2 - Image filtering

Spatial filtering

Pixel operator: Computes an output value at each pixel location, based on the input pixel value

- Transform pixel based on its value
- Gamma correction

Local operator: Computes an output value at each pixel location, based on a neighbourhood of pixels around the input pixel

- Transform pixel based on its neighbours
- e.g. sharpening filter

Linear filtering: Output pixel's value is a weighted sum of a neighbourhood around the input pixel

Cross-correlation vs. Convolution

$$g(i,j) = h(u,v) * f(i,j)$$

Output image g Kernel h Input image f

Convolution operator

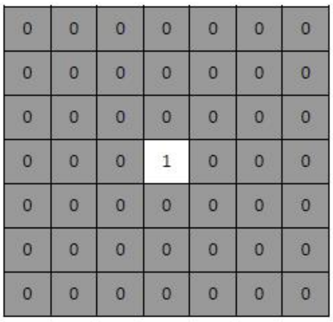
$$g(i,j) = \sum_{u,v} f(i-u, j-v)h(u,v)$$

$$g(i,j) = h(u,v) \otimes f(i,j)$$

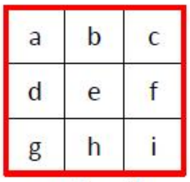
Output image g Kernel h Input image f

Cross-correlation convolution

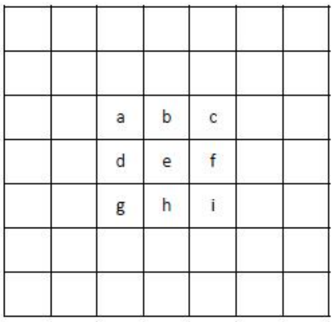
$$g(i,j) = \sum_{u,v} f(i+u, j+v)h(u,v)$$



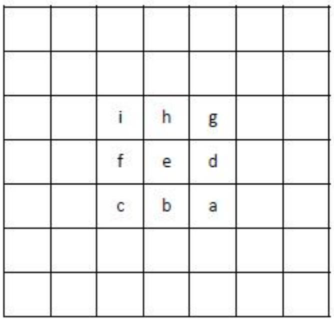
F[x,y]



H[u,v]



F[x,y] ∗ H[u,v]



F[x,y] ⊗ H[u,v]

- Cross-correlation: overlay filter on image
- Convolution: flip filter horizontally and vertically
- They are operations that apply a linear filter to an image.
- Illustration: L2.1 P13-18

Common filters

- Average/blur filters: average pixel values, blur the image
- Sharpening filters: subtract pixel from surround, increase fine detail
- Edge filters: compute difference between pixels, detect oriented edges in image

0	0	0
0	1	0
0	0	0

No change. Behave like a pixel operator

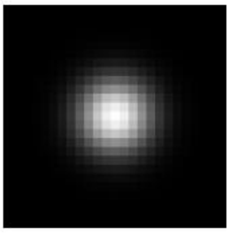
0	0	0
0	0	1
0	0	0

Shift left by 1 pixel

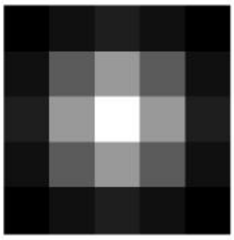
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter. Accentuates differences with local average.

Gaussian



$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$



Gaussian kernel
Kernel size: 5 x 5 px
$\sigma = 1$

Sobel: L2.1 P26, 27

- Detect edges (Vertical, Horizontal)
- More weight on center which provides a bit smoothing.

Colour image filtering: Multiple channels

- Convolve each layer: 2D convolution in each colour channel -> Output is 3 channels
- 3D kernel: Output is 1 channel

Filter design examples

- Diagonal edges: e.g. [[0, 1, 2], [-1, 0, 1], [-2, -1, 0]]
- Simulate (linear) motion blur: e.g. 1D gaussian layer

Filters in practice

Properties of linear filters

- Commutative: $f * h = h * f$
  - Theoretically, no difference between kernel and image
  - But most implementations do care about order
- Associative: $(f * h1) * h2 = f * (h1 * h2)$
  - Usually one option is faster than the other – allows for more efficient implementations
- Distributive over addition
  - $f * (h1 + h2) = (f * h1) + (f * h2)$
- Multiplication cancels out
  - $kf * h = f * kh = k(f * h)$

Efficient filtering

- Multiple filters: generally more efficient to combine 2D filters ($h1*h2*h3$...) and filter image just once L2.1 P35
- Separable filters: generally more efficient to filter with two 1D filters than one 2D filter L2.1 P37
- For example, the 2D Gaussian can be expressed as a product of two 1D Gaussians (in x and y)

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

Convolution output size

Valid convolution: The output image is smaller than the input image

Border handling L2.1 P39-42

- Pad with constant value
- Wrap image
- Clamp/replicate the border value
- Reflect image

Practical considerations

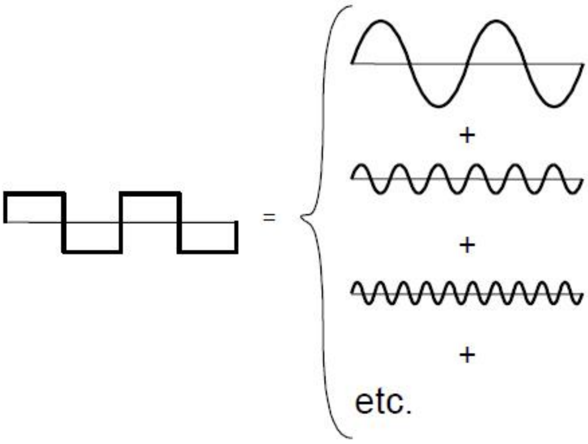
- Think about how to implement filters efficiently
  - Images are big, so efficient filtering can save a lot of time.
- Think about how to handle borders
  - No one-size-fits-all solution
  - Wrap is ideal for tilling textures (but not photos)
  - Clamp/replicate tends to work will for photos
- Linear filters: first step of almost all computer vision systems
- Linear filters are just a first step: you can’t build complex feature detectors from just linear filters.

Frequency filtering

Fourier analysis (1D)

Any signals or pattern can be described as a sum of *sinusoids* (L2.2 P7-13).

Sum of sinusoids: (L2.2 P14-17)



Fourier transform

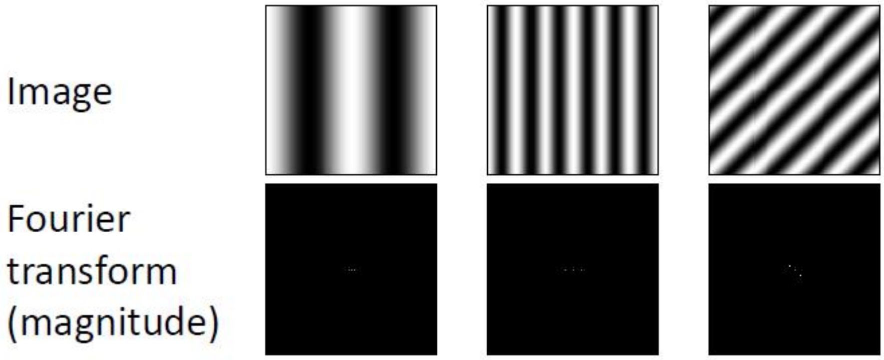
- Fourier transform decomposes signal into component frequencies
  - Values are complex numbers representing amplitude and phase of sinusoids
  - Time domain -> frequency domain (or, for images, spatial domain -> frequency domain)
- $F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-2i\pi\omega x}\, dx$
- scipy.fft (1D), scipy.fft2 (2D), scipy.fftn (3D+)
- Inverse Fourier transform converts from frequency domain back to space domain

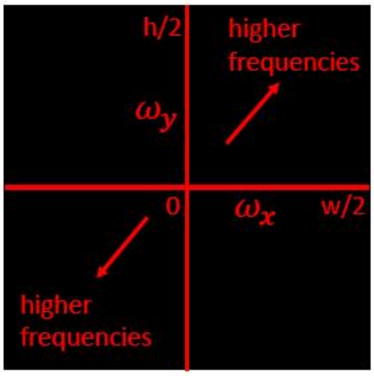
Frequency Spectrum L2.2 P19-21

- Values in frequency domain are complex numbers
- For each frequency : magnitude (= amplitude) and angle (= phase)

Fourier analysis (images) L2.2 P24-34

- Any image can be represented by its Fourier transform
- Fourier transform = for each frequency, magnitude (amplitude) + phase





Fourier transform
(magnitude)

Magnitude and phase

- Magnitude is easy to read, giving the characteristics of the texture of the image.
- Phase represents actual structures and edges.
- Magnitude captures the holistic “texture” of an image, but the edges are mainly represented by Fourier phase

Frequency filtering

- Operations in the spatial domain have equivalent operations in frequency domain
- Convolution in spatial domain = multiplication in frequency domain

$$FT[h * f] = FT[h]FT[f]$$

- Inverse:

$$FT^{-1}[hf] = FT^{-1}[h] * FT^{-1}[f]$$

Band pass filter: A filter that removes a range of frequencies from a signal

- **Low pass filter**
  - Keep low spatial frequencies, remove high frequencies L2.2 P38-40
  - Equivalent to blurring the image
- **High pass filter**
  - Keep high spatial frequencies, remove low frequencies L2.2 P41-43
- "Ringing problem": Proof by inverse convolution theorem
  - Use Gaussian low/high pass filter

Summary

- Images can be filtered in the spatial domain, or the frequency domain
- Operations in one domain have an equivalent in the other domain
  - Convolution in spatial domain = multiplication in Fourier domain
- Modelling filters in both domains can help understand/debug what a filter is doing

Applications

Image compression

- Frequency domain is a convenient space for image compression
- Human visual system is not very sensitive to contrast in high spatial frequencies
- Discarding information in high spatial frequencies doesn’t change the “look” of an image
- JPEG compression: break image into 8x8 pixel blocks, each represented in frequency space
- Discrete cosine transform (DCT)
- High spatial frequency components are quantised

Image forensic (鉴定) L2.2 P54-56

Summary

- Any image can be represented in either the spatial or the frequency domain
- Frequency domain is a convenient space for many applications:
  - Filtering
  - Compression
  - Forensics
  - Frequency-based features