Matlab Tutorial
Derivatives, Filtering, Pyramids

Gonzalo Vaca-Castano
UCF
2013
COLOR SPACES: RGB, HSV, ETC
Image Architecture

- Raster Images (RGB, Gray Scale, Logical)

(8-bit representation of colors)

Row 3
Row 2
Row 1

3 depths

Single depth

Gray Scale (0 – 255)

(1-bit representation of black or white saturation)

Logical (0 or 1)
RGB space

• $\gg I = \text{imread('board.tif')}$

• Displaying image:
  – $\gg \text{imshow}(I)$

• Check image size
  – $\gg \text{size}(I)$

• Convert color image to black and white image:
  – $\gg \text{rgb2gray}(I)$
  – % Gray = 0.2989 * R + 0.5870 * G + 0.1140 * B
Other Color spaces

- rgb2hsv(I)
- rgb2ycbcr(I)
- rgb2ntsc(I)

CIELAB or CIECAM02

http://en.wikipedia.org/wiki/HSL_and_HSV
Histogram Equalization

```matlab
%Histogram Equalization
I = imread('pout.tif');
J = histeq(I);
figure(1);
subplot(1,2,1);
imshow(I);
subplot(1,2,2);
imshow(J);
figure(2);
subplot(1,2,1);
imhist(I,64);
subplot(1,2,2);
imhist(J,64);
```
DERIVATIVES
Derivatives (Filters)

• In a continuous 1d Signal, derivative is:
  – \( \lim_{dx \to 0} \frac{f(x+dx)-f(x)}{dx} \)

• In a discrete 1d signal, derivative is:
  – \( f(x+1)-f(x) \)
  – It is a convolution with the filter:

  \[ \begin{array}{c}
  -1 \\
  1 \\
  \end{array} \]

  – Other popular filter is:

  Sobel Filter

  \[ \begin{array}{c}
  -1 \\
  0 \\
  1 \\
  \end{array} \]
### Derivatives in 2D

**Table 4.1** Derivative operators: their formal (continuous) definitions and corresponding discrete approximations

<table>
<thead>
<tr>
<th>2D derivative measure</th>
<th>Continuous case</th>
<th>Discrete case</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{\partial f}{\partial x} )</td>
<td>( \lim_{\Delta x \to 0} \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x} )</td>
<td>( f(x + 1, y) - f(x, y) )</td>
</tr>
<tr>
<td>( \frac{\partial f}{\partial y} )</td>
<td>( \lim_{\Delta y \to 0} \frac{f(x, y + \Delta y) - f(x, y)}{\Delta y} )</td>
<td>( f(x, y + 1) - f(x, y) )</td>
</tr>
<tr>
<td>( \nabla f(x, y) )</td>
<td>( \begin{bmatrix} \frac{\partial f}{\partial x} &amp; \frac{\partial f}{\partial y} \end{bmatrix} )</td>
<td>( [f(x + 1, y) - f(x, y), f(x, y + 1) - f(x, y)] )</td>
</tr>
<tr>
<td>( \frac{\partial^2 f}{\partial x^2} )</td>
<td>( \lim_{\Delta x \to 0} \frac{(\partial f/\partial x)(x + \Delta x, y) - (\partial f/\partial x)f(x, y)}{\Delta x} )</td>
<td>( f(x + 1, y) - 2f(x, y) + f(x - 1, y) )</td>
</tr>
<tr>
<td>( \frac{\partial^2 f}{\partial y^2} )</td>
<td>( \lim_{\Delta y \to 0} \frac{(\partial f/\partial y)(x, y + \Delta y) - (\partial f/\partial y)(x, y)}{\Delta y} )</td>
<td>( f(x, y + 1) - 2f(x, y) + f(x, y - 1) )</td>
</tr>
<tr>
<td>( \nabla^2 f(x, y) )</td>
<td>( \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} )</td>
<td>( f(x + 1, y) + f(x - 1, y) - 4f(x, y) + f(x, y + 1) + f(x, y - 1) )</td>
</tr>
</tbody>
</table>

**KERNEL**

\[ \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \]

\[ \begin{bmatrix} 1 & -2 & 1 \\ 1 & -2 & 1 \end{bmatrix} \]

\[ \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \]
Convolution

Figure 2.12  Discrete convolution. The centre pixel of the kernel and the target pixel in the image are indicated by the dark grey shading. The kernel is ‘placed’ on the image so that the centre and target pixels match. The filtered value of the target pixel is then given by a linear combination of the neighbourhood pixels, the specific weights being determined by the kernel values. In this specific case the target pixel of original value 35 has a filtered value of 14.
Filtering in 2D Arrays (Convolution)

\[ f_i = \sum_{i-j}^9 w_k I_k (i) \]
\[ = (-1 \times 10) + (-1 \times 11) + (-1 \times 8) + (-1 \times 40) + (8 \times 35) \\
+ (-1 \times 42) + (-1 \times 38) + (-1 \times 36) + (-1 \times 46) = 14 \]
Alternative derivatives in 2D

**Figure 4.9** First-order edge-detection filters
Derivatives in Matlab

```matlab
I = imread('trees.tif');
imshow(I)
k1=[ 1 0 -1;2 0 -2; 1 0 -1]
o=imfilter(double(I),k1,'same');
figure; imagesc(o)
colormap gray
```

QUESTION: Why `imagesc` instead of `imshow`?
Gradient Magnitude and direction

% - Display derivative.
horgradI2=abs(conv2(double(I2),hcent,'same'));
figure;imshow(uint8(horgradI2));
vergradI2=abs(conv2(double(I2),hcent,'same'));
figure;imshow(uint8(vergradI2));

% - Compute gradient magnitude and direction.
gradMag=sqrt(vergradI2.^2+horgradI2.^2);
figure;imshow(uint8(gradMag));
gradDir=atan2(vergradI2,horgradI2);
FILTERING
Mean filter

% - Box kernel for image smoothing
I1 = imread('groceries.jpg');
I2 = rgb2gray(I1);
h=ones(7)/49;
I6=uint8(conv2(I2,h,'same'));
figure;imshow(I6);
Special filters

% - fspecial. Create arbitrary sized Gaussian kernel
% with different sizes and variances and their effects on smoothing (over/under smooth)
hg = fspecial('gaussian', [7 7], 1);

% - Use Gaussian kernel for smoothing
I3=conv2(double(I2),hg,'same');
imshow(uint8(I3));
Gaussian Derivative

% - Gaussian derivative. Wrote a one line code/formula for creating a Gaussian derivative kernel, for gradient computation.
% - I emphasized Gaussian derivative and that, in general, this should be used for gradient computation of images.

```
gaussder = conv2(fspecial('gaussian',[7 7],4),[1 0 -1],'valid');
figure;surf(gaussder);
I4 = conv2(double(I2),gaussder);
imshow(I4);
```
Separable Filters

```matlab
% - Separable kernels
% - Computing derivatives
close all;
keyboard;
M=magic(7);
f1=[1;-1] % vertical gradient
f2=[-1 1] % horizontal gradient
a=conv2(double(M),f1,'same')
b=conv2(double(a),f2,'same')
f3=f1*f2
c=conv2(double(M),f3,'same')
sum(sum(abs(b-c)))
```

\[
\text{Ans} = 0
\]
Example: Removing noise

% Create a noisy image
I = imread('eight.tif');
imshow(I)
J = imnoise(I,'salt & pepper',0.02);
figure, imshow(J)

% Mean filter
K = filter2(fspecial('average',3),J)/255;
figure, imshow(K)

% Median filter
L = medfilt2(J,[3 3]);
figure, imshow(L)
Example: Removing noise

% Create a noisy image
I = imread('eight.tif');
imshow(I)
J = imnoise(I,'salt & pepper',0.02);
figure, imshow(J)

% Mean filter
K = filter2(fspecial('average',3),J)/255;
figure, imshow(K)

% Median filter
L = medfilt2(J,[3 3]);
figure, imshow(L)
Example: Removing noise

% Create a noisy image
I = imread('eight.tif');
imshow(I)
J = imnoise(I,'salt & pepper',0.02);
figure, imshow(J)

% Mean filter
K = filter2(fspecial('average',3),J)/255;
figure, imshow(K)

% Median filter
L = medfilt2(J,[3 3]);
figure, imshow(L)
Smoothing

\[ L(x, y, \sigma) = G(x, y, \sigma) \ast I(x, y) \]

\[ G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \]

L is a blurred image
- G is the Gaussian Blur operator
- I is an image
- x,y are the location coordinates
- \( \sigma \) is the “scale” parameter. Think of it as the amount of blur. Greater the value, greater the blur.
- The * is the convolution operation in x and y. It “applies” gaussian blur G onto the image I
Smoothing function

```matlab
% gauss_filter: Obtains a smoothed gaussian filter image
%   - Output:
%       smooth: image filter by a gaussian filter
%   - Input:
%       image: Matrix containing single band image.
%       sigma: Value of the sigma for the Gaussian filter
%       kernel_size: Size of the gaussian kernel (default: 6*sigma)

function [smooth]= gauss_filter(image,sigma,kernel_size)
    if nargin < 2
        sigma=1;
    end
    if nargin < 3
        kernel_size=6*sigma;
    end
    gaussian_radius=floor(kernel_size/2); %radius of the gaussian
    x=[-gaussian_radius : gaussian_radius]; % x values (gaussian kernel size)
    gaussianno= exp(-x.^2/(2*sigma^2))/(sigma*sqrt(2*pi)); %calculate the unidimensional gaussian
    temp=conv2(double(image),double(gaussianno),'same');
    smooth=conv2(double(temp),double(gaussianno),'same');
end
```
Example smoothing

```matlab
I = imread('eight.tif'); imagesc(I); colormap gray;
antialiassigma=2;
Ismooth=gauss_filter(I,antialiassigma,11);
figure; imagesc(Ismooth); colormap gray;
```
Resize

% Resizing an image
% Y = imresize(X, rows, cols, method);  % method - nearest bilinear, bicubic

[X, map] = imread ('groceries.jpg');
Y = imresize(X, [100, 200], 'bicubic');  % method - nearest bilinear, bicubic
imshow(Y);
Pyramid

- **Definition:**
  - is a type of multi-scale signal representation in which a signal or an image is subject to repeated smoothing and subsampling
Pyramid

I0 = imread('cameraman.tif');
I1 = impyramid(I0, 'reduce');
I2 = impyramid(I1, 'reduce');
I3 = impyramid(I2, 'reduce');
imshow(I0)
figure, imshow(I1)
figure, imshow(I2)
figure, imshow(I3)