

CS420 IMAGE UNDERSTANDING

© Tingfeng Xia

Fall Term, 2020

This work is licensed under a [Creative Commons “Attribution-NonCommercial-ShareAlike 4.0 International”](#) license.



Information

- Syllabus can be found through [this link](#).
- There will be office hours every weekday. :D

Contents

1	Linear Filter	3
1.1	Fourier Transform Overview	3
1.1.1	Sinusoidal Waves	3
1.1.2	Vector Forms	3
1.1.3	Inner Product on Functions Space	3
1.1.4	General Form (Periodic Function) - Fourier Series	3
1.1.5	General Form	4
1.2	Image Representation	4
1.2.1	Image	4
1.2.2	Image Coordinates	4
1.2.3	Coloured Images	4
1.2.4	Image Transformations	4
1.3	Noise Reduction	5
1.3.1	1-D Example	5
1.3.2	2-D Case	5
1.4	Correlation Defined	6
1.4.1	General Moving Average	6
1.4.2	General Filtering	6
1.4.3	Notation	6

1.4.4	Correlation - Vector Form	6
1.4.5	Normalized Cross-correlation	7
1.5	Boundary Effects	7
1.6	Smoothing	7
1.6.1	Uniform Smoothing	7
1.6.2	Isotropic Gaussian Filter	8
1.6.3	Non-isotropic Gaussian Filter	9
1.7	Convolution	9
1.7.1	Properties of Convolution	9
1.7.2	Convolution w/ Fourier Transforms	9
1.8	Separable Filters	9
1.8.1	Isotropic Gaussian as Separable Filters	10
1.8.2	Moving Average as Separable Filters	10
1.8.3	Edge Detector Kernel as Separable Filters	10
1.8.4	Separable-ness of Filters	10
2	Edge Detection	11
2.1	Characterization of Edges	11
2.1.1	Insights	11
2.1.2	Origin of Edges	11
2.1.3	Characterizing Edges	12
2.2	Convolution as Derivative - Measure of Rapid Change	12
2.2.1	Canonical Finite Difference Filters	12
2.3	Image Gradient	13
2.3.1	Gradient Defined	13
2.3.2	Edge Direction	13
2.3.3	Gradient Direction	13
2.3.4	Edge Strength	13
2.4	Effects of Noise	14
2.4.1	Overcoming Noisiness	14
2.4.2	Faster Method Through Conv / Corre	14
2.4.3	Generalization: Derivative Theorem of Convolution	14
2.4.4	Remark: on Gaussian Derivative Filters' Parameter	15
2.5	Canny's Edge Detector	15
2.5.1	Procedure	15
2.5.2	Non-Maximum Suppression (Thick Edges Problem)	15
2.5.3	Hysteresis Thresholding (Discontinuous Edges Problem)	16
2.6	Laplacian of Gaussians: Another Approach to Edge Detection	16

1 Linear Filter

1.1 Fourier Transform Overview

1.1.1 Sinusoidal Waves

Consider a sinusoidal wave, it has a general form of

$$A \cos(\omega t - \phi) \quad (1.1)$$

where A is the amplitude, $\omega = 2\pi f$, where f is the frequency, and ϕ is called phase. Phase describes the horizontal shifting for the wave away from the standard position.

1.1.2 Vector Forms

Consider a vector $\mathbf{v} = (3, 2)^\top$, we write it as

$$\mathbf{v} = 3\mathbf{i} + 2\mathbf{j} = (\mathbf{v} \cdot \mathbf{i})\mathbf{i} + (\mathbf{v} \cdot \mathbf{j})\mathbf{j} \quad (1.2)$$

and we call $\{\mathbf{i}, \mathbf{j}\}$ an orthonormal basis^{1.1}.

1.1.3 Inner Product on Functions Space

Function space is an inner product space, and we can define, for a set of parametric function, on an interval of $[a, b]$ that

$$\langle f(t), g(t) \rangle = \int_a^b f(t)g(t) dt \quad (1.3)$$

1.1.4 General Form (Periodic Function) - Fourier Series

For a function $f(t)$ that is periodic with period of T , the general form can be written as

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi kt}{T}\right) + \sum_{k=1}^{\infty} b_k \sin\left(\frac{2\pi kt}{T}\right) \quad (1.4)$$

$$= \frac{a_0}{2} + \sum_{k=1}^{\infty} A_k \cos\left(\frac{2\pi kt}{T} - \phi_k\right) \quad (1.5)$$

where in the second form, we merged all the sin and cos terms with the introduction of a phase term. In particular, this now takes the same form as discussed in Section 1.1.1.

^{1.1} $\|\mathbf{i}\| = \|\mathbf{j}\| = 1 \wedge \mathbf{i} \cdot \mathbf{j} = 0$

1.1.5 General Form

The general form also takes into consideration of non-periodic functions. In such case, we need to change the summation into a integral. For any function $f(t)$, we have

$$F(\omega) = \int f(t)e^{-i\omega t} dt \quad (1.6)$$

1.2 Image Representation

1.2.1 Image

Image is a matrix with integer values. The matrix would typically be denoted as I , and $I_{i,j}$ is called the **intensity**. For each pixel, we usually represent it use an unsigned 8 bit unsigned integer and thus have range $2^0 = 0$ to $2^8 - 1 = 255$. For High Dynamic Range (HDR) images, they will be represented with 16 bit unsigned integer. Also there are cases that we (linearly) normalize the values by squashing them into $[0, 1]$.

1.2.2 Image Coordinates

Image coordinates start from the top left. For a coordinate (i, j) , i specifies that it is in the i -th row, and j specifies column. Also worth noticing that the most upper left pixel has coordinates $(1, 1)$.

1.2.3 Coloured Images

In grey scale images mentioned in the previous two parts, for a image of size $m \times n$, we have a matrix of $m \times n$ 8-bit unsigned integers. Now with the introduction of colours, we will have a tensor of $m \times n \times 3$ 8-bit ints, corresponding to three colour channels. By convention, they usually goes in the order of $R \rightarrow G \rightarrow B$. For example, $I(2, 3, 1)$ means the intensity of **red** channel of the image at location row 2 and column 3.

1.2.4 Image Transformations

For simplicity, we start with grey scale images. We can view any image as a function $f : \mathbb{R}^2 \rightarrow \mathbb{Z}_{0-255}$, and this enables us to transform images. An easy example would be to increase the brightness of the image, which we can achieve so with

$$J(i, j) = \min \{I(i, j) + \text{amount}, 255\} \quad (1.7)$$

capping the max intensity at 255. Importantly, we can so some interesting operations by treating images as functions. Namely, correlation and convolution.

1.3 Noise Reduction

1.3.1 1-D Example

Consider a signal, which is a real to real function, and our goal is to smooth the function by imposing human knowledge that the signal should be smooth and should not contain too many jitter. We can have

- **Moving Average Filter**, which is $[1, \dots, 1]/n$, or
- **Non-uniform Weights**, for example $[1, 4, 6, 4, 1]/16$

1.3.2 2-D Case

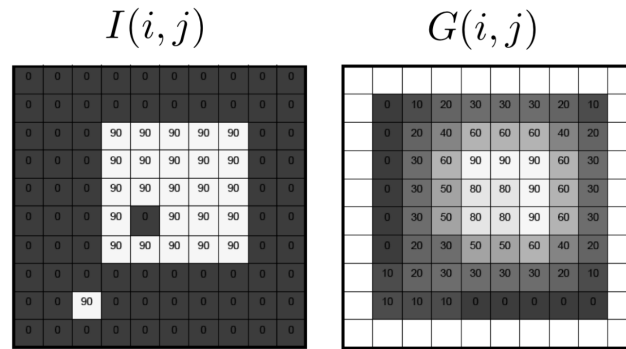


Figure 1: Example of using moving average to smooth out an image, *assuming no padding*.

Much similar to the 1-D case mentioned above, we have our choice of whether to choose an uniform filter or not.

- In the case of uniform (aka moving average), we choose (example of 3×3 filter)

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} / 9 \quad (1.8)$$

Figure 1 is an example of using moving average to smooth out an image, *assuming no padding*. Notice that the sharp boundaries that we used to have between dark and white is now smooth. Also, the isolated pixels $((6, 5)$ and $(9, 3)$) are now blended in.

- In case of non-uniform, we can again choose a gaussian like filter, such as

$$\begin{bmatrix} 1 & 4 & 1 \\ 4 & 10 & 4 \\ 1 & 4 & 1 \end{bmatrix} / 30 \quad (1.9)$$

De-noising is usually an important first step (pre-processing) in any image task

1.4 Correlation Defined

1.4.1 General Moving Average

In the general case, our filter could be any size. In particular, it needs to be of size square of an odd number. Then, the moving average becomes

$$G(i, j) = \frac{1}{(2k+1)^2} \sum_{u=-k}^k \sum_{v=-k}^k I(i+u, j+v) \quad (1.10)$$

1.4.2 General Filtering

If we apply some filter (i.e., not just a average) then we can use

$$G(i, j) = \sum_{u=-k}^k \sum_{v=-k}^k F(u, v) \cdot I(i+u, j+v) \quad (1.11)$$

where $F(\cdot, \cdot) : \mathbb{R}^2 \rightarrow \mathbb{R}$ is the called **kernel** or **mask** or **filter** such that $\sum_{u'} \sum_{v'} F(u, v) = 1$. The elements of the filter is called **filter coefficients**. Notice that if we take $(|V|, |U|)$ here denotes the max values that v, u can take)

$$F(u, v) = \frac{1}{(2|U|+1)^2} = \frac{1}{(2|V|+1)^2} \quad (1.12)$$

then Equation 1.11 just collapses to Equation 1.10.

1.4.3 Notation

The Filtering operation defined above is called *correlation*, denoted as

$$G = F \otimes I \quad (1.13)$$

where F is our filter / kernel / mask, and I is the original image.

Notice that filter is also an image, so \otimes essentially takes two images as input and outputs one image.

1.4.4 Correlation - Vector Form

Define

- $\mathbf{f} = F(\cdot)$, writing the matrix into a vector.
- $T_{ij} = I(i-k : i+k, j-k : j+k)$, the part of image covered by the filter around original image at coordinates (i, j)
- $\mathbf{t}_{ij} = T_{ij}(\cdot)$ putting the part of image selected in previous step into a vector.

then,

$$G(i, j) = \langle \mathbf{f}, \mathbf{t}_{ij} \rangle = \|\mathbf{f}\| \|\mathbf{t}_{ij}\| \cos \theta \quad (1.14)$$

which converts two for loops into one inner product. This is much faster to compute as far as codes are concerned.

TODO: above we defined correlation for one pixel as a vector operation, can we define the entire correlation into matrix form

1.4.5 Normalized Cross-correlation

In the task of finding Waldo, we wish to get a score of whether or not a patch of image looks like Waldo. In particular, we want this score to be the highest for the patch with Waldo, but not a very bright patch without Waldo. In an hope to achieve this goal, we can use normalized cross correlation: (utilizing the vector forms in the previous section)

$$G(i, j) = \frac{\mathbf{f}^\top \mathbf{t}_{ij}}{\|\mathbf{f}\| \|\mathbf{t}_{ij}\|} = \cos \theta \quad (1.15)$$

where θ is the angle between vectors \mathbf{f} and \mathbf{t}_{ij}

1.5 Boundary Effects

Assume we have image size of $m \times n$, and filter size of $k \times k$. Referring to `cv2.filter2d` in OpenCV and `filter2(F, I, SHAPE)`, we have the following cases:

- `shape = 'full'` output size is bigger than the image; infinite padding include all reasonable values. Output should have size $(m + 2k - 2) \times (n + 2k - 2)$
- `shape = 'same'` output size is same as I ; padding such that output size is equal to input size.
- `shape = 'valid'` output size is smaller than the image; no zero padding; output should be size $(n - k + 1) \times (m - k + 1)$

1.6 Smoothing

1.6.1 Uniform Smoothing

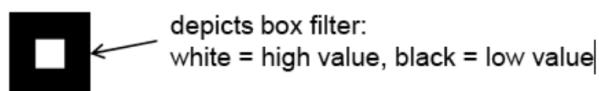


Figure 2: Box Filter

The box filter depicted in Figure 2 is the exact same filter if we only keep the white part, i.e. the 1 entries. As the size of the box filter increases, the end result gets more and more blurry.

1.6.2 Isotropic Gaussian Filter

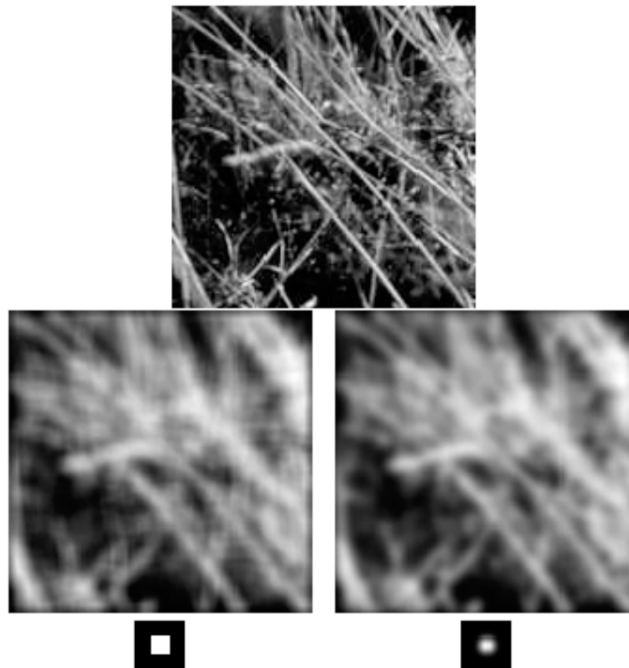


Figure 3: Comparison of filtered result using uniform filter (bottom left) and gaussian filter (bottom right)

Recall that the gaussian probability distribution is defined as

$$\text{Gaussian}(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = (2\pi)^{-\frac{k}{2}} \det(\Sigma)^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})} \quad (1.16)$$

and we can mimic this to develop a filter that has entries mimicking values taken by the gaussian pdf. These filters produce results much nicer than averaging in terms of smoothing, as we can see in Figure 3.

Specification The Gaussian Filter G is parametrized by two parameters $\Sigma = \sigma I$ (isotropic) and $\boldsymbol{\mu}$. In application, $\boldsymbol{\mu}$ doesn't matter, we always want to make sure that the peak of the gaussian pdf corresponds to the centre pixel of the filter. The size of the filter depends on our choice of Σ , e.g. it doesn't make much sense if our kernel includes values more than 2σ 's away. We also need to normalize all the taken values again, to make sure the filter we chose sums up to one!

1.6.3 Non-isotropic Gaussian Filter

In the most general case, Gaussian can be non-isotropic, meaning that its variance-covariance matrix *is not* of form σI

1.7 Convolution

The Convolution operation is defined as

$$G(i, j) = \sum_{u=-k}^k \sum_{v=-k}^k F(u, v) \cdot I(i - u, j - v) \quad (1.17)$$

Notice that this is exactly the same as Correlation defined in Equation 1.11 except that we are flipping the filter in both dimensions (bottom to top, right to left).

In the case of Gaussian / box filters, since the filter will be symmetric about both horizontal and vertical axis, $F * I = F \otimes I$.

1.7.1 Properties of Convolution

Convolution is a Linear Operation, meaning that if f, g and h are three convolution operators, and $\lambda \in \mathbb{R}$ then

- **Commutative:** $f * g = g * f$
- **Associative:** $f * (g * h) = (f * g) * h$
- **Distributive:** $f * (g + h) = f * g + f * h$
- **Assoc. with scalar multiplier:** $\lambda \cdot (f * g) = (\lambda \cdot f) * g$

1.7.2 Convolution w/ Fourier Transforms

The Fourier transform of two convolved images is the inner product of their individual Fourier Transforms, i.e.

$$\mathcal{F}(f * g) = \langle \mathcal{F}(f), \mathcal{F}(g) \rangle \quad (1.18)$$

Implications The computational complexity of Fourier Transform is much lower than that of convolution. Also notice that inner products are fast to compute.

Confirm and finish this !

1.8 Separable Filters

For a $K \times K$ sized filter / kernel, the process of performing a convolution requires K^2 operations per pixel, summing up to a total of $\# \text{pixels} \times K^2$ for an entire image. In many cases, though not all, we can speed this process up by (1) performing a 1-D horizontal convolution followed by (2) a 1D vertical convolution, requiring only $2K$ operations! When

we can do this trick, we call the kernel “separable”. And a filter is separable iff it is the outer product of two vectors (each a 1D filter):

$$\exists? \mathbf{v}, \mathbf{h} \in \mathbb{R}^K, s.t. F = \mathbf{v}\mathbf{h}^\top \quad (1.19)$$

1.8.1 Isotropic Gaussian as Separable Filters

One famous example of separable filter that we are already familiar with is the Gaussian Filter, assuming isotropic variance. In such case the density breaks down into

$$\text{Gaussian}(x, y; \boldsymbol{\mu} = \mathbf{0}, \Sigma = \sigma I) = \frac{1}{2\pi\sigma^2} \exp \left\{ -\frac{x^2 + y^2}{\sigma^2} \right\} \quad (1.20)$$

$$= \left[\frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{x^2}{\sigma^2} \right\} \right] \cdot \left[\frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{y^2}{\sigma^2} \right\} \right] \quad (1.21)$$

Such factorization of gaussian pdf indicates us that we should have two 1-D filters that are 1-D gaussian each.

1.8.2 Moving Average as Separable Filters

The naïve moving average filter that we encountered earlier is also separable, in which case

$$F = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix} / K^2 = [1/K \quad \dots \quad 1/K]^\top [1/K \quad \dots \quad 1/K] \quad (1.22)$$

1.8.3 Edge Detector Kernel as Separable Filters

The edge detector

$$F = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} / 8; \quad \mathbf{v} = [-1 \quad 0 \quad 1] / 2; \quad F = \mathbf{v}^\top \mathbf{v} \quad (1.23)$$

Come back:
left or right
edge detector?

1.8.4 Separable-ness of Filters

A systematic way of checking if a kernel is separable is by looking at the singular value decomposition of the filter. ***If only one singular value is non-zero, then it is separable***

$$F = \mathbf{U}\Sigma\mathbf{V}^\top = \sum_i \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \quad (1.24)$$

with $\Sigma = \text{diag}(\sigma_i)$. Then, we can get the vertical and horizontal filter through

$$F_{\text{vertical}} \leftarrow \sqrt{\sigma_1} \mathbf{u}_1 \quad F_{\text{horizontal}} \leftarrow \sqrt{\sigma_1} \mathbf{v}_1^\top \quad (1.25)$$

2 Edge Detection

2.1 Characterization of Edges

2.1.1 Insights

- Edge detection involves mapping image to a set of *curves* or *line segments* or *contours*.
- Such representation is more compact than pixels. Notice that for a coloured (ordinary, not HDR) $m \times n$ image, we will need $m \times n \times 3 \times 8$ bits to store all the information. However, for edges $m \times n$ would suffice.
- They are particularly useful due to their invariance towards illumination - it thus helps computers see better. Aside: edges are also important for recognition for human.

2.1.2 Origin of Edges

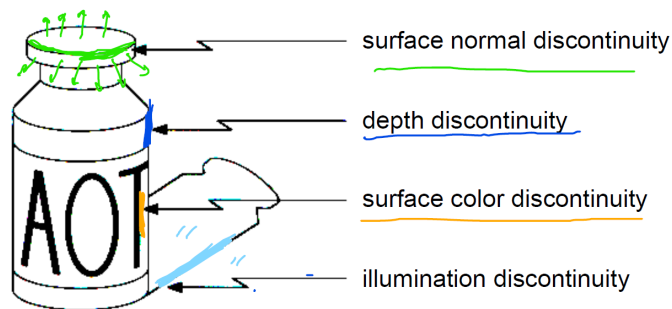


Figure 4: Four origins of edges

Figure 4 illustrates the four types of edges that can occur.

- **Surface Normal Discontinuity** is where surface normals change direction abruptly.
- **Depth Discontinuity** is caused by depth discrepancy between two objects from the angle of the viewer. For example here in Figure 4, the background and the bottle causes a depth discontinuity.
- **Surface Colour Discontinuity** is for example the edge of black text T on a white bottle.
- **Illumination Discontinuity** is when there is a shadow causing difference in light.

2.1.3 Characterizing Edges

Definition An edge is a place of rapid change in image intensity function. The means that at places where edges occur, the intensity function should be steep, and the first derivative of the intensity at that position should correspond to extrema.

2.2 Convolution as Derivative - Measure of Rapid Change

Consider an image $f(x, y)$ defined for $x \in \mathbb{Z}^{\geq 1, \leq m}$, $y \in \mathbb{Z}^{\geq 1, \leq n}$, how can we differentiate this digital image given that it is not continuous? The answer is we take the first order forward discrete derivative (finite difference), i.e.

$$\frac{\partial f(x, y)}{\partial x} \approx \frac{f[x + 1, y] - f[x, y]}{1} \quad (2.1)$$

and

$$\frac{\partial f(x, y)}{\partial y} \approx \frac{f[x, y + 1] - f[x, y]}{1} \quad (2.2)$$

Correlation Filter Clearly we can implement the above as kernels,

$$H_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad H_y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (2.3)$$

then, we have

$$[f \otimes H_x]_{(i,j)} = \left. \frac{\partial f(x, y)}{\partial x} \right|_{(i,j)} \quad \text{and} \quad [f \otimes H_y]_{(i,j)} = \left. \frac{\partial f(x, y)}{\partial y} \right|_{(i,j)} \quad (2.4)$$

2.2.1 Canonical Finite Difference Filters

The Prewitt Kernel is more symmetric, and averages each pixel from neighbouring pixels only. Also this filter applies a tiny blurring on the direction that it is not detecting edge. (M_x is blurring for vertical direction.)

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad (2.5)$$

Sobel Filter is more common. Same as Prewitt, in the direction of that the kernel is not detecting edges, it applies a bit blurring effect. *However, in Sobel, the blurring is a Gaussian blur.*

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & - \\ -1 & -2 & -1 \end{bmatrix} \quad (2.6)$$

Roberts Kernel detects diagonal edges;

$$M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{and} \quad M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad (2.7)$$

2.3 Image Gradient

2.3.1 Gradient Defined

The gradient of an image $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ is defined, exactly the same as usual, as

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \quad (2.8)$$

2.3.2 Edge Direction

The gradient always points in the direction of most rapid change in intensity. This means that if

$$\nabla f = \left[\frac{\partial f}{\partial x} \neq 0, \rightarrow 0 \right] \quad (2.9)$$

then that position corresponds to a vertical edge. If

$$\nabla f = \left[\rightarrow 0, \frac{\partial f}{\partial y} \neq 0 \right] \quad (2.10)$$

when we are at a horizontal edge. At a slanted edge, we will get

$$\nabla f = \left[\frac{\partial f}{\partial x} \neq 0, \frac{\partial f}{\partial y} \neq 0 \right] \quad (2.11)$$

2.3.3 Gradient Direction

The gradient direction (i.e. orientation of edge normal) is given by

$$\tan \theta = \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \implies \theta = \arctan \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right) \quad (2.12)$$

2.3.4 Edge Strength

The edge strength is given by the L_2 norm of the gradient vector:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2} \quad (2.13)$$

2.4 Effects of Noise

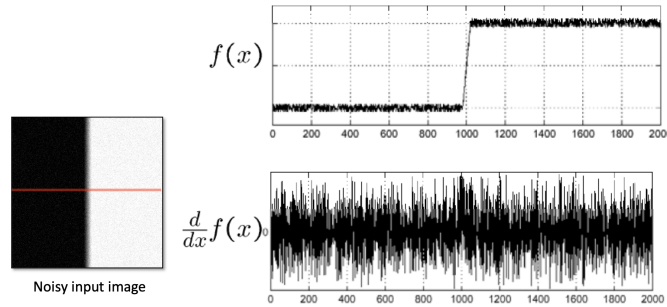


Figure 5: Illustration of noisy input problem.

As shown in Figure 5, when we have edges that are not sharp, the image is noisy and thus cause the derivative messy.

2.4.1 Overcoming Noisiness

The solution to the problem is easy. We simply first smooth the input signal and then look for edges. I.e., for an input image f , and noise reduction filter (e.g. Gaussian) h , we find extremum in $\frac{\partial}{\partial x} (h * f)$

2.4.2 Faster Method Through Conv / Corre

We know that correlation is associative, i.e. $f * (g * h) = (f * g) * h$. We can use this property to speed up our method of overcoming noisiness mentioned in Section 2.4.1. Consider an image $I \in M_{m \times n}(\mathbb{R})$, smoothing filter $G \in M_{k \times k}(\mathbb{R})$, and x -derivative kernel $F \in M_{k \times k}(\mathbb{R})$. Then,

$$\frac{\partial}{\partial x} (G * I) = F * (G * I) \quad (2.14)$$

$$= (F * G) * I \quad (2.15)$$

and since $k \ll m, n$, Equation 2.15 is much faster to compute.

2.4.3 Generalization: Derivative Theorem of Convolution

For a image f , with filter h , we have

$$\frac{\partial}{\partial x} (h * f) = \left(\frac{\partial h}{\partial x} \right) * f = h * \left(\frac{\partial f}{\partial x} \right) \quad (2.16)$$

i.e. rather than convolving the image with the filter and then take the derivative, we first get the derivative of the filter and convolve the result with the image.

This saves us one operation.

2.4.4 Remark: on Gaussian Derivative Filters' Parameter

We know that if we apply the derivative of gaussian as a filter to an image, it finds the edges on the smoothened version of the image. However, the detected structures differ depending on the Gaussian's std. deviation chosen.

- If we have a large σ , then the filter detects edges of larger scale
- else if we have a small σ , we will be detecting finer structures.

2.5 Canny's Edge Detector

2.5.1 Procedure

1. Filter images with derivative of gaussian (horizontal and vertical directions)
2. Find magnitude and orientation of gradient
3. Non-maximum suppression
4. Linking and thresholding (hysteresis)
 - Define two thresholds: low and high
 - Use the high threshold to start edge curves and the low threshold to continue them

Parameters There are three parameters (hyper-parameters) that we need to fix for the algorithm, namely scale of Gaussian in step 1, and low / high thresholds in step 4. There are no magical way in tuning them, so it requires quite a lot of experimentation.

2.5.2 Non-Maximum Suppression (Thick Edges Problem)

- check if pixel is local maximum along gradient direction?
- if yes, take it; otherwise neglect it.

i.e. in an image f we take (i, j) if in a local area

$$\forall (i', j'), I(i, j) > I(i', j') \quad (2.17)$$

then keep $\|\nabla f\| \leftarrow \|\nabla f\|$ otherwise $\|\nabla f\| \leftarrow 0$

2.5.3 Hysteresis Thresholding (Discontinuous Edges Problem)

- Filter at high threshold, getting strong edges. Call this result S
- Filter at low threshold, getting weak edges, call this result W
- Along directions where edges develop in S , if there is an edge in W at the same spot, then we adopt it. Continue until a point where norm of gradient is below the low threshold.

2.6 Laplacian of Gaussians: Another Approach to Edge Detection

Consider (noisy) image f , and a Gaussian filter h . We have the laplacian of h , as $\frac{\partial^2}{\partial x^2} h$, and then we calculate

$$\left(\frac{\partial^2}{\partial x^2} h \right) * f \tag{2.18}$$

where zero-crossings in the resulting graph correspond to edges.