

VISMAYA VEERAMANJU KALYAN

CV512 FALL-18

A20423189

Assignment 2

1.

a. SNR = Energy of signal / Energy of noise

= Variance of signal / Variance of noise

= $\left(\frac{1}{n}\right) \sum I(i, j) - I' / \text{Variance of noise}$

Variance of image can be found out by subtracting the intensity of each pixel with the average value of the image.

To find the variance of noise there are 2 methods:

- Capture the image region where all the pixel should have same value. By finding out how much these values vary we can find the variance of noise.
- Take multiple frames at a static screen assuming everything is stagnant. Here a pixel in all the frames have the same intensity, if not take the average at each pixel and how much it deviates gives the variance.
- Usually expressed in terms of Db

b. Gaussian noise follows a gaussian distribution and it's hard to see the noise in the image.

In impulse noise instead of adding noise to each pixel, this model takes certain pixel and a random value.

Median filter handles the Impulse noise better compared to the averaging filter.

c. using zero padding.

2	2	2	2
2	2	2	2
2	2	2	2
2	2	2	2
2	2	2	2

8	12	12	8
12	18	18	12
12	18	18	12
12	18	18	12
8	12	12	8

d. $\frac{d(f*g)}{dx}$ is similar to $\frac{d(f)}{dx} * g = \frac{d(g)}{dx} * f$

We can find the derivate of the image and convolve with the filter to do it more efficiently.

e.

1. Zero Padding: Convolution results in a boundary effect. The result of filtering an image will lead to 0 at the boundary forming an edge. This is because the original image is effectively being padded with 0 values when the convolution kernel extends beyond the original image boundaries.

2. Mirror/ Replicate: Here original image is effectively being padded with values from the image boundaries when the convolution kernel extends beyond the original image boundaries.

3. Ignore: This method is similar to zero padding but all the zeros at the edges are ignored. This is the best way to perform convolution.

f. $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

The sum of all the entries in the filter is 1. When convolving with a basic filter the intensity value of each pixel increases by 9 times. As averaging the neighboring pixel can give a better estimate of the pixel value. The fraction (1/9) is used to normalize the effect. Smoothing is done to remove noise.

g.

$$\begin{aligned}
 I \times G &= \sum_i \sum_j I(i,j) e^{-\frac{i^2+j^2}{2\sigma^2}} \\
 &= \sum_i e^{-\frac{i^2}{2\sigma^2}} \sum_j I(i,j) e^{-\frac{j^2}{2\sigma^2}} \\
 &= (I \times G_y) \times G_x.
 \end{aligned}$$

Two 1-D pass is better compared to one 2-D pass as the number of operations required to calculate are

Two 1-D pass is $2 * M * N * m$

One 2-D pass is $M * N * m^2$

$2MNm > MNm^2$ ($M*N$ – Image size, $m*m$ is the filter size)

No, it's not possible to implement any filter like this. As the gaussian contains exponent term it was possible to break the exponent of a sum down to product of exponent.

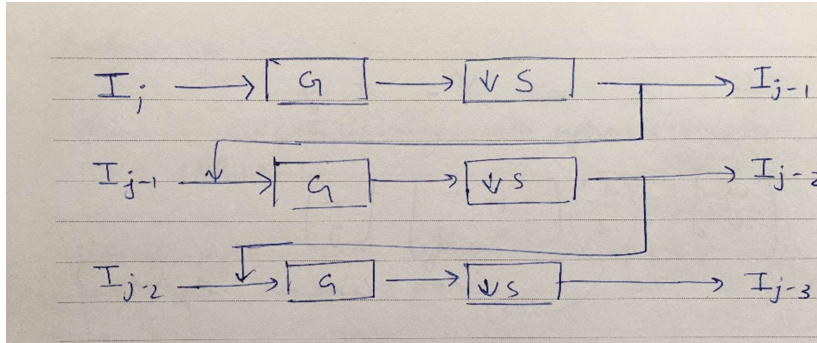
h.

$$\sigma \leq \frac{m}{5}$$

$m = \sigma * 5 = 2 * 5$

size of the filter is 10 or less than 10

i.

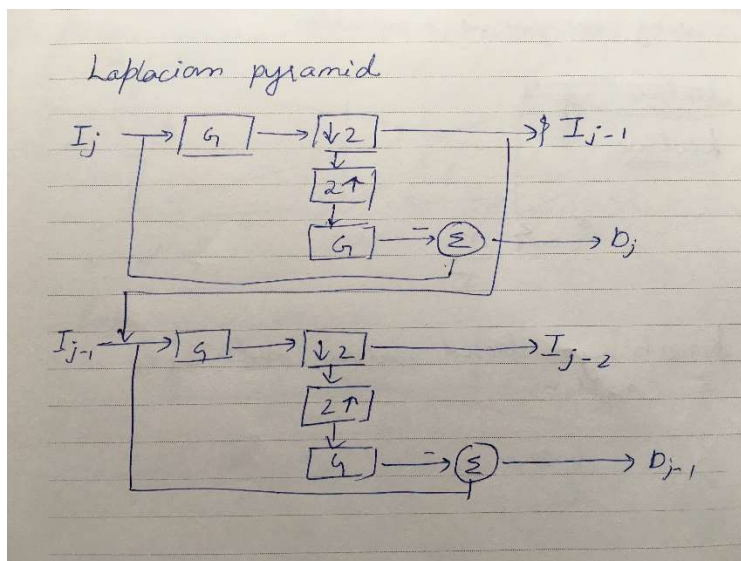


When doing multiple scale analysis, we have two options to detect the background and the foreground. Firstly, to change the window size or secondly, To keep the window size constant and reduce the image size.

When the Image size is reduced to half its size it causes aliasing and introduces noise. Hence, we use the Gaussian pyramid. The Image is first convoluted with a gaussian filter and then down sampled to remove the effect of aliasing caused during sampling.

$$\text{Computational cost} = m^2 + (1/4) m^2 + (1/16) m^2 + \dots < (4/3) m^2$$

j.



A Laplacian pyramid is similar to a Gaussian pyramid but saves the difference image of the blurred versions between each level. Laplacian pyramid is the residual of the images its used for image compression. Like the gaussian pyramid the image is smoothed then down sampled which is again up sampled and applied a gaussian. This resultant image is subtracted from the original image and the residual is stored for using it for compression.

2.EDGE DETECTION

a. To characterize an image we need detect some sort of features, edges are local feature which can be used for this purpose. Edges are useful to find the depth discontinuity, illumination discontinuity or color discontinuity, with the help to which we can recognize objects. They give the general structure of the object.

The desired properties of edges are:

- The edges detected should have some meaning in the image (correspond to an element in a scene)
- It should not be too dependent on the lighting condition or the angle in which the image is taken or the scale of the image.
- It should be a reliable detection i.e. detects edges in few conditions and doesn't work in many of the cases.

b. The basic steps for edge detection are:

1. Smooth the Image (without affecting the edges).
2. Enhancing the edges.
3. detecting the edges.
4. Localizing the edges.

Smoothing is done to reduce noise. As we take derivative of the image and they are very sensitive to noise hence first step is to do the smoothing.

As we have smoothed the image we have to enhance the edges for easy detection.

The edges in the image are the region where there is a change. Hence, we can take the derivative of the image to find out these locations.

Once the edges are found you need to localize them as they won't be pointing at the exact location where they exist on the image.

c. The 2 filter used for computing the image gradient are:

1. Forward difference: Forward difference can be used to find the image derivative. They show the direction of change. It's a 2*2 filter. As we need to calculate the difference from one pixel to its neighboring pixel we take $h = 1$

$$\frac{\partial I(x,y)}{\partial x} = \frac{I(x+h,y) - I(x,y)}{h}$$

$$\frac{\partial I(x,y)}{\partial x} = \frac{I(x+1,y) - I(x,y)}{1}$$

$$\Delta X = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \quad \Delta Y = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

This filter is not symmetric hence we can use the other method.

2. Central difference: It takes the difference between the previous and next neighbor to find the value of the current pixel. Its origin is center. It's a 3*3 matrix symmetric filter.

$$\frac{\partial I(x,y)}{\partial x} = \frac{I(x+h,y) - I(x-h,y)}{2h}$$

$$\frac{\partial I(x,y)}{\partial x} = \frac{I(x+1,y) - I(x-1,y)}{2}$$

$$\Delta X = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \Delta Y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Image gradient is the derivative of the image. It consists of a X and Y vector pointing to the direction of the maximum change in the image.

This can be used for detecting the edge

You can see the magnitude to change at each pixel, having set a threshold you can detect if there exist an edge or not.

d. Sobel first smooths the image and then take the derivative.

We can convolve both the filter to form a single filter.

Smooth * derivative derivative ->

$$\Delta x = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \uparrow \text{smooth}$$

Smooth * derivative smooth ->

$$\Delta y = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \uparrow \text{derivative}$$

e. To get the More accurate derivative of an image, the discrete image is first convolved with a function $h(x)$ to get the continuous function. This continuous function is convolved with the derivative of $h(x)$ which is then sampled to get the more accurate derivative (edges).

$h(x) = \text{sinc}(x)$ which is the perfect interpolator for reconstruction has a infinite tail which would need a infinite filter which is not practically possible hence we approximate and use the gaussian filter.

Steps involved:

1) Convolve with $G_{x,y}$ to convert the discrete image into continuous image.

$$I_x = I * G_x$$

$$I_y = I * G_y$$

2) Take the derivative of the wrt x and y to detect the edges.

Accurate derivative

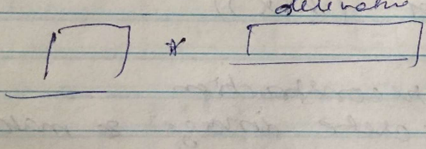
$$I_x = I * G_x$$

$$I_y = I * G_y$$

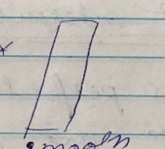
using separable property of gaussian

\downarrow 1D convolution with horizontal gaussian derivative
 $I_x = I * G'_x * G_y \leftarrow$ 1D convolution with vertical derivative

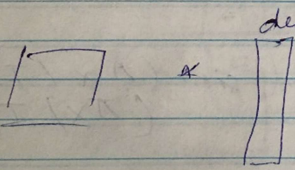
\uparrow 1D convolution with horizontal gaussian
 $I_y = I * G'_y * G_x \leftarrow$ 1D convolution with vertical gaussian derivative



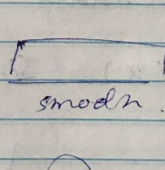
derivative



smooth



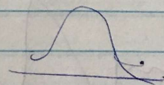
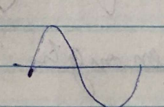
derivative



smooth

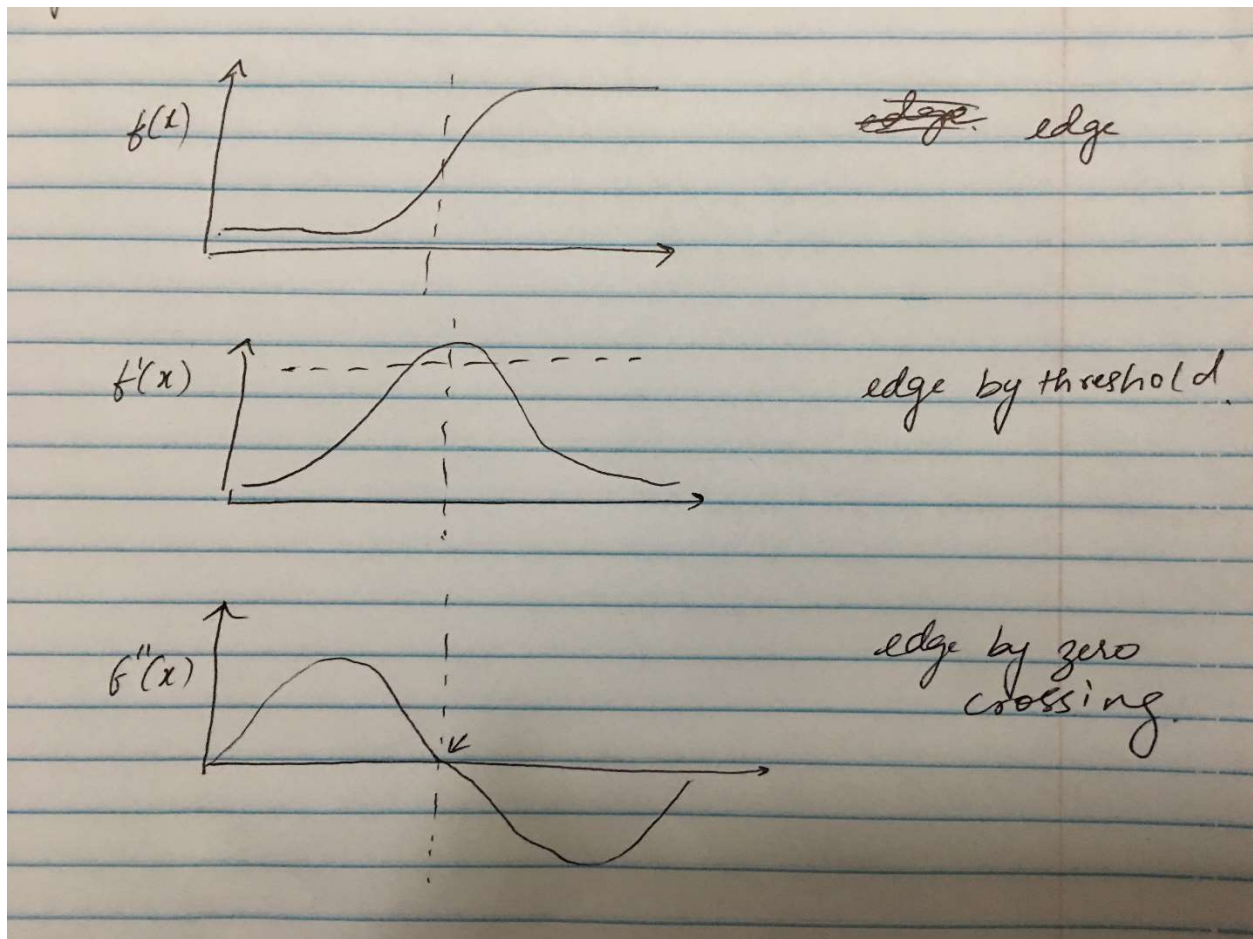
$$G(x) = e^{-\frac{x^2}{2\sigma^2}}$$

$$= \frac{2x}{2\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

3) Now sample back to get the more accurate derivative.

f. Edge can be localized with the first order derivative using the threshold. In case of second order derivative the edges are more localized as they are found at the zero crossing.



g.

1. Convolve $\text{LOG}(\nabla^2 G)$ with the Image

$$H = \nabla^2 G * I$$

2. Threshold $E(i, j) = \begin{cases} 0 & \text{if } H(i, j) < 0 \\ 1 & \text{if } H(i, j) \geq 0 \end{cases}$

3. Mark edges at transition $0 \rightarrow 1$

$1 \rightarrow 0$

Scan left to right and top to bottom.

h. The main difference between canny and other standard edge detection is that canny uses the directional derivatives instead of derivatives along x and y as in the case of standard edge detection algorithm. The other difference is standard algorithm use only one thresholding while canny uses 2 different threshold to detect edges.

Condition for detecting an edge in canny is τ_h to start tracking and τ_L to continue ($\tau_h > \tau_L$)

1. Scan image top to bottom and left to right:
If $v(i, j)$ and $|\nabla I| > \tau_h$ start tracking the edge.

2. Search for additional neighbours in direction orthogonal to ∇I such that $|\nabla I| > \tau_L$

i.

Non-maximum suppression:

Non-max suppression is a way to eliminate points that do not lie on important edges.

Once we have the gradient magnitude and direction we eliminate 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

$$\Theta = \tan^{-1}(I_y/I_x)$$

$$E(I, j) = \begin{cases} 1 & \text{if } \nabla(I * G) \text{ is a local maximum} \\ 0 & \text{otherwise} \end{cases}$$

Hysteresis thresholding

Use two thresholds. Using τ_h to start tracking and τ_L to continue ($\tau_h > \tau_L$)

1. Initialize array to a visited pixel $v(I, j) = 0$
2. Scan image top to bottom and left to right:
If $v(I, j)$ and $|\nabla I| > \tau_h$ start tracking the edge.
3. Search for additional neighbours in direction orthogonal to ∇I such that $|\nabla I| > \tau_L$