This week

- Edge and Feature Detection
  - important for subsequent analysis
- Requires Filtering
  - linear filter - replaces each pixel by a linear combination of itself and its neighbours
  - generates a new image
  - low-level vision = image processing
- Requires Convolution
  - the process of applying the filter to the image

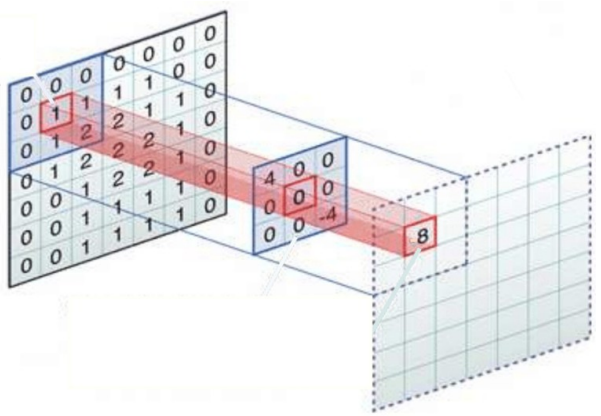
Convolution

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

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

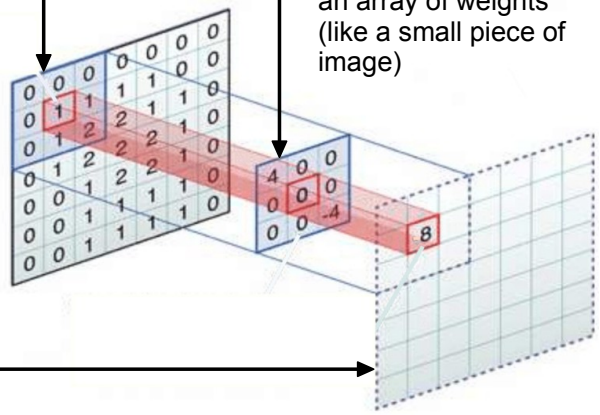
$$I'(i,j) = I * H = \sum_{k,l} I(i-k, j-l) H(k,l)$$

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 – in this example k and l range from -1 to +1)



Convolution: method

$H * I$

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

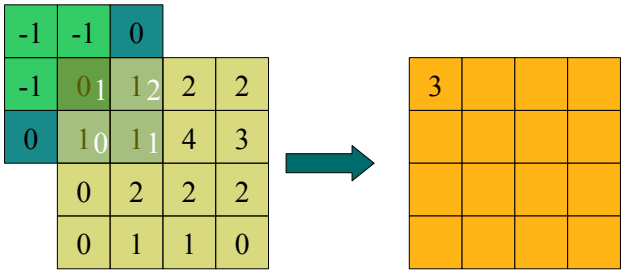
1	2	2	2
0	1	4	3
0	2	2	2
0	1	1	0

Rotate the mask:

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

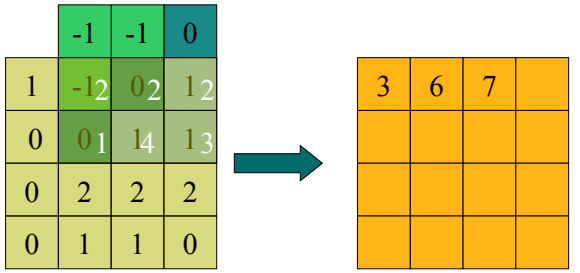
Convolution: method

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



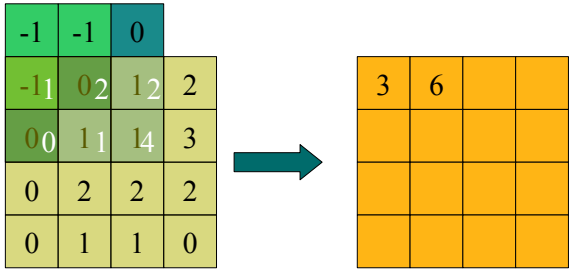
Convolution: method

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



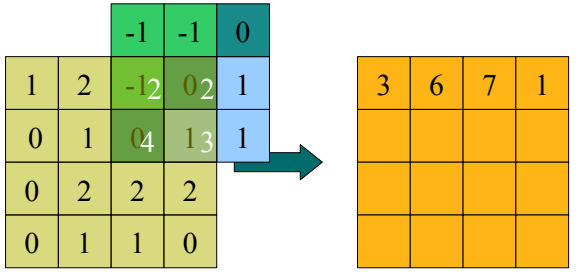
Convolution: method

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



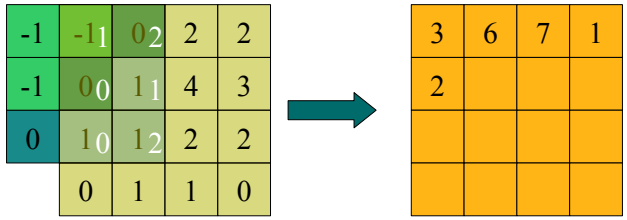
Convolution: method

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

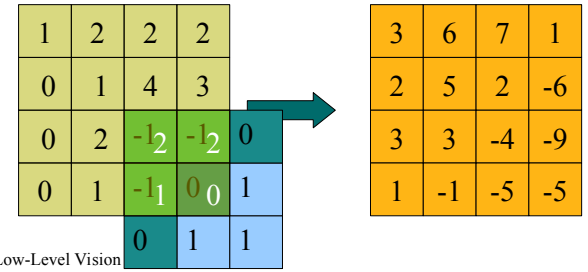
For each image pixel in turn:
1. Centre the **rotated** mask over that pixel
2. Multiply each mask element by the corresponding image pixel value
3. Sum these products and write answer in corresponding pixel location in the output image



Convolution: method

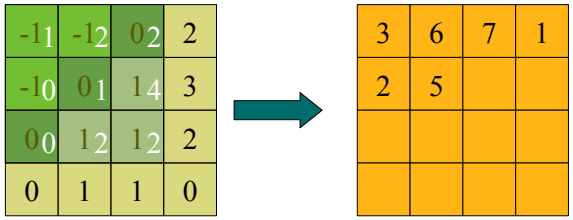
For each image pixel in turn:
1. Centre the **rotated** mask over that pixel
2. Multiply each mask element by the corresponding image pixel value
3. Sum these products and write answer in corresponding pixel location in the output image

skipping to the end:



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

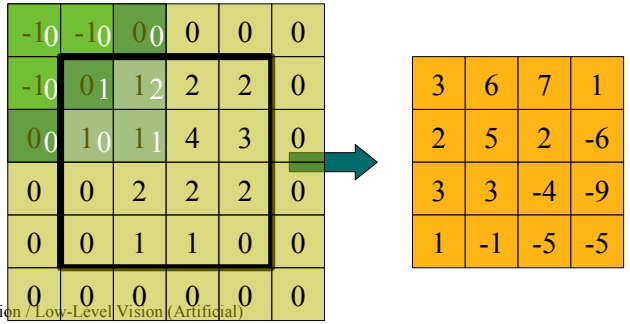


Convolution at image boundaries

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

Most common methods:

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

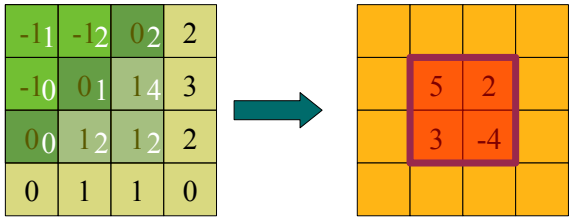


Convolution at image boundaries

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

Most common methods:

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



Convolution and Correlation

convolution:

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

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

cross-correlation:

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

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 and Correlation

convolution is:

- commutative $I * H = H * I$
- associative $(I * H) * G = I * (H * G)$

cross-correlation is **not**:

- commutative $I \star H \neq H \star I$
- associative $(I \star H) \star G \neq I \star (H \star G)$

For this reason convolution is used in preference to cross-correlation (despite the inconvenience of needing to rotate the mask)

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

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

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

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 m×n pixel kernel takes:
m×n products per pixel for 2D mask
m+n 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.

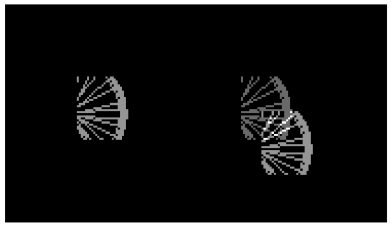
The output will be the mask itself shifted by the row and column numbers of the isolated pixel in the input.

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 intensity of an image pixel.

Input Image

Mask

Output Image



Mask weights values

The values chosen for the mask determine the effects of the convolution.

In theory, we could choose any values for the weights.

In practice, two categories of mask are commonly used:

- smoothing masks
- differencing masks

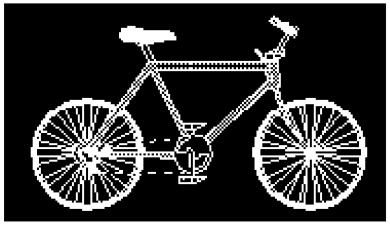
Masks as templates

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.

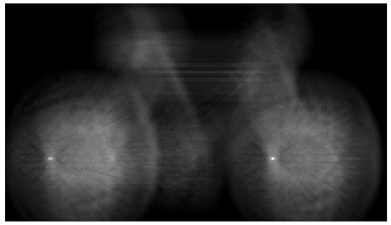
The rotated mask is like a template which is scanned across the image to find image features that match that template.

Input Image



Mask

Output Image



Mask weight values

The weights in a smoothing mask 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 add up to 0 to generate a zero response to constant image regions.

e.g.

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