Computer Vision (7CCSMCVI / 6CCS3COV)

Recap

- **Image formation**
  - physics of image formation (light, reflectance, optics)
  - geometry of image formation (pinhole camera model)
  - digital images (digitisation, representation)
  - the eye
- **Low-level vision** ←Today
- **Mid-level vision**
- **High-level vision**

Today

- Edge and Feature Detection
  - important of recognition
  - sufficient for recognition
- Requires filtering
  - linear filter - replaces each pixel by a linear combination of itself and its neighbours
  - generates a new image
- Requires convolution
  - the process of applying the filter to the image

Convolution

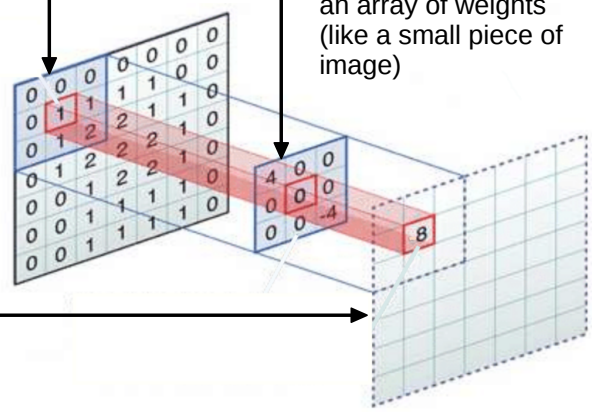
$$I'(i,j) = I * H = \sum_{k,l} I(i-k, j-l) H(k,l)$$

filtered image

mask (or template, or kernel)

an array of weights (like a small piece of image)

at each image location (i,j) a new value is calculated as a weighted sum of pixel values in the neighbourhood (defined by the range of values taken by k and l)



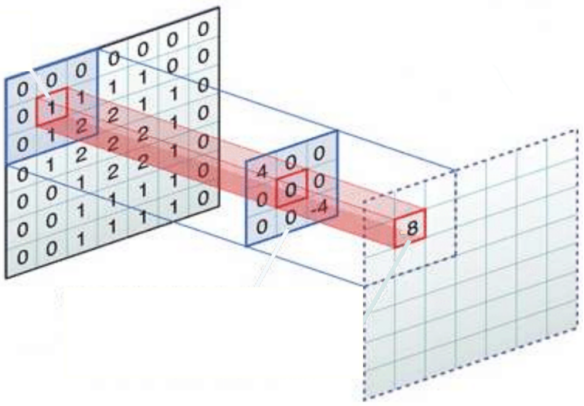
Convolution

$$I'(i,j) = I * H = \sum_{k,l} I(i-k, j-l) H(k,l)$$

To calculate values at each location in the filtered image:

You can imagine sliding the mask across the input image, filling in the values for the output (filtered) image as you go.

Alternatively, you can imagine the mask replicated at every pixel location in the output image, and the results generated in parallel (like the DoG filters in the retina).



Convolution and Correlation

convolution:

$$I'(i,j) = I * H = \sum_{k,l} I(i-k, j-l) H(k,l)$$

$$= \sum_{k,l} I(i+k, j+l) \underbrace{H(-k,-l)}_{\text{rotated mask}}$$

대칭 시켜준다

cross-correlation:

$$I'(i,j) = \sum_{k,l} I(i+k, j+l) H(k,l)$$

Hence, if mask is not rotated the result will be the cross-correlation rather than the convolution of I and H.

If mask is symmetrical (i.e. $H(k,l) = H(-k,-l)$) then cross-correlation = convolution.

Convolution: method

$H * I$

1	1	0
1	0	-1
0	-1	-1

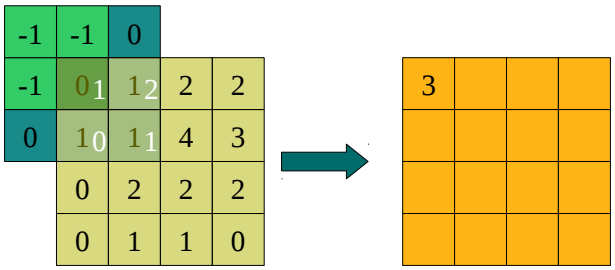
1	2	2	2
0	1	4	3
0	2	2	2
0	1	1	0

Rotate the mask:

-1	-1	0
-1	0	1
0	1	1

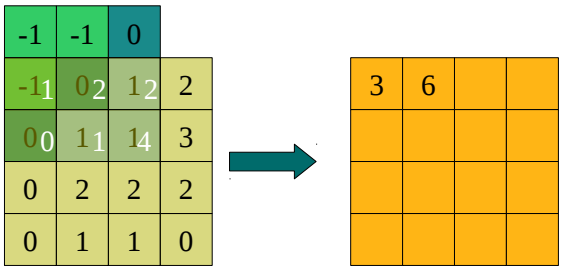
Convolution: method

For each image pixel in turn:
1. Centre the **rotated** mask over that pixel
2. Multiply each mask element by the corresponding image pixel value
3. Sum these products and write answer in corresponding pixel location in the output image



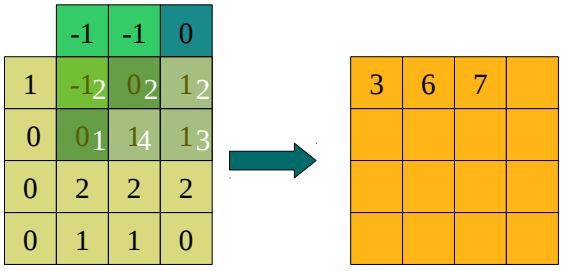
Convolution: method

For each image pixel in turn:
1. Centre the **rotated** mask over that pixel
2. Multiply each mask element by the corresponding image pixel value
3. Sum these products and write answer in corresponding pixel location in the output image



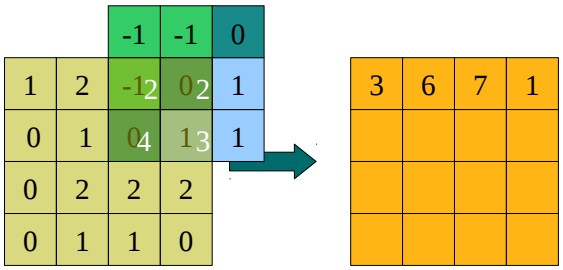
Convolution: method

For each image pixel in turn:
1. Centre the **rotated** mask over that pixel
2. Multiply each mask element by the corresponding image pixel value
3. Sum these products and write answer in corresponding pixel location in the output image



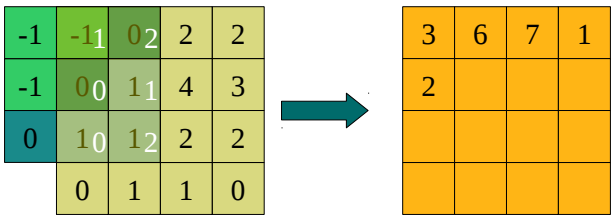
Convolution: method

For each image pixel in turn:
1. Centre the **rotated** mask over that pixel
2. Multiply each mask element by the corresponding image pixel value
3. Sum these products and write answer in corresponding pixel location in the output image



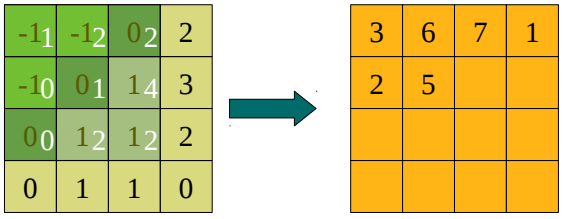
Convolution: method

For each image pixel in turn:
1. Centre the **rotated** mask over that pixel
2. Multiply each mask element by the corresponding image pixel value
3. Sum these products and write answer in corresponding pixel location in the output image



Convolution: method

For each image pixel in turn:
1. Centre the **rotated** mask over that pixel
2. Multiply each mask element by the corresponding image pixel value
3. Sum these products and write answer in corresponding pixel location in the output image



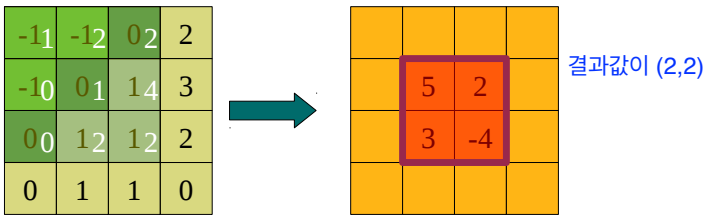
Convolution at image boundaries

How to deal with image boundaries (where mask "falls off" image)?

Several possibilities. Two most common:

make the output image smaller than the input image (red area in example below). i.e. only apply mask at locations on the input image where it does not fall off the edge.

pad the input image with zeros (area outside the black square in example below). i.e. the implicit assumption made in the preceding example.



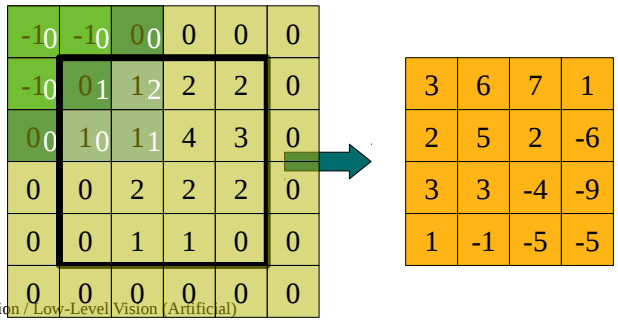
Convolution at image boundaries

How to deal with image boundaries (where mask "falls off" image)?

Several possibilities. Two most common:

make the output image smaller than the input image (red area in example below). i.e. only apply mask at locations on the input image where it does not fall off the edge.

pad the input image with zeros (area outside the black square in example below). i.e. the implicit assumption made in the preceding example.



Separable Masks

A 2D convolution is separable if the kernel H can be written as a convolution of two (row) vectors

i.e. if $H = h_1 * h_2^T$

A separable convolution can be performed as two 1D convolutions

$$I*H = I * (h_1 * h_2^T)$$
$$= (I * h_1) * h_2^T$$

This is faster

Note $h_1 * h_2^T = h_2^T \times h_1$ (i.e. the convolution of two vectors equals the product of those vectors)

Separable convolutions are much more efficient to compute.

Convolving an image with an $k \times l$ pixel kernel takes:

$k \times l$ products per pixel for 2D mask

$k + l$ products per pixel for two 1D masks

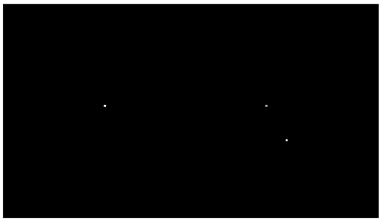
Masks as point-spread functions

Convolve a mask with a simple image that has just an isolated white dot on a black background.

The output will be the mask itself shifted by the row and column numbers of the isolated pixel in the input.

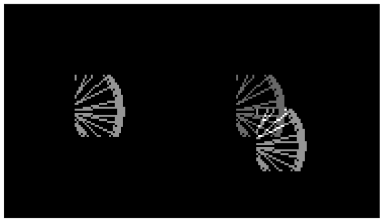
An ordinary image can be thought of as a combination of such points - one for each pixel. So the result of a convolution can be thought of as a superimposition of masks, each one weighted by the grey-level of an image pixel.

Input Image



Mask

Output Image



Masks as templates

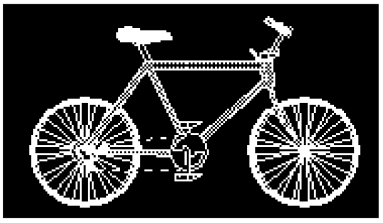
mask template matching

The convolution output is a maximum when large values in the input get multiplied by large values in the mask.

This means that convolution masks respond most strongly to image features that resemble the rotated mask.

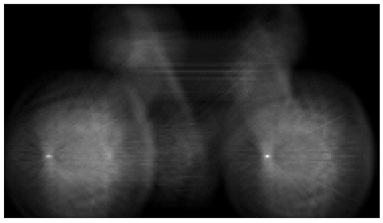
The rotated mask is like a template which is scanned across the image to find image features that match that template.

Input Image



Mask

Output Image



peak intensity value

Mask examples 0 (silly)

The values chosen for the mask determine the effects of the convolution. Example masks:

0	0	0
0	1	0
0	0	0

Each pixel replaced with itself.
Has no effect.
Not very useful!

0	0	0
0	0	1
0	0	0

Each pixel replaced by the one on the left.
Shifts image one pixel to the right.
Not very useful!

Mask examples I (smoothing)

The values chosen for the mask determine the effects of the convolution. Example masks:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

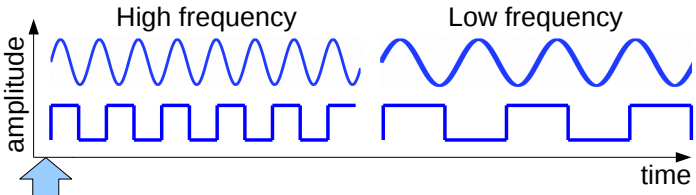
Each pixel replaced by average of itself and its eight neighbours.

Generates a smoothed (blurred) image.

Useful.

Called "mean filter" or "box mask".

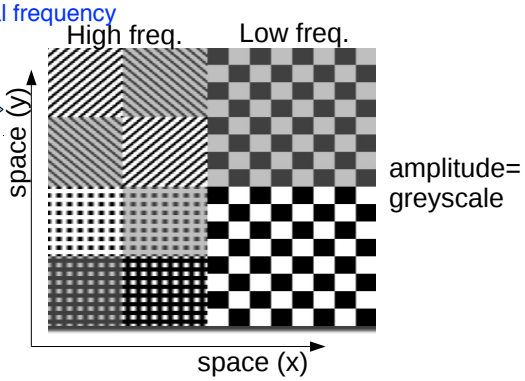
Spatial Frequency



Frequency

- **in temporal domain**
- **in spatial domain**

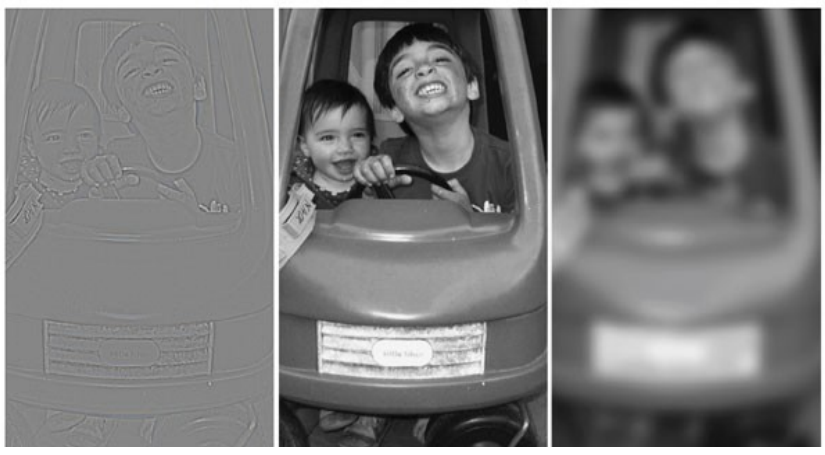
high spatial frequency



Small-scale texture is said to have a high spatial frequency.

Smoothing removes this, leaving low spatial frequencies.

Spatial Frequency: example

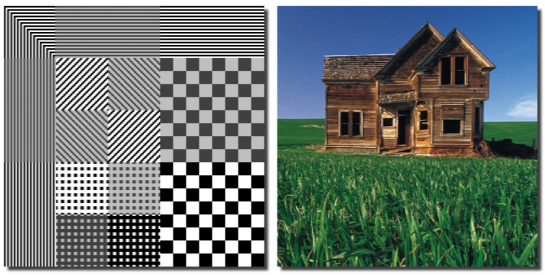


High Low

high low 사진을 겹치면 가운대 사진이 나온다.

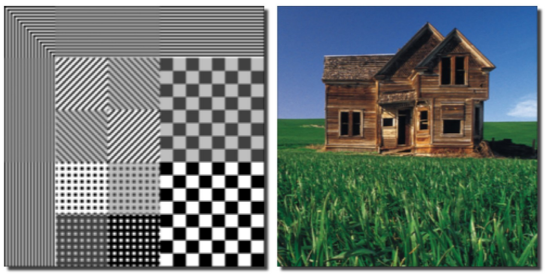
Smoothing mask example

Original Images



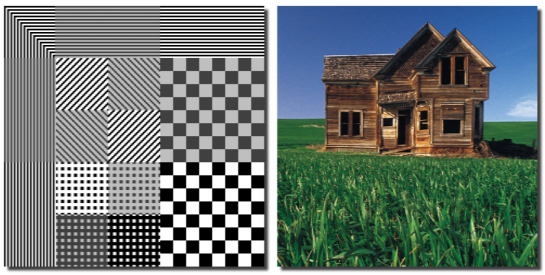
Images convolved with 3x3 box mask

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

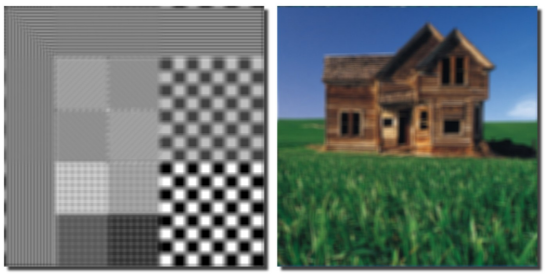


Smoothing mask example

Original Images

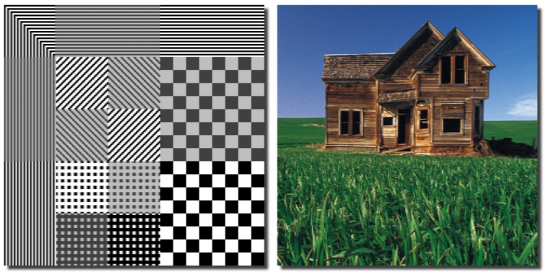


Images convolved with 9x9 box mask

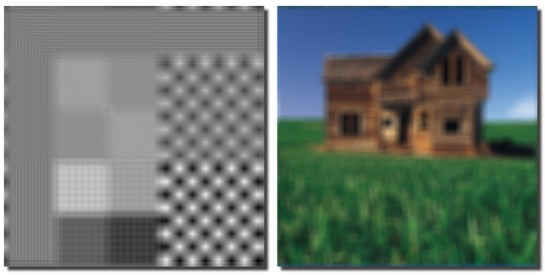


Smoothing mask example

Original Images



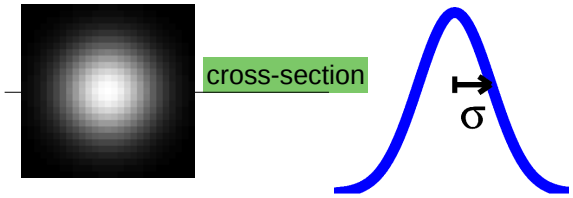
Images convolved with 17x17 box mask



Gaussian mask

The box mask is not very good as it has sharp edges.

The Gaussian mask is more effective as it falls off gently at the edges. It gives more weight to nearby pixels.

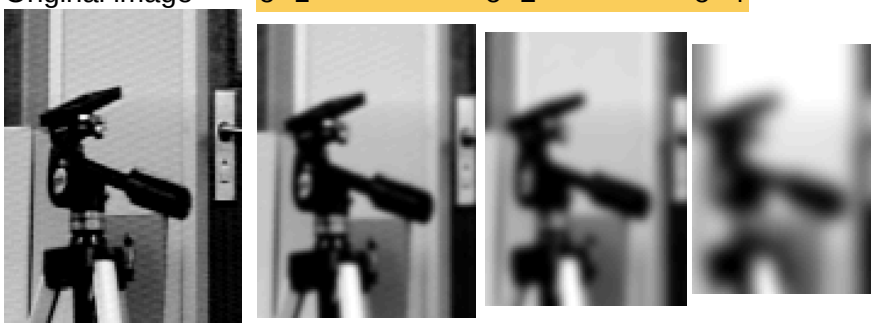


$$G(x,y)=\frac{1}{2\pi\sigma^2}\exp\left(\frac{-(x^2+y^2)}{2\sigma^2}\right)$$

The Gaussian mask effectively removes any spatial frequencies with a period or wavelength smaller than the mask dimensions (defined by σ = standard deviation, or "**scale**").

Smoothing with Gaussian mask

Original image σ=1 σ=2 σ=4



sigma 값이 커지면 mask 값도 비례적으로 커져야 한다

"Scale" = the standard deviation of the Gaussian (i.e. σ)

as this parameter goes up:

- the size of the mask needs to increase (mask width should be ≥6σ)
- more pixels are involved in the average
- the image gets more blurred (more high frequencies suppressed)
- noise is more effectively suppressed

Separability of the Gaussian mask

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

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

i.e. 2D Gaussian is the product of two 1D Gaussians.
Each 1D Gaussian is a function of one variable only (either x or y)

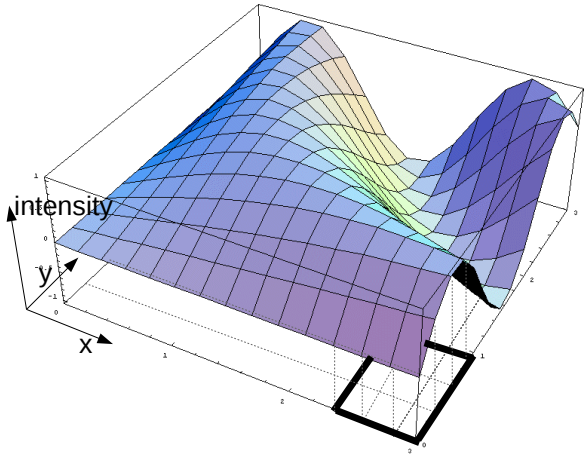
Difference Masks

As well as calculating averages (as in box and Gaussian masks), convolution can also be used to calculate differences.

The difference between pixel values measures the gradient of the intensity values.

Hence:

- smoothing masks approximate integration
- difference masks approximate differentiation

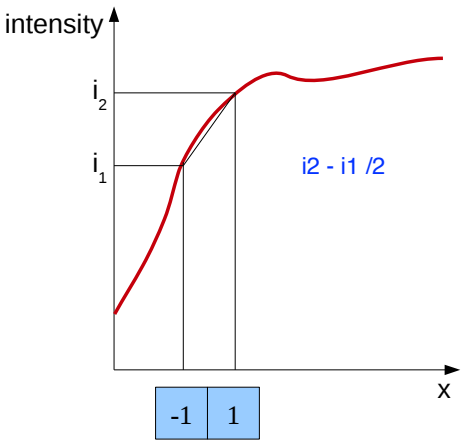


height := intensity value

Difference Masks

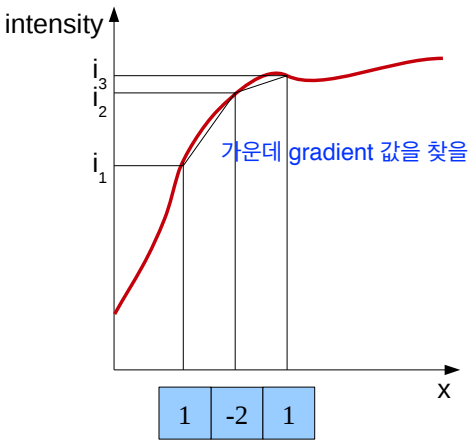
1st derivative mask: estimate of gradient is (i_2-i_1)

단면적 시점



Difference Masks

2nd derivative mask: estimate of change of gradient is $(i_3-i_2)-(i_2-i_1)$



Mask examples 2 (difference)

Differencing masks highlight locations with intensity changes

Simple differencing masks:

vertical — horizontal — diagonals ← Mask orientation

horizontal — vertical — diagonals ← Orientation of intensity change detected

-1 1 $\approx -\delta/\delta y$	-1 1 $\approx -\delta/\delta x$	-1 0 0 1	0 -1 1 0
-1 2 -1 $\approx -\delta^2/\delta y^2$	-1 2 -1 $\approx -\delta^2/\delta x^2$	-1 0 0 0 2 0 0 0 -1	0 0 -1 0 2 0 -1 0 0

Mask weight values

smoothing

The weights in a smoothing mask (any mask in which all values are positive) should add up to 1 to preserve average grey levels.

e.g.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

The weights in differencing masks (any mask with positive and negative values) should add up to 0 to generate a zero response to constant image regions.

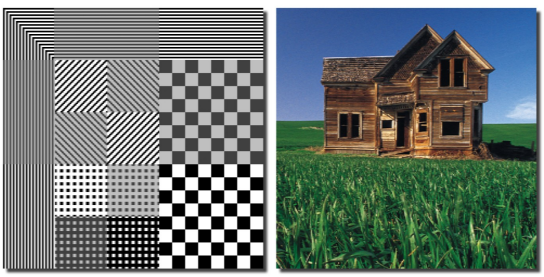
e.g.

-1	-1	-1
-1	8	-1
-1	-1	-1

다 더하고 절대값

Difference mask example

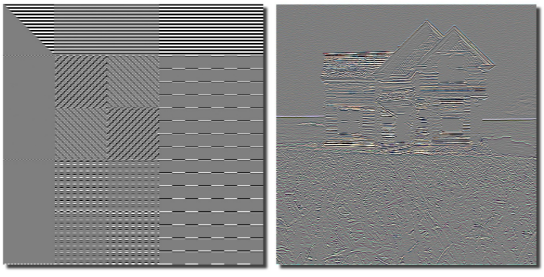
Original Images



Images convolved with vertical difference mask

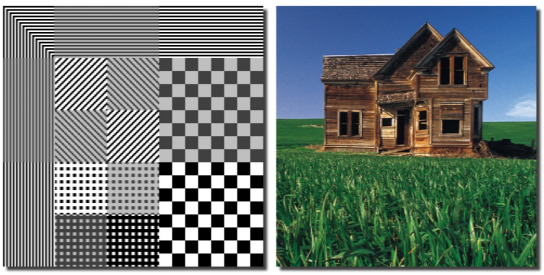
-1
2
-1

$(-\delta^2/\delta y^2)$



Difference mask example

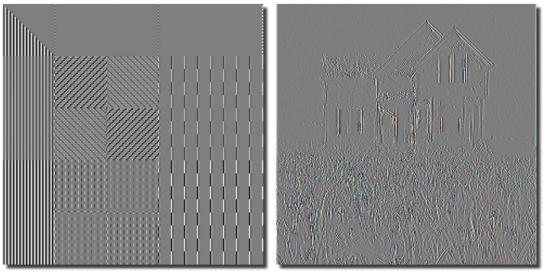
Original Images



Images convolved with horizontal difference mask

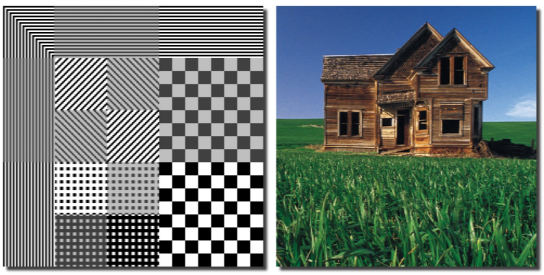
-1	2	-1

$(-\delta^2/\delta x^2)$



Difference mask example

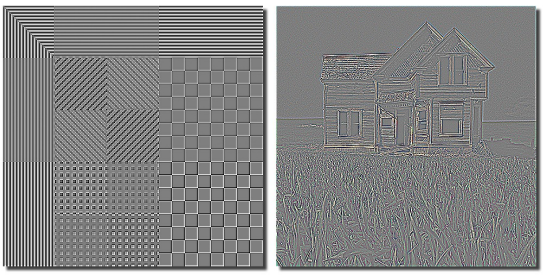
Original Images



Images convolved with horizontal + vertical difference mask

0	-1	0
-1	4	-1
0	-1	0

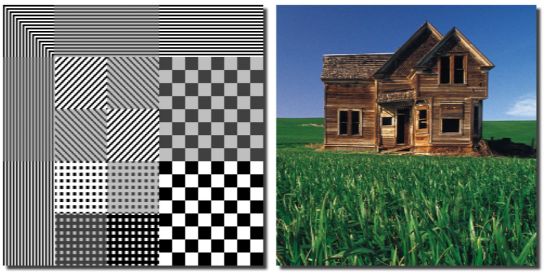
$\approx -\delta^2/\delta x^2 - \delta^2/\delta y^2$



high intensity value

Difference mask example

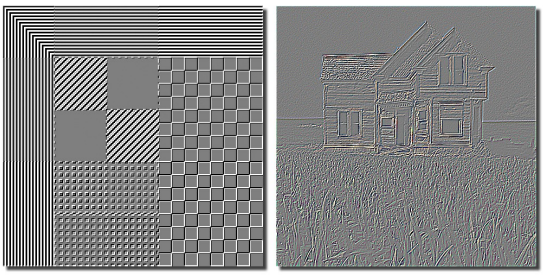
Original Images



Images convolved with diagonal difference mask

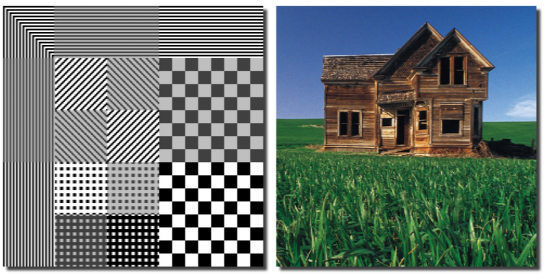
-1	0	0
0	2	0
0	0	-1

diagonal case



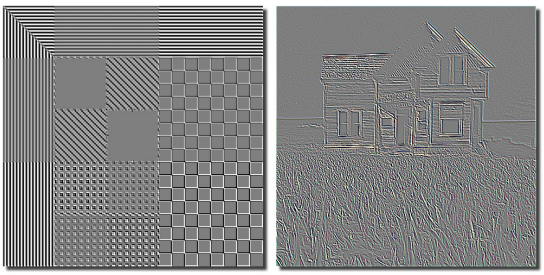
Difference mask example

Original Images



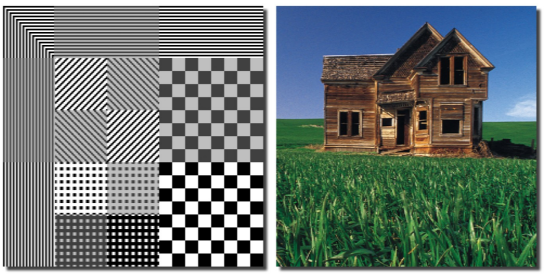
Images convolved with diagonal difference mask

0	0	-1
0	2	0
-1	0	0



Difference mask example

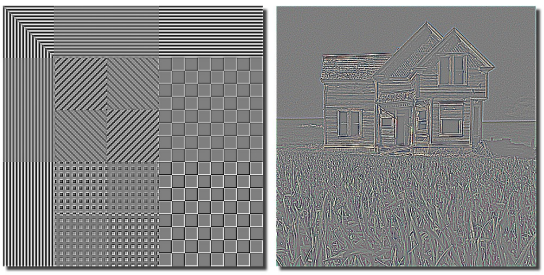
Original Images



Images convolved with vertical + horizontal + both diagonal difference mask

-1	-1	-1
-1	8	-1
-1	-1	-1

$\approx -\delta^2/\delta x^2 - \delta^2/\delta y^2$



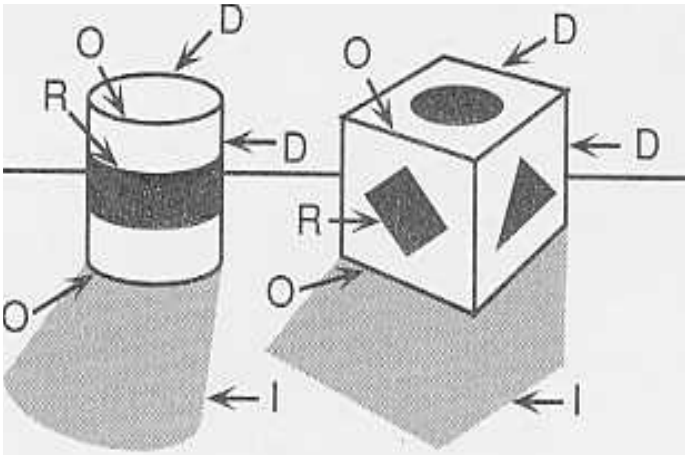
Intensity discontinuities

Discontinuities in the intensity (pixel) values of an image often correspond to useful (meaningful) physical characteristics, e.g. they occur at boundaries of objects, corners, etc. (“features” in general).

These features are useful, as they are often sufficient for recognising the content of an image.

e.g. we can understand an image that only contains edges (but not always!)

Causes of intensity discontinuities



Not just object boundary "edges"

Depth discontinuity (D) – due to surfaces at different distances
Orientation discontinuity (O) – due to changes in the orientation of a surface
Reflectance discontinuity (R) – due to change in surface material properties
Illumination discontinuity (I) – e.g. shadow boundaries not useful

Edge detection

An edge is a location where the magnitude of the intensity gradient is high in the direction orthogonal to the edge

An edge is an image region in which the grey level gradients have a common direction and large magnitudes

Idealised Edge Types:

step edges | ramp | line, bar or impulse | tent, roof

The difference masks considered previously will detect these edges:

-1	1

$-\delta / \delta x$

-1	2	-1

$-\delta^2 / \delta x^2$

The Laplacian mask

Can be seen as a combination of difference masks in each direction.
It therefore detects intensity discontinuities at all orientations.

$*$

-1	-1	-1
-1	8	-1
-1	-1	-1

$\approx -\delta^2 / \delta x^2 - \delta^2 / \delta y^2$

$=$

Note: the Laplacian is actually the additive inverse of this mask.

However, such a mask is most sensitive to a single dot, one pixel in size (i.e. noise).

Edge detection with noise suppression

Convolving with a differencing mask:

- enhances edges
- but also enhances noise

Convolving with a smoothing mask:

- removes noise
- but also blurs edges

There is a trade-off between edge detection and noise suppression.

Edge detection requires:

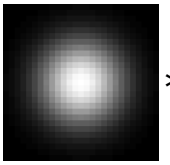
- a spatial scale, established using a smoothing operator (a Gaussian mask)
- a differencing operator to find significant grey-level changes (at that spatial scale)

Laplacian of Gaussian / DoG mask

The standard approach is to combine Gaussian smoothing with a Laplacian operator.

The two masks can be combined by convolution into a single one.

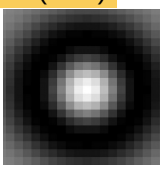
This is called the Laplacian of Gaussian (LoG) mask.



$*$

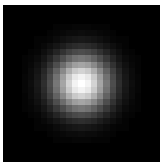
-1/8	-1/8	-1/8
-1/8	1	-1/8
-1/8	-1/8	-1/8

$=$

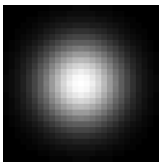


$$\approx -\frac{\delta^2}{\delta x^2} G_\sigma - \frac{\delta^2}{\delta y^2} G_\sigma$$

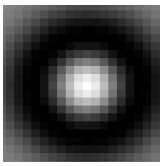
In practice, it is almost always approximated using the Difference of Gaussians or DoG mask (or “centre-surround” or “Mexican hat”).



$-$



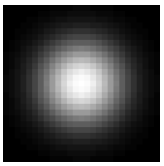
$=$



$$= G_{\sigma 1} - G_{\sigma 2}$$

Gaussian derivative masks

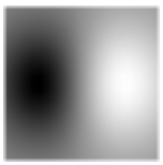
Alternatively, the simple difference operators can be combined with Gaussian smoothing. These tend to produce more robust results than the DoG mask.



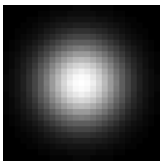
$*$

-1	1

$=$



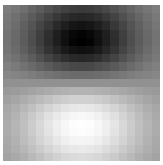
$$= \frac{-\delta}{\delta x} G_\sigma$$ x gradient (at scale σ)



$*$

-1
1

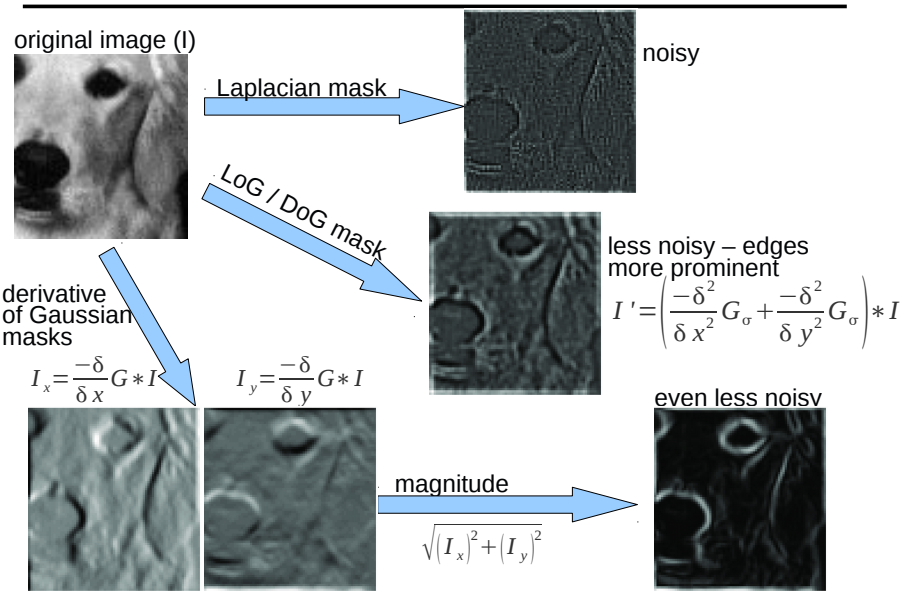
$=$



$$= \frac{-\delta}{\delta y} G_\sigma$$ y gradient (at scale σ)

Such masks emphasise vertical and horizontal structure at the scale set by the σ parameter of the Gaussian component.

Edge detection comparison

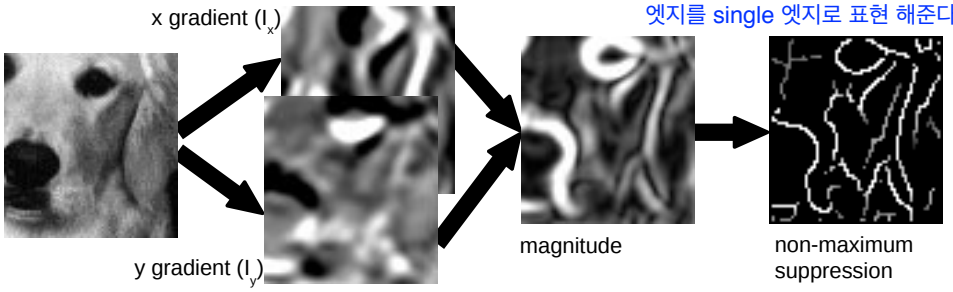


The Canny edge detector

One of the most popular edge detectors is based on Gaussian derivative masks

1. convolution with derivatives of Gaussian masks
2. calculation of magnitude $M = \sqrt{(I_x)^2 + (I_y)^2}$ and direction $D = \arctan(I_y / I_x)$
3. non-maximum suppression (thin multi-pixel wide “ridges” down to a single pixel by removing current pixel if neighbouring pixels perpendicular to the direction of the edge have a higher magnitude)
4. thresholding using a low and a high threshold (accept all edge pixels with a magnitude over low threshold that are connected to a pixel with a magnitude over the high threshold)

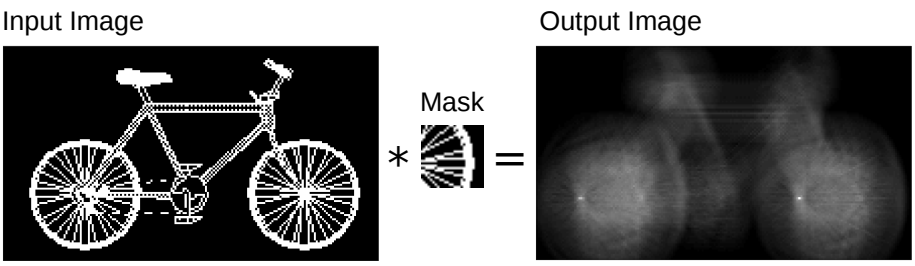
엣지를 single 엣지로 표현 해준다.



 directions / measures which intensity changing at

General algorithm for feature detection

Recall that a mask can be viewed as a template for a particular image feature (one identical to the rotated mask).



- Convolve the image with one or more masks which have large responses to the feature in question.
- Choose a threshold. 찾기 힘들다
- Detect the feature in those parts of the image where the convolved image exceeds the threshold.

Template matching across scales

Convolution scans the template across the image to find locations where the image matches the template

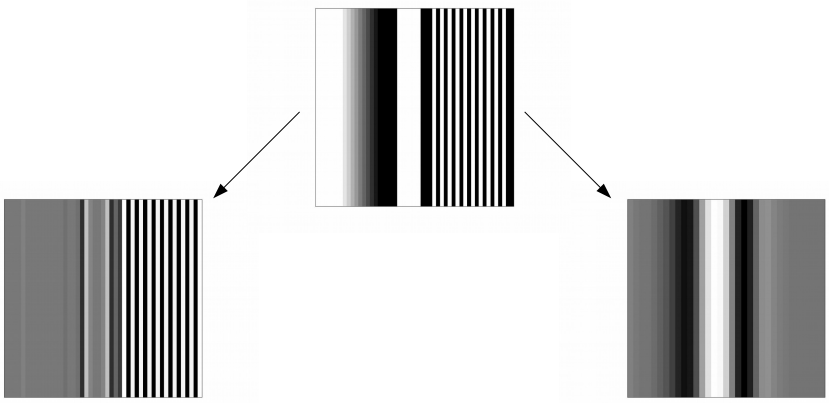
However, we don't know the scale of the feature we are trying to find, as this will depend on the unknown parameters of the image formation process

How to find image features with invariance to scale?

Edge detection across scales

Particularly, how to find edges with invariance to scale? How to find thick and thin edges? Boundaries and textures?

The size of edge detected by an edge detector will depend on the scale of the Gaussian smoothing mask used.

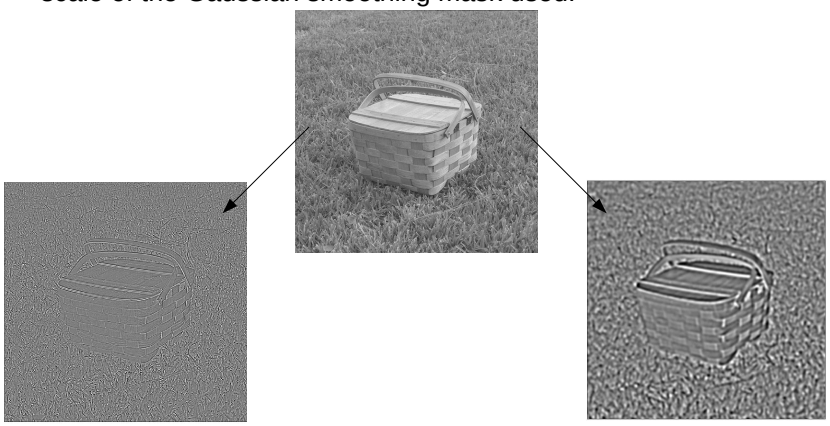


σ=0.5 image convolved with LoG mask with σ=2

Edge detection across scales

Particularly, how to find edges with invariance to scale? How to find thick and thin edges? Boundaries and textures?

The size of edge detected by an edge detector will depend on the scale of the Gaussian smoothing mask used.



σ=0.5 image convolved with LoG mask with σ=3

Image Pyramid

To find image features with invariance to scale, we could:

1. apply filters of different sizes to the image
2. apply filters of a fixed size to the image presented at different sizes

The 2nd method is usually used. This is called an image pyramid.

Decreasing the scale of an image is achieved by down-sampling (or sub-sampling): $S\downarrow(I)(i,j)=I(si,sj)$

(create a new image by taking every sth pixel of the original image).

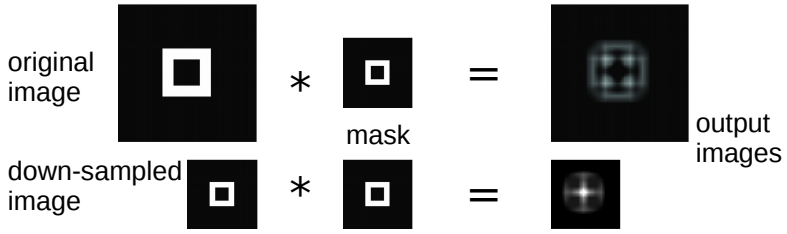
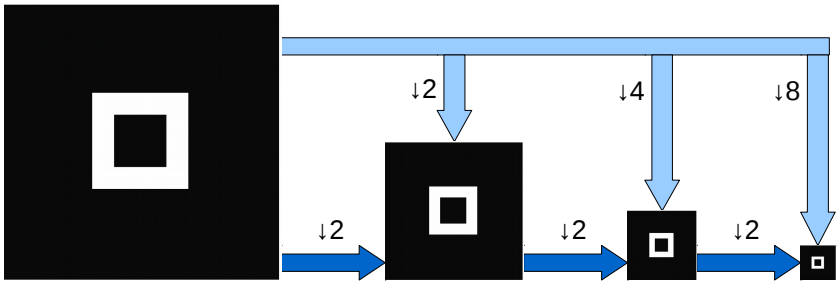


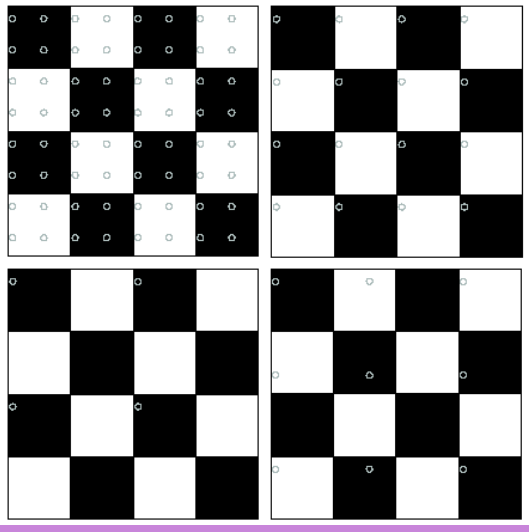
Image Pyramid creation

To find image features across a range of scales, we need a number of images at different scales.



Rather than resampling the original image, can sub-sample the previous sub-sampled image.

Down-sampling problem: aliasing



Sample chess board image at each circle:

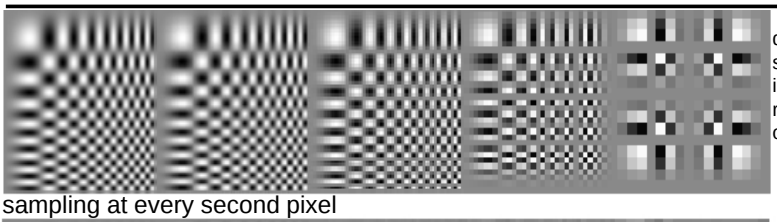
← Good sampling (output image resembles input)

← Bad sampling (aliased)

Sample may not be representative of image region.

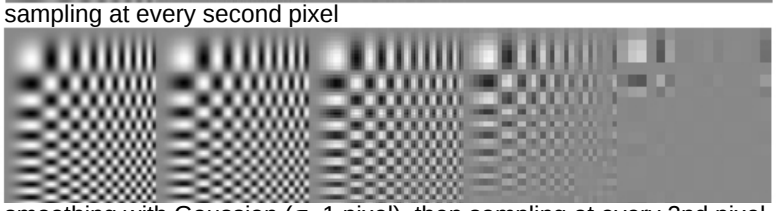
Need to take account of larger region then just taking value from a single pixel

Down-sampling solution: smoothing

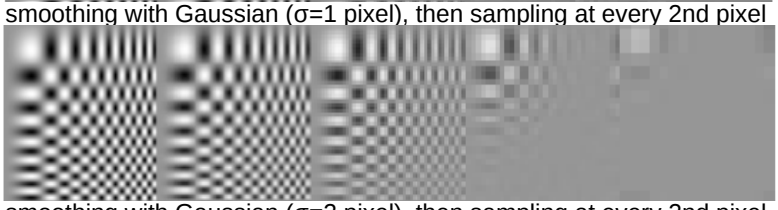


down-sampled image mis-represents original

sampling at every second pixel



smoothing with Gaussian (σ=1 pixel), then sampling at every 2nd pixel



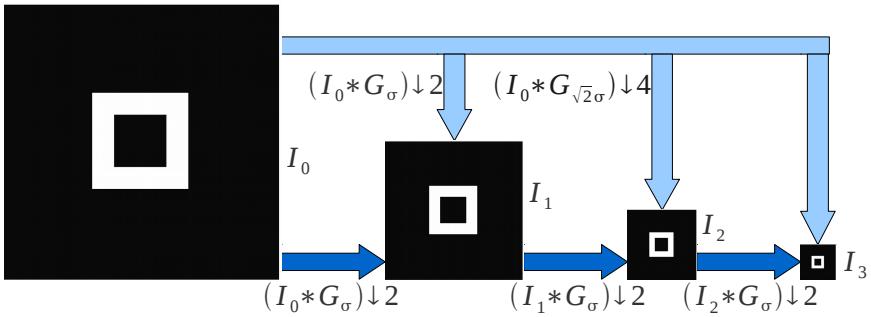
smoothing with Gaussian (σ=2 pixel), then sampling at every 2nd pixel

Gaussian Pyramid creation

If we smooth an image with a Gaussian having a standard deviation of σ, and then convolve the result with a Gaussian having a standard deviation of σ, we get the same result as smoothing the original image with a Gaussian with a standard deviation of $\sqrt{2}\sigma$

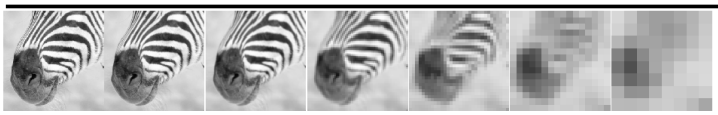
i.e. Gaussian * Gaussian=another Gaussian

$$G_\sigma * G_\sigma = G_{\sqrt{2}\sigma}$$



In general: $I_i = S\downarrow(I_{i-1} * G_\sigma)$

Gaussian Pyramid example



512 256 128 64 32 16 8

Laplacian Pyramid

Recall that:

- a Laplacian of Gaussian (LoG) mask detects intensity discontinuities (e.g. edges) at all orientations
- generally approximated by a Difference of Gaussians (DoG)

A laplacian image pyramid is an image pyramid that highlights intensity discontinuities at multiple scales.

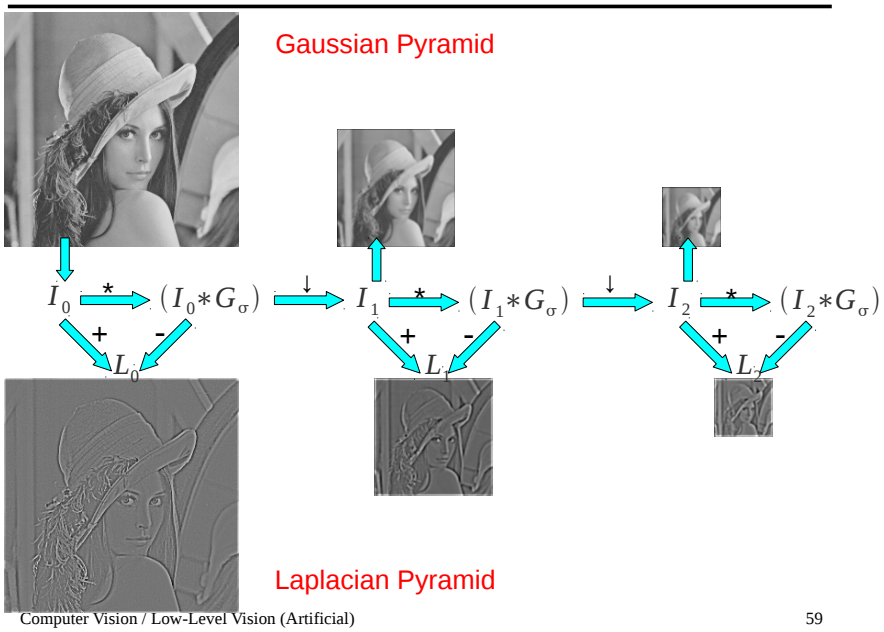
A Gaussian pyramid already provides us with images that have been convolved with Gaussians of different scales.

A difference of Gaussians can therefore be obtained by subtracting images taken from the Gaussian pyramid.

In general: $L_i = I_i - S\uparrow(I_{i+1})$ (an image in the Laplacian pyramid)

where: $I_i = S\downarrow(I_{i-1} * G_\sigma)$ (an image in the Gaussian pyramid)

Laplacian Pyramid creation



Summary

Linear Filtering:

Creates a new image with pixel values that are weighted sums of original pixel values

Requires:

- Convolution – the procedure for calculating the new pixel values
- Mask – the weights used to calculate the new pixel values

Simple (common) Masks:

Smoothing – box, Gaussian

Difference – 1st and 2nd derivatives in x and y directions,
2nd derivative in both directions (Laplacian)

Edge Detection:

Combines smoothing and differencing (to find intensity changes at a certain scale)

Gaussian derivative mask – 1st derivative of Gaussian in x and y directions

LoG/DoG mask – 2nd derivative of Gaussian in both directions

Image Pyramids:

Important for describing and searching an image at all scales

Gaussian pyramid – smoothed and downsampled $I_i = S\downarrow(I_{i-1} * G_\sigma)$

Laplacian pyramid – DoG at each scale $L_i = I_i - S\uparrow(I_{i+1})$