EDGE DETECTION

...

So, what is an ‘edge’ in an image?

An edge is basically a continuous set of points in an image at each of which the image brightness changes very sharply.

So, how to detect edges in an image?

There is no single definition for the term ‘boundary’, which is why there are many edge detectors such as :

- Canny.
- Deriche.
- Differential.
- Sobel.
- Prewitt.
- Roberts cross.

(the most common of these being Canny)

Most of these methods can be grouped into 2 categories, search-based and zero-crossing based.

The search-based method involves finding the edge strength of each pixel such as the magnitude of the gradient of pixels, after which we isolate the local maxima of the edge strength.

(To put it simply)The zero-crossing method involves finding the zero crossings of the second differentials obtained from the edges, usually the laplacian(all explained later :P)

An **image gradient** is a directional change in the intensity or color in an **image**.

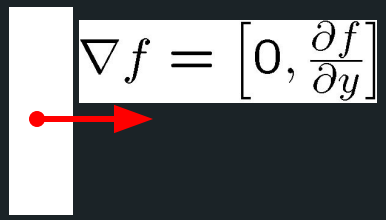
To put it mathematically :

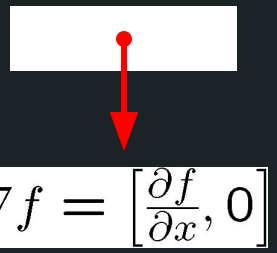
Image gradient

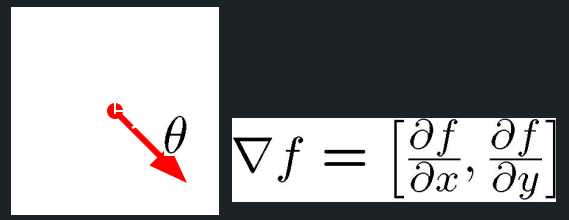
- The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- The gradient points in the direction of most rapid change in intensity







The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

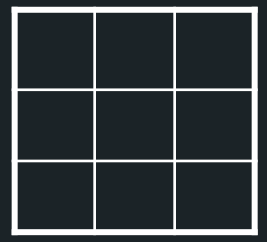
So, we find the pixels with a comparatively large edge strength (threshold defined by us), we find the edges of the image.

The discrete gradient

- How can we differentiate a *digital* image F(x,y)?
  - Option 1: reconstruct a continuous image, then take gradient
  - Option 2: take discrete derivative (finite difference)

$$\frac{\partial F}{\partial x}[x, y] \approx F[x+1, y] - F[x, y]$$

How would you implement this as a cross-correlation?



H

The Sobel operator

- Better approximations of the derivatives exist
  - The *Sobel* operators below are very commonly used

$\frac{1}{8}$

-1	0	1
-2	0	2
-1	0	1

H_x

$\frac{1}{8}$

1	2	1
0	0	0
-1	-2	-1

H_y

- The standard defn. of the Sobel operator omits the 1/8 term
  - doesn't make a difference for edge detection
  - the 1/8 term **is** needed to get the right gradient value, however

Zero-crossing method :

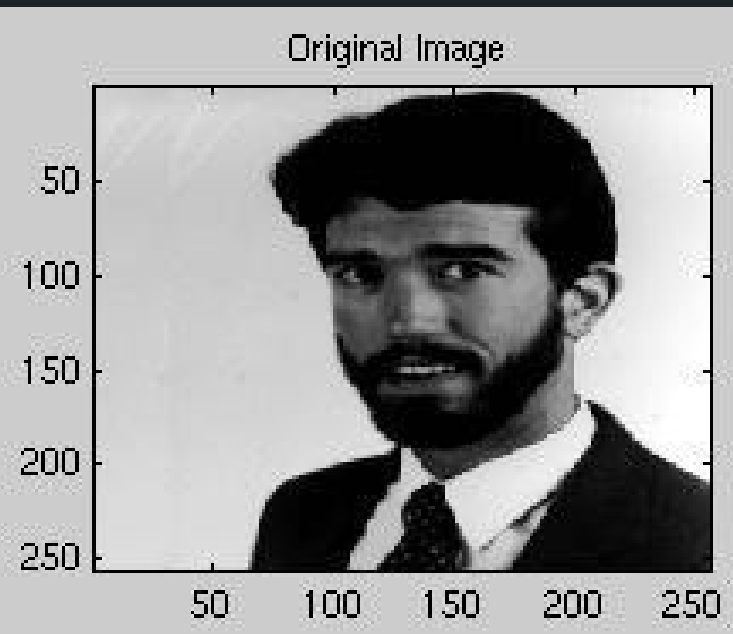
To put it simply, this involves finding out the second image gradients using a method called as convolution.

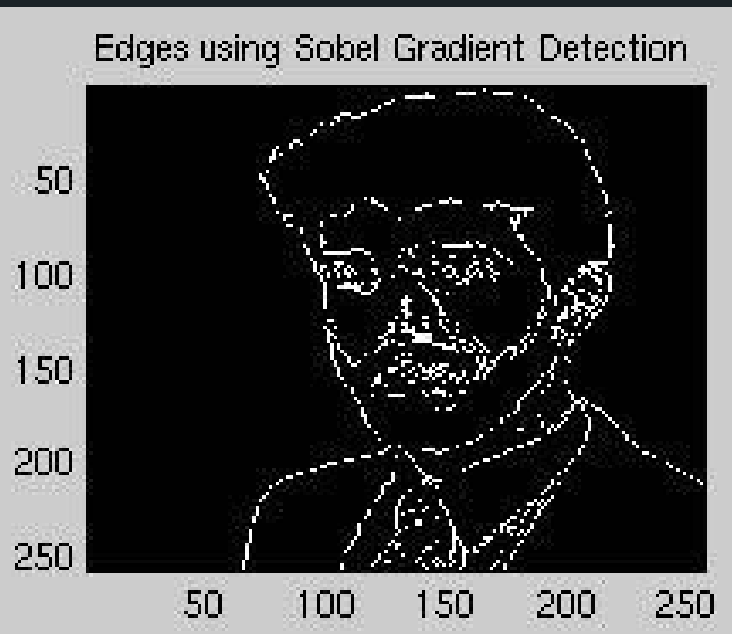
Basic intro to convolution here..

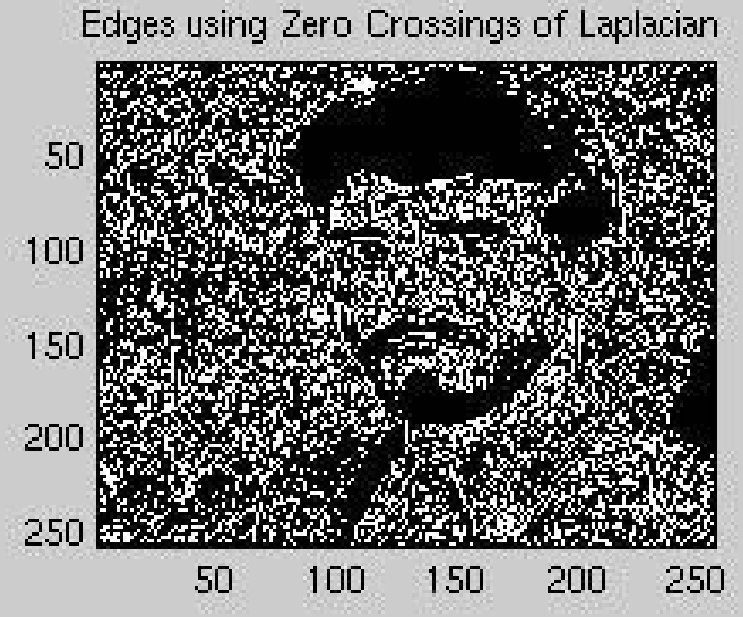
Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Two commonly used small kernels are shown in the figure :

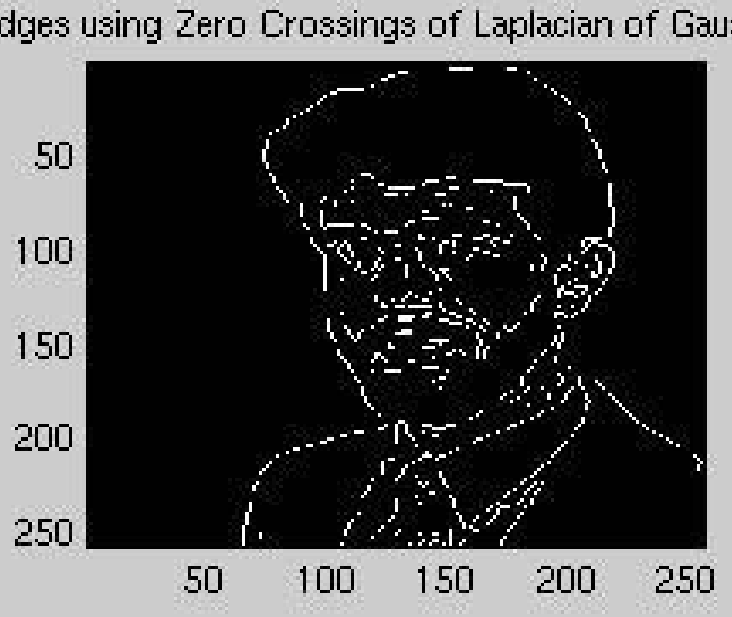
0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1


Original Image
50
100
150
200
250
50
100
150
200
250


Edges using Sobel Gradient Detection
50
100
150
200
250
50
100
150
200
250


Edges using Zero Crossings of Laplacian
50
100
150
200
250
50
100
150
200
250

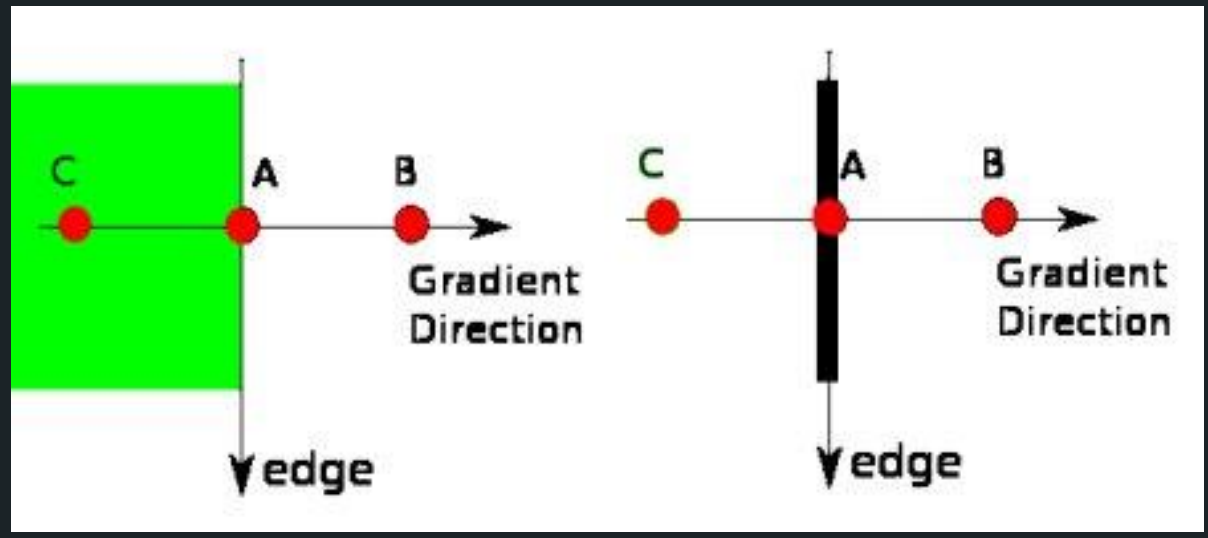

Edges using Zero Crossings of Laplacian of Gaussian
50
100
150
200
250
50
100
150
200
250

Intro to Canny Edge Detection here..

After finding out image gradients, non-maximum suppression is applied here.

Non-maximum Suppression

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient.



Point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).
In short, the result you get is a binary image with "thin edges"

Stuff about coding cv2.canny()

http://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html