Image Processing II - Detectors

INTELLIGENT SYSTEMS FOR PATTERN RECOGNITION (ISPR)

DAVIDE BACCIU – DIPARTIMENTO DI INFORMATICA - UNIVERSITÀ DI PISA

DAVIDE.BACCIU@UNIPI.IT
Visual Feature Detector

Repeatability

Detect the same feature in different image portions and in different images

- Photometric - Changes in brightness and luminance
- Translation - Changes in pixel location
- Rotation - Changes to absolute or relative angle of keypoint
- Scaling - Image resizing or changes in camera zoom
- Affine Transformations - Non-isotrophic changes
Edge Detection
Edges and Gradients

○ Image gradient (graylevel)

\[ \nabla I = \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix} \]

direction of change of intensity

○ Edges are pixel regions where...
  ● Intensity gradient changes abruptly

○ The return of finite difference methods

\[
G_x = \frac{\partial I}{\partial x} \approx I(x + 1, y) - I(x - 1, y)
\]

\[
G_y = \frac{\partial I}{\partial y} \approx I(x, y + 1) - I(x, y - 1)
\]

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

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

Prewitt operators
Convolving Gradient Operators
Sobel Operator

An additional level of smoothing of the central difference

\[
G_x = \begin{bmatrix}
+1 & 0 & -1 \\
+2 & 0 & -2 \\
+1 & 0 & -1
\end{bmatrix}
\]

\[
G_y = \begin{bmatrix}
+1 & +2 & +1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix}
\]
In Code

Matlab

% Create an horizontal (x) Prewitt filter
h = fspecial('prewitt'); % Try also 'sobel'
% Convolve it to the image Ig
imH = imfilter(Ig, h, 'replicate');
% Transpose filter for the y-derivative
imV = imfilter(Ig, h', 'replicate');
% Magnitude
M = uint8(sqrt(double((imHor.^2) + (imVer.^2))));
% Then plot . . .
imtool(imH);
% etc . . .

Python

# prewitt masks
kernelx = np.array([[[1, 1, 1], [0, 0, 0], [-1, -1, -1]]])
kernely = np.array([[[[-1, -1, -1], [0, 0, 0], [1, 1, 1]]]])

# convolving filters
img_prewittx = cv2.filter2D(img_gray, -1, kernelx)
img_prewitty = cv2.filter2D(img_gray, -1, kernely)

# sobel (CV_8U is the output data type, ksize is the kernel size)
img_sobelx = cv2.Sobel(img_gray, cv2.CV_8U, 1, 0, ksize = 3)
img_sobely = cv2.Sobel(img_gray, cv2.CV_8U, 0, 1, ksize = 3)
Blob Detection

- Blobs are connected pixels regions with little gradient variability
- Laplacian of Gaussian (LoG) $g_{\sigma}(x, y)$ has maximum response when centered on a circle of radius $\sqrt{2}\sigma$

$$\nabla^2 g_{\sigma}(x, y) = \frac{\partial^2 g_{\sigma}}{\partial x^2} + \frac{\partial^2 g_{\sigma}}{\partial y^2}$$

Typically using a scale normalized response

$$\nabla^2_{\text{norm}} g_{\sigma}(x, y) = \sigma^2 \left( \frac{\partial^2 g_{\sigma}}{\partial x^2} + \frac{\partial^2 g_{\sigma}}{\partial y^2} \right)$$
LoG Blob Detection

1. Convolve image with a LoG filter at different scales
   - $\sigma = k\sigma_0$ by varying $k$

2. Find maxima of squared LoG response
   1. Find maxima on space-scale
   2. Find maxima between scale
   3. Threshold

The LoG filter can be approximated as a Difference of Gaussians (DoG) for efficiency

$$g_{k\sigma_0}(x, y) - g_{\sigma_0}(x, y) \approx (k - 1)\sigma_0^2 \nabla^2 g_{(k-1)\sigma_0}$$
Affine Detectors

- Laplacian-based detectors are invariant to scale thanks to the maximization in scale-space

- Still not invariant to affine transformations
Maximally Stable Extremal Regions (MSER)

- Extract covariant regions (blobs) that are stable connected components of intensity sets of the image
- Key idea is to take blobs (Extremal Regions) which are nearly the same through a wide range of intensity thresholds
- The blobs are generated (locally) by binarizing the image over a large number of thresholds
  - Invariance to affine transformation of image intensities
  - Stability (they are stable on multiple thresholds)
  - Multi-scale (connected components are identified by intensity stability not by scale)
  - Sensitive to local lighting effects, shadows, etc..
- You can then fit an ellipse enclosing the stable region
Intuition on the MSER Algorithm

Generate frames from the image by thresholding it on all graylevels

- Capture those regions that from a small seed of pixel grow to a stably connected region
- Stability is assessed by looking at derivatives of region masks in time (most stable ⇒ minima of connected region variation)
MSER in Code

Matlab

% Run MSER and returns regions
regions = detectMSERFeatures ( Ig ) ;
figure ; imshow ( Ig ) ; % plot image
hold on ;
plot ( regions ) ; % overlap regions
% Alternatively can plot actual regions
plot ( regions , ‘showPixelList ’ , true , ‘showEllipses ’ , false ) ;

Again, in OpenCV

import cv2

# Load the mser detector from OpenCV
mser = cv2 . MSER_create ( )
regions = mser . detectRegions ( img , None )
# Create a convex hull enclosing stable regions
hulls = [ cv2 . convexHull ( p . reshape (−1, 1, 2 ) ) for p in regions ]
# Draw detected regions on image copy
vis = img . copy ( )
cv2 . polylines ( vis , hulls , 1 , ( 0 , 255 , 0 ) )
cv2 . imshow ( ‘img ’ , vis )
Image Segmentation

The process of partitioning an image into set of homogeneous pixels, hoping to match object or their subparts

- A naive approach?
  - Apply k-means to pixels color (typically $L \ast a \ast b$) hoping to cluster together regions

- A slightly less naive approach?
  - Apply k-means to pixels color and $(x, y)$ position hoping to enforce some level of spatial information in clusters
Normalized Cuts (Ncut)

○ Node = pixel

○ $a_{ij}$ = affinity between pixels (at a certain scale $\sigma$)

○ A cut of $G$ is the set of edges such whose removal makes $G$ a disconnected graph

○ Breaking graph into pieces by cutting edges of low affinity

○ Normalized cut problem
  ● NP-hard
  ● Approximate solution as an eigenvalue problem

Code: [https://www.cis.upenn.edu/~jshi/software/](https://www.cis.upenn.edu/~jshi/software/)
Pixel Issue

Pixels in image are **a lot**!
- Ncut can take ages to complete
- Likewise many other advanced segmentation algorithms

- Efficiency trick ⇒ **Superpixels**
  - Group together similar pixels
  - Cheap, local oversegmentation
  - Important that superpixels do not cross boundaries

- Now apply **segmentation/fusion** algorithms to superpixels: Ncut, Markov Random Fields, etc.

Code: [https://ivrl.epfl.ch/research/superpixels](https://ivrl.epfl.ch/research/superpixels)
Take Home Messages

○ Image processing is very much about **convolutions**
  ● Linear masks to perform **gradient operations**
  ● Gaussian functions to apply **scale changes** (zooming in and out)

○ Computational **efficiency** is often a driving factor
  ● Convolutions in **Fourier domain**
  ● Superpixels
  ● Lightweight feature detector? Random sampling