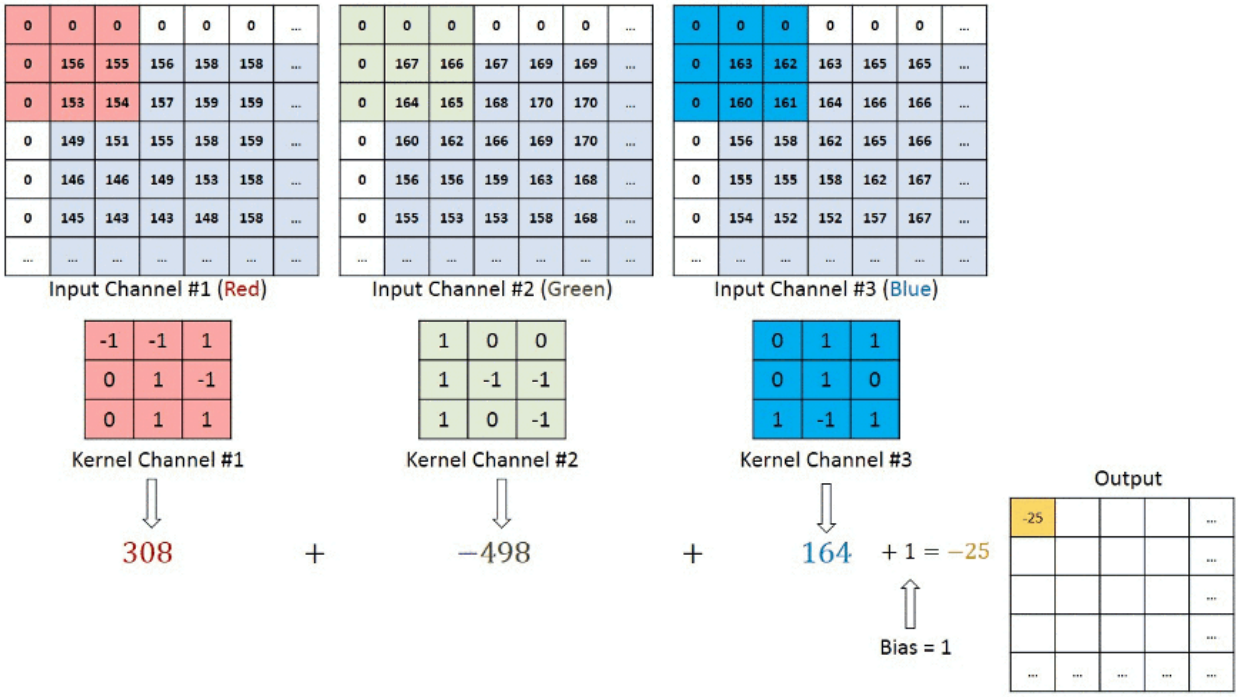
Image Processing for Computer Vision
Session 9

Low Pass Filter



Pylessons

Topics

- Low Pass filters
- Image Blurring

Types of Filters

Filters can generally be categorized into two main types based on their effect on image frequency components:

a) Low-pass Filters (Smoothing Filters)

- suppress high-frequency components (rapid changes like edges or noise)
- pass the low-frequency components of the image (i.e., slow changes in pixel values)
- blur the image, reducing noise and detail

Purpose: Noise reduction, smoothing, and blurring
Example: Box Filter (Mean Filter), Gaussian Filter, Median Filter, Bilateral Filter

b) High-pass Filters

- emphasize edges and fine details
- suppress the low-frequency components of an image
- preserve or enhance high-frequency components like edges, texture, or noise.

Purpose: Edge detection, enhancing fine details, sharpening.
Example: Sobel Filter, Laplacian Filter

Common Low-pass Filters:

1. Box Filter (Mean Filter): A simple average filter where each pixel is replaced by the mean of its neighbors. It removes noise but also blurs important details like edges.

Formula for 3x3 mean filter:

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

2. Gaussian Filter: A more sophisticated blur that uses a Gaussian function to weight neighboring pixels. Closer pixels have higher weights, and farther pixels contribute less to the output, resulting in a natural blur. Gaussian blur is often used in pre-processing to remove noise while preserving edges better than a box filter.

1/16

1	2	1
2	4	2
1	2	1

1/273

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

1/1003

0	0	1	2	1	0	0
0	3	13	22	13	3	0
1	13	59	97	59	13	1
2	22	97	159	97	22	2
1	13	59	97	59	13	1
0	3	13	22	13	3	0
0	0	1	2	1	0	0

collected

3. Median Filter: This filter takes the median of all the pixels under the kernel area and the central element is replaced with this median value.

- Central element is not newly calculated
- the central element is always replaced by some pixel value from the neighborhood
- Highly effective against salt-and-pepper noise in an image

4. Bilateral Filter: This filter smooths images while preserving edges by considering both the spatial distance between pixels and the intensity difference.

- It is especially useful in edge-aware smoothing.
- The **bilateral filter** preserves edges while smoothing areas where intensity values are similar.
- The **kernel** is computed dynamically for each pixel, so there is no fixed kernel matrix to extract like in linear filters.

Filtering documentation: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

Syntax:

cv.bilateralFilter(inp_img, d, sigmaColor, sigmaSpace)

d: Diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from sigmaSpace.

sigmaColor: applies filter based on the intensity difference or color difference. A larger value of the parameter means that farther colors within the pixel neighborhood.

sigmaSpaceFilter: applies filter based on pixel to pixel distance spatial distance. sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough. When d>0, it specifies the neighborhood size regardless of sigmaSpace. Otherwise, d is proportional to sigmaSpace.

***Filter size*:** Large filters (d > 5) are very slow, so it is recommended to use d=5 for real-time applications, and perhaps d=9 for offline applications that need heavy noise filtering.

5x5 Neighborhood:
```
[[180  176   4     171  52]
 [ 42   164   197  153  47]
 [170  13    20   6     17]
 [206  101   39   86    93]
 [205  55    75   81    27]]
```

Corresponding 5x5 Bilateral Kernel:
```
[[0.03913798  0.0410542   0.03910668  0.04045917  0.03901293]
 [0.04037025  0.03912232  0.04088214  0.03936563  0.04084944]
 [0.04034604  0.03921633  0.0390754   0.04053206  0.0394129 ]
 [0.03979309  0.04032184  0.03972152  0.03997656  0.03981697]
 [0.04014481  0.04098037  0.04089031  0.04039448  0.04001655]]
```

After applying the kernel
```
[[7.04483552  7.22553979  0.15642671  6.91851744  2.02867256]
 [1.69555067  6.41606098  8.05378104  6.02294215  1.9199239 ]
 [6.85882665  0.50981228  0.78150809  0.24319235  0.67001934]
 [8.19737611  4.07250571  1.54913947  3.43798397  3.70297829]
 [8.22968654  2.25392047  3.06677359  3.27195316  1.08044697]]
```