CITS 4402 Computer Vision

Prof Ajmal Mian
Adj/A/Prof Mehdi Ravanbakhsh, CEO at Mapizy (www.mapizy.com) and InFarm (www.infarm.io)

Lecture 04 – Greyscale Image Analysis
Lecture 03 – Summary

- Images as 2-D signals
- Linear and non-linear filters
- Fourier series
- Fourier Transform
- Discrete Fourier Transform
Image Enhancement

**Aim:**
- Make images easier to interpret for the human eye
- Generate better input for other image processing techniques

There are two main categories of techniques
- **Spatial domain methods** which operate directly on pixels
- **Frequency domain methods** which operate on the Fourier transform of the image
Single Pixel Manipulation

- The value of $g(x, y)$ depends directly on the value of $f(x, y)$
- This is a **greyscale transformation** or simply **mapping**
- Common mapping functions:

![Identity transformation](image1)

![Thresholding](image2)
Common Mapping Functions

- **Lightening**
  - Increase contrast

- **Increase contrast**
Common Mapping Functions

- Enhance contrast in dark regions
- Enhance contrast in light regions
input image

(Hawkes Bay, NZ)

histogram of intensity

output image

```
input intensity
```

```
output intensity
```
Sample MATLAB Code

The Matlab code for enhancing the Hawkes Bay image on the previous slide is extremely simple!

```matlab
im = imread('Unequalized_Hawkes_Bay_NZ.jpg');
Imshow(im);

% after observing the intensity histogram of the input image...
f = 0:255; % input intensity
g(1:100) = linspace(0,20,100);
g(101:200) = linspace(21,240,100);
g(201:256) = linspace(241,255,56);

figure, plot(f,g,'b-');

% generate output image
newim = g(im+1);
newim = uint8(newim);  % convert to uint8

figure, imshow(newim);
```
Histogram Equalization

- stretch out the histogram to produce a more uniform distribution

Image data squashed into a small range of grey values

Ideally we want the image data to spread uniformly over all grey values
Histogram Equalization

For digital images, we have a discrete formulation.
Let
\[ n_k \quad \text{number of pixels with grey level } k \]
\[ N \quad \text{total number of pixels} \]
Then the probability of obtaining grey level \( k \) in input image \( f \) is:
\[ P_f(f_k) = \frac{n_k}{N} \]
The transformation is:
\[ g_k = T(f_k) = \sum_{i=0}^{k} \frac{n_i}{N} \]

Note: the values of \( g_k \) will have to be scaled up by 255 and rounded to the nearest integer. This discretization of \( g_k \) means that the transformed image will not have a perfectly uniform distribution.
Indeed, the total number of distinct grey levels is reduced!
Histogram Equalization (cont.)

An example: suppose that we have a 64×64 image with 8 grey levels:

<table>
<thead>
<tr>
<th>$f_i$</th>
<th>$n_i$</th>
<th>$n_i/N$</th>
<th>$g_i$</th>
<th>(ideally)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>790</td>
<td>0.19</td>
<td>0.19</td>
<td>0/7=0</td>
</tr>
<tr>
<td>1/7</td>
<td>1023</td>
<td>0.25</td>
<td>0.44</td>
<td>1/7=0.14</td>
</tr>
<tr>
<td>2/7</td>
<td>850</td>
<td>0.21</td>
<td>0.65</td>
<td>2/7=0.28</td>
</tr>
<tr>
<td>3/7</td>
<td>656</td>
<td>0.16</td>
<td>0.81</td>
<td>3/7=0.42</td>
</tr>
<tr>
<td>4/7</td>
<td>329</td>
<td>0.08</td>
<td>0.89</td>
<td>4/7=0.57</td>
</tr>
<tr>
<td>5/7</td>
<td>245</td>
<td>0.06</td>
<td>0.95</td>
<td>5/7=0.71</td>
</tr>
<tr>
<td>6/7</td>
<td>122</td>
<td>0.03</td>
<td>0.98</td>
<td>6/7=0.85</td>
</tr>
<tr>
<td>1</td>
<td>81</td>
<td>0.02</td>
<td>1.00</td>
<td>7/7=1</td>
</tr>
</tbody>
</table>

$N=4096$

We want the cumulative distribution of the output image to look as much like a 45° line as possible.
Histogram Equalization (cont.)

<table>
<thead>
<tr>
<th>$f_i$</th>
<th>$n_i$</th>
<th>$n_i/N$</th>
<th>$g_i$</th>
<th>(ideally)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>790</td>
<td>0.19</td>
<td>0.19</td>
<td>0/7=0</td>
</tr>
<tr>
<td>1/7</td>
<td>1023</td>
<td>0.25</td>
<td>0.44</td>
<td>1/7=0.14</td>
</tr>
<tr>
<td>2/7</td>
<td>850</td>
<td>0.21</td>
<td>0.65</td>
<td>2/7=0.28</td>
</tr>
<tr>
<td>3/7</td>
<td>656</td>
<td>0.16</td>
<td>0.81</td>
<td>3/7=0.42</td>
</tr>
<tr>
<td>4/7</td>
<td>329</td>
<td>0.08</td>
<td>0.89</td>
<td>4/7=0.57</td>
</tr>
<tr>
<td>5/7</td>
<td>245</td>
<td>0.06</td>
<td>0.95</td>
<td>5/7=0.71</td>
</tr>
<tr>
<td>6/7</td>
<td>122</td>
<td>0.03</td>
<td>0.98</td>
<td>6/7=0.85</td>
</tr>
<tr>
<td>1</td>
<td>81</td>
<td>0.02</td>
<td>1.00</td>
<td>7/7=1</td>
</tr>
</tbody>
</table>

Thus intensity mapping is:

InputIntensity $\rightarrow$ OutputIntensity

0 $\rightarrow$ 1/7
1/7 $\rightarrow$ 3/7
2/7 $\rightarrow$ 5/7
3/7 $\rightarrow$ 6/7
4/7 $\rightarrow$ 6/7
5/7 $\rightarrow$ 1
6/7 $\rightarrow$ 1
1 $\rightarrow$ 1

Input image has 8 grey levels
Output image has only 5 grey levels
Histogram Equalization (cont.)

<table>
<thead>
<tr>
<th>$f_i$</th>
<th>$n_i$</th>
<th>$n_i/N$</th>
<th>$g_i$</th>
<th>$n_i$ of $g_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>790</td>
<td>0.19</td>
<td>0.19</td>
<td>0</td>
</tr>
<tr>
<td>1/7</td>
<td>1023</td>
<td>0.25</td>
<td>0.44</td>
<td>790</td>
</tr>
<tr>
<td>2/7</td>
<td>850</td>
<td>0.21</td>
<td>0.65</td>
<td>0</td>
</tr>
<tr>
<td>3/7</td>
<td>656</td>
<td>0.16</td>
<td>0.81</td>
<td>1023</td>
</tr>
<tr>
<td>4/7</td>
<td>329</td>
<td>0.08</td>
<td>0.89</td>
<td>0</td>
</tr>
<tr>
<td>5/7</td>
<td>245</td>
<td>0.06</td>
<td>0.95</td>
<td>850</td>
</tr>
<tr>
<td>6/7</td>
<td>122</td>
<td>0.03</td>
<td>0.98</td>
<td>985</td>
</tr>
<tr>
<td>1</td>
<td>81</td>
<td>0.02</td>
<td>1.00</td>
<td>448</td>
</tr>
</tbody>
</table>

Thus intensity mapping is:

InputIntensity $\rightarrow$ OutputIntensity

$0 \rightarrow 1/7$
$1/7 \rightarrow 3/7$
$2/7 \rightarrow 5/7$
$3/7 \rightarrow 6/7$
$4/7 \rightarrow 6/7$
$5/7 \rightarrow 1$
$6/7 \rightarrow 1$
$1 \rightarrow 1$

Input image has 8 grey levels

Output image has only 5 grey levels
Histogram Equalization (cont.)

<table>
<thead>
<tr>
<th>$f_i$</th>
<th>$n_i$</th>
<th>$n_i/N$</th>
<th>$g_i$</th>
<th>$n_i$ of $g_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>790</td>
<td>0.19</td>
<td>0.19</td>
<td>0</td>
</tr>
<tr>
<td>1/7</td>
<td>1023</td>
<td>0.25</td>
<td>0.44</td>
<td>790</td>
</tr>
<tr>
<td>2/7</td>
<td>850</td>
<td>0.21</td>
<td>0.65</td>
<td>0</td>
</tr>
<tr>
<td>3/7</td>
<td>656</td>
<td>0.16</td>
<td>0.81</td>
<td>1023</td>
</tr>
<tr>
<td>4/7</td>
<td>329</td>
<td>0.08</td>
<td>0.89</td>
<td>0</td>
</tr>
<tr>
<td>5/7</td>
<td>245</td>
<td>0.06</td>
<td>0.95</td>
<td>850</td>
</tr>
<tr>
<td>6/7</td>
<td>122</td>
<td>0.03</td>
<td>0.98</td>
<td>985</td>
</tr>
<tr>
<td>1</td>
<td>81</td>
<td>0.02</td>
<td>1.00</td>
<td>448</td>
</tr>
</tbody>
</table>

Intensity histogram of output image

- Grey value: 0, 1, 2, 3, 4, 5, 6, 7
- Frequency: 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
A very noisy image
Using Neighborhood Pixels

- So far we modified values of single pixels
- What if we want to take neighborhood into account?

- A common application of linear filtering is **image smoothing** using an **averaging filter** or averaging **mask** or averaging **kernel**
- Each point in the smoothed image \( g(x, y) \) is obtained from the average pixel value in a neighbourhood of \((x, y)\) in the input image

  - The averaging filter is also known as the **box filter**. Each pixel under the mask is multiplied by 1/9, summed, and the result is placed in the output image
  - The mask is successively moved across the image. That is, we **convolve** the image with the mask.
Example of 2-D Convolution

Convolution with box filters of size 1, 5, 15, 3, 9, 35 (reading order)
Image filtering $g[\ldots]$.

$h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l]$
\[ f[\ldots] \]
\[ g[\ldots] = \frac{1}{9} \]
\[ h[\ldots] \]

\[
h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l]
\]
\[ h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l] \]
$f[\ldots]$ \hspace{1cm} $g[\ldots]$ \hspace{1cm} $h[\ldots]$ \hspace{1cm} $\frac{1}{9}$

\[
h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l]
\]
\[ h[m,n] = \sum_{k,l} g[k,l] f[m+k, n+l] \]
\[ h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l] \]
\[ f[\ldots] \quad \frac{1}{9} \quad g[\ldots] \quad h[\ldots] \]

\[
h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l]
\]
\[
f[\ldots]\]
\[
g[\ldots]\]
\[
h[\ldots]\]
\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 \\
\end{array}
\]
\[
\begin{array}{cccccccc}
0 & 0 & 10 & 20 & 30 & 30 & 30 & 20 \\
0 & 0 & 20 & 40 & 60 & 60 & 60 & 40 \\
0 & 0 & 30 & 60 & 90 & 90 & 90 & 60 \\
0 & 30 & 60 & 90 & 90 & 90 & 90 & 60 \\
0 & 0 & 30 & 50 & 80 & 80 & 90 & 60 \\
0 & 0 & 30 & 50 & 80 & 80 & 90 & 60 \\
0 & 0 & 30 & 50 & 80 & 80 & 90 & 60 \\
0 & 0 & 30 & 50 & 80 & 80 & 90 & 60 \\
\end{array}
\]
\[
\begin{array}{cccccccc}
10 & 20 & 30 & 30 & 30 & 30 & 20 & 10 \\
10 & 10 & 10 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
\[
h[m,n] = \sum_{k,l} g[k,l] f[m+k, n+l]
\]
Practice with Linear Filters

Original

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Practice with Linear Filters

Original

Filtered
(no change)
Practice with Linear Filters

Original

0 0 0
0 0 1
0 0 0

?
Practice with Linear Filters

Original

Shaded left
By 1 pixel
Practice with Linear Filters

Original

(Note that filter sums to 1)
Practice with Linear Filters

Original

Sharpening filter
- Accentuates differences with local average
Sharpening

before

after
Other Averaging Filters

- One expects the value of a pixel to be more closely related to the values of pixels close to it than to those further away.
- Accordingly it is usual to weight the pixels near the centre of the mask more strongly than those at the edge.

\[ \frac{1}{3} \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{array} \]
Other Averaging Filters

- One expects the value of a pixel to be more closely related to the values of pixels close to it than to those further away
- Accordingly it is usual to weight the pixels near the centre of the mask more strongly than those at the edge
Gaussian Filter

Slide credit: Christopher Rasmussen

Use Matlab’s `fspecial` function to create a Gaussian filter.

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

$\sigma^2$ is also known as the width of the kernel.

$$\begin{array}{cccccc}
0.003 & 0.013 & 0.022 & 0.013 & 0.003 \\
0.013 & 0.059 & 0.097 & 0.059 & 0.013 \\
0.022 & 0.097 & 0.159 & 0.097 & 0.022 \\
0.013 & 0.059 & 0.097 & 0.059 & 0.013 \\
0.003 & 0.013 & 0.022 & 0.013 & 0.003 \\
\end{array}$$

$5 \times 5, \sigma = 1$
Example – Box Filter Smoothing
Example – Gaussian Smoothing
Key Properties of Linear Filters

- **Linearity:**
  \[ \text{filter}(f_1 + f_2) = \text{filter}(f_1) + \text{filter}(f_2) \]

- **Shift invariance:** same behavior regardless of pixel location
  \[ \text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f)) \]

Any linear, shift-invariant operator can be represented as a convolution.
Key Properties of Linear Filters

- **Commutative:** \( a \ast b = b \ast a \)
  - Conceptually no difference between filter and signal

- **Associative:** \( a \ast (b \ast c) = (a \ast b) \ast c \)
  - Often apply several filters one after another: \(((a \ast b_1) \ast b_2) \ast b_3\)
  - This is equivalent to applying one filter: \(a \ast (b_1 \ast b_2 \ast b_3)\)

- **Distributes over addition:** \( a \ast (b + c) = (a \ast b) + (a \ast c) \)

- **Scalars factor out:** \( ka \ast b = a \ast kb = k (a \ast b) \)

- **Identity:** unit impulse \( e = [0, 0, 1, 0, 0] \), \( a \ast e = a \)
Gaussian Filters

- Linear filters
- Remove “high-frequency” components from the image (low-pass filter)
  - Images become more smooth
- Convolution of a Gaussian with a Gaussian is another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving two times with Gaussian kernel of width $\sigma$ is same as convolving once with kernel of width $\sigma \sqrt{2}$
- *Separable* kernels
  - Factors into product of two 1D Gaussians
Gaussian Filters

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

\[ = \left(\frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{x^2}{2\sigma^2}\right)\right) \left(\frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{y^2}{2\sigma^2}\right)\right) \]

The 2D Gaussian can be expressed as the product of two functions, one a function of \(x\) and the other a function of \(y\).

In this case, the two functions are the (identical) 1D Gaussian.
Practical Matters

- What about near the edge?
- The filter window falls off the edge of the image
- Need to extrapolate
- Methods:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge
Practical Matters

Methods (MATLAB):
• clip filter (black): \( \text{imfilter}(f, g, 0) \)
• wrap around: \( \text{imfilter}(f, g, \text{‘circular’}) \)
• copy edge: \( \text{imfilter}(f, g, \text{‘replicate’}) \)
• reflect across edge: \( \text{imfilter}(f, g, \text{‘symmetric’}) \)
Practical Matters

MATLAB: `filter2(g, f, shape)`

- `shape = 'full'`: output size is sum of sizes of `f` and `g`
- `shape = 'same'`: output size is same as `f`
- `shape = 'valid'`: output size is difference of sizes of `f` and `g`
Low-Pass Filtering

Removing all high spatial frequencies from a signal to retain only low spatial frequencies is called low-pass filtering.

Old Spectrum  New Spectrum  Low-Pass Filtered Image
High-Pass Filtering

Removing all *low* spatial frequencies from a signal to retain only *high* spatial frequencies is called **high-pass filtering**.
Low-Pass Filtering – An Example

![Input Image]

![Low pass filter 1, cutoff=0.05, order=2]

![Low pass filter 2, cutoff=0.8, order=2]
Low-Pass Filtering – An Example

Input image

Low pass filter 1
\( \text{cutoff}=0.05, \text{order}=2 \)

Low pass filter 2
\( \text{cutoff}=0.8, \text{order}=2 \)

Filtered output image using filter 1
(more smoothing/blurring)

Fourier transform of input image

Filtered output image using filter 2
High-Pass Filtering – An Example
High-Pass Filtering – An Example

![Input Image](image1)

![High pass filter 1](image2)

![High pass filter 2](image3)

![Fourier transform of input image](image4)

![Filtered output image using filter 1](image5)

![Filtered output image using filter 2 (more sharpening)](image6)
Filtering in the Spatial Domain

- Low-pass filtering -> convolve the image with a box / Gaussian filter
- High-pass filtering -> ?
Filtering in the Spatial Domain

- Low-pass filtering -> convolve the image with a box / Gaussian filter
- High-pass filtering ->

<table>
<thead>
<tr>
<th>-1/9</th>
<th>-1/9</th>
<th>-1/9</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1/9</td>
<td>8/9</td>
<td>-1/9</td>
</tr>
<tr>
<td>-1/9</td>
<td>-1/9</td>
<td>-1/9</td>
</tr>
</tbody>
</table>

- Since the sum of the weights is 0, the resulting signal will have a **0 DC value** (i.e., the average value or the coefficient of the zero frequency component).
- To display the image you will need to take the absolute values.
Non-Linear Filtering

- Neighbourhood averaging or Gaussian smoothing will tend to blur edges because the high frequency components are attenuated.
- An alternative approach is to use **median filtering**. Here we set the grey level to be the median of the pixel values in the neighbourhood.
- Example: pixel values in $3 \times 3$ neighbourhood

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>

Sort the values 10 15 20 20 20 20 20 25 100

- Pixels with outlying values are forced to become more like their neighbours.
Median Filtering – An Example

- Median filter removes outliers
- Median filter smooths the image without blurring the edges
High-Boost Filtering

- Here we take the original image and boost the high frequency components.
- Can think of $\text{HighPass} = \text{Original} – \text{LowPass}$. Thus
  $$\text{HighBoost} = b \times \text{Original} – \text{LowPass}$$
  $$= (b-1) \times \text{Original} + \text{Original} – \text{LowPass}$$
  $$= (b-1) \times \text{Original} + \text{HighPass}$$
  $b$ is the boosting factor.
  When $b=1$, $\text{HighBoost} = \text{HighPass}$

- High-boost filtering – useful for emphasizing high frequencies while retaining some low frequency components of the original image to aid in the interpretation of the image
High Boost Filtering (cont.)

How can we perform high-boost filtering in the spatial domain?

$$w = 9b - 1$$

where $w$ is the weight and $b$ is the number of pixels that will be boosted.

<table>
<thead>
<tr>
<th>-1/9</th>
<th>-1/9</th>
<th>-1/9</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1/9</td>
<td>$w/9$</td>
<td>-1/9</td>
</tr>
<tr>
<td>-1/9</td>
<td>-1/9</td>
<td>-1/9</td>
</tr>
</tbody>
</table>

Original | High Boosted (intermediate result) | High Boosted (contrast stretching)
Homomorphic Filtering

- The brightness of an image point \( f(x, y) \) is a function of the illumination at that point and the reflectance of the object at that point, i.e.,

\[
f(x, y) = i(x, y) \cdot r(x, y)
\]

- It is the reflectance that tells us information about the scene. We want to reduce the influence of illumination.

- **Assumptions:**
  1. Illumination variations vary with low spatial frequency
  2. Features of interest are the result of different reflectance properties of objects across the image and these vary with high(er) spatial frequency
Homomorphic Filtering (cont.)

- Let \( z(x, y) = \log(f(x, y)) \)
  \[ = \log(i(x, y)) + \log(r(x, y)) \]

- In the frequency domain, we have
  \[ Z(\omega, \nu) = I(\omega, \nu) + R(\omega, \nu) \]

  - Fourier transform of \( \log(i(x, y)) \)
  - Fourier transform of \( \log(r(x, y)) \)

- \( Z(\omega, \nu) \) represents the Fourier Transform of the sum of two images:
  - a low frequency illumination image
  - a high frequency reflectance image

If we apply a high boost filter, then we can suppress the low frequency illumination components and enhance the reflectance components.
Homomorphic Filtering (cont.)

- Apply the high-boost filter $H(\omega, \nu)$:
  
  $$S(\omega, \nu) = Z(\omega, \nu) \cdot H(\omega, \nu)$$

- Take the inverse FFT:
  
  $$s(x, y) = F^{-1}(S(\omega, \nu))$$

- Finally, exponentiate the result to account for taking the log of the original image:
  
  $$g(x, y) = \exp(s(x, y))$$
Homomorphic Filtering – An Example

Original image

Filtered image
Smoothing and Sub-sampling

In many Computer Vision applications, sub-sampling is often needed, e.g.,

• to build an image pyramid, or
• simply to reduce the resolution for efficient storage, transmission, processing, …
Throw away every other row and column to create a 1/2 size image
Sub-sampling Issues

Sub-sampling may be dangerous…. Why?

Resample the checkerboard by taking one sample at each circle. In the case of the top left board, new representation is reasonable. Top right also yields a reasonable representation. Bottom left is all black (dubious) and bottom right has checks that are too big.
Aliasing in Videos

Imagine a spoked wheel moving to the right (rotating clockwise). Mark wheel with dot so we can see what’s happening.

If camera shutter is only open for a fraction of a frame time (frame time = 1/30 sec. for video, 1/24 sec. for film):

Without dot, wheel appears to be rotating slowly backwards! (counterclockwise)
Aliasing in Graphics

Disintegrating textures
Sampling and Aliasing

Top row shows the images, sampled at every second pixel to get the next; bottom row shows the magnitude of frequency spectrum of these images.
Anti-aliasing

1. Sample more often

2. Get rid of all frequencies that are greater than half the new sampling frequency (Nyquist frequency)

- Will lose information
- But it’s better than aliasing
- Apply a smoothing filter
Sampling and Aliasing

Sampling with smoothing. Top row shows the images. We get the next image by smoothing the image with a Gaussian with sigma 1 pixel, then sampling at every second pixel to get the next; bottom row shows the magnitude spectrum of these images.
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
Image Pyramids

- Key component of many high level computer vision tasks
- How to create an image pyramid?
  - Represent each image as a layer
  1. Convolve layer $i$ with a Gaussian filter
  2. Subsample layer $i$ by a factor of two (remove all even-numbered rows and columns) to get layer $i + 1$
- Repeat steps 1 and 2 until a stopping criteria is satisfied
Image Pyramids
Template Matching

- Finding object of known shape and appearance in an image
- To identify the object, we have to compare the template image against the source image by sliding it.
- At each location, we find the matching score between the image and the template.
Template Matching Scores

- Sum of Squared Difference (SSD) in pixel values
- (Normalized) Correlation coefficient
- (Normalized) Cross-correlation

Source Image

Template
Template Matching Scores

SSD

Normalized Correlation Coefficient

Normalized Cross-Correlation
Summary

- Single Pixel Operations
- Histogram Equalization
- Filtering in the Spatial Domain
- Subsampling and Anti-aliasing
- Template Matching

Acknowledgements: The slides are based on previous lectures by A/Prof Du Huynh and Prof Peter Koveski. Other material has been taken from Wikipedia, computer vision textbook by Forsyth & Ponce, and OpenCV documentation.