Chapter 3.1
Local Processing on Images

Image Processing and Computer Vision

LE Thanh Sach
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

1. **Local processing**
2. **Linear processing**
   - Correlation
   - Convolution
   - Linear Filtering
   - Popular Linear Filter
   - Convolution's Properties
   - Convolution's implementation

Sources of slides

Sources

This presentation uses figures, slides and information from the following sources:

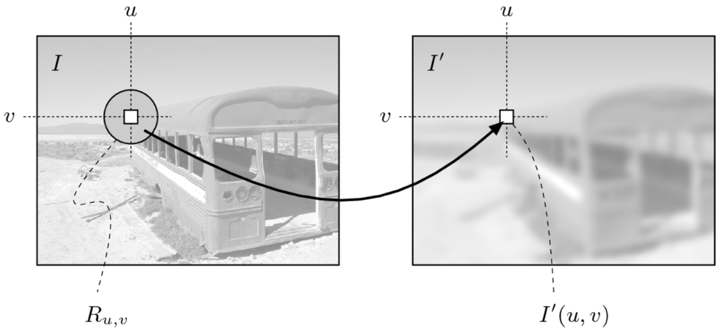
1. Rafael C. Gonzalez, Richard E. Woods, "Digital Image Processing", 2^{nd} Editions.
2. Maria Petrou and Costas Petrou, "Image Processing: The Fundamentals", 2^{nd} Editions.
3. Slides of Course "CS 4640: Image Processing Basics", from Utah University.

What is local processing?

What is local processing?

Definition

Local processing is an image operation where each pixel value $I(u, v)$ is changed by a **function** of the intensities of pixels in a **neighborhood** of the pixel (u, v).



What is local processing?

Example

- An image $I(u, v)$; a pixel (u, v) and its neighborhood of 3×3 pixels

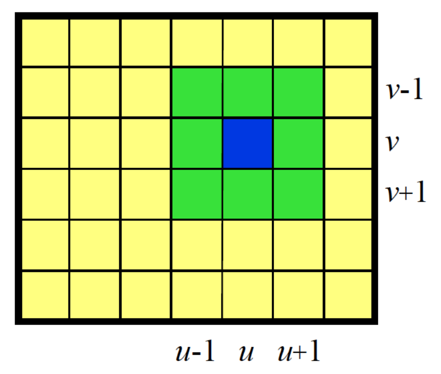


Figure: Example of neighborhood

What is local processing?

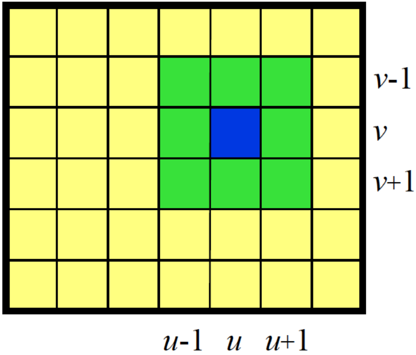
Example

Examples of some processing functions

- Linear functions
  1. Averaging function
  2. Shifting function
  3. Gaussian function
  4. Edge detecting function
- Non-linear functions
  1. Median function
  2. Min function
  3. Max function

What is local processing?

- Example of an averaging function



$$I'(u,v) = \frac{1}{9}\sum_{i=-1}^{1}\sum_{j=-1}^{1} I(u+i, v+j)$$

What is local processing?

- Example of an averaging function

Input image

Output image

- The output image is obtained by averaging the input with neighborhood of 9×9 pixels.

What is local processing?

- Example of an averaging function

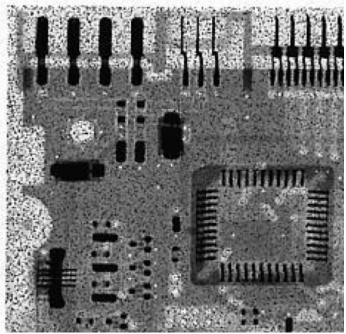
Input image

Output image
blurred, smoothed

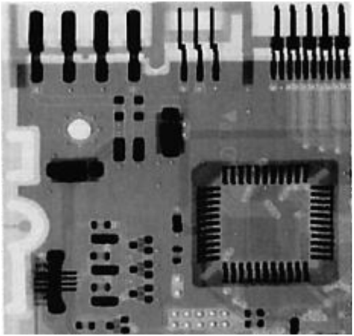
$$I'(u,v) = \frac{1}{9\times 9}\sum_{i=-4}^{4}\sum_{j=-4}^{4} I(u+i, v+j)$$

What is local processing?

- Example of a median function (a non-linear function)



Input image



Output image
Noises have been removed

- The output image is obtained by computing the median value of a set of pixels in a neighborhood of 3×3 pixels.

Linear processing function

Mean function

Consider an averaging function on square window. In general, the window can have different size of each dimension. The output of the averaging is determined by.

$$I^{'}(u,v) = \frac{1}{(2r+1)\times(2r+1)}\sum_{i=-r}^{r}\sum_{j=-r}^{r} I(u+i,v+j)$$

$I^{'}(u,v)$ can be written as

$$I^{'}(u,v) = \sum_{i=-r}^{r}\sum_{j=-r}^{r} I(u+i,v+j).H_{corr}(i,j)$$

Mean function

1. H_{corr} is a matrix of size $(2r+1) \times (2r+1)$
2. $H_{corr} = \frac{1}{(2r+1)\times(2r+1)} M_{ones}$
   and,
3. M_{ones} : is an matrix of size $(2r+1) \times (2r+1)$ containing value 1 for all elements.

Example

Matrix for averaging pixels in a neighborhood of size 5×5, i.e., $r = 2$.

$$H_{corr} = \frac{1}{5 \times 5} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

From averaging function to others

A way to construct other linear processing functions

If one changes H_{corr} to other kinds of matrix, he obtains other linear function.

Example

Edge detecting function (Sobel)

$$H_{corr} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad H_{corr} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Shifting function

$$H_{corr} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Correlation

Definition

Input data:

1. Input image, $I(u,v)$
2. Matrix $H_{corr}(i,j)$ of size $(2r+1) \times (2r+1)$. In general, the size on two dimensions maybe different.

Correlation is defined as follows:

$$I'_{corr}(u,v) = \sum_{i=-r}^{r} \sum_{j=-r}^{r} I(u+i, v+j).H_{corr}(i,j)$$

Correlation: How does it works?

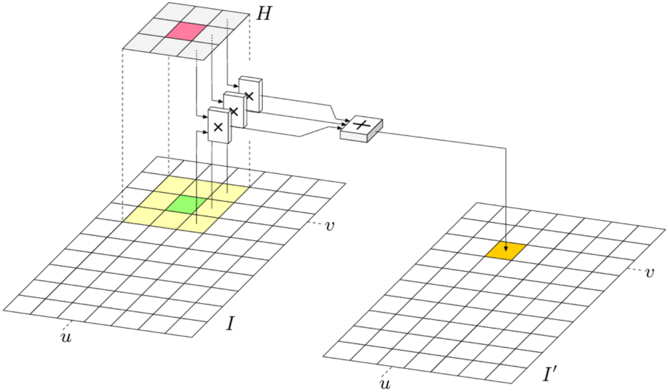


Figure: Method for computing the correlation for one pixel

Correlation: How does it works?

A computation process

For each pixel (u, v) on the output image, do:

1. **Place** matrix H_{corr} centered at the corresponding pixel, i.e., pixel (u, v), on the input image
2. **Multiply** coefficients in matrix H_{corr} with the underlying pixels on the input image.
3. **Compute** the sum of all the resulting products in the previous step.
4. **Assign** the sum to the $I^{'}(u, v)$.

Convolution

Definition

Input data:

1. Input image, $I(u, v)$
2. Matrix H_{conv} of size $(2r+1) \times (2r+1)$. In general, the size on two dimensions maybe different.

Convolution is defined as follows:

$$I'_{conv}(u, v) = \sum_{i=-r}^{r} \sum_{j=-r}^{r} I(u-i, v-j).H_{conv}(i, j)$$

Convolution

Natation

- Operator * is used to denote the convolution between image I and matrix H_{conv}
- That is

$$
\begin{aligned}
I'_{conv}(u,v) &= I * H_{conv} \\
&= \sum_{i=-r}^{r}\sum_{j=-r}^{r} I(u-i, v-j).H_{conv}(i,j)
\end{aligned}
$$

Convolution

Attention!

- When I is an gray image, both of I and H_{conv} are matrices.
- However, $I * H_{conv}$ is convolution between I and H_{conv}, **instead of matrix multiplication**!

Convolution and Correlation

Mathematics

$$I'_{conv}(u,v) = \sum_{i=-r}^{r}\sum_{j=-r}^{r} I(u-i, v-j).H_{conv}(i,j)$$

$$I'_{corr}(u,v) = \sum_{i=-r}^{r}\sum_{j=-r}^{r} I(u+i, v+j).H_{corr}(i,j)$$

In mathematics, convolution and correlation are different in the **sign** of i and j inside of $I(u+i, v+j)$ and $I(u-i, v-j)$

Convolution and Correlation

Convolution to Correlation

- Let $s = -i$ and $t = -j$
- We have

$$I'_{conv}(u, v) = \sum_{i=-r}^{r} \sum_{j=-r}^{r} I(u - i, v - j).H_{conv}(i, j)$$
$$= \sum_{i=-r}^{r} \sum_{j=-r}^{r} I(u + s, v + t).H_{conv}(-s, -t)$$

$H_{conv}(-s, -t)$ from $H_{conv}(i, j)$

$H_{conv}(-s, -t)$ can be obtained from $H_{conv}(i, j)$ by either

- **Flipping** $H_{conv}(i, j)$ on x and then on y axis
- **Rotating** $H_{conv}(i, j)$ around its center 180^0

Convolution and Correlation

Example

Demonstration of rotation and flipping.

$$H = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad H_{flipped\_x} = \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{bmatrix}$$

$H_{flipped\_x}$ is obtained from H by flipping H on x-axis.
After flipping $H_{flipped\_x}$ around y axis

$$H_{flipped\_xy} = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

$H_{flipped\_xy}$ can obtained from H by rotating H 180^0 around H's center.

Convolution and Correlation

Correlation to Convolution

- Let $s = -i$ and $t = -j$
- We have

$$I'_{corr}(u,v) = \sum_{i=-r}^{r}\sum_{j=-r}^{r} I(u+i, v+j).H_{corr}(i,j)$$
$$= \sum_{i=-r}^{r}\sum_{j=-r}^{r} I(u-s, v-t).H_{corr}(-s,-t)$$

$H_{corr}(-s,-t)$ from $H_{corr}(i,j)$

$H_{corr}(-s,-t)$ can be obtained from $H_{corr}(i,j)$ by either

- **Flipping** $H_{corr}(i,j)$ on x and then on y axis
- **Rotating** $H_{corr}(i,j)$ around its center 180^0

Convolution-Correlation Conversion

Relationship

1. Convolution can be computed by correlation and vice versa.
2. For example, convolution can be computed by correlation by: first, (a) rotating the matrix 180^0 and then (b) computing the correlation between the rotated matrix with the input image.

Convolution: How does it works?

A Computation process

1. **Rotate** matrix H_{conv} around its center 180^0 to obtain H_{corr}

For each pixel (u, v) on the output image, do:

1. **Place** matrix H_{corr} centered at the corresponding pixel, i.e., pixel (u, v), on the input image
2. **Multiply** coefficients in matrix H_{corr} with the underlying pixels on the input image.
3. **Compute** the sum of all the resulting products in the previous step.
4. **Assign** the sum to the $I^{'}(u, v)$.

MATLAB's function

Example

MATLAB's functions supports correlation and convolution

1. **corr2**
2. **xcorr2**
3. **conv2**
4. **filter2**
5. **imfilter**

MATLAB's function supports creating special matrix

1. **fspecial**

Linear Filtering

Definition

Linear filtering is a process of applying the **convolution or the correlation** between an matrix H to input image $I(u,v)$.

Model of a filter system

Figure: Filter image $I(u,v)$ with matrix H to obtain $I^{'}(u,v)$

$$I^{'}(u,v) = I(u,v) * H(i,j)$$

Filter's kernel

Definition

In filtering, matrix H is called the filter's kernel.

Other names of H

- Filter's kernel
- Window
- Mask
- Template
- Matrix
- Local region

Popular Linear Filter: Mean filter

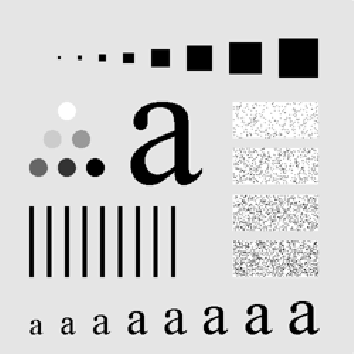
Example

Mean filter's kernel

- General case:

$$H_{corr} = \frac{1}{(2r+1)^2} \times \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & \dots & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix}_{(2r+1)\times(2r+1)}$$

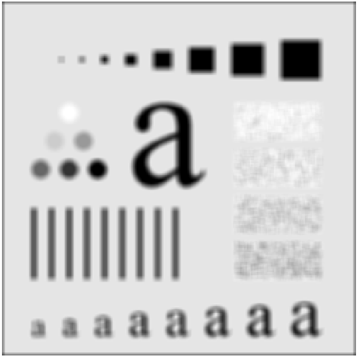
Popular Linear Filter: Mean filter



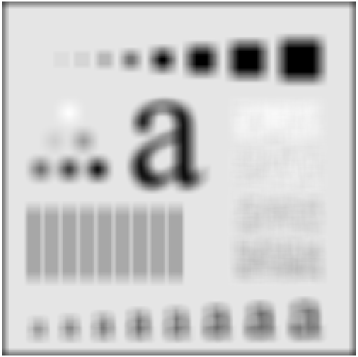
Original image

Filtered with H's size: 3×3

Popular Linear Filter: Mean filter



Filtered with H's size: 5×5



Filtered with H's size: 11×11

Popular Linear Filter: Gaussian filter

2D-Gaussian Function

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} exp(-\frac{x^2 + y^2}{2\sigma^2})$$

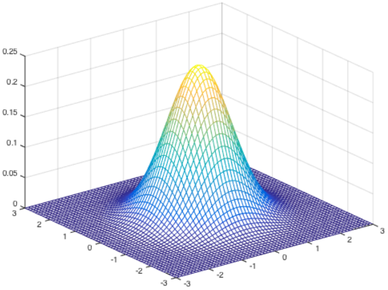


Figure: 2D-Gaussian Function

Popular Linear Filter: Gaussian filter

Example

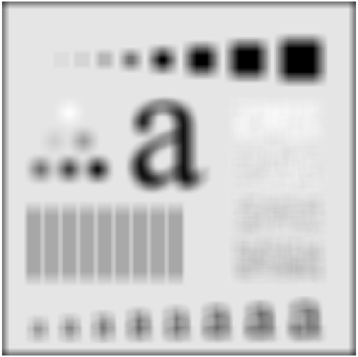
- H's size is 3×3
- $\sigma = 0.5$

$$H = \begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix}$$

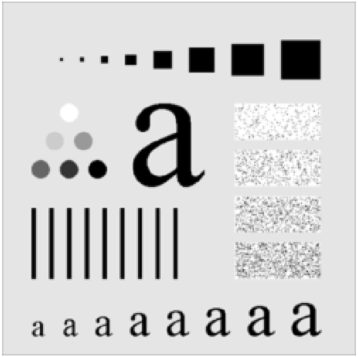
- H's size is 5×5
- $\sigma = 0.5$

$$H = \begin{bmatrix} 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0002 & 0.0837 & 0.6187 & 0.0837 & 0.0002 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \end{bmatrix}$$

Popular Linear Filter: Gaussian filter



Filtered with Mean filter
H's size: 11×11



Filtered with Gaussian filter
H's size: 11×11
$\sigma = 0.5$

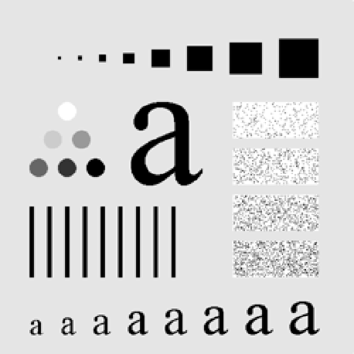
Popular Linear Filter: Shifting filter

Example

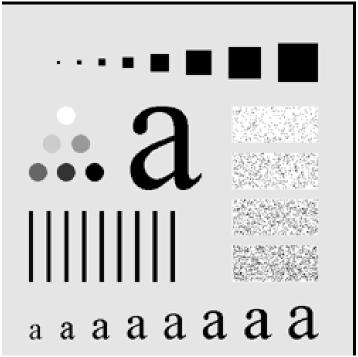
- In order to shift pixel (u, v) to $(u-2, v+2)$, use the following kernel

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Popular Linear Filter: Shifting filter



Original image



Shifted with $d = [-2, 2]$

Convolution's Properties

Commutativity:

$$I * H = H * I$$

Meaning

1. This means that we can think of the image as the kernel and the kernel as the image and get the same result.
2. In other words, we can leave the image fixed and slide the kernel or leave the kernel fixed and slide the image.

Convolution's Properties

Associativity:

$$(I * H_1) * H_2 = I * (H_1 * H_2)$$

Meaning

1. This means that we can apply H_1 to I followed by H_2, or we can convolve the kernel $H_2 * H_1$ and then apply the resulting kernel to I.

Convolution's Properties

Linearity:

$$(\alpha.I) * H = \alpha.(I * H)$$
$$(I_1 + I_2) * H = I_1 * H + I_2 * H$$

Meaning

1. This means that we can multiply an image by a constant before or after convolution, and we can add two images before or after convolution and get the same results.

Convolution's Properties

Shift-Invariance

Let S be an operator that shifts an image I:

$$S(I)(u, v) = I(u + a, v + b)$$

Then,

$$S(I * H) = S(I) * H$$

Meaning

1. This means that we can convolve I and H and then shift the result, or we can shift I and then convolve it with H.

Convolution's Properties

Separability

A kernel H is called separable if it can be broken down into the convolution of two kernels:

$$H = H_1 * H_2$$

More generally, we might have:

$$H = H_1 * H_2 * ... * H_n$$

Convolution's Properties

Example

$$H_x = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad H_y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Then,

$$H_x = H_x * H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Processing at the boundary of image

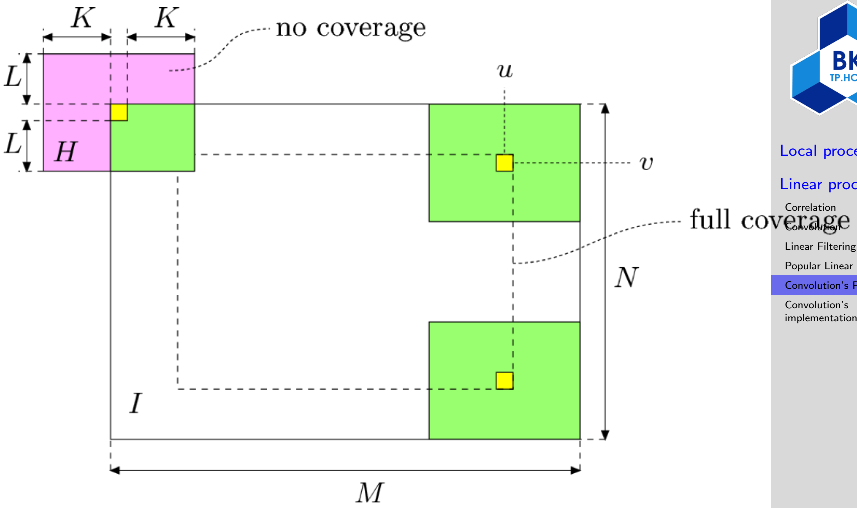


Figure: Without any special consideration, the processing is invalid at boundary pixels

Processing at the boundary of image

Methods for processing boundary pixels

1. **Cropping**:do not process boundary pixels. Just obtain a smaller output image by cropping the output image.
2. **Padding**: pad a band of pixels (with zeros) to the boundary of input image. Perform the processing and the crop to get the output image.
3. **Extending**: copy pixels on the boundary to outside to get a new image. Perform the processing and the crop to get the output image.
4. **Wrapping**: reflect pixels on the boundary to outside to get a new image. Perform the processing and the crop to get the output image.

Convolution's implementation

Methods for implementing convolution and correlation

1. **In space domain**:
   - Use sliding widow technique
   - Use speed-up methods for special cases
2. **In frequency domain**: will be presented in next chapter

Convolution's implementation

Sliding window technique for correlation

For each pixel (u, v) on the output image, do:

1. **Place** matrix H_{corr} centered at the corresponding pixel, i.e., pixel (u, v), on the input image
2. **Multiply** coefficients in matrix H_{corr} with the underlying pixels on the input image.
3. **Compute** the sum of all the resulting products in the previous step.
4. **Assign** the sum to the $I^{'}(u, v)$.

Convolution can be computed by rotating the kernel 180^0 followed by the above algorithm.

Convolution's complexity

Computational Complexity of sliding window technique

- Input image I has size $N \times M$
- Kernel's size is $(2r+1) \times (2r+1)$
- Then, the number of operations is directly proportional to: $MN[(2r+1)^2 + (2r+1)^2 - 1]$.
- **The computational complexity is $O(MNr^2)$**

Attention!

- The cost for computing convolution and correlation is directly proportional to the kernel's size!
- The filtering process will be slower if the kernel's size is bigger.
- The computational cost of the implementation in frequency domain is independent with the kernel's size.

Convolution's complexity

Example

To shift an image I to left 10 pixels. We can apply the following methods:

1. **Method 1:** Filter I with a shifting kernel of size 21×21.
2. **Method 2:** Apply 10 times shifting kernel of size 3.

Which method can result better computation cost?

Answer

Number of operations for each method is proportional to:

1. **Method 1:** $MN \times (21^2) = 441MN$
2. **Method 2:** $10 \times MN \times (3^2) = 90MN$

Method 2 is better than Method 1!

Convolution's complexity

Because, we have

Associativity:

$$(I * H_1) * H_2 = I * (H_1 * H_2)$$

So, we can save the computational cost by using separability

If we can separate a kernel H into two smaller kernels $H = H_1 * H_2$, then it will often be cheaper to apply H_1 followed by H_2, rather than H.

Convolution's complexity

Example

Kernel H can be decomposed into $H = H_1 * H_2$, as follows:

$$H_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad H_2 = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$H = H_1 * H_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Using associativity, we can filter image I with H by either

1. **Method 1:** $I' = I * H$
2. **Method 2:** $I' = (I * H_1) * H_2$

Which method is better?

Convolution's complexity

Answer

Number of operations for each method is proportional to:

1. **Method 1:** $MN \times (3^2) = 9MN$
2. **Method 2:** $(MN \times 3) + (MN \times 3) = 6MN$

Method 2 is better than Method 1!

Convolution's implementation

Special cases

1. **Separability:** In the case that kernel filter H can be separated into smaller kernels. Consecutively apply smaller kernels to the input image can reduce the computation time, as shown in previous slide.
2. **Box Filtering:** Kernel's coefficients of box filter are equal (value 1). So, we can use **integral image** to speed up.

Convolution's implementation: Integral image

CASE 1: 1D-array

- Input data: array $A[i]$, for $i \in [1, n]$
  - The first element, $A[0]$, is not used

$A[i]$		3	8	2	6	9	7	1
	0	1	2	3	4	5	6	7

- Output data: **integral array**, denoted as $C[i]$
- The integral array is computed as follows
  1. $C[0] = 0$
  2. $C[i] = C[i-1] + A[i]$, for $i \in [1, n]$.

$C[i]$	0	3	11	13	19	28	35	36
	0	1	2	3	4	5	6	7

Convolution's implementation: Integral image

CASE 1: 1D-array

1D Box filter's response

The sum of elements in any window, occupying from $A[i]$ to $A[j]$, can be computed fast by: $C[j] - C[i-1]$

$A[i]$		3	8	2	6	9	7	1
	0	1	2	3	4	5	6	7
$C[i]$	0	3	11	13	19	28	35	36
	0	1	2	3	4	5	6	7

Example

1. $\sum_{i=1}^{4} A[i] = C[4] - C[0] = 19 - 0 = 19$
2. $\sum_{i=2}^{6} A[i] = C[6] - C[1] = 35 - 3 = 32$
3. $\sum_{i=3}^{6} A[i] = C[6] - C[2] = 35 - 11 = 24$

Convolution's implementation: Integral image

CASE 2: 2D-array

- Input data: Image $I(u,v)$, for $u,v \in [1,n]$
  - The first row and the first column are not used
- Output data: **integral image** $S(u,v)$. It is defined as follows.
  1. The first row and the first column contains zeros, i.e.,

$$S(0,i) = S(j,0) = 0; i,j \in [0,n]$$

  2.

$$S(u,v) = \sum_{i=1}^{u} \sum_{j=1}^{v} I(i,j)$$

Convolution's implementation: Integral image

CASE 2: 2D-array

A method for computing $S(u, v)$

1. for each element in the first row and the first column in $S(u, v)$, assign zero to it.
2. for each remaining element at (u, v): $S(u, v) = S(u-1, v) + S(u, v-1) - S(u-1, v-1) + I(u, v)$

Convolution's implementation: Integral image

CASE 2: 2D-array

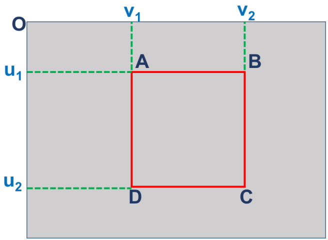


Image I(u,v)

The Way to compute the filter's response

1. Let $A, B, C, andD$ are the sum of all pixels in rectangle from O to $A, B, C, andD$ respectively.
2. The sum of pixels inside of rectangle $ABCD$ is $(C - B - D + A)$

Convolution's implementation: Integral image

CASE 2: 2D-array

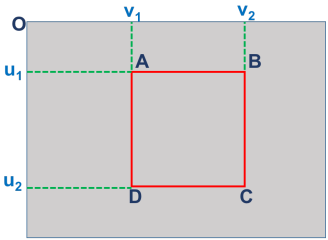


Image I(u,v)

The Way to compute the filter's response

$$
\begin{aligned}
I^{'}(u_2, v_2) &= C - B - D + A \\
&= S(u_2, v_2) - S(u_1, v_2) - S(u_2, v_1) + S(u_1, v_1)
\end{aligned}
$$