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

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

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

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

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

\[ \rho = -1 \iff W(r, c) = \alpha T + \beta , \quad \alpha < 0 \]
A `numpy` warning: Slices are `views`, not copies
Cross-Correlation

(without normalization)

\[ j(r, c) = t^T w(r, c) \]
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:**

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

**Convolution:**

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

Same as

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

Convolution with \(H(u, v)\) is correlation with \(H(-u, -v)\)
What’s the Big Deal?

Simplify

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

to

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

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

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

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 \( z = \frac{1}{n} e^{-i2\pi \ell/n} \)
- 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]
Point-Spread Function

$T$ was a template

$H$ is called a (convolutional) kernel

A.k.a. point-spread function

If the image $I$ is a point, then $H$ spreads the point:

$$\delta(u, v) = \begin{cases} 
1 & \text{for } u = v = 0 \\
0 & \text{elsewhere}
\end{cases}$$

$$J(r, c) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} H(r - u, c - v) \delta(u, v) = H(r, c)$$
More Generally:

\[ \delta_{a,b}(u, v) = \begin{cases} 
1 & \text{for } u = a \text{ and } v = b \\
0 & \text{elsewhere} 
\end{cases} \]

\[ J(r, c) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} H(r - u, c - v) \delta_{a,b}(u, v) = H(r - a, c - b) \]

(No flip in the output!)
Image Boundaries: “Valid” Convolution

If $I$ is $m \times n$ and $H$ is $k \times \ell$, then $J$ is $(m - k + 1) \times (n - \ell + 1)$
Image Boundaries: “Full” Convolution

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 Boundaries: “Same” Convolution

If \( I \) is \( m \times n \) and \( H \) is \( k \times \ell \), then \( J \) is \( m \times n \)
Filters

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

• We’ll see the first two next, CNNs later

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

- Assume: Image varies slowly enough to be *locally linear*
- Assume: Noise is zero-mean and white
2 Dimensions: The Pillbox Kernel
Issues with the Pillbox
The Gaussian Kernel
Gaussian versus Pillbox
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
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 $H(u, v) = h(u)\ell(v)$ is separable
- The Gaussian is separable with $h = \ell$:

  $$G(u, v) = e^{-\frac{1}{2} \frac{u^2 + v^2}{\sigma^2}} = g(u)g(v) \text{ with } g(u) = e^{-\frac{1}{2} \left( \frac{u}{\sigma} \right)^2}$$

- A separable kernel leads to efficient convolution:

  $$J(r, c) = \sum_{u=-h}^{h} \sum_{v=-k}^{k} H(u, v) I(r - u, c - v)$$
  
  $$= \sum_{u=-h}^{h} h(u) \sum_{v=-k}^{k} \ell(v) I(r - u, c - v)$$
  
  $$= \sum_{u=-h}^{h} h(u) \phi(r - u, c) \text{ where } \phi(r, c) = \sum_{v=-h}^{h} \ell(v) I(r, c - v)$$
Separable Convolution

Computational Complexity

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

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

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