y)$ $c(x, y)$

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

Edge

Auto-correlation with an edge patch

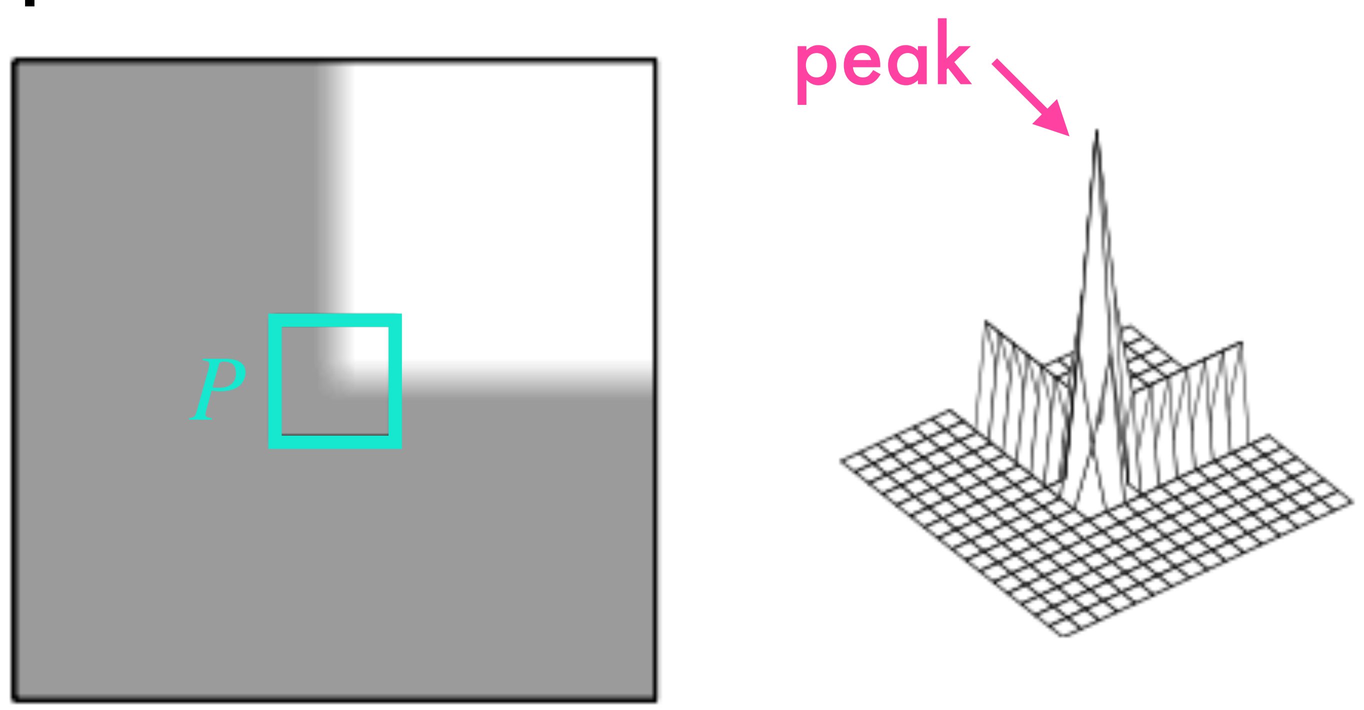


$I(x, y)$ $c(x, y)$

Produces ridge in $c(x, y)$

Corner

Auto-correlation with a corner patch



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

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

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* (simpler variant) 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:

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

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* pixel values often vary *smoothly* so we can approximate last term in $SSD(x, y)$, $I(x + u, y + v)^2$, by a constant (when summing over 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

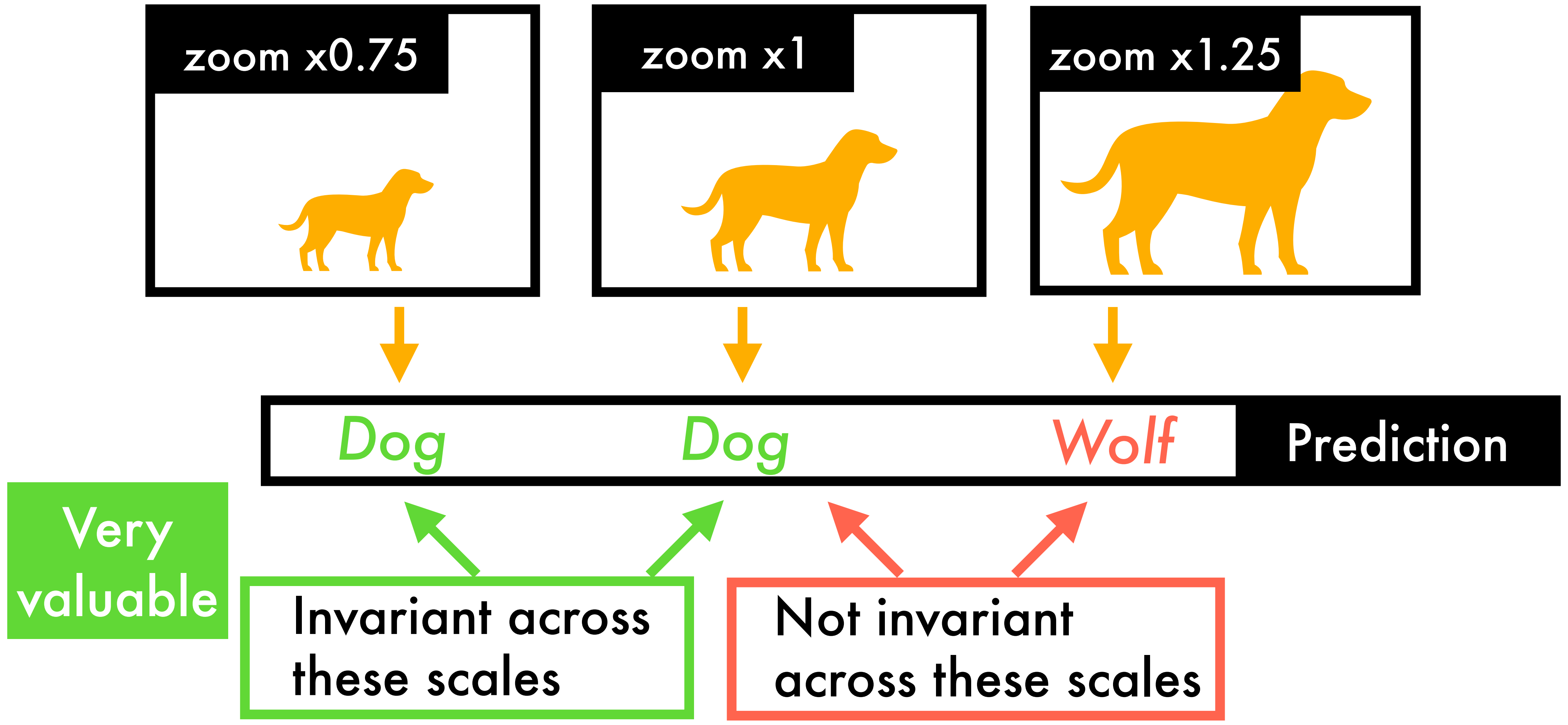
Useful reference: http://www.cse.yorku.ca/~kosta/CompVis_Notes/ssd_cross-correlation.pdf

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"*



zoom x0.75	zoom x1	zoom x1.25	
Dog	Dog	Wolf	Prediction

Very valuable

Invariant across these scales

Not invariant across these scales

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

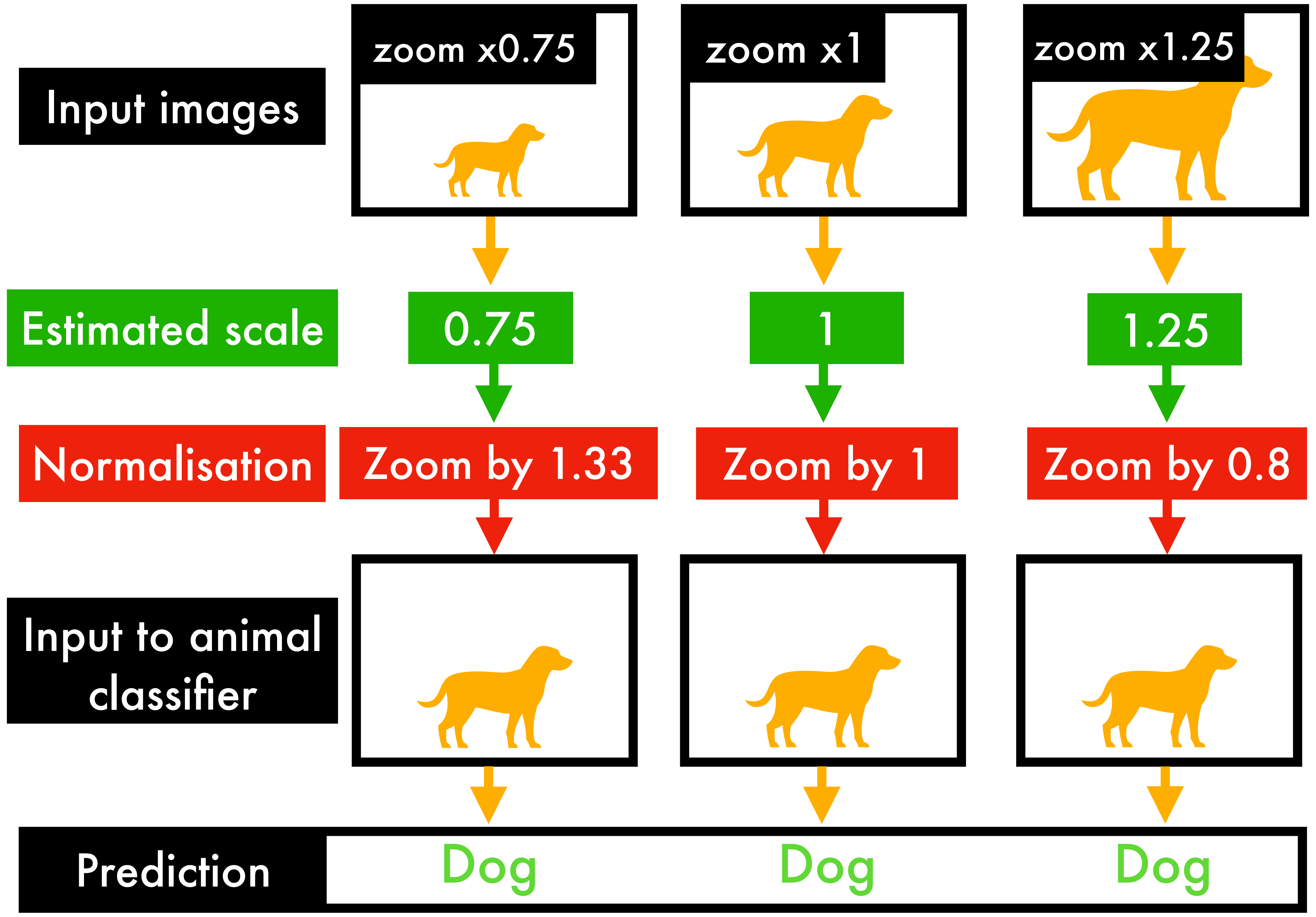
How to achieve invariance?

Suppose animal classifier works at **one** scale

Animal classifier

"Dog"

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



Input images	zoom x0.75	zoom x1	zoom x1.25
Estimated scale	0.75	1	1.25
Normalisation	Zoom by 1.33	Zoom by 1	Zoom by 0.8
Input to animal classifier			
Prediction	Dog	Dog	Dog

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

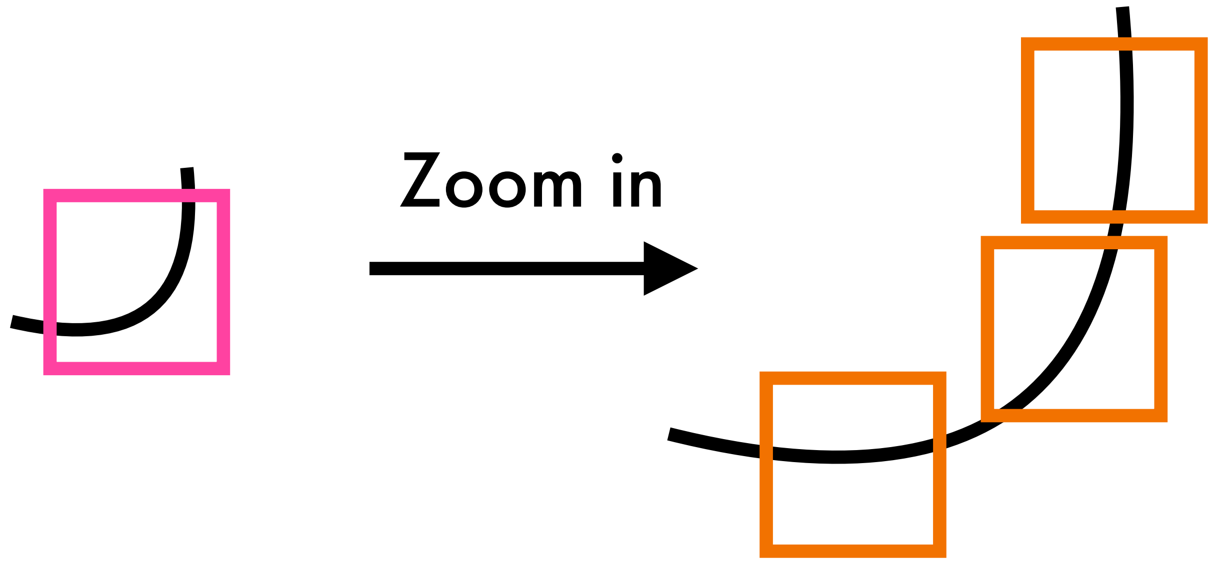
Inferring scale

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

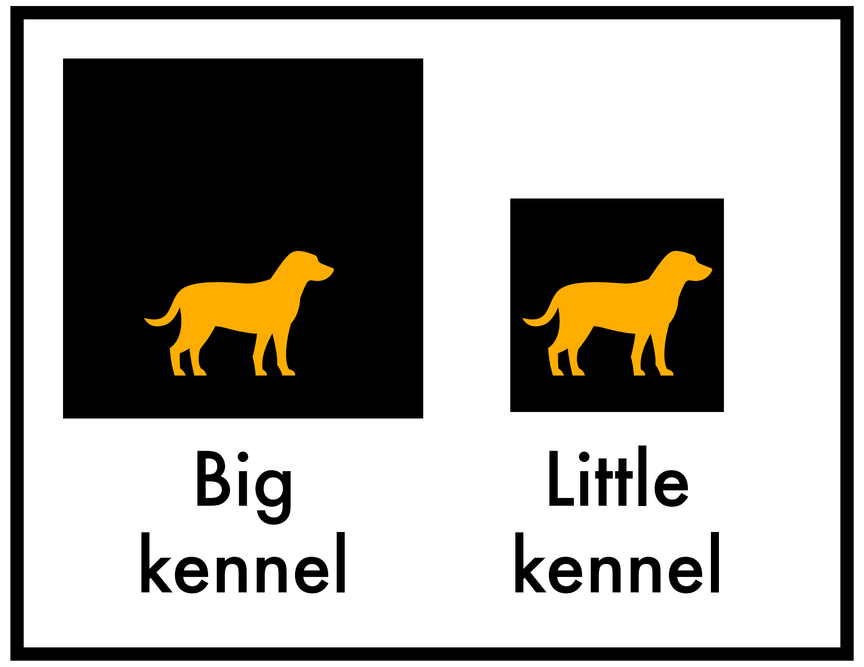
A corner at one scale



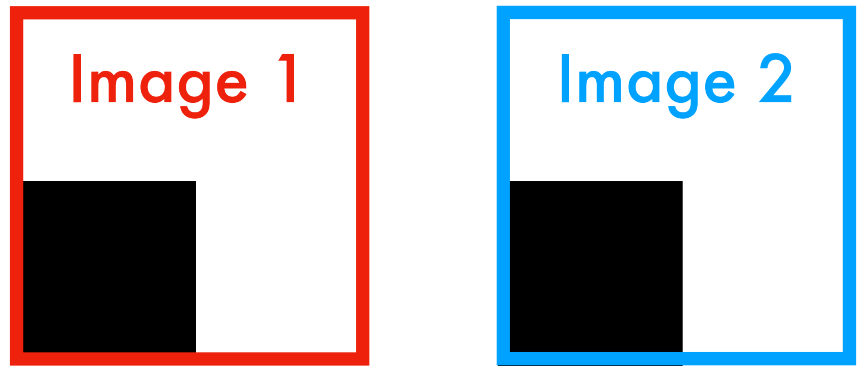
...becomes edges at another scale!

Sometimes no

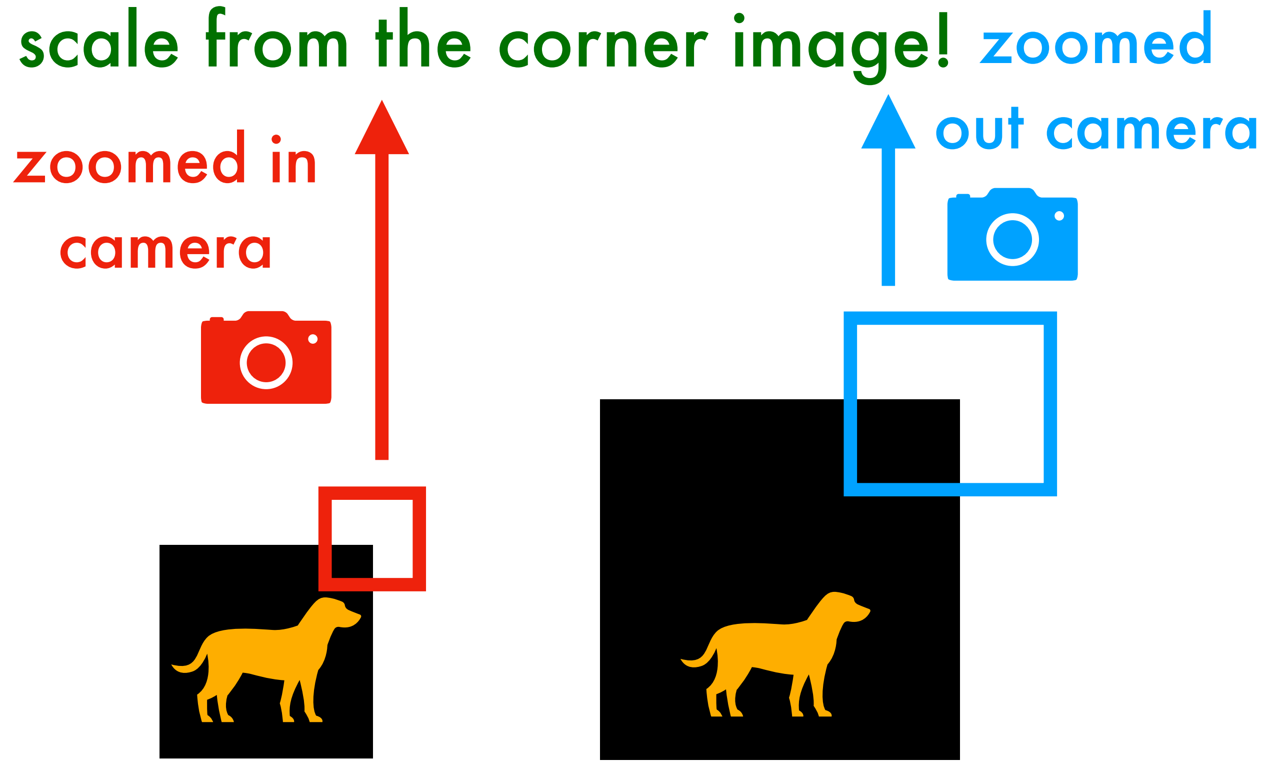
A delivery robot with a camera needs to recognise two kennels for a dog food delivery:



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).

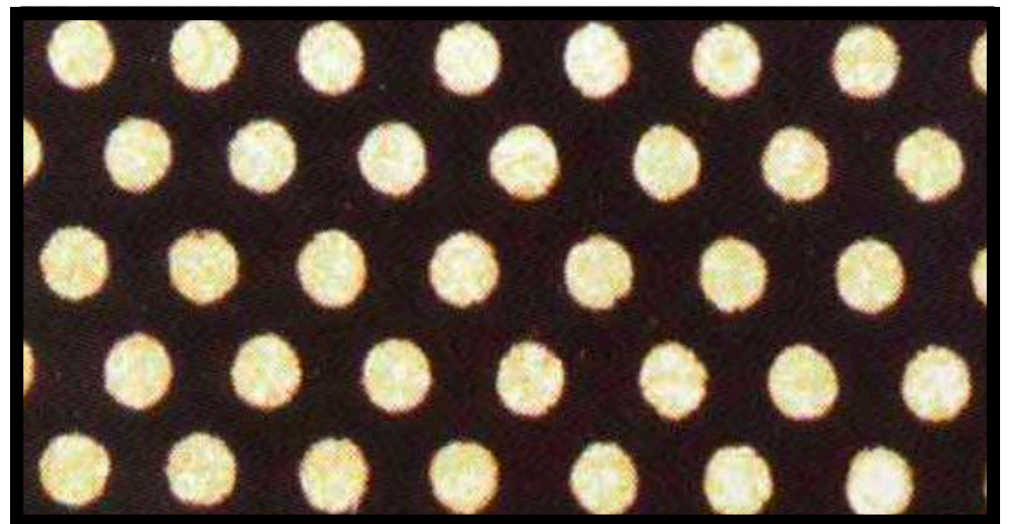
Blobs

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



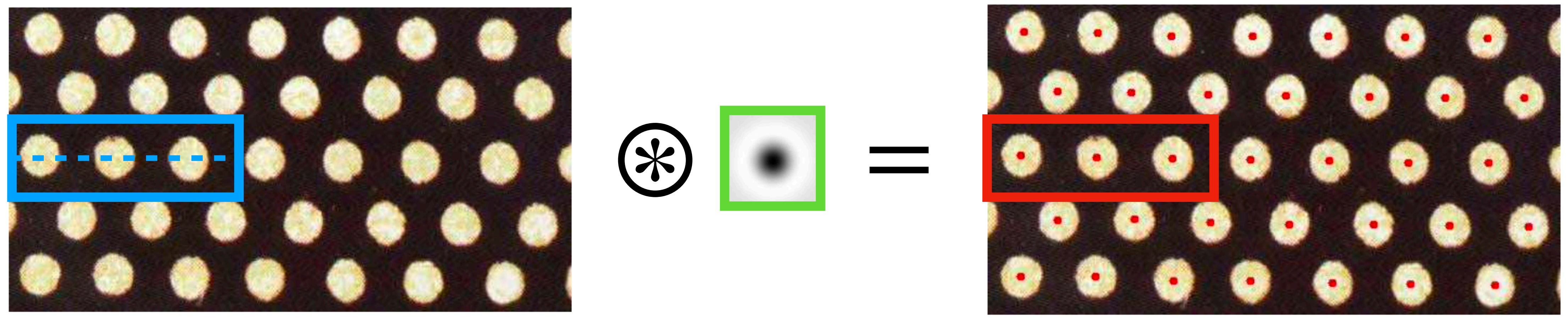
Blobs

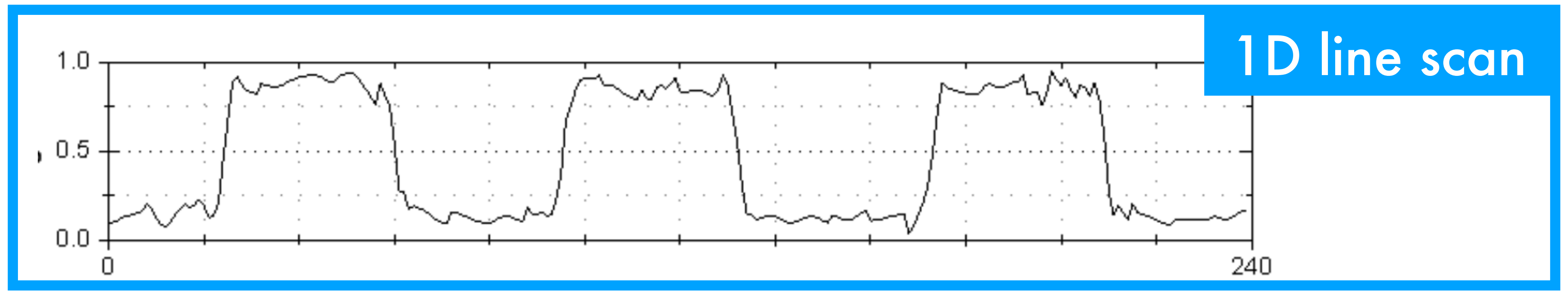
While edges and corners are features which are found at discontinuities, blobs are localised in the middle of areas of similar intensity which are surrounded by pixels of a different intensity on their boundaries

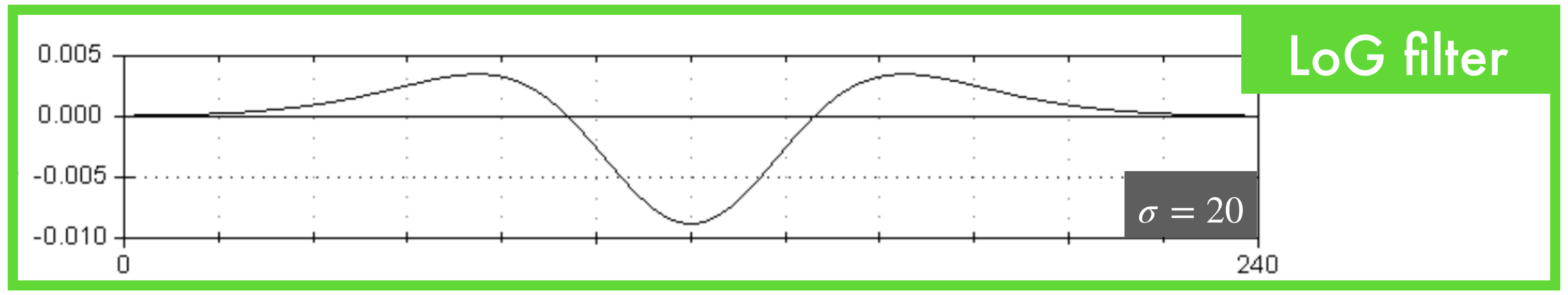
Reference: T. Lindeberg. "Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention." *IJCV, 1993*

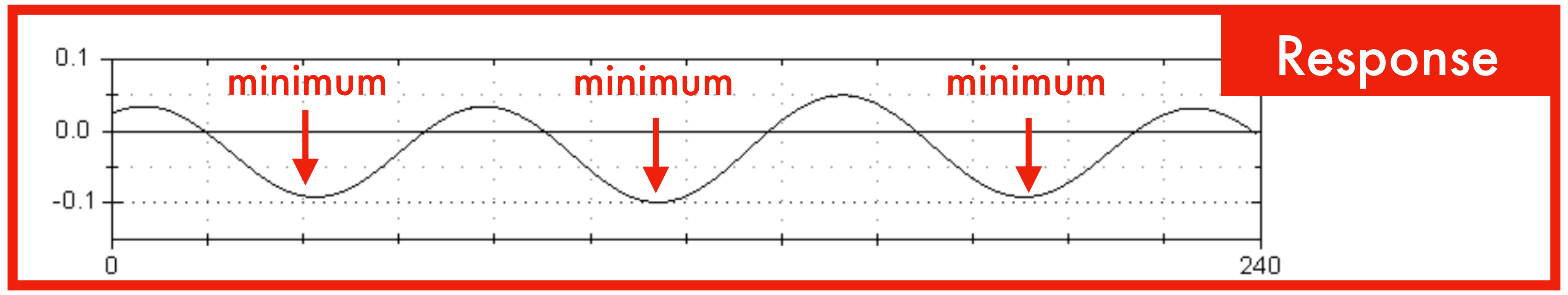
Detecting blobs

Blobs can be detected with the Laplacian of Gaussian filter




1D line scan


LoG filter


Response

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

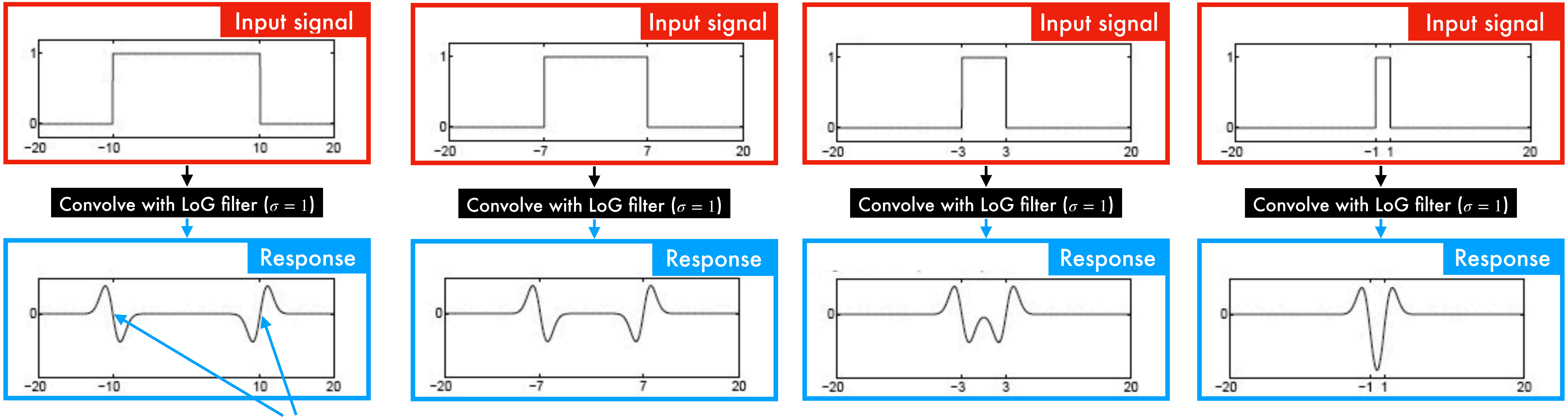
Dark blobs on a **bright background** produce maxima

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

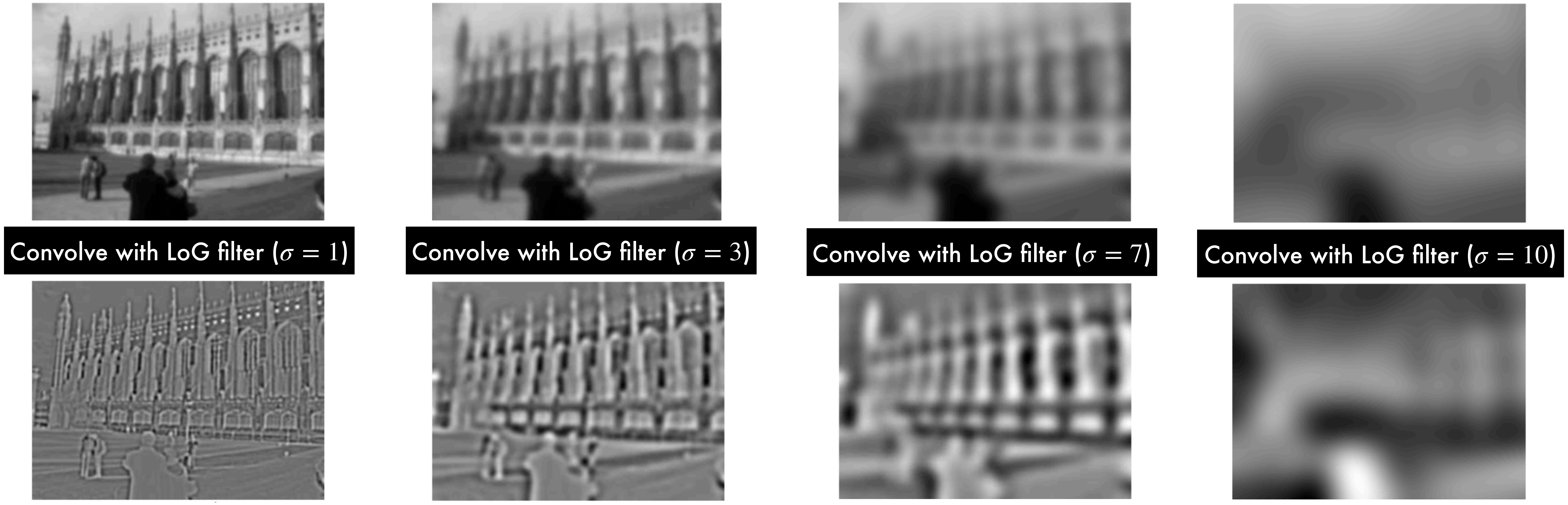
Input image

↓

Filter operation

↓

Response



Convolve with LoG filter ($\sigma = 1$)

Convolve with LoG filter ($\sigma = 3$)

Convolve with LoG filter ($\sigma = 7$)

Convolve with LoG filter ($\sigma = 10$)

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

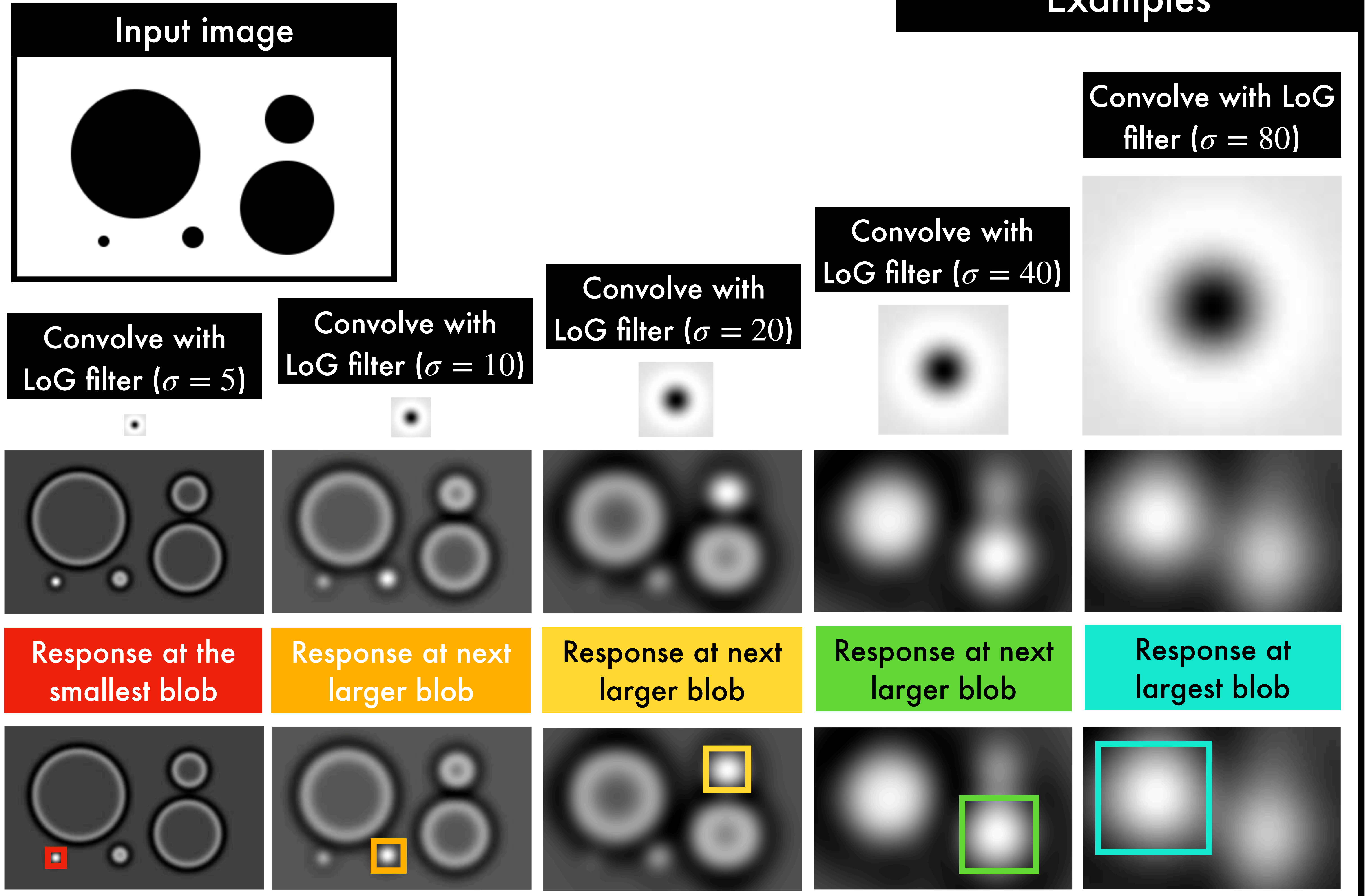
Blobs have a range of scales over which they will be detected

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



Input image

Convolve with LoG filter ($\sigma = 5$)

Convolve with LoG filter ($\sigma = 10$)

Convolve with LoG filter ($\sigma = 20$)

Convolve with LoG filter ($\sigma = 40$)

Convolve with LoG filter ($\sigma = 80$)

Response at the smallest blob

Response at next larger blob

Response at next larger blob

Response at next larger blob

Response at largest blob

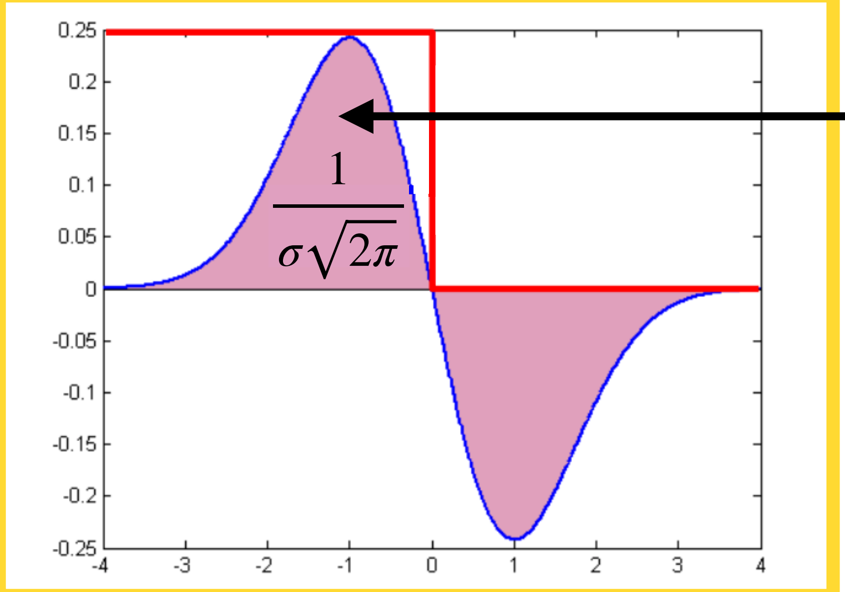
Key takeaway: different σ values can identify blobs at different scales

A technical detail: the scale-normalised LoG filter

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

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

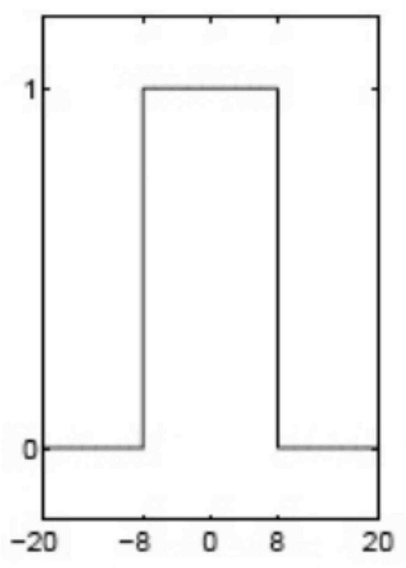
To produce the same response across different σ values we must multiply the Gaussian derivative by σ

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

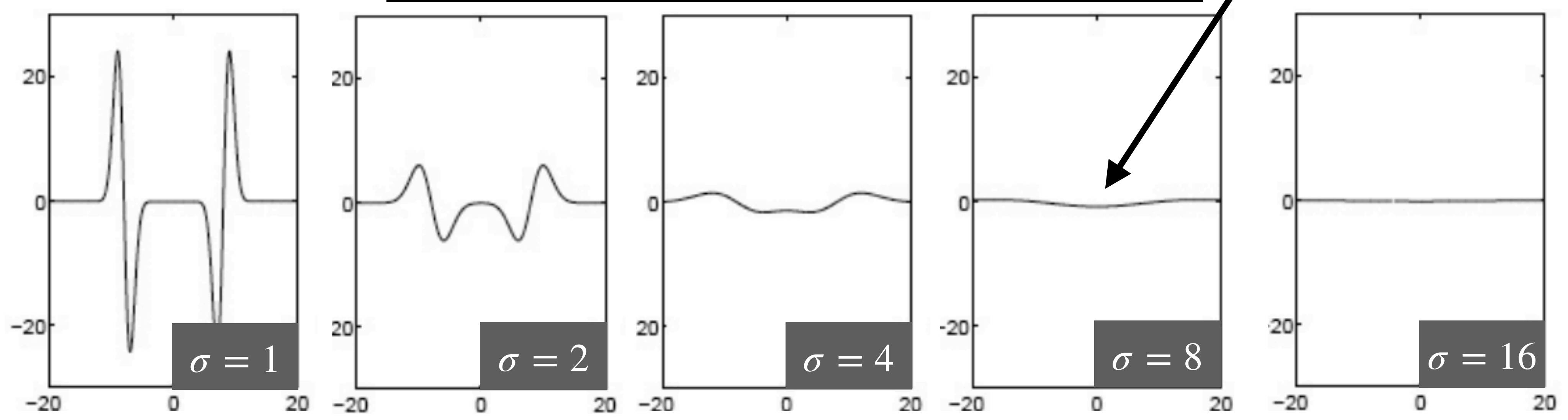
Input signal



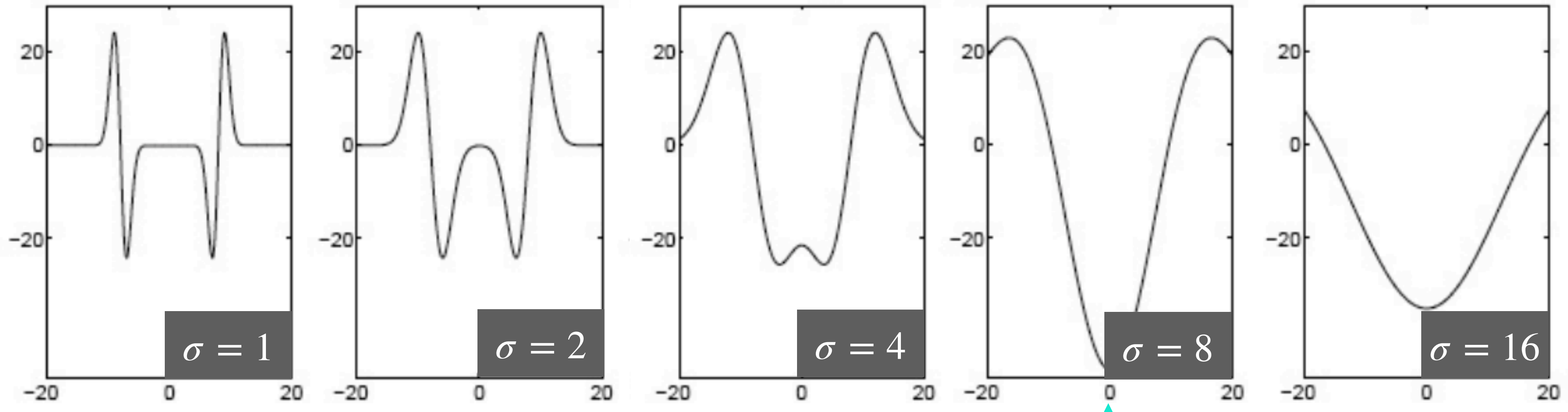
Box function with radius 8

Oh dear, the minimum is barely visible. This will make it hard to **select** the $\sigma = 8$ scale!

Convolve with unnormalised LoG



Convolve with scale-normalised LoG



Clear winner! We call σ the characteristic scale

Slide content credits: Svetlana Lazebnik

Selecting the characteristic scale

Core idea

Different scales are ideal for interest points of different sizes

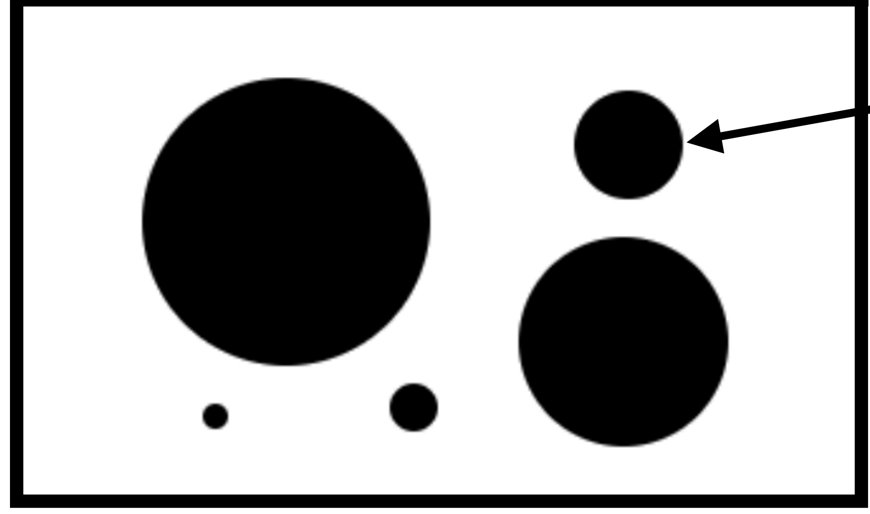
The ideal scale for a keypoint (the *characteristic scale*[1]) is the scale corresponding to the maximum of the detector response at that point.

E.g., with a blob, we want to find 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

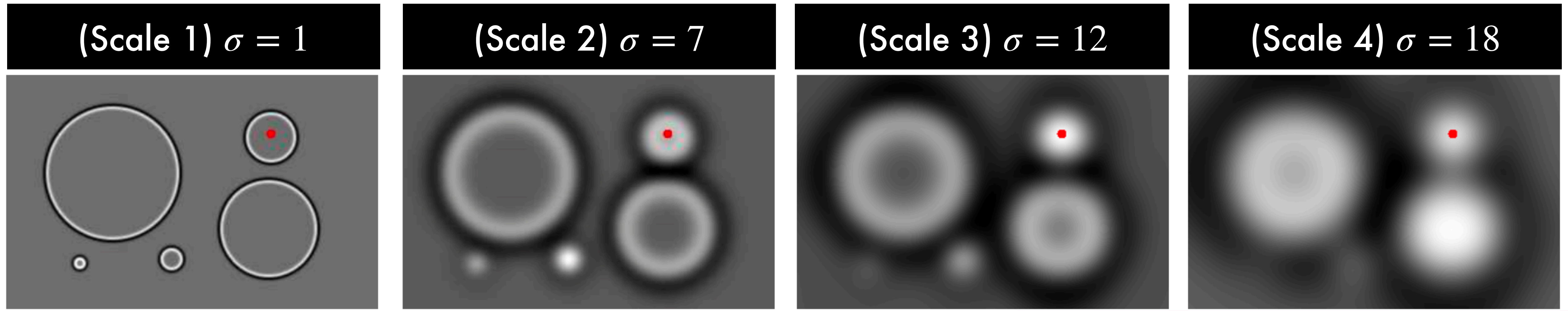
Scale space motivation

Input image



Suppose we want to find the scale of this blob

Convolve with scale-normalised LoG filter for different σ values



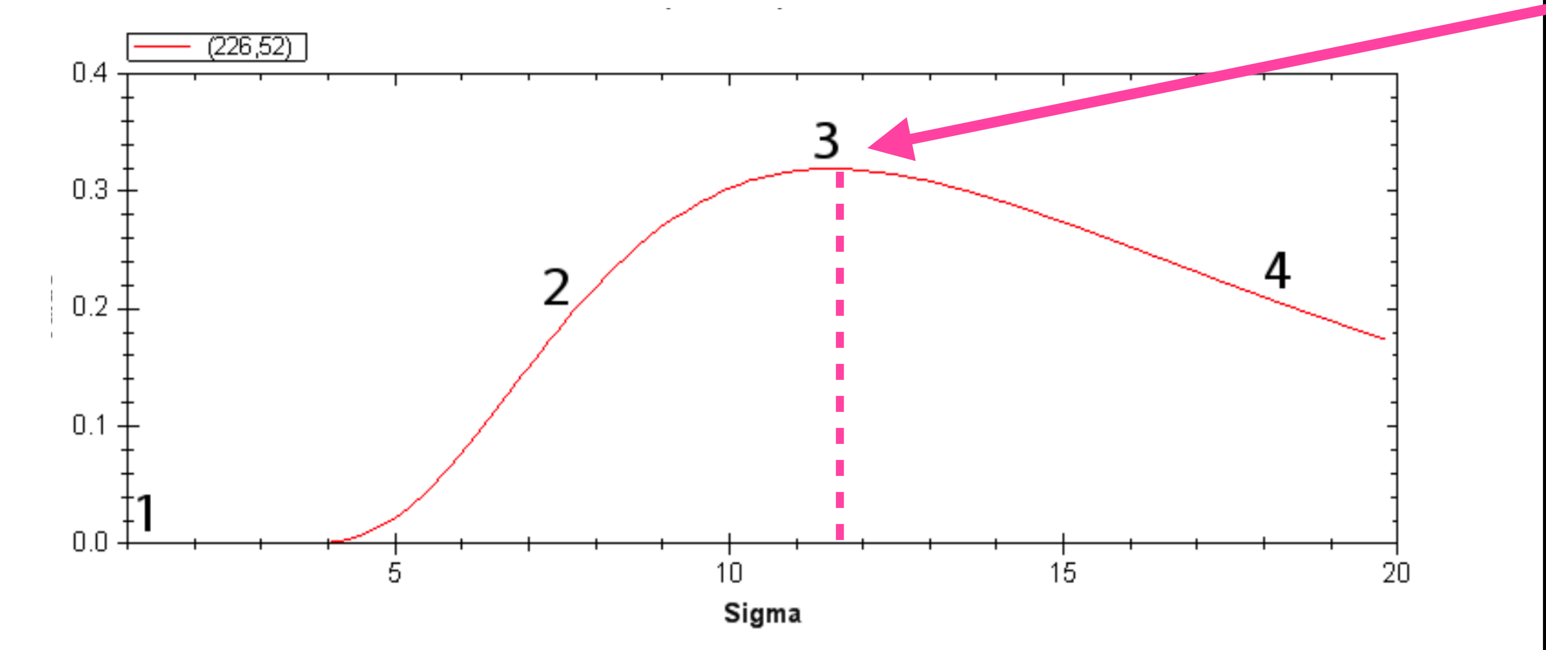
(Scale 1) $\sigma = 1$

(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 can achieve scale invariance by accurately estimating the scale of a structure, then normalising

We obtain scale invariance 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 <u>infinite number</u> 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 scales, 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 <u>impossible</u> 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 <u>artefacts</u> 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

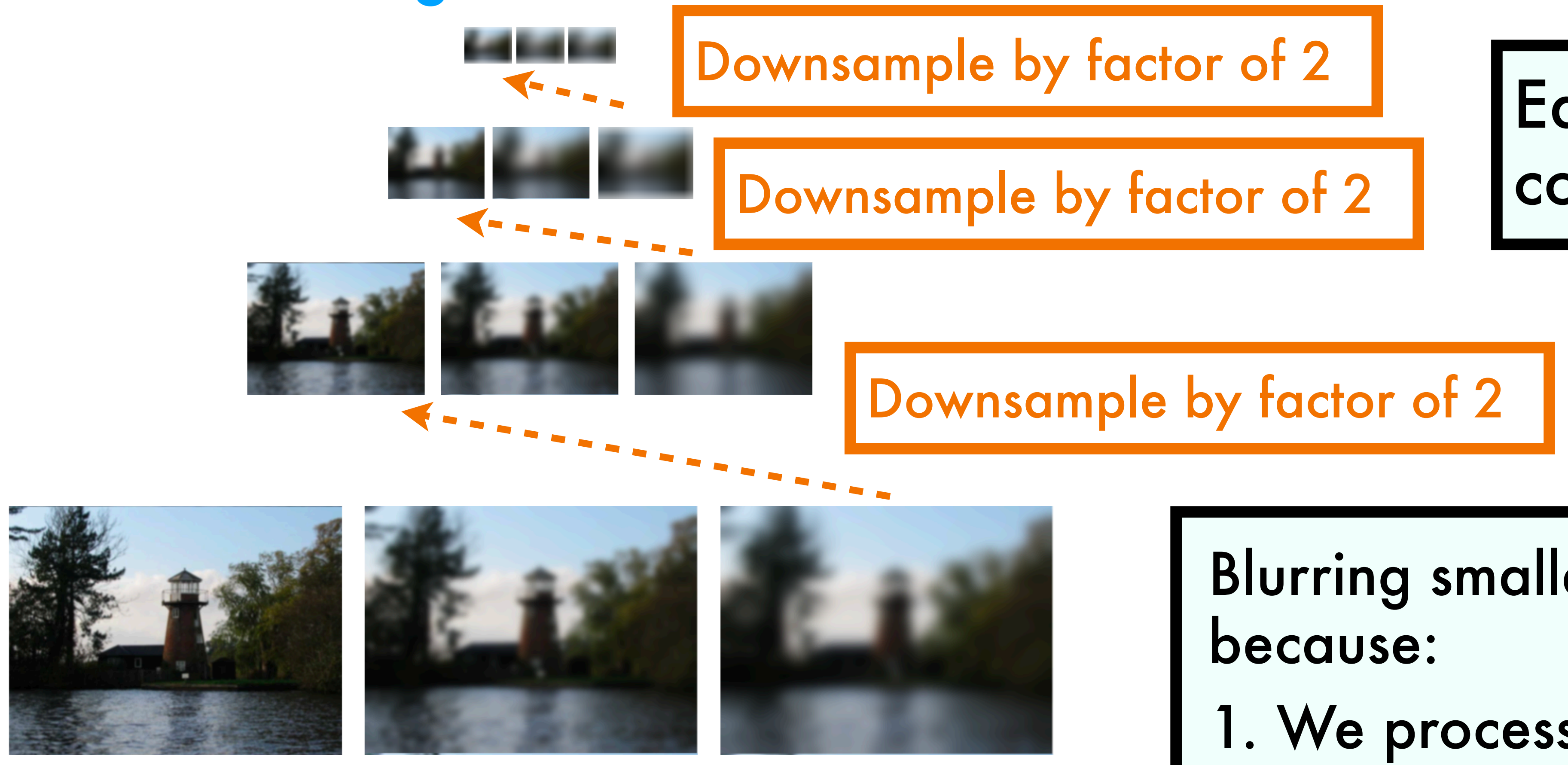
$$\sigma_i = 2^{\frac{i}{s}}\sigma_0$$

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!



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

Each layer of the pyramid corresponds to one octave.

Blurring smaller images is 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 $\sigma_{i+1} = 2^{\frac{1}{s}}\sigma_i$

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

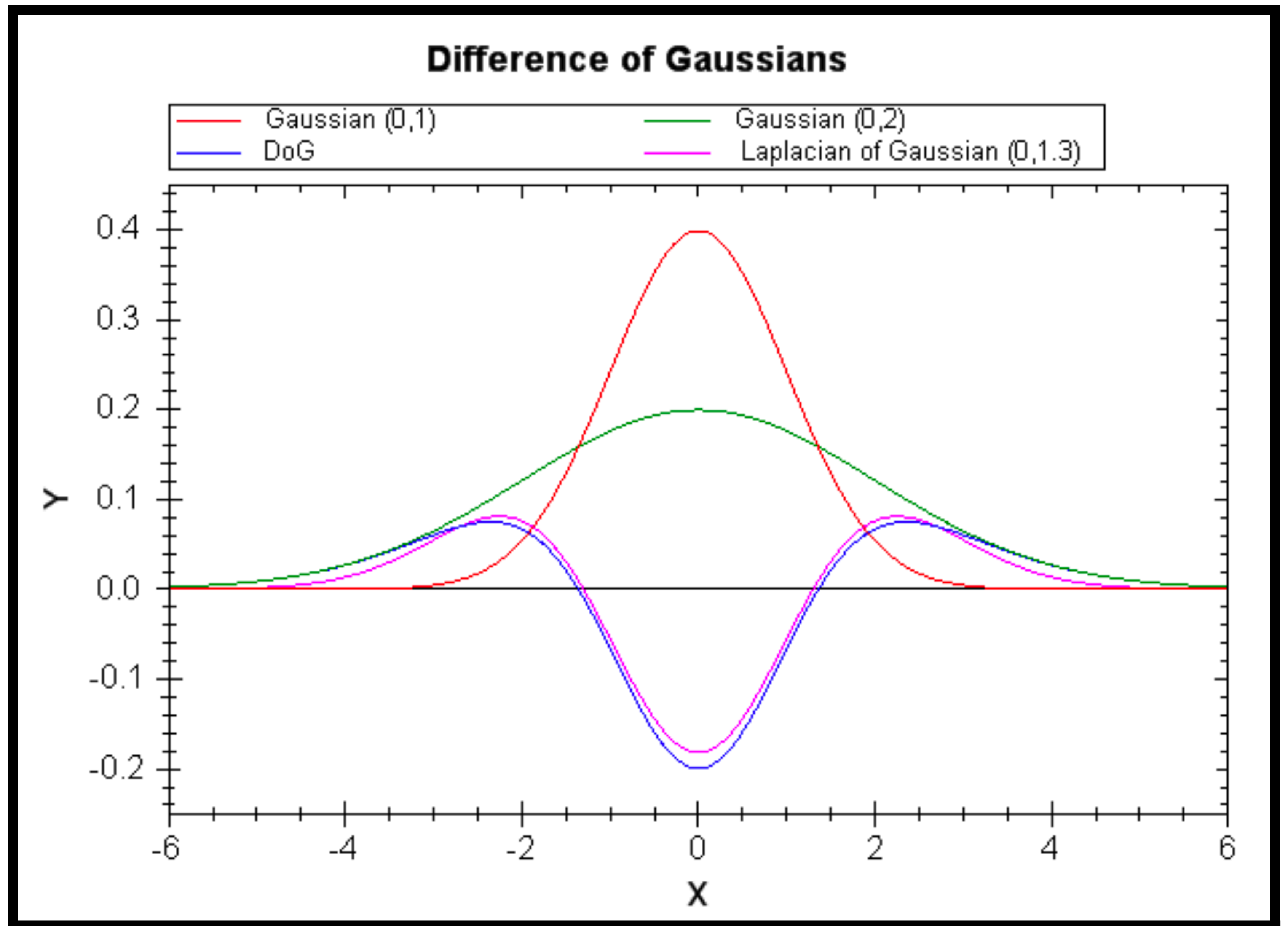
The Difference of Gaussians filter (or "DoG" as it is often called), is also a blob detector

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!

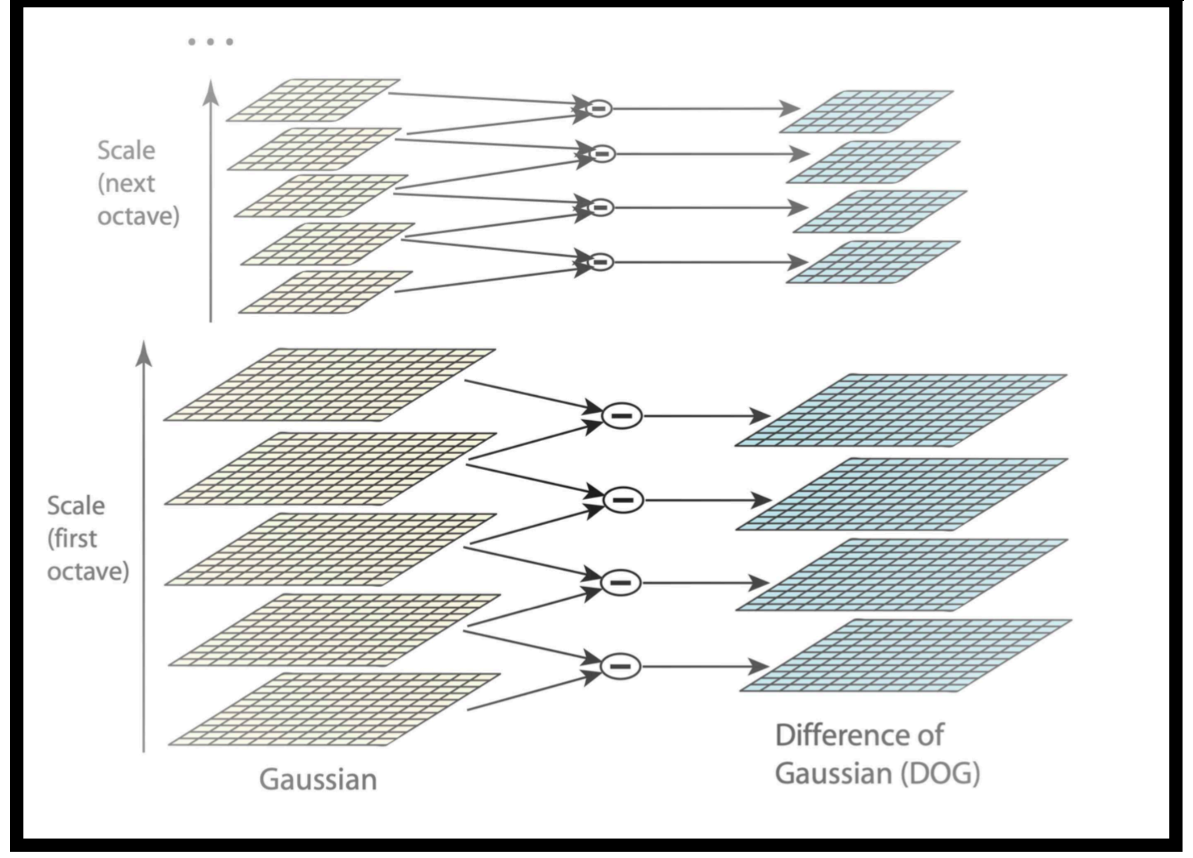
Putting it together: efficient scale-invariant keypoint detection

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

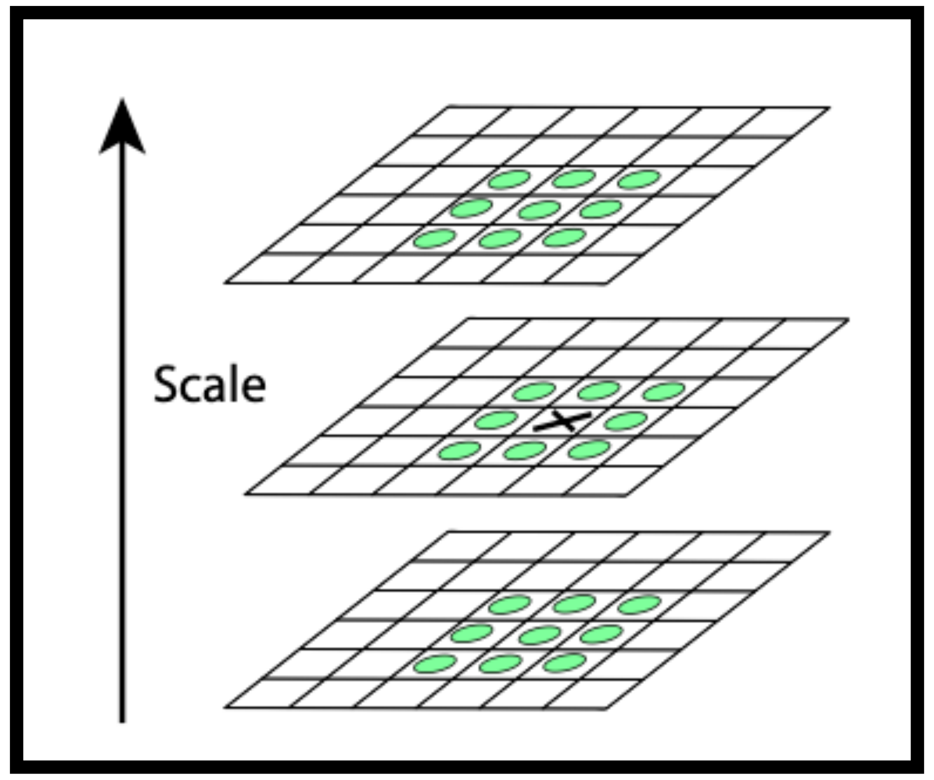
This is done efficiently by subtracting neighbouring images of same dimension in the Image Pyramid[1]

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*

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)