CAP 4453
Robot Vision

Dr. Gonzalo Vaca-Castaño

Gonzalo.vacacastano@ucf.edu

Administrative details

- Homework 1 issues ?

Questions?

Robot Vision

3. Image Filtering

Credits

- Some slides comes directly from:
  - Yogesh S Rawat (UCF)
  - Noah Snavely (Cornell)
  - Ioannis (Yannis) Gkioulekas (CMU)
  - Mubarak Shah (UCF)
  - S. Seitz
  - James Tompkin
  - Ulas Bagci

Outline (next 2 weeks)

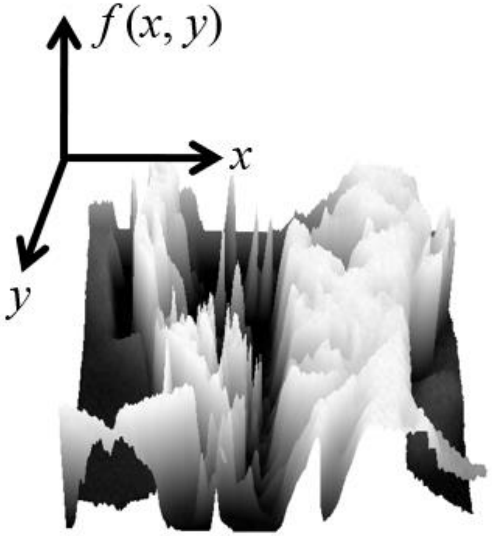
- ~~Image as a function~~
  - Linear algebra
- Extracting useful information from Images
  - ~~Histogram~~
  - Noise
  - Filtering (linear)
  - Smoothing/Removing noise
  - Convolution/Correlation
  - Image Derivatives/Gradient
  - Edges
- Colab Notes/ homeworks
- Read Szeliski, Chapter 3.
- Read/Program CV with Python, Chapter 1.

From last class

What is an image?

- We can think of a (grayscale) image as a **function**, f, from R^2 to R:
  - $f(x,y)$ gives the **intensity** at position (x,y)

snoop

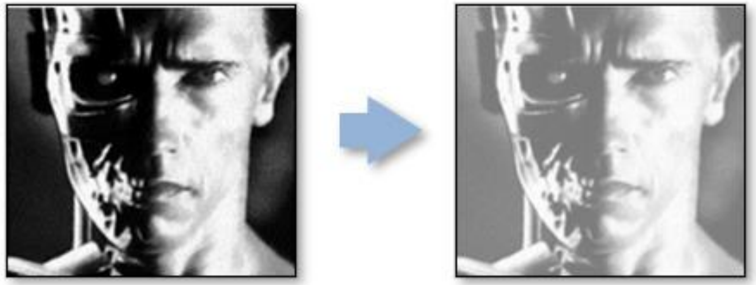


3D view

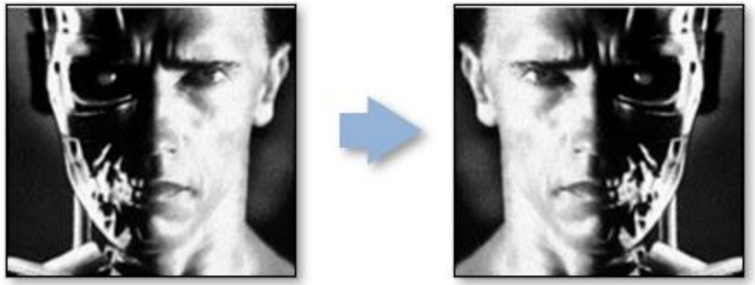
  - A **digital** image is a discrete (**sampled**, **quantized**) version of this function

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)

Basic Linear Algebra

Linear Algebra basics

- Vectors
  - Operations
- Matrix
  - Operations

Linear Algebra basics
Vector

- Scalar: $x \in \mathbb{R}$
- Vector: $\boldsymbol{x} \in \mathbb{R}^N$
  - Row Vector v $\in \mathbb{R}^{1\times n}$

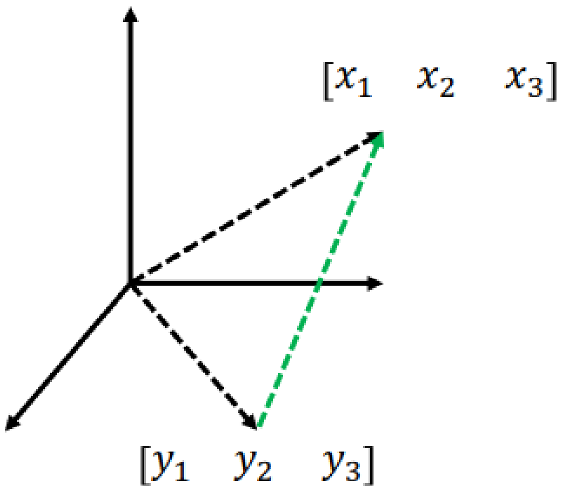
    $$\boldsymbol{x} = [x_1 \quad x_2 \quad \cdots \quad x_n]$$

  - Column vector v $\in \mathbb{R}^{n\times 1}$: $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = [x_1 \quad x_2 \quad \cdots \quad x_n]^{\boldsymbol{T}}$
  - Transpose

Linear Algebra Basics
Vectors - use

- Store data in memory
  - Feature vectors
  - Pixel values
  - Any other data for processing
- Any point in coordinate system
  - Can be n dimensional
- Difference between two points

$$[x_1 - y_1 \quad x_2 - y_2 \quad x_3 - y_3]$$

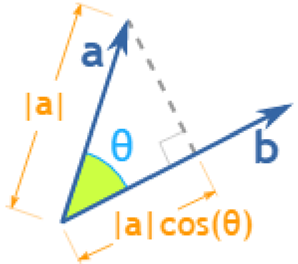


Linear Algebra Basics
Vector operations

- Norm – size of the vector
- p-norm $$\|x\|_p = \left(\sum_i |a_i|^p\right)^{\frac{1}{p}} \qquad p \geq 1$$
- Euclidean norm $$\|x\|_2 = \left(\sum_i |a_i|^2\right)^{1/2}$$
- L1-norm $$\|x\|_1 = \left(\sum_i |a_i|\right)$$
- L-infinity $$||\boldsymbol{x}||_\infty = \max_i |x_i|$$

Linear Algebra Basics
Vector operations

- Inner product (dot product)
  - Scalar number
  - Multiply corresponding entries and add



$$\boldsymbol{x}^T\,\boldsymbol{y} = [x_1 \quad x_2 \quad \cdots \quad x_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_k^n x_k\ y_k$$

[Dot Product (mathsisfun.com)]

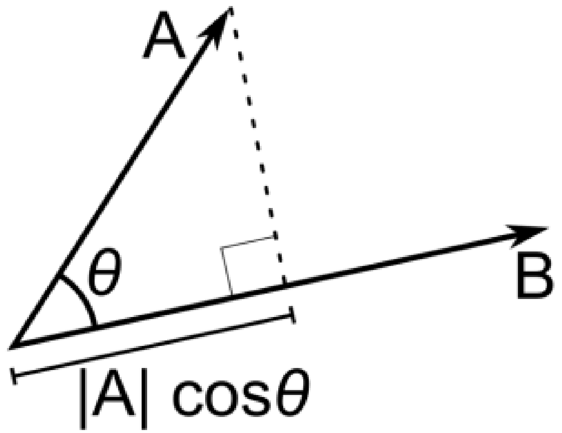
Linear Algebra Basics

Vector operations

- Inner product (dot product)

$$\boldsymbol{x}_i^T \boldsymbol{x}_i = \sum_k^n \left(x_k^i\right)^2 = \text{squared norm of } \boldsymbol{x}_i$$

- x.y is also |x||y|cos(angle between x and y)



- If B is a unit vector, A.B gives projection of A on B

Linear Algebra Basics
Vector operations

- Outer product

$$\boldsymbol{x}_i \boldsymbol{x}_j^T = \begin{bmatrix} x_1^i x_1^j & x_1^i x_2^j & \cdots & x_1^i x_n^j \\ x_2^i x_1^j & x_2^i x_2^j & \cdots & x_2^i x_2^j \\ \vdots & \vdots & \ddots & \vdots \\ x_n^i x_1^j & x_n^i x_2^j & \cdots & x_n^i x_m^j \end{bmatrix} \text{ (a matrix)}$$

Linear Algebra Basics

Matrix

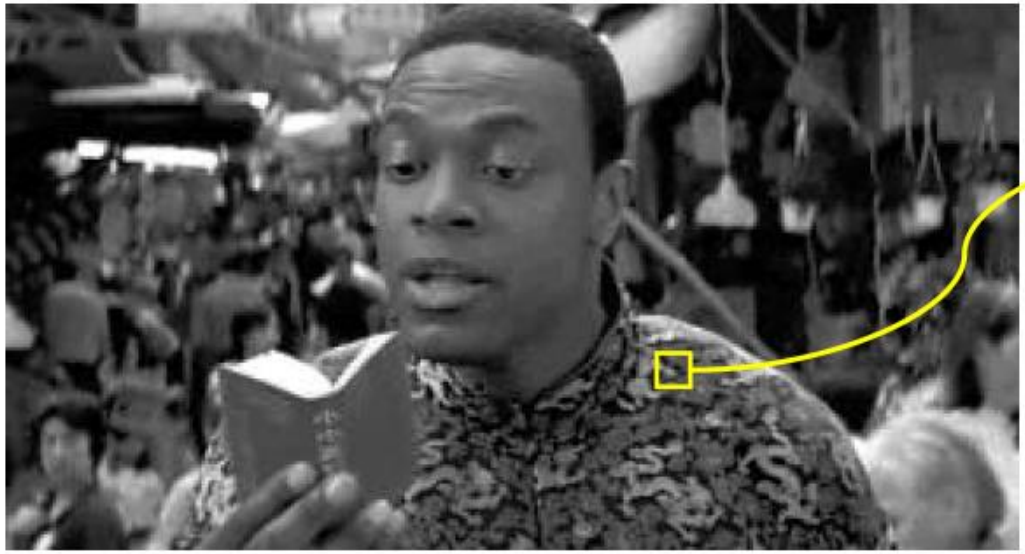
- Array $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ of numbers with shape m by n,
  - m rows and n columns

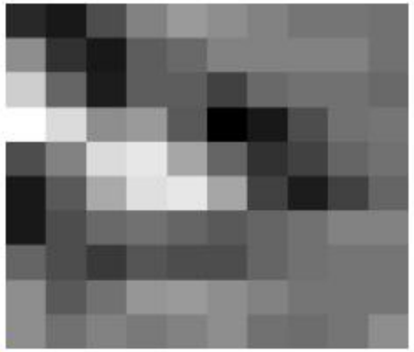
$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- A row vector is a matrix with single row
- A column vector is a matric with single column

Linear Algebra Basics
Matrix - use

- Image representation – grayscale
  - One number per pixel
  - Stored as nxm matrix



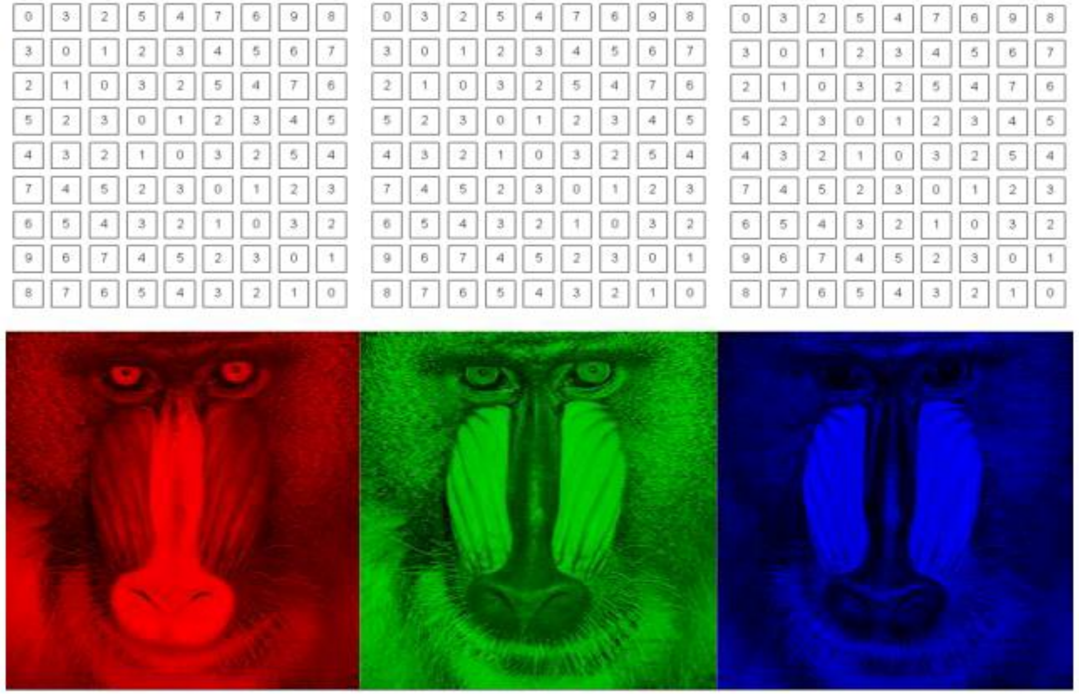


0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

Linear Algebra Basics

Matrix - use

- Image representation – RGB
  - 3 numbers per pixel
  - Stored as nxmx3 matrix



Linear Algebra Basics
Matrix operations

- Addition

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

- Both matrices should have same shape, except with a scalar

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + 2 = \begin{bmatrix} a+2 & b+2 \\ c+2 & d+2 \end{bmatrix}$$

- Same with subtraction

Linear Algebra Basics

Matrix operations

- Scaling

$$s \text{ x} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} s\text{x}a & s\text{x}b \\ s\text{x}c & s\text{x}d \end{bmatrix}$$

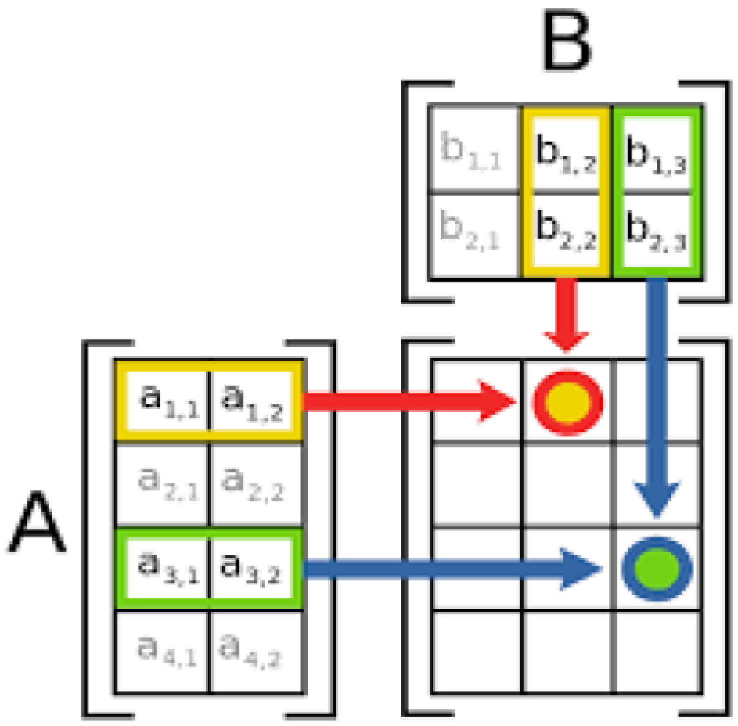
- Hadamard product

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \odot \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a\text{x}e & b\text{x}f \\ c\text{x}g & d\text{x}h \end{bmatrix}$$

Linear Algebra Basics

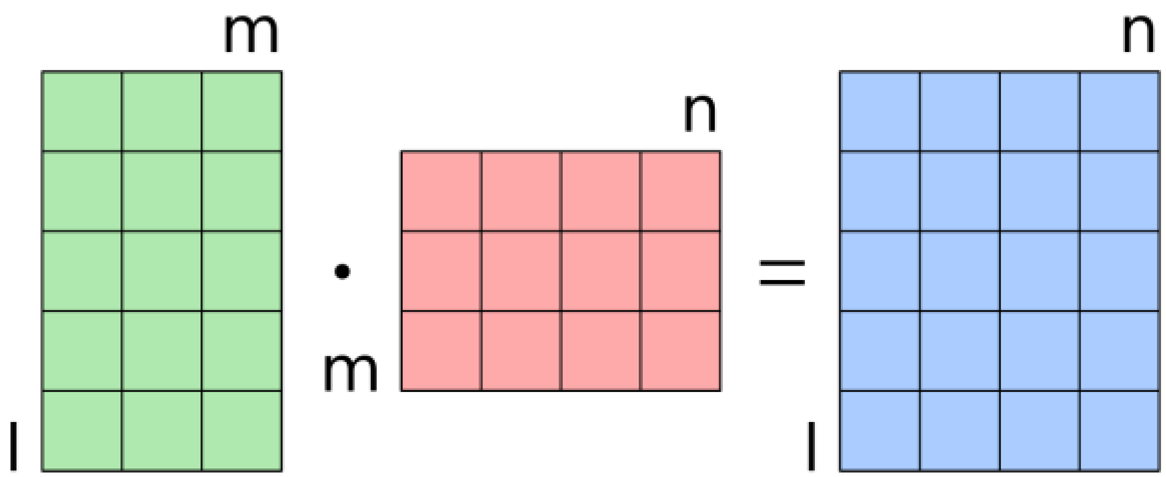
Matrix operation

- Matrix Multiplication
  - Compatibility?
  - mxn and nxp
  - Results in mxp matrix



Linear Algebra Basics

Matrix operation



$A \cdot B = C$

Linear Algebra Basics

Matrix operation

- Transpose

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

$$\boldsymbol{A}^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

Linear Algebra Basics

Matrix operation

- Inverse
  - Given a matrix A, its inverse A^{-1} is a matrix such that

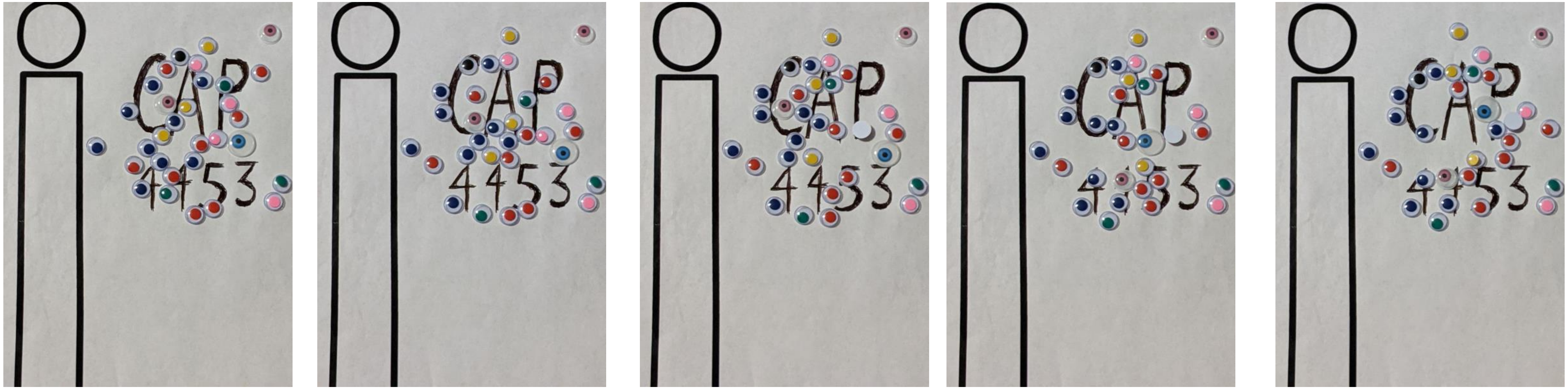
    $$\mathbf{AA^{-1} = A^{-1}A = I}$$
- Inverse does not always exist
  - Singular vs non-singular
- Properties
  - $(A^{-1})^{-1} = A$
  - $(AB)^{-1} = B^{-1}A^{-1}$

Linear Algebra Basics

MORE WILL BE INTRODUCED DURING THE COURSE AS IT IS NEEDED

Question: Noise reduction

- Given a camera and a still scene, how can you reduce noise?

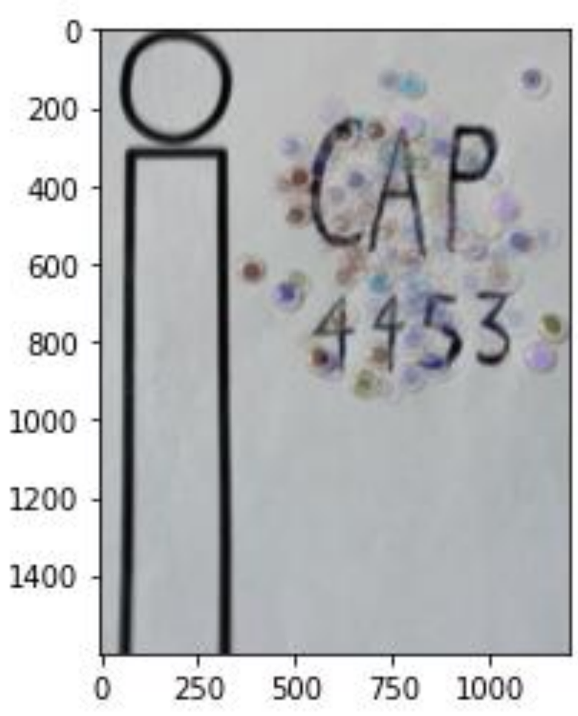


Take lots of images and average them!

Source: S. Seitz

Question: Noise reduction

- Given a camera and a still scene, how can you reduce noise?



Take lots of images and average them!

Can we something else?

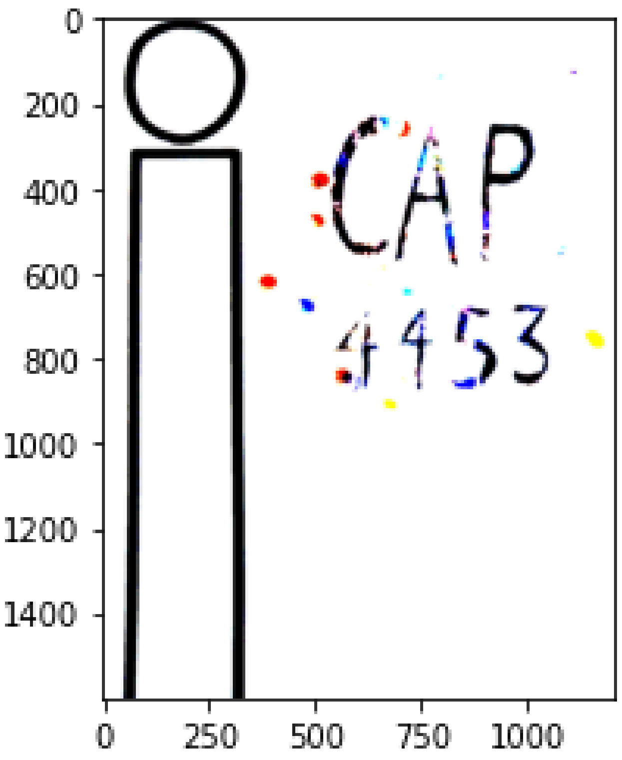
Source: S. Seitz

Thresholding !

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

Question: Noise reduction

- This is not a gray scale image




```python
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt

folder='C:/Users/gonza/OneDrive/Teaching/CAP4453/class3/'
list_dir = [fil for fil in os.listdir(folder ) if fil[-3:]=='jpg']

for iFile, fname in enumerate(list_dir):
    if iFile == 0:
        sumFile = cv2.imread(folder + fname)
        sumFile = sumFile.astype(np.float)
    else:
        sumFile = sumFile + cv2.imread(folder + fname).astype(np.float)

sumFile = sumFile/len(list_dir)
sumFile[sumFile>90]=255
sumFile[sumFile<=90]=0

plt.imshow(sumFile.astype(np.uint8))
```


Source: S. Seitz

Outline

- ~~Image as a function~~
    - ~~Linear algebra~~
- Extracting useful information from Images
    - ~~Histogram~~
    - Noise
    - Filtering (linear)
    - Smoothing/Removing noise
    - Convolution/Correlation
    - Image Derivatives/Gradient
    - Edges
- Colab Notes/ homeworks
- Read Szeliski, Chapter 3.
- Read/Program CV with Python, Chapter 1.

Image noise

- Light Variations
- Camera Electronics
- Surface Reflectance
- Lens

- Noise is random,
  - it occurs with some probability
  - It has a distribution

Additive Noise

$$I_{observed}\,(x, y) = I_{original}(x, y) + n(x, y)$$

True pixel value at x,y

Noise at x,y

Multiplicative Noise

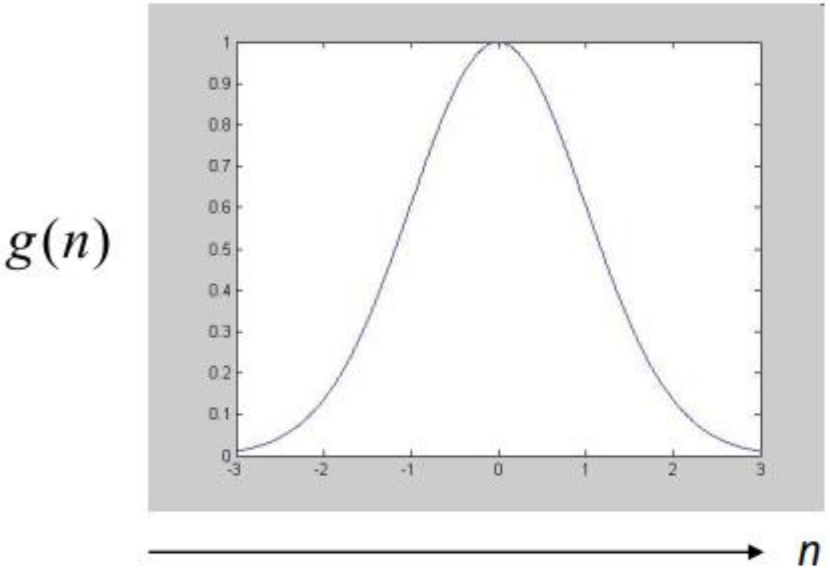
$$I_{observed}(x,y) = I_{original}(x,y) \times n(x,y)$$

True pixel value at x,y

Noise at x,y

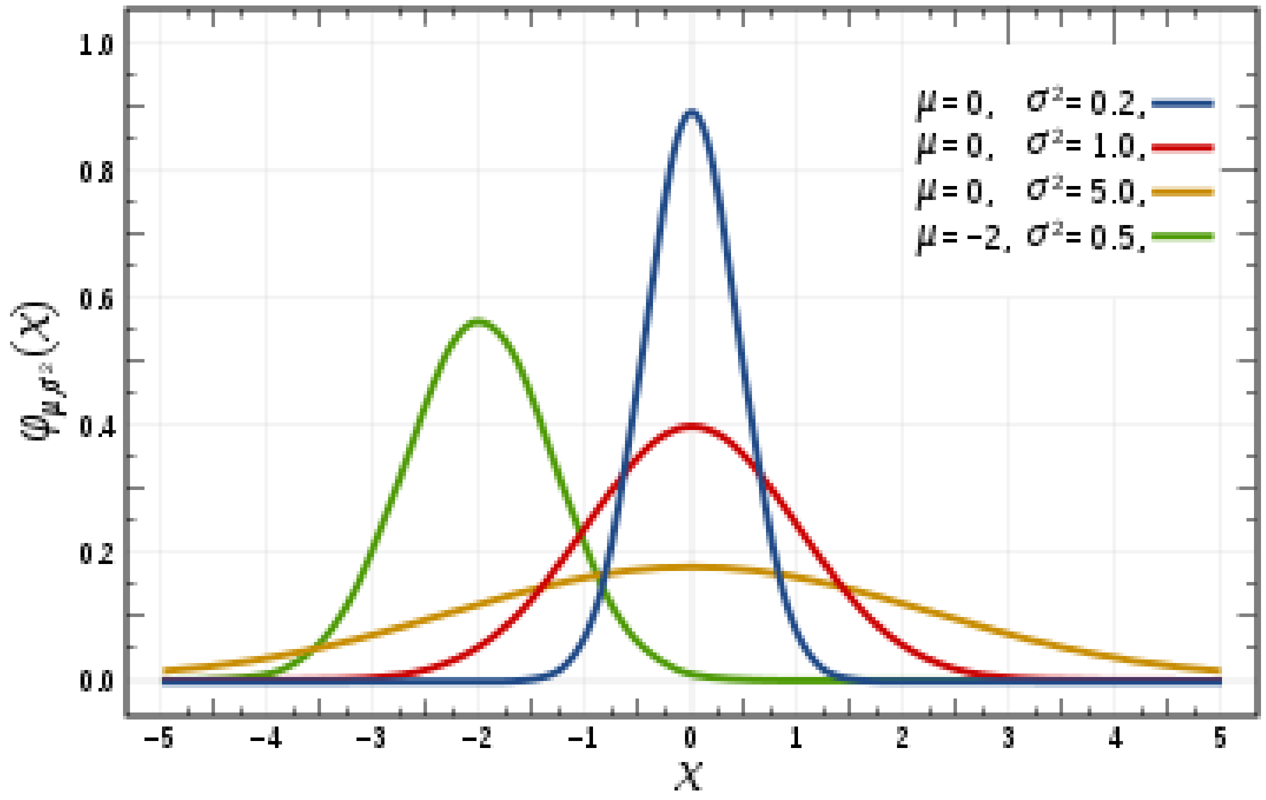
Gaussian Noise

$$n(x,y) \approx g(n) = e^{\frac{-n^2}{2\sigma^2}}$$



Probability Distribution
n is a random variable

Gaussian function



$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right).$$

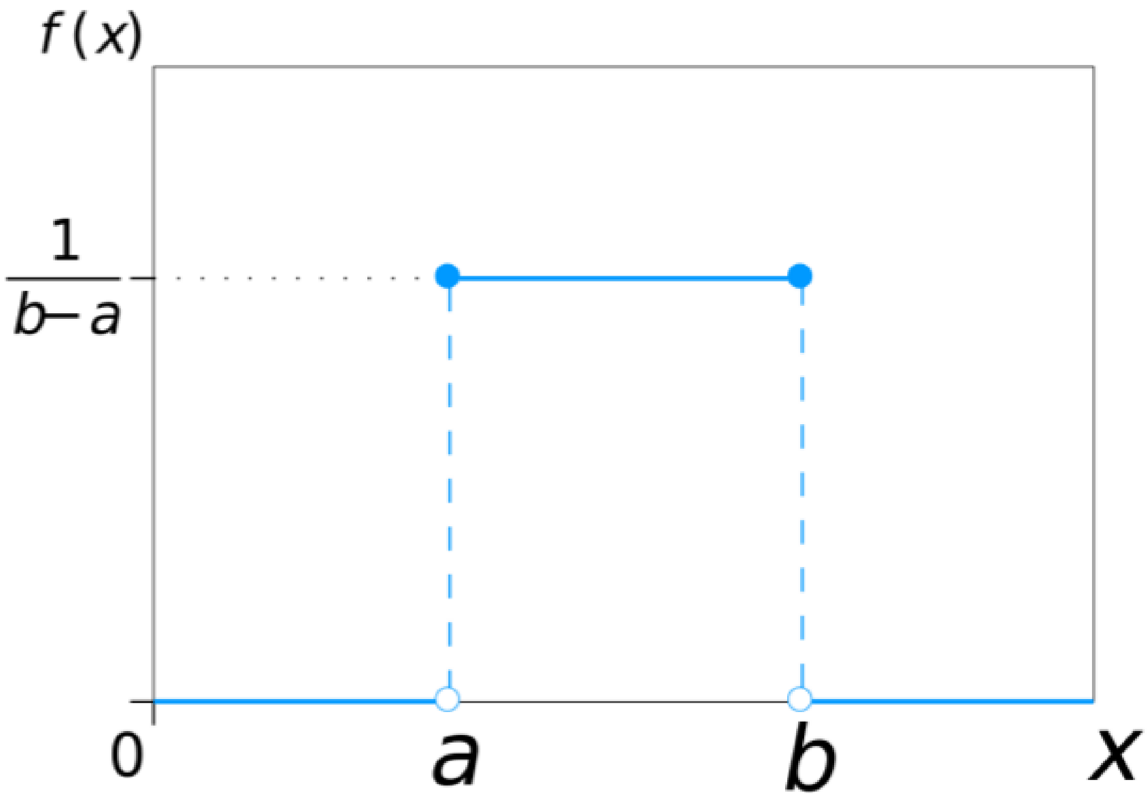
Salt and pepper noise

- Each pixel is randomly made black or white with a uniform probability distribution

➔

Salt-pepper

Uniform distribution



Noise implementation

```
#Parameters
#----------
#image : ndarray
#    Input image data. Will be converted to float.
#mode : str
#    One of the following strings, selecting the type of noise to add:

#    'gauss'     Gaussian-distributed additive noise.
#    'poisson'   Poisson-distributed noise generated from the data.
#    's&p'       Replaces random pixels with 0 or 1.
#    'speckle'   Multiplicative noise using out = image + n*image,where
#                n,is uniform noise with specified mean & variance.
import numpy as np
import os
import cv2

def noisy(noise_typ,image):
    if noise_typ == "gauss":
        row,col,ch= image.shape
        mean = 0
        var = 1
        sigma = var**0.5
        gauss = np.random.normal(mean,sigma,(row,col,ch))
        gauss = gauss.reshape(row,col,ch)
        noisy = image + gauss
        return noisy
    elif noise_typ == "s&p":
        row,col,ch = image.shape
        s_vs_p = 0.5
        amount = 0.004
        out = image
        # Salt mode
        num_salt = np.ceil(amount * image.size * s_vs_p)
        coords = [np.random.randint(0, i - 1, int(num_salt))
                  for i in image.shape]
        out[coords] = 1

        # Pepper mode
        num_pepper = np.ceil(amount* image.size * (1. - s_vs_p))
        coords = [np.random.randint(0, i - 1, int(num_pepper))
                  for i in image.shape]
        out[coords] = 0
        return out
    elif noise_typ == "poisson":
        vals = len(np.unique(image))
        vals = 2 ** np.ceil(np.log2(vals))
        noisy = np.random.poisson(image * vals) / float(vals)
        return noisy
    elif noise_typ =="speckle":
        row,col,ch = image.shape
        gauss = np.random.randn(row,col,ch)
        gauss = gauss.reshape(row,col,ch)
        noisy = image + image * gauss
        return noisy
```

Outline

- ~~Image as a function~~
- Extracting useful information from Images
  - ~~Histogram~~
  - ~~Noise~~
  - **Filtering (linear)**
  - Smoothing/Removing noise
  - Convolution/Correlation
  - Image Derivatives/Gradient
  - Edges
- Colab Notes/ homeworks
- Read Szeliski, Chapter 3.
- Read/Program CV with Python, Chapter 1.

Filters

- Filtering
  - Form a new image whose pixels are a combination of the original pixels
- Why?
  - To get useful information from images
    - E.g., extract edges or contours (to understand shape)
  - To enhance the image
    - E.g., to remove noise
    - E.g., to sharpen and “enhance image” a la CSI
  - A key operator in Convolutional Neural Networks

Linear shift-invariant image filtering

- Replace each pixel by a *linear* combination of its neighbors (and possibly itself).
- The combination is determined by the filter's *kernel*.
- The same kernel is *shifted* to all pixel locations so that all pixels use the same linear combination of their neighbors.

Filtering

- Modify pixels based on some function of neighborhood

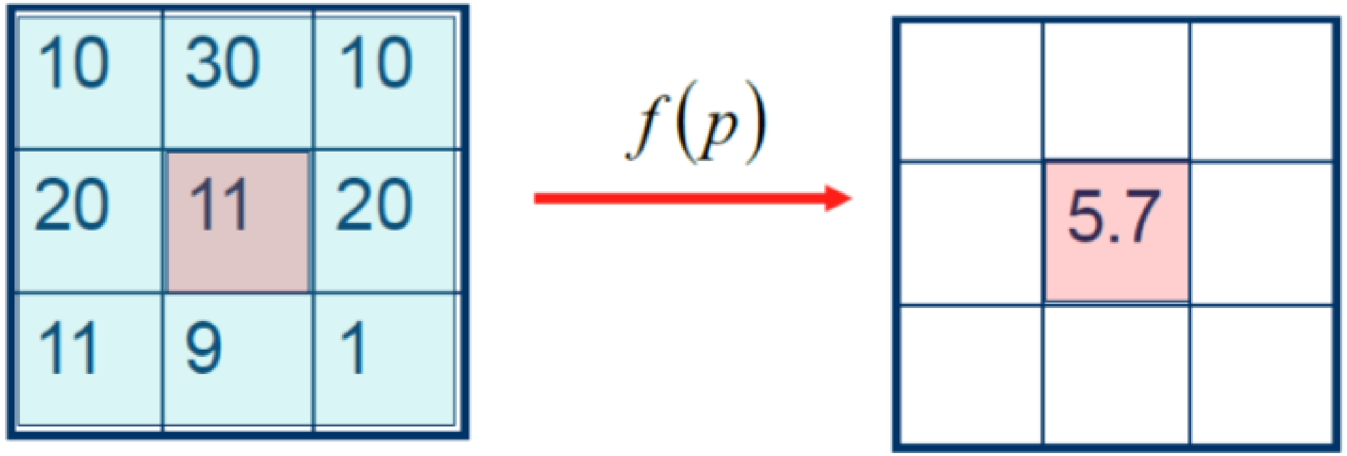


Image filtering

- Image filtering: compute function of local neighborhood at each position

(kernel)

h=output f=filter I=image

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

2d coords=k,l 2d coords=m,n

[ ] [ ] [ ]

Image filtering

- Image filtering: compute function of local neighborhood at each position

- Enhance images
  - Denoise, resize, increase contrast, etc.
- Extract information from images
  - Texture, edges, distinctive points, etc.
- Detect patterns
  - Template matching

Let's run the box filter

Box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

note that we assume that the kernel coordinates are centered

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output   filter   image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0								

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0								

shift-invariant: as the pixel shifts, so does the kernel

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}} \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10							

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10							

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20						

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20						

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30					

$$h[m,n] = \sum_{k,l} g[k,l]\,f[m+k,n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30	30				

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output — filter — image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30			

$$h[m,n] = \sum_{k,l} g[k,l]\,f[m+k,n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20		

$$h[m,n] = \sum_{k,l} g[k,l] f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0								

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0	20							

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0	20	40						

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0								

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

kernel $g[\cdot,\cdot]$

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30							

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

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

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

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

$$h[m,n] = \sum_{k,l} g[k,l] f[m+k, n+l]$$

output filter image (signal)

… and the result is

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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	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	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

output $h[\cdot,\cdot]$

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

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

output filter image (signal)

Correlation (linear relationship)

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

$$f = \text{Image}$$

$$h = \text{Kernel}$$

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

$$\begin{aligned} f \otimes h = & f_1h_1 + f_2h_2 + f_3h_3 \\ & + f_4h_4 + f_5h_5 + f_6h_6 \\ & + f_7h_7 + f_8h_8 + f_9h_9 \end{aligned}$$

Convolution

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

f = Image

h = Kernel

$$h = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \xrightarrow{X-flip} \begin{bmatrix} h_7 & h_8 & h_9 \\ h_4 & h_5 & h_6 \\ h_1 & h_2 & h_3 \end{bmatrix} \xrightarrow{Y-flip} \begin{bmatrix} h_9 & h_8 & h_7 \\ h_6 & h_5 & h_4 \\ h_3 & h_2 & h_1 \end{bmatrix}$$

$$f = \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} * \begin{bmatrix} h_9 & h_8 & h_7 \\ h_6 & h_5 & h_4 \\ h_3 & h_2 & h_1 \end{bmatrix} \longrightarrow$$

$$\begin{aligned} f * h = {} & f_1 h_9 + f_2 h_8 + f_3 h_7 \\ & + f_4 h_6 + f_5 h_5 + f_6 h_4 \\ & + f_7 h_3 + f_8 h_2 + f_9 h_1 \end{aligned}$$

Correlation and Convolution

- Convolution is a filtering operation
  - expresses the amount of overlap of one function as it is shifted over another function

- Correlation compares the similarity of two sets of data
  - relatedness of the signals!

Key properties of linear filters

Linearity:

$\texttt{filter}(f_1 + f_2) = \texttt{filter}(f_1) + \texttt{filter}(f_2)$

Shift invariance: same behavior regardless of pixel location

$\texttt{filter}(\texttt{shift}(f)) = \texttt{shift}(\texttt{filter}(f))$

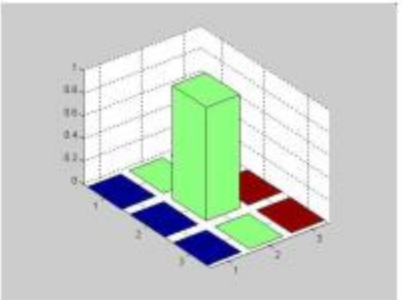
Any linear, shift-invariant operator can be represented as a convolution

Source: S. Lazebnik

More properties

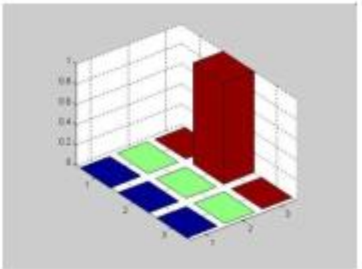
- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
  - But particular filtering implementations might break this equality
- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: a * $(b_1 * b_2 * b_3)$
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k\,(a * b)$
- Identity: unit impulse e = [0, 0, 1, 0, 0],
  $a * e = a$

Filtering Examples - 1



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

Filtering Examples - 2



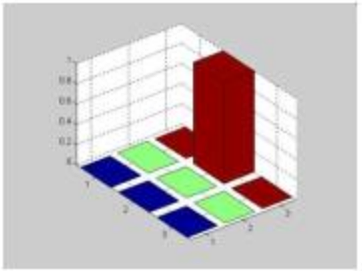
$*$

0	0	0
1	0	0
0	0	0

$=$

Filtering Examples - 2

*



0	0	0
1	0	0
0	0	0

=

Example: box filter

Average: mean

- Dividing the sum of N values by N

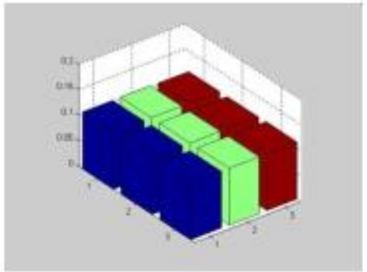
What does it do?

- Replaces each pixel with an average of its neighborhood

$$g[\cdot\,,\cdot\,]$$

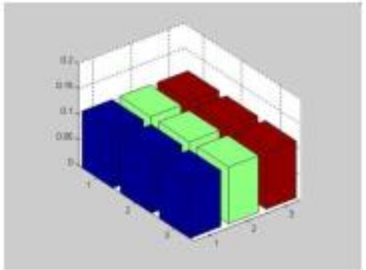
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Filtering Examples - 3



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

Filtering Examples - 3



$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$

Example: box filter

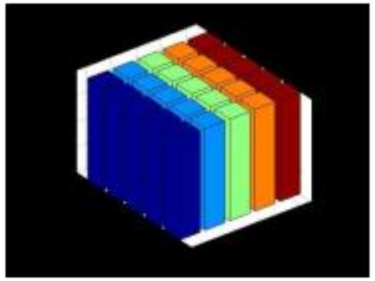
What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

$$g[\cdot\,,\cdot\,]$$

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

Filtering Examples - 4



$$* \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

Separable filters

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

example:
box filter

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\text{column}} * \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}}_{\text{row}}$$

What is the rank of this filter matrix?

Separable filters

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

example:
box filter

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\text{column}} * \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}}_{\text{row}}$$

What is the rank of this filter matrix?

Matrix rank is 1 for separable filters

```
s = svd(G);
sum(s > eps('single'))
```

Let's say our 2D Linear Operator is given by the Matrix $G \in \mathbb{R}^{n \times n}$.
Using the SVD Decomposition the operator can be written as:

$$G = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

Separable Linear 2D Operator is defined as operator which can be composed by Outer Product of 2 vectors.
Looking at the SVD Decomposition of G we can conclude that G is separable operator if and only if $\forall i > 1 \; \sigma_i = 0$ and it is given by:

$$G = \sigma_1 u_1 v_1^T$$

Usually LPF 2D Linear Operators, such as the Gaussian Filter, in the Image Processing world are normalized to have sum of 1 (Keep DC) which suggests $\sigma_1 = 1$ moreover, they are also symmetric and hence $u_1 = v_1$ (If you want, in those cases, it means you can use the Eigen Value Decomposition instead of the SVD).

So basically, to prove that a Linear 2D Operator is Separable you must show that it has only 1 non vanishing singular value.

image processing - How to Prove a 2D Filter Is Separable? - Signal Processing Stack Exchange

Separable filters

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

example:
box filter

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\text{column}} * \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}}_{\text{row}}$$

Why is this important?

Separable filters

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

example:
box filter

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\text{column}} * \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}}_{\text{row}}$$

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

Separable filters

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

example: box filter

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\text{column}} * \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}}_{\text{row}}$$

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?

Separable filters

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

example:
box filter

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \underset{\text{column}}{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}} * \underset{\text{row}}{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}}$$

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

Separable filters

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

example:
box filter

$$\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \underbrace{\begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline 1 \\ \hline \end{array}}_{\text{column}} * \underbrace{\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline \end{array}}_{\text{row}}$$

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:

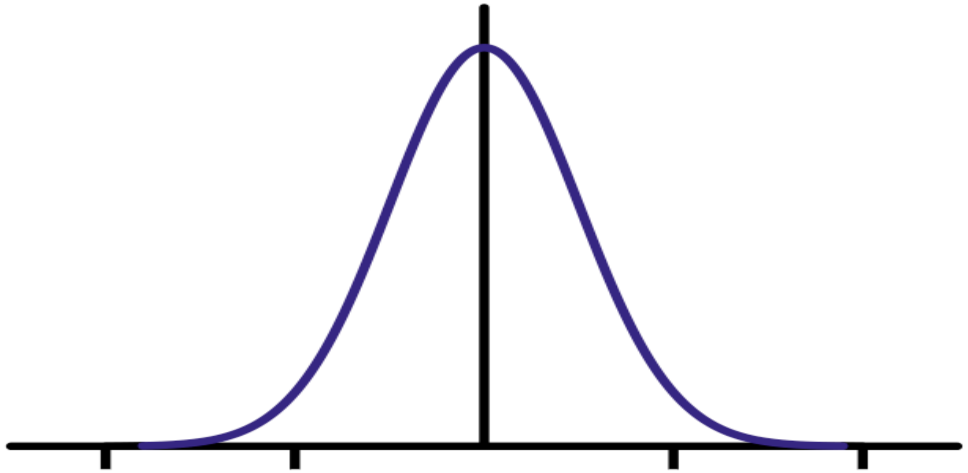
- What is the cost of convolution with a non-separable filter? $\longrightarrow$ $M^2 \times N^2$
- What is the cost of convolution with a separable filter? $\longrightarrow$ $2 \times N \times M^2$

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:

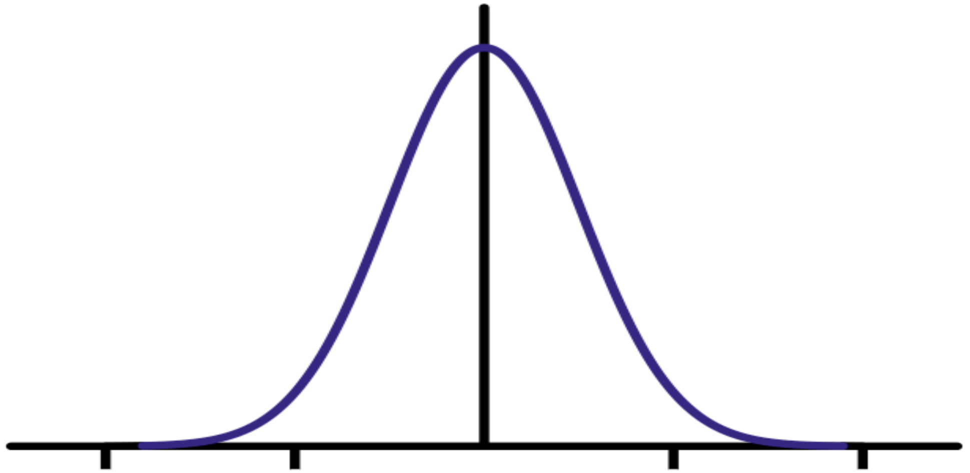
$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- weight falls off with distance from center pixel

- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?

- usually at 2-3σ



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:

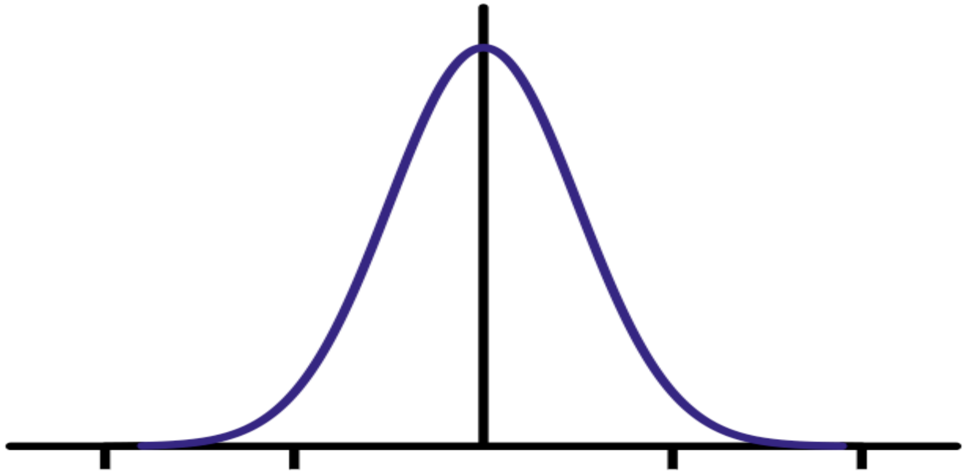
$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- weight falls off with distance from center pixel

- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?

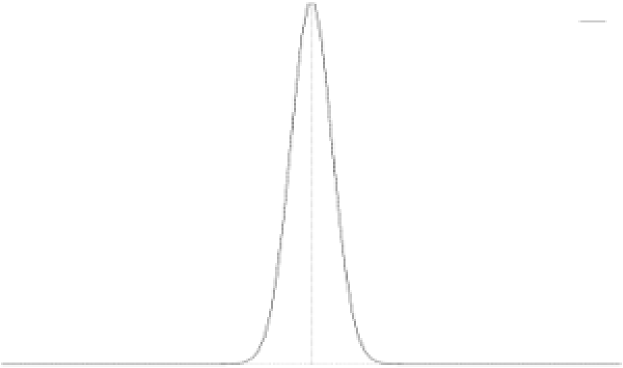
- usually at 2-3σ

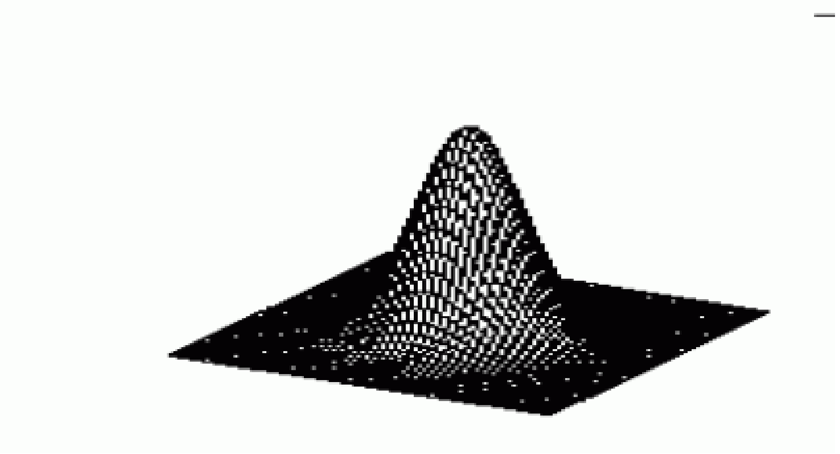


Is this a separable filter? Yes!

kernel $\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$

The Gaussian Filter

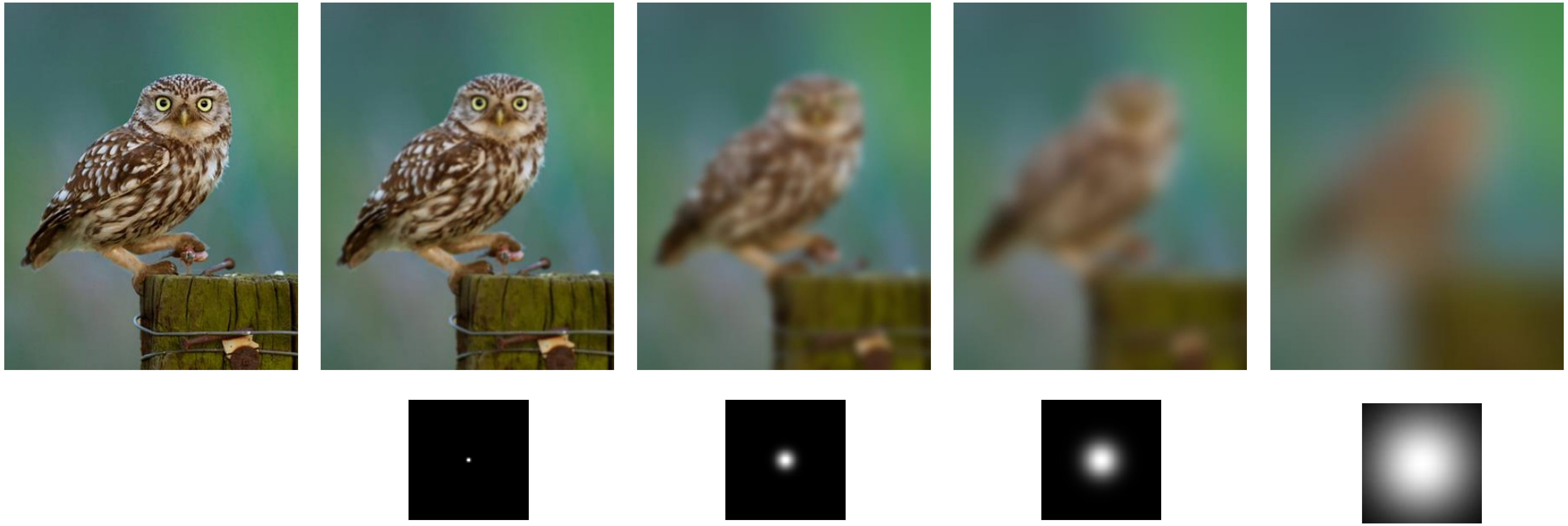




$$g(x) = e^{\frac{-x^2}{2\sigma^2}} \qquad g(x,y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

$$g(x) = [.011 \quad .13 \quad .6 \quad 1 \quad .6 \quad .13 \quad .011] \qquad \sigma = 1$$

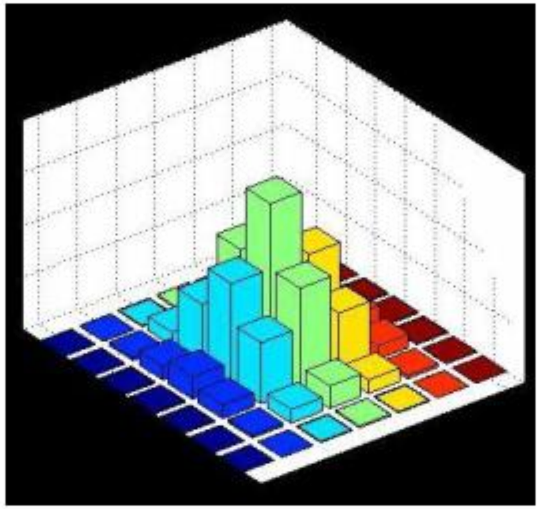
Gaussian filters



σ = 1 pixel σ = 5 pixels σ = 10 pixels σ = 30 pixels

Filtering Examples - 5

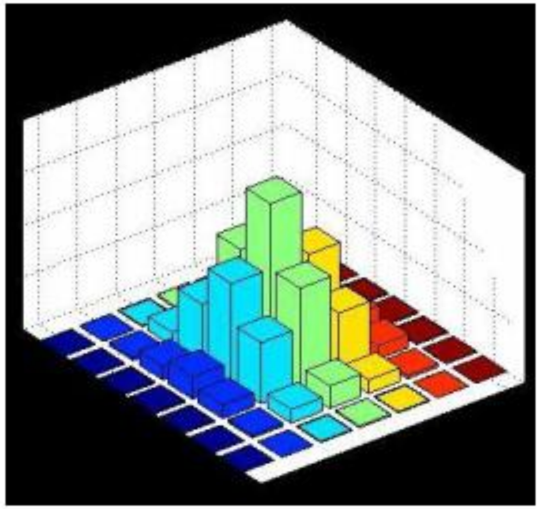
$*$



$=$

Filtering Examples - 5

*



=

Gaussian Smoothing

Filtering Examples - 6

Gaussian Smoothing

Smoothing by Averaging

Filtering Examples - 7

After additive
Gaussian Noise

After Averaging

After Gaussian Smoothing

Filtering Examples – 8
Sharpening

input

filter

$$\begin{bmatrix}0 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

(Note that filter sums to 1)

output

- do nothing for flat areas
- stress intensity peaks

Filtering Examples – 8
Sharpening

Accentuates differences with local average

input

filter

$$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} - \frac{1}{9} \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

output

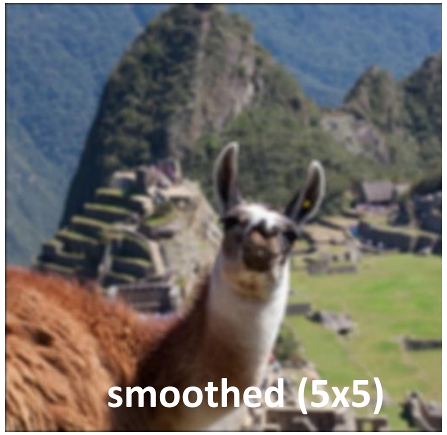
(Note that filter sums to 1)

- do nothing for flat areas
- stress intensity peaks

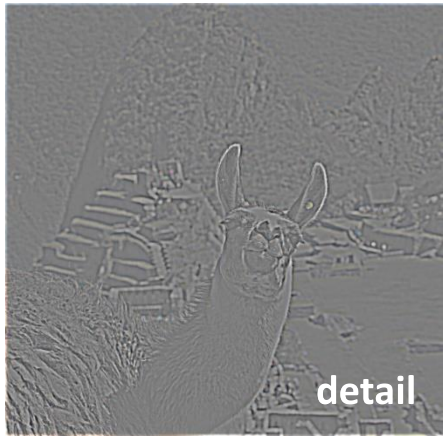
Sharpening

- What does blurring take away?

–



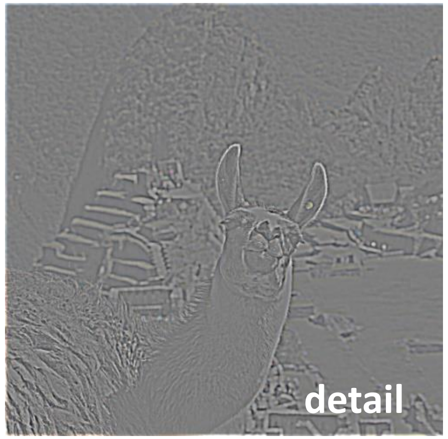
=



(This "detail extraction" operation is also called a ***high-pass filter***)

Let's add it back:

+



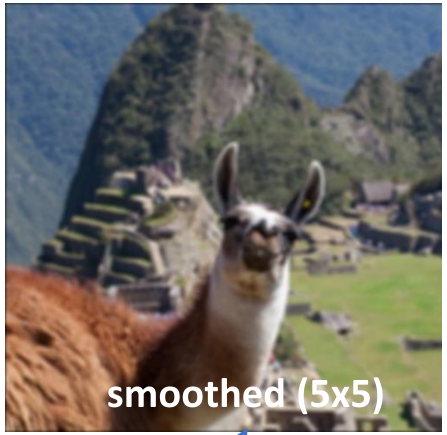
=

Photo credit: https://www.flickr.com/photos/geezaweezer/16089096376/

Sharpening

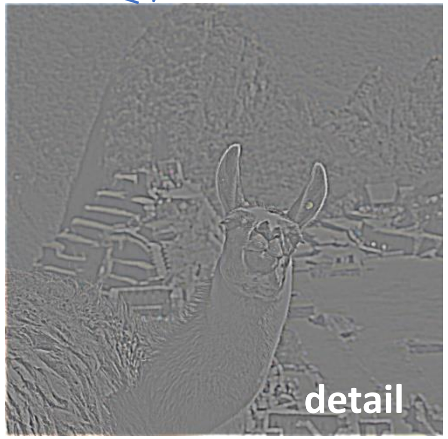
- What does blurring take away?

−



(This "detail extraction" operation is also called a ***high-pass filter***)

+



=

Photo credit: https://www.flickr.com/photos/geezaweezer/16089096376/

Sharpening

- What does blurring take away?

2 times original

Smoothed

$$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

(This “detail extraction” operation is also called a ***high-pass filter***)

+

–

=

Photo credit: https://www.flickr.com/photos/geezaweezer/16089096376/

Sharpening

- What does blurring take away?

(This "detail extraction" operation is also called a ***high-pass filter***)

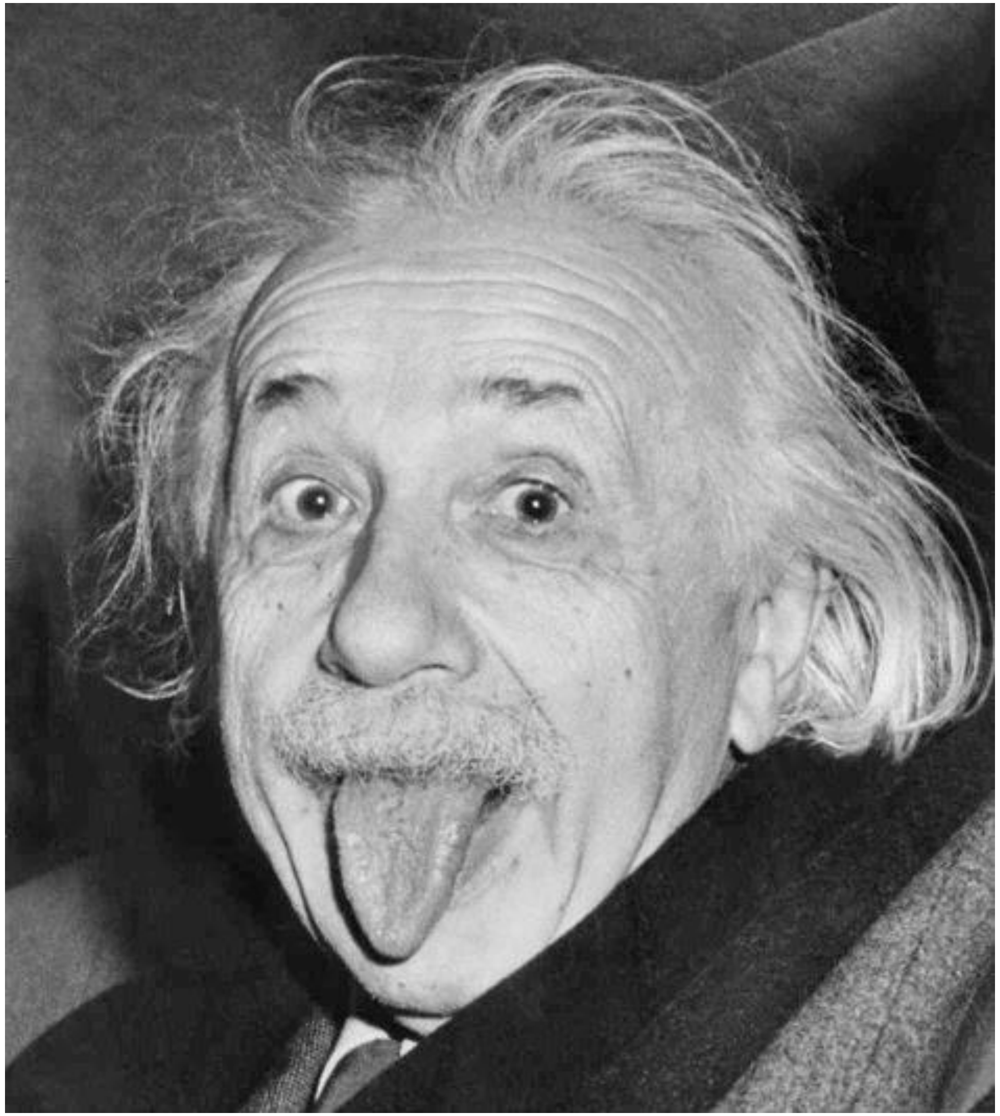
+

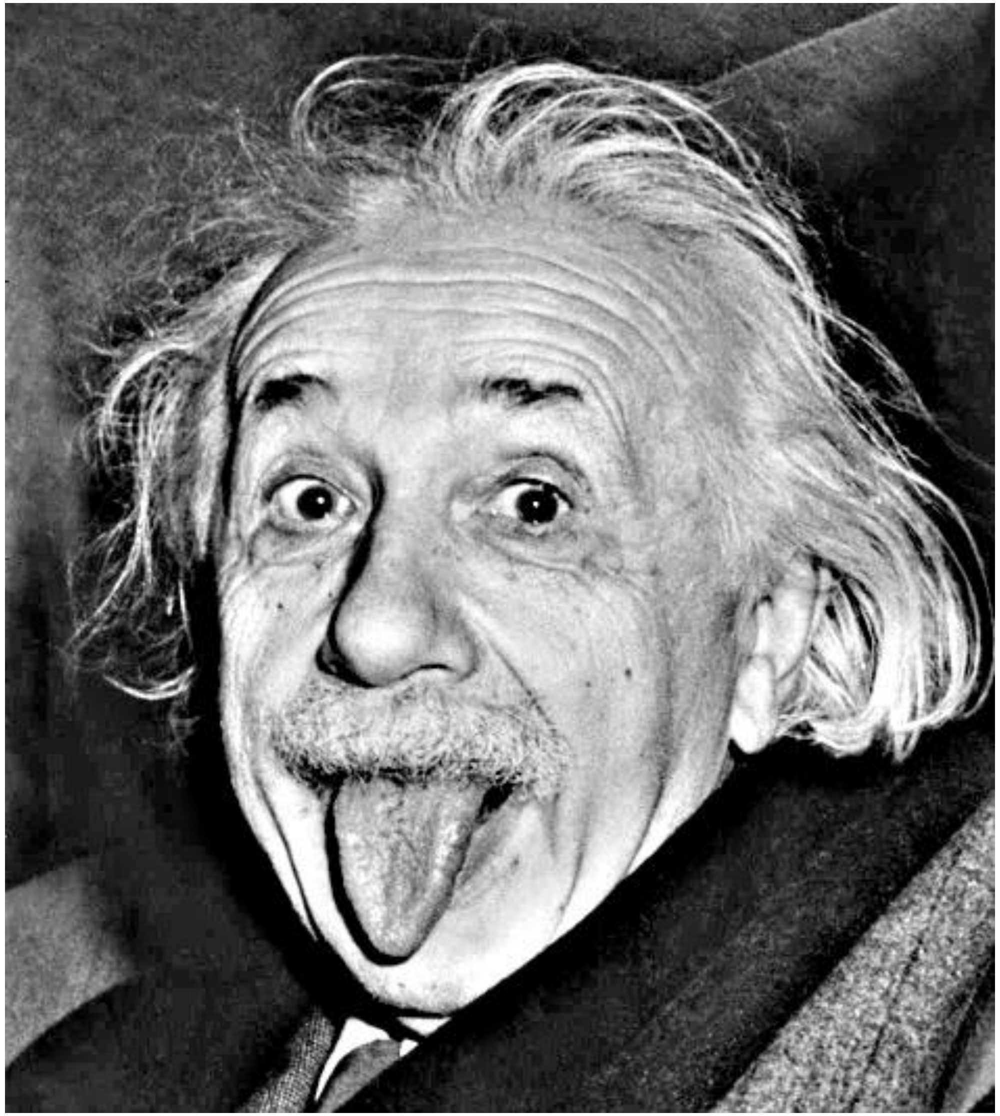
–

=

Photo credit: https://www.flickr.com/photos/geezaweezer/16089096376/

Sharpening examples





Median Filter

- A **Median Filter** operates over a window by selecting the median intensity in the window.

Image filtering - median

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
				?					

Image filtering - median

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
				90					

Median of {0,0,0,0, 90, 90,90,90,90}

Median Filter

- A **Median Filter** operates over a window by selecting the median intensity in the window.

- Great to deal with salt and pepper noise !

Median Filter

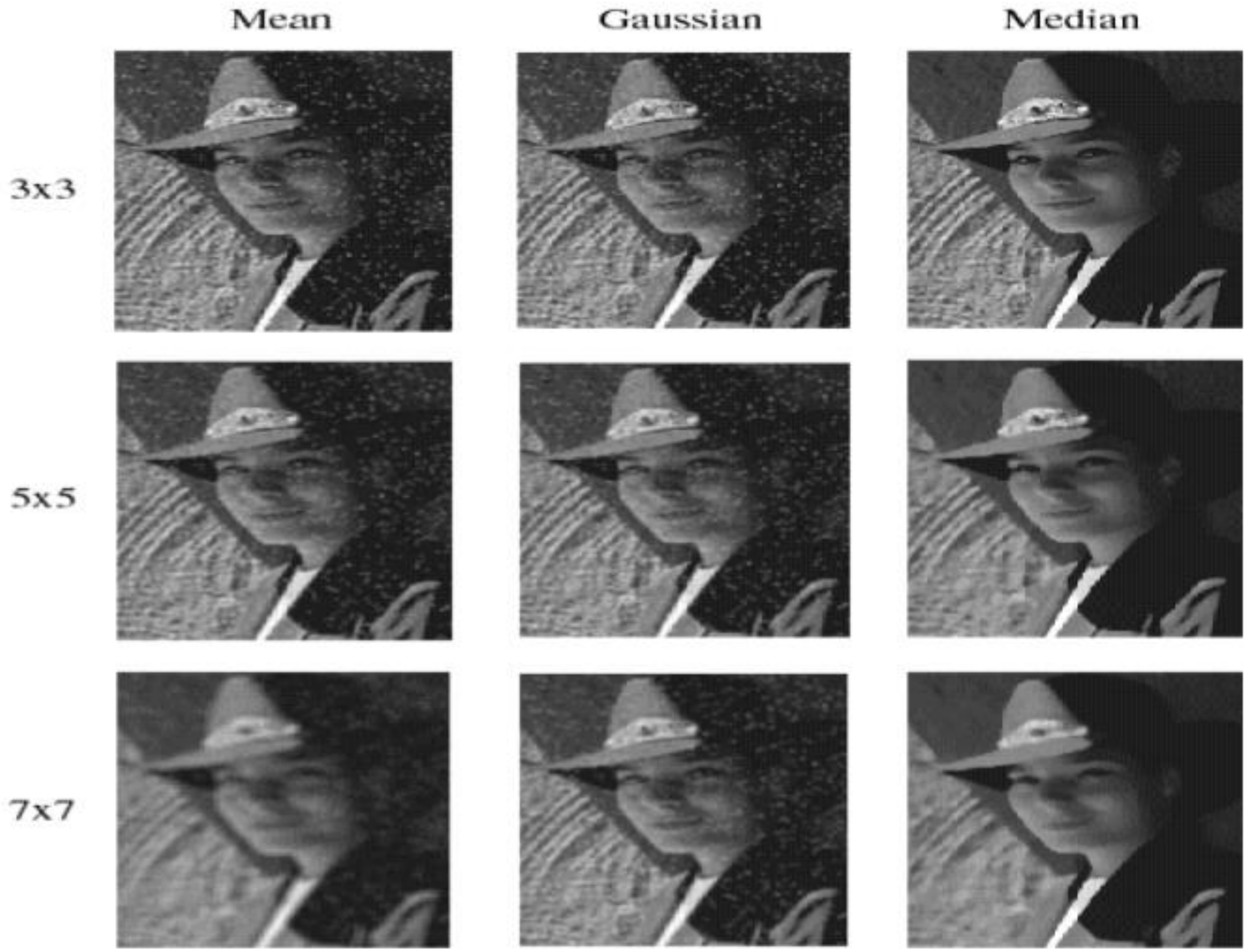


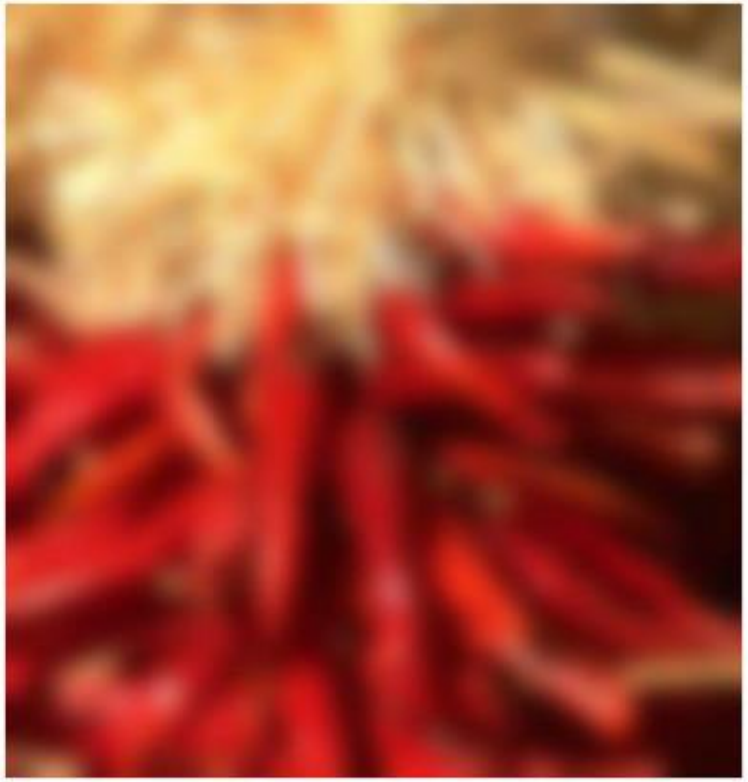
Image Boundary Effect

The filter window falls off at the edge of image.

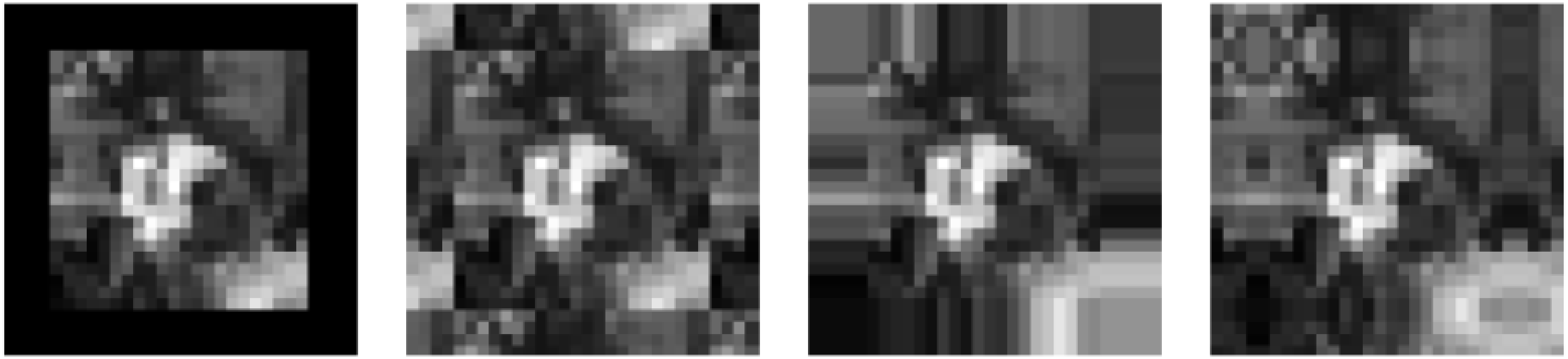
Practical matters

What about near the edge?

- The filter window falls off the edge of the image
- Need to extrapolate
- methods:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge



Source: S. Marschner



zero   wrap   clamp   mirror

Copy edge   Reflect across edge

Questions?