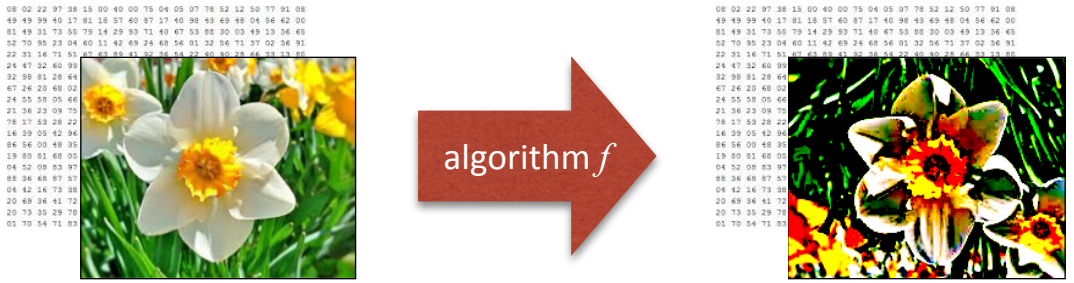
Image processing 101

Outline

- Introduction
  - What is image processing?
  - Relationship to other fields
  - Basic concepts
- Point operations
  - Scaling, inverse, gamma correction, histogram equalization
- Spatial filters
  - Filtering, correlation, convolution
  - Blurring, sharpening, edge detection
- Filtering in Fourier space
  - Frequency domain, low- and high-pass filters

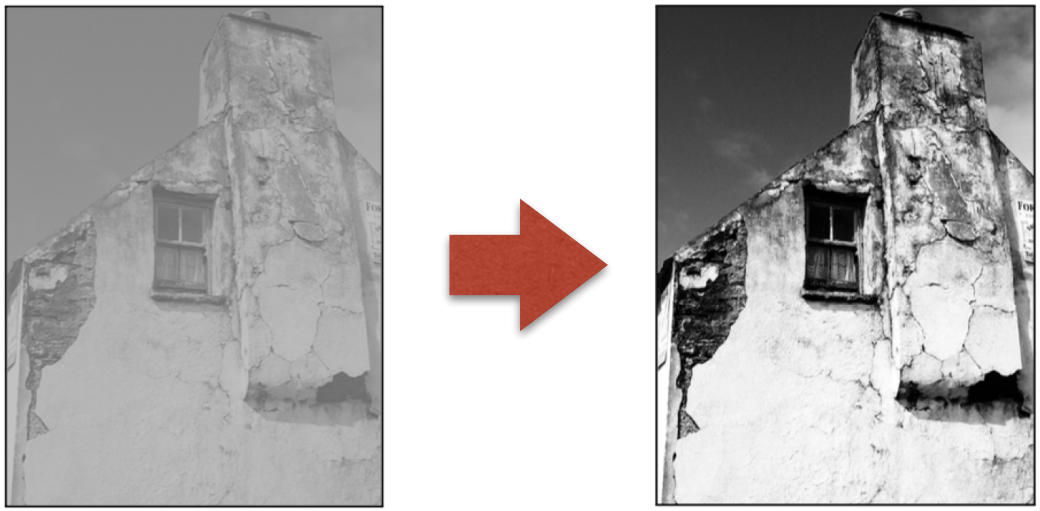
What is image processing? (1/3)

- Image processing is the study of any algorithm that takes an **image as input** and returns an **image as output**



I $\qquad$ $J = f(I)$

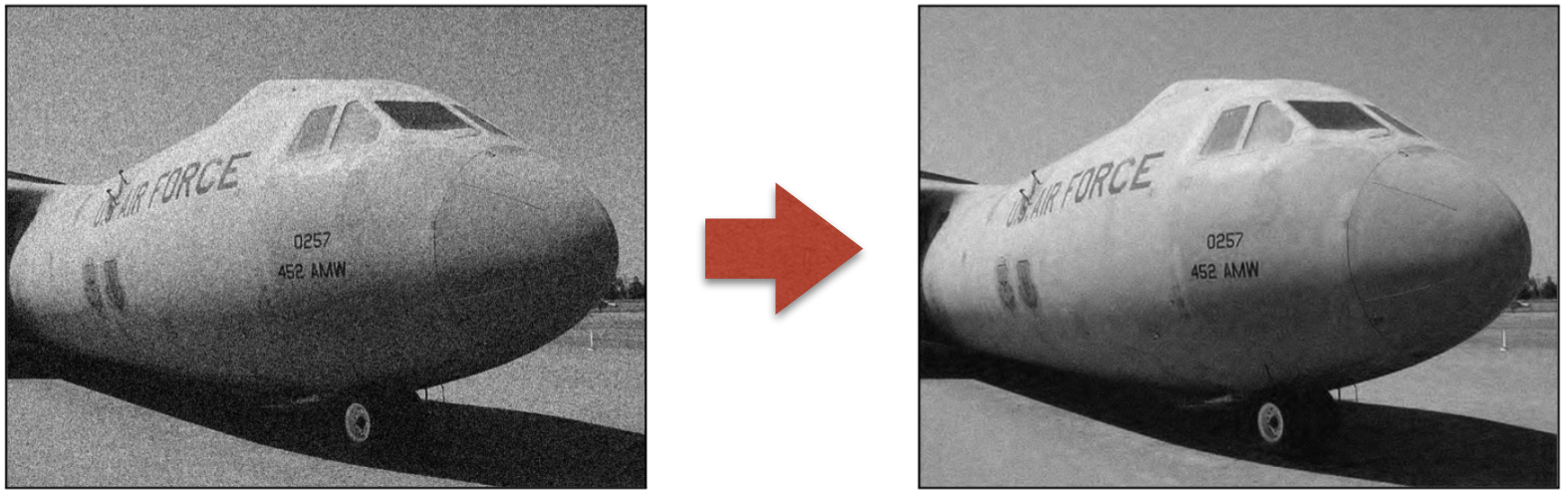
- Example: **contrast adjustment**



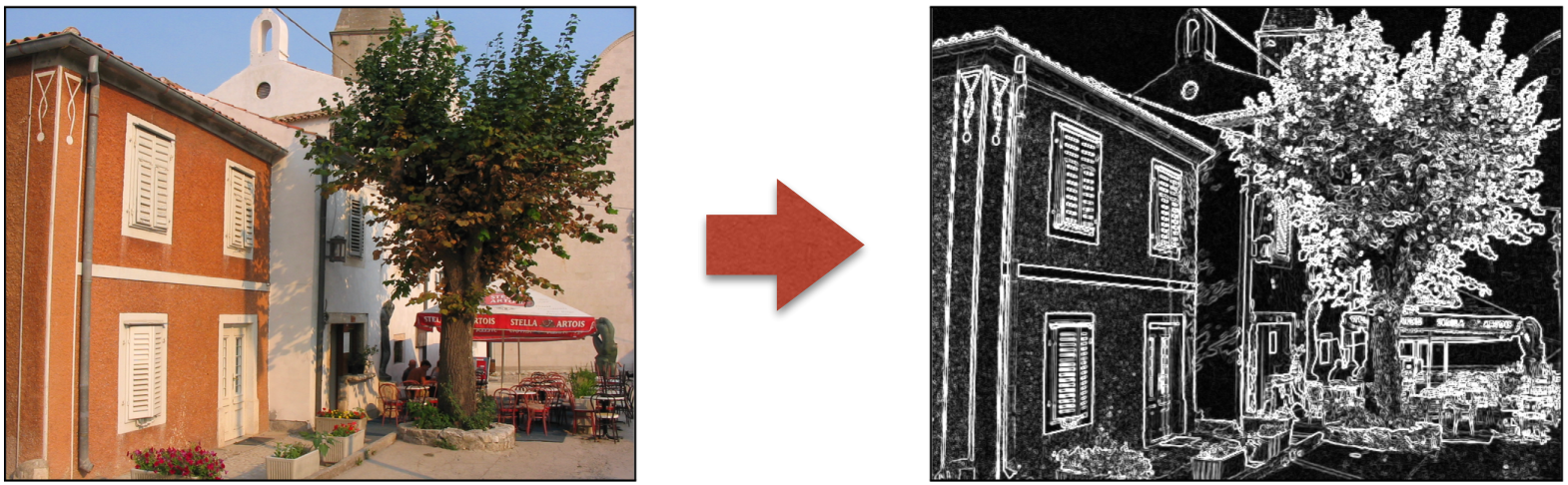
Input image : I $\qquad$ Output image : J

What is image processing? (2/3)

- Example: **noise removal**



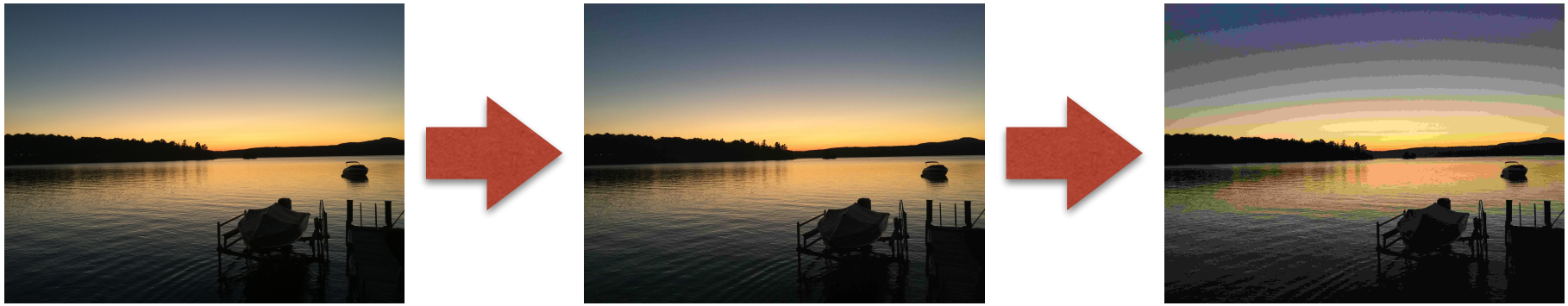
- Example: **edge detection**



What is image processing? (3/3)

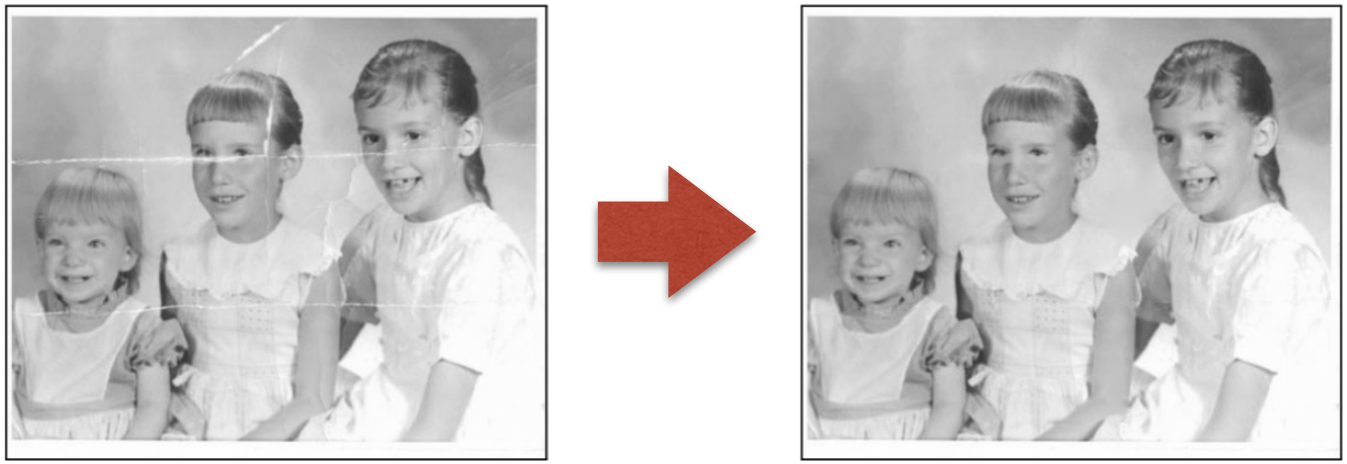
Advanced examples (not seen in this course)

- Compression



Original (1.9MB) | Compressed (230 KB, 12%) | Compressed (126 KB, 7%)

- Restoration



Damaged | Restored

Relationship to other fields

- Image Processing
  - Image enhancement, noise removal, feature detection …
- Image Analysis
  - Segmentation, image registration, matching …
  - Medical diagnosis from an image
- Computer Vision
  - Object detection/recognition, shape analysis, tracking
  - Use of Artificial Intelligence (AI) and Machine Learning (ML)

- **NOTE**
  - *Computer graphics*/*visualization* deals with the **synthesis of images**

Recall of basic concepts (1/2)

Images as functions

- We can think of the *intensity of an image* as a function of position (u, v)
- Let $\Omega \subset \mathbb{N}^2$ be the *image domain*. Then an image is a **discrete function**:

$$I : \Omega \rightarrow \mathbb{R}$$

Example

A simple image

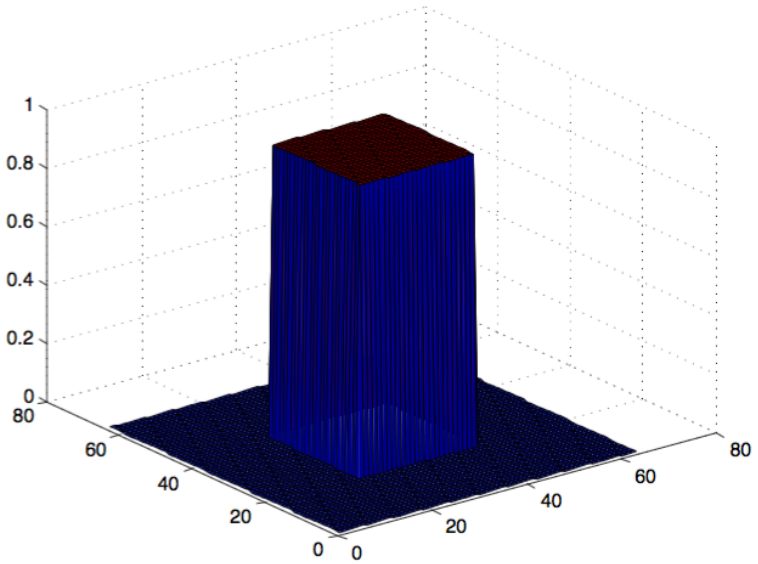
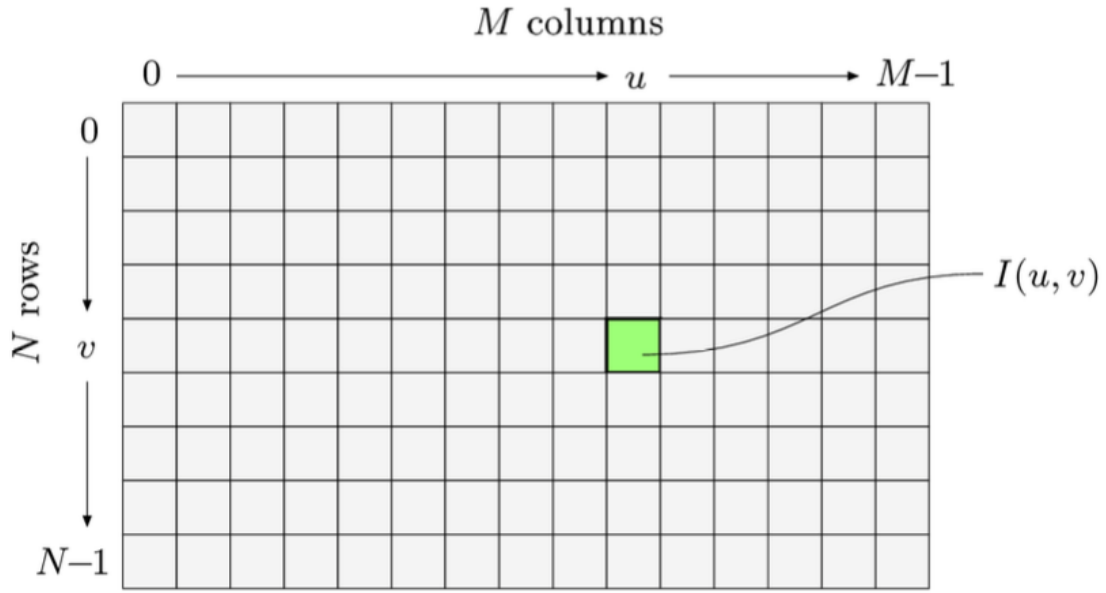


Image function as a height field

Recall of basic concepts (2/2)

Representing an image

- The data structure for an image is simply a **2D array of values**:



- The values in the array can be any data type (8-bit, 16-bit…, signed/unsigned etc)

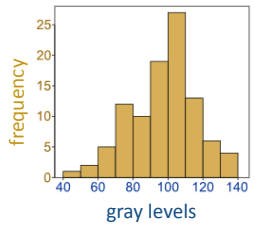
Note

- Here we work with **grayscale 2D images**…
- …but in medical imaging they can *have more dimensions and channels*!

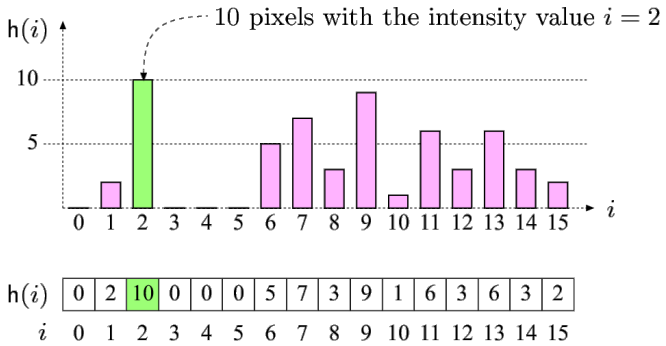
Histogram of an image (1/2)

- A **histogram** is a function $h(i)$ that gives the *frequency of each intensity i* that occur in an image
  - Given an image $I : \Omega \to [0 \dots K-1]$, its histogram is the function:

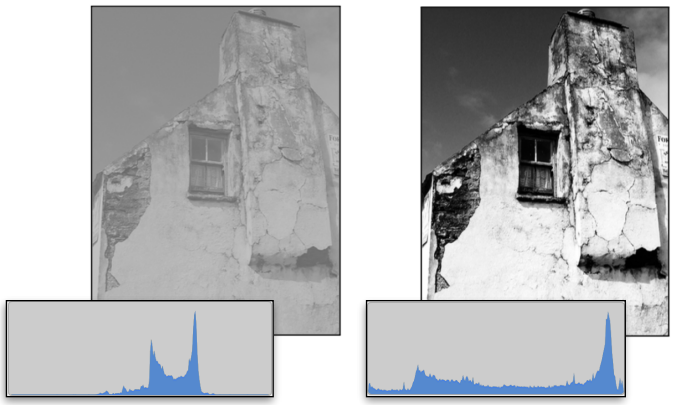
$$h(i) = \text{card}\{\ (u,v) \mid I(u,v) = i\ \}$$



- In other words
  - $h(i)$ = number of pixels with intensity i



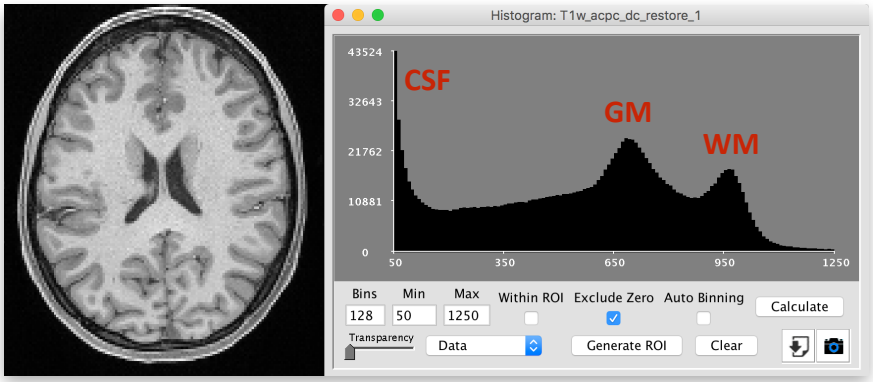
- Notes
  - *Low-contrast* image ➜ histogram is *narrow*
  - *High-contrast* image ➜ histogram is *spread out*
  - In general, *image processing* alters the histogram



Histogram of an image (2/2)

Probabilistic interpretation

- **Question**:
  - if we pick a pixel at random, what is the probability that the intensity of this voxel is equal to i?
- **Answer**:
  - $\mathrm{P}(\ I(u,v)=i\ ) =$ # pixels with value i / # pixels of the image
  - $\mathrm{P}(\ I(u,v)=i\ ) = h(i) \,/\, MN$



Generalization: a *binned histogram* gives the frequency of image intensities that fall into *small intervals* (or bins)

- Given an image $I : \Omega \to [0 \ldots K-1]$, its binned histogram is the function:

$$h(i) = \mathrm{card}\,\{\ (u,v) \mid a_i \le I(u,v) < a_{i+1}\ \}$$

where $0 < a_0 < a_1 < \ldots < K$

- Typically, we choose equally spaced bins

Point operations

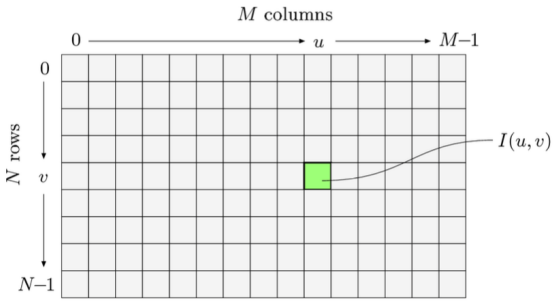
Definition

- A **point operation** on an image is an algorithm that *changes each pixel value* according to some function

$$J(u,v) \mapsto f\Big[I(u,v)\Big]$$

  - The function f depends only on the pixel value
  - It is *independent of the spatial location* (u, v)
  - The range of f determines the output datatype, e.g. uint16 or float32

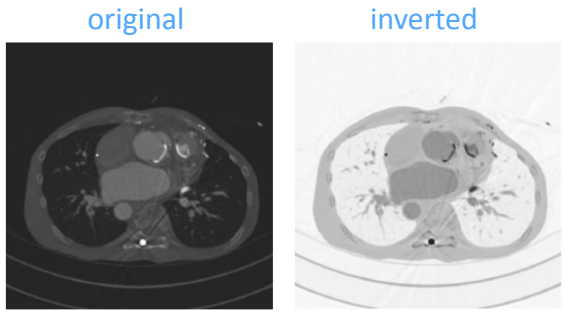
- Pseudocode
  - **Input**: image $I(u, v)$ defined on $[0 \ldots M\text{-}1] \times [0 \ldots N\text{-}1]$
  - **Output**: new image $J(u, v)$
  - *for* $v = 0 \ldots N\text{-}1$
    for $u = 0 \ldots M\text{-}1$
    set $J(u, v) = f[\, I(u, v)\,]$

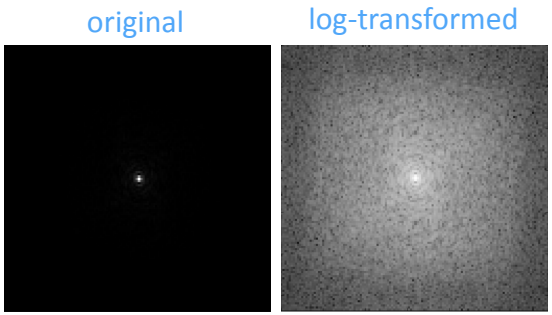


Basic examples

- Any function ca be used, e.g. $f(x)=x^2$, but *not all are useful*

- Most common
  - Addition and multiplication
    - $f(p) = p + k$
    - $f(p) = p \cdot x$
  - Inverse
    - $f(p) = L - p$
  - Gamma correction
    - $f(p) = p^{\gamma}$
  - Log transform
    - $f(p) = \log(1+p)$





- Notes
  - What happens to their *histograms*?
  - Beware of *output data type* (overflow)

Useful functions

Clamping

- *Limit intensities* to a given interval $[a, b]$

$$f(p) = \begin{cases} a & \text{if } p < a \\ p & \text{if } a \leq p \leq b \\ b & \text{if } p > b \end{cases}$$

Windowing

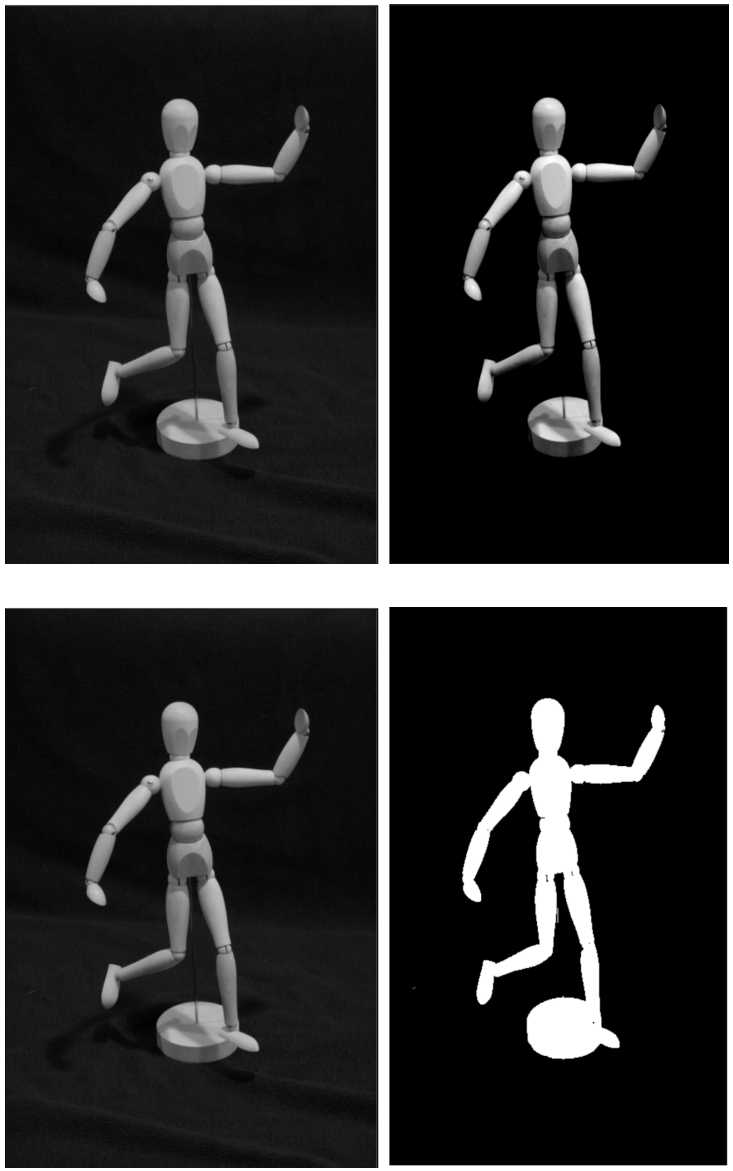
- *Clamping* followed by *intensity stretching* to fill the full possible range $[0, M]$

$$f(p) = \begin{cases} 0 & \text{if } p < a \\ M \times \frac{p-a}{b-a} & \text{if } a \leq p \leq b \\ M & \text{if } p > b \end{cases}$$

Threshold

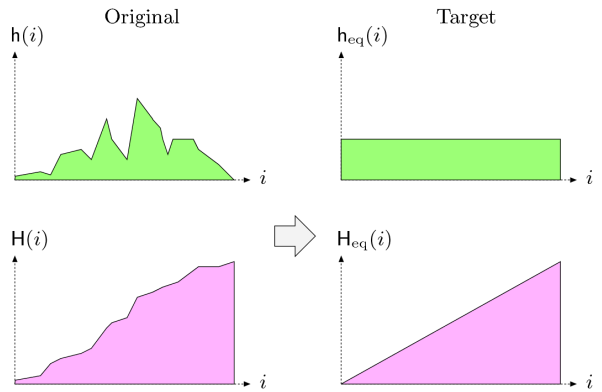
- Also called *image binarization*

$$f(p) = \begin{cases} 0 & \text{if } p \leq a \\ 1 & \text{if } p > a \end{cases}$$



Automatic contrast adjustment (1/2)

- **Histogram equalization** aims at improving the contrast by *rescaling the histogram*:
  - The histogram of the new image is *spread out*
  - The *intensities range* over all possible gray levels



- Based on the *cumulative distribution function* of $h(i)$ (also called **cumulative histogram**)

$$H(i) = \sum_{j=0}^{i} h(j)$$

  - $H(i)$ = number of pixels that have *intensity less than or equal* to i

- **Note:** ideally, the resulting image has a *flat histogram*
  - Histogram cannot be made exactly flat (peaks cannot be increased/decreased by point operations)
  - This point operation makes the histogram *as close as possible to flat*

Automatic contrast adjustment (2/2)

Procedure

(1) Compute the *histogram* $h(i)$ of the image

(2) *Normalize* $h(i)$ s.t. its range is $[0, 1]$ (i.e. divide by the total number of pixels)

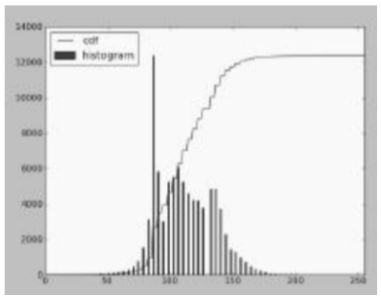
(3) Calculate the *cumulative histogram* $H(i)$

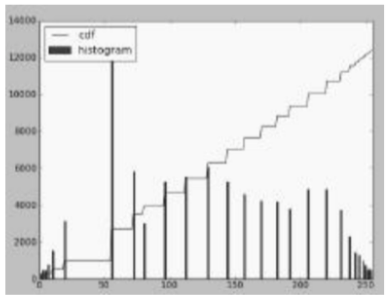
(4) Apply the following *point operation function* to every pixel p:

$$f(p) = \text{round}\left(\frac{H(p) - H_{\min}}{1 - H_{\min}} \cdot (L-1)\right)$$

where L is the number of gray levels of the image (i.e. [0, L-1] range)

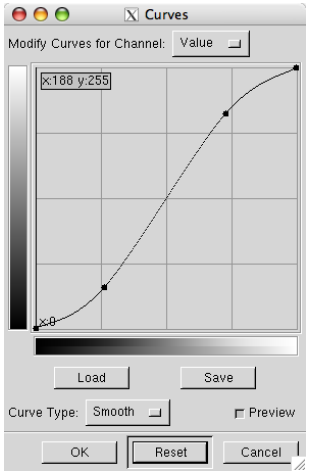
Example





“Curves” operation

- Most *software packages* allow us to manually alter histograms by adjusting the contrast by a **continuous curve**



- **Continuous** point-operation functions
  - Let’s assume for simplicity:
    - That the image has *continuous pixel type* (floating point)
    - The *intensity range* is in the interval $[0, 1]$, i.e. $I : \Omega \rightarrow [0, 1]$
  - A *continuous point-operation* is any function

$$f : [0, 1] \rightarrow [0, 1]$$

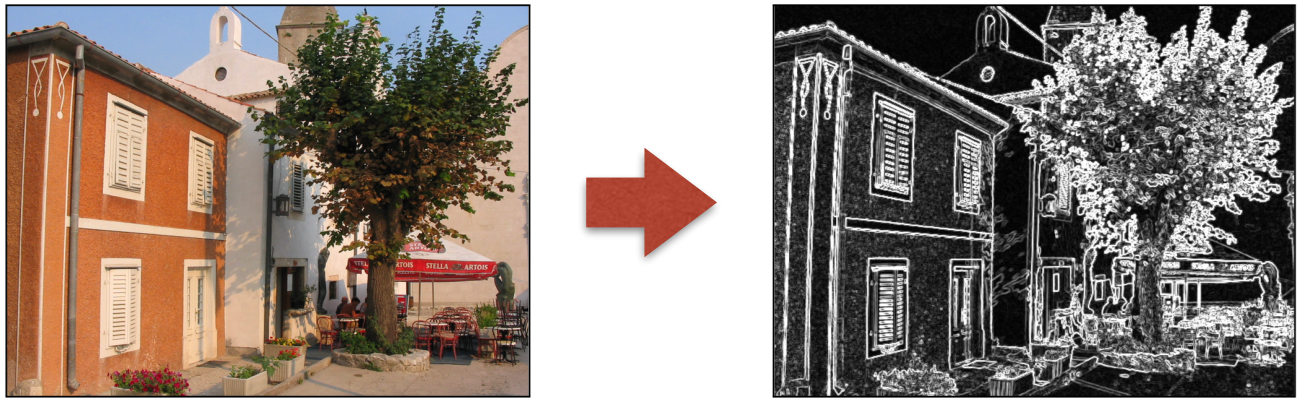
- The contrast is changed as…
  - slope = 1 ➜ no contrast change
  - slope < 1 ➜ contrast is decreased
  - slope > 1 ➜ contrast is increased

Spatial filters

Limitations of point operations

- Limitations
  - They don't know *where they are* in an image
  - They don't know anything about their *neighbors*

- Most image features (e.g. edges) involve a **spatial neighborhood** of pixels

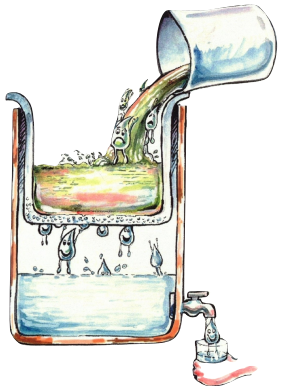


Requires derivatives (i.e. spatial information)

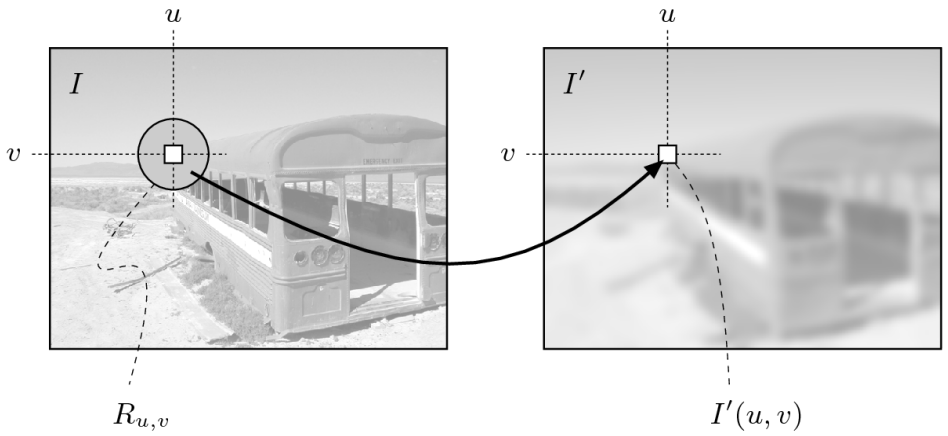
- If we want to enhance these features
  we need to **go beyond point operations**

Filtering concept (1/2)

- Analogy: as a *water filter* removes impurities, *image processing filters* **remove undesired features** from an image (e.g. noise)

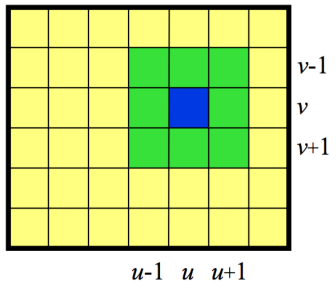


- **Spatial filtering** is an image operation where a pixel $I(u, v)$ is changed by a function of the *pixels in a neighborhood* of (u, v)



- Example
  - Consider **computing the mean** in a 3 × 3 neighborhood

$$\frac{1}{9}\sum_{i=-1}^{1}\sum_{j=-1}^{1} I(u+i, v+j)$$



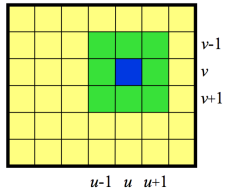
Filtering concept (2/2)

General filter equation

$$I'(u,v) = \sum_{(i,j)\in R_H} I(u+i, v+j) \cdot H(i,j)$$

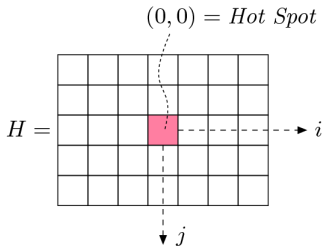
This operation is known as **correlation** of I and H ($I \otimes H$)

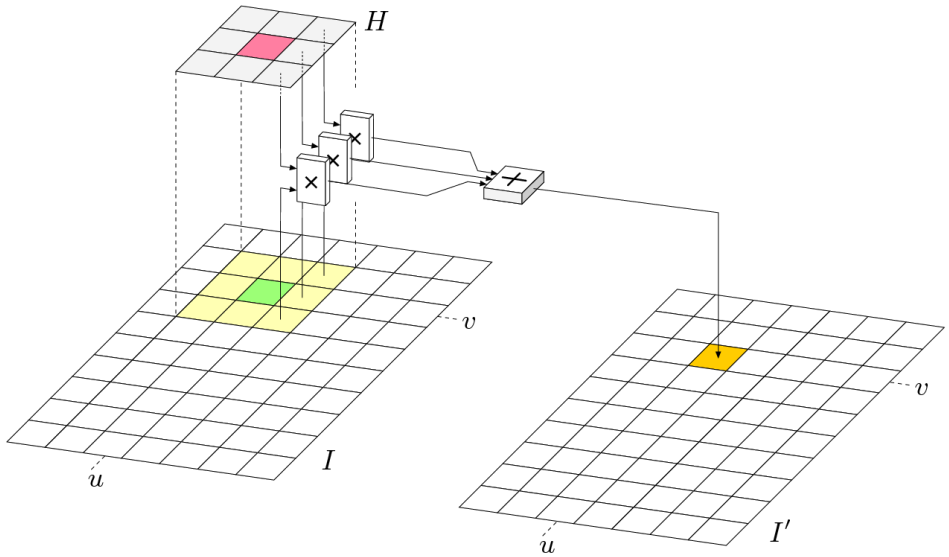
- $I(u, v)$ and $I'(u, v)$ are, respectively, the input and output images
- $H(i, j) : R_H \rightarrow [0, K-1]$ is the filter (a.k.a. *kernel* or *mask*)
- R_H describes the *spatial neighborhood* of a voxel



In other words

- A **small 2D matrix** moves across the image affecting *only one pixel at a time*
- The *coefficients in H* **determine the effect** on the output image





What do these filters do? (1/2)

- Filter 1 ➜ **nothing**!

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

- Filter 2 ➜ **blurring** (mean)

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

What do these filters do? (2/2)

- Filter 3 ➜ **sharpening** (identity minus mean filter)

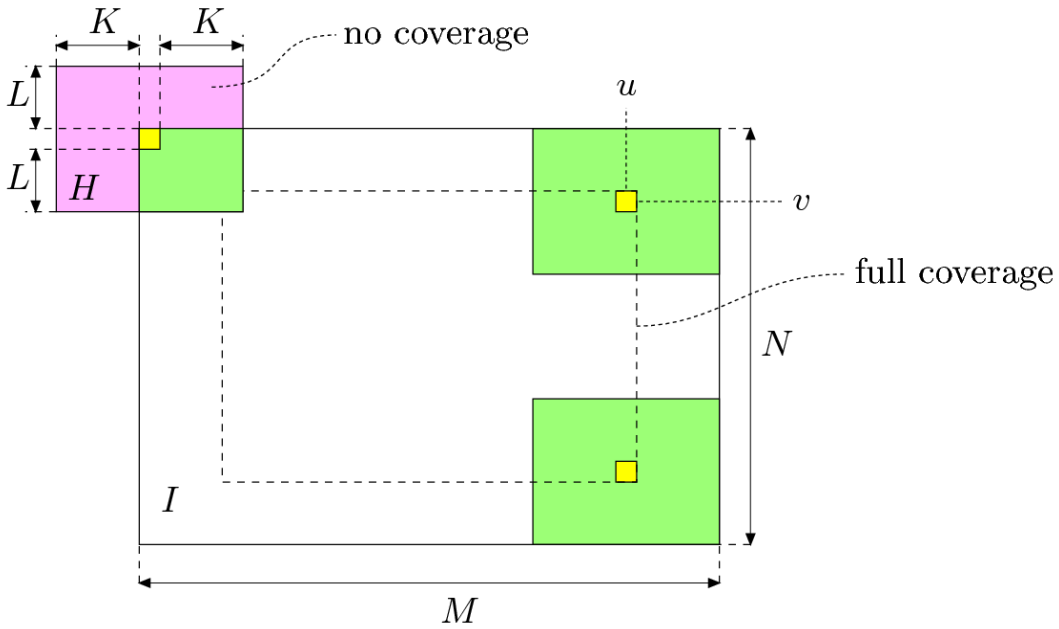
$$\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 17 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Filter 4 ➜ **shift left by one pixel**

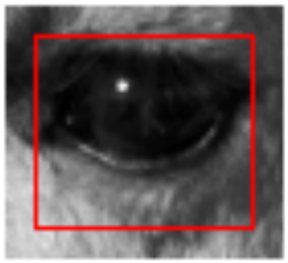
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Notes: what to do at the boundary?

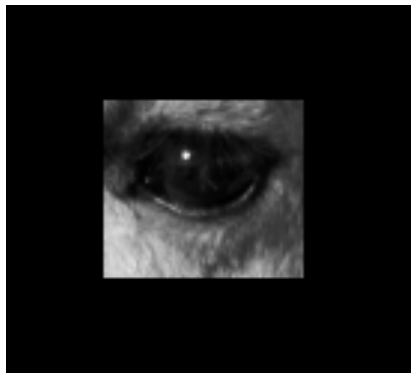
- The **neighborhood is not available** at the boundary



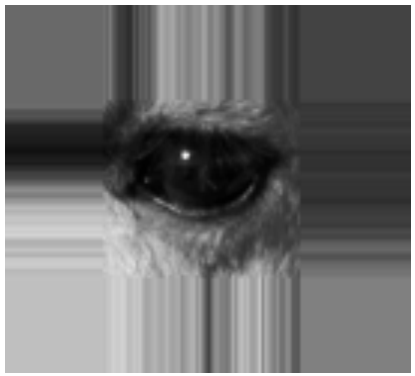
- Some options



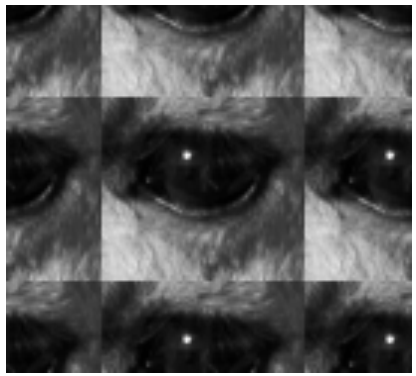
crop



pad



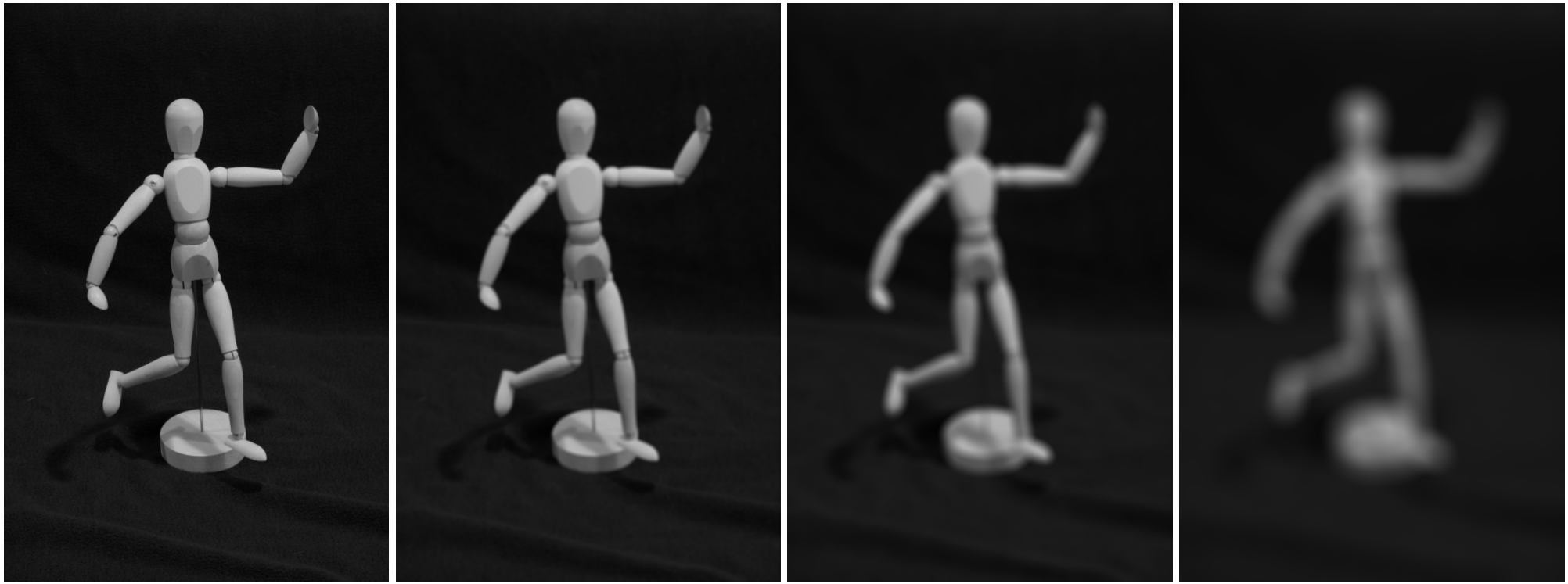
extend



wrap

Notes: effect of filter size

- **Example**: mean filters
  (similar effects with any filter)



original | 7 × 7 | 15 × 15 | 41 × 41

Notes: filter normalization

- Multiplying all entries in H by a constant would cause the **image** to be **multiplied** by that constant

$$I'(u,v) = \sum_{i,j} I(u+i, v+j) \cdot (cH(i,j))$$
$$= c \sum_{i,j} I(u+i, v+j) \cdot H(i,j)$$

- Hence, to **keep the overall brightness constant** we need H to sum up to one

- Notice how all *previous filter examples* indeed sum up to one!

0	0	0
0	1	0
0	0	0

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$\frac{1}{9}$

-1	-1	-1
-1	17	-1
-1	-1	-1

0	0	0
0	0	1
0	0	0

Edge detection (1/4)

What is an edge?

- An *abrupt transition* in intensity between two regions

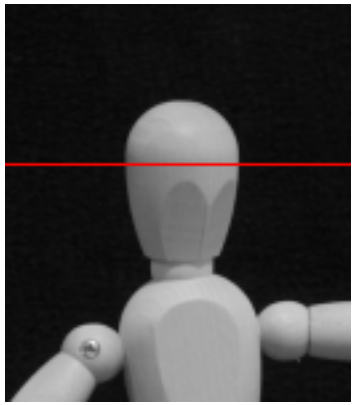
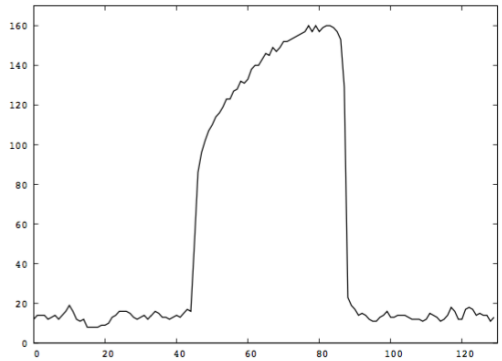
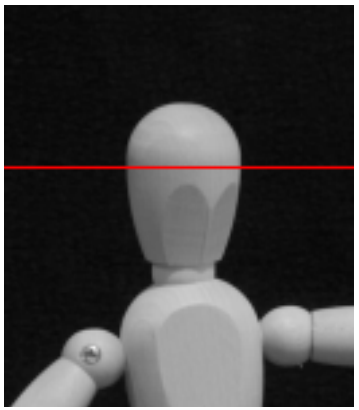


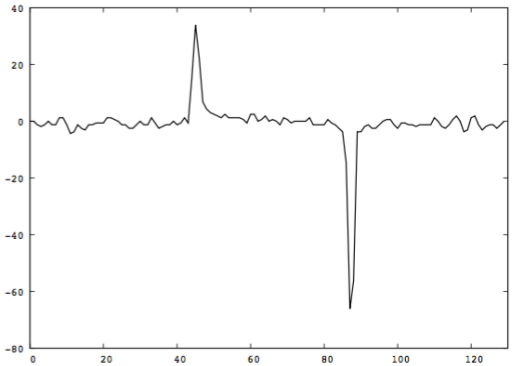
image intensity along red line



- Image *derivatives* are high (or low) at edges



corresponding ***x*-derivative**



How do we compute **image derivatives**?

Edge detection (2/4)

- **Discrete functions**: *left* and *right derivatives* don't match
  - *Forward* differences

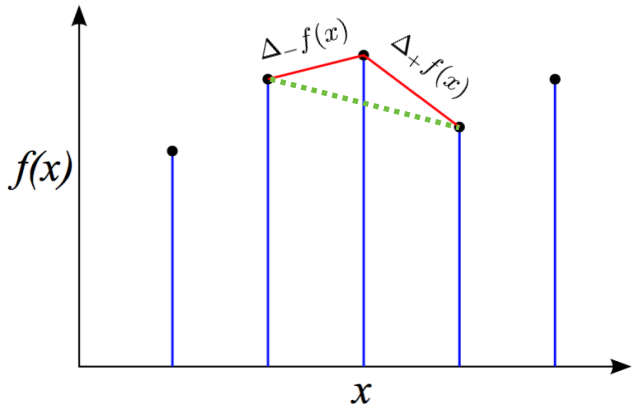
$$\Delta_+ f(x) = f(x+1) - f(x)$$

(right slope)

  - *Backward* differences

$$\Delta_- f(x) = f(x) - f(x-1)$$

(left slope)



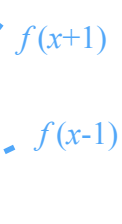
- **Solution**: take the average
  - *Central* differences

$$\Delta f(x) = \frac{1}{2}\left(\Delta_+ f(x) + \Delta_- f(x)\right) = \frac{1}{2}\left(f(x+1) - f(x-1)\right)$$

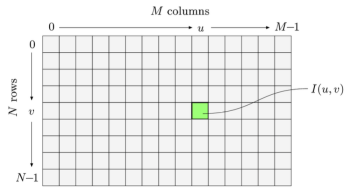
(average slope)

- Central differences as **spatial filter**

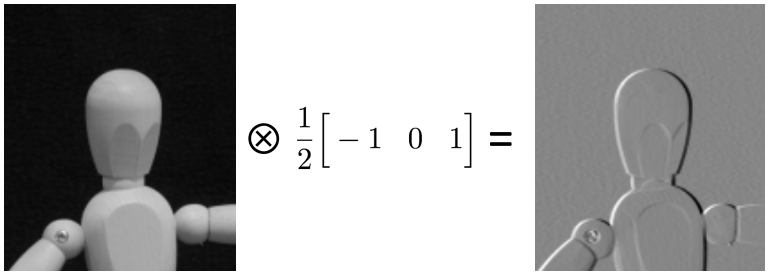
$$H = \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$



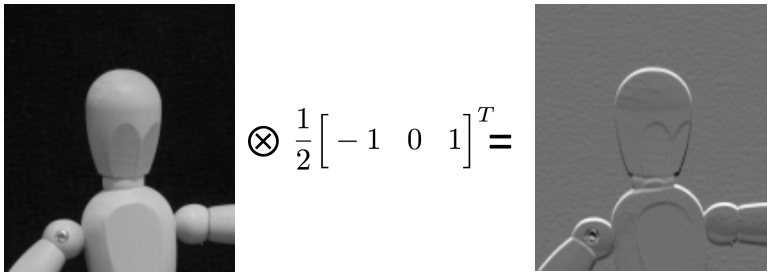
Derivatives of images



- Images have two dimensions: $I(u, v)$
- We can take derivatives with respect to u or v



u-derivative using central differences



v-derivative using central differences

Combining u and v derivatives

- The **discrete gradient** of $I(u, v)$ is the 2D vector

$$\nabla I(u,v) = \begin{bmatrix} \nabla_u I(u,v) \\ \nabla_v I(u,v) \end{bmatrix}$$

- The **gradient magnitude** is

$$||\nabla I(u,v)|| = \sqrt{(\nabla_u I(u,v))^2 + (\nabla_v I(u,v))^2}$$

gradient magnitude

Prewitt operator

- Finite differences are **sensitive to noise**
- More robust derivatives by *averaging in the neighborhood*

$$\nabla_u I(u,v) = \frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix} \qquad \nabla_v I(u,v) = \frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}^T$$

$$\nabla_u I(u,v) = \frac{1}{6}\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix} \qquad \nabla_v I(u,v) = \frac{1}{6}\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix}^T$$

- **Notice**: $H = \frac{1}{6}\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix} \cdot \frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$ (averaging · derivative)

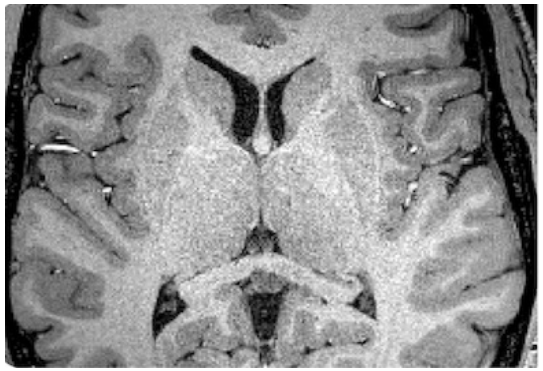
Sobel operator

- Similar to Prewitt, but averaging is *higher in middle*

$$\nabla_u I(u,v) = \frac{1}{8}\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix} \qquad \nabla_v I(u,v) = \frac{1}{8}\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}^T$$

Noise reduction (1/3)

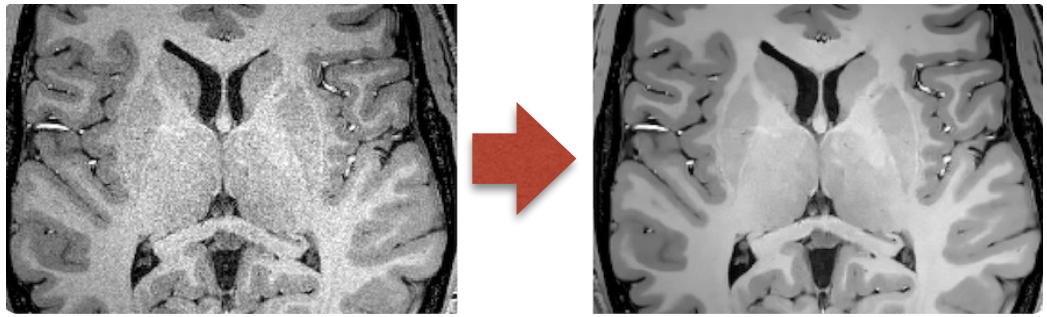
- Image pixels are the result of a **signal intensity measurement**
  - All recording devices are susceptible to *noise*
  - Noise causes *fluctuations* in actual signal intensities
  - Noise properties depend on acquisition equipment



- "Additive" noise model

$$\hat{s} = s + \epsilon$$

  - $\hat{s}$ is observed signal intensity
  - s is the true value
  - ϵ is the noise value



- Noise reduction is the process of **removing noise from a signal** (i.e. all pixels in the image)

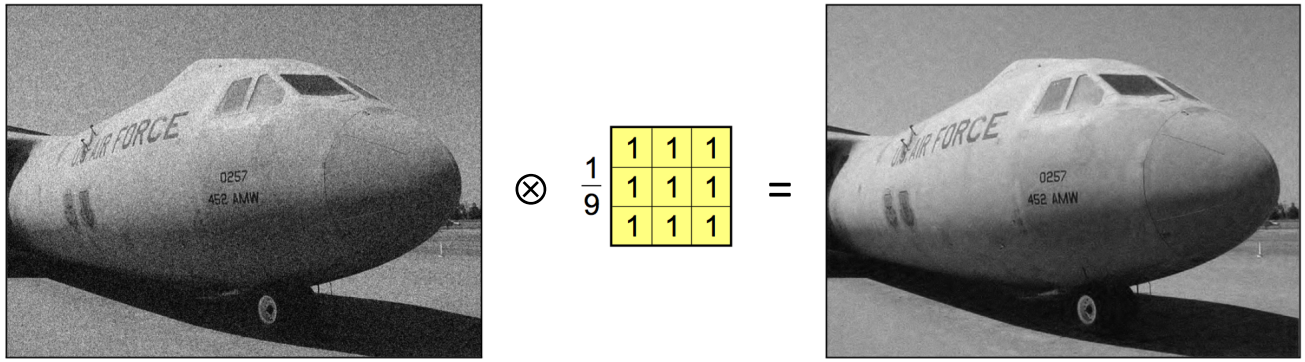
Noise reduction (2/3)

- **Typical noise**: Additive White Gaussian Noise (AWGN)
  - *Many random processes* that occur in nature follow this model
  - **Note**: white noise has *zero mean*

- **IDEA**: if we can average several pixels in a neighborhood with the same signal, the noise will "average out"

$$E[\hat{s}] = E[s + \epsilon] = E[s] + E[\epsilon] = s$$

- Any filter that *averages in a neighborhood* will reduce noise

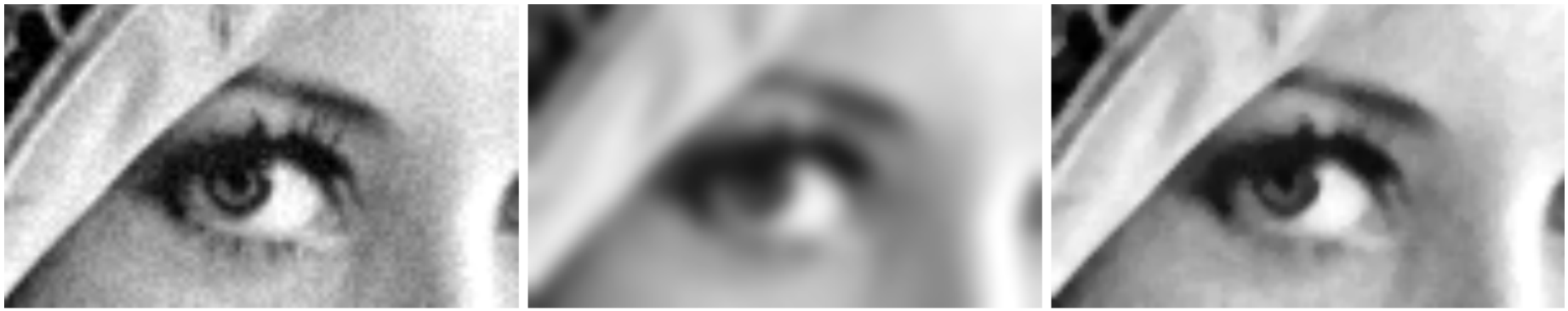


  - *Fourier theory* tells us why some are better than others

Noise reduction (3/3)

- **Note 1**: particular noise sources require specific algorithms
  - **Example**: *Salt-and-Pepper* noise
  - *Averaging does not work* well because the outlier is included in the mean (biasing the result)
  - Taking the *median* does a good job

- **Note 2**: image features (e.g. edges) are *not constant* and they *also get blurred* by averaging filters



measured image | denoised 1 | denoised 2

- **To summarize**: more advanced filters are needed

Fourier filtering

Convolution (1/3)

- **Convolution** of an *image* I by a *kernel* H is given by

$$I'(u,v) = \sum_{(i,j)\in R_H} I(u-i, v-j) \cdot H(i,j)$$

Convolution of I and H ($I * H$)

  - $I(u, v)$ and $I'(u, v)$ are, respectively, the *input and output images*
  - $H(i, j) : R_H \rightarrow [0, K-1]$ is the *filter*
  - R_H describes the *spatial neighborhood* of a voxel

- Notes
  - Recall the definition of **correlation**:

$$I'(u,v) = \sum_{(i,j)\in R_H} I(u+i, v+j) \cdot H(i,j)$$

Correlation of I and H ($I \otimes H$)

  - Similar to correlation, but with negative signs on the I indices
  - Equivalent to **vertical and horizontal flipping** of H

$$I'(u,v) = \sum_{(-i,-j)\in R_H} I(u+i, v+j) \cdot H(-i,-j)$$

Convolution (2/3)

Why then another definition?

- **Correlation** was easier to explain for introducing spatial filters
- **Convolution** has more useful properties

Basic properties of convolution

- *Linear* $(a \cdot I) * H = a \cdot (I * H)$ $(I_1 + I_2) * H = (I_1 * H) + (I_2 * H)$
- *Commutative* $I * H = H * I$
- *Associative* $(I * H_1) * H_2 = I * (H_1 * H_2)$

Convolution theorem

- Let f, g be two functions with *convolution* $f*g$ and *Fourier transforms* $\mathrm{F}\{f\}$ and $\mathrm{F}\{g\}$. Then:

$$\mathrm{F}\{f * g\} = \mathrm{F}\{f\} \cdot \mathrm{F}\{g\} \qquad \mathrm{F}\{f \cdot g\} = \mathrm{F}\{f\} * \mathrm{F}\{g\}$$

- *In other words*, convolution in a domain equals point-wise multiplication in the other

Convolution (3/3)

Filtering is **simpler in Fourier space!**

- In spatial domain, $I*H$ is performed by sliding H on the image I (*very slow*)
- In frequency domain, the operation is *replaced by a simple multiplication*

$$\mathcal{F}\{ I * H \} = \mathcal{F}\{ I \} \cdot \mathcal{F}\{ H \} \quad \Rightarrow \quad I * H = \mathcal{F}^{-1}\{ \mathcal{F}\{ I \} \cdot \mathcal{F}\{ H \} \}$$

- **NB:** *FFT* is efficient, *multiplication* is efficient. **Filtering in Fourier space is faster!**

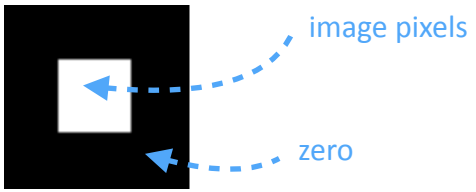
To make the theory work out, we **need a mathematical trick**

- Let's define our *image* and *kernel* **domains to be infinite**: $\Omega = \mathbb{Z} \times \mathbb{Z}$
- Now convolution is an **infinite sum**:

$$I'(u,v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) \cdot H(i,j)$$

New definition of $I*H$

- We can still imagine that the image is **defined on a finite domain** $[0, M] \times [0, N]$ but is **set to zero outside this range**



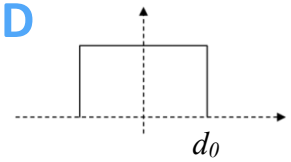
Filtering in frequency domain (1/4)

Example 1: lowpass filters

- *Low frequencies* of the image (from the Fourier transform) are kept
- *High frequencies* are blocked (containing all fine details)
- Used to **smooth the image** or reduce noise

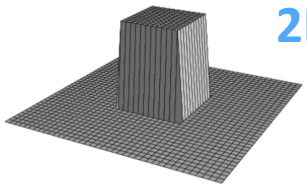
Ideal filter

1D



$$H(x) = \begin{cases} 1, & \text{if } |x| \leq d_0 \\ 0, & \text{if otherwise} \end{cases}$$

2D



0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Do you remember the **mean filter**?

Example

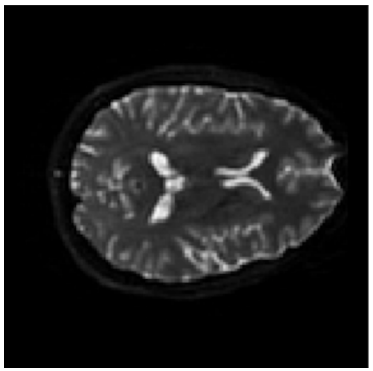
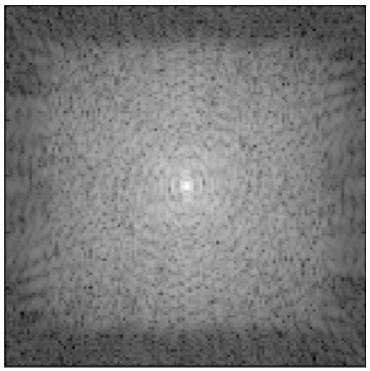
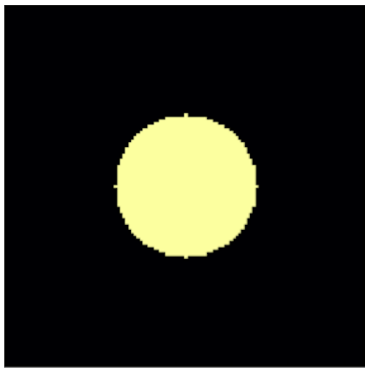


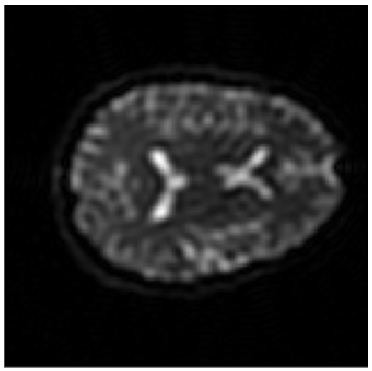
image I



FFT{ I }



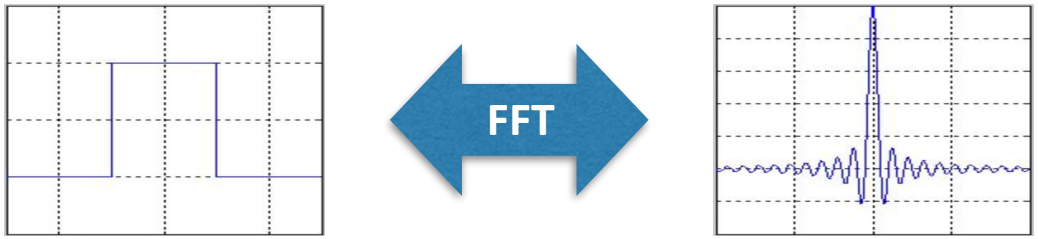
filter H



$I' = I * H$

Filtering in frequency domain (2/4)

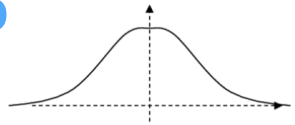
- Note: the mean filter gives **blocky blurring/ringing**. Why?



box sinc

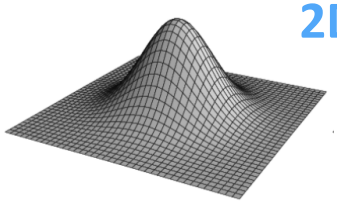
- Gaussian filter

1D



$$H(x) = e^{-\frac{x^2}{2\sigma^2}}$$

2D



0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

- Example

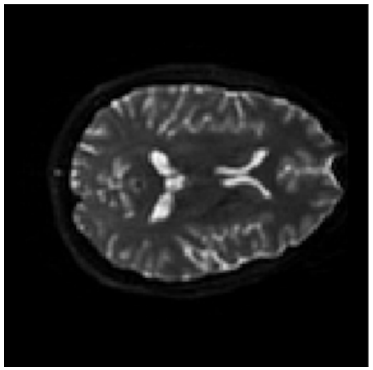
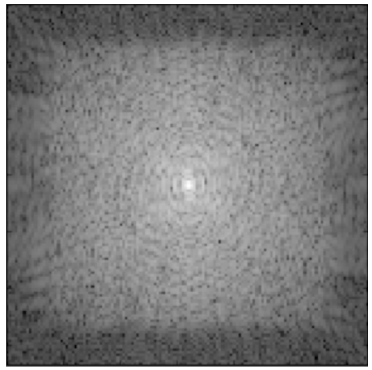
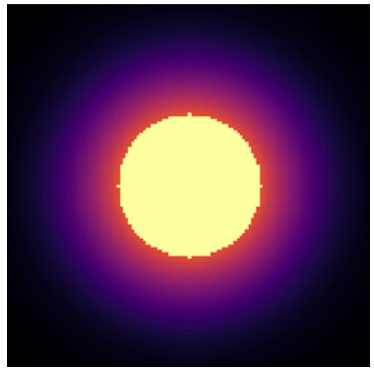


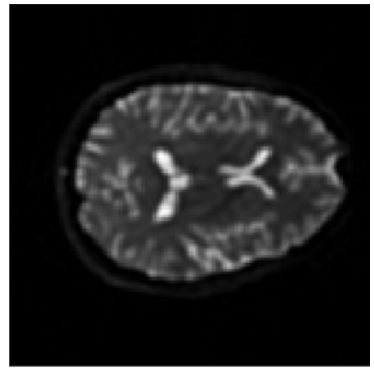
image I



FFT{ I }

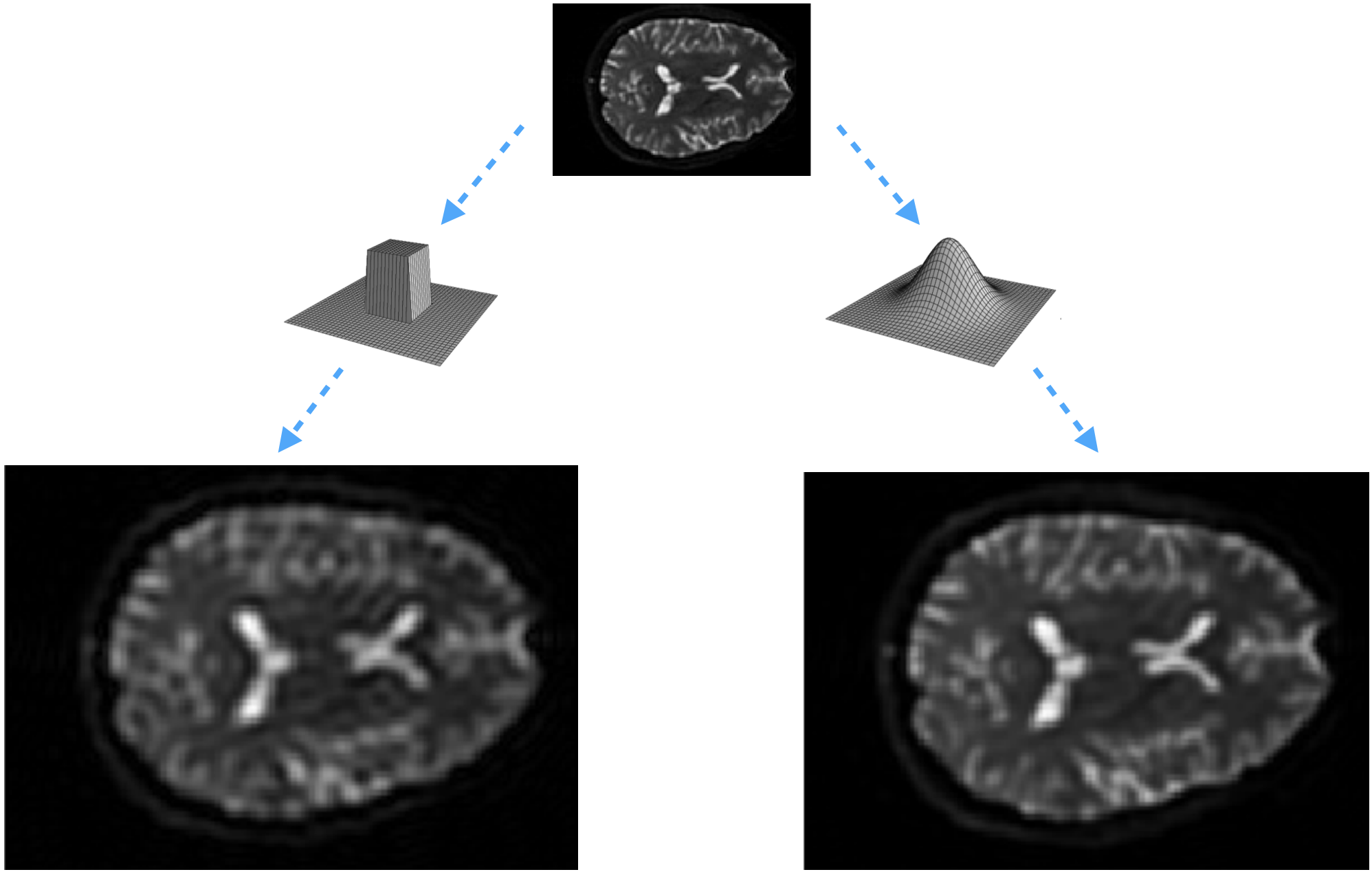


filter H



$I' = I * H$

Filtering in frequency domain (3/4)



Filtering in frequency domain (4/4)

Example 2: highpass filters

- *Inverse* of lowpass filters (high frequencies are kept, while low ones are blocked)
- Used to **sharpen the edges** of the image

Ideal vs **Gaussian** filters

- Same pros/cons of lowpass filters

NB: any filter can be used in frequency domain

- Usually they are *more efficient*
- The process to move to the frequency domain provides valuable insight into the nature of the image and filter