Correlation, Convolution, Filtering

COMPSCI 527 — Computer Vision
Outline

1. Template Matching and Correlation
2. Image Convolution
3. Filters
4. Separable Convolution
Template Matching
Normalized Cross-Correlation

\[ \rho(r, c) = \tau^T \omega(r, c) \]

\[ \tau = \frac{t - m_t}{\| t - m_t \|} \quad \text{and} \quad \omega(r, c) = \frac{w(r, c) - m_{w(r,c)}}{\| w(r, c) - m_{w(r,c)} \|} \]

\[ -1 \leq \rho(r, c) \leq 1 \]

\[ \rho = 1 \iff W(r, c) = \alpha T + \beta , \quad \alpha > 0 \]
\[ \rho = -1 \iff W(r, c) = \alpha T + \beta , \quad \alpha < 0 \]
Results
Cross-Correlation

(ignoring normalization for simplicity)

\[ J(r, c) = t^T w(r, c) \]
for r = 1:m
    for c = 1:n
        J(r, c) = 0
        for u = -h:h
            for v = -h:h
                J(r, c) = J(r, c) + T(u, v) * I(r+u, c+v)
            end
        end
    end
end

\[ J(r, c) = \sum_{u=-h}^{h} \sum_{v=-h}^{h} I(r+u, c+v) T(u, v) \]
Convolution

Correlation:

\[ J(r, c) = \sum_{u=-h}^{h} \sum_{v=-h}^{h} I(r + u, c + v) T(u, v) \]

Convolution:

\[ J(r, c) = \sum_{u=-h}^{h} \sum_{v=-h}^{h} I(r - u, c - v) H(u, v) \]

Same as

\[ J(r, c) = \sum_{u=-h}^{h} \sum_{v=-h}^{h} I(r + u, c + v) H(-u, -v) \]

Convolution with kernel \( H(u, v) \) is correlation with template \( T(u, v) = H(-u, -v) \)
What’s the Big Deal?

Simplify \( J(r, c) = \sum_{u=-h}^{h} \sum_{v=-h}^{h} l(r-u, c-v)H(u, v) \)

to \( J(r, c) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} l(r-u, c-v)H(u, v) \)

Changes of variables \( u \leftarrow r - u \) and \( v \leftarrow c - v \)

\( J(r, c) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} H(r-u, c-v)l(u, v) \)

Convolution commutes: \( l \ast H = H \ast l \)

(Correlation does not)
Importance of Convolution in Mathematics

- Polynomials: coefficients of product are “full” convolutions of coefficients:
  
  \[ P(x) = p_0 + p_1 x + \ldots + p_m x^m \]
  
  \[ Q(x) = q_0 + q_1 x + \ldots + q_n x^n \]
  
  \[ R(x) = p_0 q_0 + (p_0 q_1 + p_1 q_0) x + \ldots + p_m q_n x^{m+n} \]

- Example:
  
  \[ P(x) = p_0 + p_1 x + p_2 x^2 + p_3 x^3 \rightarrow (p_0, p_1, p_2, p_3) \]
  
  \[ Q(x) = q_0 + q_1 x + q_2 x^2 \rightarrow (q_0, q_1, q_2) \]
  
  Convolve \( (p_0, p_1, p_2, p_3) \) with \( (q_0, q_1, q_2) \) to get \( (r_0, r_1, r_2, r_3, r_4, r_5) \)
Important Consequence

- Discrete Fourier transform is a polynomial:
  \[ p = (p_0, \ldots, p_{n-1}) \]
- \( \mathcal{F}[p](\ell) = p_0 + p_1 z + \ldots + p_{n-1} z^{n-1} \) where \( z = \frac{1}{n} e^{-i \frac{2\pi \ell}{n}} \)
- All of spectral signal theory follows
- Example: The Fourier transform of a convolution is the product of the Fourier transforms
- [We will not see this]
Image Boundaries: “Valid” Convolution

- Full overlap of image and kernel
- If $I$ is $m \times n$ and $H$ is $k \times \ell$, then $J$ is $(m - k + 1) \times (n - \ell + 1)$

Input image

$$(m, n) = (4, 6)$$

Kernel

$$(k, l) = (3, 2)$$

Output image

$$(2, 5)$$
Image Boundaries: “Full” Convolution

- Any non-empty overlap of image and kernel
- If \( I \) is \( m \times n \) and \( H \) is \( k \times \ell \), then \( J \) is \((m+k-1) \times (n+\ell-1)\)
  [Pad with either zeros or copies of boundary pixels]

![Image Convolution Diagram]

\[
\begin{align*}
(m, n) &= (4, 6) \\
(k, l) &= (3, 2)
\end{align*}
\]
Image Boundaries: “Same” Convolution

- Require the output to have the same size as the input
- If \( I \) is \( m \times n \) and \( H \) is \( k \times \ell \), then \( J \) is \( m \times n \)

\[
\begin{array}{cccccccc}
\text{input image} \\
(\text{m, n}) &=& (4, 6) \\
\hline
\hline
\text{kernel} \\
(\text{k, l}) &=& (3, 2) \\
\end{array}
\]
Filters

- What is convolution for?
  - Smoothing for noise reduction
  - Image differentiation
  - Convolutional Neural Networks (CNNs)
  - ... 

- Smoothing and differentiation are examples of filtering: Local, linear image $\rightarrow$ image transformations
Smoothing for Noise Reduction

- Assume: Image varies slowly enough to be \textit{locally affine}
- Assume: Noise is zero-mean and white
Averaging as Convolution

\[ J(c) = \frac{1}{2h+1} \sum_{v=-h}^{h} l(c - v) \] is the same as

\[ J(c) = \sum_{v=-h}^{h} l(c - v) H(v) \] where \( H(v) = \frac{1}{2h+1} [1, \ldots, 1] \), a convolution with the box kernel

Box kernel in two dimensions:
Box versus Gaussian Kernel

- The Gaussian kernel does a *weighted* average
- Emphasizes nearby values more than distant ones
- Blurs less than the box kernel for the same averaging effect
Box versus Gaussian Kernel
Truncation

\[ G(u, v) = e^{-\frac{1}{2} \frac{u^2 + v^2}{\sigma^2}} \]

- The larger \( \sigma \), the more smoothing
- \( u, v \) integer, and cannot keep them all
- Truncate at \( 3\sigma \) or so
\[ e^{-\frac{3^2}{2}} \approx 0.01 \]
Normalization

\[ G(u, v) = e^{\frac{-1}{2} \frac{u^2 + v^2}{\sigma^2}} \]

- We want \( I \ast G \approx I \)
- For \( I = c \) (constant), \( I \ast G = I \)
- Normalize by computing \( \gamma = 1 \ast G \), and then let \( G \leftarrow G/\gamma \)
Separability

- A kernel that satisfies $H(u, v) = a(u)b(v)$ is separable.
- The Gaussian is separable with $a = b$:
  $$G(u, v) = e^{-\frac{1}{2} \frac{u^2 + v^2}{\sigma^2}} = g(u) g(v) \quad \text{with} \quad g(u) = e^{-\frac{1}{2} \left( \frac{u}{\sigma} \right)^2}$$
- A separable kernel leads to efficient convolution:
  $$J(r, c) = \sum_{u=-h}^{h} \sum_{v=-k}^{k} H(u, v) l(r-u, c-v)$$
  $$= \sum_{u=-h}^{h} a(u) \sum_{v=-k}^{k} b(v) l(r-u, c-v)$$
  $$= \sum_{u=-h}^{h} a(u) \phi(r-u, c) \quad \text{where} \quad \phi(r, c) = \sum_{v=-h}^{h} b(v) l(r, c-v)$$
Computational Complexity

General: \( J(r, c) = \sum_{u=-h}^{h} \sum_{v=-k}^{k} H(u, v) I(r-u, c-v) \)

Separable: \( J(r, c) = \sum_{u=-h}^{h} a(u) \phi(r-u, c) \) where \( \phi(r, c) = \sum_{v=-h}^{h} b(v) I(r, c-v) \)

Let \( m = 2h + 1 \) and \( n = 2k + 1 \)

General: About \( 2mn \) operations per pixel
Separable: About \( 2m + 2n \) operations per pixel

Example:

When \( m = n \) (square kernel), the gain is \( 2m^2/4m = m/2 \)

With \( m = 20 \): About 80 operations per pixel instead of 800