Templates, Image Pyramids, and Filter Banks

Computer Vision
Derek Hoiem, University of Illinois
Review: filtering in spatial domain

\[
\begin{pmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{pmatrix}
\]

\[
\text{intensity image} \ast \text{kernel} = \text{resulting image}
\]
Review: filtering in frequency domain
Log Magnitude

Strong Vertical Frequency (Sharp Horizontal Edge)

Diagonal Frequencies

Strong Horz. Frequency (Sharp Vert. Edge)

Low Frequencies

Log Magnitude
Today’s class

• Template matching

• Image Pyramids

• Filter banks and texture

• Denoising, Compression
Template matching

• Goal: find 🔄 in image

• Main challenge: What is a good similarity or distance measure between two patches?
  – Correlation
  – Zero-mean correlation
  – Sum Square Difference
  – Normalized Cross Correlation
Matching with filters

- Goal: find 🕒 in image
- Method 0: filter the image with eye patch

\[ h[m,n] = \sum_{k,l} g[k,l] \cdot f[m+k,n+l] \]

What went wrong?
Matching with filters

• Goal: find 🕳️ in image

• Method 1: filter the image with zero-mean eye

\[
h[m, n] = \sum_{k, l} (g[k, l] - \bar{g}) (f[m + k, n + l])
\]

mean of template \(g\)
Matching with filters

- **Goal:** find an object in image
- **Method 2:** SSD

\[
    h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2
\]

![Input](image1)
![1- sqrt(SSD)](image2)
![Thresholded Image](image3)

True detections
Matching with filters

Can SSD be implemented with linear filters?

\[ h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2 \]
Matching with filters

- Goal: find 🕳️ in image
- Method 2: SSD

\[ h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2 \]
Matching with filters

• Goal: find eye in image

• Method 3: Normalized cross-correlation

\[ h[m,n] = \frac{\sum_{k,l} (g[k,l] - \bar{g})(f[m+k,n+l] - \bar{f}_{m,n})}{\left(\sum_{k,l} (g[k,l] - \bar{g})^2 \sum_{k,l} (f[m+k,n+l] - \bar{f}_{m,n})^2\right)^{0.5}} \]

Matlab: `normxcorr2(template, im)`
Matching with filters

• Goal: find 🕳️ in image
• Method 3: Normalized cross-correlation
Matching with filters

- **Goal:** find 🧐 in image
- **Method 3:** Normalized cross-correlation
Q: What is the best method to use?

A: Depends

- Zero-mean filter: fastest but not a great matcher
- SSD: next fastest, sensitive to overall intensity
- Normalized cross-correlation: slowest, invariant to local average intensity and contrast
Q: What if we want to find larger or smaller eyes?

A: Image Pyramid
Review of Sampling

Image → Gaussian Filter → Low-Pass Filtered Image → Sample → Low-Res Image
Gaussian pyramid

Source: Forsyth
Template Matching with Image Pyramids

Input: Image, Template
1. Match template at current scale
2. Downsampling image
   - In practice, scale step of 1.1 to 1.2
3. Repeat 1-2 until image is very small
4. Take responses above some threshold, perhaps with non-maxima suppression
Laplacian filter

unit impulse

Gaussian

≈

Laplacian of Gaussian

Source: Lazebnik
Laplacian pyramid

Source: Forsyth
Creating the Gaussian/Laplacian Pyramid

Image = $G_1$

- Smooth, then downsample

Downsample $(\text{Smooth}(G_1))$  

$G_2$

Downsample $(\text{Smooth}(G_2))$  

$G_3$

...  

$G_N = L_N$

$G_1 - \text{Smooth(Upsample}(G_2))$

$L_1$

$G_2 - \text{Smooth(Upsample}(G_3))$

$L_2$

$G_3 - \text{Smooth(Upsample}(G_4))$

$L_3$

- Use same filter for smoothing in each step (e.g., Gaussian with $\sigma = 2$)
- Downsampling/upsampling with “nearest” interpolation
Hybrid Image in Laplacian Pyramid

High frequency → Low frequency
Reconstructing image from Laplacian pyramid

Image = \( L_1 + \text{Smooth}(\text{Upsample}(G_2)) \)

\[ G_2 = L_2 + \text{Smooth}(\text{Upsample}(G_3)) \]

\[ G_3 = L_3 + \text{Smooth}(\text{Upsample}(L_4)) \]

- Use same filter for smoothing as in deconstruction
- Upsample with “nearest” interpolation
- Reconstruction will be lossless
Major uses of image pyramids

• Compression

• Object detection
  – Scale search
  – Features

• Detecting stable interest points

• Registration
  – Course-to-fine
Coarse-to-fine Image Registration

1. Compute Gaussian pyramid
2. Align with coarse pyramid
3. Successively align with finer pyramids
   - Search smaller range

Why is this faster?

Are we guaranteed to get the same result?
Image representation

- Pixels: great for spatial resolution, poor access to frequency

- Fourier transform: great for frequency, not for spatial info

- Pyramids/filter banks: balance between spatial and frequency information
Application: Representing Texture

Source: Forsyth
Texture and Material

http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/
Texture and Orientation

http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/
Texture and Scale

http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/
What is texture?

Regular or stochastic patterns caused by bumps, grooves, and/or markings
How can we represent texture?

• Compute responses of blobs and edges at various orientations and scales
Overcomplete representation: filter banks

LM Filter Bank

Code for filter banks: www.robots.ox.ac.uk/~vgg/research/texclass/filters.html
Filter banks

- Process image with each filter and keep responses (or squared/abs responses)
How can we represent texture?

• Measure responses of blobs and edges at various orientations and scales

• Idea 1: Record simple statistics (e.g., mean, std.) of absolute filter responses
Can you match the texture to the response?

Filters

Mean abs responses

A

B

C
Representing texture by mean abs response

Filters

Mean abs responses
Representing texture

- Idea 2: take vectors of filter responses at each pixel and cluster them, then take histograms (more on this in coming weeks)
How is it that a 4MP image (12000KB) can be compressed to 400KB without a noticeable change?
Lossy Image Compression (JPEG)

Block-based Discrete Cosine Transform (DCT)

Slides: Efros
Using DCT in JPEG

- The first coefficient $B(0,0)$ is the DC component, the average intensity
- The top-left coeffs represent low frequencies, the bottom right – high frequencies
Image compression using DCT

• Quantize
  – More coarsely for high frequencies (which also tend to have smaller values)
  – Many quantized high frequency values will be zero

• Encode
  – Can decode with inverse dct

Filter responses

\[
G = \begin{bmatrix}
-415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\
-46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\
-48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\
12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\
-7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\
-1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\
-0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68
\end{bmatrix}
\]

\[ u \]

\[ v \]

Quantization table

\[
Q = \begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99
\end{bmatrix}
\]

Quantized values

\[
B = \begin{bmatrix}
-26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\
0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]
JPEG Compression Summary

1. Convert image to YCrCb
2. Subsample color by factor of 2
   - People have bad resolution for color
3. Split into blocks (8x8, typically), subtract 128
4. For each block
   a. Compute DCT coefficients
   b. Coarsely quantize
      • Many high frequency components will become zero
   c. Encode (e.g., with Huffman coding)

http://en.wikipedia.org/wiki/YCbCr
http://en.wikipedia.org/wiki/JPEG
Lossless compression (PNG)

1. Predict that a pixel’s value based on its upper-left neighborhood
2. Store difference of predicted and actual value
3. Pkzip it (DEFLATE algorithm)
Denoising

Additive Gaussian Noise

Gaussian Filter
Reducing Gaussian noise

Smoothing with larger standard deviations suppresses noise, but also blurs the image.

Source: S. Lazebnik
Reducing salt-and-pepper noise by Gaussian smoothing

3x3  5x5  7x7
Alternative idea: Median filtering

• A **median filter** operates over a window by selecting the median intensity in the window.

• Is median filtering linear?

Source: K. Grauman
Median filter

• What advantage does median filtering have over Gaussian filtering?
  – Robustness to outliers

Source: K. Grauman
Median filter

- MATLAB: `medfilt2(image, [h w])`

Source: M. Hebert
Median vs. Gaussian filtering

Gaussian

Median

3x3  5x5  7x7
Other non-linear filters

- Weighted median (pixels further from center count less)

- Clipped mean (average, ignoring few brightest and darkest pixels)

- Bilateral filtering (weight by spatial distance and intensity difference)
Bilateral filters

• Edge preserving: weights similar pixels more

Original
Gaussian
Bilateral

\[ I^b_p = \frac{1}{W^b_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) \ G_{\sigma_r}(\|I_p - I_q\|) \ I_q \]

with \[ W^b_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) \ G_{\sigma_r}(\|I_p - I_q\|) \]

Summary

• Applications of filters
  – Template matching (SSD or Normxcorr2)
    • SSD can be done with linear filters, is sensitive to overall intensity
  – Gaussian pyramid
    • Coarse-to-fine search, multi-scale detection
  – Laplacian pyramid
    • More compact image representation
    • Can be used for compositing in graphics
  – Compression
    • In JPEG, coarsely quantize high frequencies
  – Filter banks for representing texture
  – Denoising
Next class: edge detection