中山大學
SUN YAT-SEN UNIVERSITY

Lecture 2:Pixels and Filters

Pattern Recognition and Computer Vision

Guanbin Li,
School of Computer Science and Engineering, Sun Yat-Sen University

扫码签到

What we will learn today?

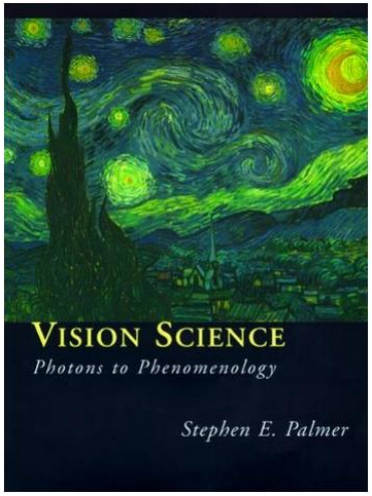
- Color
- Image sampling and quantization
- Image histograms
- Images as functions
- Linear systems (filters)
- Convolution and correlation

What we will learn today?

- Color
- Image sampling and quantization
- Image histograms
- Images as functions
- Linear systems (filters)
- Convolution and correlation

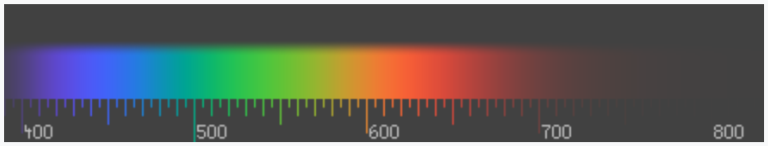
What is color?

- The result of interaction between physical light in the environment and our visual system.
- A psychological property of our visual experiences when we look at objects and lights, not a physical property of those objects or lights.



可见光的光谱

颜色	波长	频率
红色	约625—740 nm	约480—405 THz
橙色	约590—625 nm	约510—480 THz
黄色	约565—590 nm	约530—510 THz
绿色	约500—565 nm	约600—530 THz
蓝色	约485—500 nm	约620—600 THz
靛色	约440—485 nm	约680—620 THz
紫色	约380—440 nm	约790—680 THz



Designed for monitors with gamma

对颜色的辨认是人的眼睛受到电磁波辐射的刺激所引起的视觉神经感受

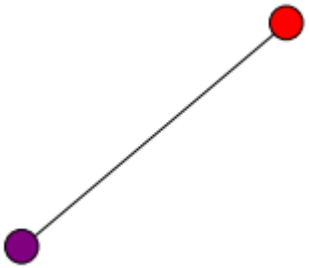
Interaction of light and surfaces

- What is the observed color of any surface under monochromatic light?

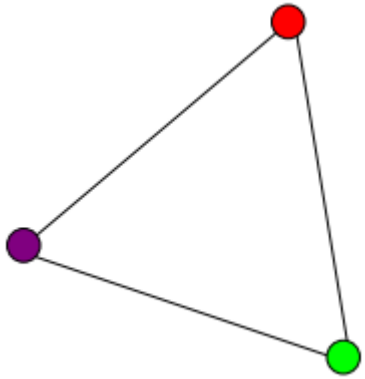
Linear color spaces

三原色学说：视网膜存在三种视锥细胞，分别含有对红、绿、蓝三种光线敏感的视色素，当光线作用于视网膜时，以一定的比例使三种视锥细胞分别产生不同程度的兴奋，传至大脑中枢就产生特定颜色的感觉。

- Defined by a choice of three *primaries.*
- The coordinates of a color are given by the weights of the primaries used to match it.



mixing two lights produces colors that lie along a straight line in color space

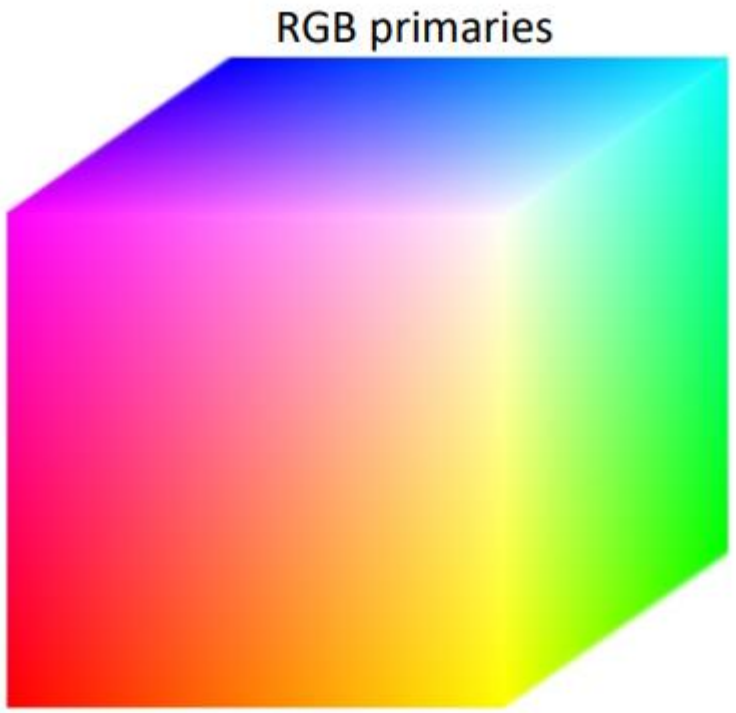


mixing three lights produces colors that lie within the triangle they define in color space

RGB space

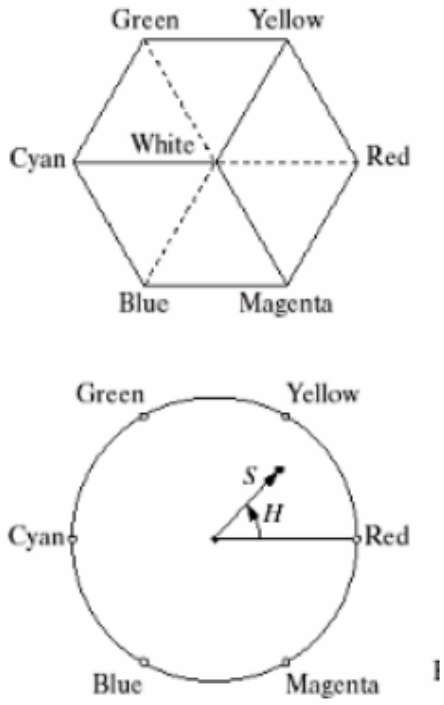
- Primaries are monochromatic lights (for monitors, they correspond to the three types of phosphors)

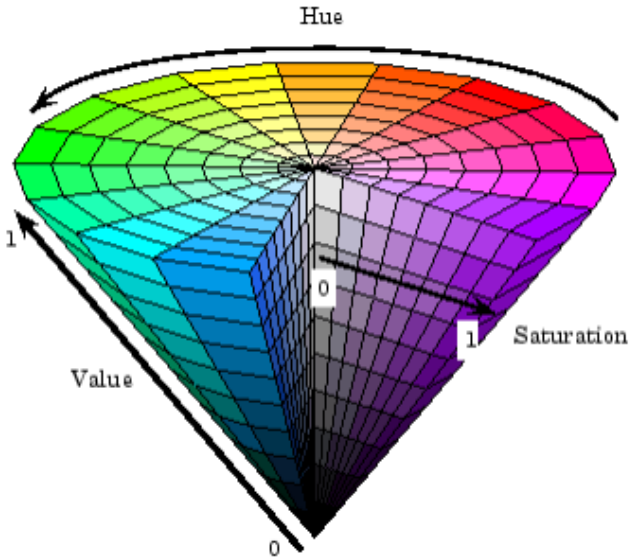
RGB 颜色空间：直观（容易理解）、
三个分量高度相关、均匀性较差

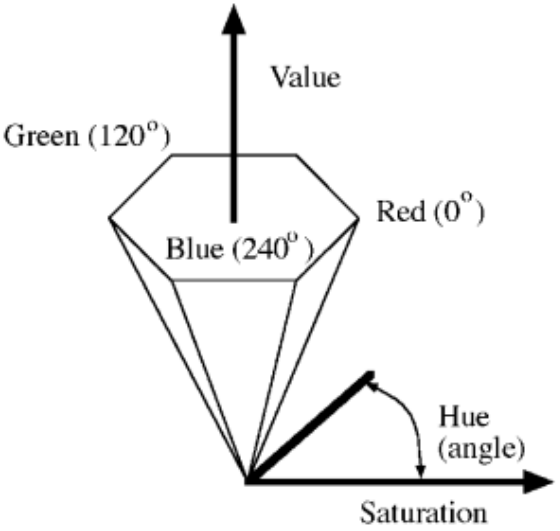


Nonlinear color spaces: HSV

HSV(Hue, Saturation, Value)是根据颜色的直观特性创建的一种颜色空间, 也称六角锥体模型 (Hexcone Model)。



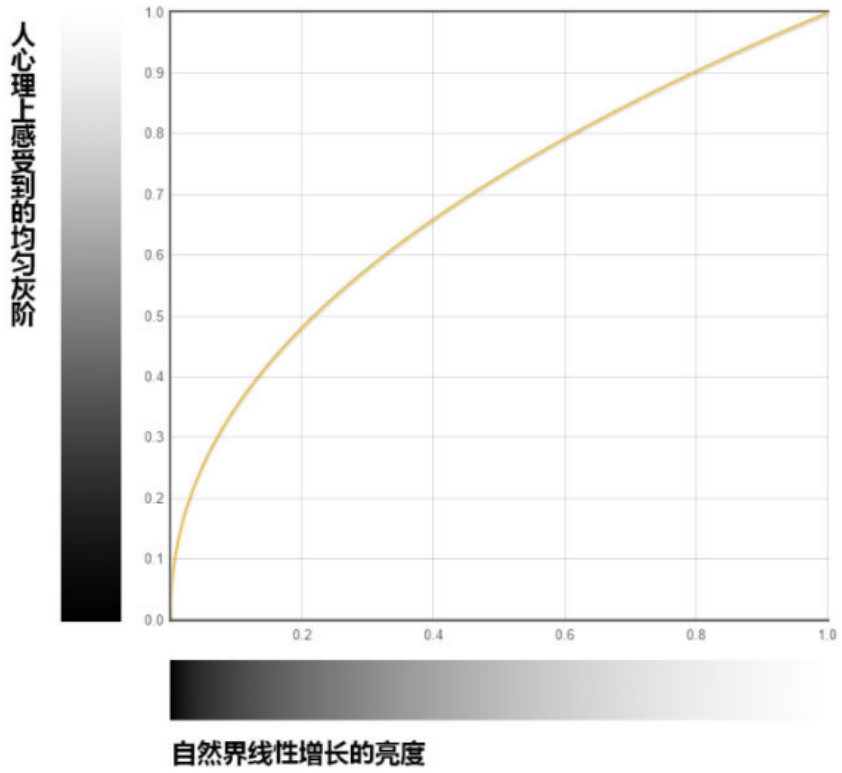


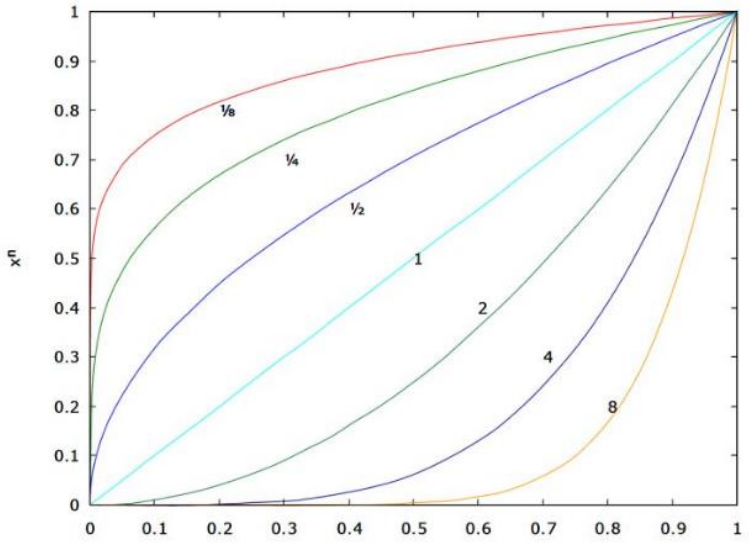


- Perceptually meaningful dimensions: Hue, Saturation, Value (Intensity)
- RGB cube on its vertex

Gamma Space

将自然光以接近人眼感知能力曲线的函数压缩到 8 位图像。

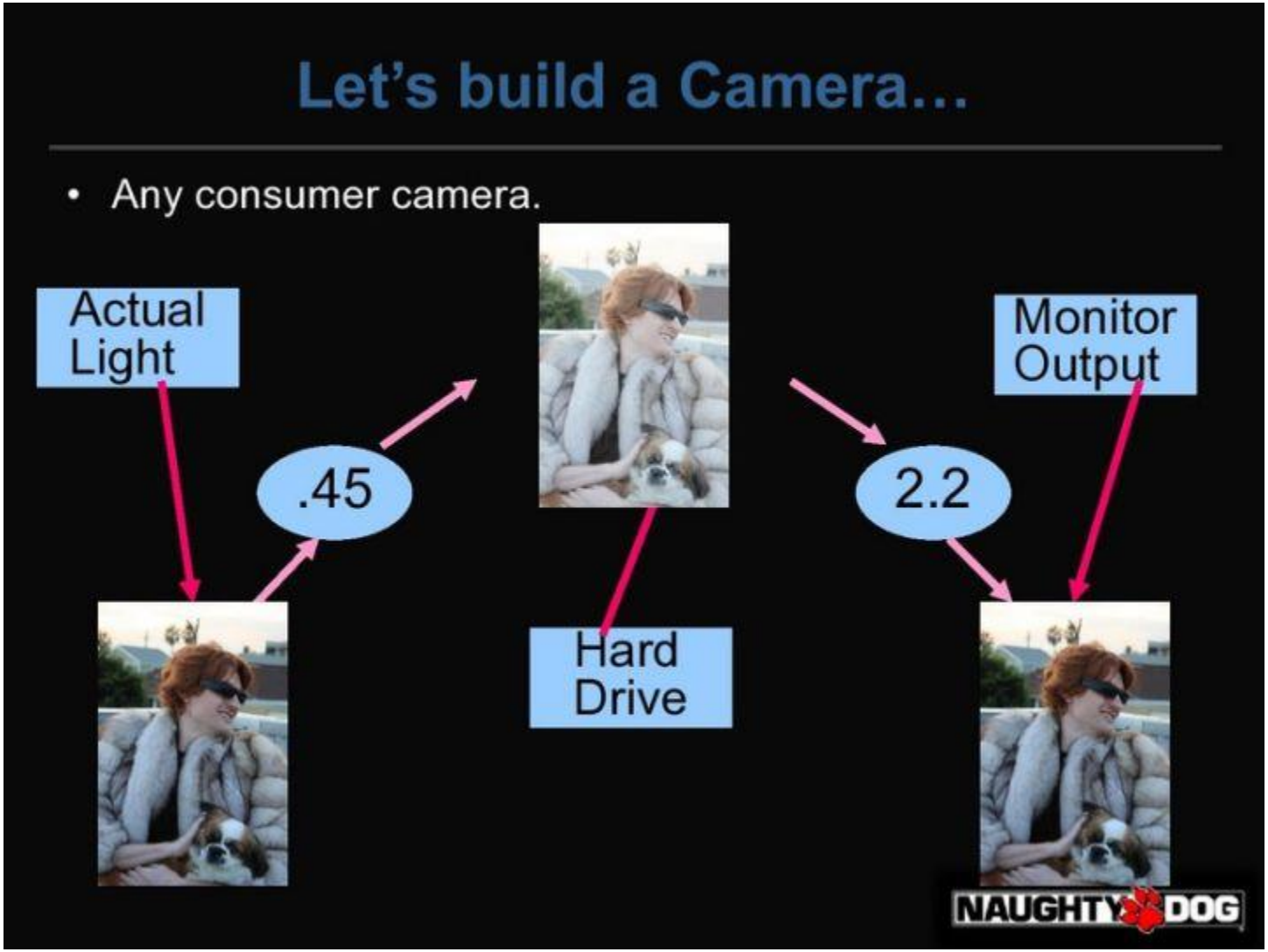




$$V_{out} = V_{in}^{Gamma}$$

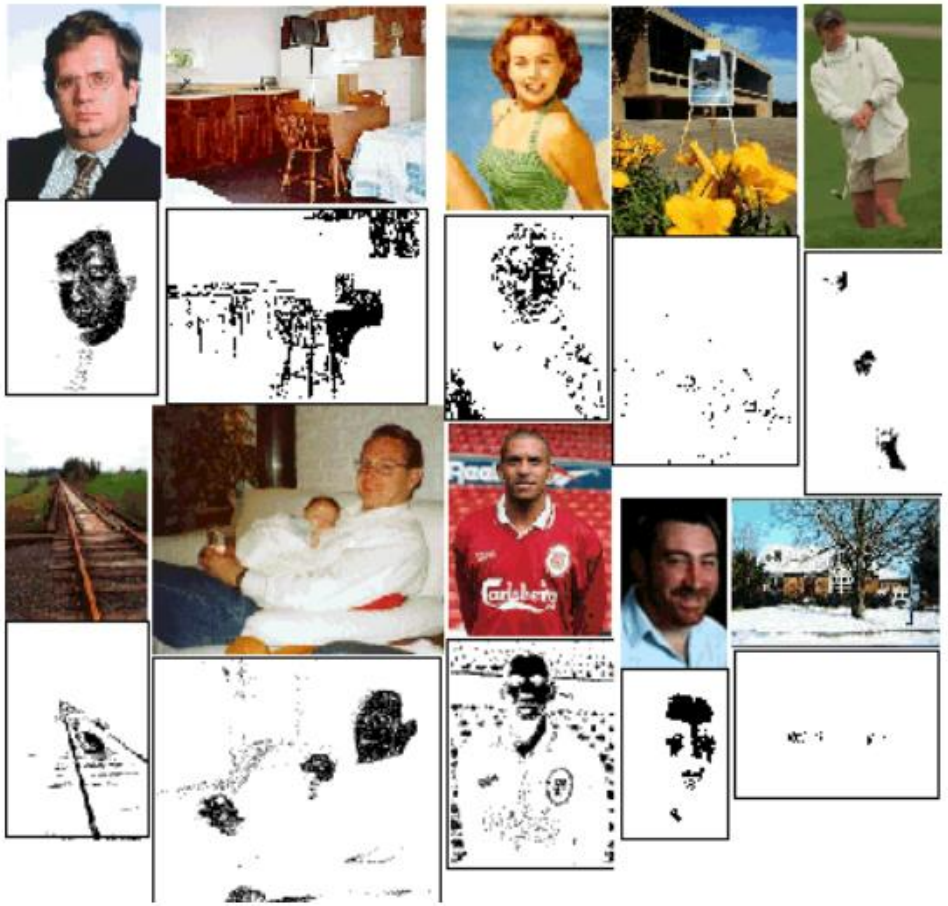
Gamma压缩（Gamma=0.45）
Gamma矫正（Gamma=2.2）

Gamma Space


Let's build a Camera…
• Any consumer camera.
Actual Light
.45
Hard Drive
2.2
Monitor Output
NAUGHTY DOG

Uses of color in computer vision

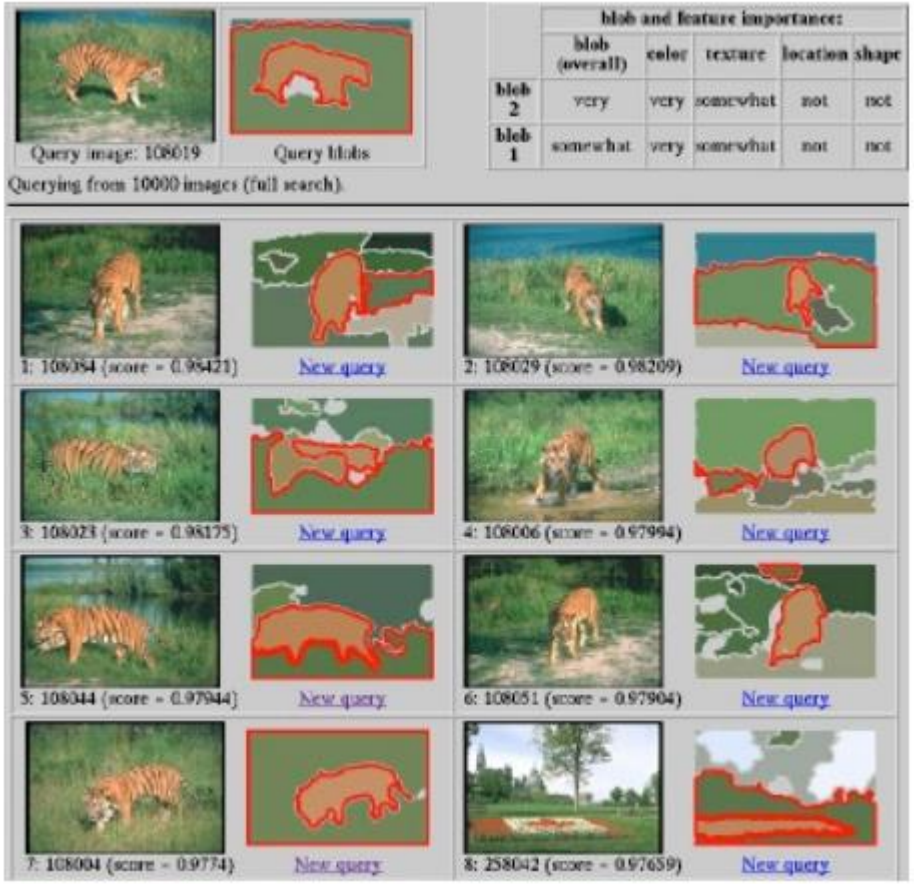
Skin detection



M. Jones and J. Rehg, Statistical Color Models with Application to Skin Detection, IJCV 2002.

Uses of color in computer vision

Image segmentation and retrieval



C. Carson, S. Belongie, H. Greenspan, and Ji. Malik, Blobworld: Image segmentation using Expectation-Maximization and its application to image querying, ICVIS 1999.

What we will learn today?

- Color
- Image sampling and quantization
- Image histograms
- Images as functions
- Linear systems (filters)
- Convolution and correlation

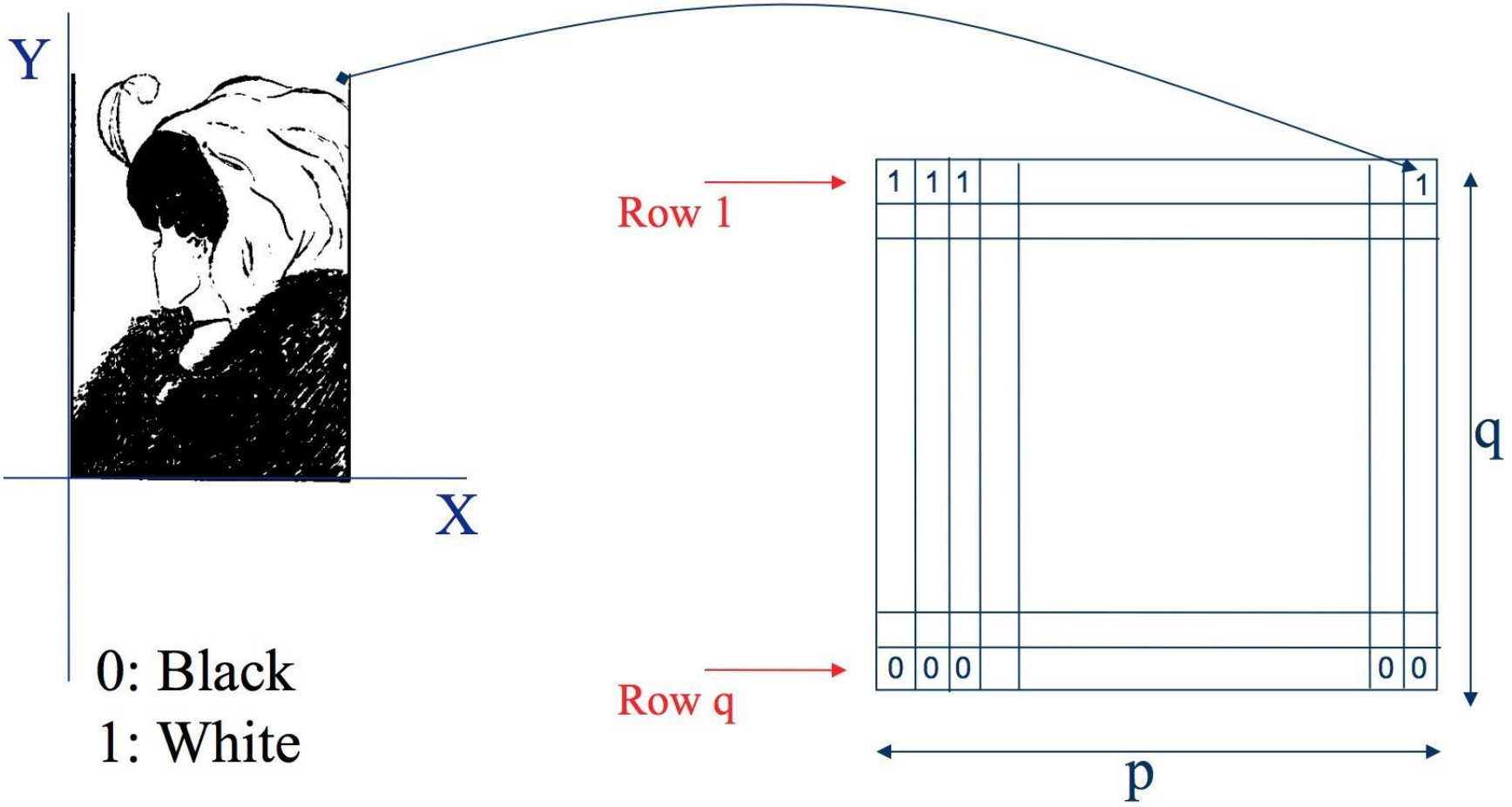
Types of Images

Binary

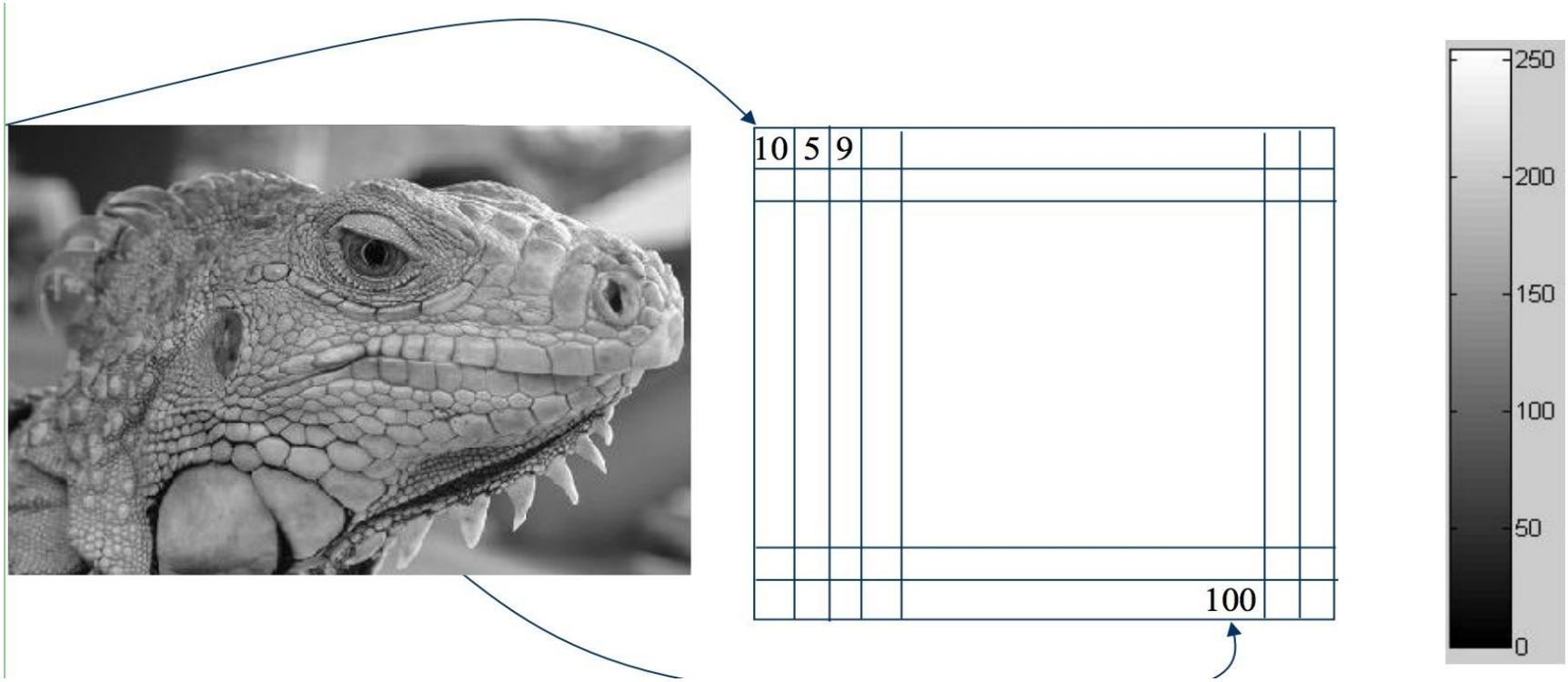
Gray Scale

Color

Binary Image Representation



Gray Scale Image Representation



Color Image Representation

B Channel

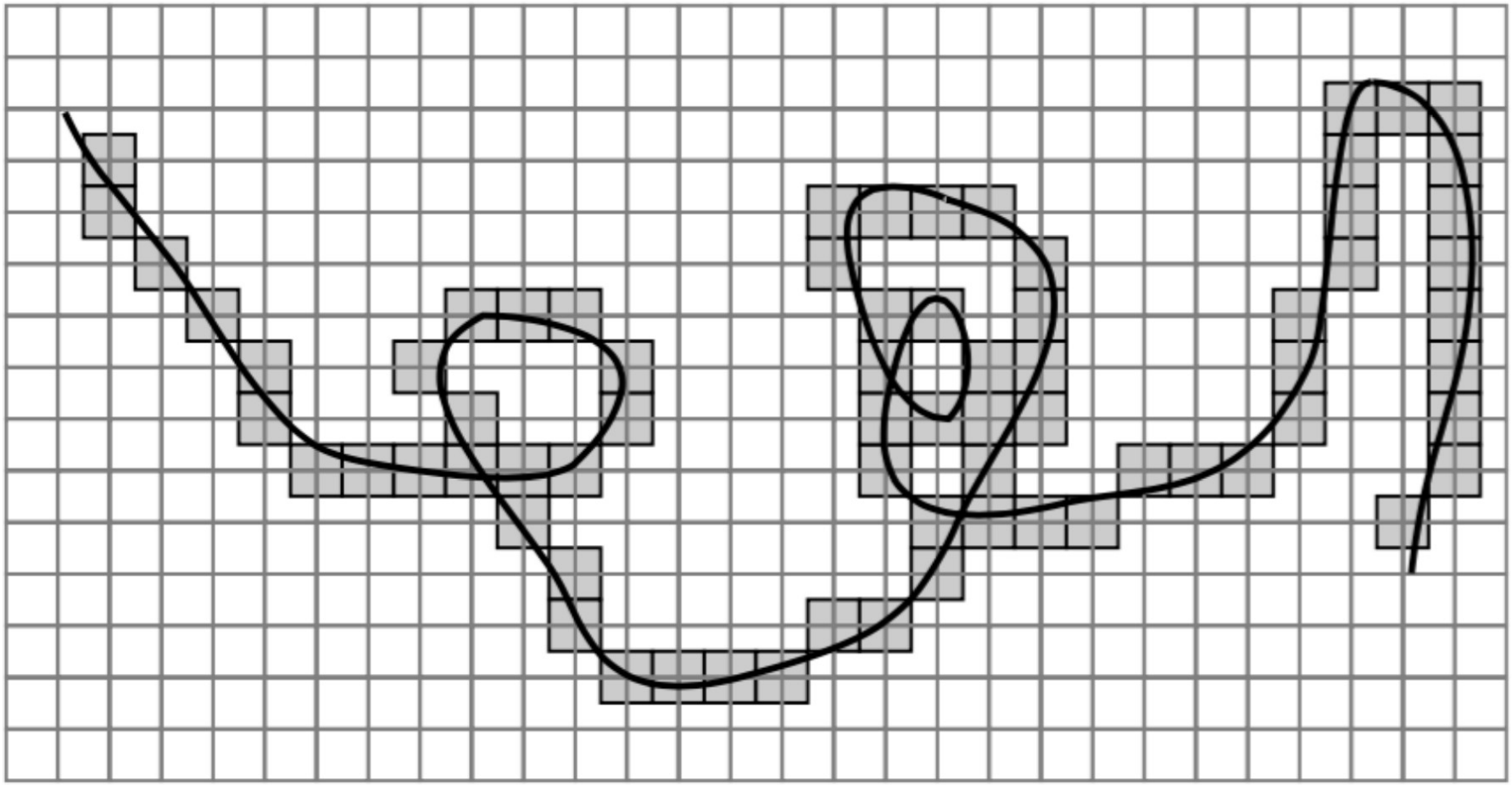
G Channel

R Channel

Images are sampled

- What happens when we zoom into the images we capture?

Errors due Sampling

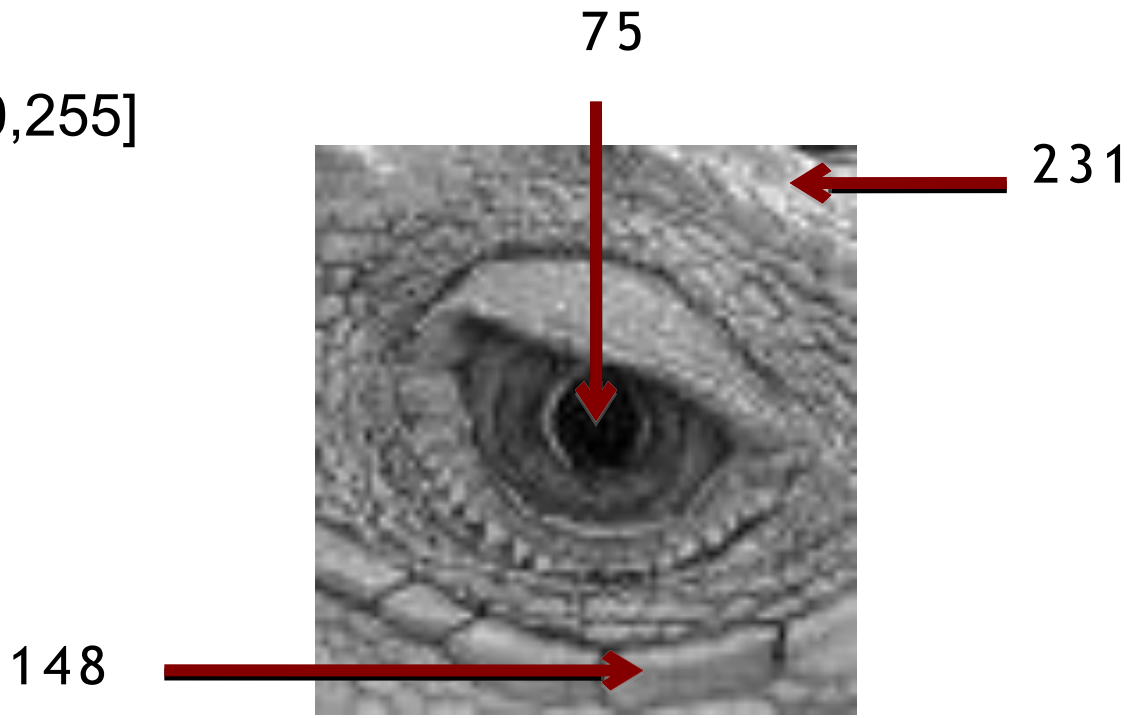


Resolution

- Resolution is a sampling parameter, defined in dots per inch (DPI) or equivalent measures of spatial pixel density.

Images are Sampled and Quantized

- An image contains discrete number of pixels.
  - –Pixel value:
    - “grayscale” (or “intensity”): [0,255]



Images are Sampled and Quantized

- An image contains discrete number of pixels.
  - –Pixel value:
    - “grayscale” (or “intensity”): [0,255]
    - “color”
      - –RGB: [R, G, B]

Images are Sampled and Quantized

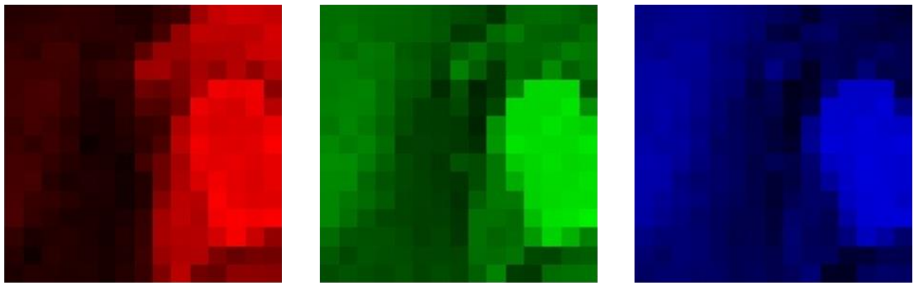
A (color) image is a 3D tensor of numbers.

Images are Sampled and Quantized

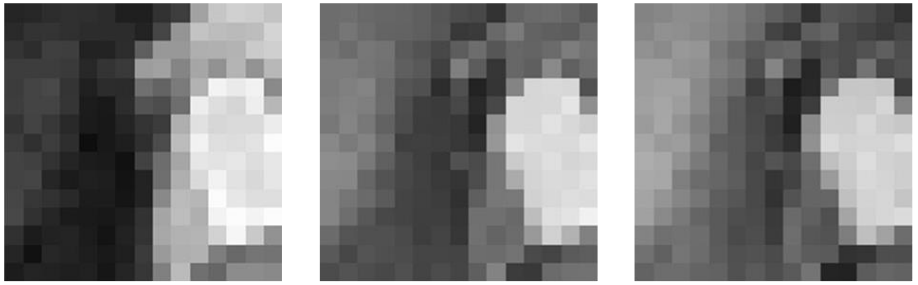
color image patch

How many bits are the intensity values?

red green blue



colorized for visualization



actual intensity values per channel

Each channel is a 2D array of numbers.

Thinking

With this loss of information (from sampling and quantization),

Can we still use images for useful tasks?

What we will learn today?

- Color
- Image sampling and quantization
- Image histograms
- Images as functions
- Linear systems (filters)
- Convolution and correlation

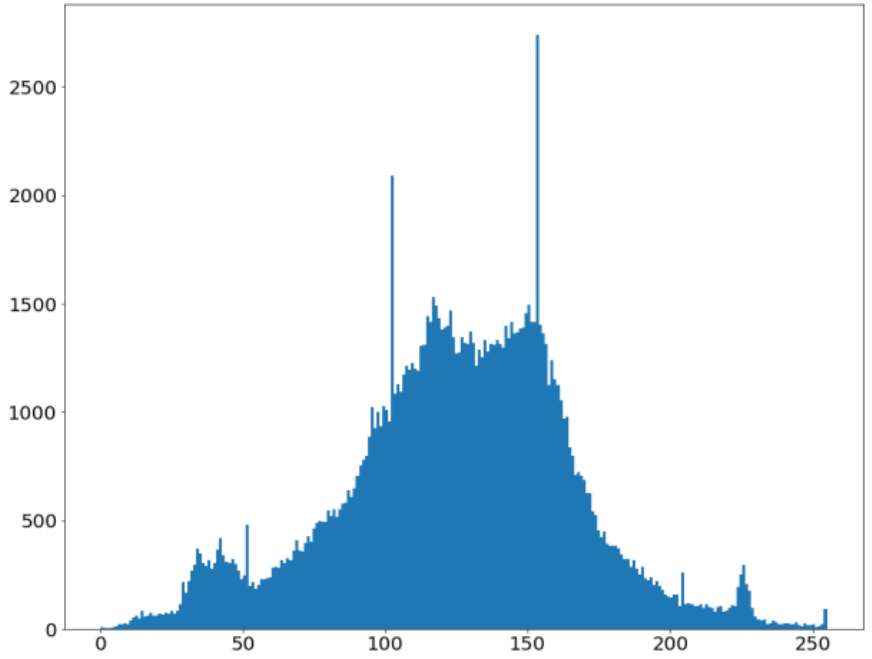
Histograms

- Histogram of an image provides the frequency of the brightness (intensity) value in the image.

```
def histogram(im):
    h = np.zeros(255)
    for row in im.shape[0]:
        for col in im.shape[1]:
            val = im[row, col]
            h[val] += 1
```

Histograms

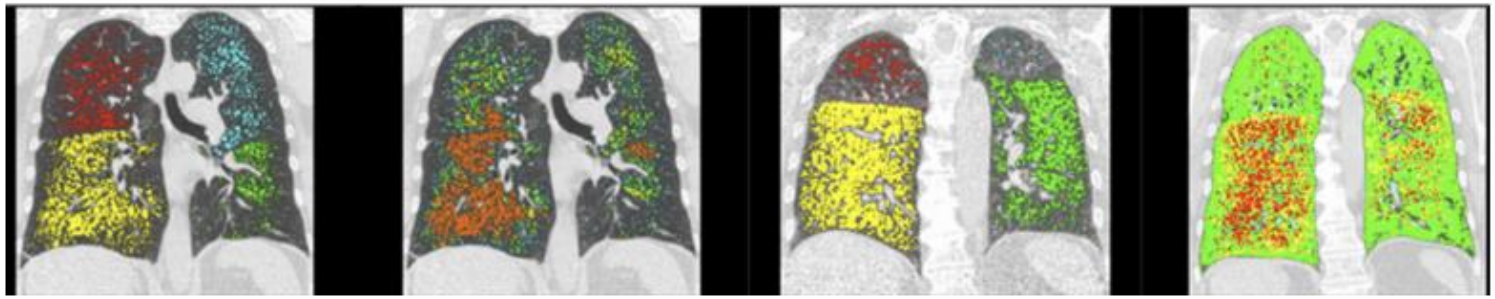
- Histogram captures the distribution of gray levels in the image.

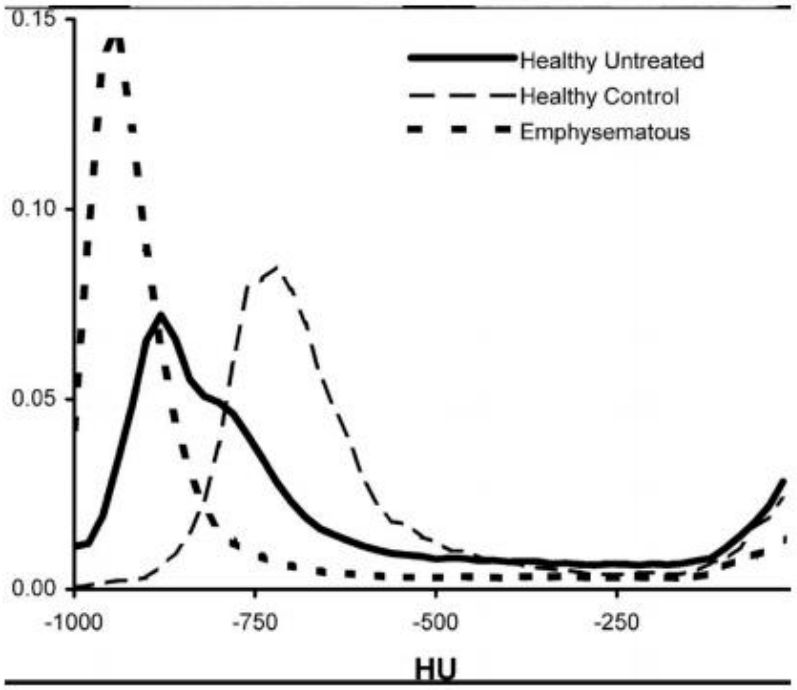


Histograms

- Histogram captures the distribution of gray levels in the image.

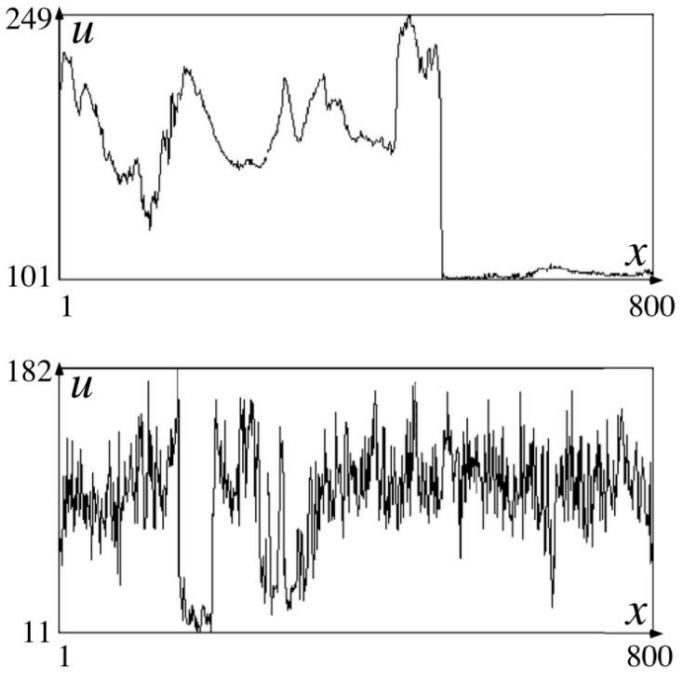
Histograms - use case





Histogram

- Histograms can be used to provide a quantifiable description of what things look like
  - this can be used as input to classifiers



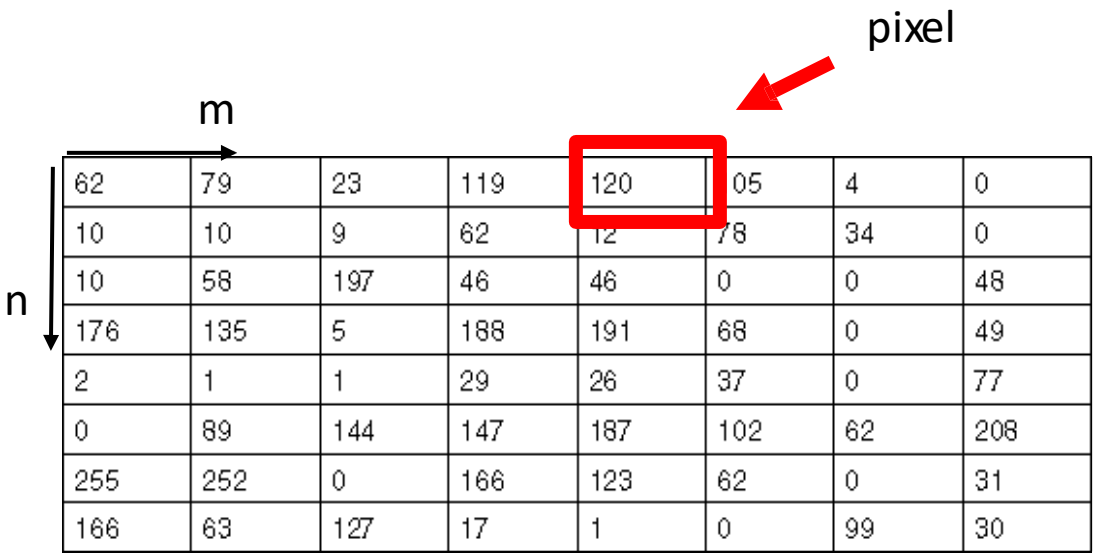
What we will learn today?

- Color
- Image sampling and quantization
- Image histograms
- Images as functions
- Linear systems (filters)
- Convolution and correlation

Images as discrete functions

- Images are usually **digital** (**discrete**):
  – **Sample** the 2D space on a regular grid

- Represented as a matrix of integer values



Images as discrete functions

Cartesian coordinates

$$f[n,m] = \begin{bmatrix} \ddots & \vdots & & \\ f[-1,1] & f[0,1] & f[1,1] & \\ \dots \quad f[-1,0] & \underline{f[0,0]} & f[1,0] & \dots \\ f[-1,-1] & f[0,-1] & f[1,-1] & \\ & \vdots & & \ddots \end{bmatrix}$$

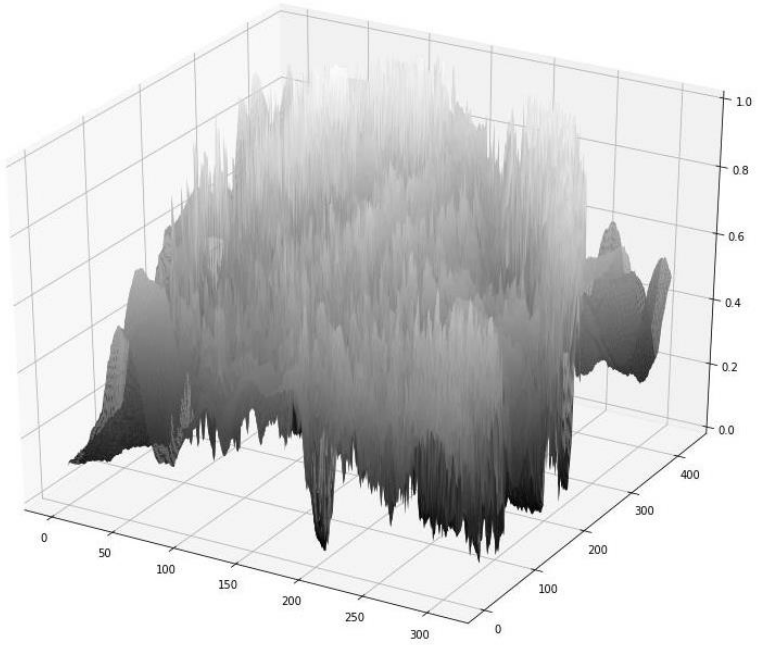
Notation for discrete functions

Images as discrete functions

- **An Image** as a function f from R^2 to R^M:
  - *f [x, y]* gives the **intensity** at position [x, y]
  - Defined over a rectangle, with a finite range:
    f: [a, b] x [c, d] → [0,255]

Domain support

range



Images as discrete functions

- **An Image** as a function f from R^2 to R^M:
  - *f [x, y]* gives the **intensity** at position [x, y]
  - Defined over a rectangle, with a finite range:
    f: [a, b] x [c, d] → [0,255]

Domain support

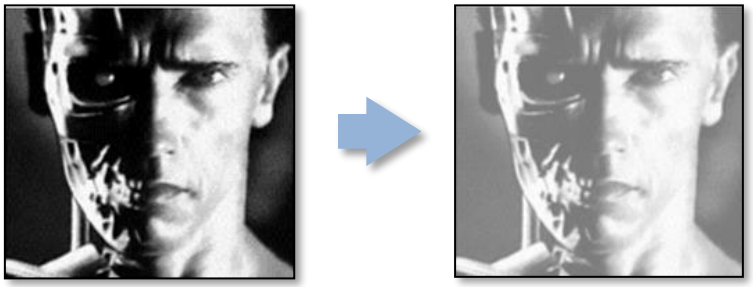
range

- A color image:

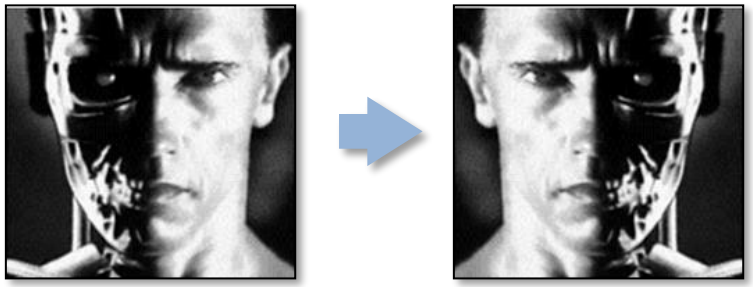
$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

Image transformations

- As with any function, we can apply operators to an image



$g(x,y) = f(x,y) + 20$



$g(x,y) = f(-x,y)$

- Today we'll talk about a special kind of operator, *convolution* (linear filtering)

What we will learn today?

- Color
- Image sampling and quantization
- Image histograms
- Images as functions
- Linear systems (filters)
- Convolution and correlation

Applications of Linear Filters(Systems)

De-nosing

Super-resolution

In-painting

Bertamio et al

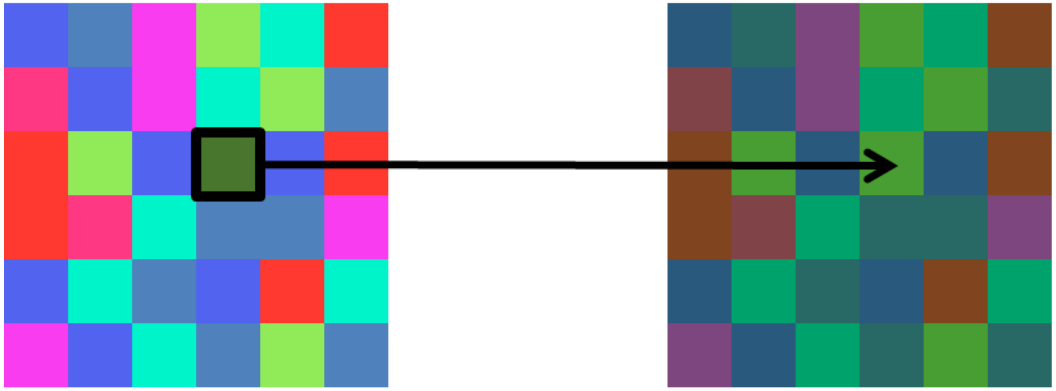
Systems or Filters

- **Filtering:** Forming a new image whose pixel values are transformed from original pixel values
- **Goals:**

1. Extract useful information from images: Features (edges, corners, blobs…)
2. Transform images to modify/enhance image properties: Super-resolution;in-painting; de-noising
3. A key operator in Convolutional Neural Networks

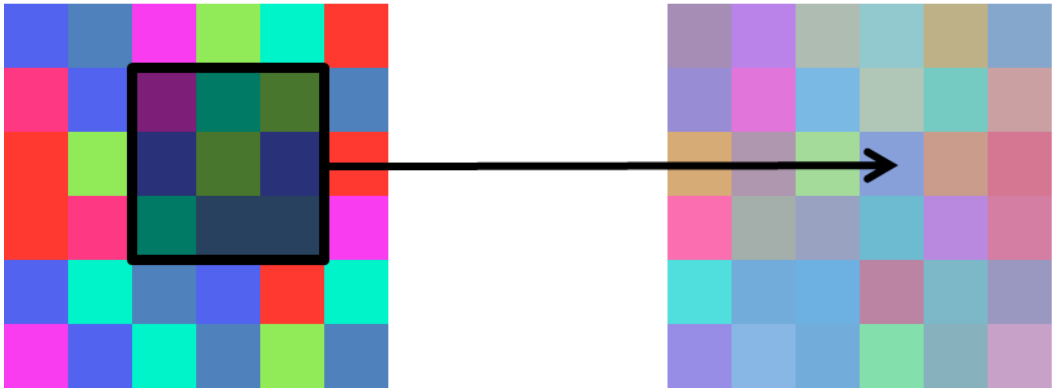
Point Operation and Image filtering

Point Operation



point processing

Neighborhood Operation



“filtering”

Point Operation

original

x

darken

$x - 128$

lower contrast

$\frac{x}{2}$

non-linear lower contrast

$\left(\frac{x}{255}\right)^{1/3} \times 255$

invert

$255 - x$

lighten

$x + 128$

raise contrast

$x \times 2$

non-linear raise contrast

$\left(\frac{x}{255}\right)^{2} \times 255$

Other types of point processing

camera output

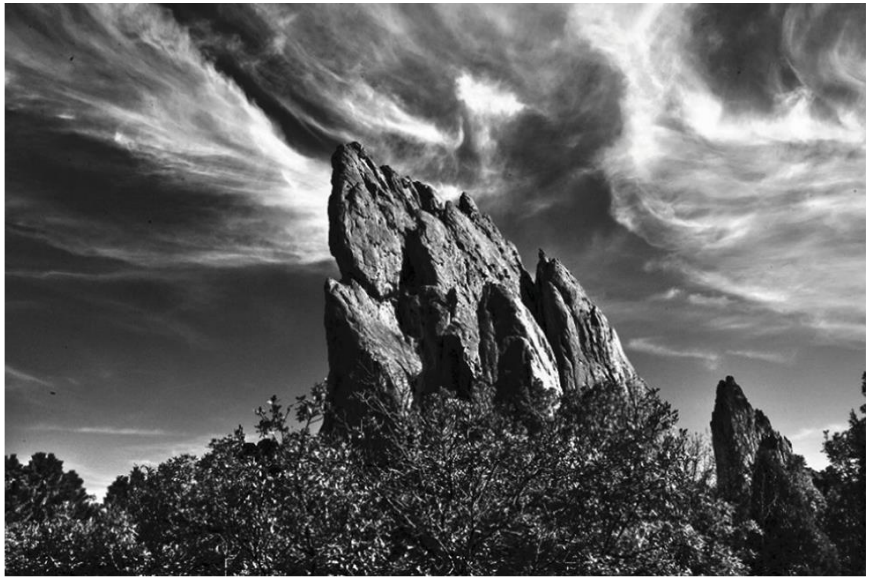
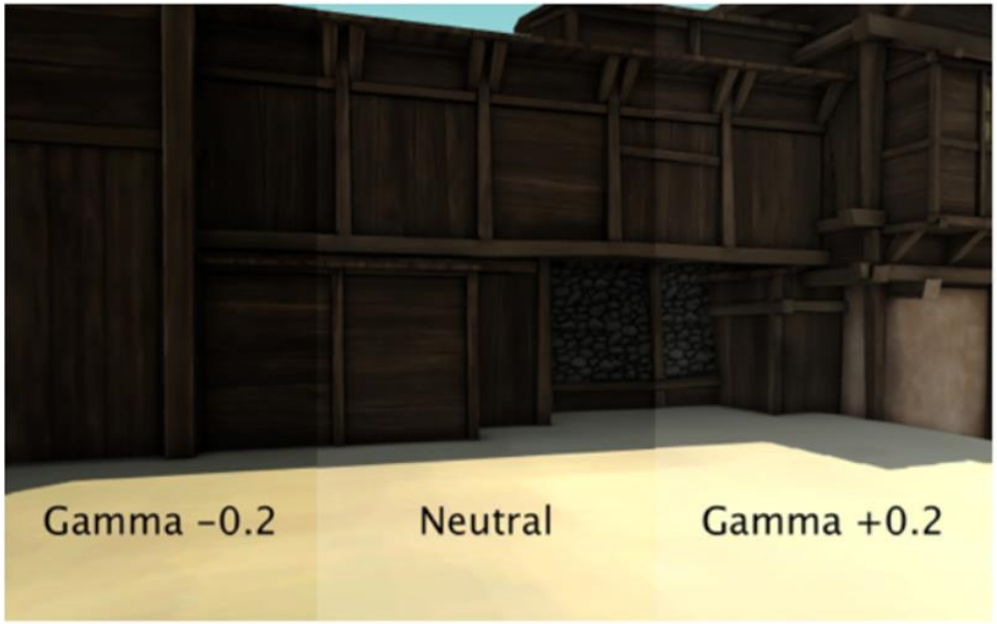


image after stylistic tonemapping

[Bae et al., SIGGRAPH 2006]

Other types of point processing



Systems or Filters

- We define a system as a unit that converts an input function f[n,m] into an output function g[n,m], where [n, m] are the independent variables.
  - In the case of images, [n, m] represents the **spatial position in the image.**

$$f[n,m] \rightarrow \boxed{\text{System } \mathcal{S}} \rightarrow g[n,m]$$

Systems and Filters

- S is the **system operator**, defined as a mapping/assignment that transforms the input f into the output g.

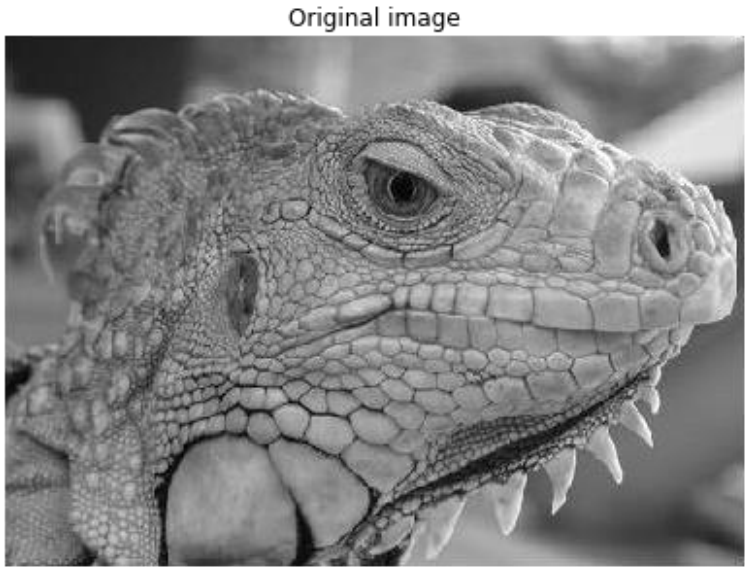
$$f[n,m] \rightarrow \boxed{\text{System } \mathcal{S}} \rightarrow g[n,m]$$

$$g = \mathcal{S}[f], \quad g[n,m] = \mathcal{S}\{f[n,m]\}$$

$$f[n,m] \xrightarrow{\mathcal{S}} g[n,m]$$

Filter example #1: Moving Average

- 2D moving average over a 3 × 3 neighborhood window



Filter example #1: Moving Average

- 2D moving average over a 3 × 3 window of neighborhood

Sum over rows **Sum over columns**

$$g[n,m] = \frac{1}{9}\sum_{k=n-1}^{n+1}\sum_{l=m-1}^{m+1} f[k,l]$$

$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f[n-k, m-l]$$

h

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

Filter example #1: Moving Average

$f[n, m]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[n, m]$

Courtesy of S. Seitz

Filter example #1: Moving Average

$f[n,m]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[n,m]$

	0	10							

Courtesy of S. Seitz

Filter example #1: Moving Average

$f[n, m]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[n, m]$

	0	10	20						

Courtesy of S. Seitz

Filter example #1: Moving Average

$f[n,m]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[n,m]$

	0	10	20	30					

Courtesy of S. Seitz

Filter example #1: Moving Average

$f[n, m]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[n, m]$

	0	10	20	30	30				

Courtesy of S. Seitz

Filter example #1: Moving Average

$f[n, m]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[n, m]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Courtesy of S. Seitz

Filter example #1: Moving Average

- **In summary:**

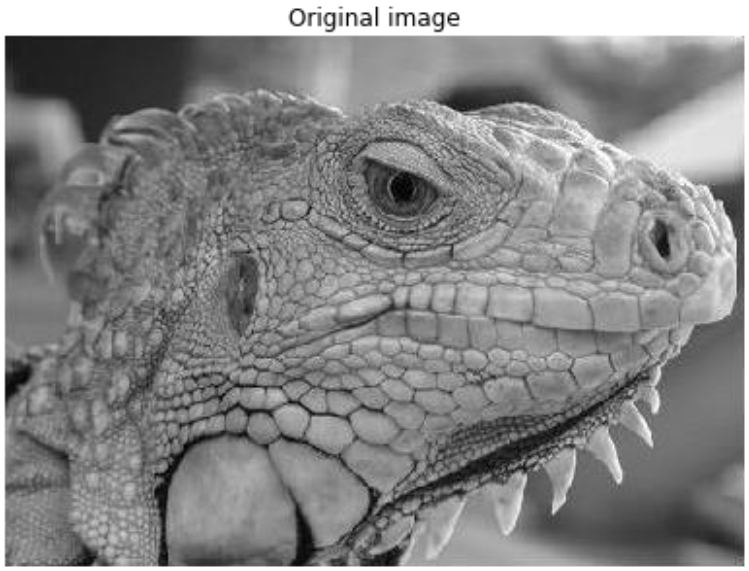
1. This filter “transforms” each pixel value into the average value of its neighborhood.
2. Achieve smoothing effect (remove sharp features)

$$h[\cdot,\cdot]$$

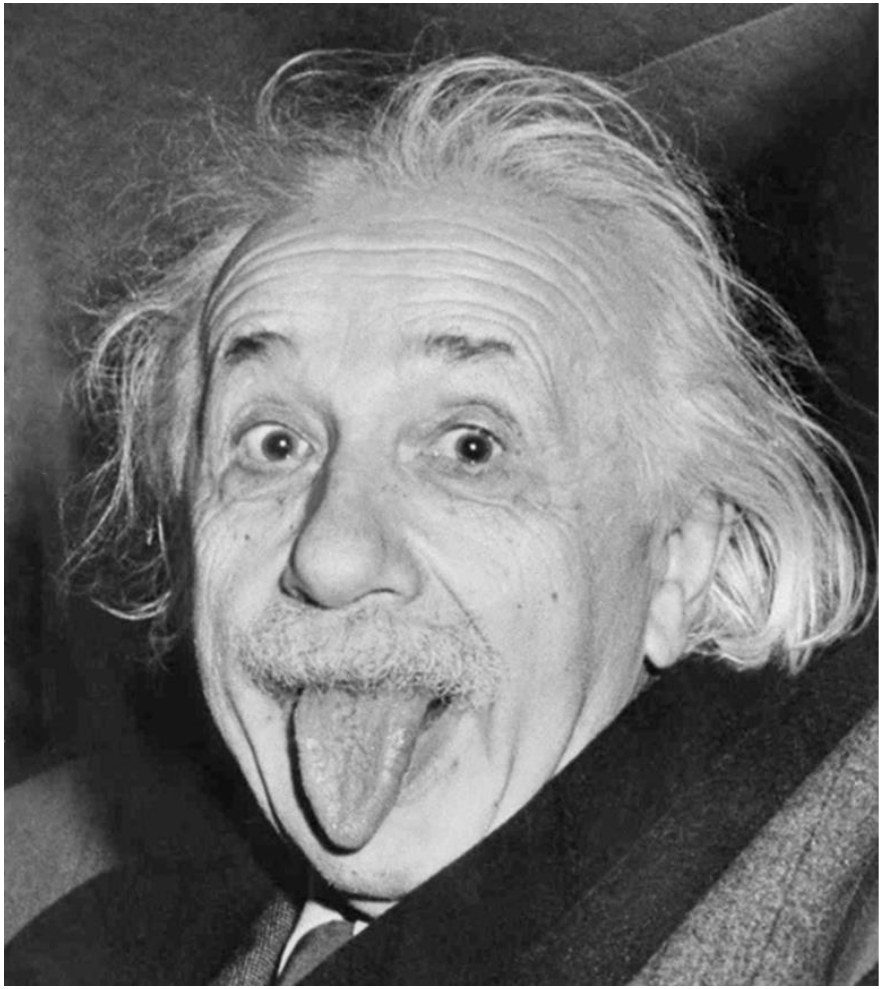
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

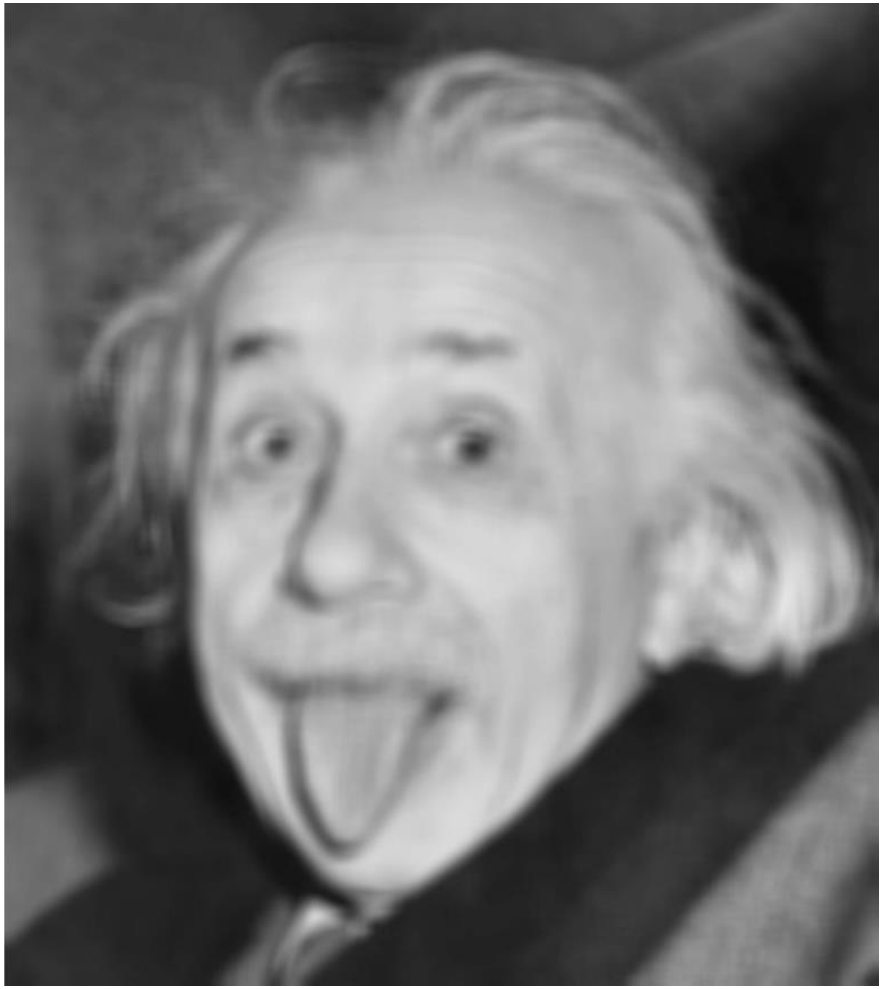
Filter example #1: Moving Average

- 2D moving average over a 3 × 3 neighborhood window

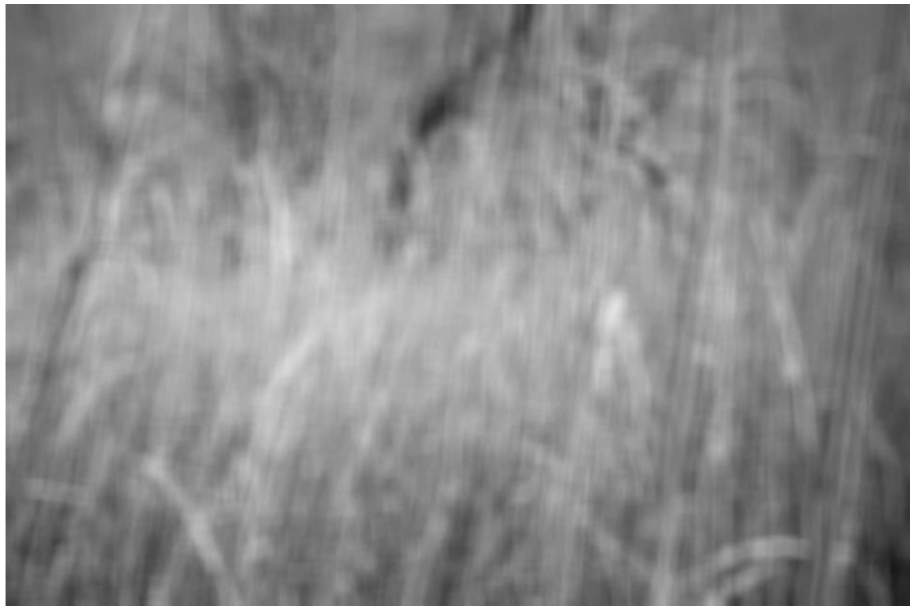


Some more realistic examples





Some more realistic examples



Some more realistic examples

A more general version

10	5	3
4	5	1
1	1	7

Local image data

	7	

Kernel / filter

$$S[f](m,n) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w(i,j) f(m+i, n+j)$$

A more general version

0	10	5	7	0
5	11	6	8	3
9	22	4	5	1
2	9	14	6	7
3	10	15	12	9

Local image data

		7		

Kernel size = 2k+1

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

A more general version

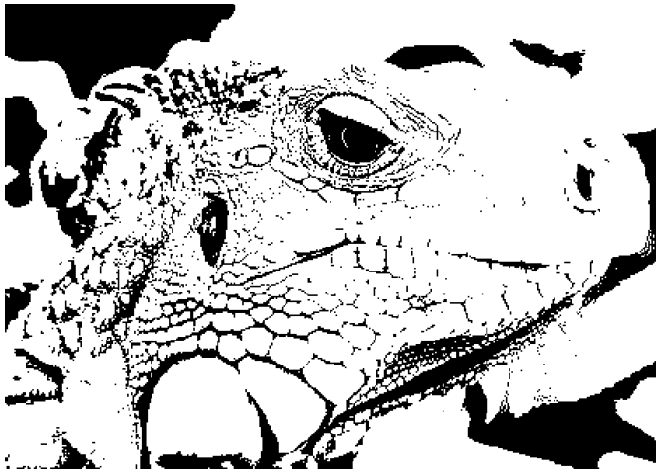
$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

- w(i,j) = $1/(2k+1)^2$ for mean filter
- If w(i,j) >=0 and sum to 1, *weighted mean*
- But w(i,j) can be *arbitrary real numbers!*

Filter example #2: Image Segmentation

- Image segmentation based on a simple threshold:

$$g[n,m] = \begin{cases} 255, & f[n,m] > 100 \\ 0, & \text{otherwise.} \end{cases}$$



Properties of systems

- Amplitude properties:
  - Additivity

$$S[f_i[n,m] + f_j[n,m]] = S[f_i[n,m]] + S[f_j[n,m]]$$

  - Homogeneity

$$S[\alpha f_i[n,m]] = \alpha S[f_i[n,m]]$$

  - Superposition

$$S[\alpha f_i[n,m] + \beta f_j[n,m]] = \alpha S[f_i[n,m]] + \beta S[f_j[n,m]]$$

  - Stability

$$|f[n,m]| \leq k \implies |g[n,m]| \leq ck$$

  - Invertibility

$$S^{-1}[S[f_i[n,m]]] = f[n,m]$$

- Spatial properties

$$f[n-n_0, m-m_0] \xrightarrow{\mathcal{S}} g[n-n_0, m-m_0]$$

Shift-Invariant (SI) Systems

- We have $f[n,m] \xrightarrow{\mathcal{S}} g[n,m]$
  - if for every input image f[n,m] and shifts n0, m0

$$f[n-n_0, m-m_0] \xrightarrow{\mathcal{S}} g[n-n_0, m-m_0]$$

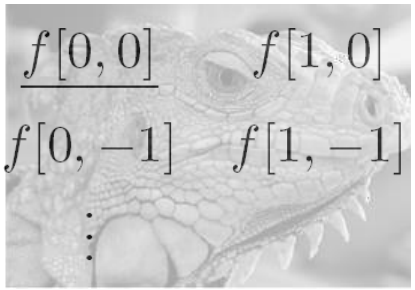
  - then, S is Shift Invariant

Shift-Invariant (SI) Systems

- What does shifting an image look like?

Cartesian coordinates

$$f[n,m] = \begin{bmatrix} \ddots & & \vdots & & \\ & f[-1,1] & f[0,1] & f[1,1] & \\ \dots & f[-1,0] & \underline{f[0,0]} & f[1,0] & \dots \\ & f[-1,-1] & f[0,-1] & f[1,-1] & \\ & & \vdots & & \ddots \end{bmatrix}$$



Shift-Invariant (SI) Systems

- Is the moving average system shift invariant?

$$g[n,m] = \frac{1}{9} \sum_{k=n-1}^{n+1} \sum_{l=m-1}^{m+1} f[k,l]$$

$\mathrm{f}[n,m]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$\mathrm{g}[n,m]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Shift-Invariant (SI) Systems

- Is the moving average system shift invariant?
  - Let's start with passing f through our system

$$f[n,m] \xrightarrow{S} g[n,m]$$

$$g[n,m] = \frac{1}{9}\sum_{k=n-1}^{n+1}\sum_{l=m-1}^{m+1} f[k,l] = \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f[n-k,m-l]$$

  - Now, let's see what we get when we pass in a shifted version of the input f:

$$f[n-n_0,m-m_0] \xrightarrow{S} \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f[(n-n_0)-k,(m-m_0)-l]$$

$$= g[n-n_0,m-m_0]$$

Yes!

Linear Systems (filters)

$$f[n,m] \rightarrow \boxed{\text{System } \mathcal{S}} \rightarrow g[n,m]$$

- Linear filtering:

  – Form a new image whose pixels are a weighted sum of original pixel values

  – Use the same set of weights at each point

- S is a linear system (filter) if and only if it satisfies

$$S\{\alpha f_1[n,m] + \beta f_2[n,m]\} = \alpha S\{f_1[n,m]\} + \beta S\{f_2[n,m]\}$$

superposition property

Linear Systems (filters)

$$S\{\alpha f_1[n,m] + \beta f_2[n,m]\} = \alpha S\{f_1[n,m]\} + \beta S\{f_2[n,m]\}$$

- Is the moving average a linear system? Yes!

- Is thresholding a linear system? No!

–We could have f1[n,m] + f2[n,m] > T, when f1[n,m] < T and f2[n,m] < T

Linear Shift Invariant (LSI) systems

- An LSI system satisfies two properties:
  - Superposition property

$$S\{\alpha f_1[n,m] + \beta f_2[n,m]\} = \alpha S\{f_1[n,m]\} + \beta S\{f_2[n,m]\}$$

  - Shift invariance

$$f[n-n_0, m-m_0] \xrightarrow{\mathcal{S}} g[n-n_0, m-m_0]$$

Linear Shift Invariant (LSI) systems

- moving average system is **shift invariant** and **linear**
  - Why are linear shift invariant systems important?

Our visual system is a

linear shift invariant system

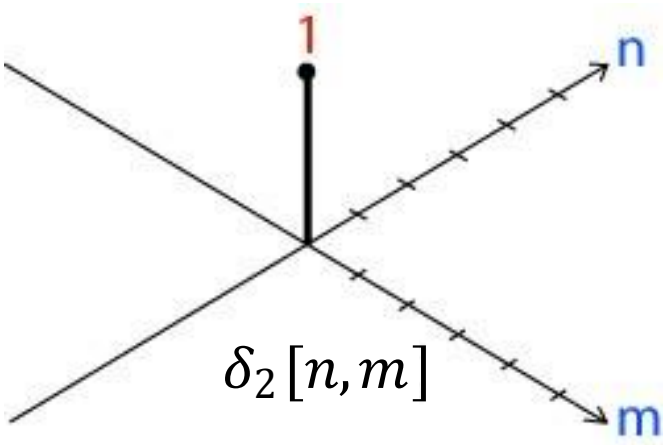
  - We are going to use this as an example to dive into interesting properties about linear shift-invariant systems.

Linear Shift Invariant (LSI) systems

- 2D impulse function $\delta_2[n,m]$

1. 1 at [0,0].
2. 0 everywhere else

	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	
	0	0	0	0	0	0	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	0	0	0	0	



Linear Shift Invariant (LSI) systems

- Impulse response to the moving average filter
  - Recall the expression for our 3x3 moving average filter:

$$f[n,m] \xrightarrow{S} \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} f[n-k, m-l]$$

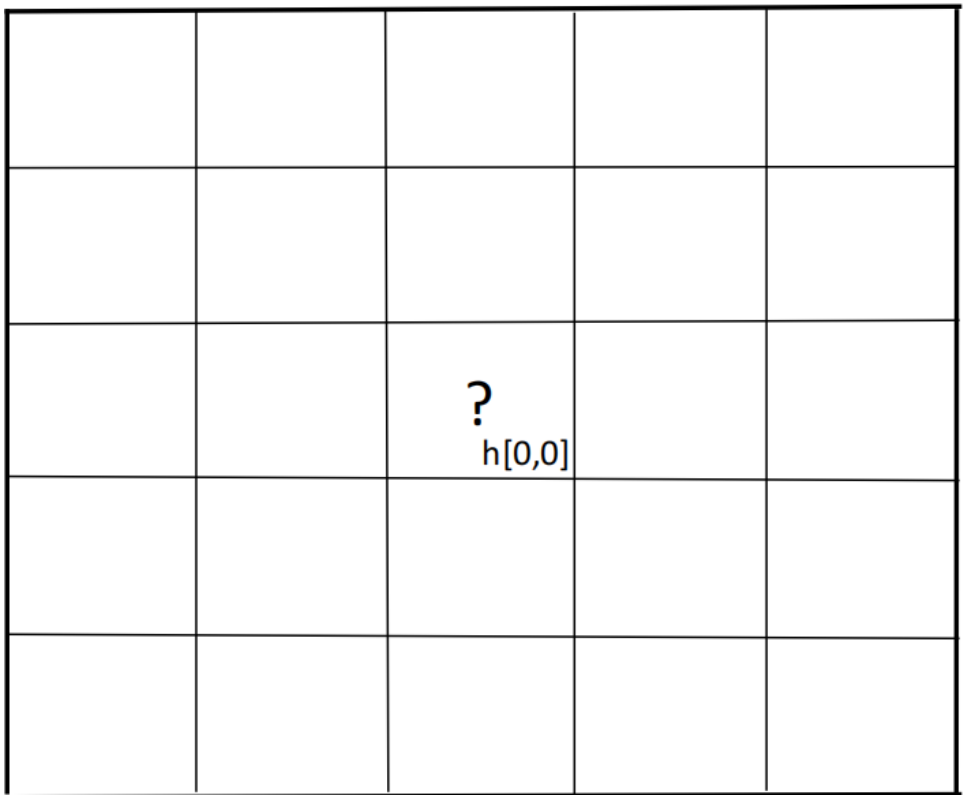
  - We can use it to obtain an expression for the impulse response

$$\delta_2[n,m] \xrightarrow{S} h[n,m]$$

$$= \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} \delta_2[n-k, m-l]$$

Linear Shift Invariant (LSI) systems

- Impulse response to the moving average filter

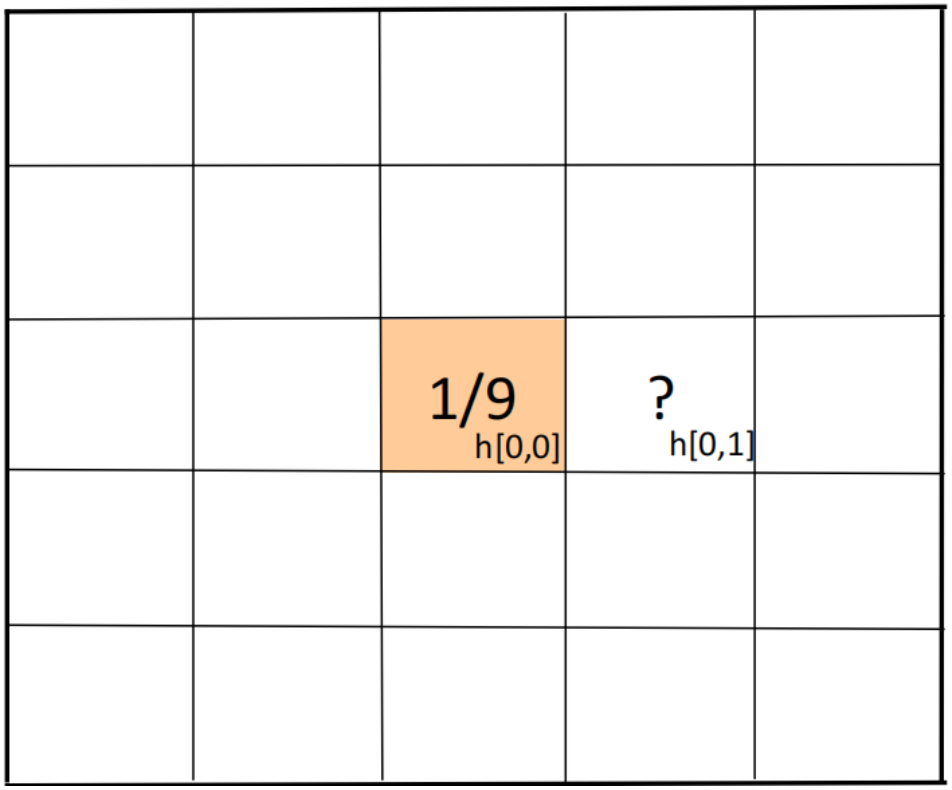


$$\delta_2 \xrightarrow{S} h[n,m]$$

$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1}\delta_2[n-k,m-l]$$

Linear Shift Invariant (LSI) systems

- Impulse response to the moving average filter

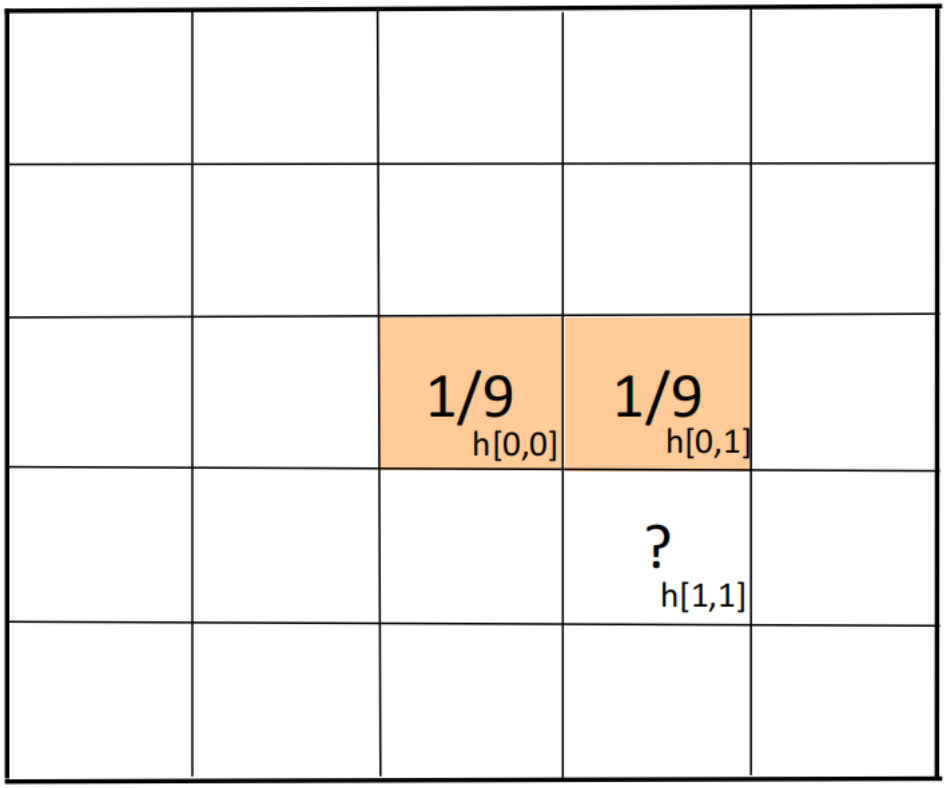


$$\delta_2 \xrightarrow{S} h[n,m]$$

$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1}\delta_2[n-k,m-l]$$

Linear Shift Invariant (LSI) systems

- Impulse response to the moving average filter

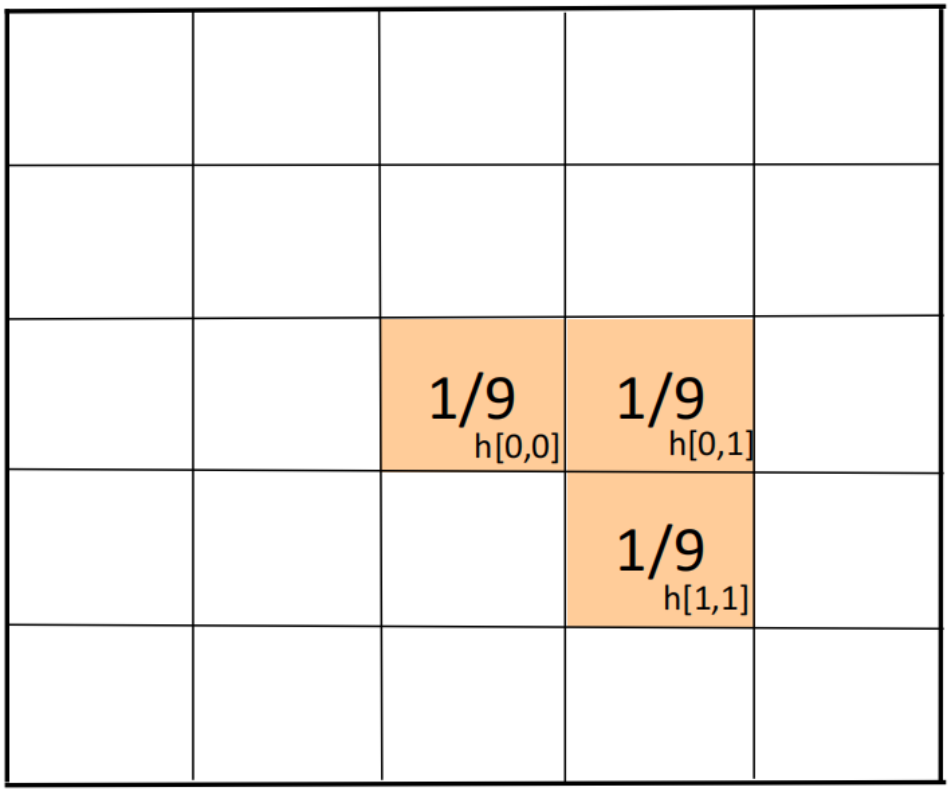


$$\delta_2 \xrightarrow{S} h[n,m]$$

$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1}\delta_2[n-k,m-l]$$

Linear Shift Invariant (LSI) systems

- Impulse response to the moving average filter

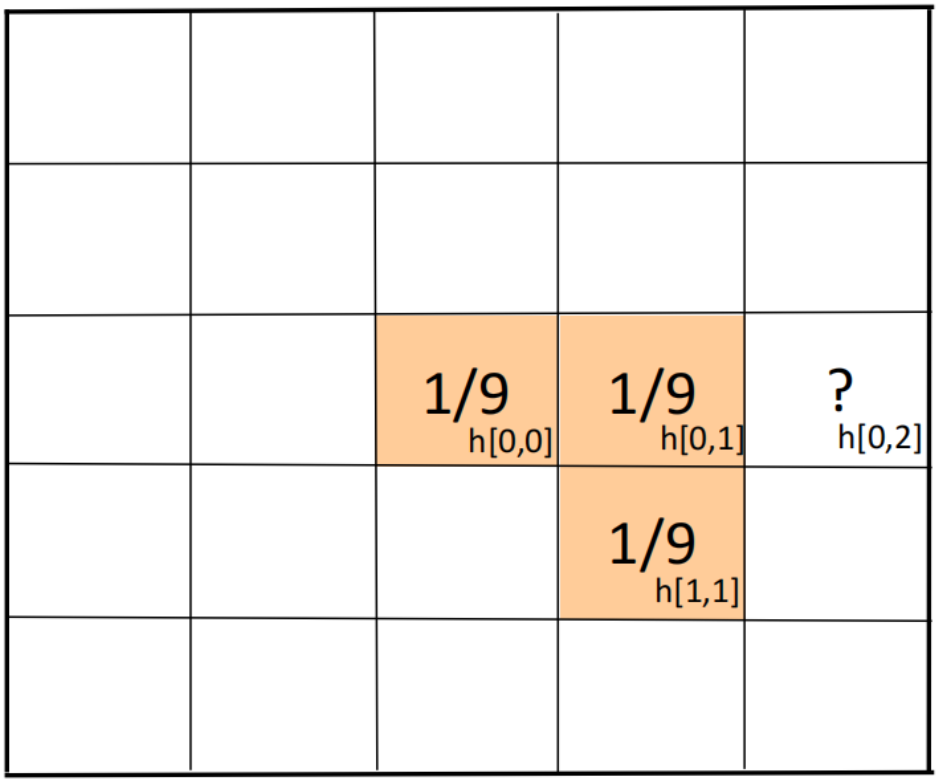


$$\delta_2 \xrightarrow{S} h[n,m]$$

$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1}\delta_2[n-k,m-l]$$

Linear Shift Invariant (LSI) systems

- Impulse response to the moving average filter

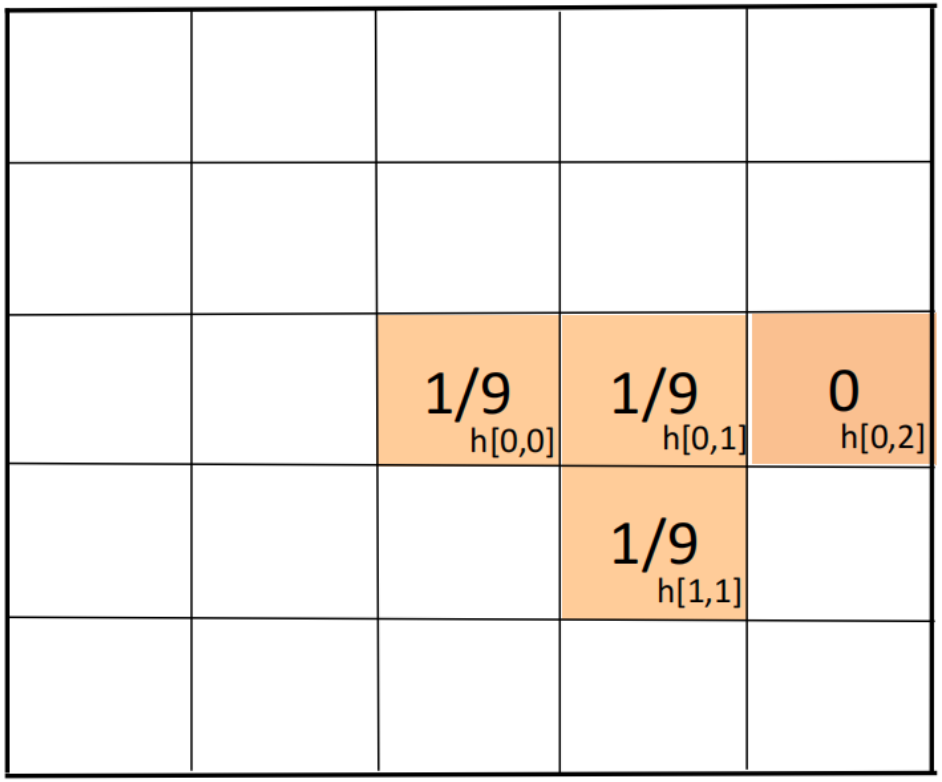


$$\delta_2 \xrightarrow{S} h[n,m]$$

$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1}\delta_2[n-k,m-l]$$

Linear Shift Invariant (LSI) systems

- Impulse response to the moving average filter



$$\delta_2 \xrightarrow{S} h[n,m]$$

$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1}\delta_2[n-k,m-l]$$

Linear Shift Invariant (LSI) systems

- Impulse response to the moving average filter

0	0	0	0	0
0	1/9 h[-1,-1]	1/9	1/9	0
0	1/9	1/9 h[0,0]	1/9 h[0,1]	0 h[0,2]
0	1/9	1/9	1/9 h[1,1]	0
0	0	0	0	0

$$\delta_2 \xrightarrow{S} g[n,m]$$

$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1}\delta_2[n-k,m-l]$$

Linear Shift Invariant (LSI) systems

- Impulse response of the 3 by 3 moving average filter

$$h[n,m] = \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1}\delta_2[n-k,m-l]$$

$$= \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

h

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Linear Shift Invariant (LSI) systems

- An LSI system is completely specified by its impulse response.

$$\delta_2[n,m] \xrightarrow{S} h[n,m]$$

Pass in an impulse function Record its response

- By passing an impulse function into an LSI system, we get it's impulse response.

Linear Shift Invariant (LSI) systems

- Key idea: write down f as a sum of impulses
  - Let's say our input f is a 3x3 image:

f[0,0]	f[0,1]	f[1,1]
f[1,0]	f[1,1]	f[1,2]
f[2,0]	f[2,1]	f[2,2]

=

f[0,0]	0	0
0	0	0
0	0	0

+

0	f[0,1]	0
0	0	0
0	0	0

+ ... +

0	0	0
0	0	0
0	0	f[2,2]

=

f[0,0] *

1	0	0
0	0	0
0	0	0

+

f[0,1] *

0	1	0
0	0	0
0	0	0

+ ... + f[2,2]*

0	0	0
0	0	0
0	0	1

$$= f[0,0] \cdot \delta_2[n,m] + f[0,1] \cdot \delta_2[n,m-1] + \ldots + f[2,2] \cdot \delta[n-2,n-2]$$

Linear Shift Invariant (LSI) systems

- 3 properties we need:

1. We know what happens when we send a delta function through a LSI system:

$$\delta_2[n,m] \rightarrow \boxed{\text{System } \mathcal{S}} \rightarrow h[n,m]$$

2. We also know that LSI systems shift the output if the input is shifted:

$$\delta_2[n-k,m-l] \rightarrow \boxed{\text{System } \mathcal{S}} \rightarrow h[n-k,m-l]$$

3. Finally, the superposition principle:

$$S[\,\alpha f_i[n,m] + \beta f_j[k,l]\,] = \alpha\, S[\,f_i[n,m]] + \beta\, S[\,f_j[k,l]\,]$$

Linear Shift Invariant (LSI) systems

- We have:

$$f[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot \delta_2[n-k, m-l]$$

$$\xrightarrow{S} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot S\{\delta_2[n-k, m-l]\}$$

- Recall, by the shift invariance property that:

$$S\{\delta_2[n-k, m-l]\} = h[n-k, m-l]$$

Linear Shift Invariant (LSI) systems

- We have:

$$f[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot \delta_2[n-k, m-l]$$

$$\xrightarrow{S} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot S\{\delta_2[n-k, m-l]\}$$

- Which means,

$$f[n,m] \xrightarrow{S} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$

Linear Shift Invariant (LSI) systems

- An LSI system is completely specified by its impulse response.
  - For any input f, we can compute the output g in terms of the impulse response h.

$$f[n,m] \xrightarrow{S} g[n,m]$$

$$f[n,m] \xrightarrow{S} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$

$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$

Discrete Convolution

$$g[n,m] = f[n,m] * h[n,m]$$

Summary

- Systems and Filters
- LSI systems and convolution

What we will learn today?

- Color
- Image sampling and quantization
- Image histograms
- Images as functions
- Linear systems (filters)
- Convolution and correlation

What we will learn today?

- **Convolution**
- **Correlation**

– Some background reading:

Forsyth and Ponce, Computer Vision, Chapter 7

Convolution for 1D continuous signals

Definition of filtering as convolution:

notice the flip

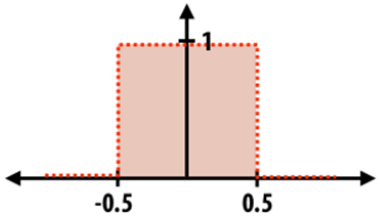
$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy$$

filtered signal — filter — input signal

Consider the box filter example:

1D continuous box filter

$$f(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & otherwise \end{cases}$$



filtering output is a blurred version of g

$$(f * g)(x) = \int_{-0.5}^{0.5} g(x-y)dy$$

Convolution for 2D discrete signals

Definition of filtering as convolution:

notice the flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j) I(x - i, y - j)$$

filtered image — filter — input image

If the filter $f(i, j)$ is non-zero only within $-1 \leq i, j \leq 1$, then

$$(f * g)(x, y) = \sum_{i,j=-1}^{1} f(i, j) I(x - i, y - j)$$

The kernel we saw earlier is the 3x3 matrix representation of $f(i, j)$.

Convolution for 2D discrete signals

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

f = Image

h = Kernel

h

h_1	h_2	h_3
h_4	h_5	h_6
h_7	h_8	h_9

$X - flip$

h_7	h_8	h_9
h_4	h_5	h_6
h_1	h_2	h_3

$Y - flip$

f

f_1	f_2	f_3
f_4	f_5	f_6
f_7	f_8	f_9

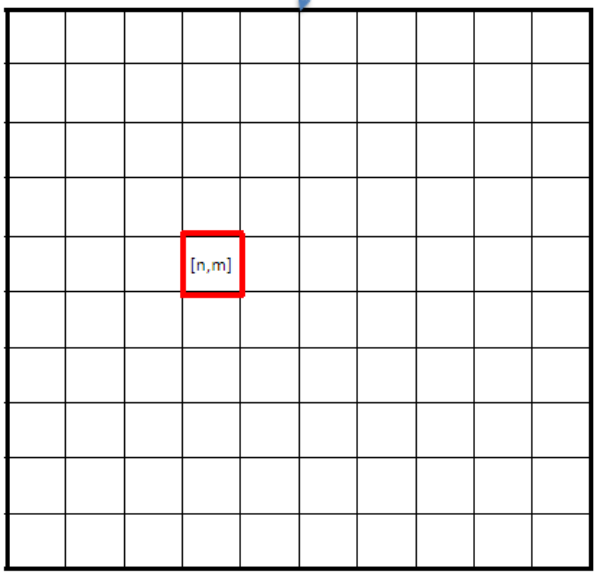
$*$

h_9	h_8	h_7
h_6	h_5	h_4
h_3	h_2	h_1

$$f * h = f_1h_9 + f_2h_8 + f_3h_7 + f_4h_6 + f_5h_5 + f_6h_4 + f_7h_3 + f_8h_2 + f_9h_1$$

2D Discrete Convolution

$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$



Output $f*h$

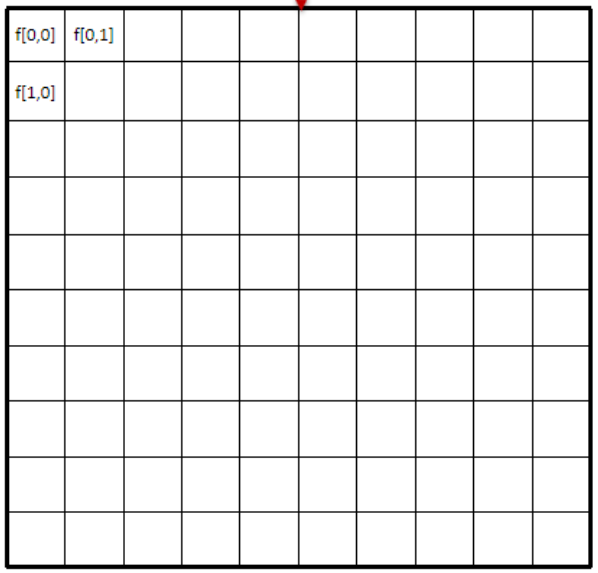


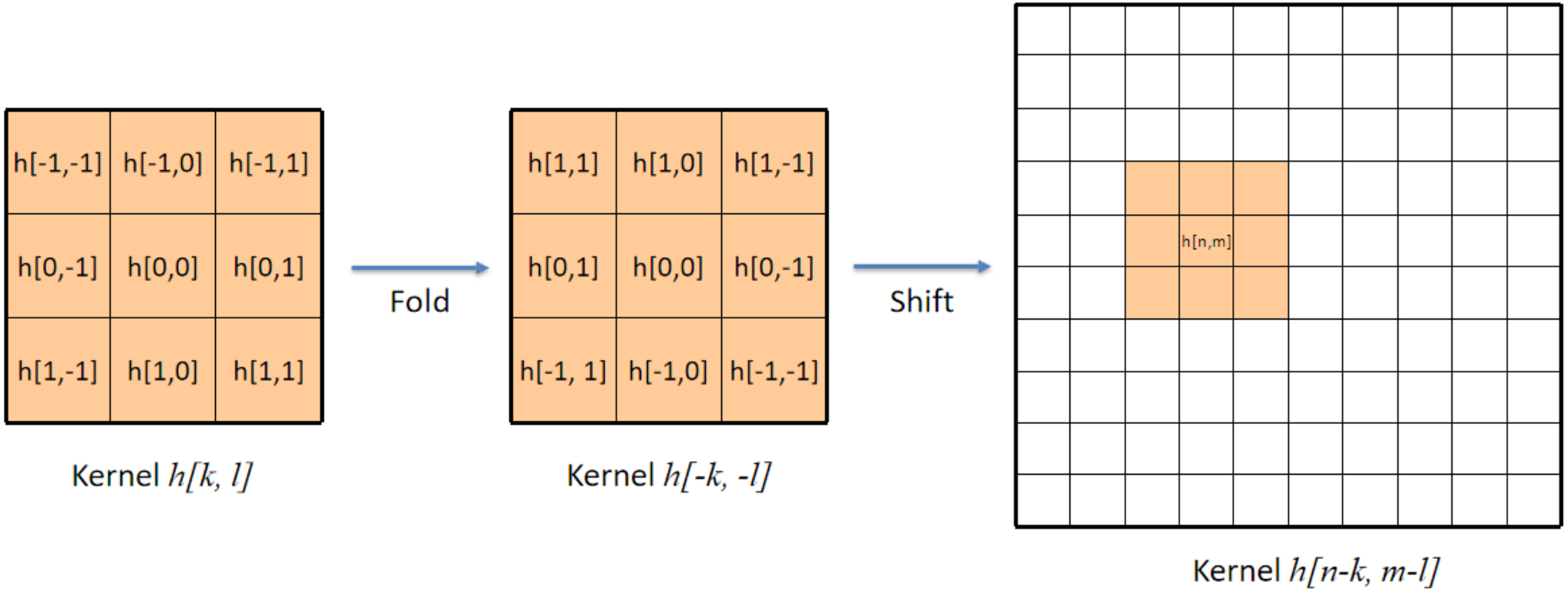
Image $f[k, l]$

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

Kernel $h[k, l]$

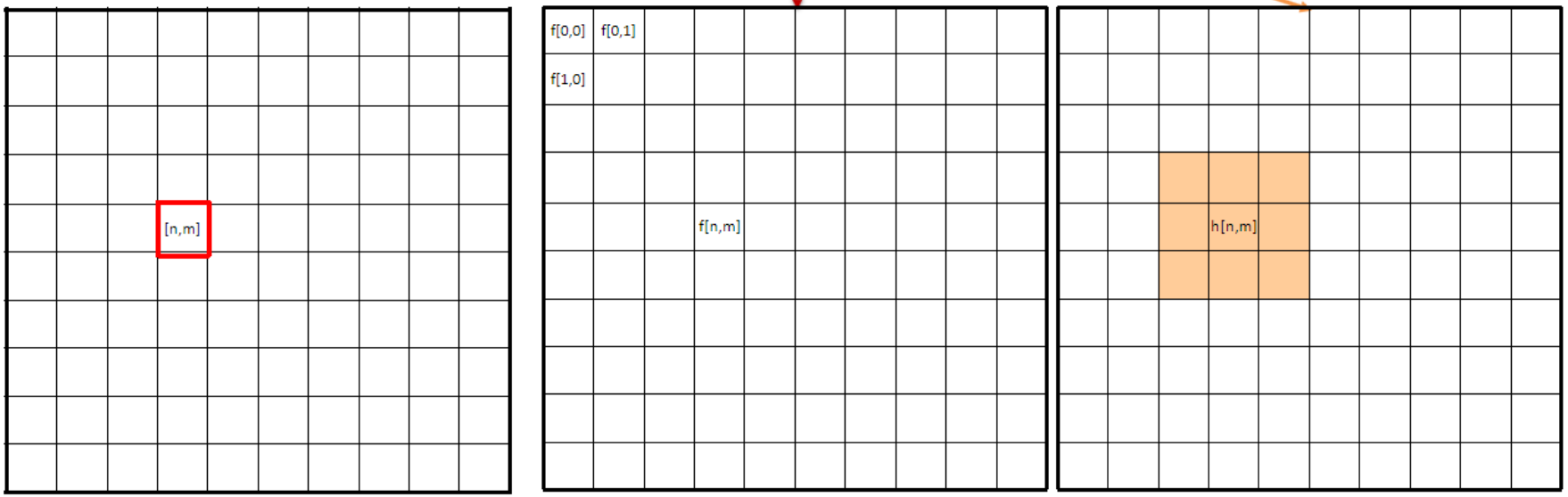
2D Discrete Convolution

$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$



2D Discrete Convolution

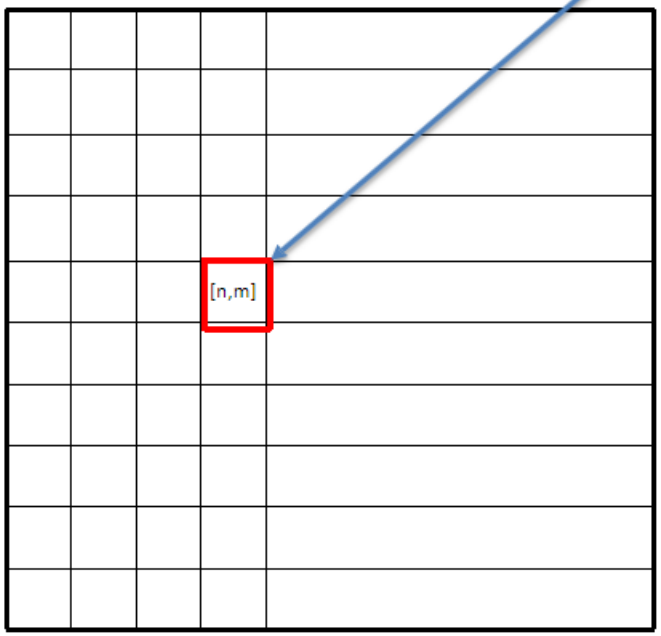
$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$



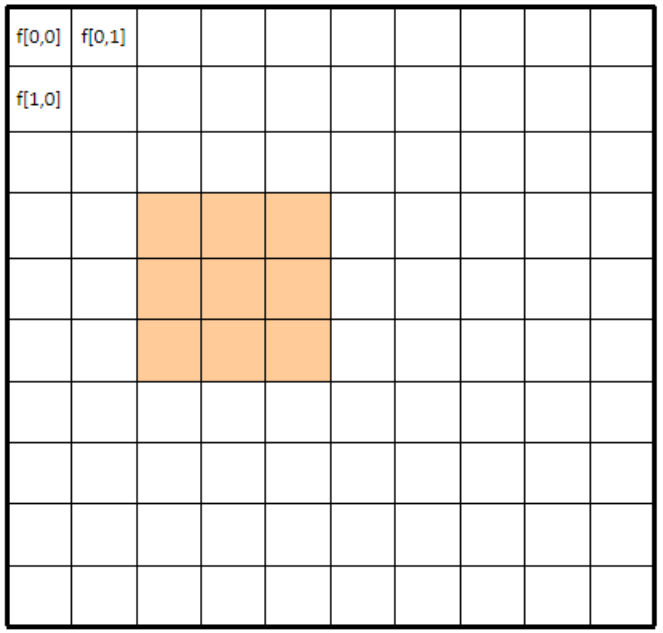
Output $f*h$ Image $f[k, l]$ Kernel $h[n-k, m-l]$

2D Discrete Convolution

$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$



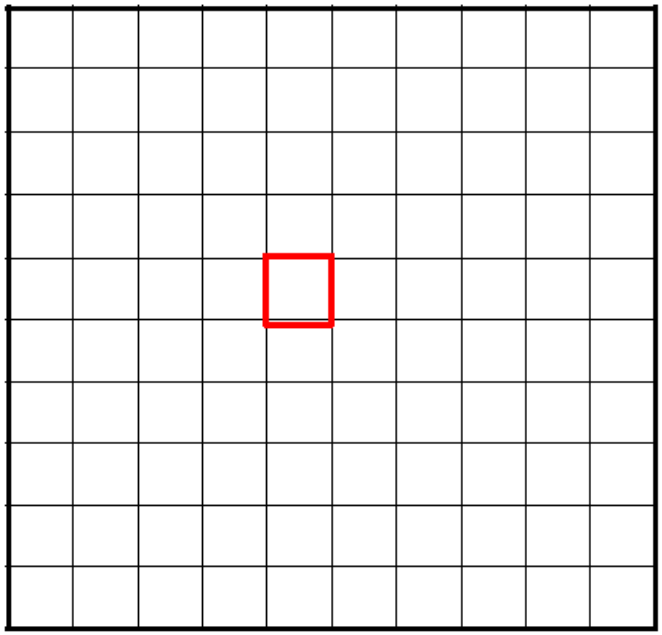
Output $f*h$



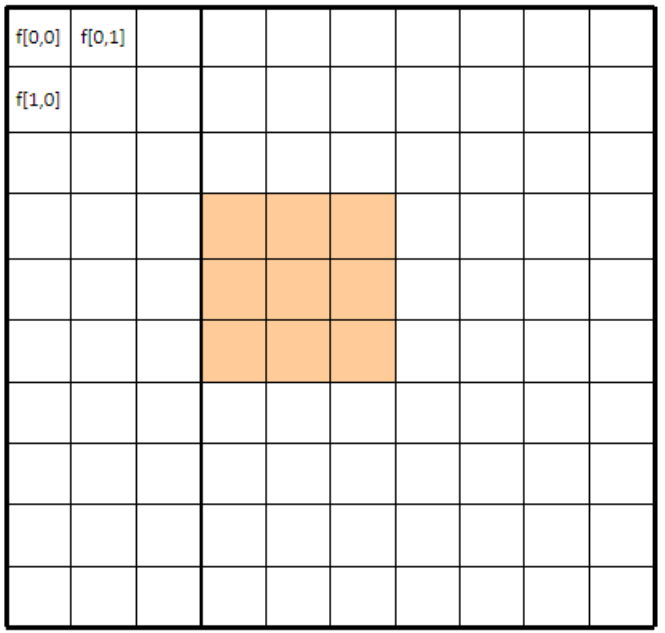
Element-wise multiplication
Image $f[k, l]$ • Kernel $h[n-k, m-l]$

2D Discrete Convolution

$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$



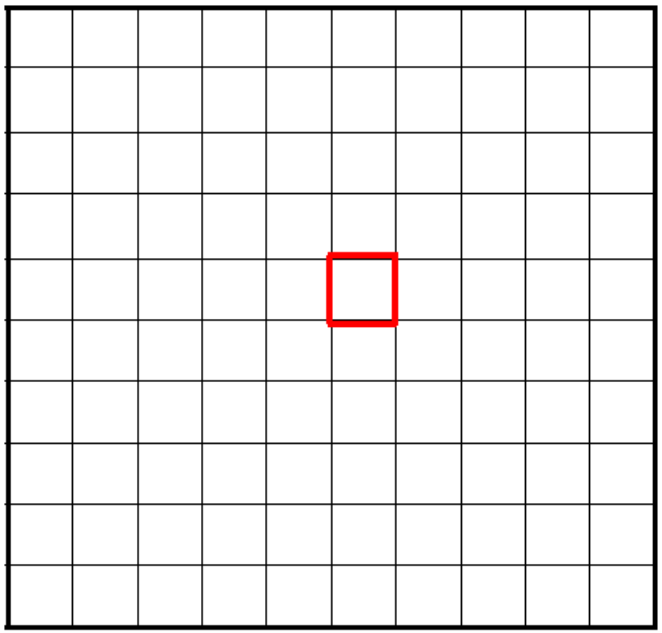
Output $f*h$



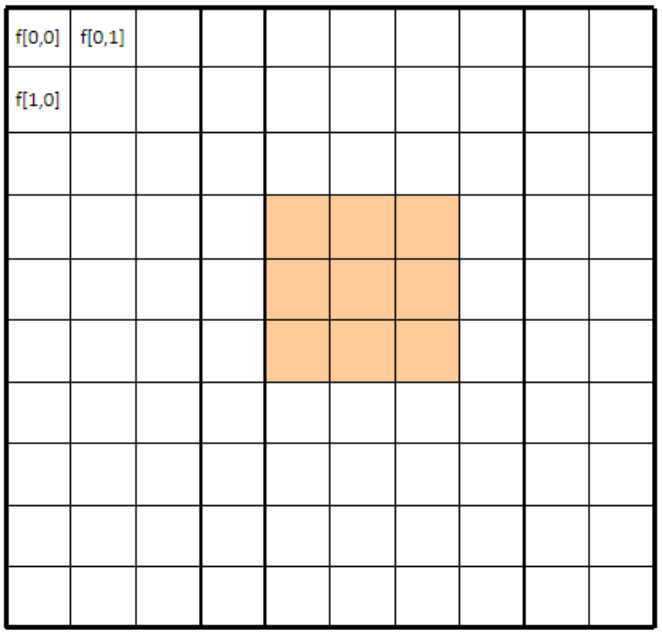
Element-wise multiplication
Image $f[k, l]$ • Kernel $h[n-k, m-l]$

2D Discrete Convolution

$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$



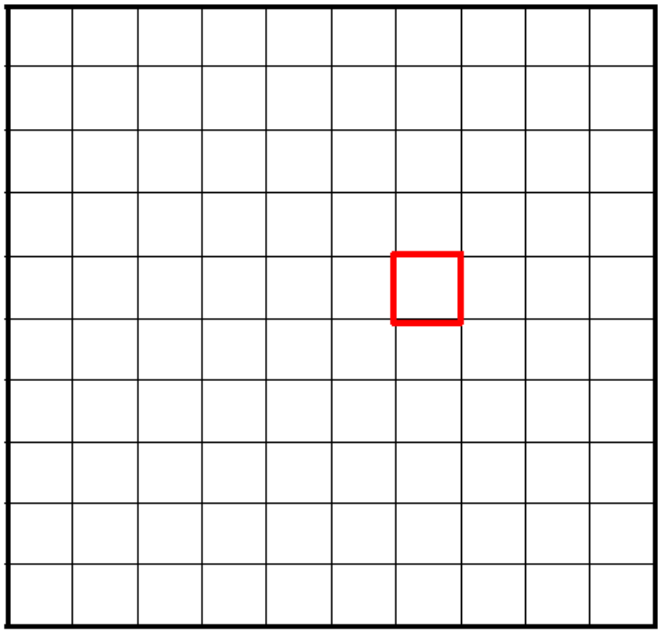
Output $f*h$



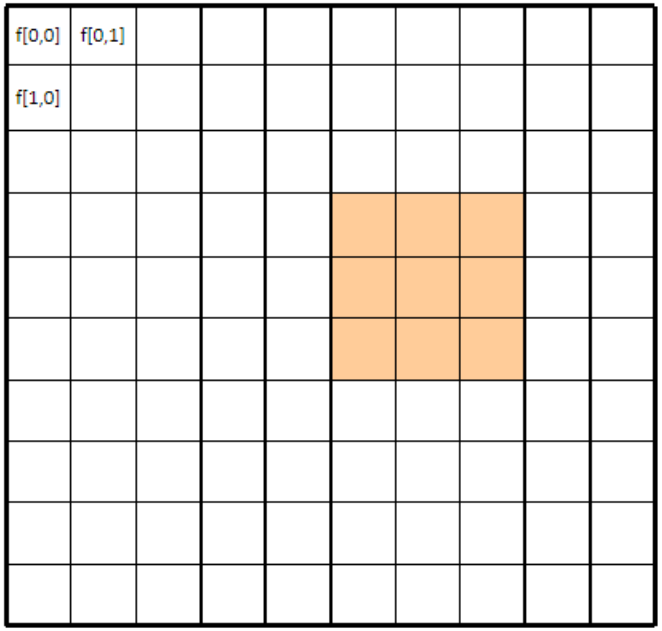
Element-wise multiplication
Image $f[k, l]$ • Kernel $h[n-k, m-l]$

2D Discrete Convolution

$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$



Output $f*h$



Element-wise multiplication
Image $f[k, l]$ • Kernel $h[n-k, m-l]$

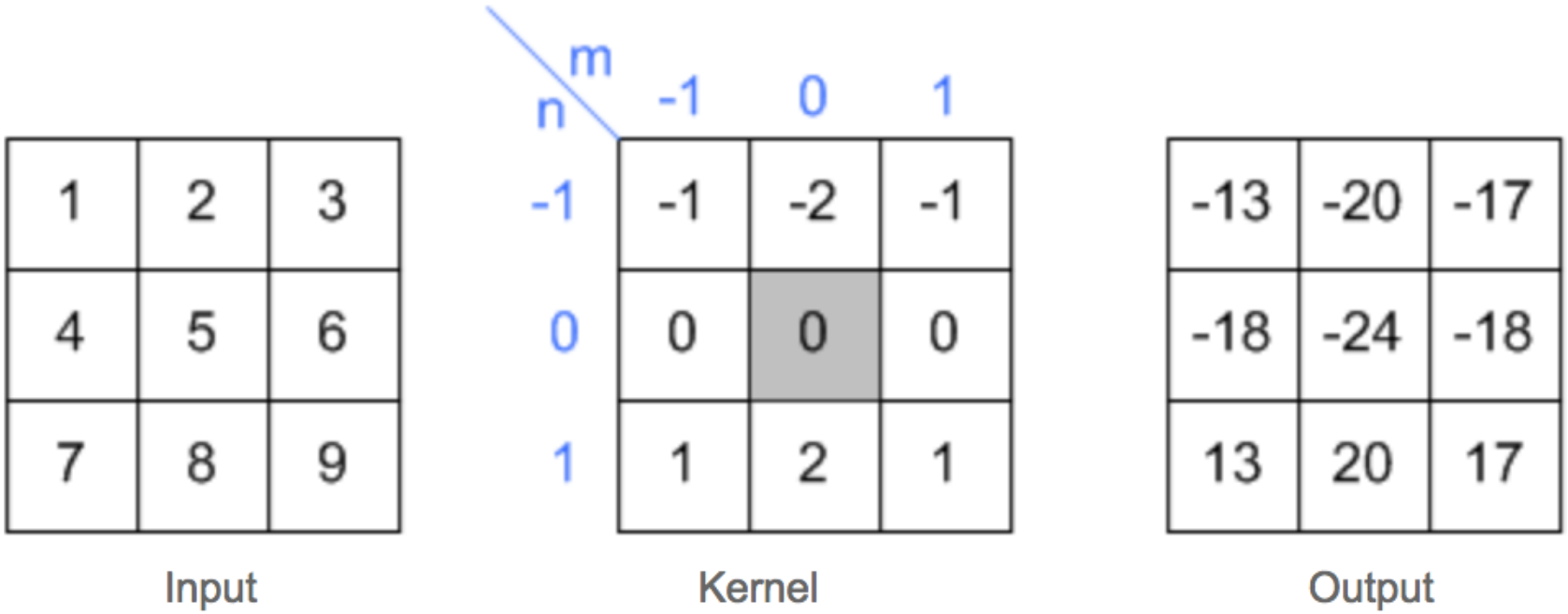
2D Discrete Convolution

$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n-k, m-l]$$

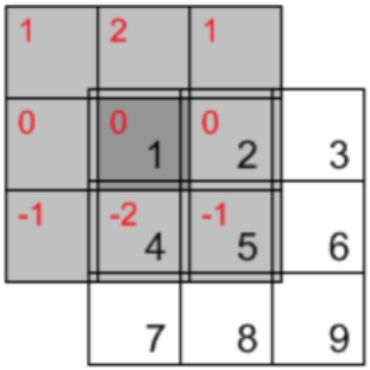
- **Algorithm:**

1. Fold h[k, l] about origin to form h[−k, −l]
2. Shift the folded results by n, m to form h[n − k, m − l]
3. Multiply h[n − k, m − l] by f[k, l]
4. Sum over all k, l
5. Repeat for every n, m

2D convolution example



2D convolution example



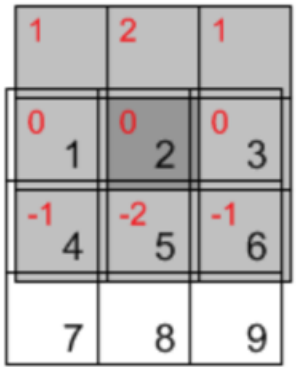
$$
\begin{aligned}
&= x[-1,-1]\cdot h[1,1] + x[0,-1]\cdot h[0,1] + x[1,-1]\cdot h[-1,1] \\
&\quad + x[-1,0]\cdot h[1,0] + x[0,0]\cdot h[0,0] + x[1,0]\cdot h[-1,0] \\
&\quad + x[-1,1]\cdot h[1,-1] + x[0,1]\cdot h[0,-1] + x[1,1]\cdot h[-1,-1] \\
&= 0\cdot 1 + 0\cdot 2 + 0\cdot 1 + 0\cdot 0 + 1\cdot 0 + 2\cdot 0 + 0\cdot(-1) + 4\cdot(-2) + 5\cdot(-1) = -13
\end{aligned}
$$

-13	-20	-17
-18	-24	-18
13	20	17

Output

Slide credit: Song Ho Ahn

2D convolution example



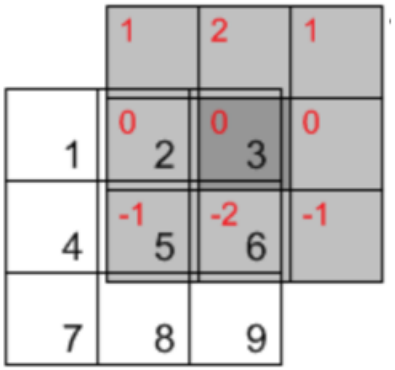
$$
\begin{aligned}
&= x[0,-1]\cdot h[1,1] + x[1,-1]\cdot h[0,1] + x[2,-1]\cdot h[-1,1] \\
&\quad + x[0,0]\cdot h[1,0] + x[1,0]\cdot h[0,0] + x[2,0]\cdot h[-1,0] \\
&\quad + x[0,1]\cdot h[1,-1] + x[1,1]\cdot h[0,-1] + x[2,1]\cdot h[-1,-1] \\
&= 0\cdot 1 + 0\cdot 2 + 0\cdot 1 + 1\cdot 0 + 2\cdot 0 + 3\cdot 0 + 4\cdot(-1) + 5\cdot(-2) + 6\cdot(-1) = -20
\end{aligned}
$$

-13	-20	-17
-18	-24	-18
13	20	17

Output

Slide credit: Song Ho Ahn

2D convolution example



$$
\begin{aligned}
&= x[1,-1]\cdot h[1,1] + x[2,-1]\cdot h[0,1] + x[3,-1]\cdot h[-1,1] \\
&\quad + x[1,0]\cdot h[1,0] + x[2,0]\cdot h[0,0] + x[3,0]\cdot h[-1,0] \\
&\quad + x[1,1]\cdot h[1,-1] + x[2,1]\cdot h[0,-1] + x[3,1]\cdot h[-1,-1] \\
&= 0\cdot 1 + 0\cdot 2 + 0\cdot 1 + 2\cdot 0 + 3\cdot 0 + 0\cdot 0 + 5\cdot(-1) + 6\cdot(-2) + 0\cdot(-1) = -17
\end{aligned}
$$

-13	-20	-17
-18	-24	-18
13	20	17

Output

Slide credit: Song Ho Ahn

2D convolution example

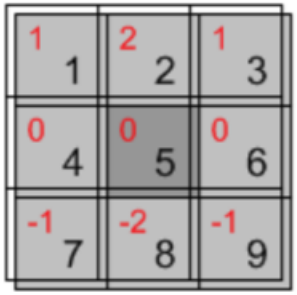
$$
\begin{aligned}
&= x[-1,0]\cdot h[1,1] + x[0,0]\cdot h[0,1] + x[1,0]\cdot h[-1,1] \\
&\quad + x[-1,1]\cdot h[1,0] + x[0,1]\cdot h[0,0] + x[1,1]\cdot h[-1,0] \\
&\quad + x[-1,2]\cdot h[1,-1] + x[0,2]\cdot h[0,-1] + x[1,2]\cdot h[-1,-1] \\
&= 0\cdot 1 + 1\cdot 2 + 2\cdot 1 + 0\cdot 0 + 4\cdot 0 + 5\cdot 0 + 0\cdot(-1) + 7\cdot(-2) + 8\cdot(-1) = -18
\end{aligned}
$$

-13	-20	-17
-18	-24	-18
13	20	17

Output

Slide credit: Song Ho Ahn

2D convolution example



$$
\begin{aligned}
&= x[0,0]\cdot h[1,1] + x[1,0]\cdot h[0,1] + x[2,0]\cdot h[-1,1] \\
&\quad + x[0,1]\cdot h[1,0] + x[1,1]\cdot h[0,0] + x[2,1]\cdot h[-1,0] \\
&\quad + x[0,2]\cdot h[1,-1] + x[1,2]\cdot h[0,-1] + x[2,2]\cdot h[-1,-1] \\
&= 1\cdot 1 + 2\cdot 2 + 3\cdot 1 + 4\cdot 0 + 5\cdot 0 + 6\cdot 0 + 7\cdot(-1) + 8\cdot(-2) + 9\cdot(-1) = -24
\end{aligned}
$$

-13	-20	-17
-18	-24	-18
13	20	17

Output

Slide credit: Song Ho Ahn

2D convolution example

$$
\begin{aligned}
&= x[1,0]\cdot h[1,1] + x[2,0]\cdot h[0,1] + x[3,0]\cdot h[-1,1] \\
&\quad + x[1,1]\cdot h[1,0] + x[2,1]\cdot h[0,0] + x[3,1]\cdot h[-1,0] \\
&\quad + x[1,2]\cdot h[1,-1] + x[2,2]\cdot h[0,-1] + x[3,2]\cdot h[-1,-1] \\
&= 2\cdot 1 + 3\cdot 2 + 0\cdot 1 + 5\cdot 0 + 6\cdot 0 + 0\cdot 0 + 8\cdot(-1) + 9\cdot(-2) + 0\cdot(-1) = -18
\end{aligned}
$$

-13	-20	-17
-18	-24	-18
13	20	17

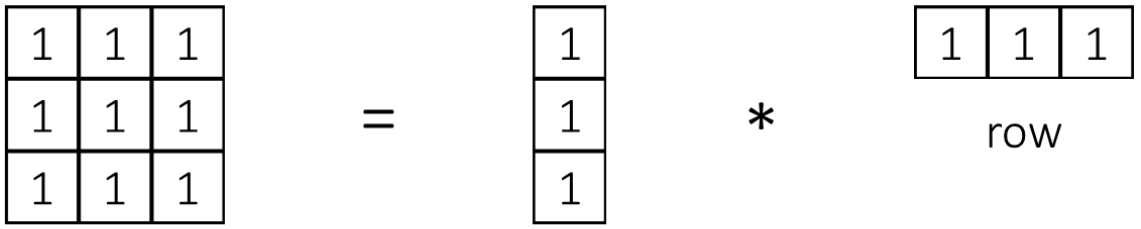
Output

Slide credit: Song Ho Ahn

Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter



2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

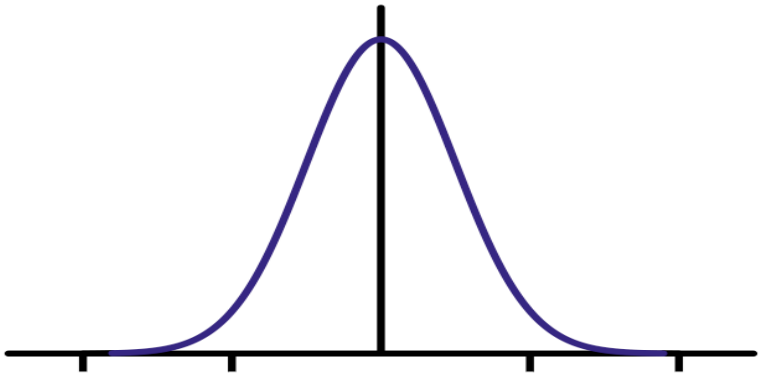
If the image has M x M pixels and the filter kernel has size N x N:

- What is the cost of convolution with a non-separable filter? ⟶ $M^2 \times N^2$
- What is the cost of convolution with a separable filter? ⟶ $2 \times N \times M^2$

The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss
- kernel values sampled from the 2D Gaussian function:

$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$



- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

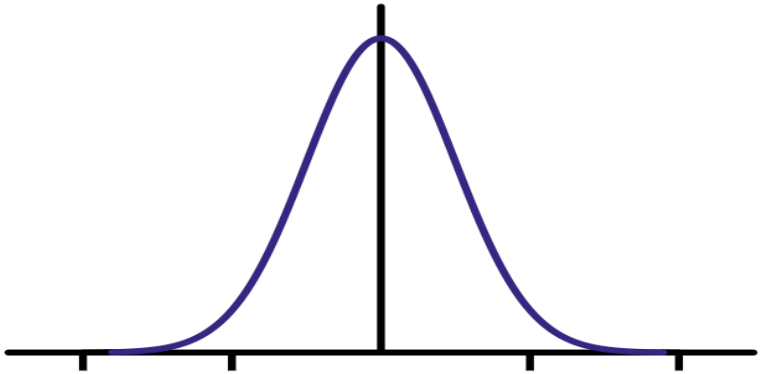
Any heuristics for selecting where to truncate?

The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss

- kernel values sampled from the 2D Gaussian function:

$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$



- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?

Is this a separable filter?

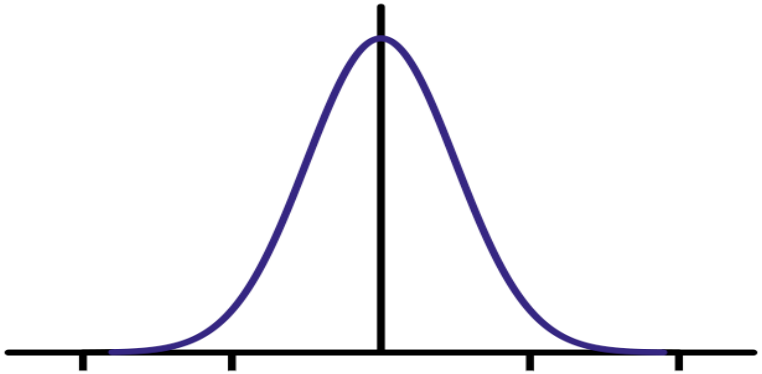
kernel $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss
- kernel values sampled from the 2D Gaussian function:

$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$



- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?

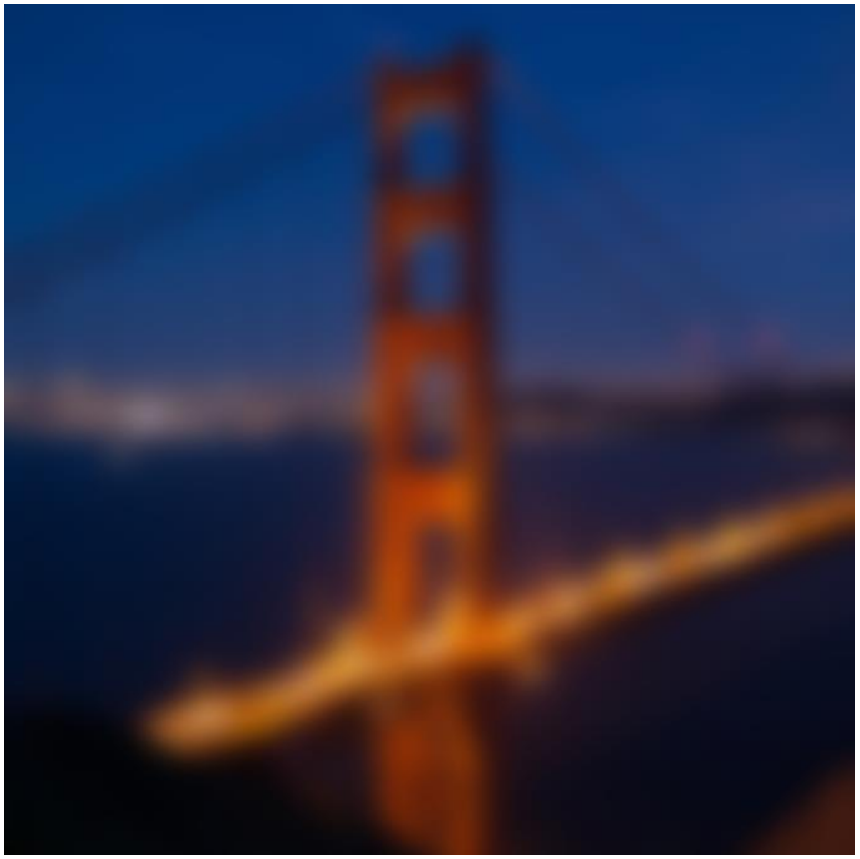
usually at 2-3σ

Is this a separable filter? Yes!

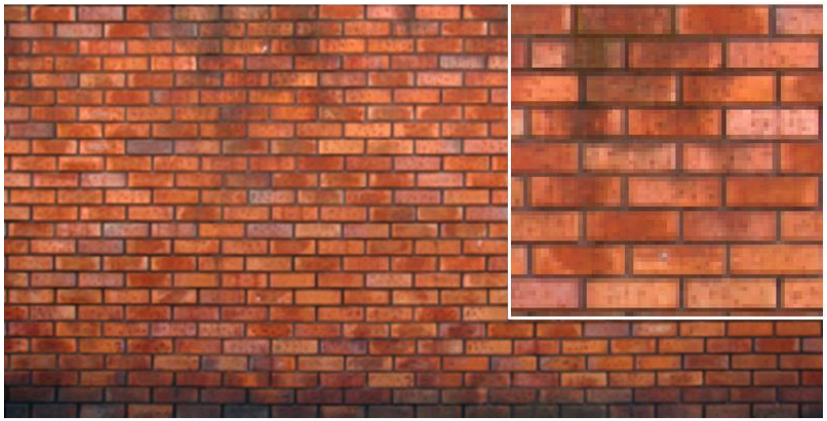
kernel $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

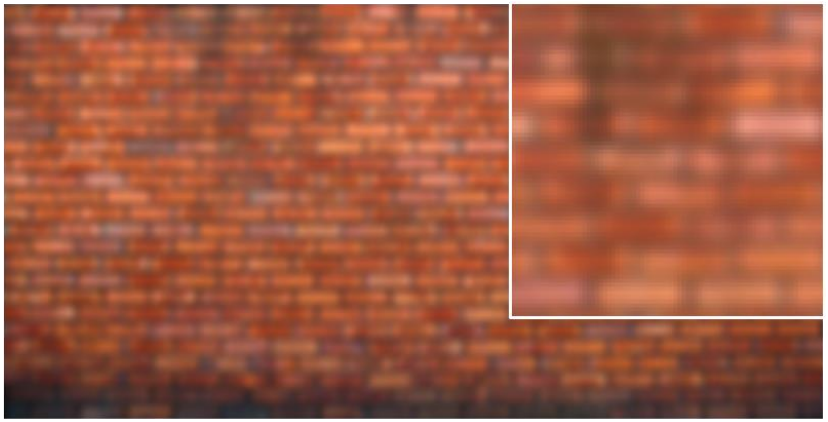
Gaussian filtering example



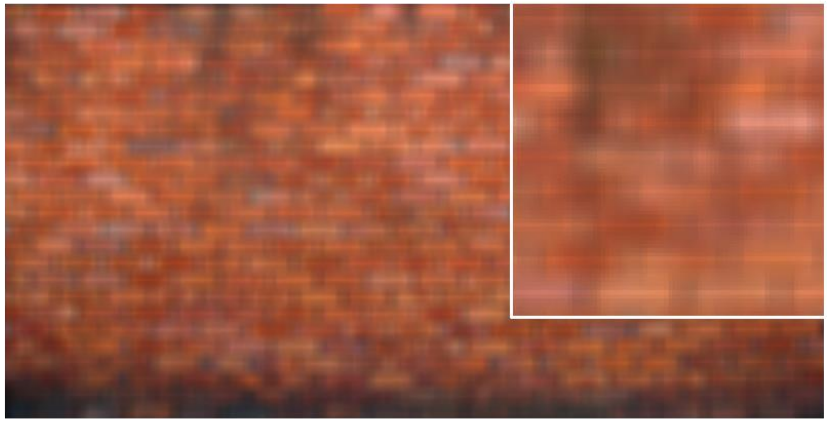
Gaussian vs box filtering



original



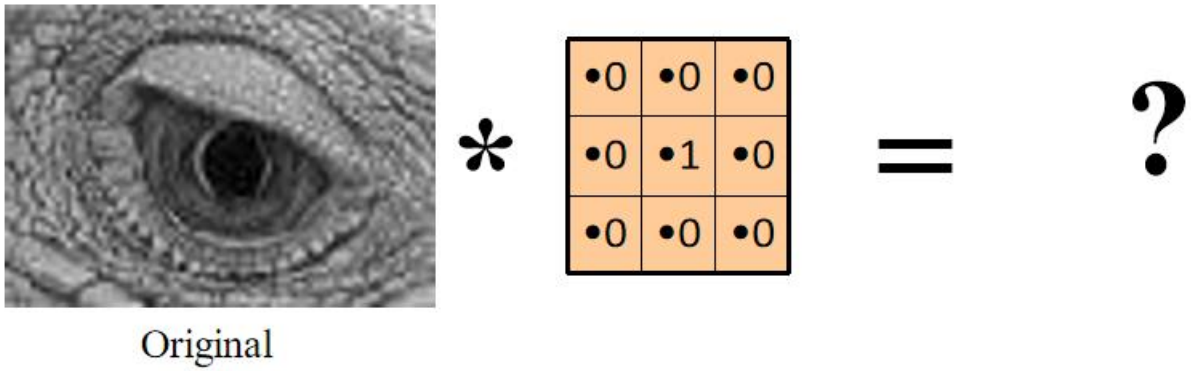
7x7 Gaussian



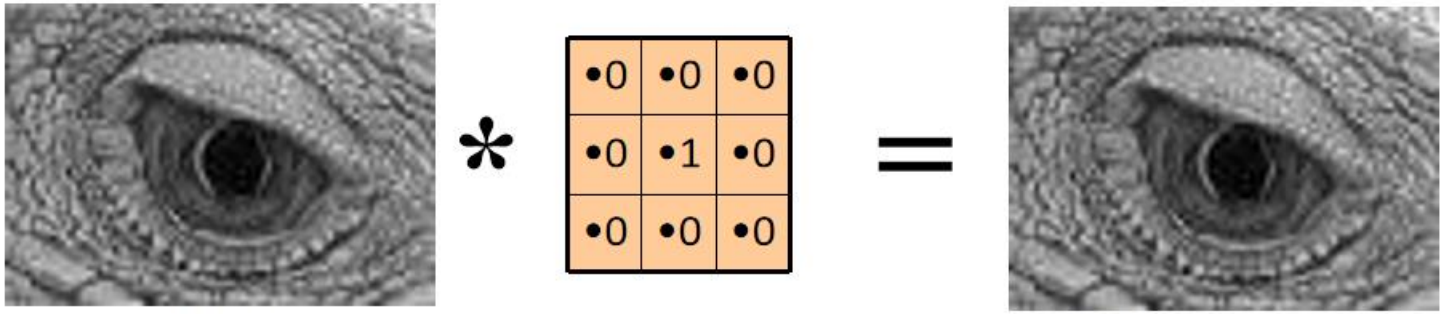
7x7 box

Which blur do you like better?

Convolution in 2D - examples



Convolution in 2D - examples

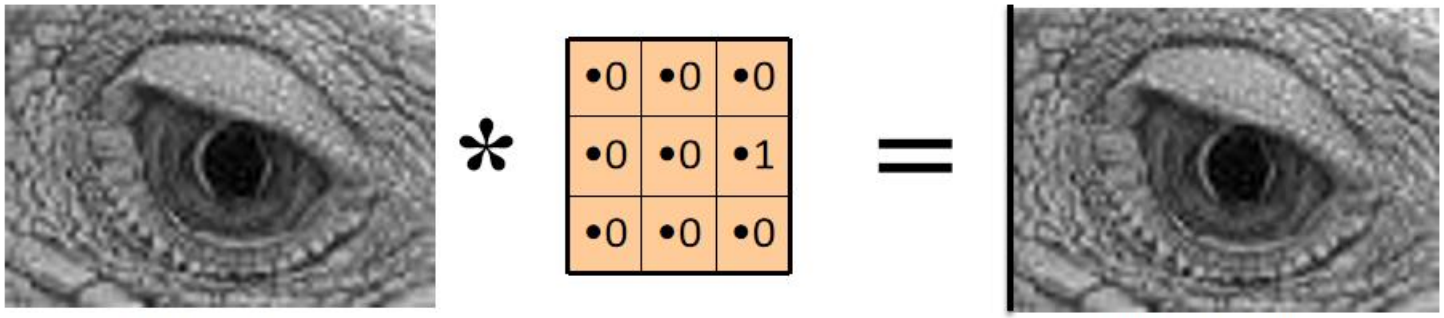


Convolution in 2D - examples

Original

$$*\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = ?$$

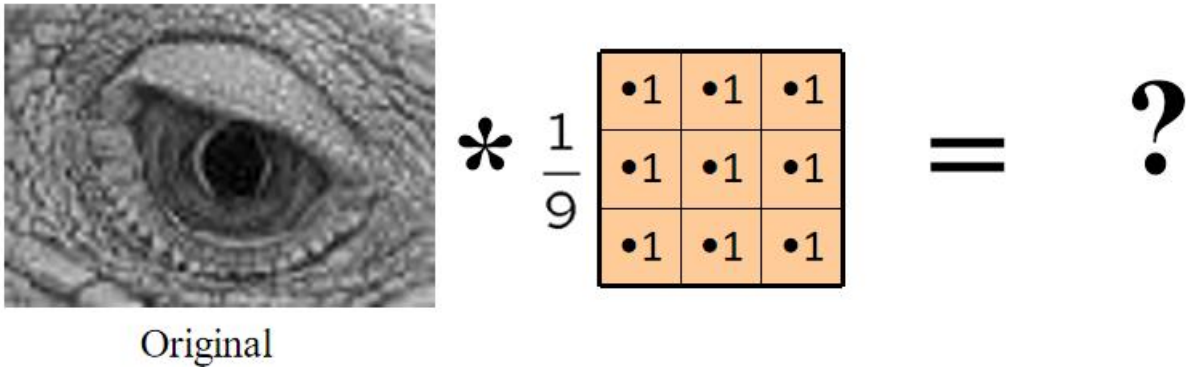
Convolution in 2D - examples



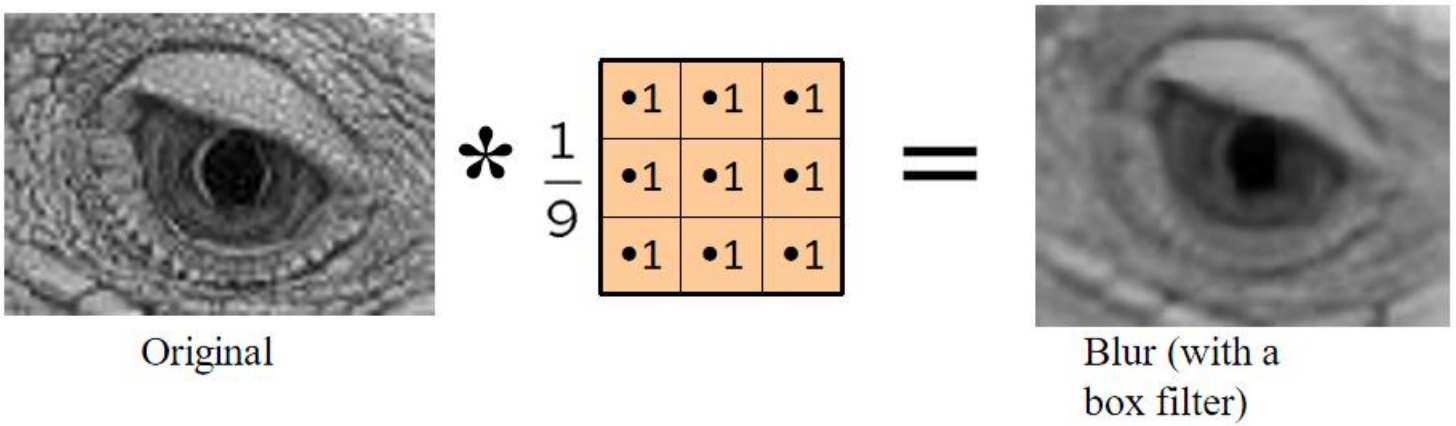
Original

Shifted right
By 1 pixel

Convolution in 2D - examples



Convolution in 2D - examples



Convolution in 2D - examples

Original

$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) = ?$$

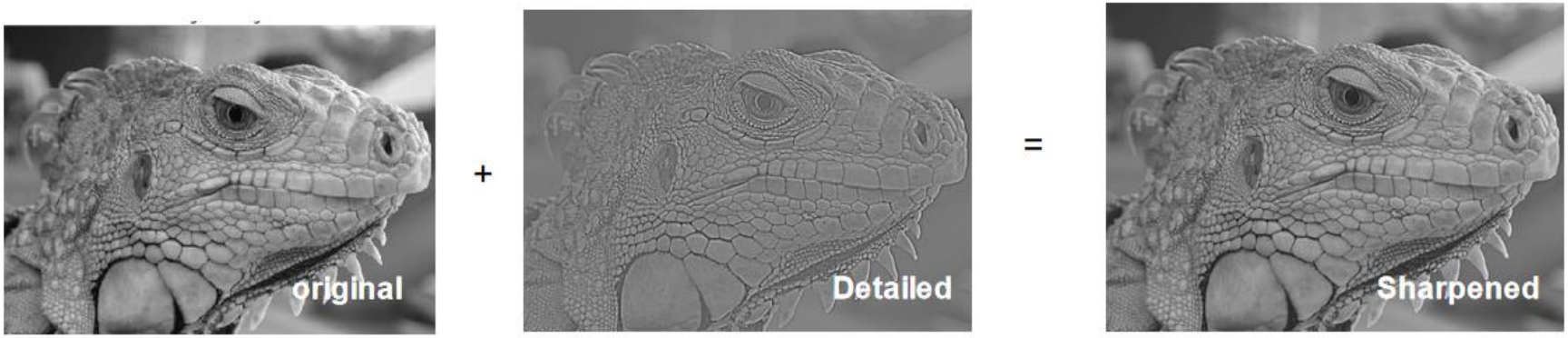
(Note that filter sums to 1)

"details of the image"

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

What does blurring take away?

- Let's add it back:

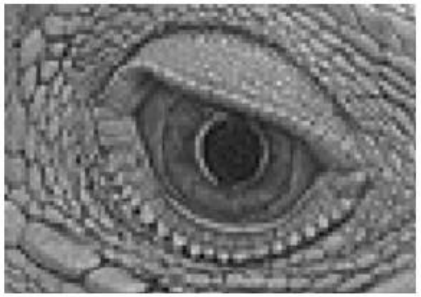


Convolution in 2D - examples

Original

$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) =$$

(Note that filter sums to 1)

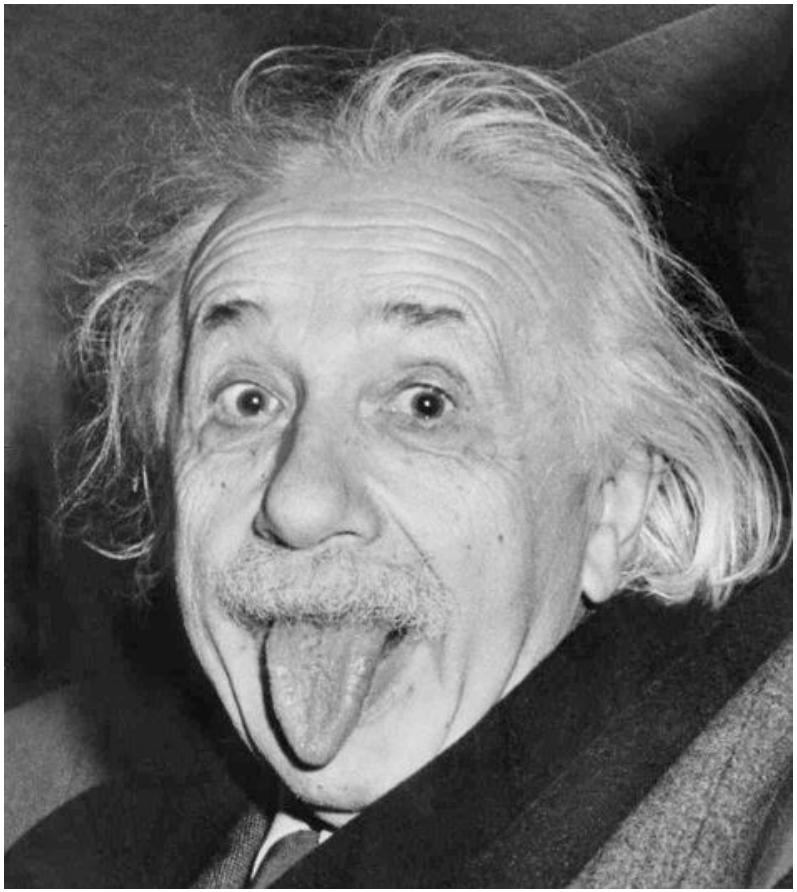


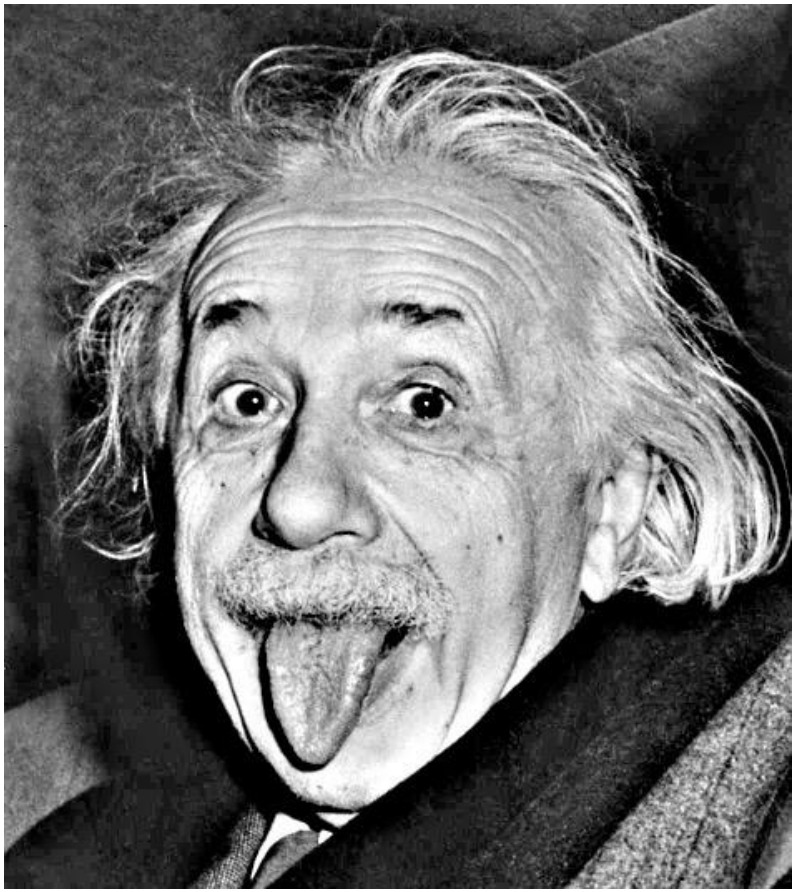
"details of the image"

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter: Accentuates differences with local average

Sharpening examples





Sharpening examples

Sharpening examples

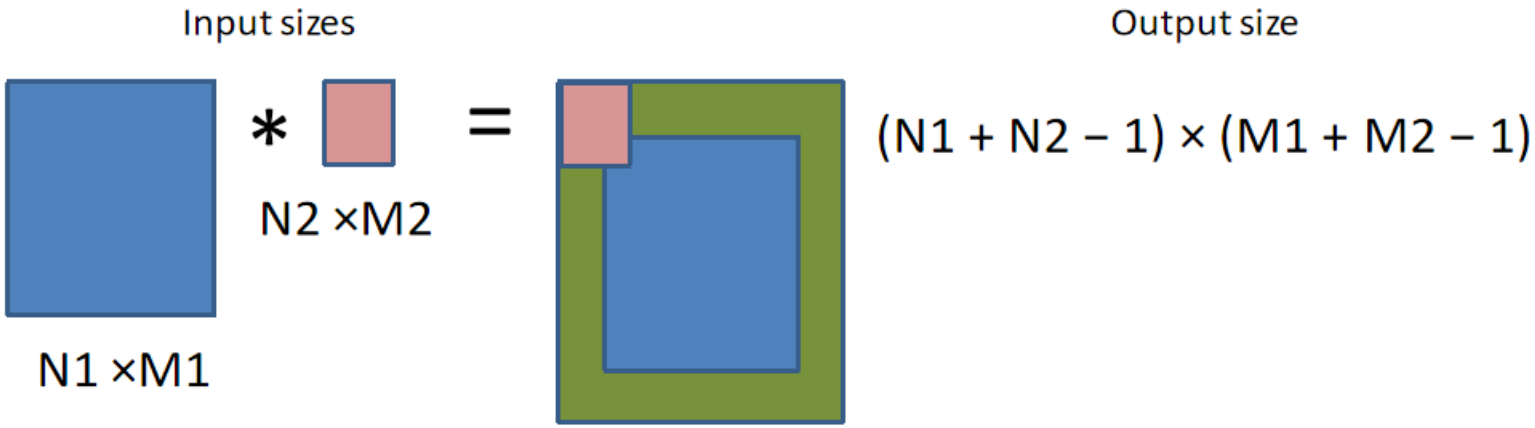
original

sharpened

oversharpened

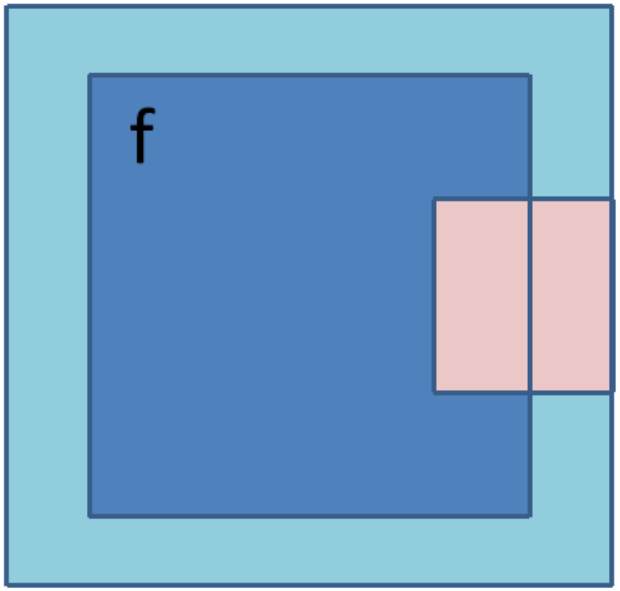
Implementation detail: Image support and edge effect

- A computer will only convolve **finite support signals**.
  - That is: images that are zero for n, m outside some rectangular region;
- Numpy's convolution performs 2D Discrete convolution of finite support signals.



Implementation detail: Image support and edge effect

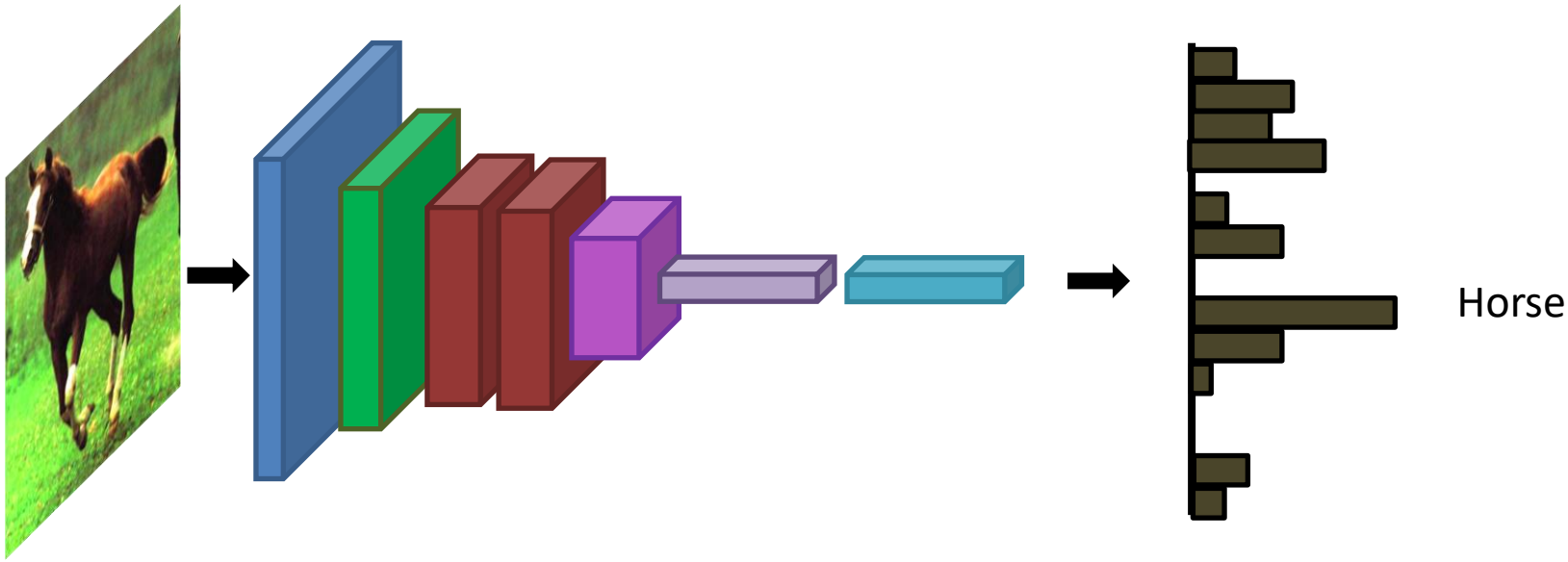
- A computer will only convolve finite support signals.
- What happens at the edge?



h

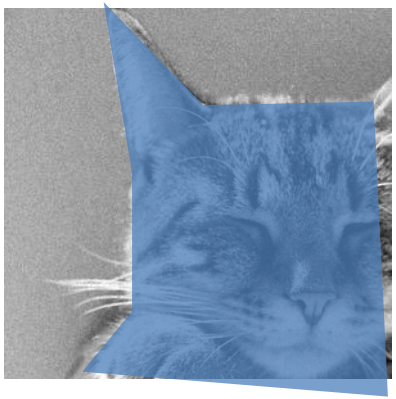
- zero “padding”
- edge value replication
- mirror extension
- more (beyond the scope of this class)

Convolution is everywhere



Why is convolution important?

- Shift invariance is a crucial property

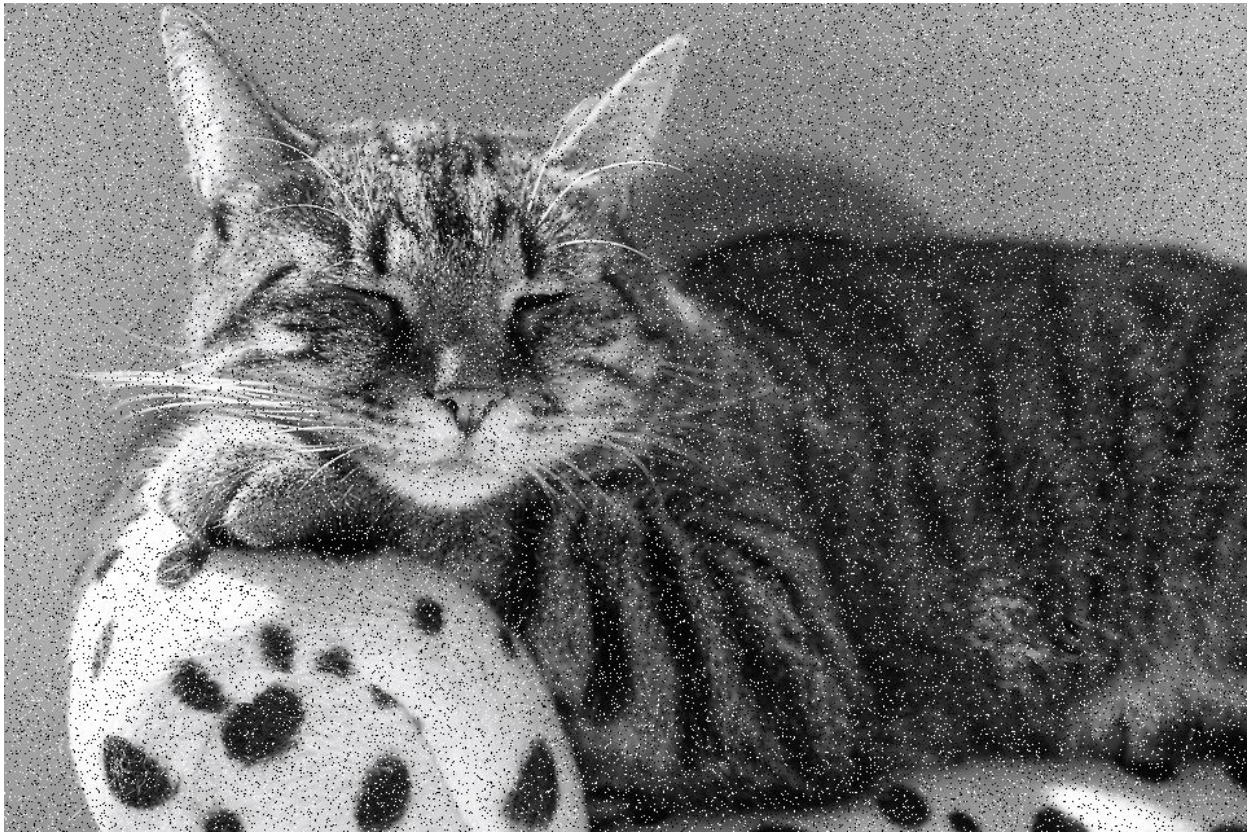


Non-linear filters: Thresholding

$$g(m,n) = \begin{cases} 255, & f(m,n) > A \\ 0 & otherwise \end{cases}$$

Non-linear filters

- Sometimes mean filtering does not work



Non-linear filters

- Sometimes mean filtering does not work

Non-linear filters

- Mean is sensitive to outliers
- Median filter: Replace pixel by *median* of neighbors

Non-linear filters

What we will learn today?

- Convolution
- **Correlation**

– Some background reading:

Forsyth and Ponce, Computer Vision, Chapter 7

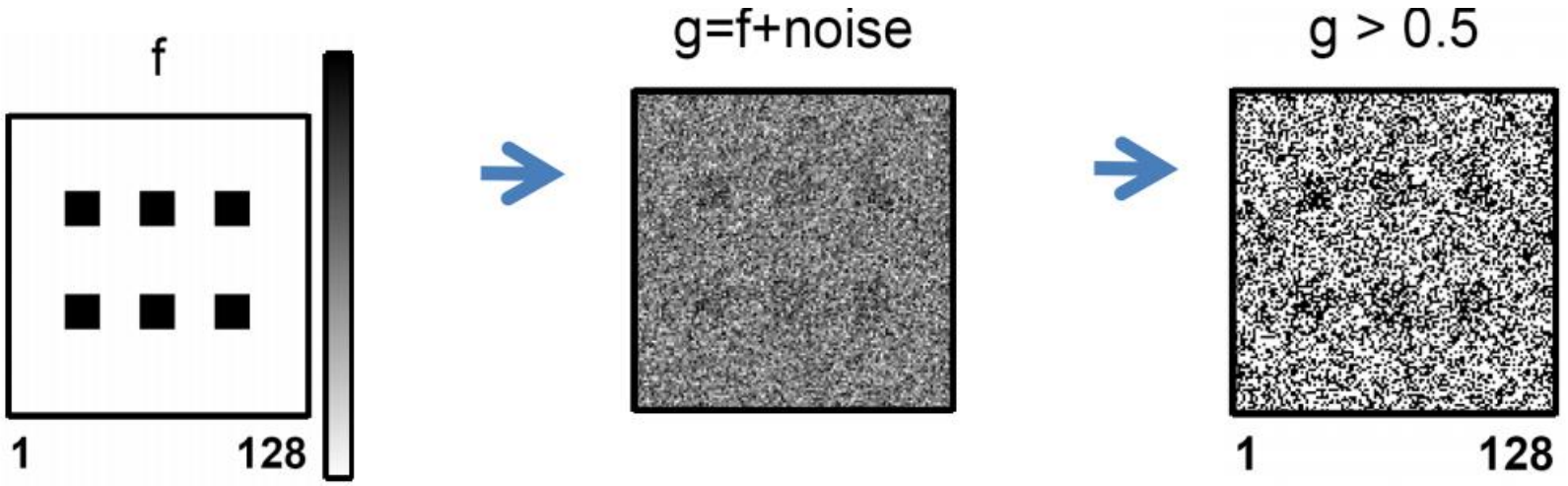
(Cross) correlation – symbol: **

- Cross correlation of two 2D signals f[n,m] and h[n,m]

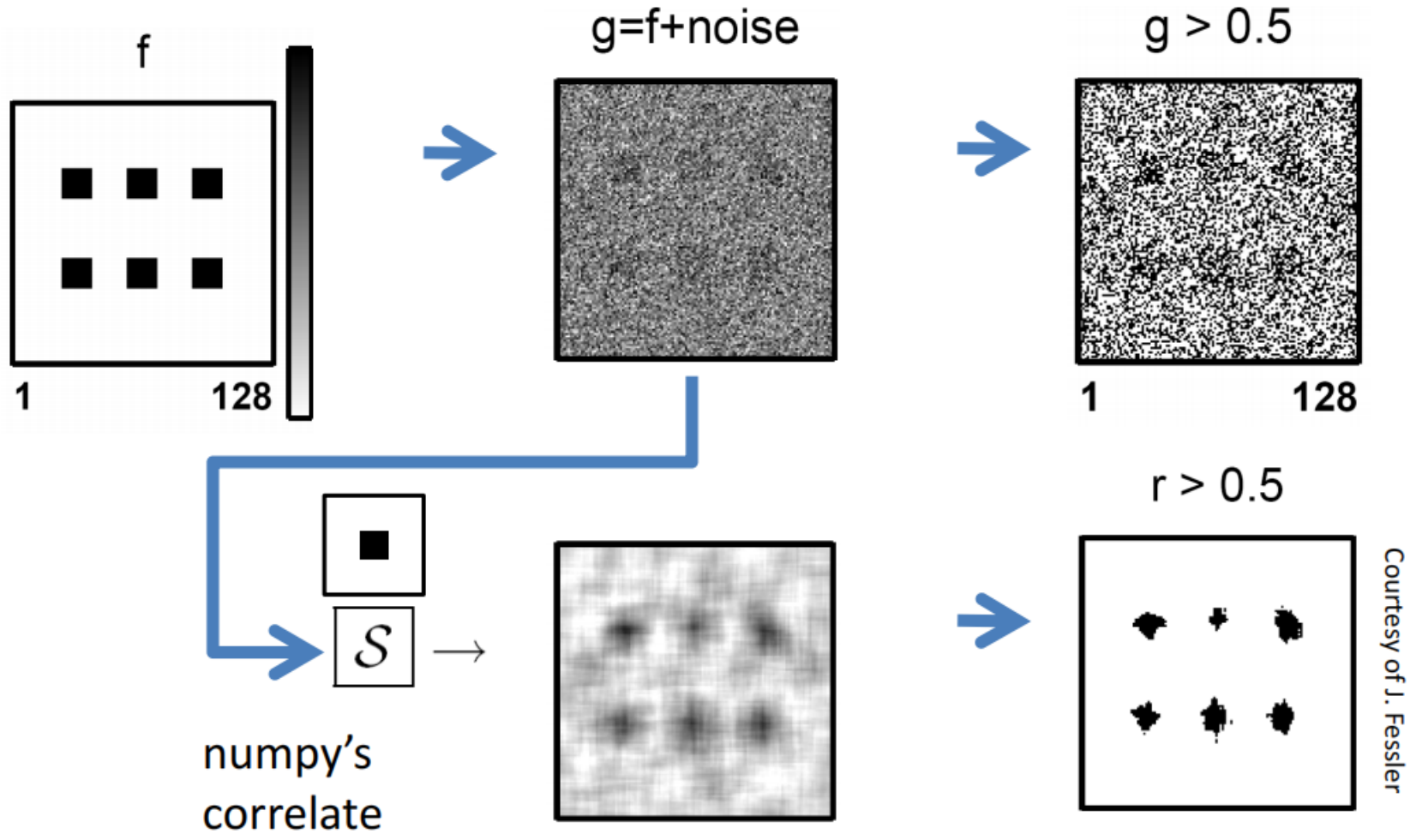
$$f[n,m] * *h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \cdot h[n+k, m+l]$$

  – Equivalent to a convolution without the flip
  – Use it to measure ‘similarity’ between f and h.

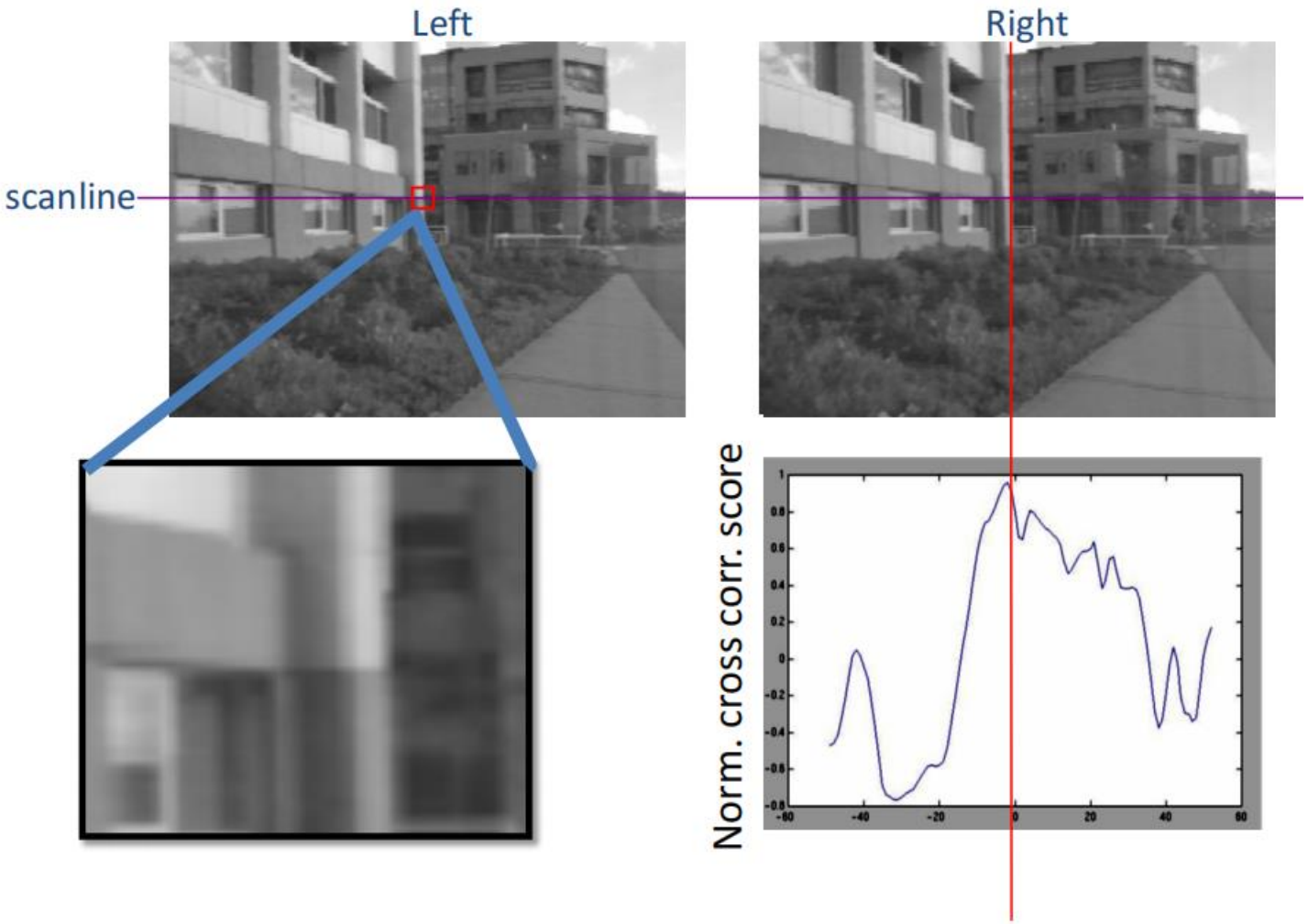
(Cross) correlation – example



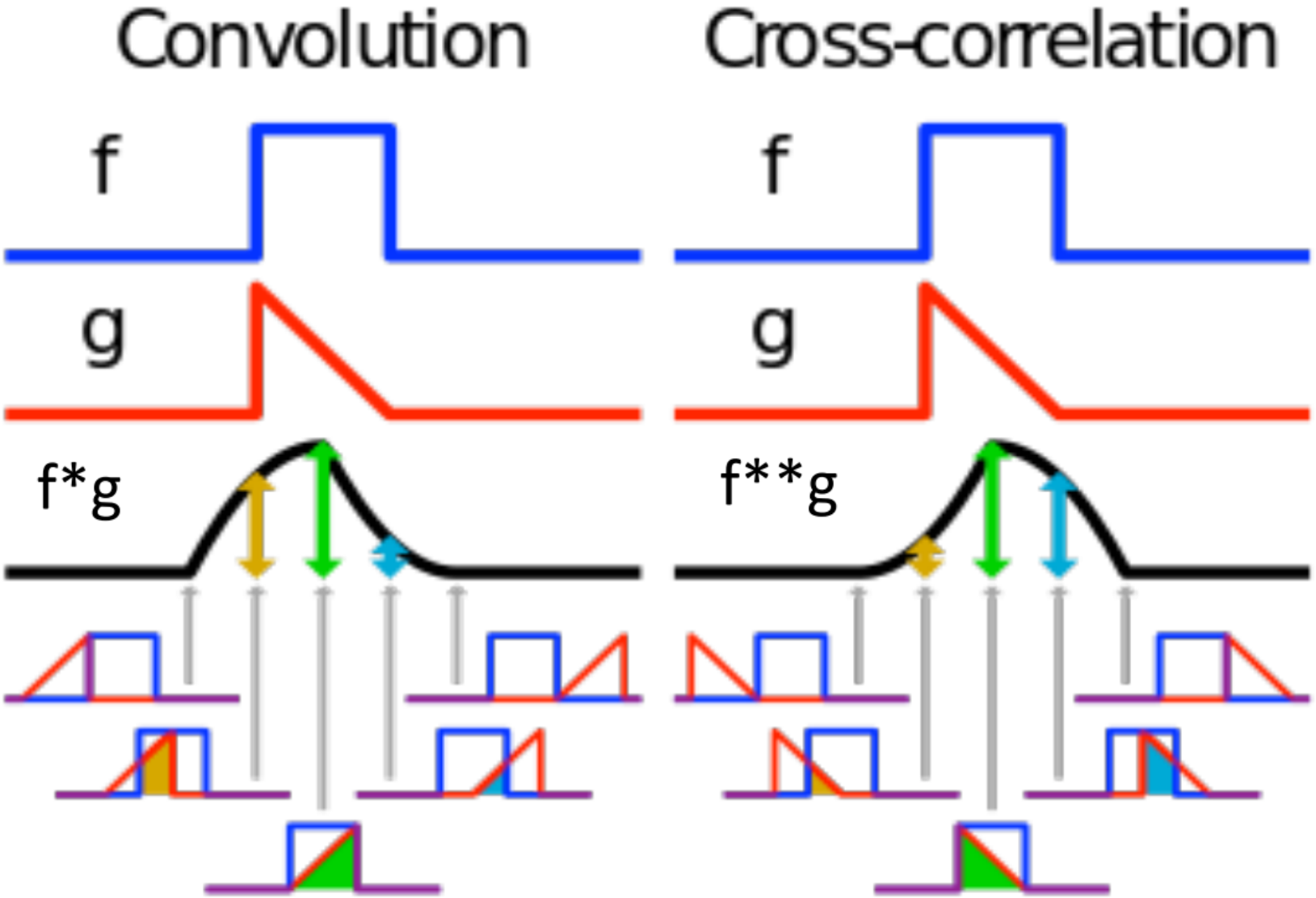
(Cross) correlation – example



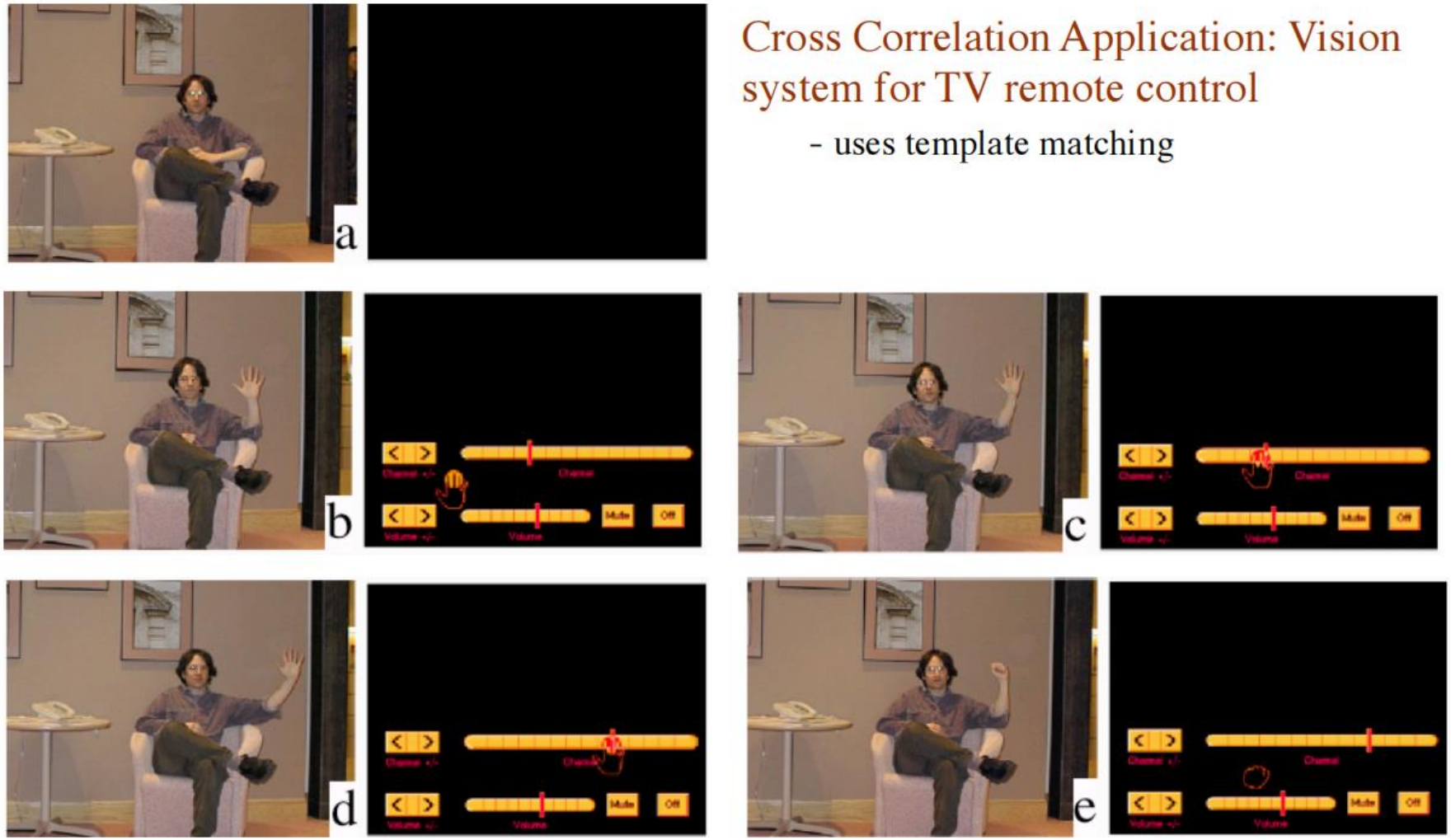
(Cross) correlation – example



(Cross) correlation – example



(Cross) correlation – example



Cross Correlation Application: Vision system for TV remote control

- uses template matching

Figure from "Computer Vision for Interactive Computer Graphics," W.Freeman et al, IEEE Computer Graphics and Applications, 1998 copyright 1998, IEEE

Properties

- Associative property:

$$(f ** h_1) ** h_2 = f ** (h_1 ** h_2)$$

- Distributive property:

$$f ** (h_1 + h_2) = (f ** h_1) + (f ** h_2)$$

- The order doesn't matter! $h_1 ** h_2 = h_2 ** h_1$

Properties

- Shift property:

$$f[n,m] ** \delta_2[n-n_0, m-m_0] = f[n-n_0, m-m_0]$$

- Shift-invariance:

$$\begin{aligned} g[n,m] &= f[n,m] ** h[n,m] \\ &\Longrightarrow f[n-l_1, m-l_1] ** h[n-l_2, m-l_2] \\ &= g[n-l_1-l_2, m-l_1-l_2] \end{aligned}$$

Convolution vs. (Cross) Correlation

- When is correlation equivalent to convolution?
- In other words, when is f**g = f*g?

Convolution vs. (Cross) Correlation

- A **convolution** is an integral that expresses the amount of overlap of one function as it is shifted over another function.
  - convolution is a filtering operation

- **Correlation** compares the **similarity** of **two** sets of **data**. Correlation computes a measure of similarity of two input signals as they are shifted by one another. The correlation result reaches a maximum at the time when the two signals match best .
  - correlation is a measure of relatedness of two signals

What we have learned today?

- Color
  - A **psychological property** of our visual experiences
  - Color spaces
- Image sampling and quantization
  - Images are samples consisting of discrete pixels which are quantized
  - Resolution: a sampling parameter
- Image histograms
  - frequency of brightness within the image

What we have learned today?

- Images as functions
  - discrete representations as a matrix of integer values
  - function f defined over a rectangle with finite range
- Image filtering
  - Point operation
  - Linear shift-invariant image filtering
  - Box filter and general filter version

What we have learned today?

- Convolution and correlation
  - Convolution
  - (Cross)Correlation
  - Convolution is a filtering operation
  - Correlation compares the similarity of two sets of data