CS 1674: Intro to Computer Vision
Image Filtering and Texture

Prof. Seong Jae Hwang
University of Pittsburgh
January 28, 2021
Plan for next three lectures

• Filters: motivation, math and properties
• Types of filters
  – Linear
    • Smoothing
    • Other
  – Non-linear
    • Median
• Applications of filters
  – Texture representation with filters
  – Anti-aliasing for image subsampling
How images are represented (Matlab)

- Color images represented as a matrix with multiple channels (1 if grayscale)
- Suppose we have a NxM RGB image called “im”
  - $im(1,1,1) =$ top-left pixel value in R-channel
  - $im(y, x, b) =$ y pixels down (rows), x pixels to right (cols) in b\(^{th}\) channel
  - $im(N, M, 3) =$ bottom-right pixel in B-channel
- `imread(filename)` returns a uint8 image (values 0 to 255)

Adapted from Derek Hoiem
What a noisy world

- The same object will look very different across images
- Even multiple images of same static scene won’t be identical
- How could we reduce the noise, i.e., give an estimate of the true intensities?
- What if there’s only one image?
Common types of noise

- **Impulse noise**: random occurrences of white pixels
- **Salt and pepper noise**: random occurrences of black and white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Source: S. Seitz
Gaussian noise

>> noise = randn(size(im)) .* sigma;
>> output = im + noise;

What is impact of the sigma?
First attempt at a solution

- Let’s replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:

Source: S. Marschner
First attempt at a solution

• Let’s replace each pixel with an average of all the values in its neighborhood

• **Why/when** will this work?

• Assumptions:
  – Expect pixels to be *like their neighbors*
  – Expect noise processes to be *independent* from pixel to pixel
  – Expect noise process to follow an *identical* pattern
Weighted Moving Average

- Can add weights to our moving average
- \textit{Weights} \ [1, 1, 1, 1, 1] / 5
- What may be good/bad weights for removing noise?
Weighted Moving Average

• Non-uniform weights $[1, 4, 6, 4, 1] / 16$
  – Q: why divide by 16?

Adapted from S. Marschner
Moving Average In 2D

\[ F[x, y] \]

\[ G[x, y] \]

Source: S. Seitz
Moving Average In 2D

\[ F[x, y] \]

\[ G[x, y] \]

Source: S. Seitz
Moving Average In 2D

\[ F[x, y] \]

\[ G[x, y] \]
Moving Average In 2D

\[ F[x, y] \quad G[x, y] \]
Moving Average In 2D

\[ F[x, y] \]

\[ G[x, y] \]

Source: S. Seitz
Moving Average In 2D

\[ F[x, y] \quad G[x, y] \]

In what ways is this output good/expected?  
In what ways is it bad (what was lost)?

Source: S. Seitz
Image filtering

- Compute a function of the local neighborhood at each pixel in the image
  - Function specified by a “filter” or mask saying how to combine values from neighbors
  - Element-wise multiplication of filter and image patch

- Uses of filtering:
  - Enhance an image (denoise, smoothing, resize, etc.)
  - Extract information (texture, edges, etc.)
  - Detect patterns (template matching)
Quick realization: smoothing is everywhere!

- **1D**: clean mic sounds by smoothing the sound signal
- **2D**: clean image by smoothing the 2D intensity signal
- **3D**: stabilize the video by smoothing the camera trajectory over time
Correlation filtering

Non-weighted, **averaging** window size is $2k+1 \times 2k+1$:

$$G[i, j] = \frac{1}{(2k + 1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i + u, j + v]$$

- Attribute uniform weight to each pixel
- Loop over all pixels in neighborhood around image pixel $F[i,j]$

Now generalize to allow **different weights** depending on neighboring pixel’s relative position:

$$G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v]$$

- Non-uniform weights
Correlation filtering

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v] \]

This is called **cross-correlation**, denoted \( G = H \otimes F \).

Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter “**kernel**” or “**mask**” \( H[u,v] \) is the prescription for the weights in the linear combination.
Averaging filter

• What values belong in the kernel $H$ for the moving average example?

\[ G = H \otimes F \]
Smoothing by averaging

What if the filter size was 5 x 5 instead of 3 x 3?
Gaussian filter

• What if we want nearest neighboring pixels to have the most influence on the output?

This kernel is an approximation of a 2d Gaussian function:

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

• Removes high-frequency components from the image (“low-pass filter”).
Smoothing with a Gaussian

vs. box filter
Gaussian filters

• What parameters matter here?
• **Size** of kernel or mask
  – Note, Gaussian function has infinite support (i.e., “domain”), but discrete filters use finite kernels

\[
\sigma = 5 \text{ with } 10 \times 10 \text{ kernel} \quad \sigma = 5 \text{ with } 30 \times 30 \text{ kernel}
\]
Gaussian filters

• What parameters matter here?

• **Standard deviation** $\sigma$ of Gaussian: “width” of the bump which determines extent of smoothing

\[ \sigma = 2 \] with 30 x 30 kernel

\[ \sigma = 5 \] with 30 x 30 kernel
Gaussian filters

How big should the filter be?

- Values at edges should be near zero
- Common convention: set filter width $hsize = 2 \times \text{ceil}(3 \times \sigma ) + 1$
Gaussian filter in Matlab

```matlab
>> hsize = 31;
>> sigma = 5;
>> h = fspecial('gaussian', hsize, sigma);

>> mesh(h);

>> imagesc(h);

>> outim = imfilter(im, h); % correlation
>> imshow(outim);
```
Smoothing with a Gaussian

Parameter \( \sigma \) is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.

\[
\begin{align*}
\text{for } \sigma = 1:3:10 \\
h &= \text{fspecial('gaussian', hsize, sigma)}; \\
\text{out} &= \text{imfilter(im, h)}; \\
\text{imshow(out);} \\
\text{pause}; \\
\text{end}
\end{align*}
\]
Convolution

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

\[
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v]
\]

\[ G = H \ast F \]

*Notation for convolution operator*
Convolution vs. correlation

Convolution

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v]F[i - u, j - v] \]

\[ G = H \ast F \]

Cross-correlation

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v]F[i + u, j + v] \]

\[ G = H \otimes F \]

For a Gaussian or box filter, how will the outputs differ?
Not so important now but theoretically and practically useful fact

Convolution is
- Commutative
\[ f \ast g = g \ast f \]
- Associative
\[ f \ast (g \ast h) = (f \ast g) \ast h \]
- Efficient to compute in the frequency space

Cross-correlation is
- Not commutative
- Not associative
Convolution vs. correlation

**Cross-correlation**

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v] \]

\[ G = H \otimes F \]

**Convolution**

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v] \]

\[ G = H \ast F \]
Convolution vs. correlation

**Cross-correlation**

\[
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v]
\]

\[G = H \otimes F\]

**Convolution**

\[
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v]
\]

\[G = H \ast F\]
Convolution vs. correlation

Cross-correlation

\[
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v]
\]

\[
G = H \otimes F
\]

Convolution

\[
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v]
\]

\[
G = H \ast F
\]
Convolution vs. correlation

Cross-correlation

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v]F[i + u, j + v] \]

\[ G = H \otimes F \]

Convolution

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v]F[i - u, j - v] \]

\[ G = H \ast F \]
Convolution vs. correlation

Cross-correlation

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v] \]

\[ G = H \otimes F \]

Convolution

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v] \]

\[ G = H \ast F \]
Convolution vs. correlation

Cross-correlation

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v] \]

\[ G = H \otimes F \]

Convolution

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v] \]

\[ G = H \ast F \]
Convolution vs. correlation

Cross-correlation

\[
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v]F[i + u, j + v]
\]

\[
G = H \otimes F
\]

Convolution

\[
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v]F[i - u, j - v]
\]

\[
G = H \ast F
\]
Convolution vs. correlation

Cross-correlation

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v] \]

\[ G = H \otimes F \]

Convolution

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v] \]

\[ G = H \ast F \]
Properties of smoothing filters

- **Smoothing**
  - Values positive
  - Sum to 1 → overall intensity same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter
Predict the outputs using correlation filtering

\[
\begin{array}{ccc}
* & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
= ?
\]

\[
\begin{array}{ccc}
* & 0 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
\end{array}
= ?
\]

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{array}
* \frac{1}{9}
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
= ?
\]
Practice with linear filters

Original

0 0 0
0 1 0
0 0 0

Source: D. Lowe
Practice with linear filters

Original

Filtered (no change)

Source: D. Lowe
Practice with linear filters

Original

0 0 0
0 0 1
0 0 0

Source: D. Lowe
Practice with linear filters

Original

Shifted left by 1 pixel with correlation

Source: D. Lowe
Practice with linear filters

Original

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\frac{1}{9}
\]

Source: D. Lowe
Practice with linear filters

Original

Blur (with a box filter)

Source: D. Lowe
Practice with linear filters

Original

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{array}
\quad - \quad \frac{1}{9}
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

Source: D. Lowe
Image minus blur = details
Practice with linear filters

Sharpening filter:
Amplify strong intensity while suppressing nearby intensities.

Compared to its neighbor, if the center pixel (of the kernel) has
1. High intensity: make it more apparent
2. Low intensity: make it less apparent (nearby strong intensity pixel will reduce the filter output)

Source: D. Lowe
Sharpening

before

after
Filters for computing gradients

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

* intensity image

=

Adapted from Derek Hoiem
Filters for computing gradients

Input image
White = 1, black = 0

Filter: [+1, 0, -1]
Note: Convolution!

Output image
Median filter

- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
- Non-linear filter
Median filter

- Median filter is edge preserving
Median filter

Salt and pepper noise

Median filtered

Plots of a row of the image

Matlab: output_im = medfilt2(im, [h w]);

Source: M. Hebert
Boundary issues

- What is the size of the output?
  - ‘full’: output size is larger than the size of \( f \)
  - ‘same’: output size is same as \( f \)

\[ f = \text{image} \]
\[ h = \text{filter} \]

Adapted from S. Lazebnik
Boundary issues

• What about near the edge?
  – the filter window might fall off the edge of the image (in ‘same’ or ‘full’)
  – need to extrapolate
  – methods:
    • clip filter (black)
    • wrap around
    • copy edge
    • reflect across edge

Source: S. Marschner
Properties of convolution

• Commutative:
  \[ f * g = g * f \]

• Associative:
  \[ (f * g) * h = f * (g * h) \]

• Distributes over addition:
  \[ f * (g + h) = (f * g) + (f * h) \]

• Scalars factor out:
  \[ kf * g = f * kg = k(f * g) \]

• Identity:
  unit impulse \( e = [..., 0, 0, 1, 0, 0, ...] \). \( f * e = f \)
Separability

• In some cases, filter is separable, and we can factor into two steps:
  – Convolve all rows
  – Convolve all columns
Separability example

2D filtering (center location only)

The filter factors into an outer product of 1D filters:

Perform filtering along rows:

Followed by filtering along the remaining column:

Asymptotic cost for 2D vs 1D filtering? Let image be Pxn, filter be Nxn.
Application: Hybrid Images

What you see...
I see an angry guy

From Far Away

Up Close
It's a woman!
Application: Hybrid Images


Gaussian Filter

Laplacian Filter (sharpening)

unit impulse

Gaussian

≈

Laplacian of Gaussian
Application: Hybrid Images
Application: Hybrid Images

Changing expression

Sad ↔ Surprised
Plan for next two lectures

• Filters: math and properties
• Types of filters
  – Linear
    • Smoothing
    • Other
  – Non-linear
    • Median
• Applications
  – Texture representation with filters
  – Anti-aliasing for image subsampling
Texture

Due to:
Patterns, marks, etches, blobs, holes, relief, etc.
Regular (top), random (bottom) patterns
Why analyze texture?

• Important for how we perceive objects
• Can be an important appearance cue that allows us to distinguish objects, especially if shape is similar across objects
Same shape, different texture/object
Same object, different texture
Texture representation

• Textures are made up of repeated local patterns:
  – To find the patterns
    • Use filters that look like patterns (spots, bars, raw patches...)
    • Consider magnitude of response to the filters
  – Describe their statistics within each local window
    • Statistics: ways to “summarize” the information
    • e.g. mean, standard deviation, histogram
Derivative of Gaussian filter

Figures from Noah Snavely
Texture representation: example

Original image

Derivative filter responses, squared

Statistics to summarize patterns in small windows:

<table>
<thead>
<tr>
<th>Win. #1</th>
<th>mean (d/dx) value</th>
<th>mean (d/dy) value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

Kristen Grauman
Texture representation: example

original image

derivative filter responses, squared

statistics to summarize patterns in small windows

<table>
<thead>
<tr>
<th>Window</th>
<th>mean d/dx value</th>
<th>mean d/dy value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Win. #2</td>
<td>18</td>
<td>7</td>
</tr>
</tbody>
</table>

Kristen Grauman
## Texture representation: example

### Original Image

![Original Image](image)

### Derivative Filter Responses, Squared

![Derivative Filter Responses, Squared](image)

### Statistics to Summarize Patterns in Small Windows

<table>
<thead>
<tr>
<th>Window #</th>
<th>$\text{mean } \frac{d}{dx}$ Value</th>
<th>$\text{mean } \frac{d}{dy}$ Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Win. #2</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>Win. #9</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

- Kristen Grauman
Texture representation: example

- **Dimension 1 (mean d/dx value)**
- **Dimension 2 (mean d/dy value)**

<table>
<thead>
<tr>
<th>Window</th>
<th>Mean d/dx Value</th>
<th>Mean d/dy Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Win. #2</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>Win. #9</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

**statistics to summarize patterns in small windows**

Kristen Grauman

School of Computing and Information

University of Pittsburgh
Texture representation: example

Windows with primarily horizontal edges

Windows with small gradient in both directions

Windows with primarily vertical edges

Both

Dimension 1 (mean d/dx value)

Dimension 2 (mean d/dy value)

<table>
<thead>
<tr>
<th>Windows</th>
<th>mean d/dx value</th>
<th>mean d/dy value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Win. #2</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>Win. #9</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

Statistics to summarize patterns in small windows
Texture representation: example

original image

derivative filter responses, squared

visualization of the assignment to texture “types”
Texture representation: example

- Dimension 1 (mean d/dx value)
- Dimension 2 (mean d/dy value)

Far: dissimilar textures
Close: similar textures

<table>
<thead>
<tr>
<th></th>
<th>mean d/dx value</th>
<th>mean d/dy value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Win. #2</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>Win. #9</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

Statistics to summarize patterns in small windows

Kristen Grauman
Computing distances using texture

\[ D(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} \]

\[ = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \]

Euclidean distance (L2)
Distance reveals how dissimilar texture from window a is from texture in window b.
Filter banks

• Our previous example used two filters, and resulted in a 2-dimensional feature vector to describe texture in a window.
  – x and y derivatives revealed something about local structure.

• We can generalize to apply a collection of multiple \( (d) \) filters: a “filter bank”.

• Then our feature vectors will be \( d \)-dimensional.
Filter banks

- What filters to put in the bank?
  - Typically we want a combination of scales and orientations, different types of patterns.

Which filters would you use to distinguish buildings from animals? Cheetahs from tigers? Ladybugs from dalmatians?

Adapted from Kristen Grauman, Matlab code: http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html
Filter bank
Kristen Grauman
Vectors of texture responses

To represent pixel, form a “feature vector” from the responses at that pixel.

1x38 vector representation feature

[r1, r2, …, r38]

Adapted from Kristen Grauman
Vectors of texture responses

To represent pixel, form a “feature vector” from the responses at that pixel.

To represent image, compute statistics over all pixel feature vectors, e.g. their mean.

$$\begin{bmatrix}
  r_{(1,1)}^1, & r_{(1,1)}^2, & \ldots, & r_{(1,1)}^{38} \\
  r_{(1,2)}^1, & r_{(1,2)}^2, & \ldots, & r_{(1,2)}^{38} \\
  \vdots & \vdots & \ddots & \vdots \\
  r_{(W,H)}^1, & r_{(W,H)}^2, & \ldots, & r_{(W,H)}^{38} \\
\end{bmatrix}$$

$$\begin{bmatrix}
\text{mean}(r_{(:)}^1), & \text{mean}(r_{(:)}^2), & \ldots, & \text{mean}(r_{(:)}^{38})
\end{bmatrix}$$
You try: Can you match the texture to the response?

Filters

1

2

3

Mean abs responses

A

B

C

Derek Hoiem
Representing texture by mean absolute response

Filters

Mean abs responses
Classifying materials, “stuff”

Figure by Varma & Zisserman
Plan for next two lectures

• Filters: math and properties
• Types of filters
  – Linear
    • Smoothing
    • Other
  – Non-linear
    • Median
• Applications
  – Texture representation with filters
  – Anti-aliasing for image subsampling
Why does a lower resolution image still make sense to us? What do we lose?
Subsampling by a factor of 2

Throw away every other row and column to create a 1/2 size image
Aliasing problem

• 1D example (sinewave):
Aliasing problem

- 1D example (sinewave):
Aliasing problem

• Sub-sampling may be dangerous....

• Characteristic errors may appear:
  – “Wagon wheels rolling the wrong way in movies”
  – “Striped shirts look funny on color television”

Adapted from D. Forsyth, image from https://en.wikipedia.org/wiki/Aliasing
Sampling and aliasing
Nyquist-Shannon Sampling Theorem

- When sampling a signal at discrete intervals, the **sampling frequency** must be $\geq 2 \times f_{\text{max}}$
- $f_{\text{max}} = \text{max frequency of the input signal}$
- This will allow to reconstruct the original perfectly from the sampled version
Anti-aliasing

Solutions:

• Sample more often

• Get rid of high frequencies
  – What are these in the case of images?
  – Will lose information, but it’s better than aliasing
  – Apply a smoothing filter
Algorithm for downsampling by factor of 2

1. Start with image(h, w)
2. Apply low-pass filter
   \[ \text{im\_blur} = \text{imfilter(image, fspecial('gaussian', 7, 1))} \]
3. Sample every other pixel
   \[ \text{im\_small} = \text{im\_blur(1:2:end, 1:2:end);} \]
Anti-aliasing

256x256  128x128  64x64  32x32  16x16

256x256  128x128  64x64  32x32  16x16
Subsampling without pre-filtering

1/2

1/4 (2x zoom)

1/8 (4x zoom)
Subsampling with Gaussian pre-filtering

Gaussian 1/2  G 1/4  G 1/8
Subsampling away...

{\(f_0, f_1, ..., f_n\)}
= Gaussian pyramid

{\(h_0, h_1, ..., h_n\)}
= Laplacian pyramid

\[ f_0 - l_0 = h_0 \]

\[ f_1 - l_1 = h_1 \]

\[ h_2 = f_2 \]

Why would we want to do this?
Can we reconstruct the original from the Laplacian pyramid?
Gaussian pyramid
Laplacian pyramid

Source: Forsyth
Summary

• Filters useful for
  – Enhancing images (smoothing, removing noise), e.g.
    • Box filter (linear)
    • Gaussian filter (linear)
    • Median filter
  – Detecting patterns (e.g. gradients)

• Texture is a useful property that is often indicative of materials, appearance cues
  – Texture representations summarize repeating patterns of local structure

• Can use filtering to reduce the effects of subsampling