CSE428: Image Processing

Lecture 6

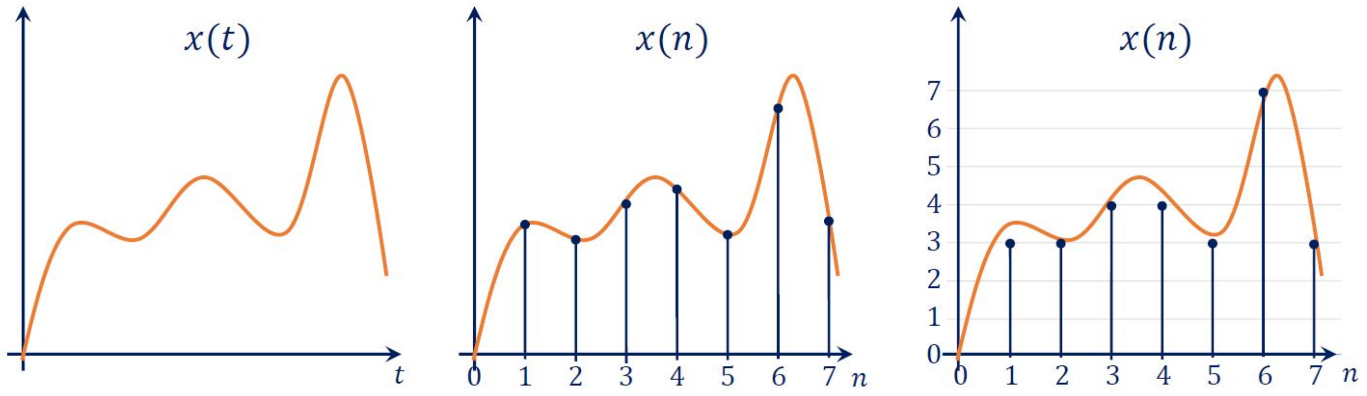
Neighborhood processing: Part 2

Contents

Continuous vs. Discrete

Analog to Digital conversion (sampling + quantization)

- Original signal is sampled in time axis and quantized in y axis to represent in digital systems



Derivatives

Derivatives

- Gives you the rate at which quantities change
- For continuous function **y = *f*(x)**, the derivative is defined as the following limit:

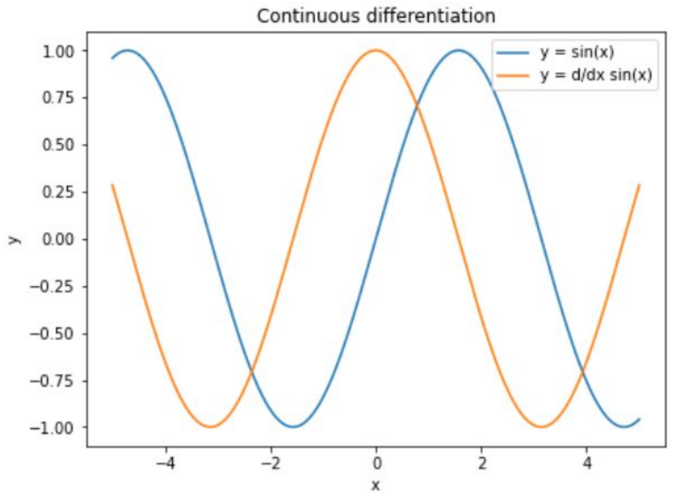
$$\frac{dy}{dx} = f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

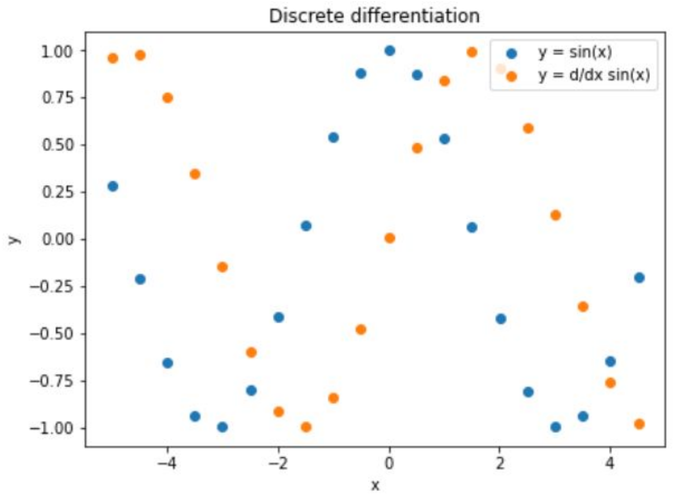
- For discrete functions, the derivatives are calculated numerically, called **Numerical Differentiation**
- **lim: Δ*x*⟶0** becomes problematic for discrete cases (you can't go lower than a pixel)

Derivatives: Discrete Example

Derivatives in discrete cases are approximated by “Numerical Differentiation”

- Tensorflow automatic differentiation



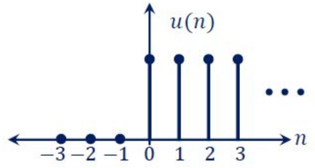


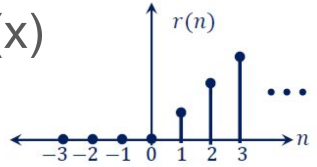
Digital Signal Derivatives: 1D Case

The derivatives of a digital function are defined in terms of ***differences***

We *require* that any definition we use for a **first derivative**

1. must be zero in areas of constant value (intensity for images)
2. must be nonzero at the onset of an intensity step *u*(x) or ramp *r*(x)
3. must be nonzero along ramps





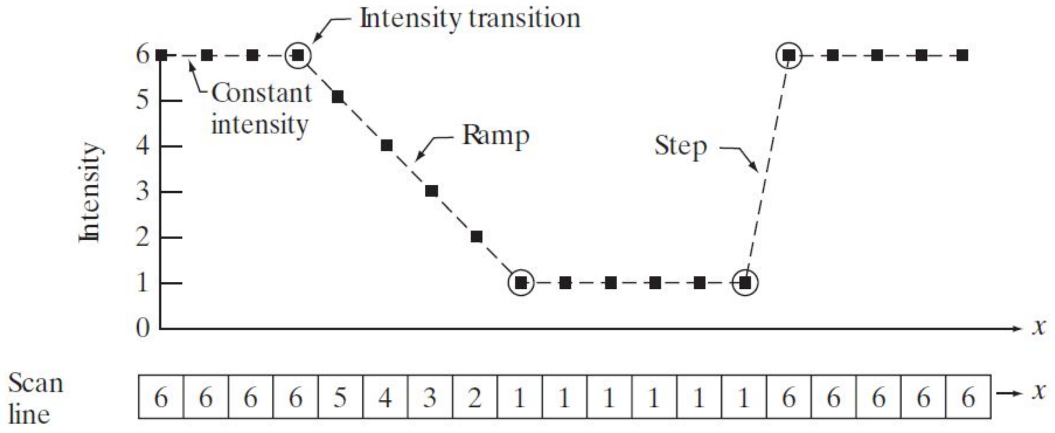
A basic definition of the first-order derivative of a one-dimensional digital function *f*(x) is the difference:

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

Forward difference equation

Digital Signal Derivatives: 1D Case

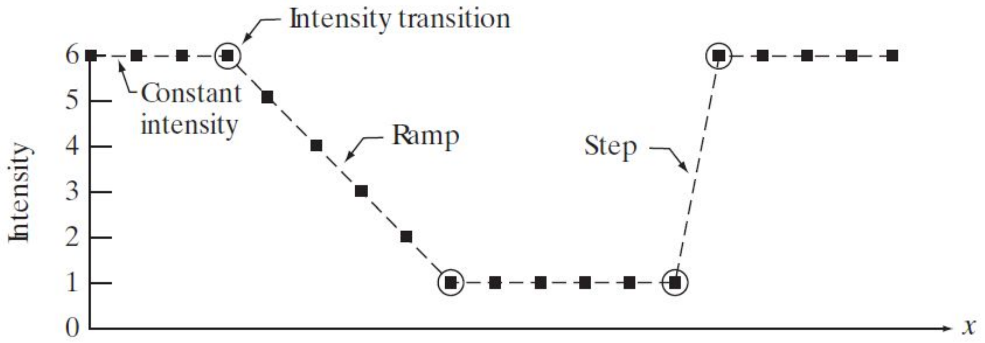
Example of a 1D digital function, *f*(*x*)



Digital Image Processing, Third Edition,
Rafael C. Gonzalez & Richard E. Woods

Digital Signal Derivatives: 1D Case

Calculate the first derivative using the forward difference equation *f*(x+1) - *f*(x)



Scan line	6	6	6	6	5	4	3	2	1	1	1	1	1	1	6	6	6	6	6
1st derivative		0	0	−1	−1	−1	−1	−1	0	0	0	0	0	5	0	0	0	0	

→ x

Digital Image Processing, Third Edition, Rafael C. Gonzalez & Richard E. Woods

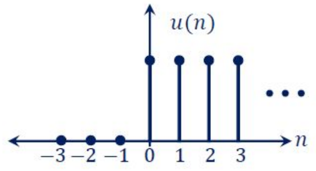
Also can be thought of as linear filtering with kernel [-1 1]

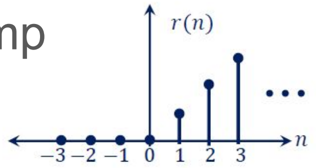
Digital Signal Derivatives: 1D Case

The derivatives of a digital function are defined in terms of ***differences***

we require that any definition we use for a **second derivative**

1. must be zero in areas of constant value (intensity for images)
2. must be nonzero at the onset and end of an intensity step or ramp
3. must be zero along ramps of constant slope



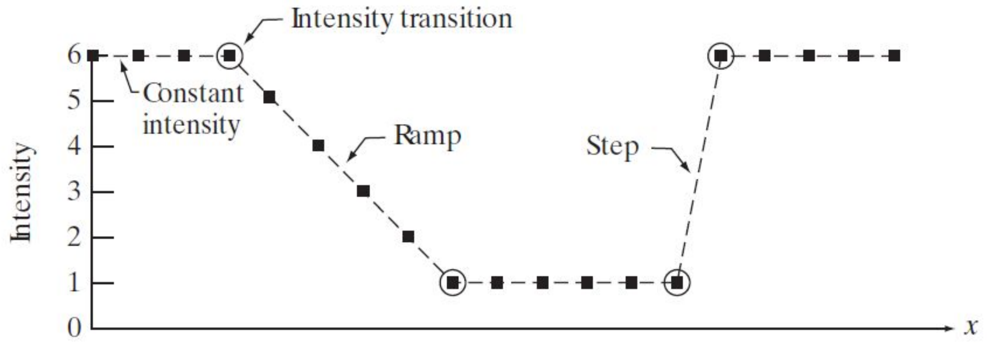


A basic definition of the second-order derivative of a one-dimensional digital function *f*(x) is the difference:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

Digital Signal Derivatives: 1D Case

Calculate the second derivative using the difference equation *f*(x+1) + *f*(x-1) - 2*f*(x)



Scan line	6	6	6	6	5	4	3	2	1	1	1	1	1	1	6	6	6	6	6
1st derivative		0	0	−1	−1	−1	−1	−1	0	0	0	0	0	5	0	0	0	0	
2nd derivative		0	0	−1	0	0	0	0	1	0	0	0	0	5	−5	0	0	0	

→ x

Digital Image Processing, Third Edition,
Rafael C. Gonzalez & Richard E. Woods

Also can be thought of as linear filtering with kernel [1 -2 1]

Norm

Norm of a vector

- gives you an “idea” of the “size” of a vector (the magnitude)
- maps a vector to a non-negative quantity

The p-norm (*L^p*) of a vector ***x*** is defined as

$$||\boldsymbol{x}||_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$

Digital Signal Derivatives: 2D Case

1st order derivative and the <u>Gradient</u>

- For a function *f*(x, y) the gradient of *f* at coordinates (x, y) is defined as the two-dimensional column vector

$$\nabla f \equiv \operatorname{grad}(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix}$$

- And its 2-norm (L^2):

$$M(x, y) = \operatorname{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2}$$

Application: Gradient Based Edge Detection

Sharpening kernel formulation based on image **Gradient**

- The Sobel Operator

Using the L^1 norm of the gradient

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

x-axis : $$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

Y-axis: $$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

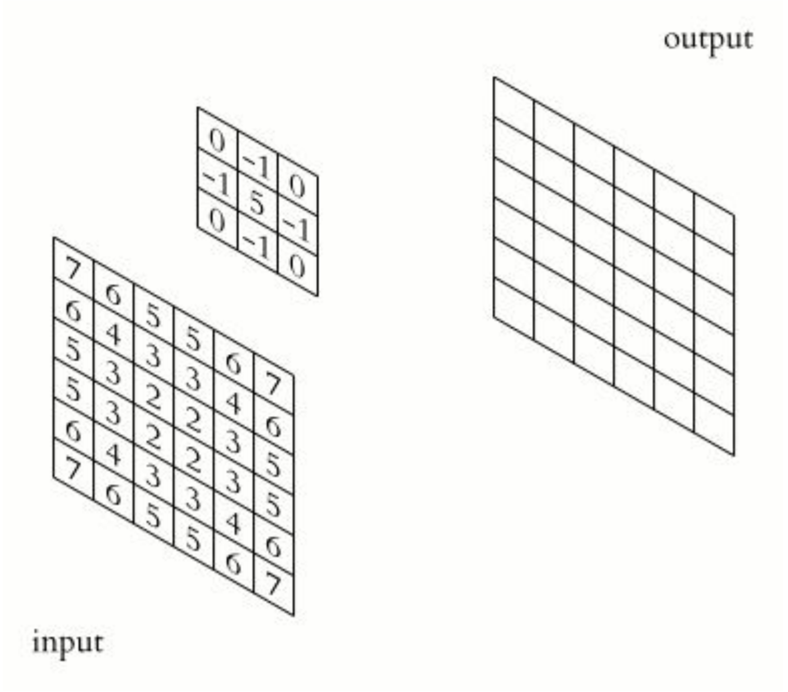
The derivation is not very straightforward, but a good exercise

Read more: https://nrsyed.com/2018/02/18/edge-detection-in-images-how-to-derive-the-sobel-operator/

Application: Gradient Based Edge Detection

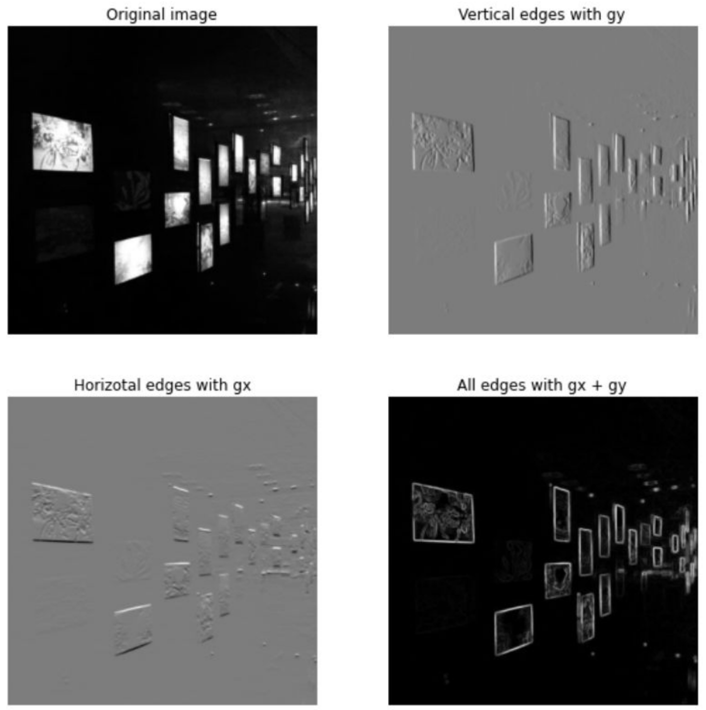
Once you have the kernel

- Do the usual spatial filtering
- 2D signal correlation
- Output image: $||\nabla \boldsymbol{f}(\mathbf{x}, \mathbf{y})||_1$
- Edges should be detected



https://commons.wikimedia.org/wiki/File:2D_Convolution_Animation.gif

Edge Detection using the Sobel Operator



Digital Signal Derivatives: 2D Case

2nd order derivative and the Laplacian

- For a function *f*(x, y) the laplacian of *f* at coordinates (x, y) is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- Using our previous definition of second derivative the second order derivatives in x and y directions can be calculated as

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

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

Digital Signal Derivatives: 2D Case

2nd order derivative and the Laplacian

- For a function *f*(x, y) the laplacian of *f* at coordinates (x, y) is defined as

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

Kernel implementation:

0	1	0
1	−4	1
0	1	0

Digital Signal Derivatives: 2D Case

2nd order derivative and the Laplacian

- Another version of the kernel can be developed by taking into account the cross derivatives as well $N_8(p)$ neighbourhood

Kernel implementation:

1	1	1
1	−8	1
1	1	1

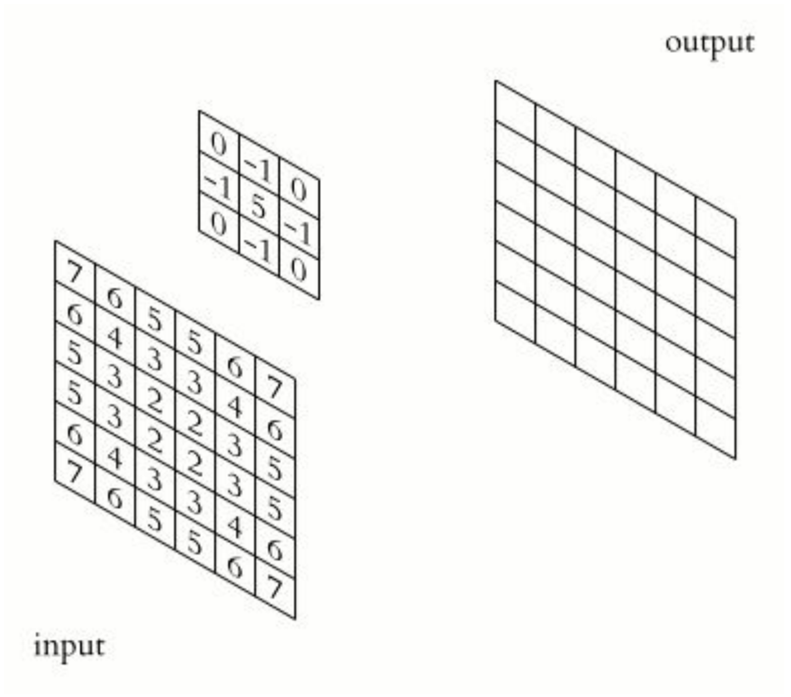
Application: Laplacian Edge Detection & Sharpening

Edge Detection

- Do the usual spatial filtering
- Output image: $\nabla^2 f(x, y)$
- Soft edges should be detected

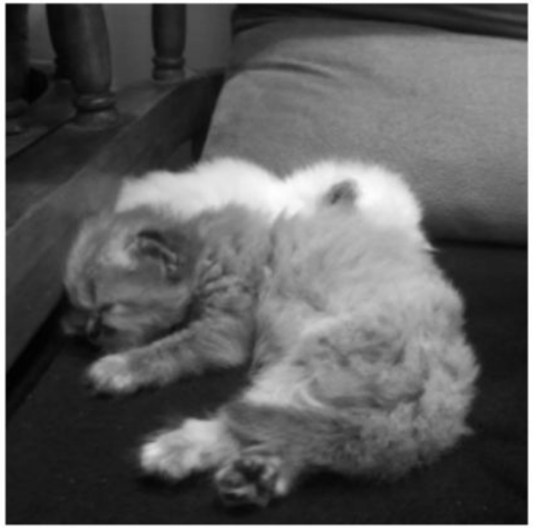
Image Sharpening

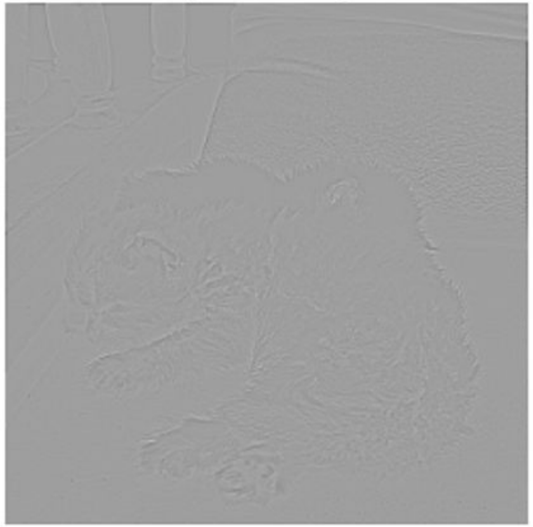
- $g(x, y) = f(x, y) + c[\nabla^2 f(x, y)]$
- c = -1 for the kernels derived
- Input image: $f(x, y)$
- Sharpened image: $g(x, y)$



https://commons.wikimedia.org/wiki/File:2D_Convolution_Animation.gif

Laplacian Edge Detection & Sharpening


Original image


Laplacian image

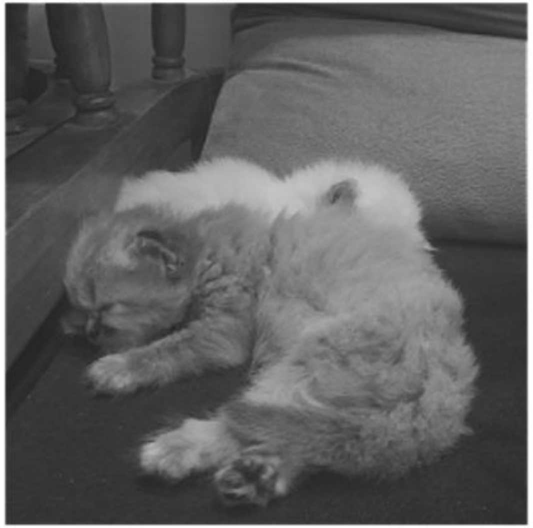

Sharpened image with laplace masking

Image Denoising Using Nonlinear Filters

Bilateral filtering

- What if we were to combine the idea of a weighted filter kernel with a better version of outlier rejection?
- What if instead of rejecting a fixed percentage , we simply reject (in a soft way) pixels whose values differ too much from the central pixel value?

Bilateral Filtering

1. In the bilateral filter, the output pixel value depends on a weighted combination of neighboring pixel values

$$\mathbf{g}(i,j) = \frac{\sum_{k,l} \mathbf{f}(k,l)w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

2. The weighting coefficient w(i; j; k; l) depends on the product of a domain kernel

$$d(i,j,k,l) = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}\right)$$

Bilateral Filtering

3. and a data-dependent range kernel

$$r(i,j,k,l) = \exp\left(-\frac{\|\mathbf{f}(i,j) - \mathbf{f}(k,l)\|^2}{2\sigma_r^2}\right)$$

4. When multiplied together, these yield the data-dependent bilateral weight function

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|\mathbf{f}(i,j) - \mathbf{f}(k,l)\|^2}{2\sigma_r^2}\right)$$

Bilateral Filtering

(a) noisy step edge input

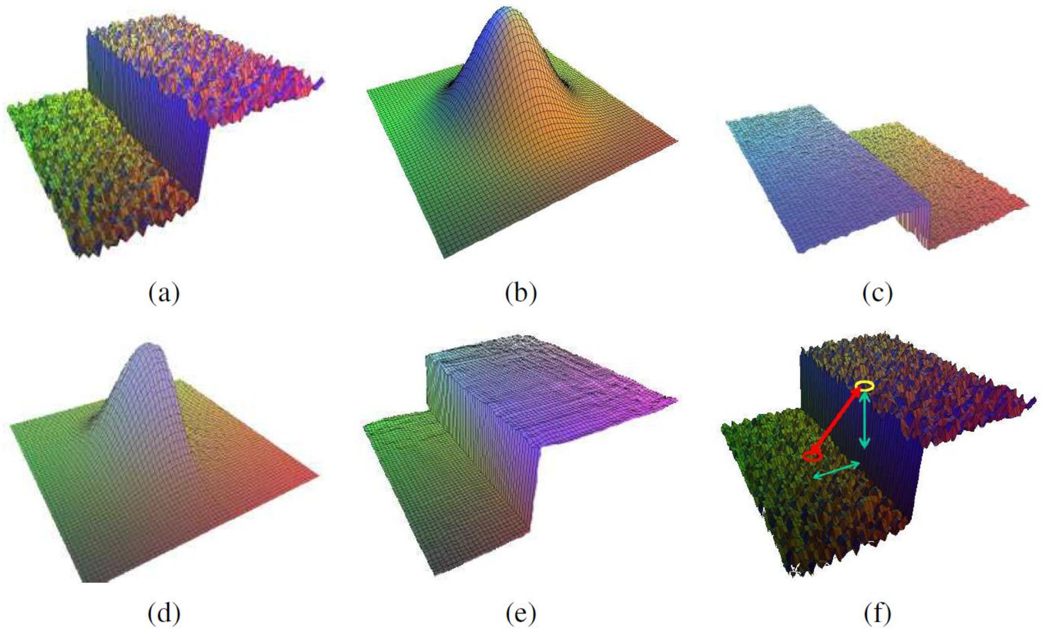
(b) domain filter (Gaussian)

(c) range filter (similarity to center pixel value)

(d) bilateral filter

(e) filtered step edge output

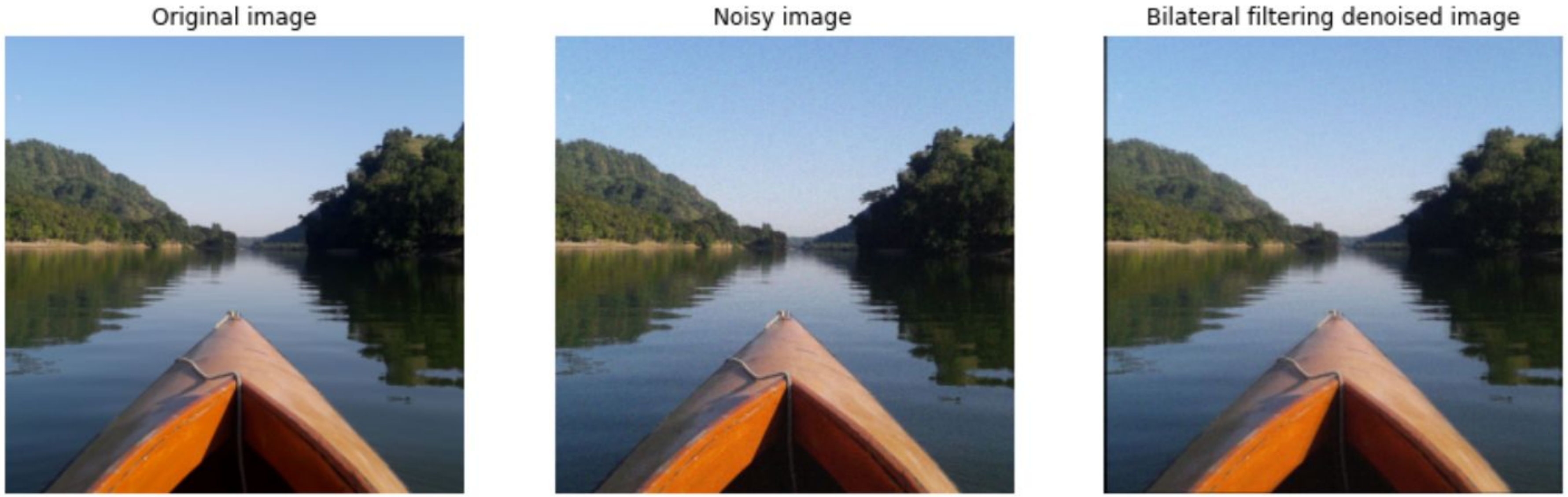
(f) 3D distance between pixels.



ACM Transactions on GraphicsVolume 21Issue 3 July 2002 pp 257–266 https://doi.org/10.1145/566654.566574

Computer Vision: Algorithms and Applications, 2nd Edition, Richard Szeliski

Bilateral Filtering



Total Variation Denoising

Total Variation Denoising

- Based on the principle that signals with excessive and possibly spurious detail have high total variation, that is, the integral of the absolute gradient of the signal is high
- According to this principle, reducing the total variation of the signal—subject to it being a close match to the original signal—removes unwanted detail whilst preserving important details such as edges.

Total Variation Denoising

1. The total-variation norm proposed by the 1992 article is and is isotropic and not differentiable. Here, ***x*** is the noisy image and ***y*** is the denoised image.

$$V(y) = \sum_{i,j} \sqrt{|y_{i+1,j} - y_{i,j}|^2 + |y_{i,j+1} - y_{i,j}|^2}$$

2. A variation that is sometimes used, since it may sometimes be easier to minimize, is an anisotropic version

$$V_{\text{aniso}}(y) = \sum_{i,j} \sqrt{|y_{i+1,j} - y_{i,j}|^2} + \sqrt{|y_{i,j+1} - y_{i,j}|^2} = \sum_{i,j} |y_{i+1,j} - y_{i,j}| + |y_{i,j+1} - y_{i,j}|$$

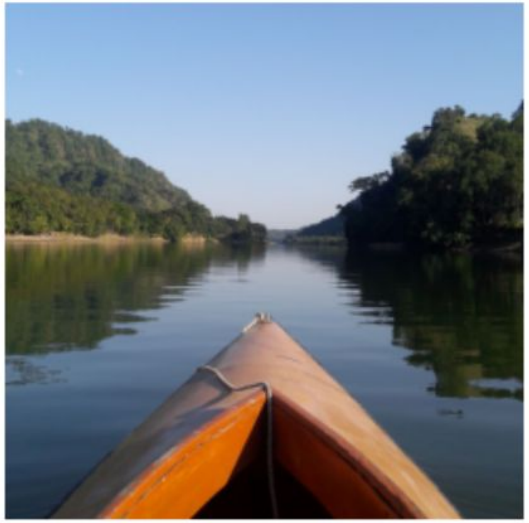
3. The standard total-variation denoising problem is of the following form, where E is the L^2 norm

$$\min_{y} [\mathrm{E}(x, y) + \lambda V(y)]$$

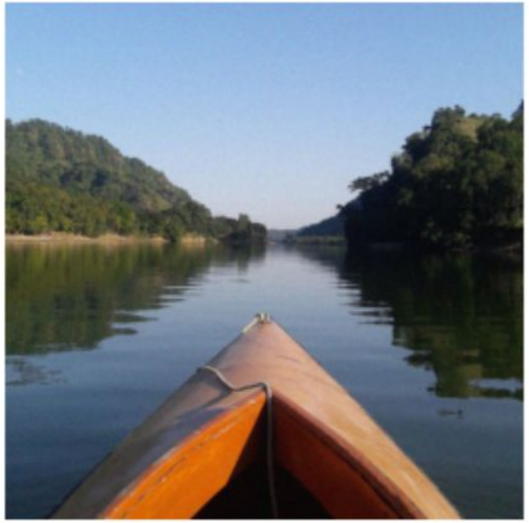
https://en.wikipedia.org/wiki/Total_variation_denoising

Total Variation Denoising

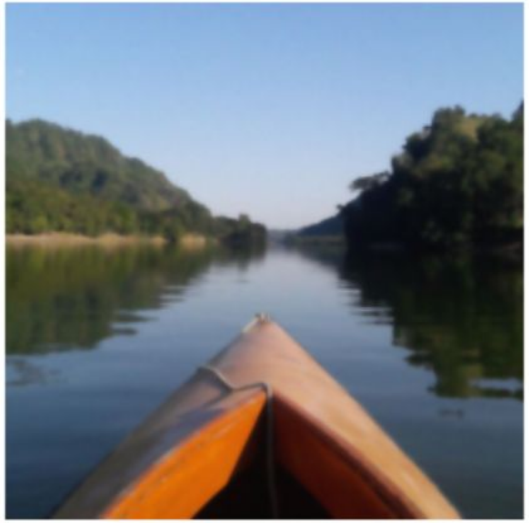
Original image



Noisy image



Total Variation Denoised image



Separable Kernels

A 2D function $G(x, y)$ is separable if it can be written as the product of two 1D functions $G_1(x)$ and $G_2(x)$; or $\mathbf{G(x, y) = G_1(x)G_2(y)}$

A kernel w of size $m \times n$ is separable if $\mathbf{w = vw^T}$

v is a vector of size m x 1

w is a vector of size n x 1

For a square kernel w of size $m \times m$ is separable if $\mathbf{w = vv^T}$

v is a vector of size m x 1

Linear Spatial Filtering: Separable Kernel Advantages

Observation: Matrix product of a column vector and a row vector is the same as the 2D convolution of the same vectors. So,

$$\boldsymbol{vw^T = v \bigstar w}$$

If we have a kernel *w* that is separable such that $\boldsymbol{w = w_1 \bigstar w_2}$, then it follows from the commutative and associative properties of convolution that

$$\boldsymbol{w \bigstar f = (w_1 \bigstar w_2) \bigstar f = (w_2 \bigstar w_1) \bigstar f}$$

$$\boldsymbol{= w_2 \bigstar (w_1 \bigstar f) = (w_1 \bigstar f) \bigstar w_2}$$

So the original convolution with 2D kernel can be turned into two different convolutions with 1D kernels if the original 2D kernel is separable.

Linear Spatial Filtering: Separable Kernel Advantages

For an image of size ***M*** x ***N*** and a kernel of size ***m*** x ***n***

- Single convolution/correlation operation with a 2D kernel
  - Requires *MNmn* multiplications and additions

  Complexity: O(mn) or $\boldsymbol{O(m^2)}$ if m = n

- Double convolution/correlation operation with two 1D kernels
  - Requires *MNm* multiplications and additions in the first part
  - Requires *MNn* multiplications and additions in the second part

  Complexity: O(m+n) or $\boldsymbol{O(m)}$ if m = n

Linear Spatial Filtering: Separable Kernel Advantages

Computational advantage of performing convolution with a separable kernel as opposed to a non-separable kernel is defined as

$$C = \frac{mn}{m + n}$$

For a kernel of size ***m*** x ***n***.

But, how do we know if our kernel is separable or not?

Is the Kernel Separable?

From linear algebra, we know that a matrix resulting from multiplying a column vector and a row vector **is always of rank 1**

Hence, to figure out if a kernel is separable or not we just need to find its rank!

We can find the rank of a matrix using the SVD theorem [Next few slides]

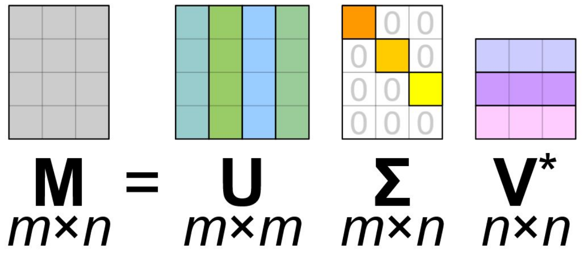
Which, by the way, is already implemented in several python packages, so we don’t need to worry about it implementing it

Can be done using NumPy in a single line of code

```python
def is_seperable(kernel):
    return np.linalg.matrix_rank(kernel) == 1
```

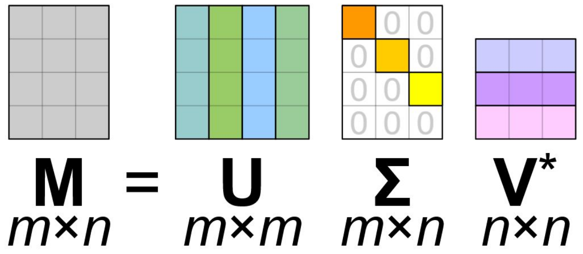
Singular Value Decomposition

Any matrix ***M*** can be decomposed into 3 matrices: ***U***, ***Σ*** and ***V***** or, ***M* = *U Σ V****.



And the total number of *non zero diagonal entries* (also called the singular values) in ***Σ*** is equal to the rank of the matrix ***M***, ***r***.

For a separable kernel ***w***, there should be only 1 non zero value in ***Σ***. In that case, for ***w* = *vw***^{T}, ***v*** is just the first column of ***U*** and ***w*** is just the first row of ***V***.



```
U, S, V = np.linalg.svd(M)
```

Separating the Separable Kernels

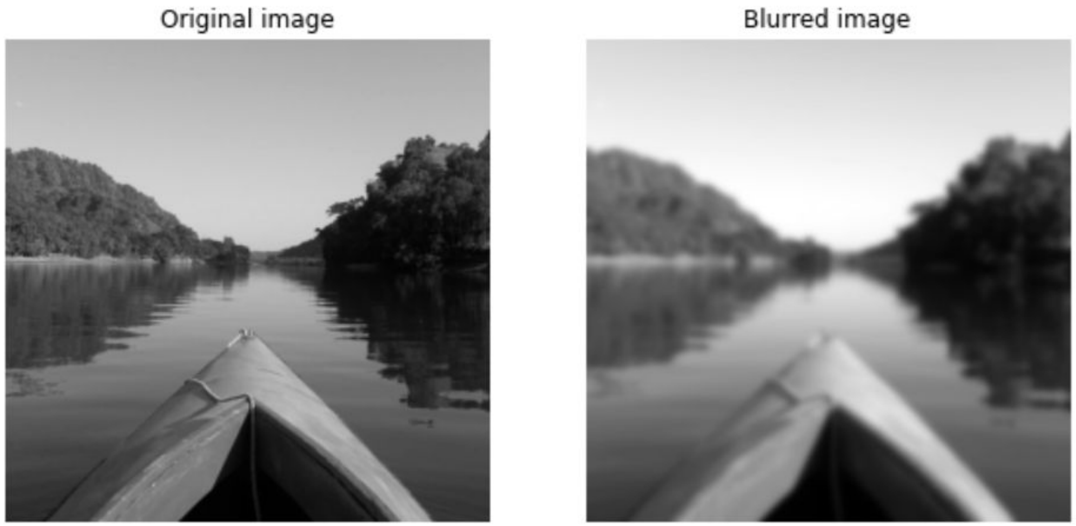
Once you have determined that the rank of a kernel matrix is 1, it is not difficult to find two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ such that $\boldsymbol{w} = \boldsymbol{v}\boldsymbol{w}^T$

1. Find any nonzero element in the kernel and let $\boldsymbol{E}$ denote its value
2. Form vectors $\boldsymbol{c}$ and $\boldsymbol{r}$ equal, respectively, to the column and row in the kernel containing the element $\boldsymbol{E}$ found in Step 1
3. Let $\boldsymbol{v} = \boldsymbol{c}$ and $\boldsymbol{w}^T = \boldsymbol{r}/\boldsymbol{E}$

Blurring vs Sharpening Filters

Image Blurring

Tends to *preserve the lower level variations* in the image (the variation in the sky or the variation in the kayak) *while discarding the higher level variations* (like the waves in the water or the pattern in the greens or the edges)

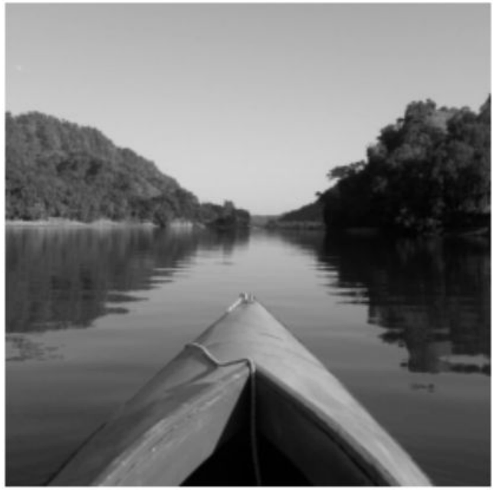


Blurring vs Sharpening Filters

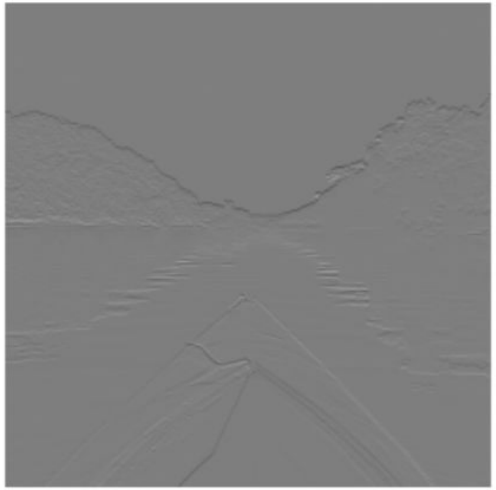
Image Sharpening

Tends to *discard the lower level variations* in the image (the variation in the sky or the variation in the kayak) *while preserving the higher level variations* (like the waves in the water or the pattern in the greens or the edges)

Original image



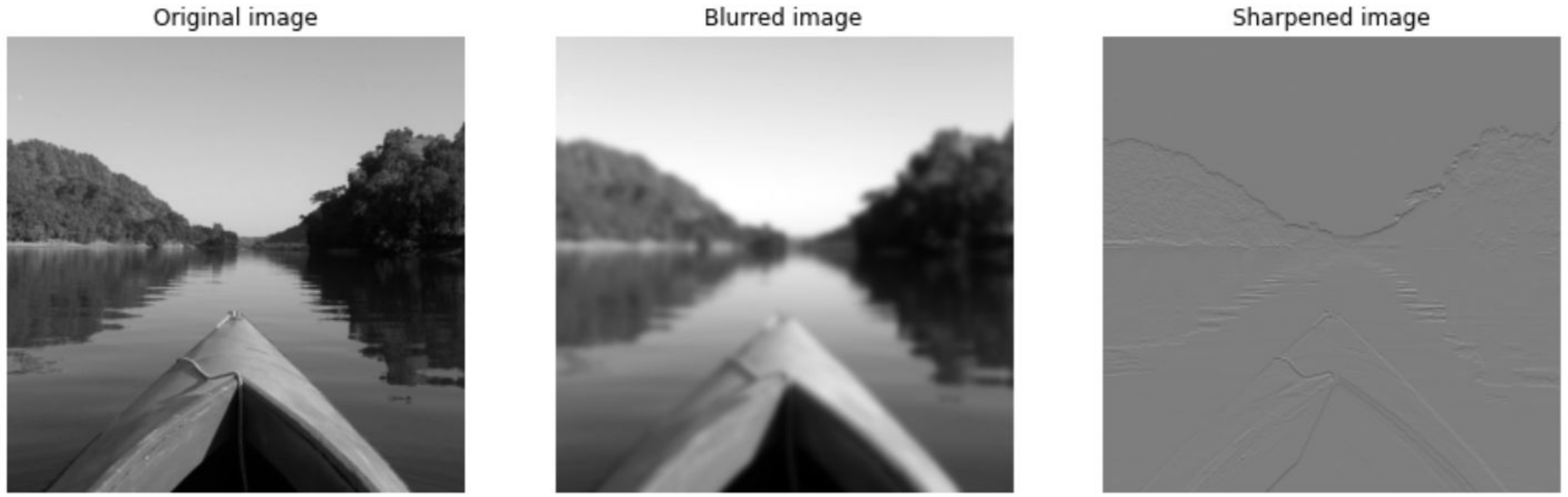
Sharpened image



Blurring vs Sharpening Filters: Frequency Intuition

Image blurring > Preserving lower variations > *Low pass* filter characteristics

Image sharpening > Preserving higher variations > *High pass* filter characteristics



Importance of Spatial Filtering

Easy to compute

Wide range of usefulness from image blurring to image sharpening

Also helpful for denoising

Deep learning applications (huge!)

Colab Tutorial

https://colab.research.google.com/drive/1n4MPV4ZhY-z0lcGb0gZTDkPFJGBjTcVf?usp=sharing

Thank you!