Image, filtering, convolution

ISTD 50.035

Computer Vision

Acknowledgement: Some images are from various sources: UCF, Stanford cs231n, etc.

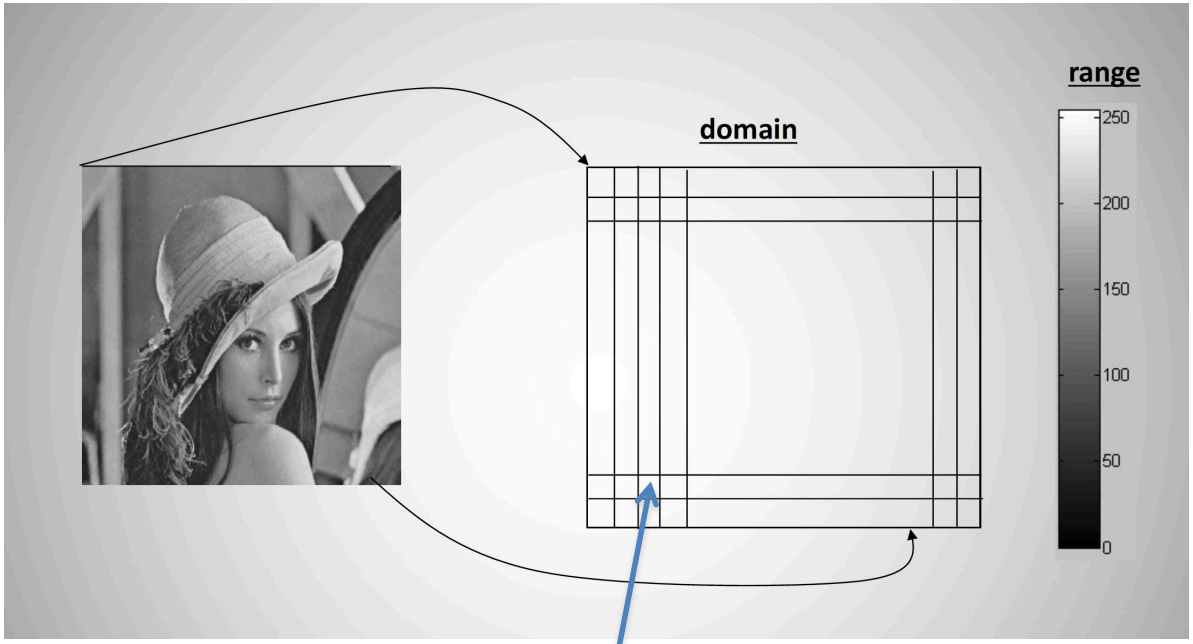
Image is an array of numbers

-Grayscale image

-2D array of numbers (pixels) / matrix

-Number indicates the intensity: [0,255] for 8-bit representation

-Image resolution / number of pixel in an image: 100x100, 1920x1080, etc.



A pixel

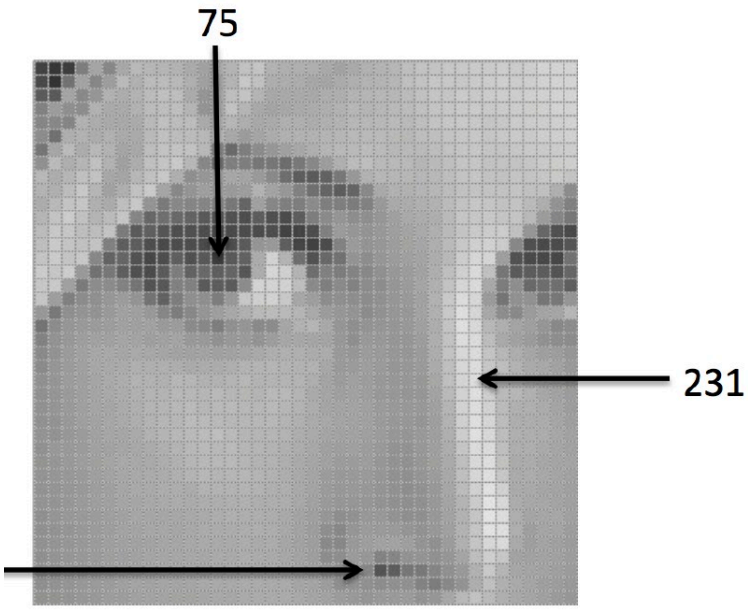
0: black, 255: white

Image is a discrete array of numbers

-Samples from continuous object

-quantized to have a finite number of possible values: [0,1,2 to 255]

- Sometimes normalized from 0 to 1



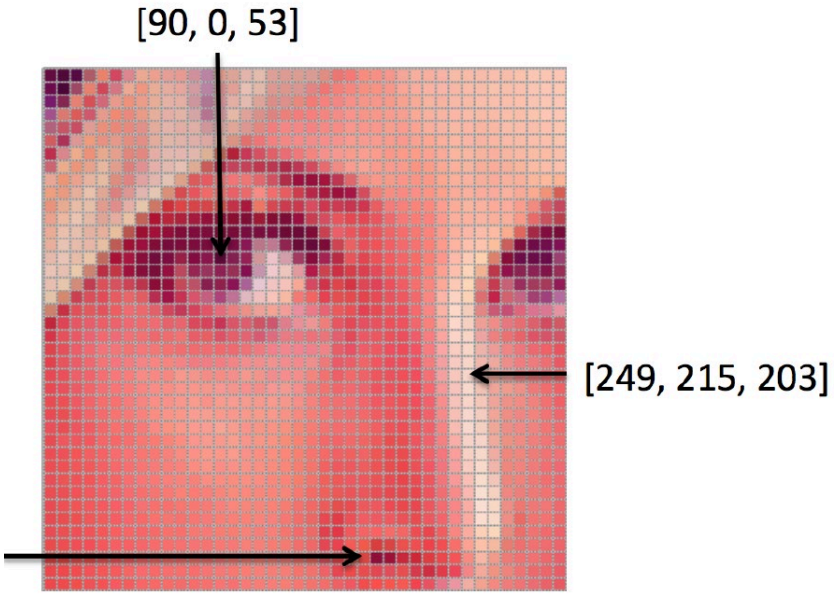
Color image

-Each pixel has three numbers to represent the red, green, blue color intensity

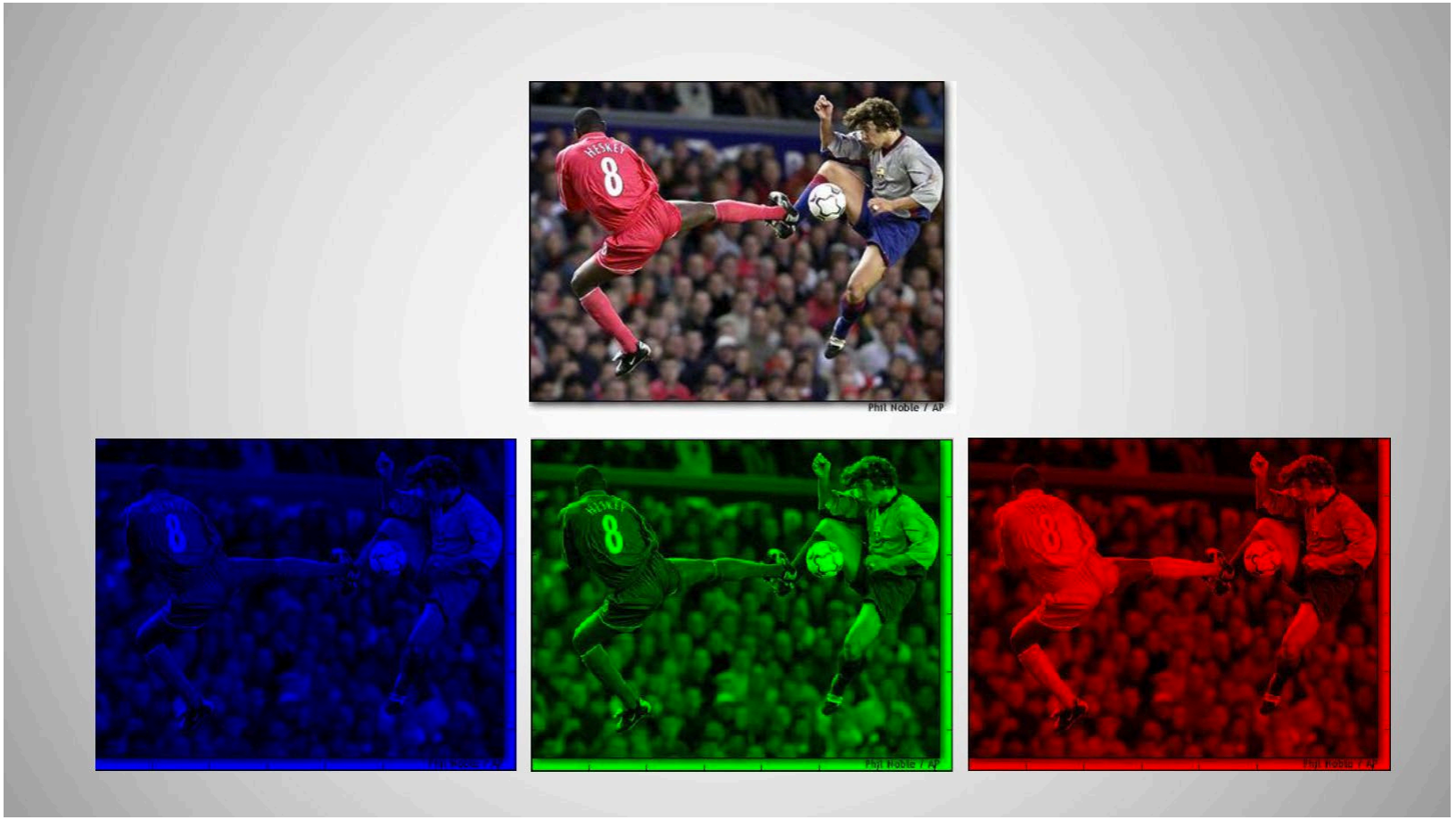
-RGB

-Other color format: HSV, YUV

-Image can be stored in different format: JPEG, PNG, TIF, etc.

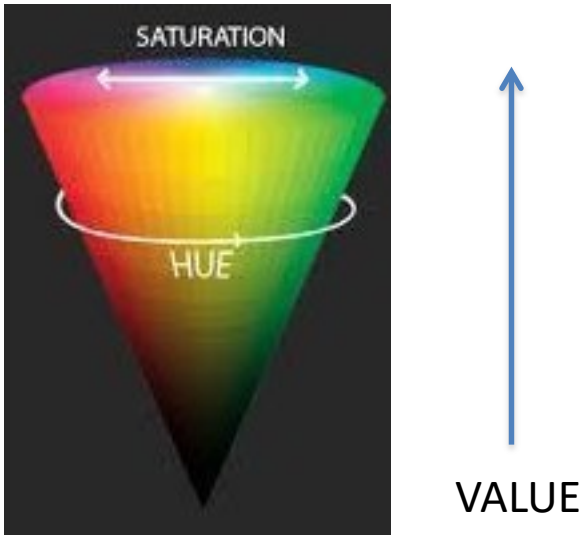


Color image with R, G, B channels



HSV color space

- **Hue** represents color in angle of HSV cone
  - 0-60: red
  - 60-120: yellow
  - 120-180: green
  - 180-240: cyan
  - 240-300: blue
  - 300-360: magenta
- **Saturation** represents the amount of 'grey' in the color: '0' is grey, '1' is pure primary color
  - Small saturation means faded color
  - It is the radius of the HSV cone
- **Value** represents intensity
  - It is the height of the HSV cone



HSV represents color in a similar way as humans perceive color, is easy to work with in some applications

Video

- A video is a sequence of images (frames)
- 30 frame per second (fps)
- Spatial dimension: x,y
- Temporal dimension: t

Image Filtering

- An image processing operation
- Remove some unwanted components: noise
- Extract useful information
- Linear filtering: The output is a linear combination of pixel values in some neighborhood

Filtering is an essential operation in deep neural networks

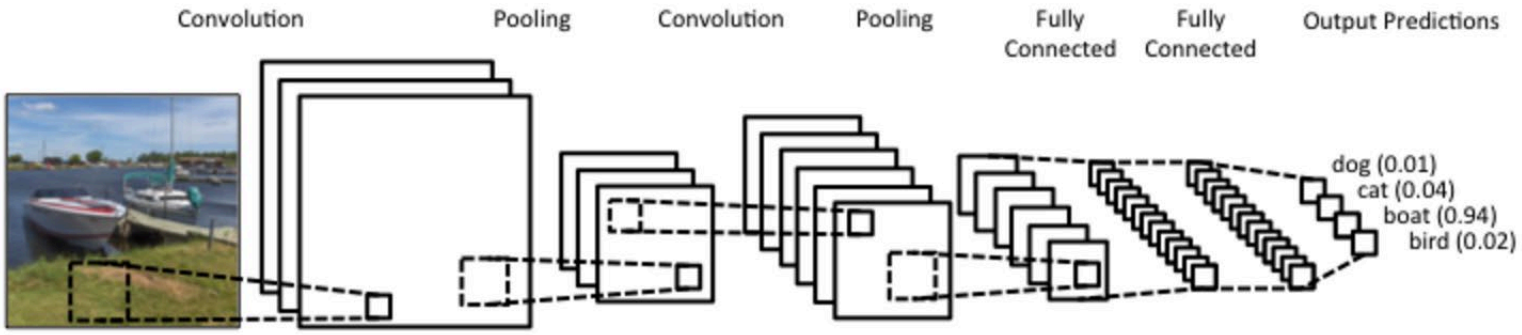


Image Filtering

- Correlation / convolution (precisely, there are subtle differences)

Correlation

$$f \otimes h = \sum_k \sum_l f(k,l)h(k,l)$$

f = Image

h = Kernel

Essentially weighted sum of inputs (over a small neighborhood)

f

f_1	f_2	f_3
f_4	f_5	f_6
f_7	f_8	f_9

$\otimes$

h

h_1	h_2	h_3
h_4	h_5	h_6
h_7	h_8	h_9

$$f \otimes h = f_1h_1 + f_2h_2 + f_3h_3 + f_4h_4 + f_5h_5 + f_6h_6 + f_7h_7 + f_8h_8 + f_9h_9$$

Image Filtering

- Convolution (Math)
- Flipping is skipped in this course

Convolution

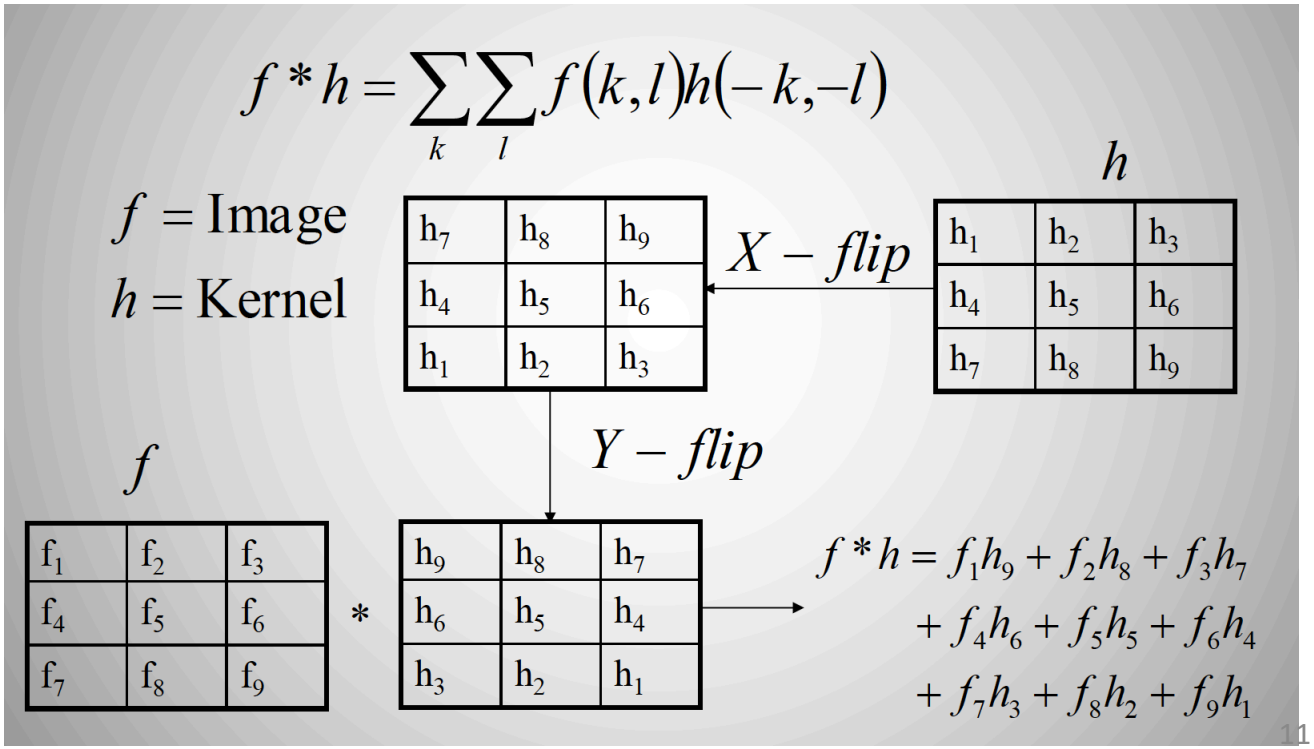


Image Filtering

- Filtering (convolution) operation

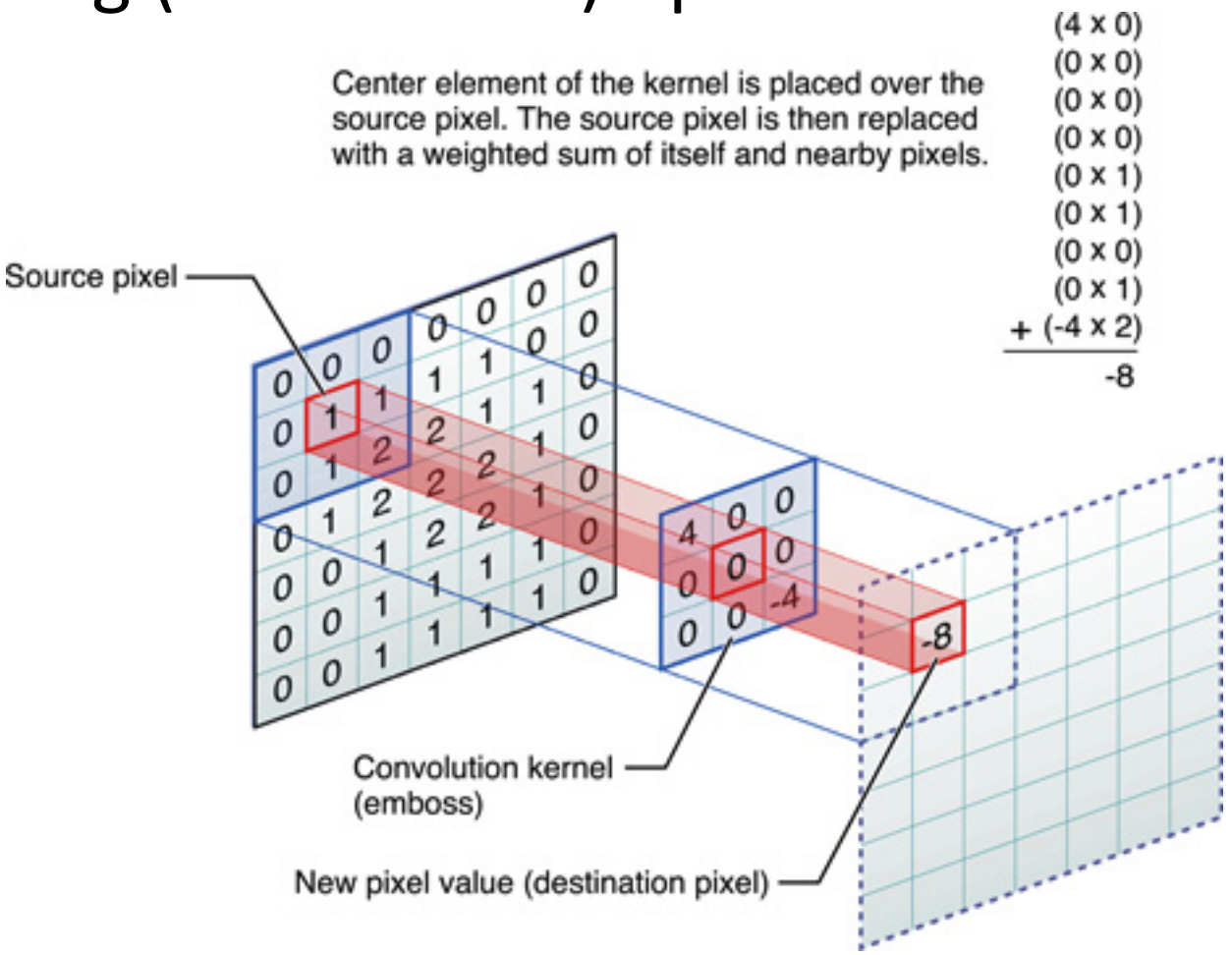


Image Filtering

- Filtering (convolution) operation
- Slide the filter kernel over the entire image to produce the output (image/activation)

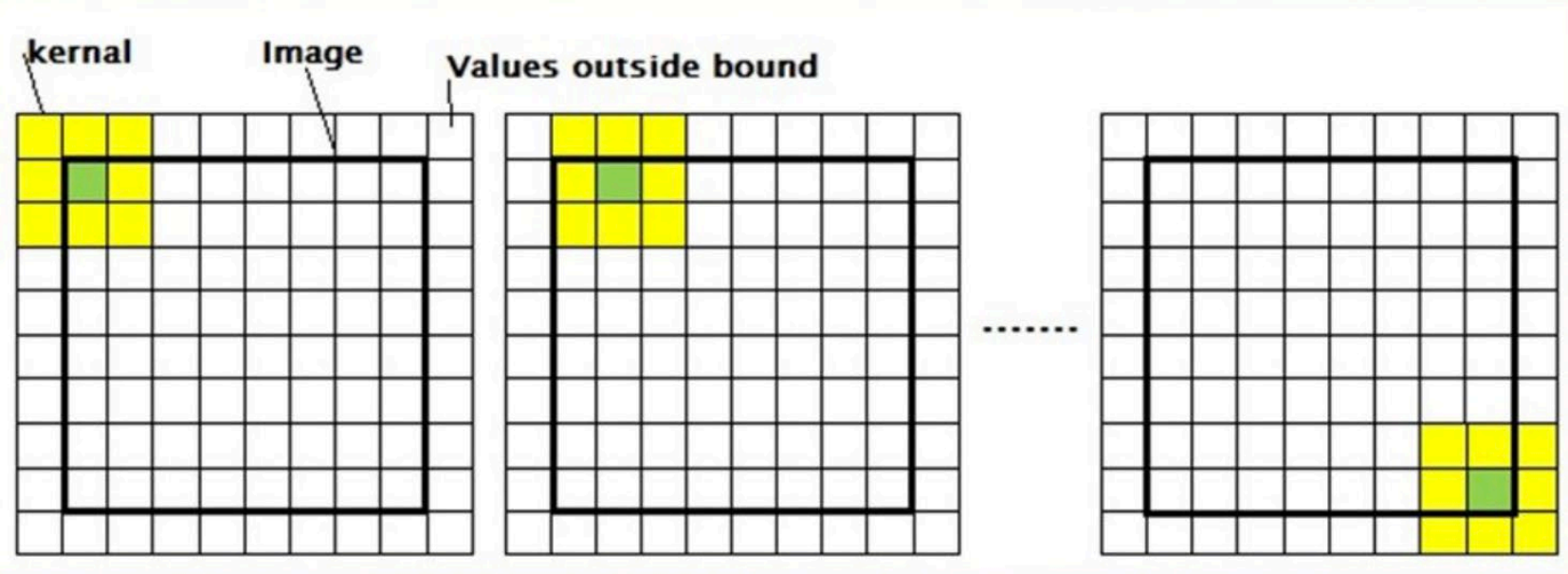


Image Filtering

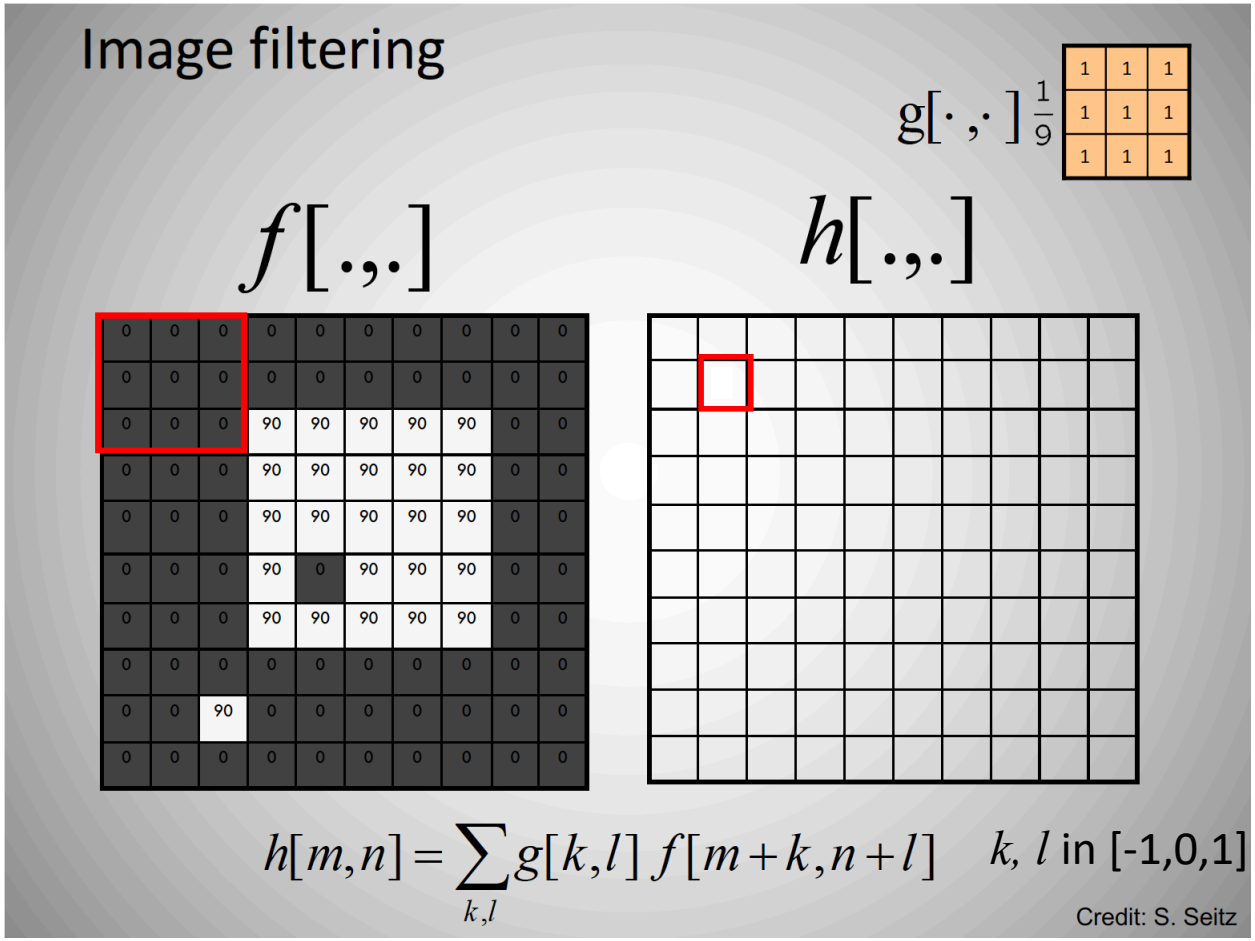


Image Filtering

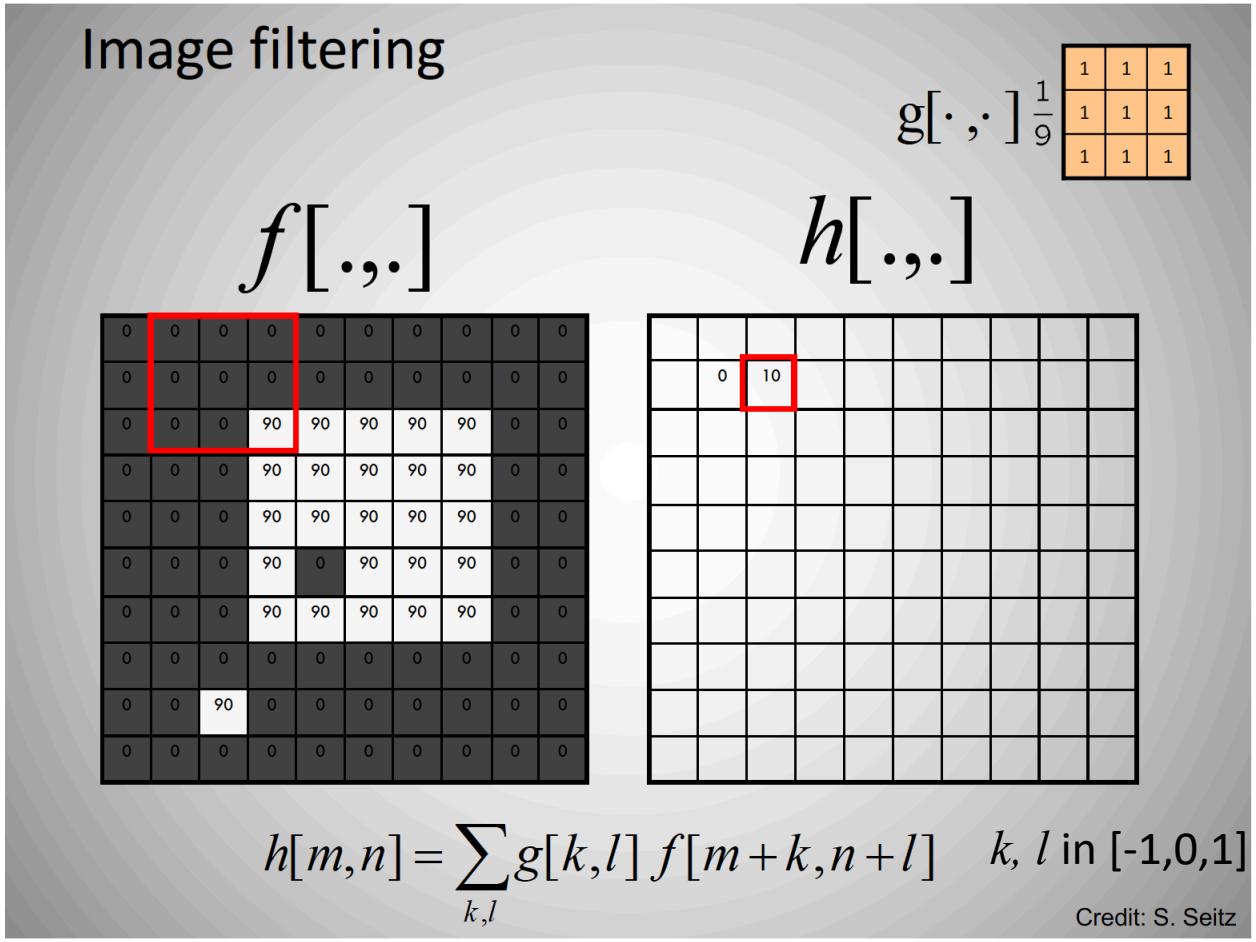


Image Filtering

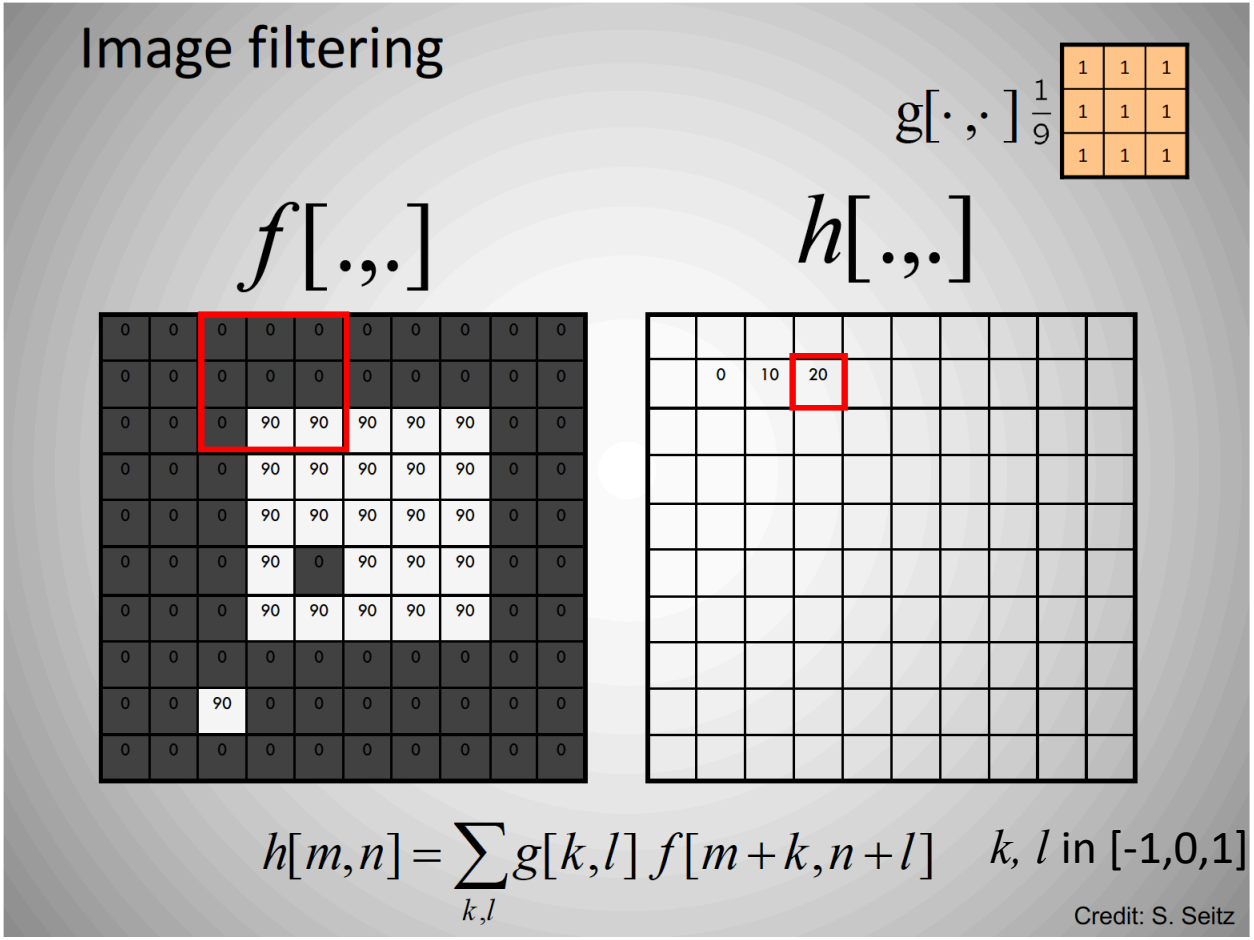


Image Filtering

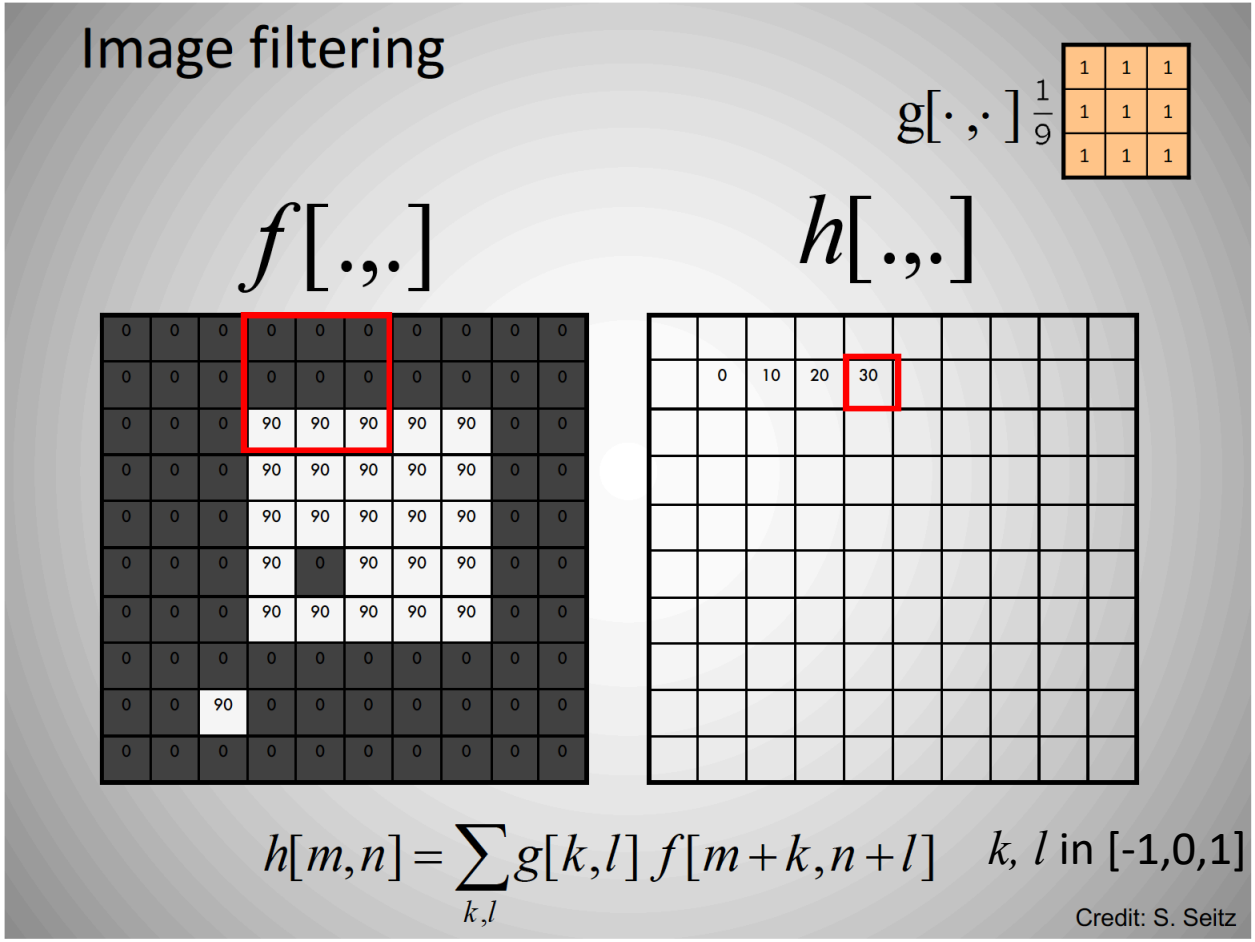


Image Filtering

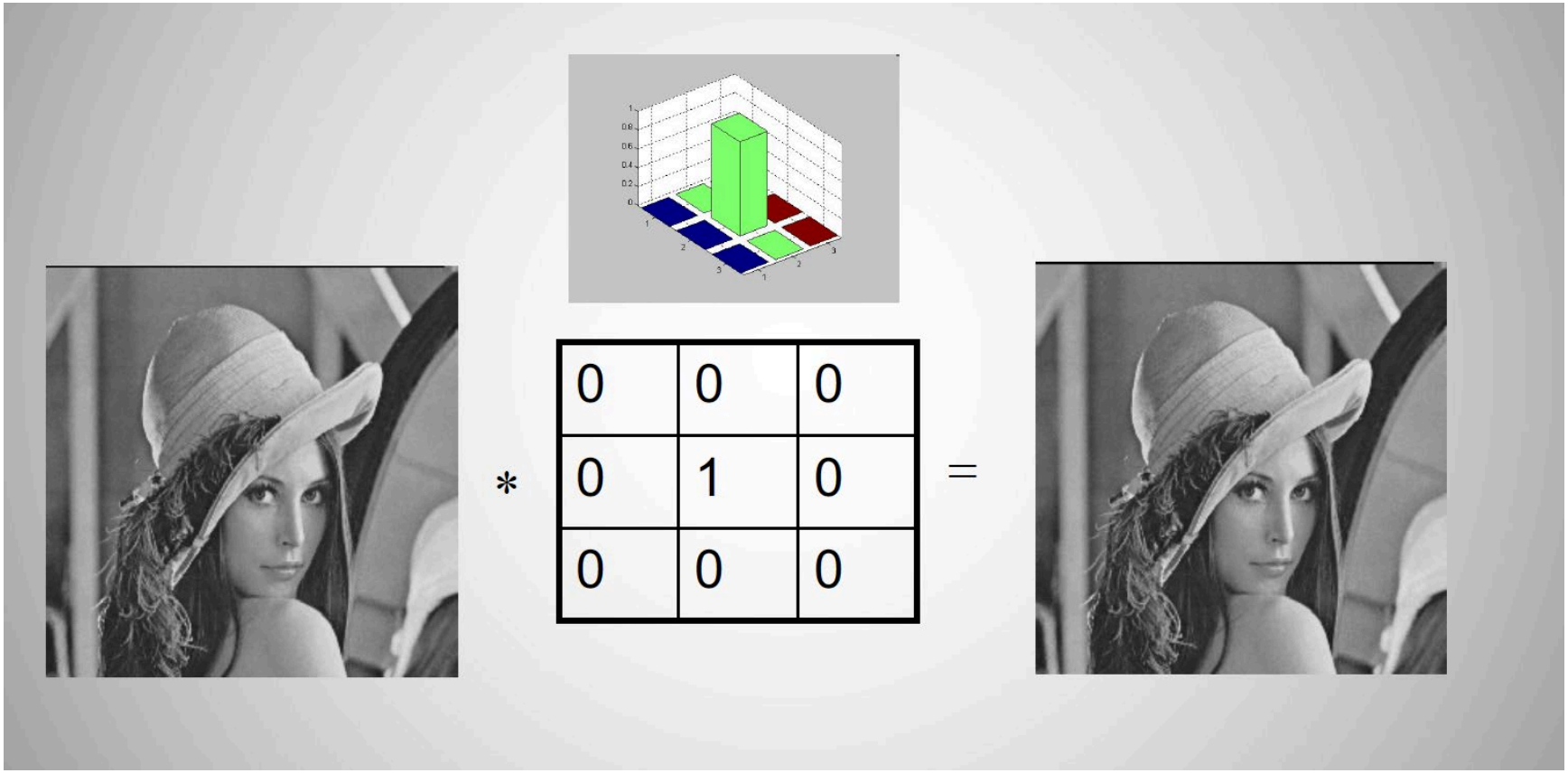
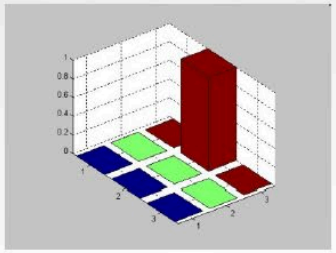


Image Filtering

*



0	0	0
1	0	0
0	0	0

=

Image Filtering

- Cohort exercise: calculate the output of the following filtering (you can ignore the boundary condition)

Img =

```
[[30 40 20 30 40]
 [40 20 30 40 30]
 [20 30 40 30 40]
 [30 40 30 40 20]
 [40 30 40 20 30]]
```

kernel =

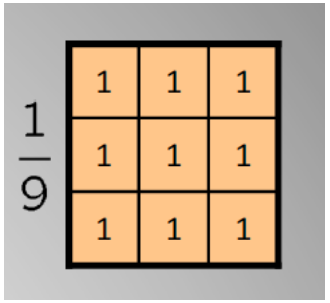
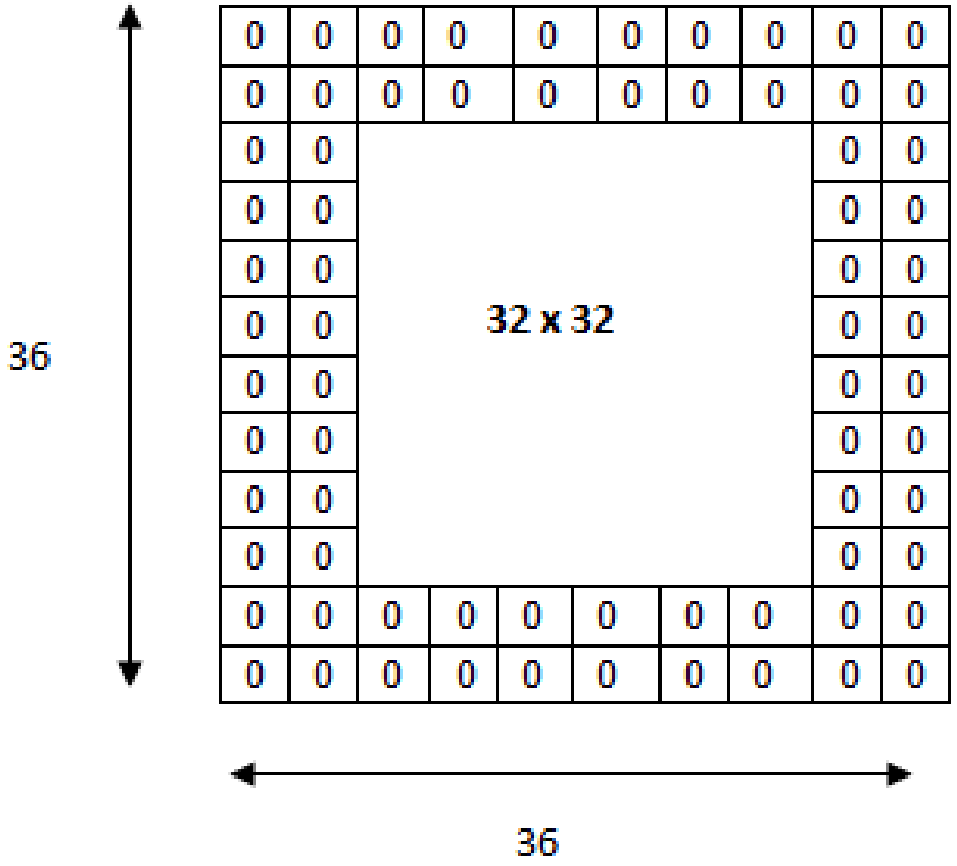


Image Filtering

- Padding to handle boundary condition

- Stride: how many pixels to shift the filter kernel in each step

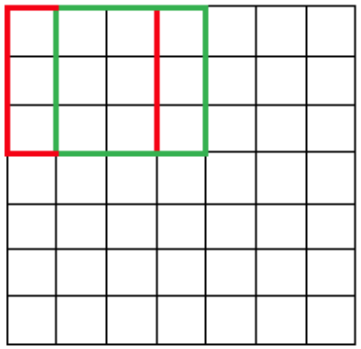
Padding



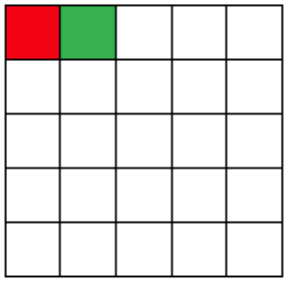
What would be the size of the kernel for an output of equal size (32x32)?

Stride

7 x 7 Input Volume

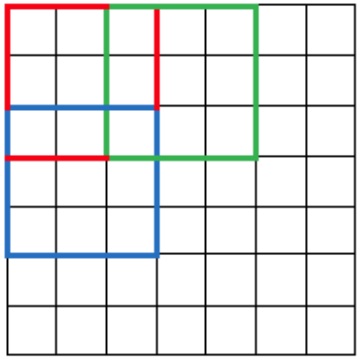


5 x 5 Output Volume

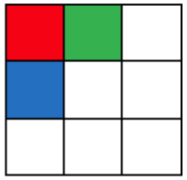


Stride=1

7 x 7 Input Volume



3 x 3 Output Volume

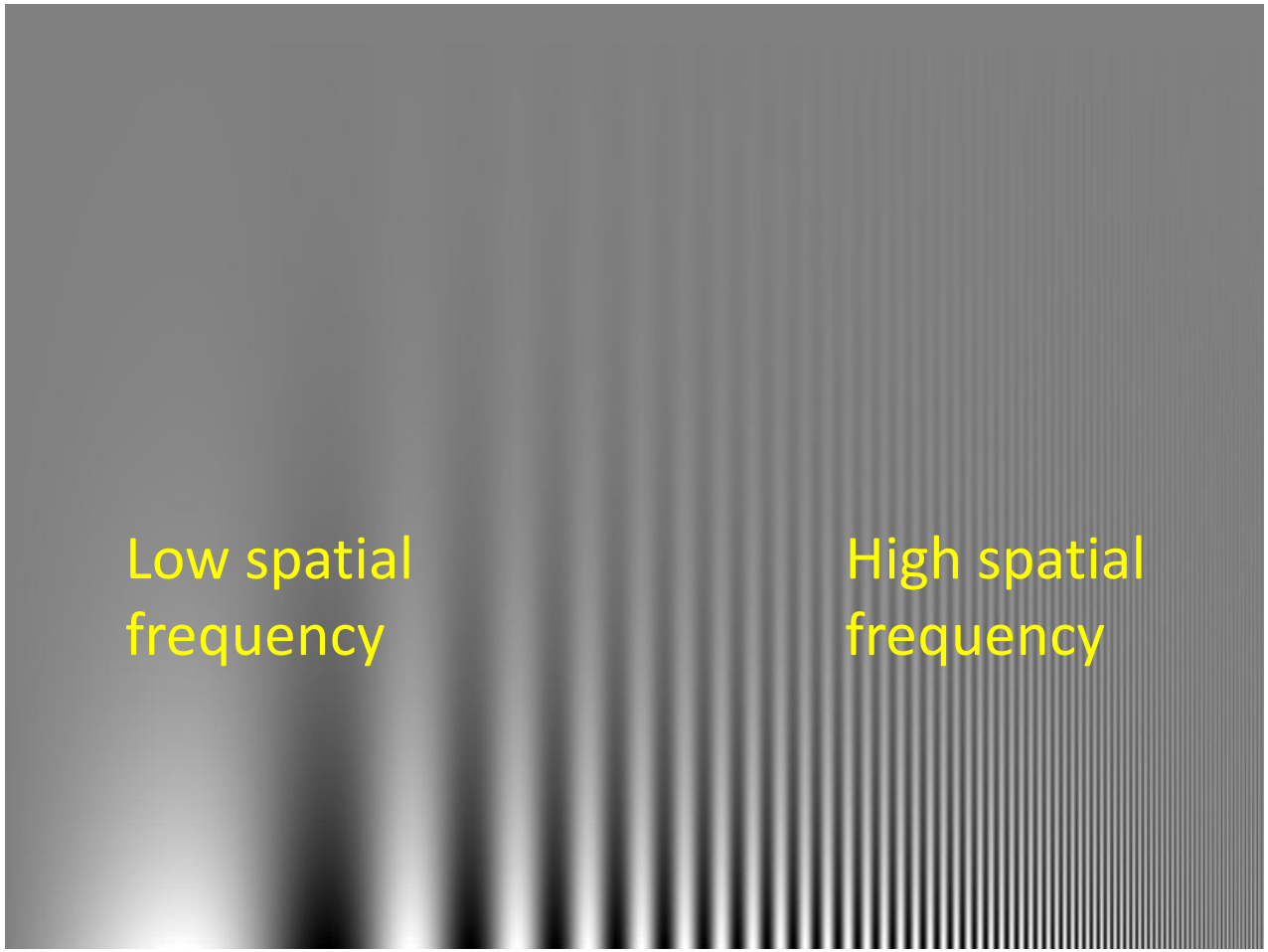


Stride=2

Low-pass filtering

- LPF retains low spatial frequency components, remove high spatial frequency components (noise, texture)

Image Frequency



contrast

frequency

Image Frequency

- Pixel intensity at two rows

Which row has higher spatial frequency?

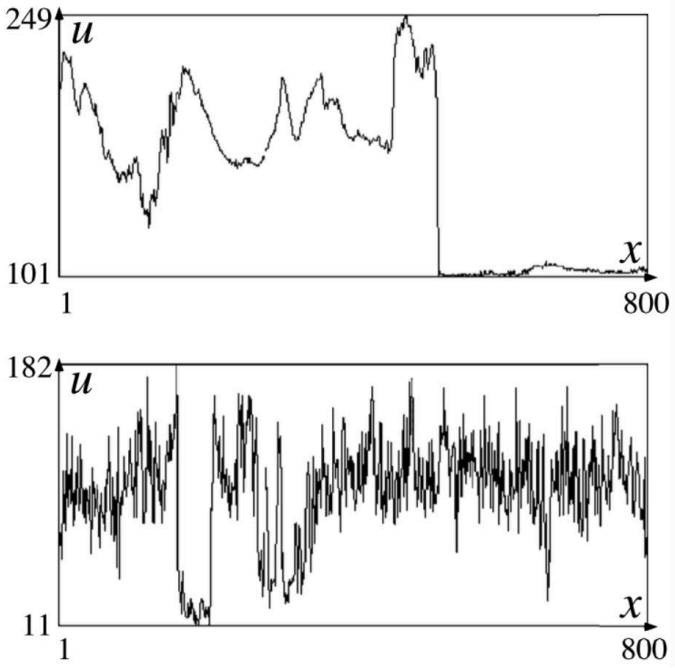
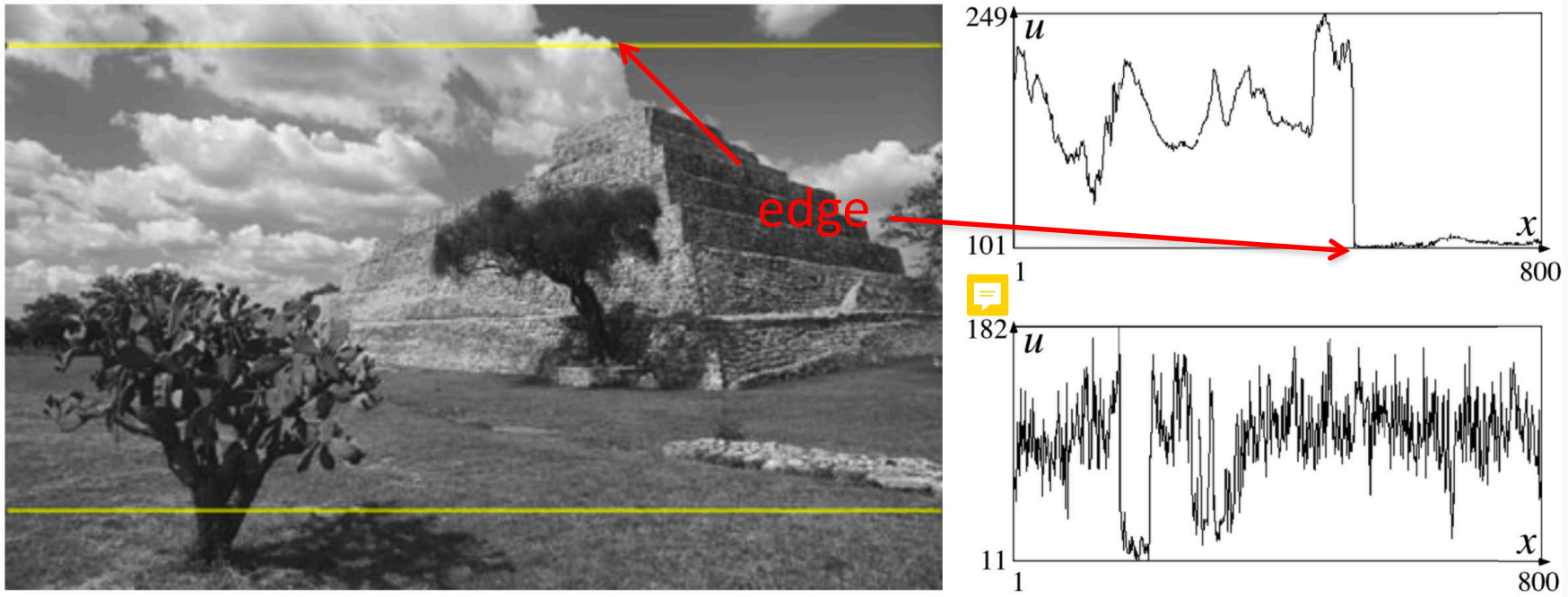


Image Frequency

- Pixel intensity at two rows

Low spatial frequency



High spatial frequency

Image Frequency

Low spatial frequency

High spatial frequency

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

Image Frequency

- Image frequency can be obtained quantitatively using 2D Fourier Transform, which decomposes the an image into sine and cosine components

Definition

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)}\,dx\,dy,$$
$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{j2\pi(ux+vy)}\,du\,dv$$

Vertical spatial frequency

$|F(u,v)|$

Horizontal spatial frequency

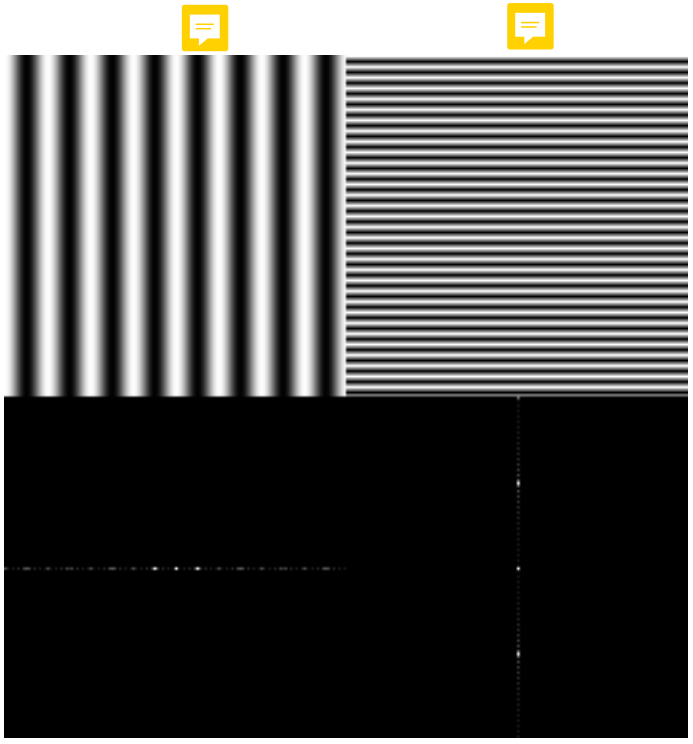
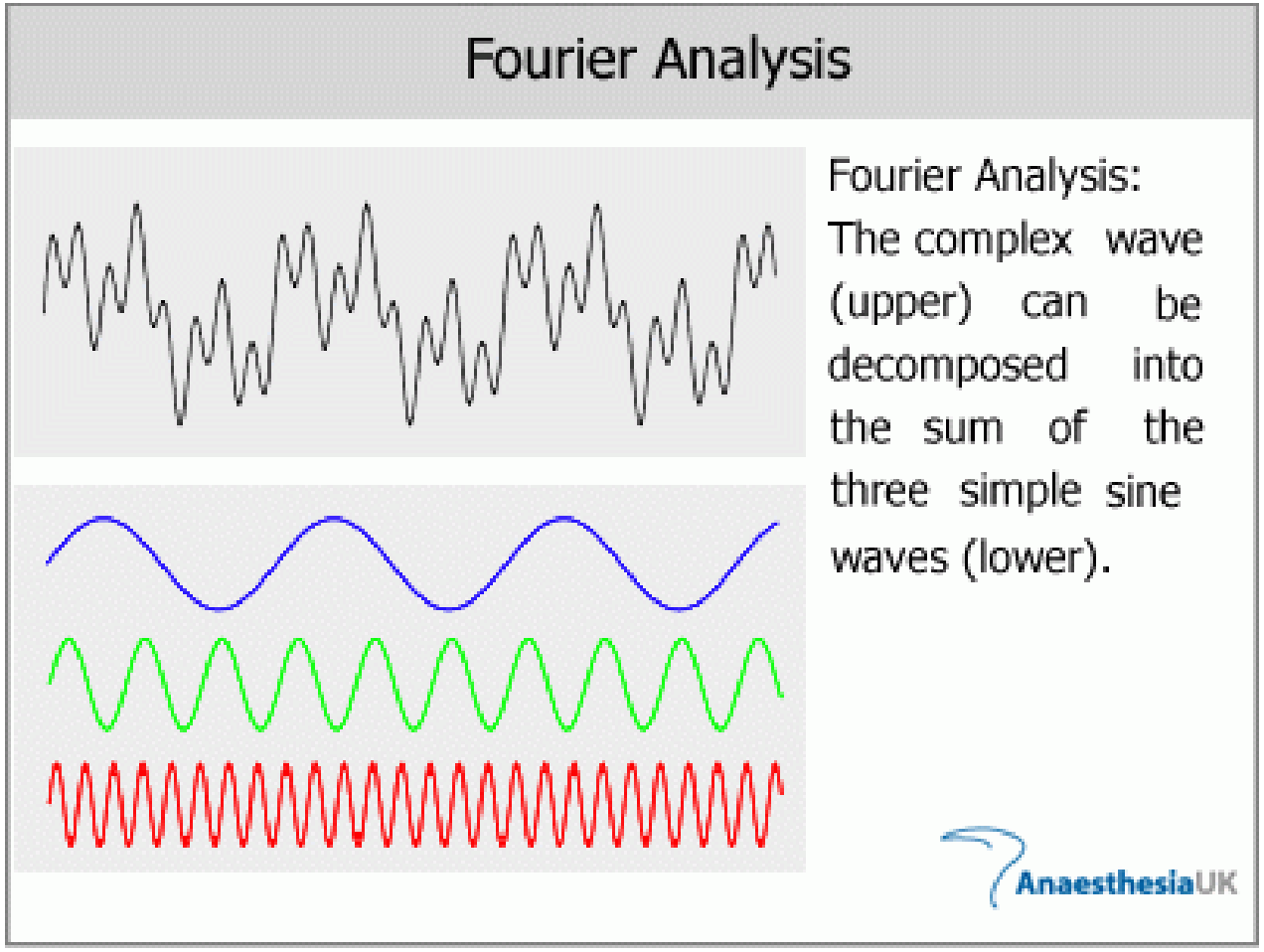


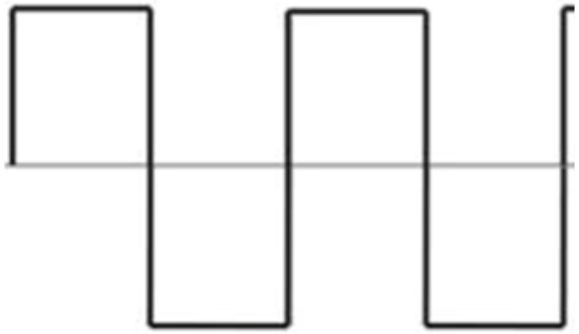
Image in frequency domain

Center point is due to the average intensity

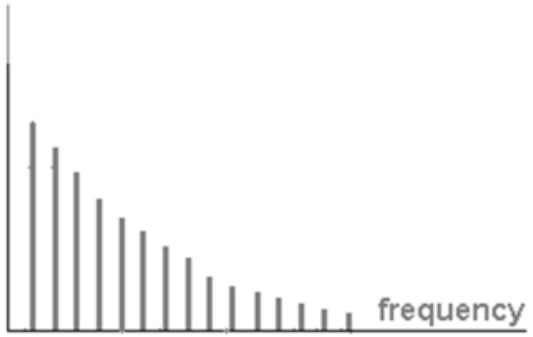
Fourier transform



Fourier transform



$$= \quad f(x) \quad = \sum_{n=1,3,5,\ldots} \frac{1}{n} \sin nx$$



[A. Zisserman]

Image Frequency

- Image frequency can be obtained quantitatively using 2D Fourier Transform, which decomposes the an image into sine and cosine components

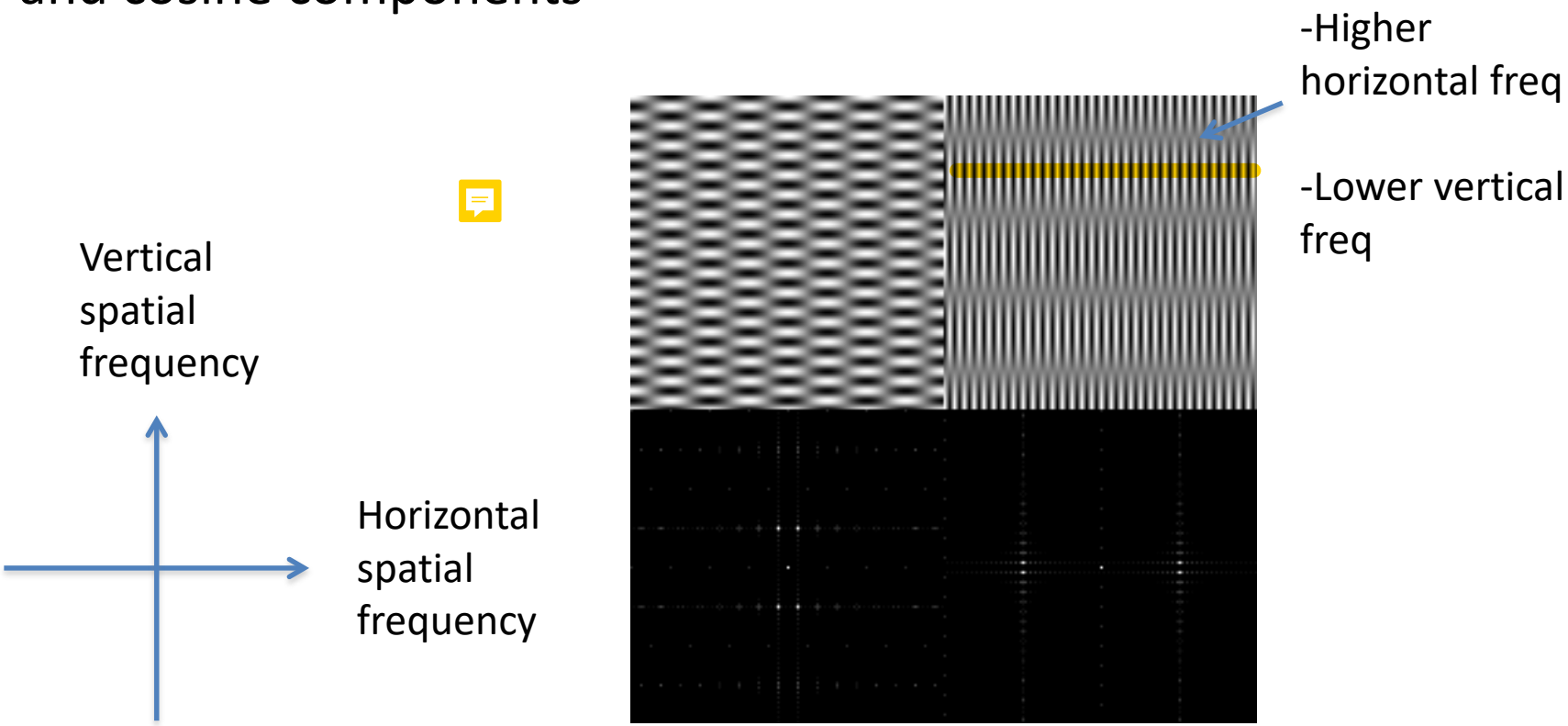
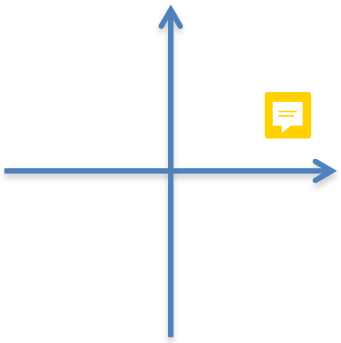
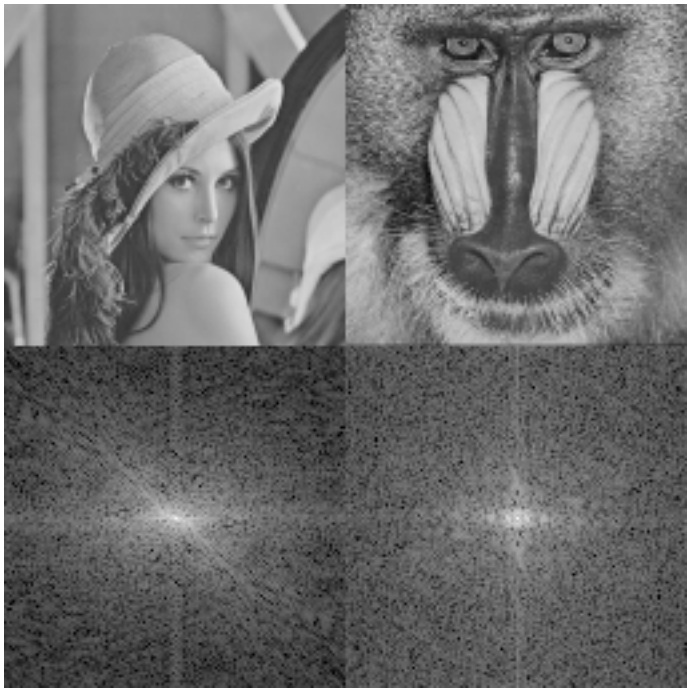


Image Frequency

- Image frequency can be obtained quantitatively using 2D Fourier Transform, which decomposes the an image into sine and cosine components

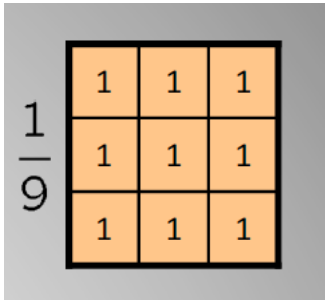


Horizontal
spatial
frequency



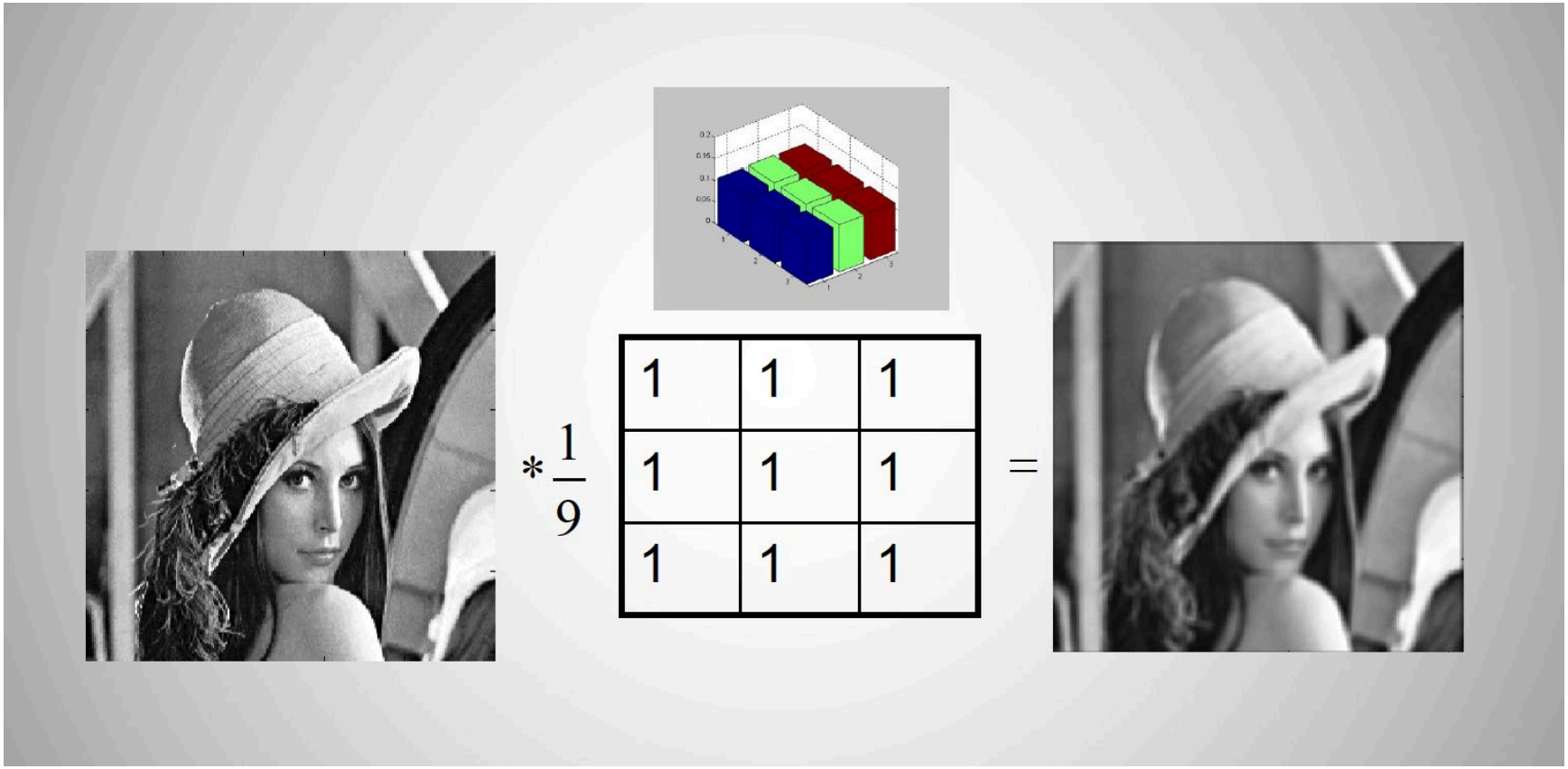
Low-pass filtering

- LPF retains low spatial frequency components, remove high spatial frequency components (noise, texture)
- An example of low-pass filter kernel

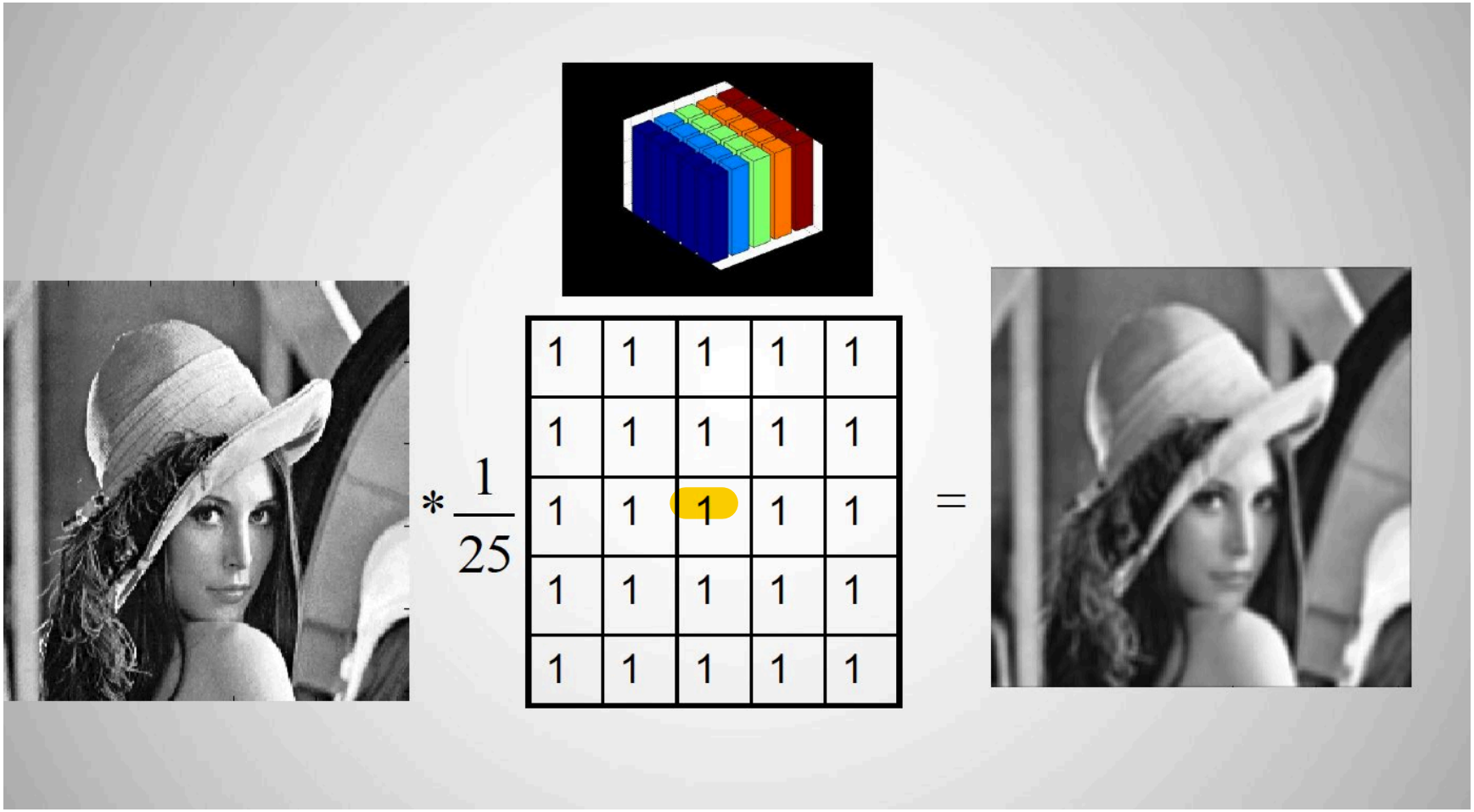


(see cohort exercise in filtering)

Low-pass filtering

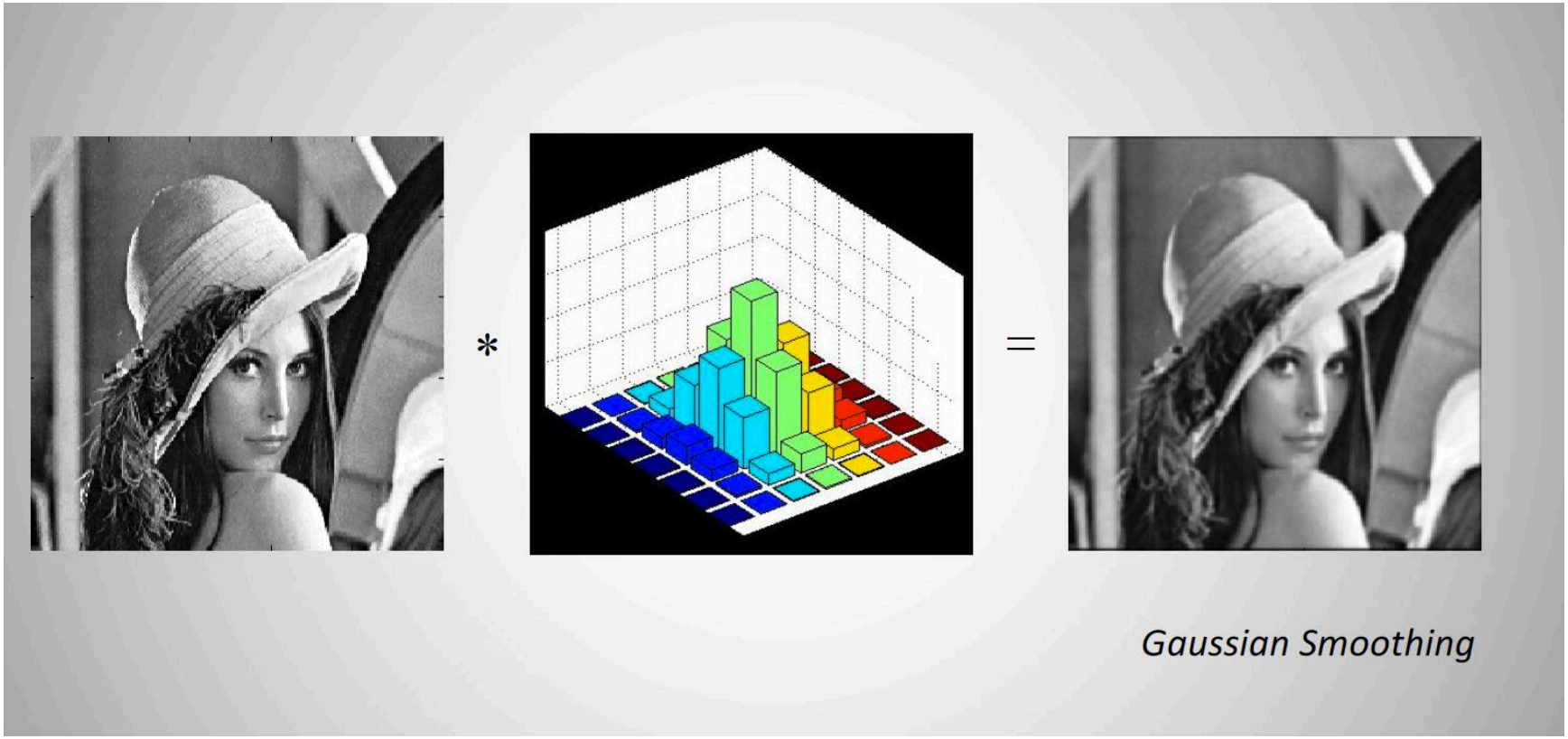


Low-pass filtering



Larger kernel -> more blurry

Low-pass filtering



Gaussian function: $f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$

Low-pass filtering

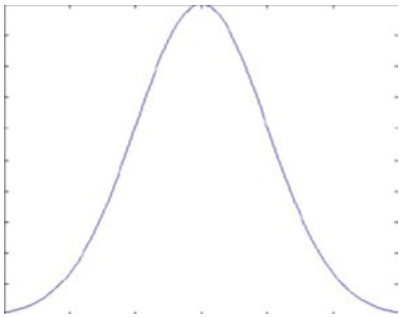
Gaussian Smoothing

Smoothing by Averaging

Noise and denoising

Additive Gaussian Noise

$$I_{distorted} = I_{original} + n$$



n is a normally distributed

$n \sim p(n; \mu, \sigma)$

Noise: high spatial frequency

Denoising: low-pass filtering to remove the signal component with high spatial frequencies

Denoising

After additive
Gaussian Noise

After Averaging

After Gaussian Smoothing

Denoising by median filter

-**Non-linear filtering**: use the median as the filter output

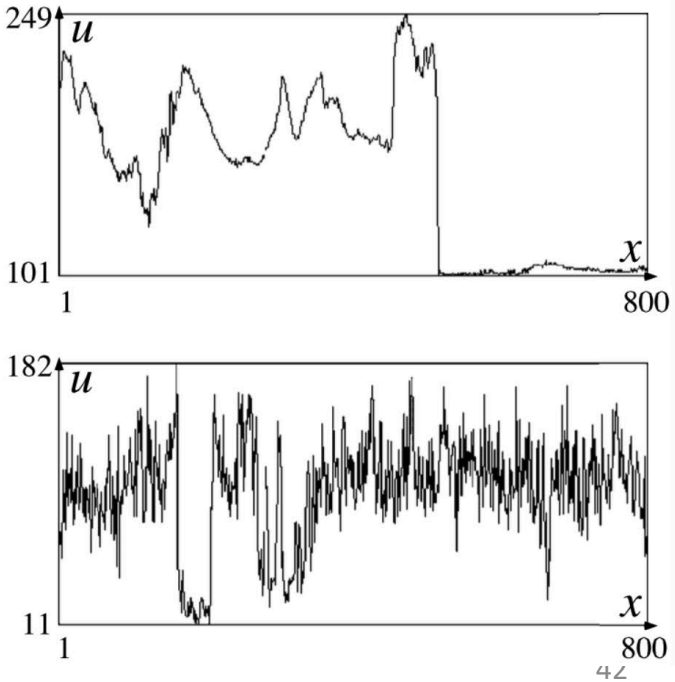
Median(2,80,6) -> 6

-More robust against outliers

-But more computationally-intensive

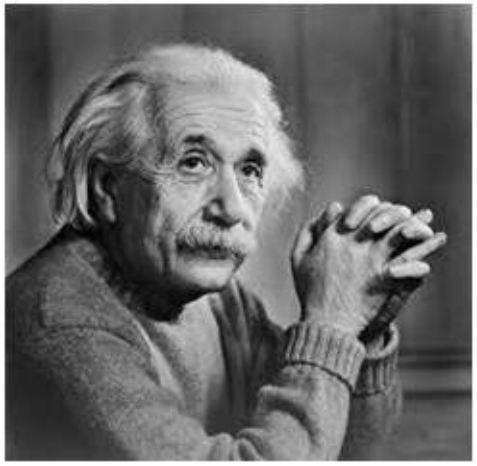
Edge detection

- Use image filtering to extract useful information: edge
- Edge: significant change in intensity values



Edge detection

- Why edge detection:
  - Edge provides important shape information: fundamental in understanding the image
  - Edge sharpen to improve visual quality



Edge detection

- Prewitt operator (as the filter kernel/mask)
  - Detect horizontal edge, vertical edge
  - Compute the difference of the neighboring pixels to indicate the likelihood of edges

```
[[-1,0,1],
 [-1,0,1],
 [-1,0,1]]
```

Detect vertical edge

```
[[-1,-1,-1],
 [ 0,  0, 0],
 [ 1,  1, 1]]
```

Detect horizontal edge

Edge detection

- Prewitt operator (as the filter kernel/mask)

[[-1,0,1],
[-1,0,1],
[-1,0,1]]

Detect vertical edge

[[-1,-1,-1],
[0, 0, 0],
[1, 1, 1]]

Detect horizontal edge

Edge detection

- Sobel operator (as the filter kernel/mask)

[[-1,0,1],
[-2,0,2],
[-1,0,1]]

Detect vertical edge

[[-1,-2,-1],
[0, 0, 0],
[1, 2, 1]]

Larger weight near the center pixel

Detect horizontal edge

Image Filtering

- Cohort exercise: calculate the output of the following filtering (you can ignore the boundary condition)

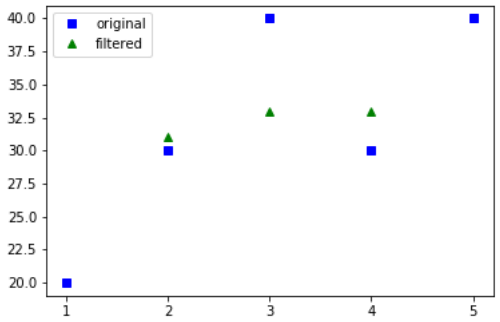
Img =

```
[[30 40 20 30 40]
 [40 20 30 40 30]
 [20 30 40 30 40]
 [30 40 30 40 20]
 [40 30 40 20 30]]
```

kernel =

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

out = [[30,31,33],
[31,33,33],
[33,33,32]]



After filtering, the intensity values are smoother