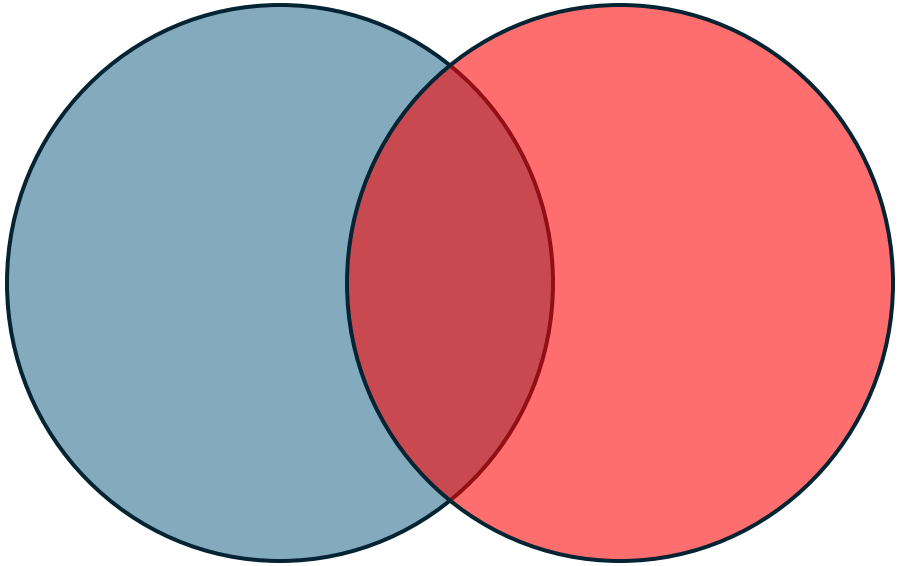
Lecture 2
Operations on Images

ECE 1390/2390

Learning Objectives:
- Mathematical operations
- Bitwise/Logical operations
- Dealing with overflow
- Masking and thresholding



An image is a matrix of information (tensor for color images)

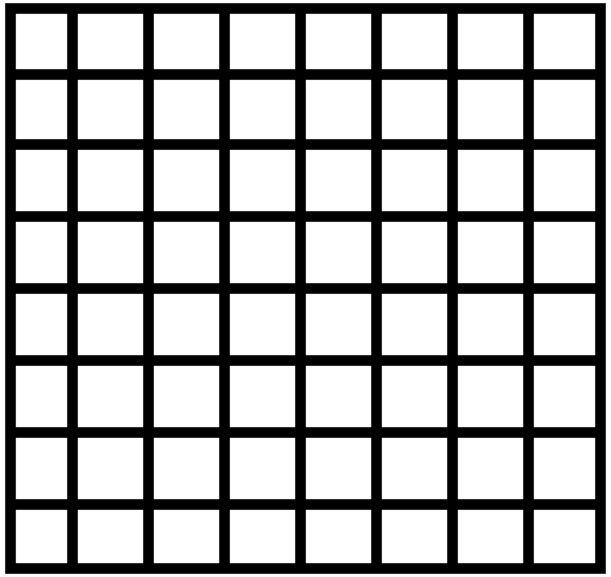
Math based operations

$$Y = a * X$$

$$Y = fn(X)$$

$$Y = H * X * H'$$

X



Morphological operations

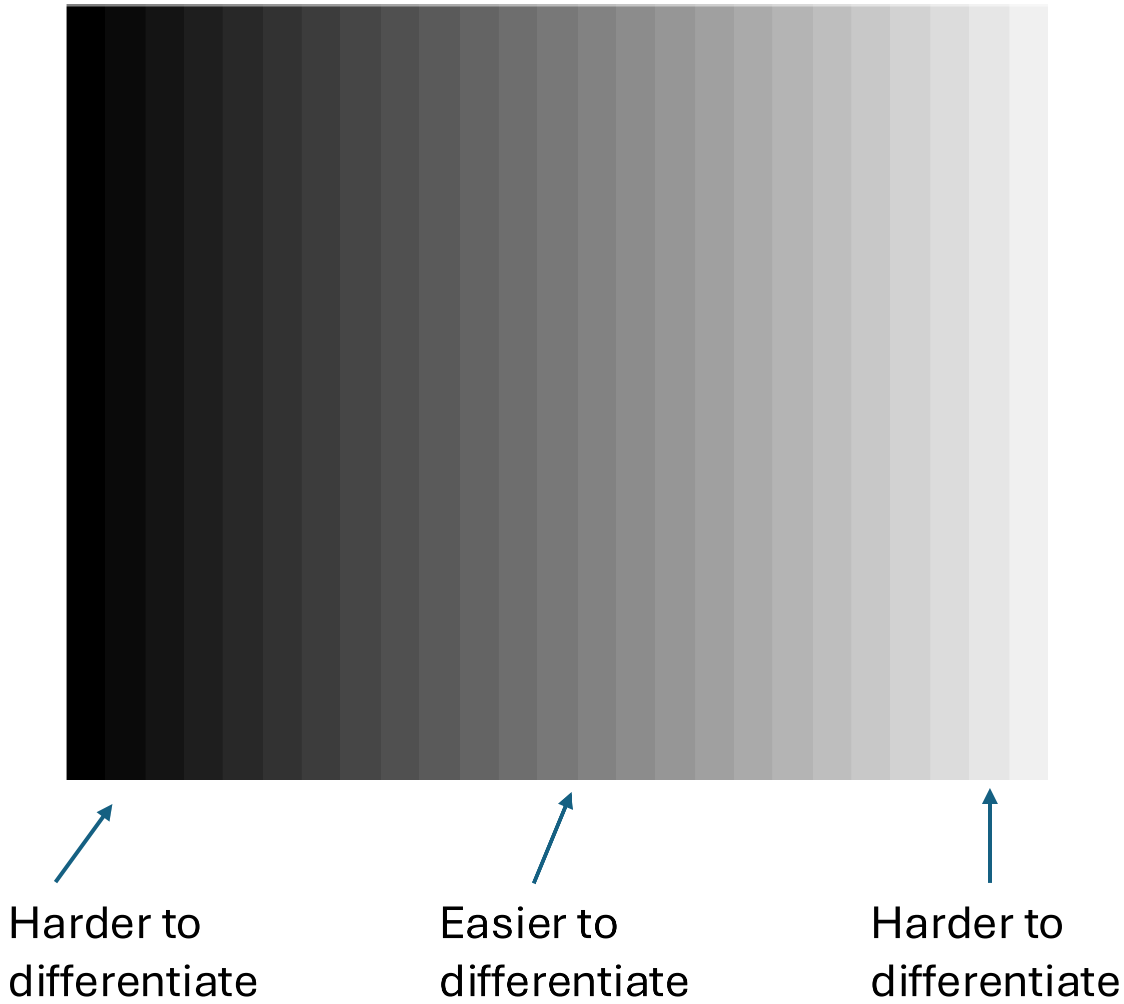
E.g. Dilate/Erode

```
For (i=0; i<n; i++)
        do something at {i}
end
```

Brightness

$$Y = X + a$$

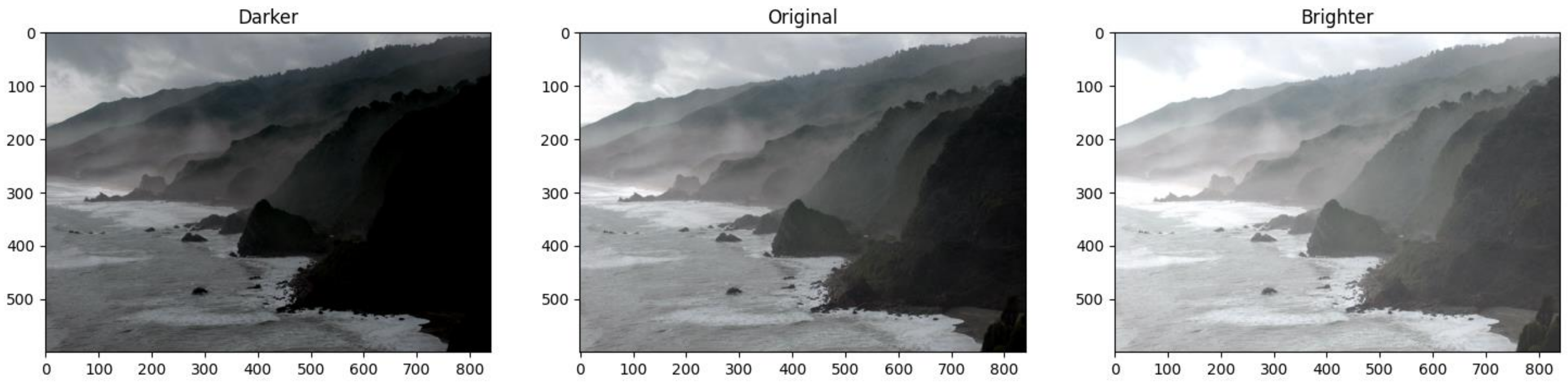
Changing the brightness of an image means adding/subtracting a scalar from the image



Gamma correction (and similar) designed to adjust intensity → brightness (next lecture)

From Lecture1/04_Image_Enhancement.ipnb

```
matrix = np.ones(img_rgb.shape, dtype="uint8") * 50

img_rgb_brighter = cv2.add(img_rgb, matrix)

img_rgb_darker = cv2.subtract(img_rgb, matrix)
```



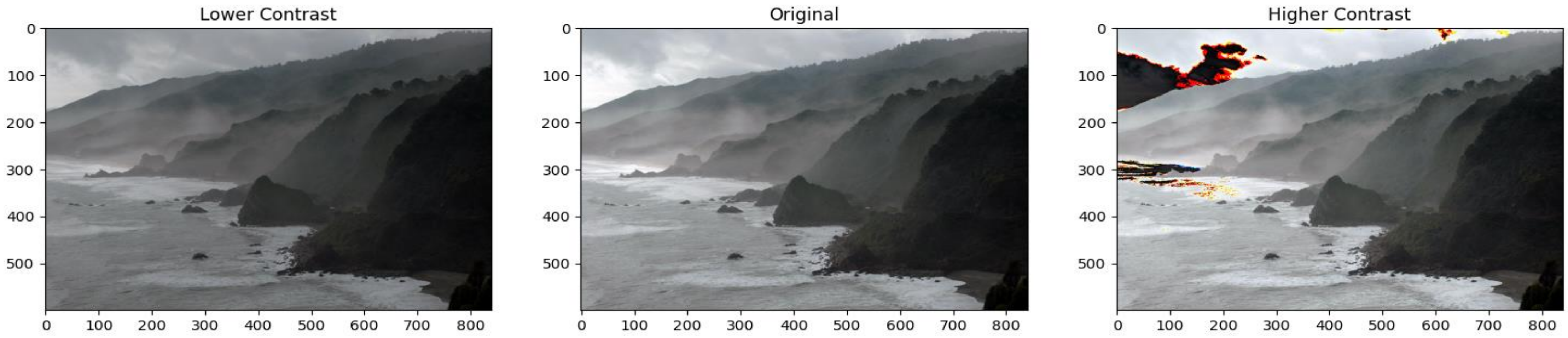
Contrast

$$Y = a * X$$

Contrast is the range of values in the image. Multiplying by a scalar changes the contrast

From Lecture1/04_Image_Enhancement.ipnb

```
matrix_low_contrast = np.ones(img_rgb.shape) * 0.8
matrix_high_contast = np.ones(img_rgb.shape) * 1.2

img_rgb_darker = np.uint8(cv2.multiply(np.float64(img_rgb), matrix_low_contrast))

img_rgb_brighter = np.uint8(cv2.multiply(np.float64(img_rgb), matrix_high_contast))
```



Clipping

Python:

```
[1]: import numpy as np
     import cv2

[2]: a=np.array(250,dtype=np.uint8)
     b=np.array(10,dtype=np.uint8)

[3]: # In openCV
     cv2.add(a,b)

[3]: array([[255]], dtype=uint8)

[4]: # Directly as NP array
     a+b

[4]: 4

[5]: a+10

[5]: 260
```

OpenCV clips at 255

numpy rolls over (260 -> 255, 0, 1, 2, 3, 4)

np + scalar => converts type

Clipping

Python:

a = [100 150 200 250] → convert to uint16
b = [10 10. 10. 10] → convert to uint16

a + b = [110 160 210 260] (uint16)

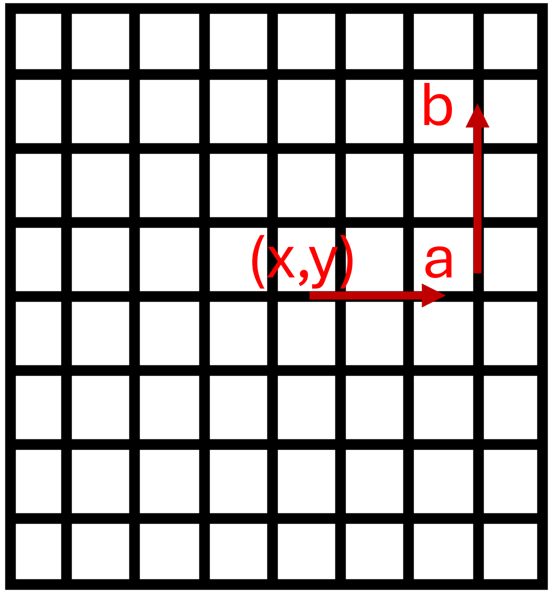
Normalize

= [110 160 210 260] * 255 /260

= [107 157 206 255]

Point Spread Function

General



$$h(x, a, y, b)$$

Operation varies depending on which position (x,y) in the image

Shift invariant

$$h(a - x, b - y)$$

Operation is independent of the position (x,y) in the image

Windowed FIR Filters

$$\mathcal{F}(f) * H(f)$$

Multiplication in the Fourier domain

$$\mathrm{f}(t) \otimes h(t)$$

Convolution in the Time domain

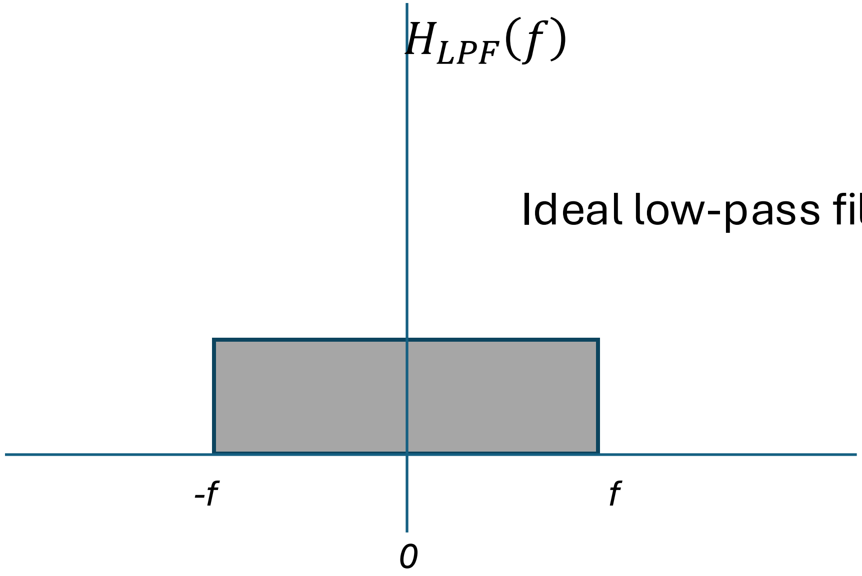
Windowed FIR Filters

$$\mathcal{F}(f) * H(f)$$

Multiplication in the
Fourier domain

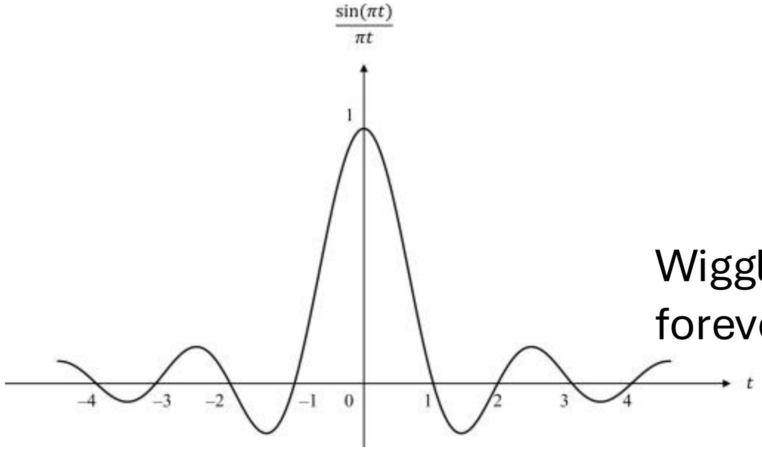
$$\mathrm{f}(t) \otimes h(t)$$

Convolution in the
Time domain



Ideal low-pass filter

$h_{LPF}(t)$



Wiggles extend
forever →

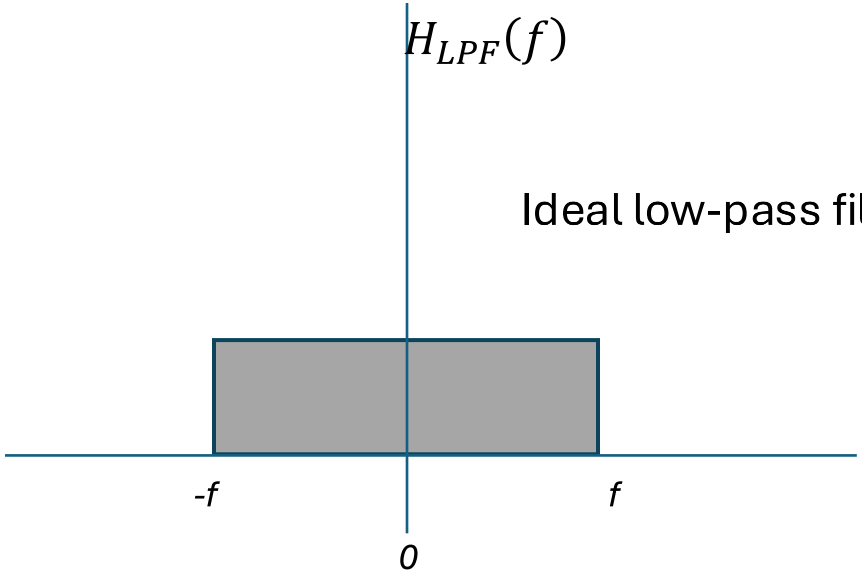
Windowed FIR Filters

$$\mathcal{F}(f) * H(f)$$

Multiplication in the
Fourier domain

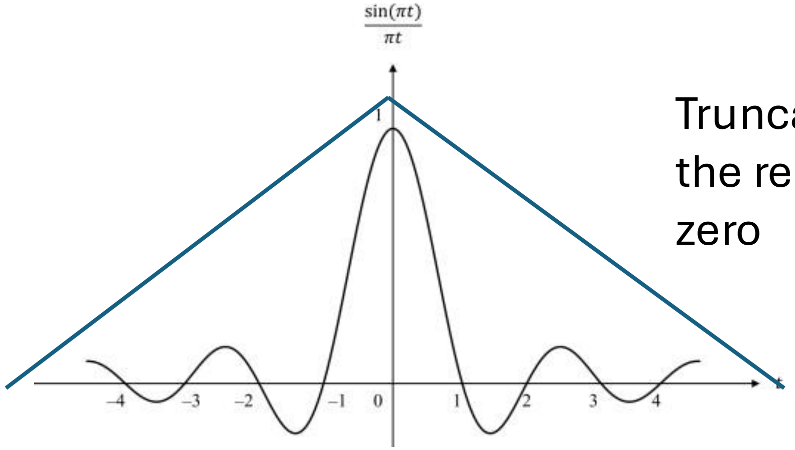
$$\mathrm{f}(t) \otimes h(t)$$

Convolution in the
Time domain



Ideal low-pass filter

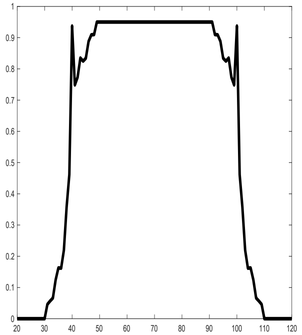
$h_{LPF}(t)$



Truncate or taper the response to zero

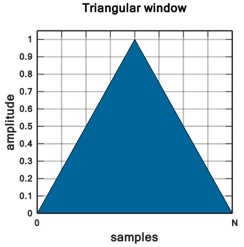
Windowed FIR Filters

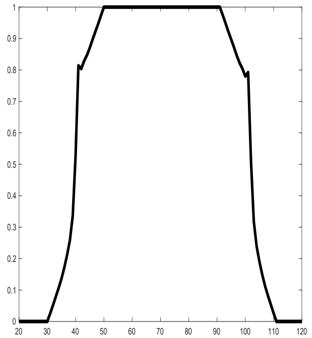
Truncated window
(square window)



The larger the window & more side lobes of the Sinc included → the sharper the transition in the Fourier domain

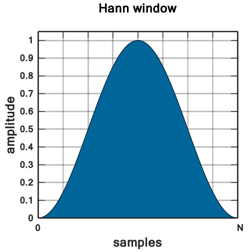
Triangle window

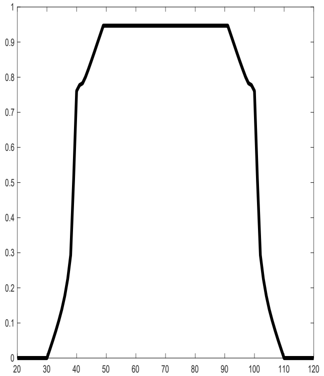




The smoother the window transitions to zero → the fewer ripples in the pass/stop band in the Fourier domain

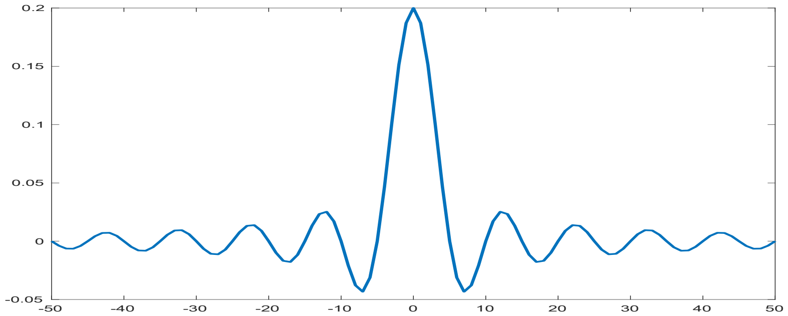
Hann window





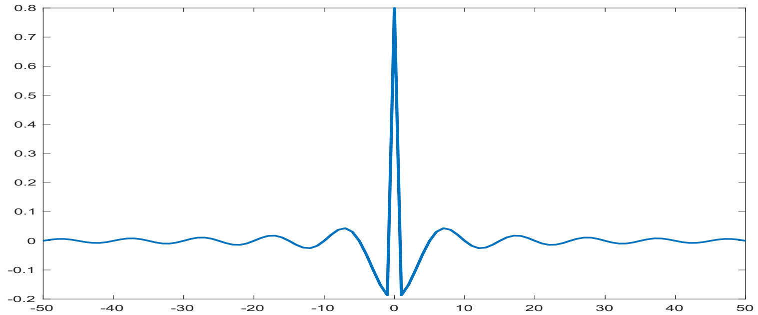
Low-pass

$$\frac{\sin(kw_c)}{\pi k} \qquad \lim_{n\to 0} = \frac{w_c}{\pi}$$



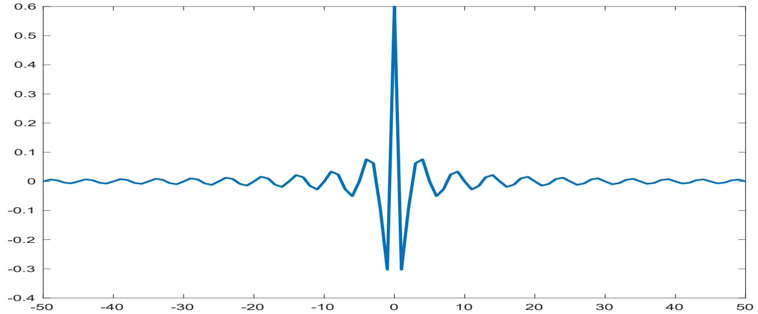
High-pass

$$\frac{-\sin(kw_c)}{\pi k} \qquad \lim_{n\to 0} = 1 - \frac{w_c}{\pi}$$



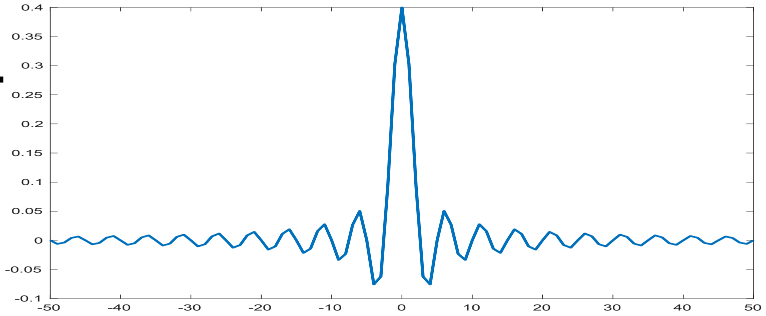
Band-pass

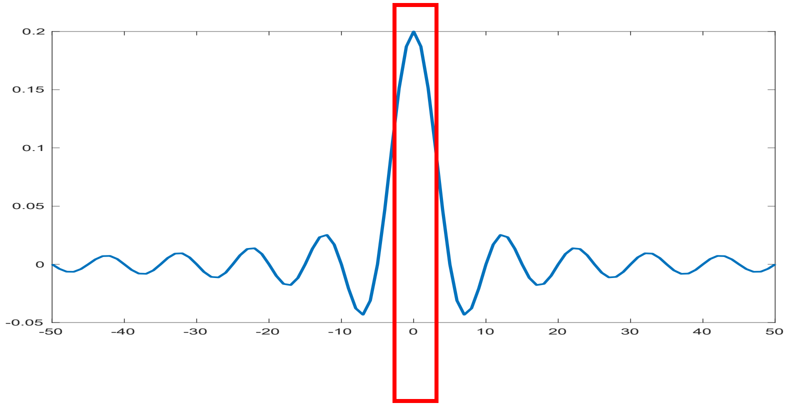
$$\frac{\sin(kw_{up})}{\pi k} - \frac{\sin(kw_{low})}{\pi k} \qquad \lim_{n\to 0} = \frac{w_{up} - w_{low}}{\pi}$$



Notch

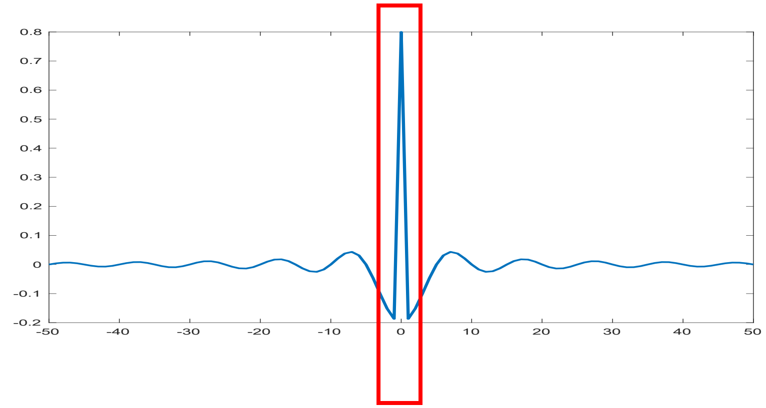
$$\frac{\sin(kw_{low})}{\pi k} - \frac{\sin(kw_{up})}{\pi k} \qquad \lim_{n\to 0} = 1 - \frac{w_{up} - w_{low}}{\pi}$$





1/16*

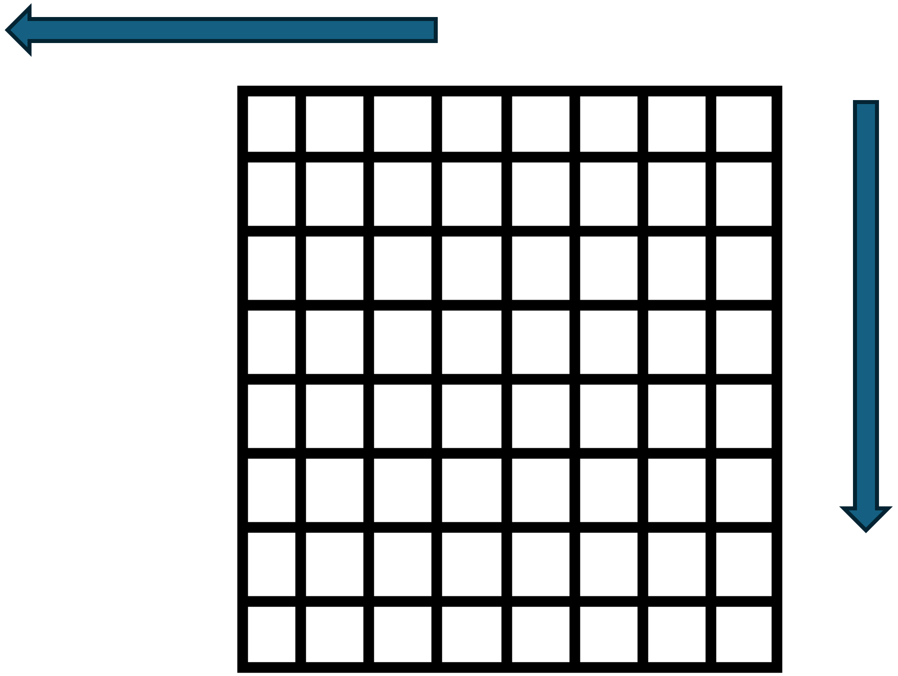
1	2	1
2	4	2
1	2	1



-1	-2	-1
-2	12	-2
-1	-2	-1

$$Y = H * X * H'$$

$$H * X$$



$$X * H'$$

Out = kron(kron(H,X), H')

Out = np.convolution(np.convolution(X,H,axis=0), H,axis=1)

2D convolution as a matrix operation

$$\begin{matrix} a & b & c & d & e & f \\ g & h & \cdots & & & \\ \vdots & & & & & \\ & & & & & z \end{matrix}$$

< 6 x 6 >

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

< 3 x 3 >

$$\overbrace{\boxed{\begin{matrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 \end{matrix}}}^{6} \left.\vphantom{\begin{matrix}1\\0\\0\\0\end{matrix}}\right\} n = (6 - 3) + 1$$

$$\boxed{\begin{matrix} 4 & 5 & 6 & 0 & 0 & 0 \\ 0 & 4 & 5 & 6 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 0 & 5 & 6 & 7 \end{matrix}}$$

$$\boxed{\begin{matrix} 7 & 8 & 9 & 0 & 0 & 0 \\ 0 & 7 & 8 & 9 & 0 & 0 \\ 0 & 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 7 & 8 & 9 \end{matrix}}$$

Construction of “H” matrix

2D convolution as a matrix operation

$$\begin{array}{cccccc}
\begin{bmatrix}1&2&3&0&0&0\\0&1&2&3&0&0\\0&0&1&2&3&0\\0&0&0&1&2&3\end{bmatrix} & \begin{bmatrix}4&5&6&0&0&0\\0&4&5&6&0&0\\0&0&4&5&6&0\\0&0&0&5&6&7\end{bmatrix} & \begin{bmatrix}7&8&9&0&0&0\\0&7&8&9&0&0\\0&0&7&8&9&0\\0&0&0&7&8&9\end{bmatrix} & & & \\
 & \begin{bmatrix}1&2&3&0&0&0\\0&1&2&3&0&0\\0&0&1&2&3&0\\0&0&0&1&2&3\end{bmatrix} & \begin{bmatrix}4&5&6&0&0&0\\0&4&5&6&0&0\\0&0&4&5&6&0\\0&0&0&5&6&7\end{bmatrix} & \begin{bmatrix}7&8&9&0&0&0\\0&7&8&9&0&0\\0&0&7&8&9&0\\0&0&0&7&8&9\end{bmatrix} & & \\
 & & \begin{bmatrix}1&2&3&0&0&0\\0&1&2&3&0&0\\0&0&1&2&3&0\\0&0&0&1&2&3\end{bmatrix} & \begin{bmatrix}4&5&6&0&0&0\\0&4&5&6&0&0\\0&0&4&5&6&0\\0&0&0&5&6&7\end{bmatrix} & \begin{bmatrix}7&8&9&0&0&0\\0&7&8&9&0&0\\0&0&7&8&9&0\\0&0&0&7&8&9\end{bmatrix} & \\
 & & & \begin{bmatrix}1&2&3&0&0&0\\0&1&2&3&0&0\\0&0&1&2&3&0\\0&0&0&1&2&3\end{bmatrix} & \begin{bmatrix}4&5&6&0&0&0\\0&4&5&6&0&0\\0&0&4&5&6&0\\0&0&0&5&6&7\end{bmatrix} & \begin{bmatrix}7&8&9&0&0&0\\0&7&8&9&0&0\\0&0&7&8&9&0\\0&0&0&7&8&9\end{bmatrix}
\end{array} = H$$

<16 x 36 >

n = (6 – 3) +1

2D convolution as a matrix operation

$$\begin{matrix} a & b & c & d & e & f \\ g & h & \dots & & & \end{matrix}$$

<16 x 36 > <36 x 1 >

$$H * g' = x$$

$$reshape(x) \rightarrow < 4\ x\ 4 >$$

< 6 x 6 >

$$\begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ \vdots \end{matrix} = \mathrm{g'}$$

2D convolution as a matrix operation

In MATLAB: conv2(B,A,'valid')

H = kron(eye(N),reshape(A’,1,[]))
g = reshape(B’,1,[])
x = reshape(H*g’ , N, N)

$$< 6\,x\,6 > * < 3\,x\,3 > = < 4\,x\,4 >$$

N = (6 – 3) +1

If you want the same size output as input image:

N=6

= (8 – 3)

need to pad the image from <6x6> to <8x8>

$$\begin{array}{cccccccc} a & a & b & c & d & e & f & f \\ a & a & b & c & d & e & f & f \\ g & g & h & i & j & k & l & l \\ \vdots & & & & & & & \vdots \\ & & & & & & & \\ & & & & & & z & z \end{array} \qquad \begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a & b & c & d & e \\ 0 & g & h & i & j & k \\ \vdots & & & & & \end{array}$$

2D convolution as a matrix operation

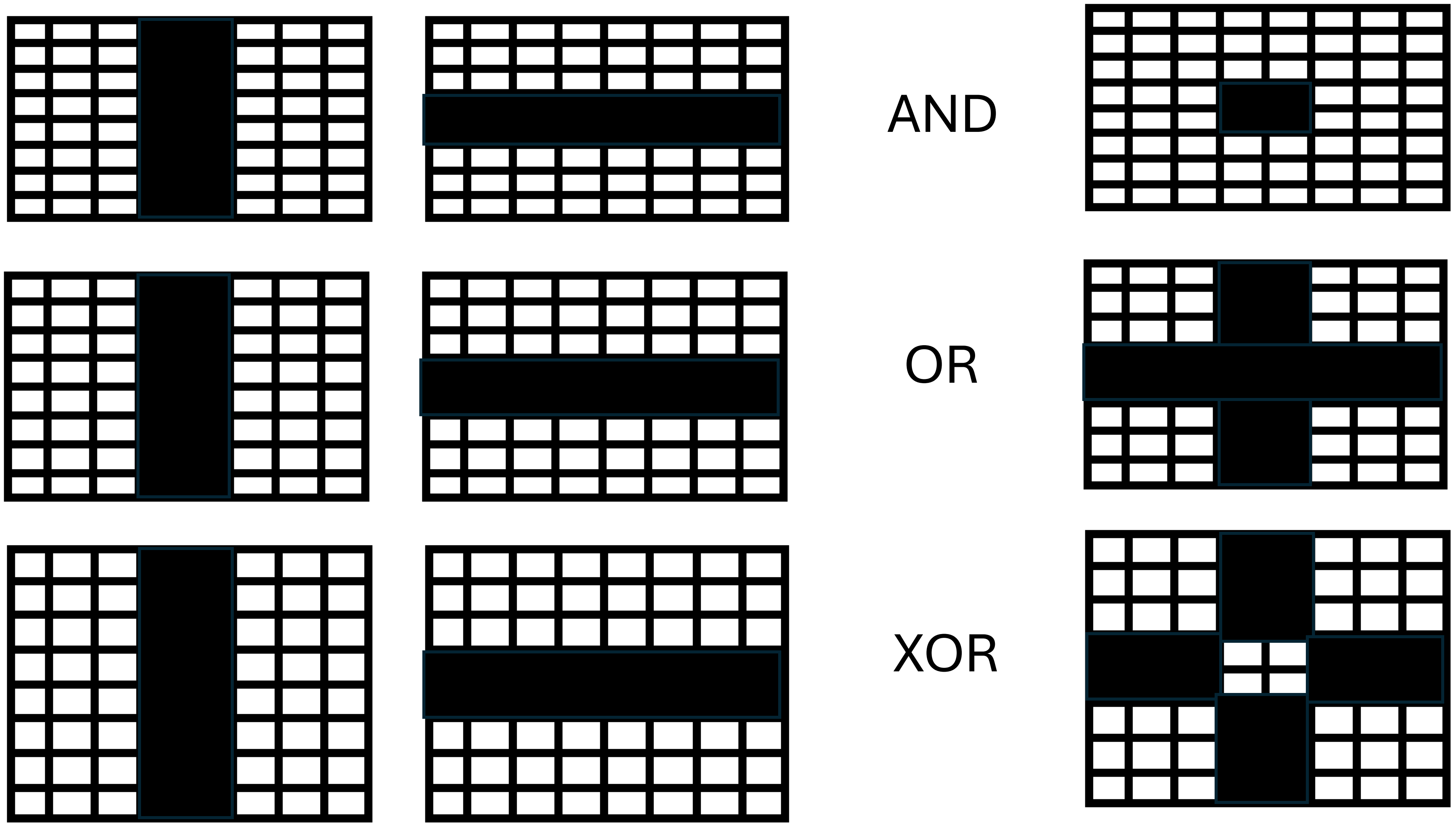
```
kernel= np.ones((3,3))/9

img_smooth = cv2.filter2D(img,-1,kernel)
```

Depth

-1 : gives output image data type as input

cv2.CV_8U = 0	uint8
cv2.CV_8S = 1	int8
cv2.CV_16U = 2	uint16
cv2.CV_16S = 3	int16
cv2.CV_32S = 4	int32
cv2.CV_32F = 5	float32
cv2.CV_64F = 6	float64

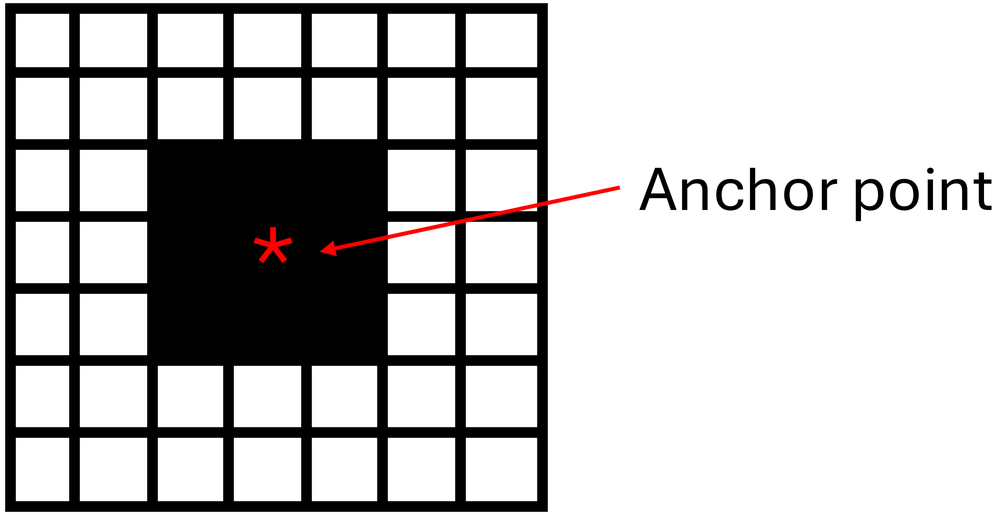
Boolean (Bit-wise) operations



Morphological Operations
Erosion/Dilation

Kernel is Boolean (mask)

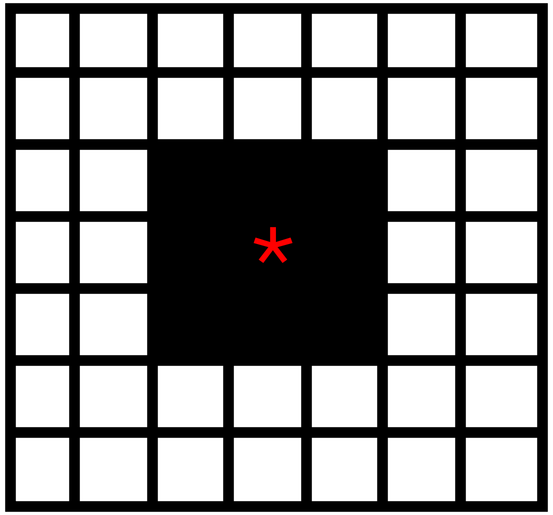
Structure/Kernel



Erosion/Dilation

Anchor point

Structure/Kernel



For {i,j} in image:

move kernel anchor to {i,j}

list = find (kernel ==true)

newVal{i,j} = fcn (image[list])

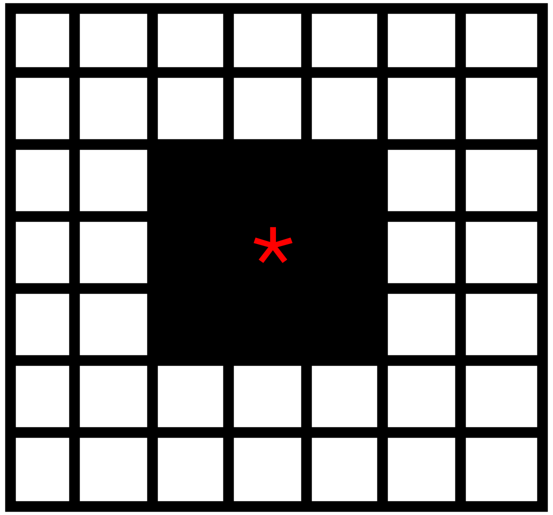
Dilation - MAX (image[list])

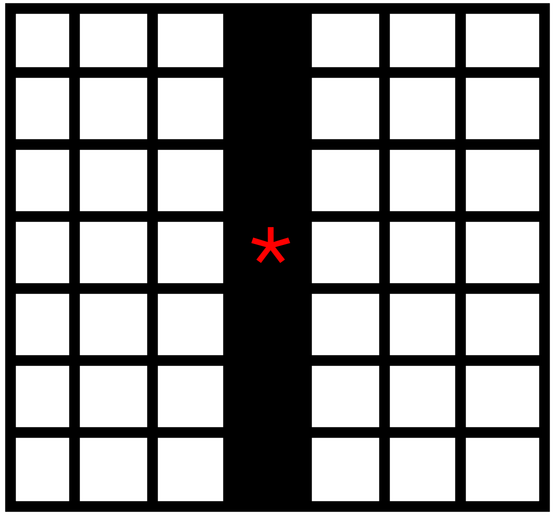
Erosion - MIN (image[list])

Denoising - MEDIAN (image[list])

Smoothing - MEAN (image[list])

Erosion/Dilation

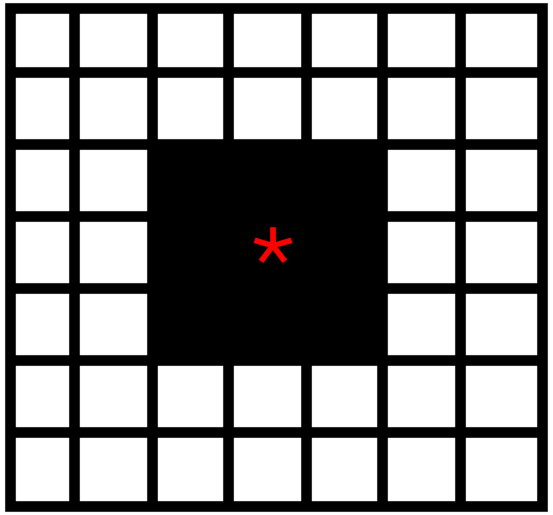




Convolutional (Filtering) Operations

Anchor point

Structure/Kernel



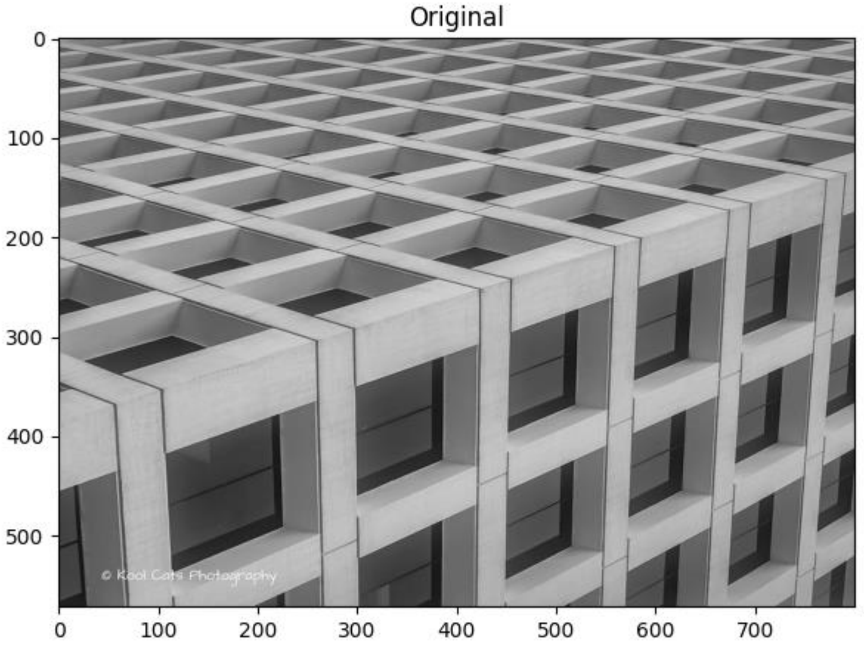
For {i,j} in image:

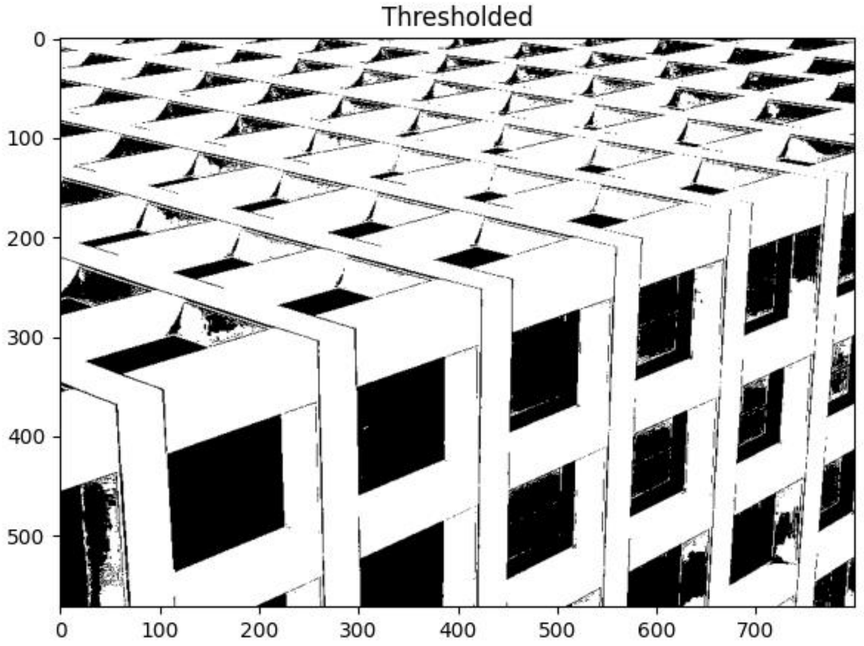
move kernel anchor to {i,j}

newVal{i,j} = sum(image .* kernel)

Thresholding

thres, img_out = cv2.threshold(img, thres, replacement value, type)





Thresholding

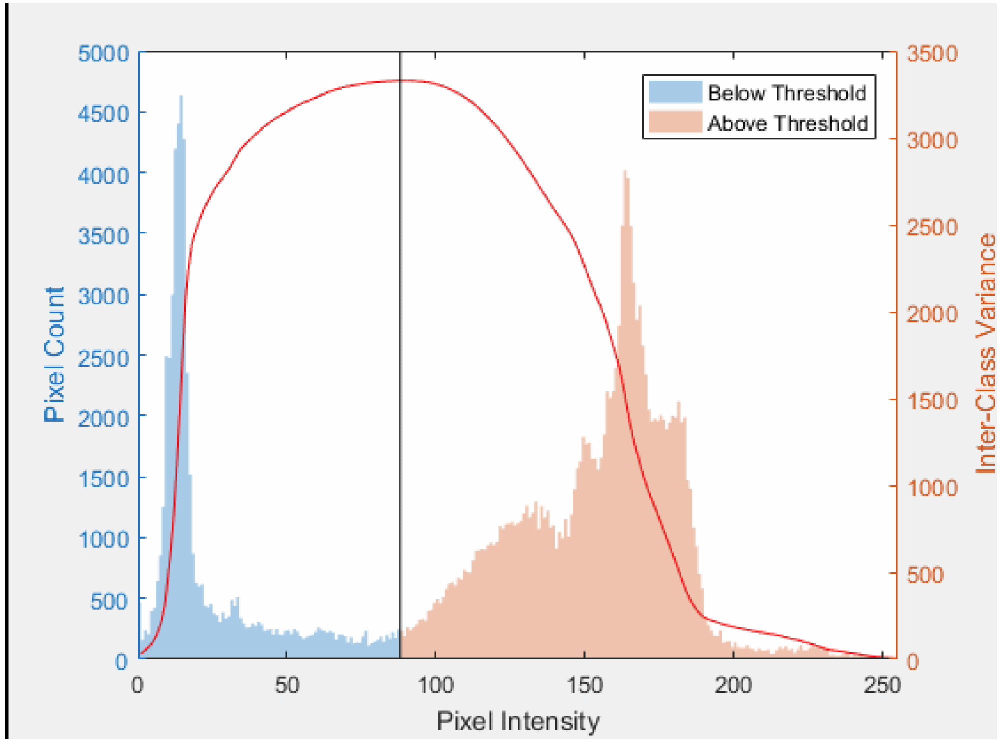
thres, img_out = cv2.threshold(img, thres, replacement value, type)

Types:

• **THRESH_BINARY**	**If x > thresh; x= rep_value; else x=0**
• **THRESH_BINARY_INV**	**If x > thresh; x= 0; else x= rep_value**
• **THRESH_TRUNC**	**If x > thresh; x= x ; else x= thresh**
• **THRESH_TOZERO**	**If x > thresh; x= x; else x= 0**
• **THRESH_TOZERO_INV**	**If x > thresh; x= 0; else x= x**

Otsu’s thresholding

- Nobuyuki Otsu (1979). "A threshold selection method from gray-level histograms". *IEEE Transactions on Systems, Man, and Cybernetics*. **9** (1): 62–66.



Variance values above thresh

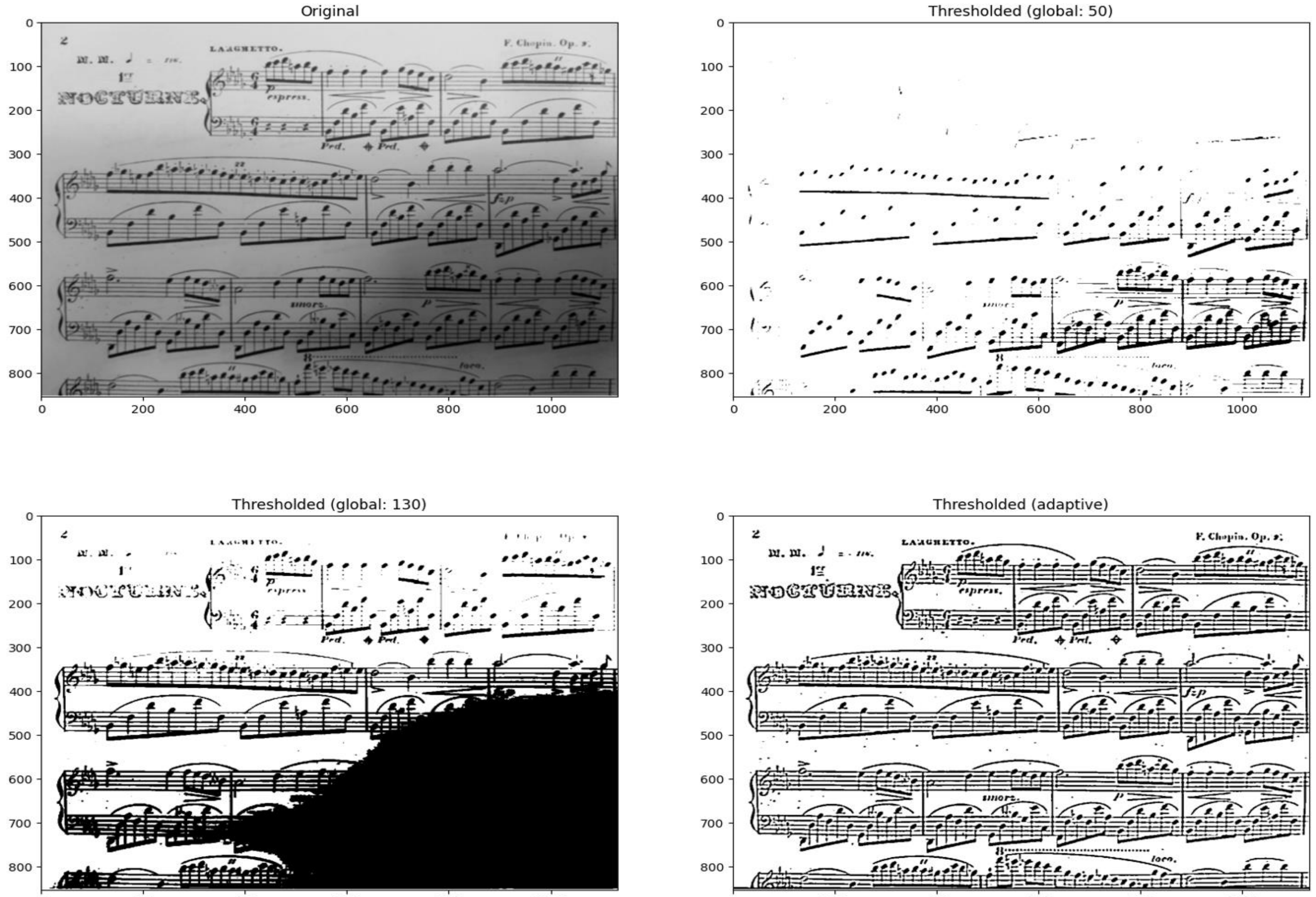
Variance values below thresh

$$\sigma^2 = w_0 \cdot \sigma_0{}^2 + w_1 \cdot \sigma_1{}^2$$

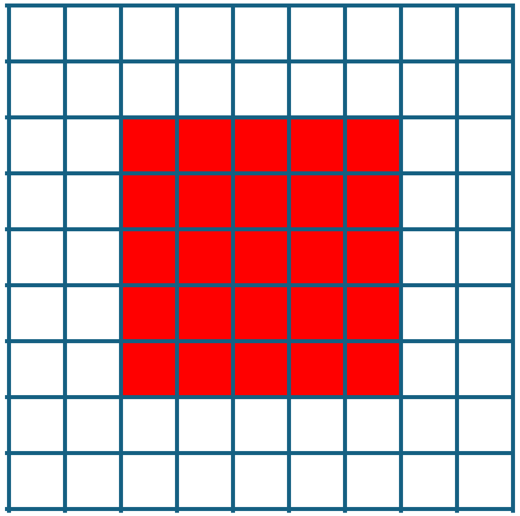
Probability of values below thresh

Probability of values above thresh

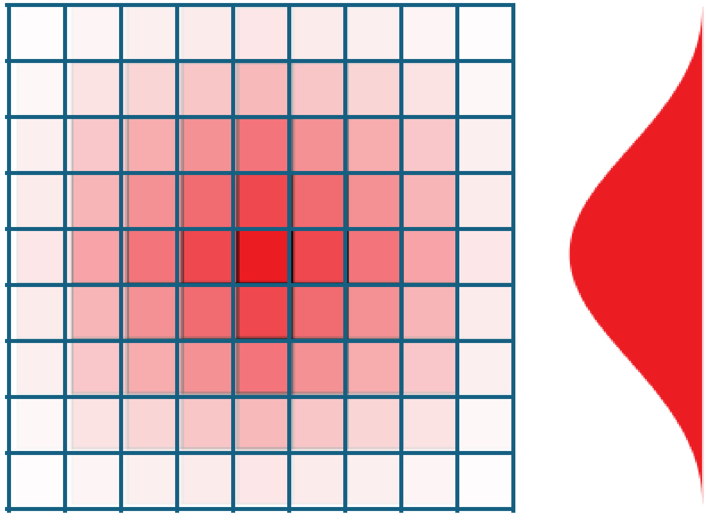
Adaptive thresholding



Adaptive thresholding



$$thres\{i,j\} = \left(\frac{1}{n * m} \sum_{i=1}^{n} \sum_{j=1}^{m} x[i,j] \right) - C$$



$$\text{thes}\{\text{i,j}\} = \left(\frac{1}{\sum_{i=1}^{n} \sum_{j=1}^{m} G[i,j]} \sum_{i=1}^{n} \sum_{j=1}^{m} G[i,j] * x[i,j] \right) - C$$