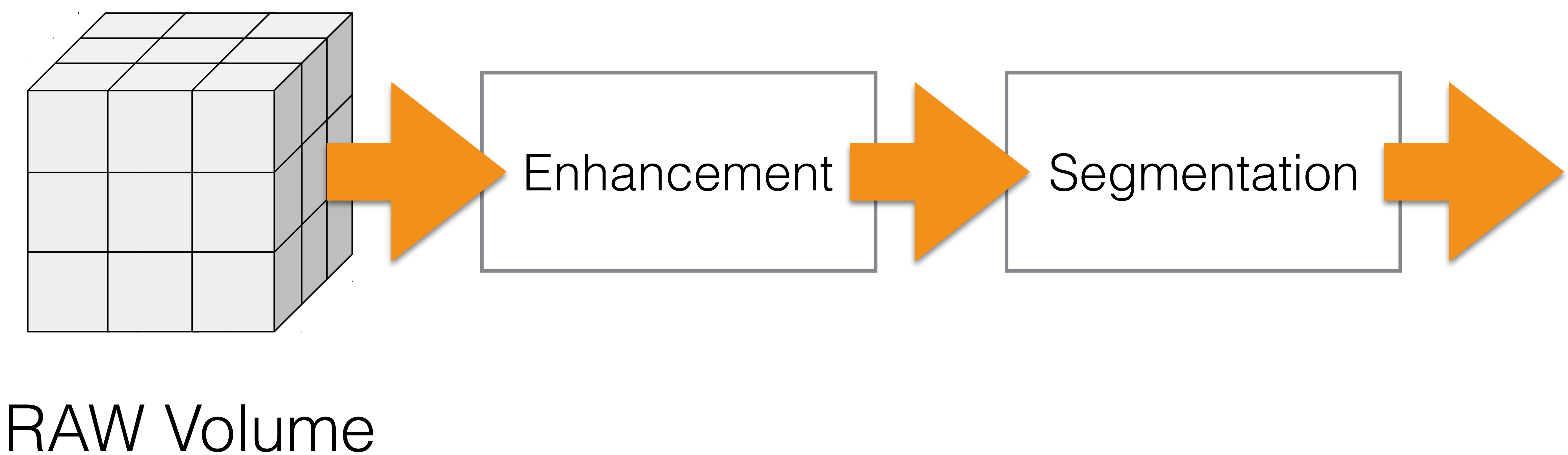
3D from Volume: Part I

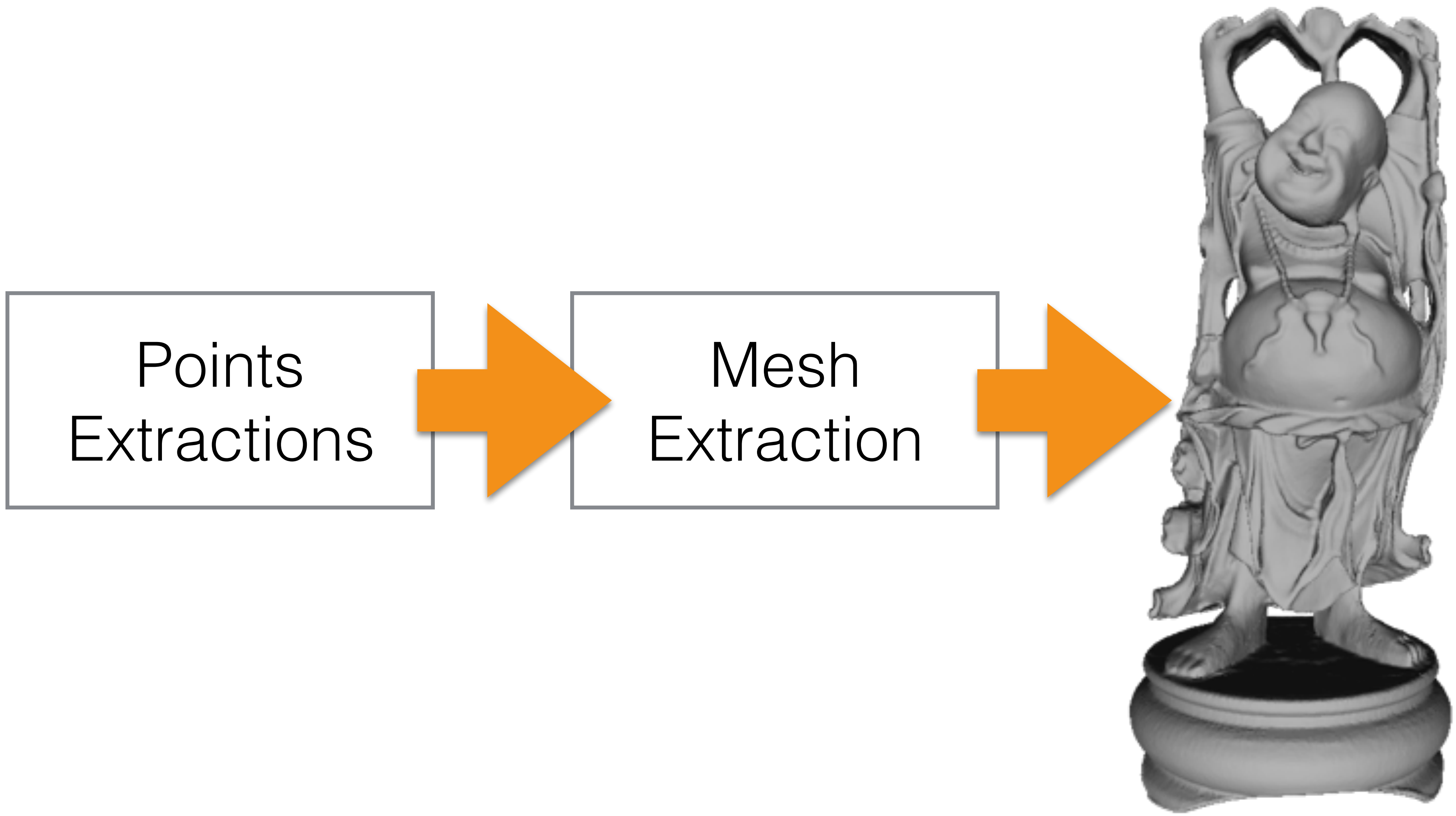
Dr. Francesco Banterle,
francesco.banterle@isti.cnr.it
banterle.com/francesco

The Main Pipeline

The Processing Pipeline

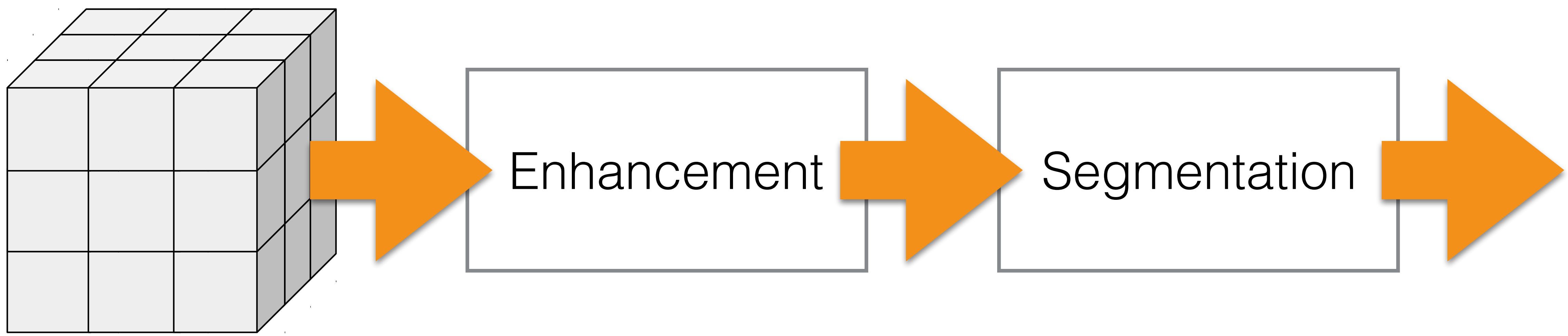


The Processing Pipeline



3D Mesh

The Processing Pipeline



RAW Volume

Image Enhancement

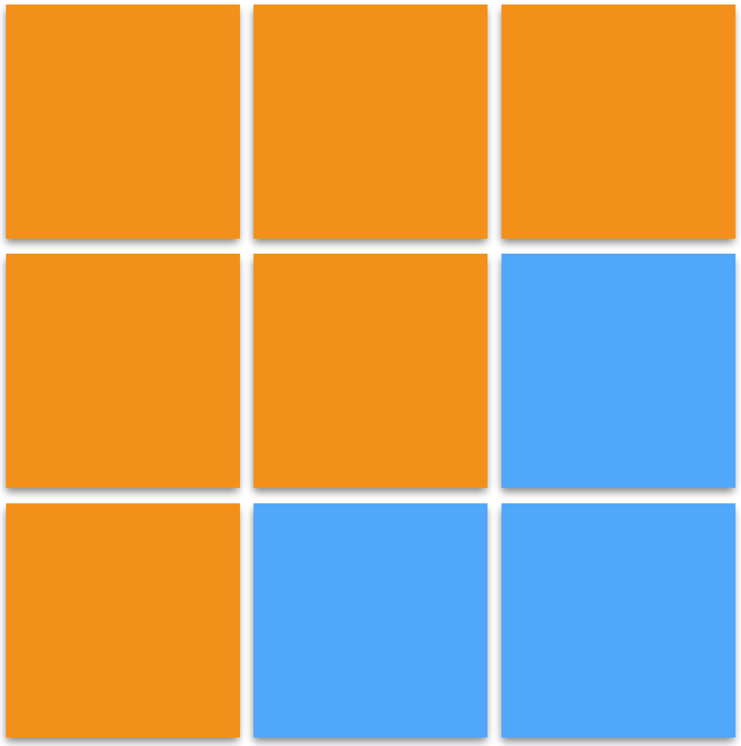
- **Step 1**: we have to improve the dynamic range of the input images in the volume; i.e., increase/decrease it.

- **Step 2**: we have to filter the image in order to elicit some features and/or to remove noise (salt-and-pepper, Gaussian noise, etc).

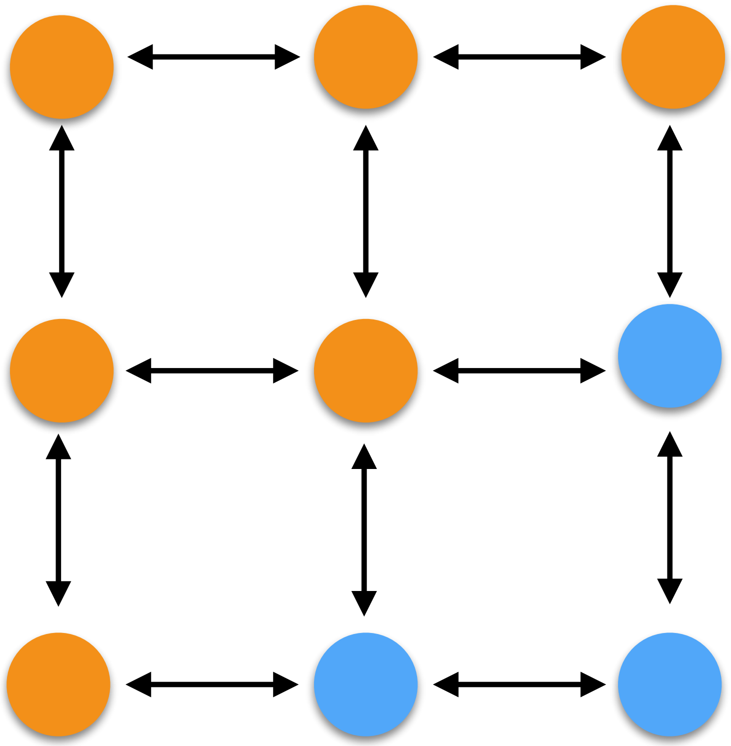
2D Images

2D Images

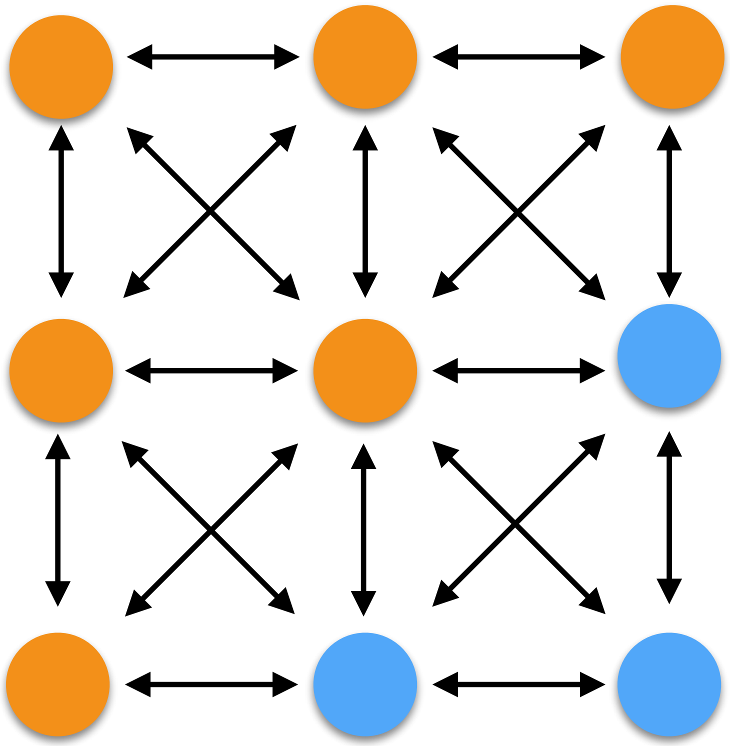
- A 2D image is a graph:



2D Image, 3x3 pixels



4-connected pixel adjacency graph



8-connected pixel adjacency graph

A Graph

- A graph is a pair G = (V,E), where:
  - V is a set of vertices. Each element of V is called a **vertex** of G.
  - E is a pairs of elements in V; e.g, (V_1; V_2), etc. Each element of E is called an **edge** of G.

Image Coordinate System

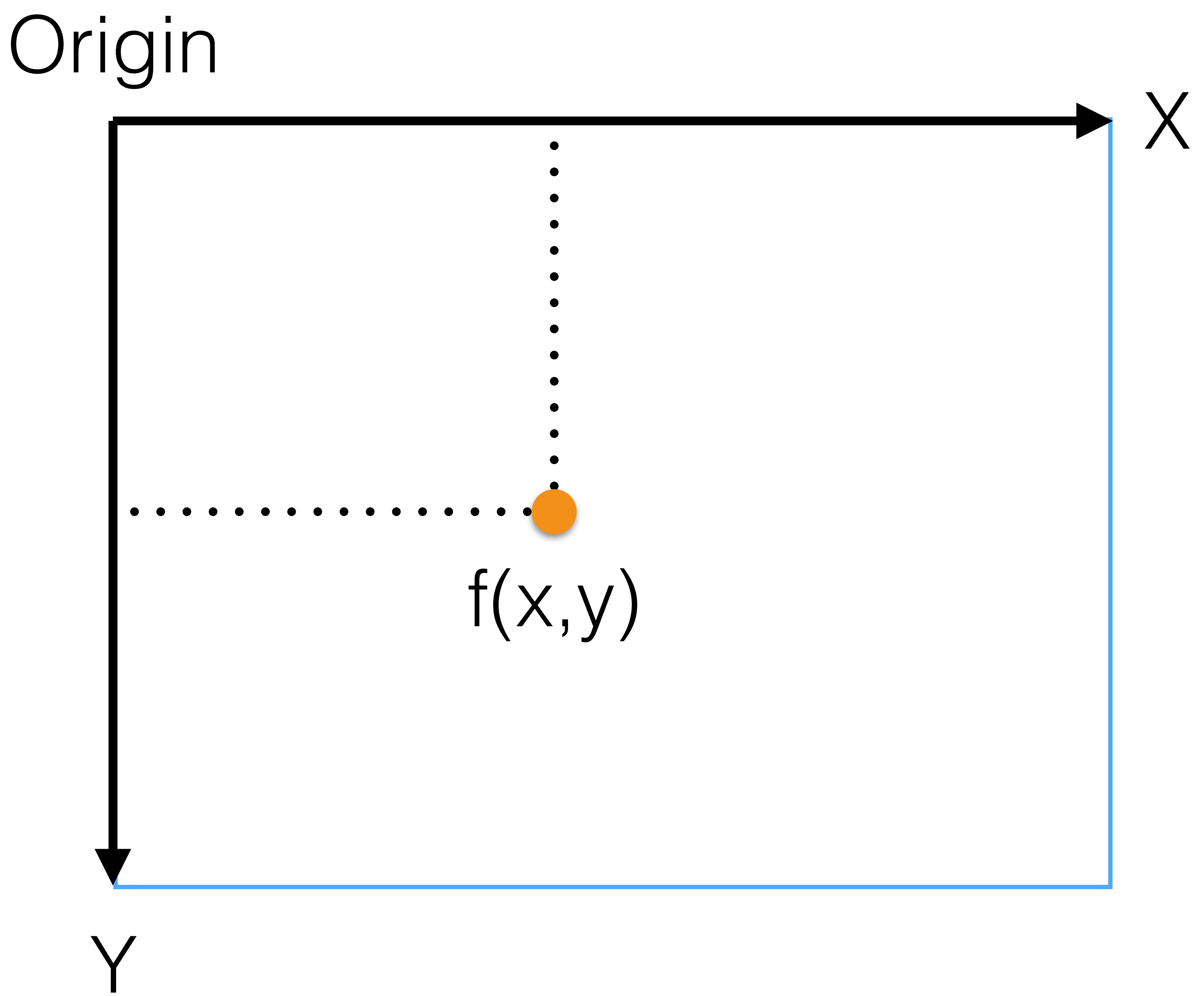


Image Coordinate System: MATLAB

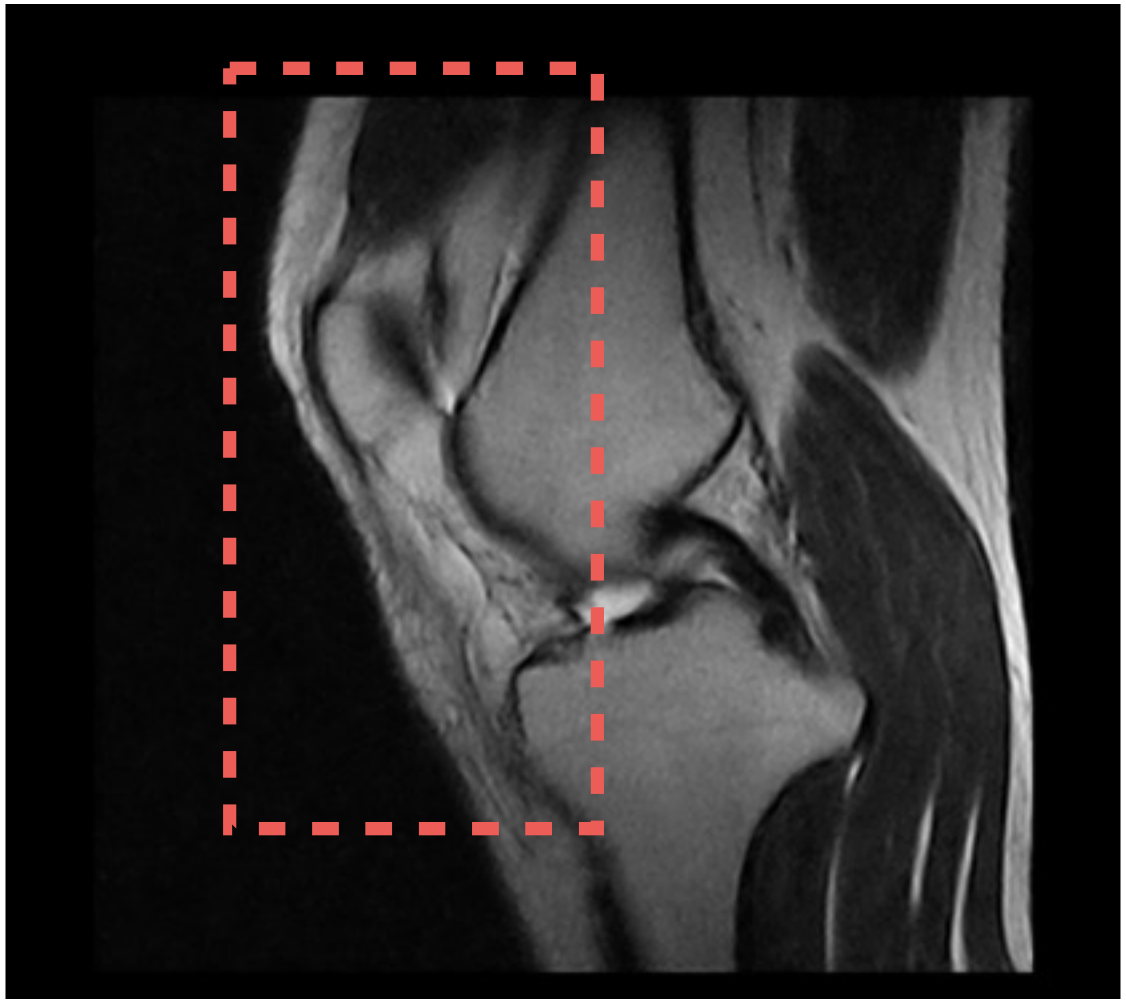
- MATLAB origin —> (1,1)
- Given an image, img, as m-n matrix to access to f:

$$f = img(y, x)$$

Region Of Interest (ROI)

- We may be interested to process not the full image/volume but an area/volume.

- This area is typically called region of interest (ROI).



Noise in Medical Imaging

- Images are not perfect: device, patient moves, etc.
- What we really see is:

$$f(x,y) \approx f'(x,y)$$
$$f(x,y) = [(f' + n_T) \otimes h](x,y) \cdot g(x,y) + n(x,y)$$

Noise in Medical Imaging

- Images are not perfect: device, patient moves, etc.
- What we really see is:

$$f(x,y) \approx f'(x,y)$$

$$f(x,y) = [(f' + \boxed{n_T}) \otimes h](x,y) \cdot g(x,y) + n(x,y)$$

Other tissues

Noise in Medical Imaging

- Images are not perfect: device, patient moves, etc.
- What we really see is:

$$f(x, y) \approx f'(x, y)$$
$$f(x, y) = [(f' + n_T) \otimes \boxed{h}](x, y) \cdot g(x, y) + n(x, y)$$

discrete spatial-temporal
process

Noise in Medical Imaging

- Images are not perfect: device, patient moves, etc.
- What we really see is:

$$f(x, y) \approx f'(x, y)$$

$$f(x, y) = [(f' + n_T) \otimes h](x, y) \cdot \boxed{g(x, y)} + n(x, y)$$

discrete spatial-temporal process

Noise in Medical Imaging

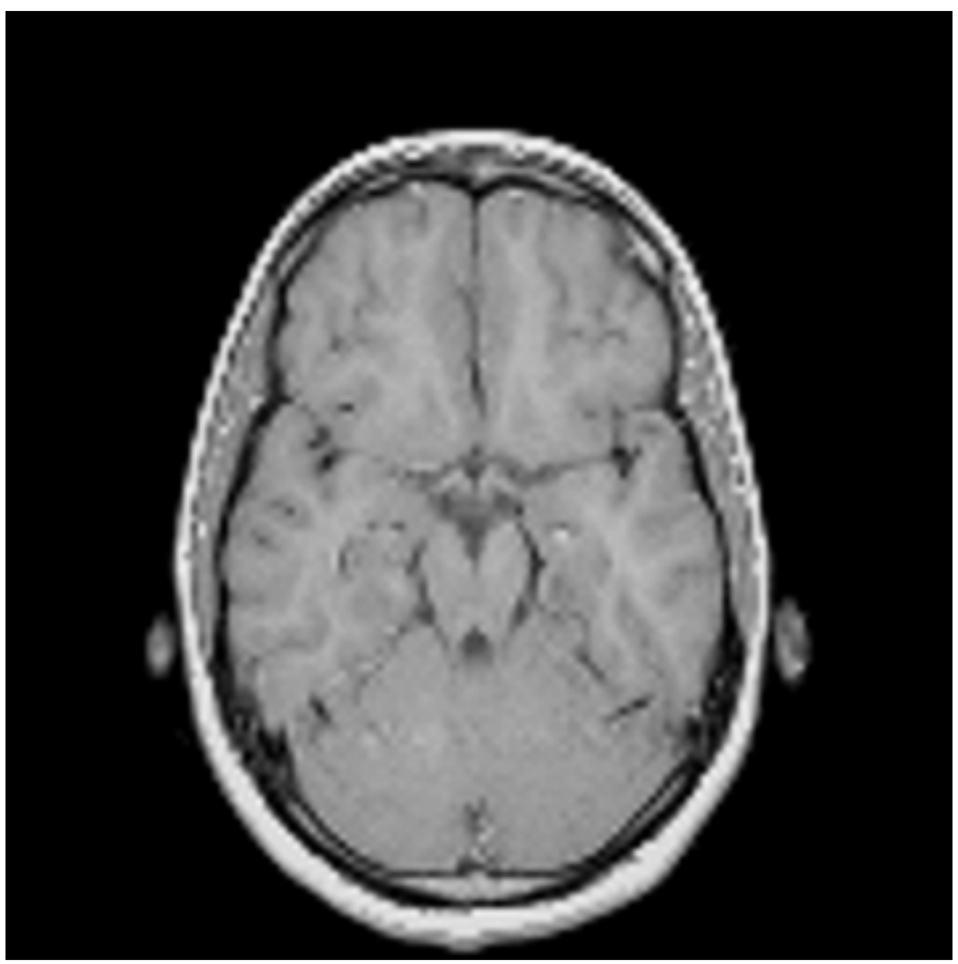
- Images are not perfect: device, patient moves, etc.
- What we really see is:

$$f(x,y) \approx f'(x,y)$$
$$f(x,y) = [(f' + n_T) \otimes h](x,y) \cdot g(x,y) + \boxed{n(x,y)}$$

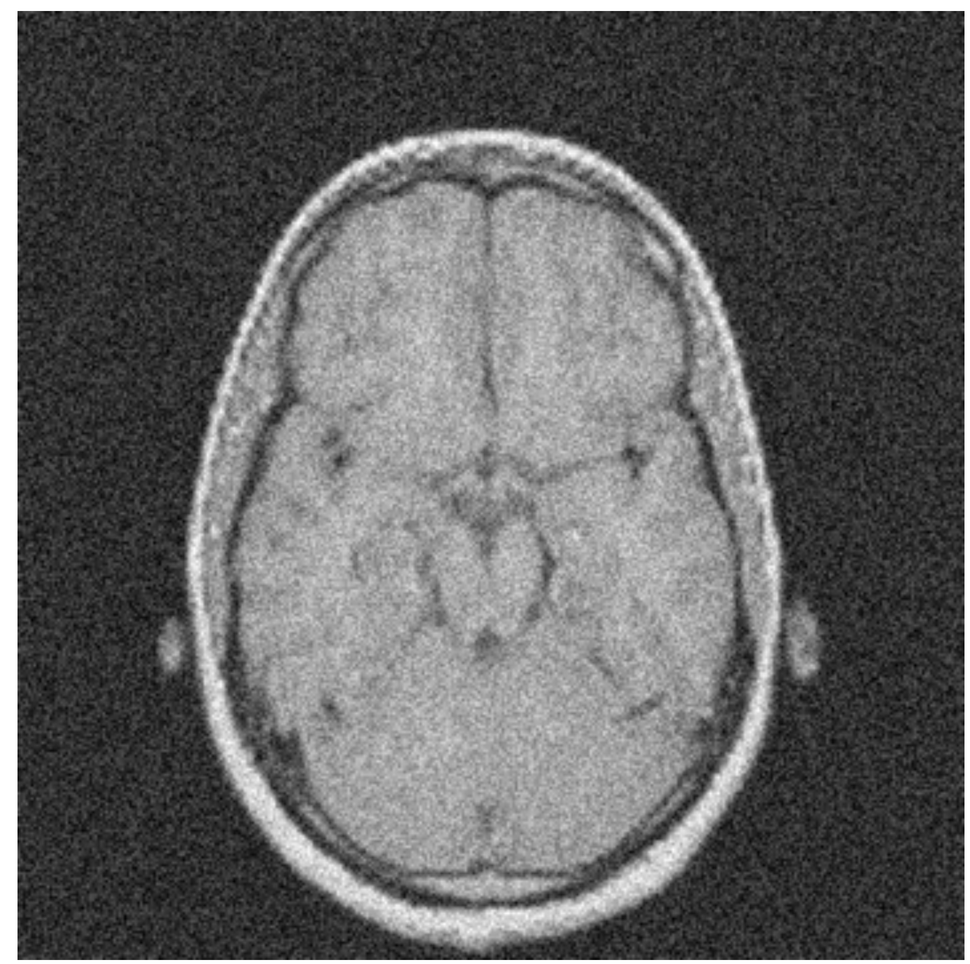
device noise

Noise Measure

- A classic: $\mathrm{SNR} = \dfrac{\mu}{\sigma}$



SNR = 0.8



SNR = 1.5

Medical File Format

DICOM

- **D**igital **I**maging and **CO**mmunications in **M**edicine
- It is a standard for producing, storing, displaying, printing, and sending, retrieving, and querying medical images
- Data: 2D images (may be compressed using JPG/JPG2000)
- Metadata: patient's personal information, date of the exam, position of the patient, etc.
- Issue: many extra fields, which are filled without consistency amongst different software/scanners

DICOM

- File extension: name_file.dcm
  - The media format does not allow files to have and extension; the folders structure gives meaning to the file!
- Standard official web-site: http://DICOM.nema.org
- MATLAB and Slicer can open them natively.

Point-wise Operators

Point-wise Operators

- An operator takes as input one or two images, and the result is another image.
- Unary operator T_1:

$$g(x,y) = T_1\Big[f(x,y)\Big]$$

- Binary operator T_2:

$$g(x,y) = T_2\Big[f(x,y); h(x,y)\Big]$$

Unary Operators: Negative

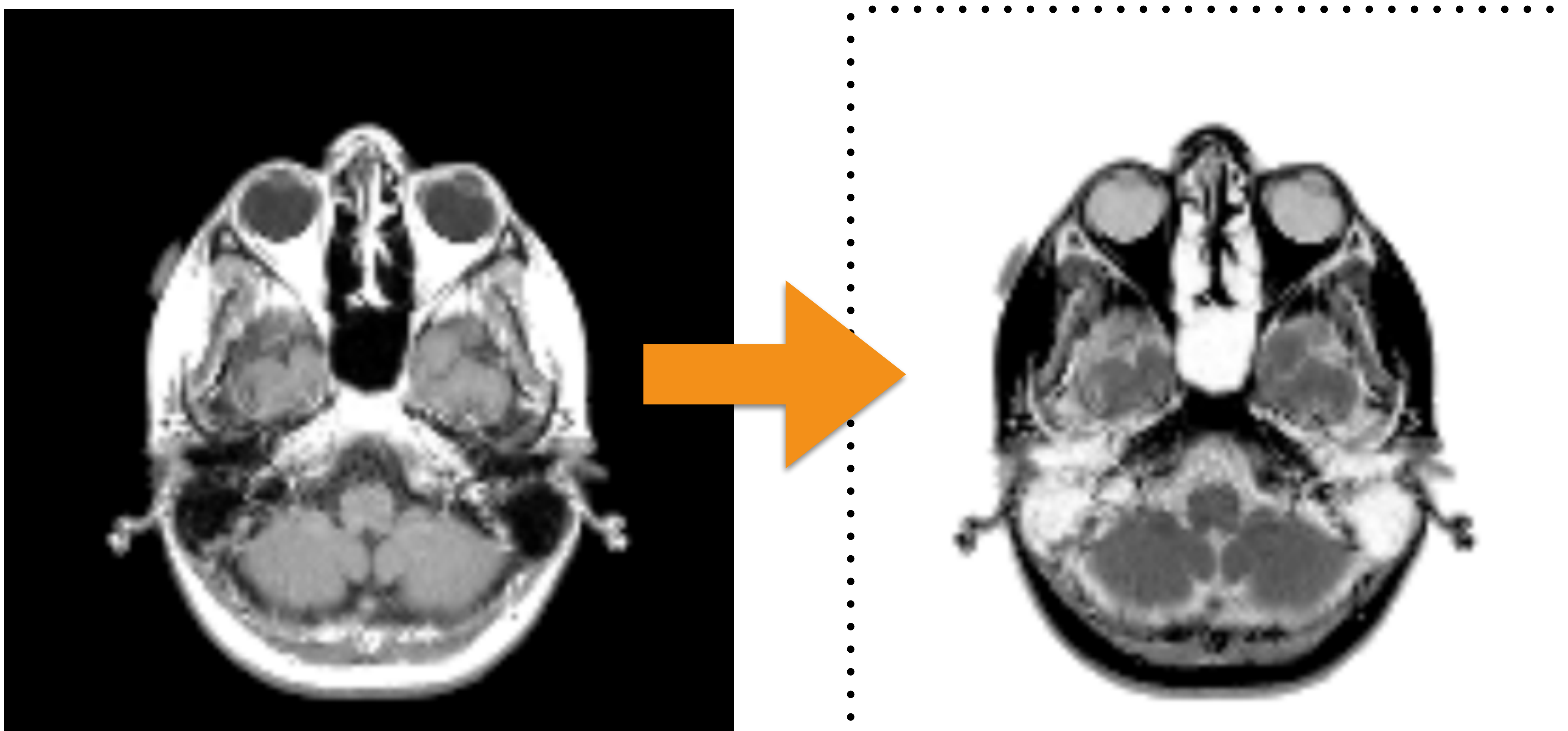
- Negative or inverter:

$$g(x, y) = \mathrm{Neg}\big[f(x, y)\big] = 1.0 - f(x, y)$$

- It is usually helpful to highlight some structures.

- Note: this operator assumes images' values are in the rage [0,1].

Negative: Example



Unary Operators: Contrast Stretching

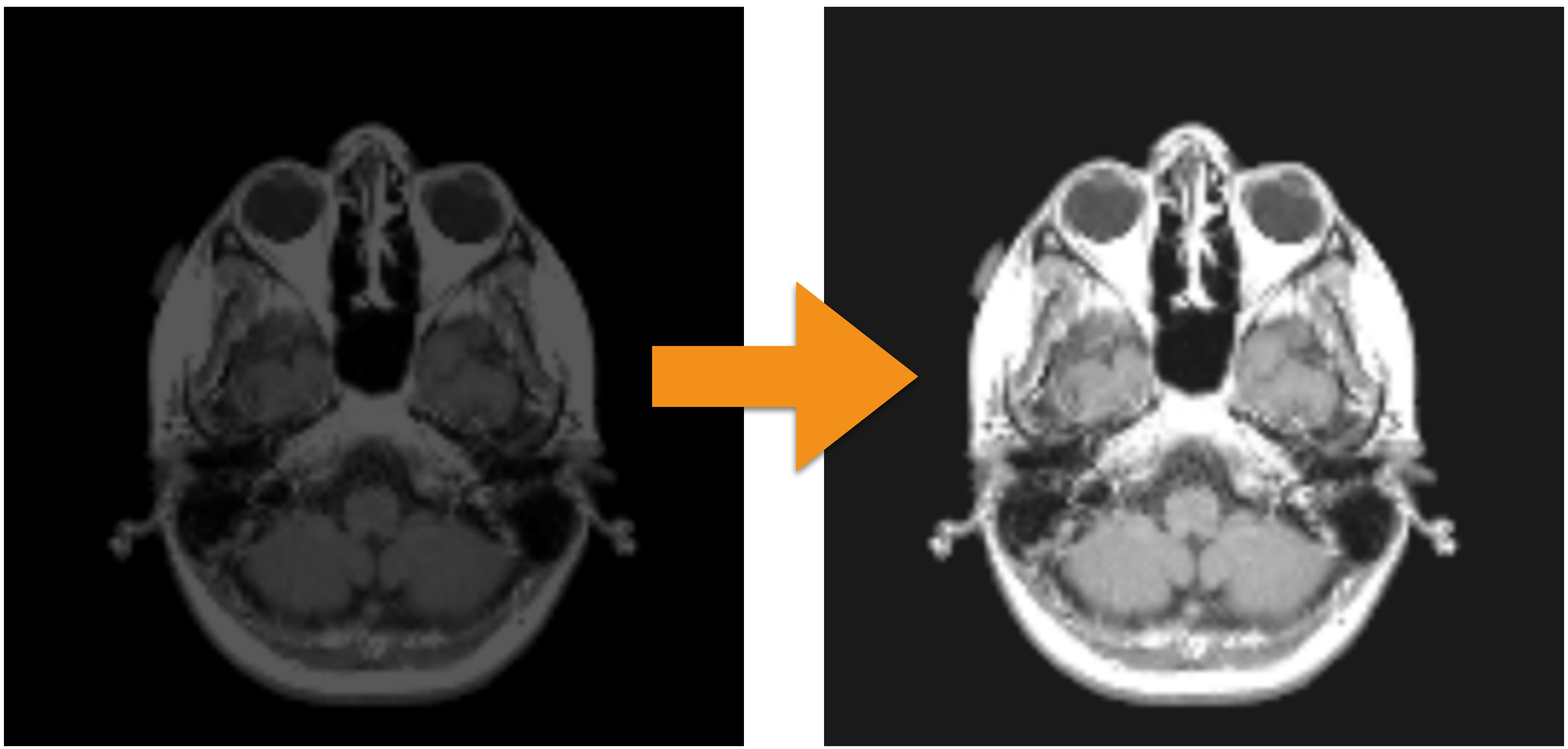
- This operator increases the dynamic range of the input image linearly:

$$g(x,y) = \mathrm{CS}\big[f(x,y); E_{\mathrm{min}}; E_{\mathrm{max}}\big] =$$

$$= (f(x,y) - \min(f))\frac{E_{\mathrm{max}} - E_{\mathrm{min}}}{\max(f) - \min(f)} + E_{\mathrm{min}}$$

- It is useful when the contrast is low.

Contrast Stretching Example



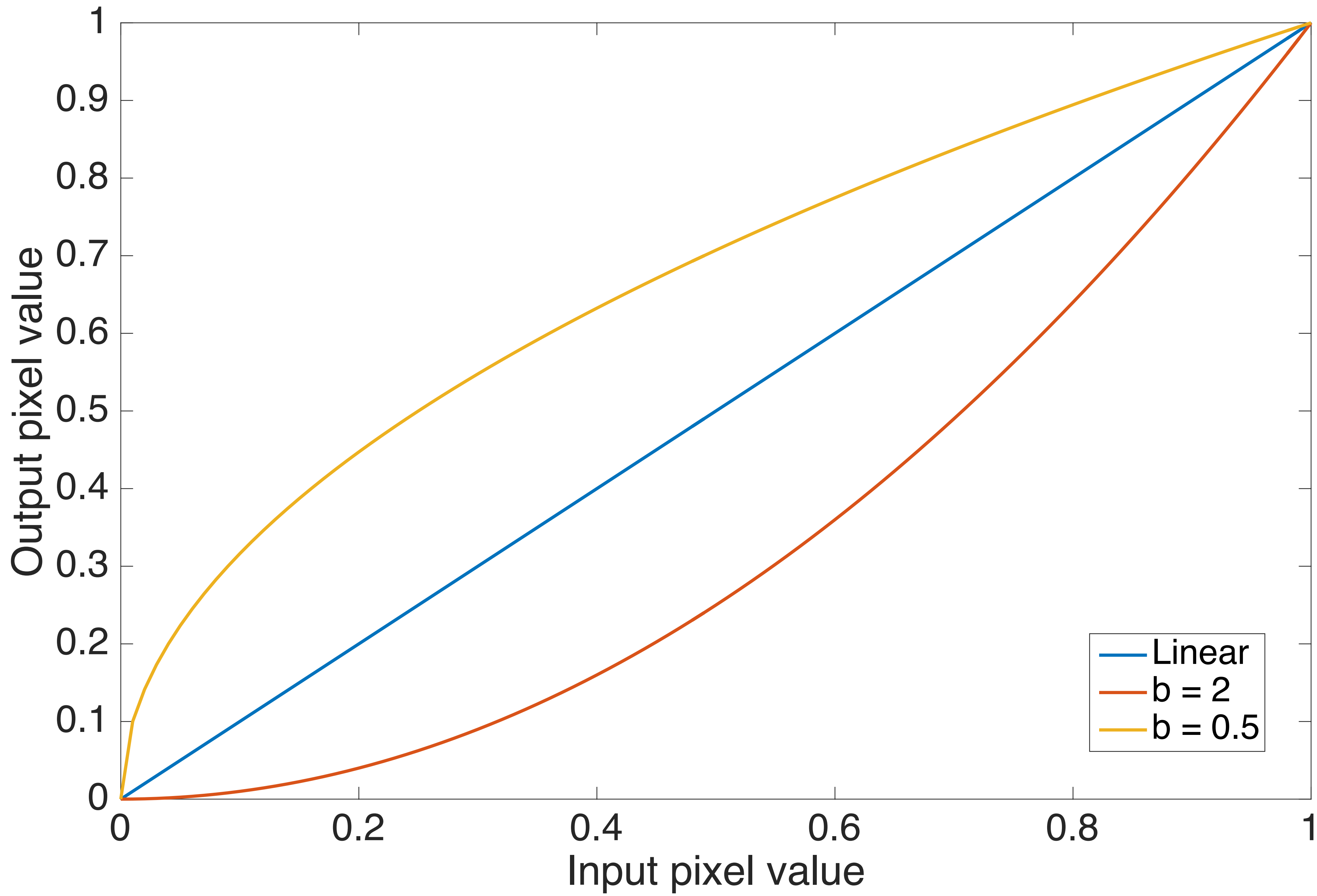
Unary Operators: Gamma

- Another method for increasing the dynamic range:

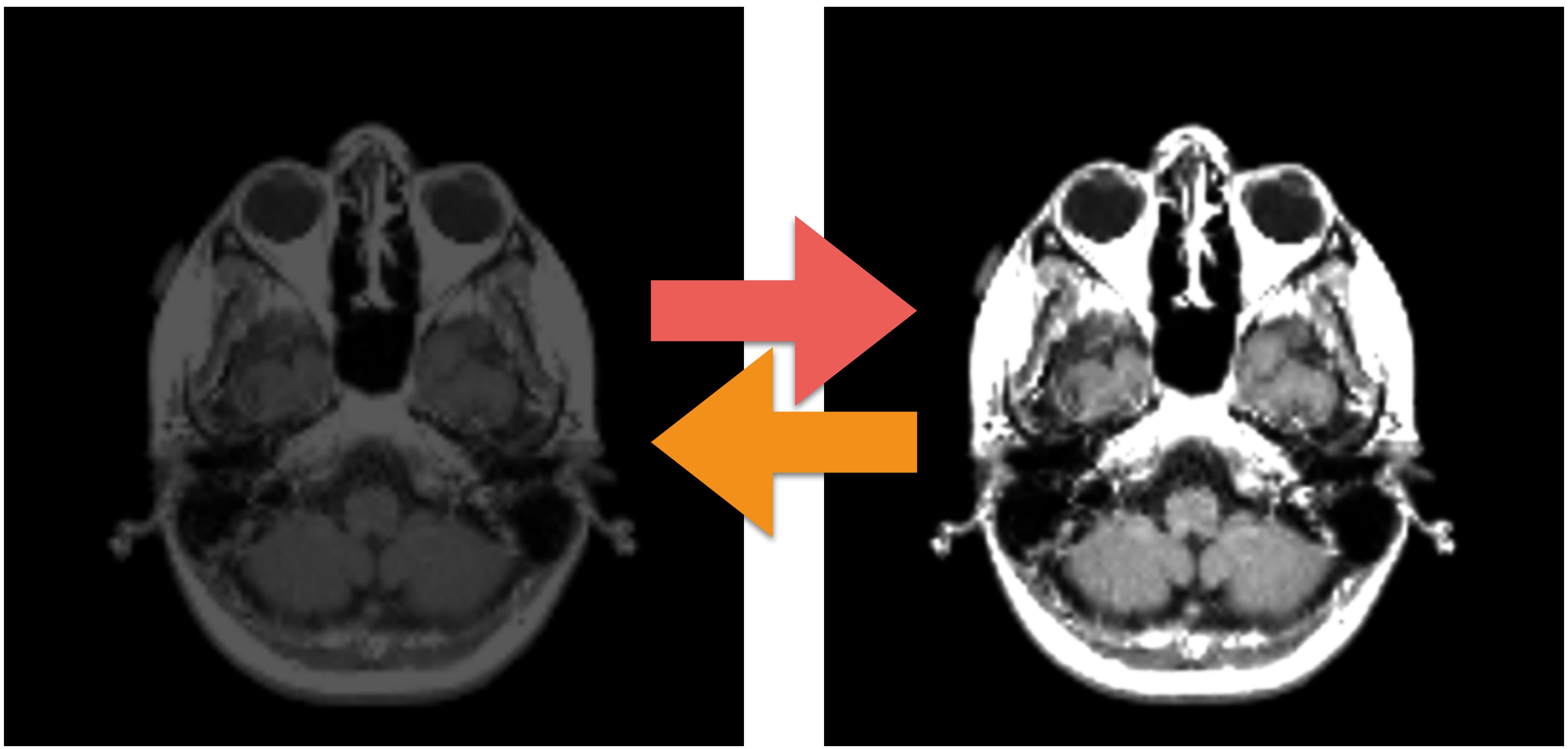
$$\begin{aligned} g(x,y) &= \mathrm{G}\big[f(x,y);k;\gamma\big] = \\ &= k \cdot f(x,y)^{\gamma} \end{aligned}$$

- It is more intuitive.

Unary Operators: Gamma



Gamma Example



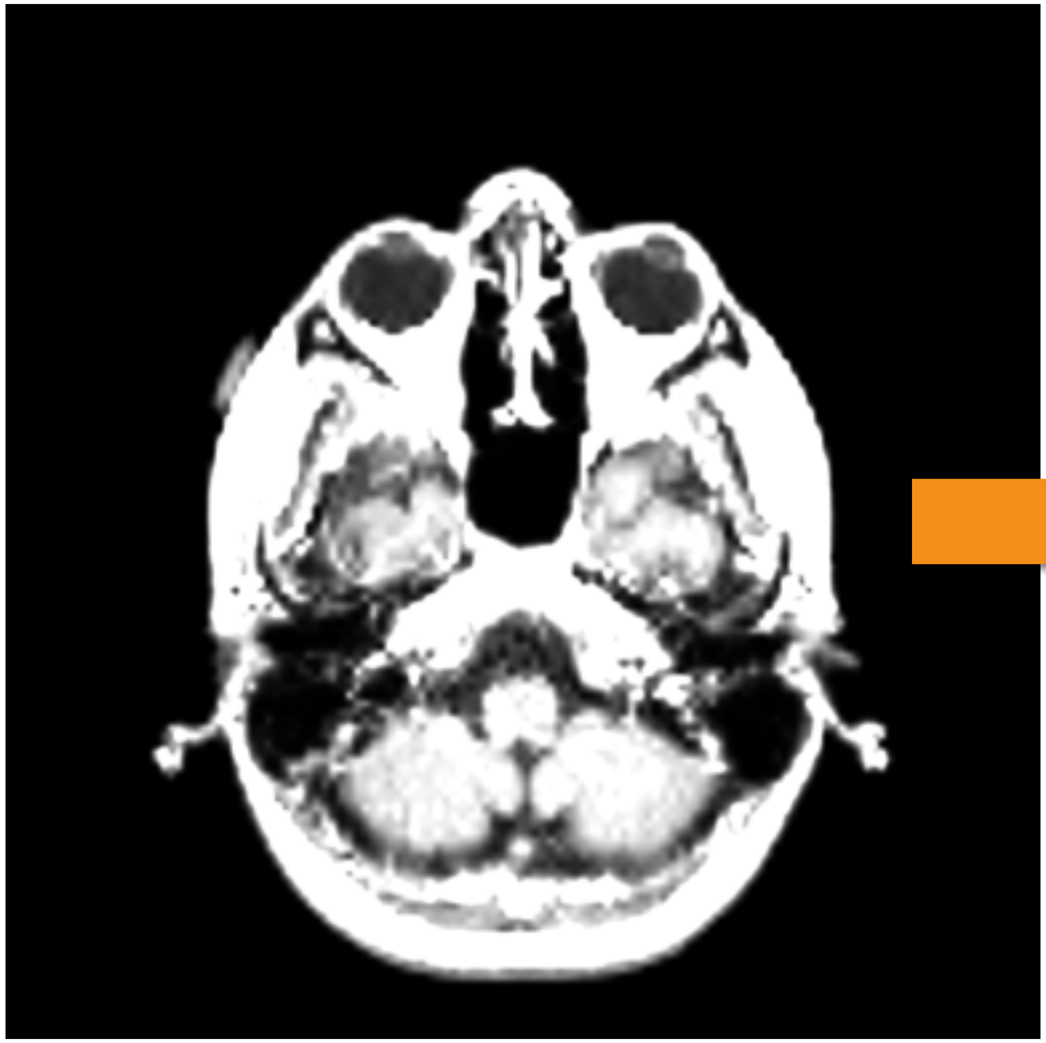
Unary Operators: Logarithmic Operator

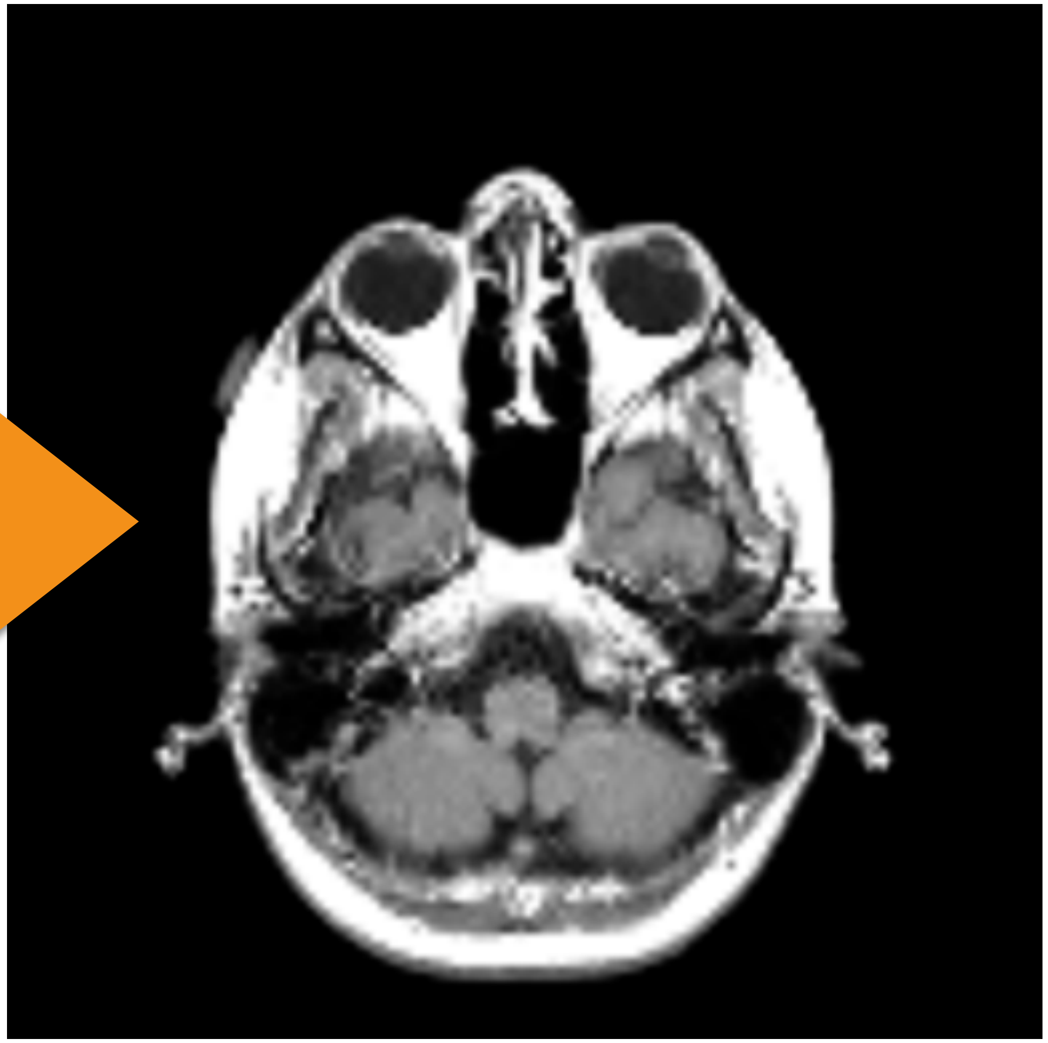
- The dynamic range may be too large, (16-bit), and most monitors handle only 8-bit!

- The operator is defined as

$$
\begin{aligned}
g(x,y) &= \log\big[f(x,y); E_{\min}; E_{\max}\big] = \\
&= (E_{\max} - E_{\min}) \cdot \frac{\log(1 + f(x,y))}{\log(1 + \max(f))} + E_{\min}
\end{aligned}
$$

Logarithmic Example





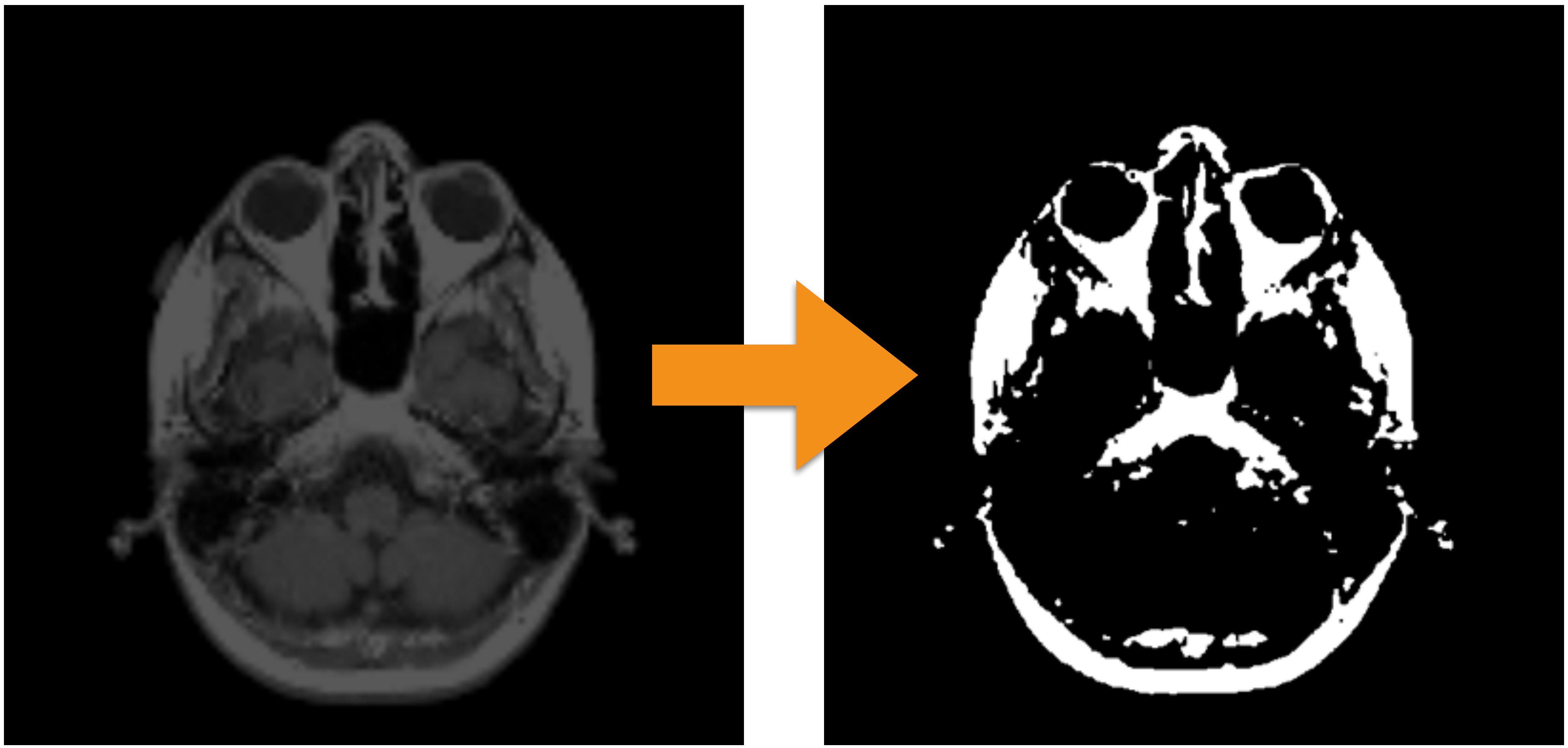
Unary Operators: Thresholding

- This operator creates a mask (0 or 1 values):

$$g(x,y) = \mathrm{Thr}\big[f(x,y);a;b\big] = \begin{cases} 1 & \text{if } f(x,y) \in [a,b], \\ 0 & \text{otherwise.} \end{cases}$$

- It can be used for segmentation.

Thresholding Example



Binary Operators

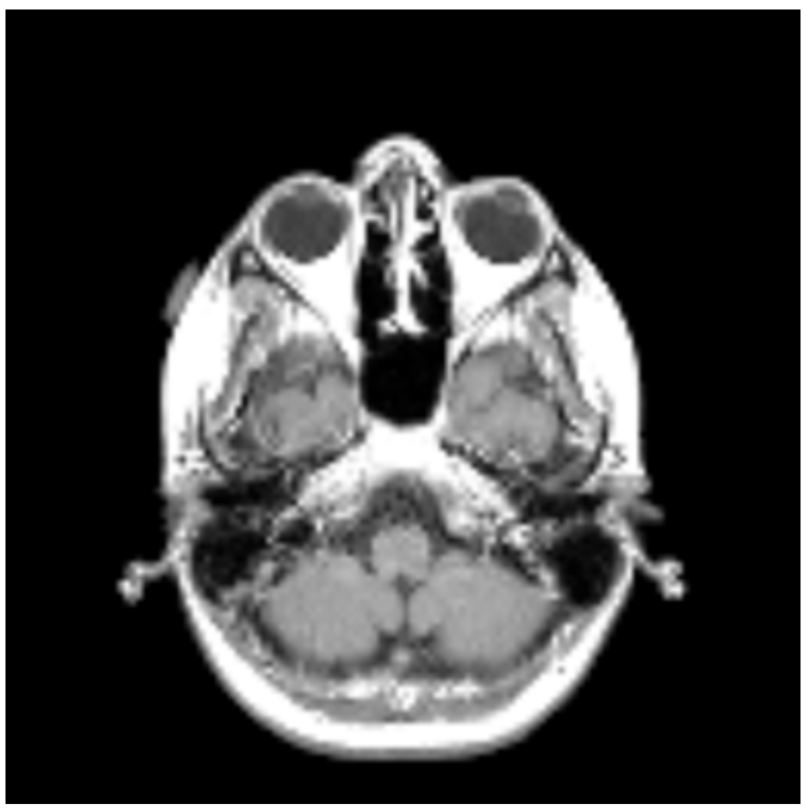
- Binary operators are typically the classic arithmetic operators defined over images:
    - +, -, *, /
- Note that using + and *, we can increase the dynamic range and obtaining values in [0,2]:
    - Logarithmic operator
    - Linear scaling in [0,1]

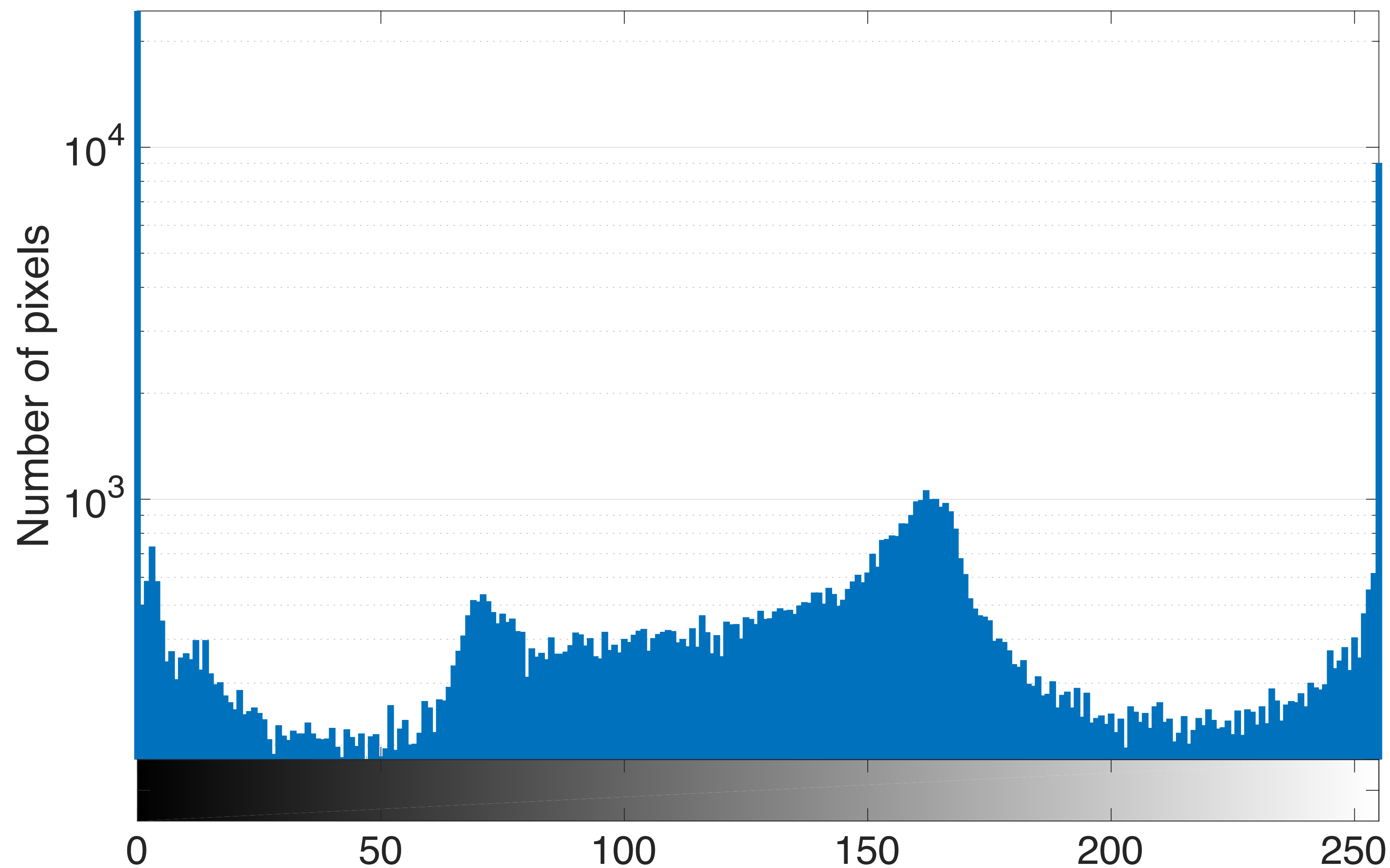
Histograms

Image Histogram

- The distribution of intensity pixel values.
  - This can be seen as the probability of a pixel of having a given intensity value.
- How to compute?
  - For each unique intensity value J:
    - Count how many pixels have J as intensity
  - MATLAB: **imhist** built-in function

Example





Each intensity value is called “bin”

Histogram Equalization

- A technique to improve automatically the contrast of the image.

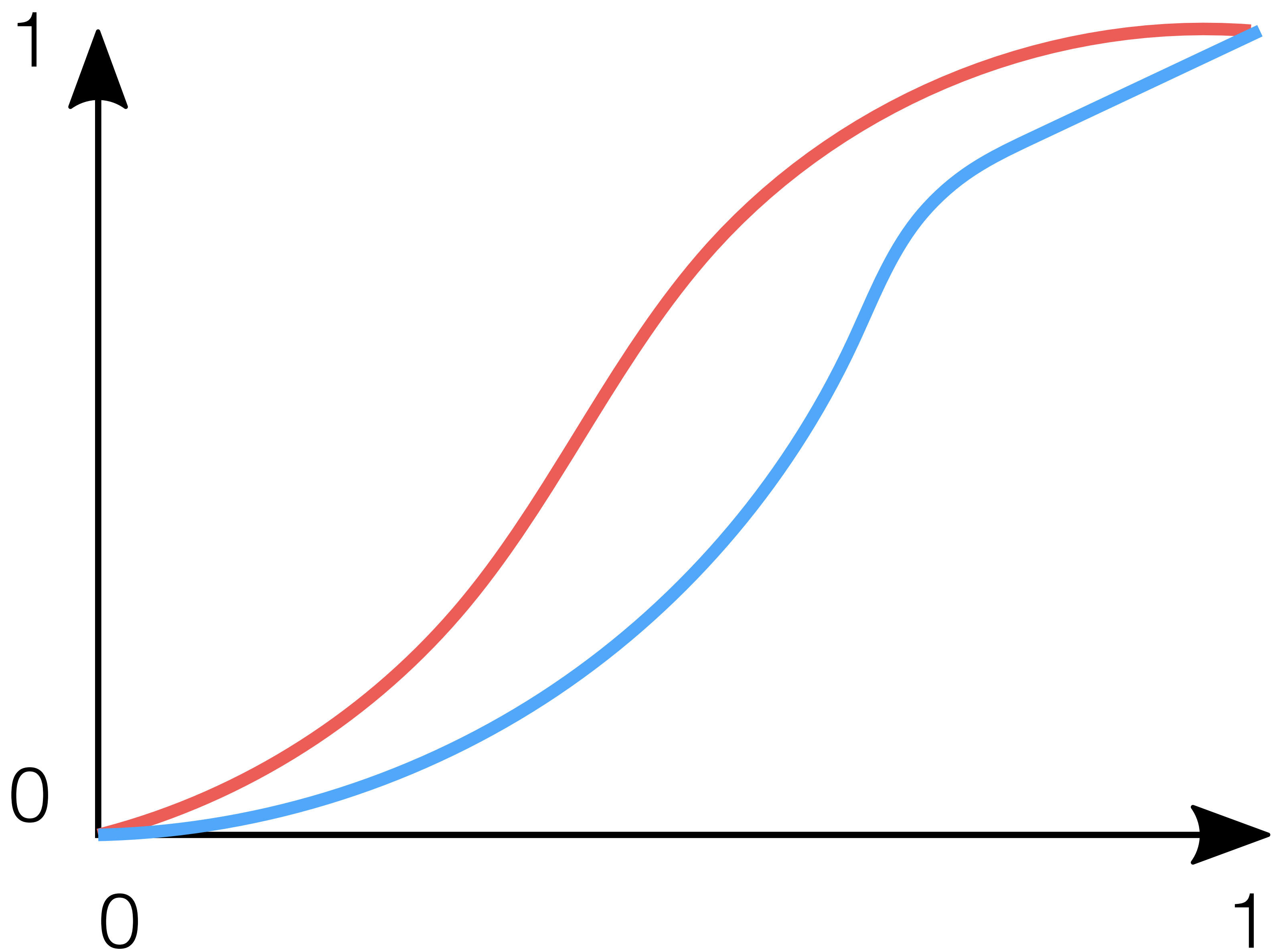
- The main idea is to have an histogram in which each intensity value J (or bin) has the **same** (more or less) number of pixels:

    - $H[J] = n\_pixels\_image / 2^{n\_bit}$

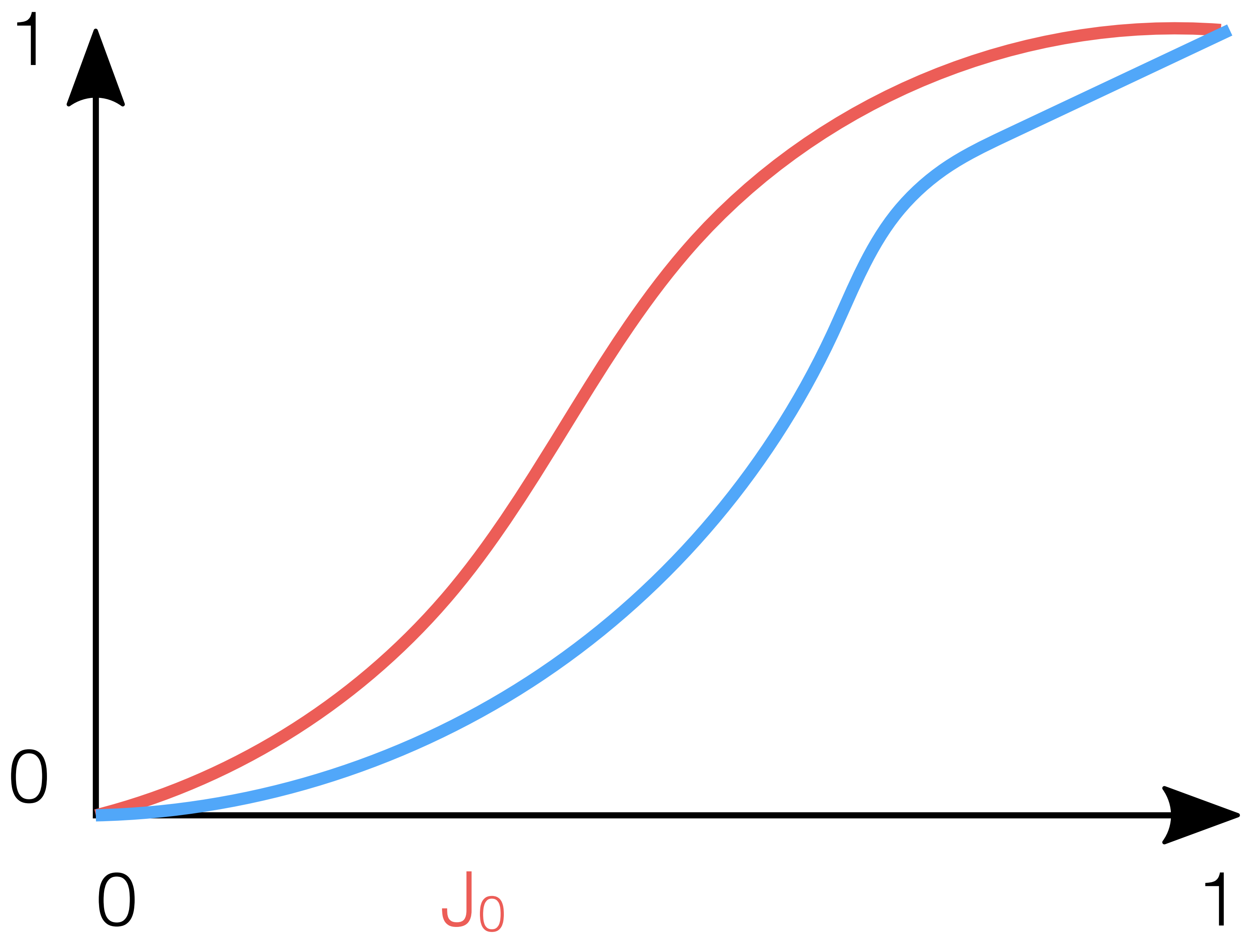
- How? Matching the CDF (cumulative distribution function) of the histogram with the CDF of a uniform histogram.

    - This uniform CDF —> $y(x) = x$ with x in $[0,1]$

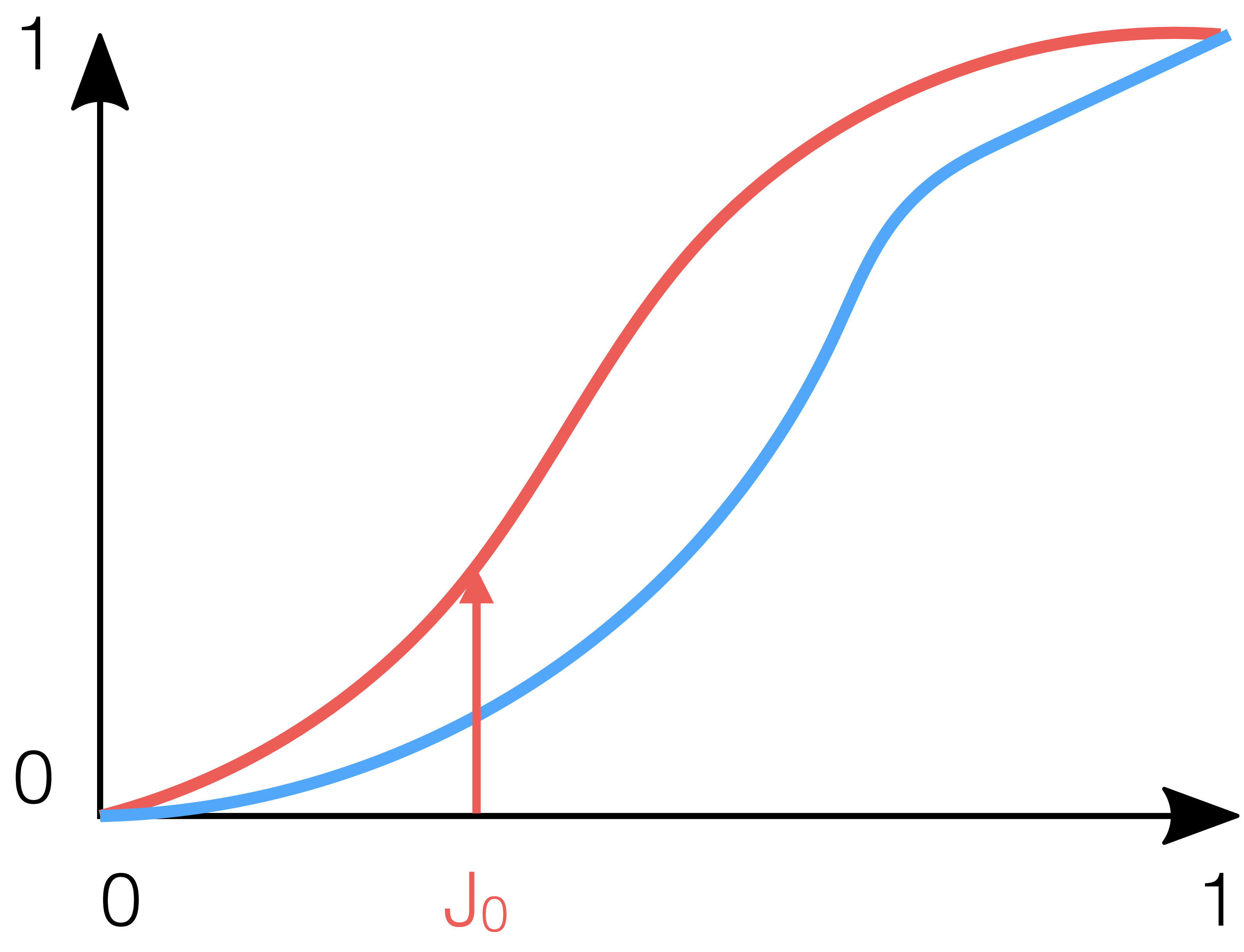
Histogram Equalization: Histogram Matching



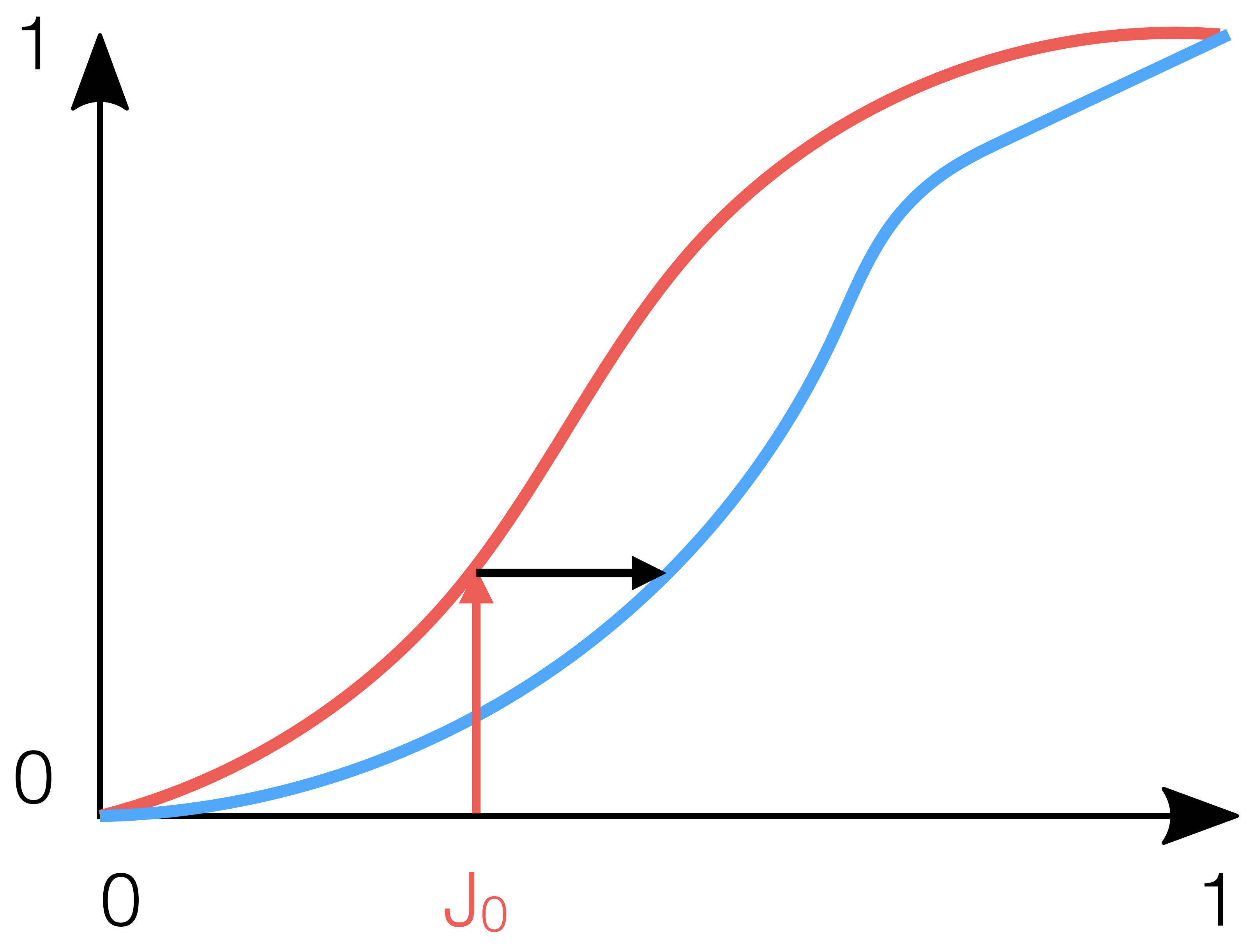
Histogram Equalization: Histogram Matching



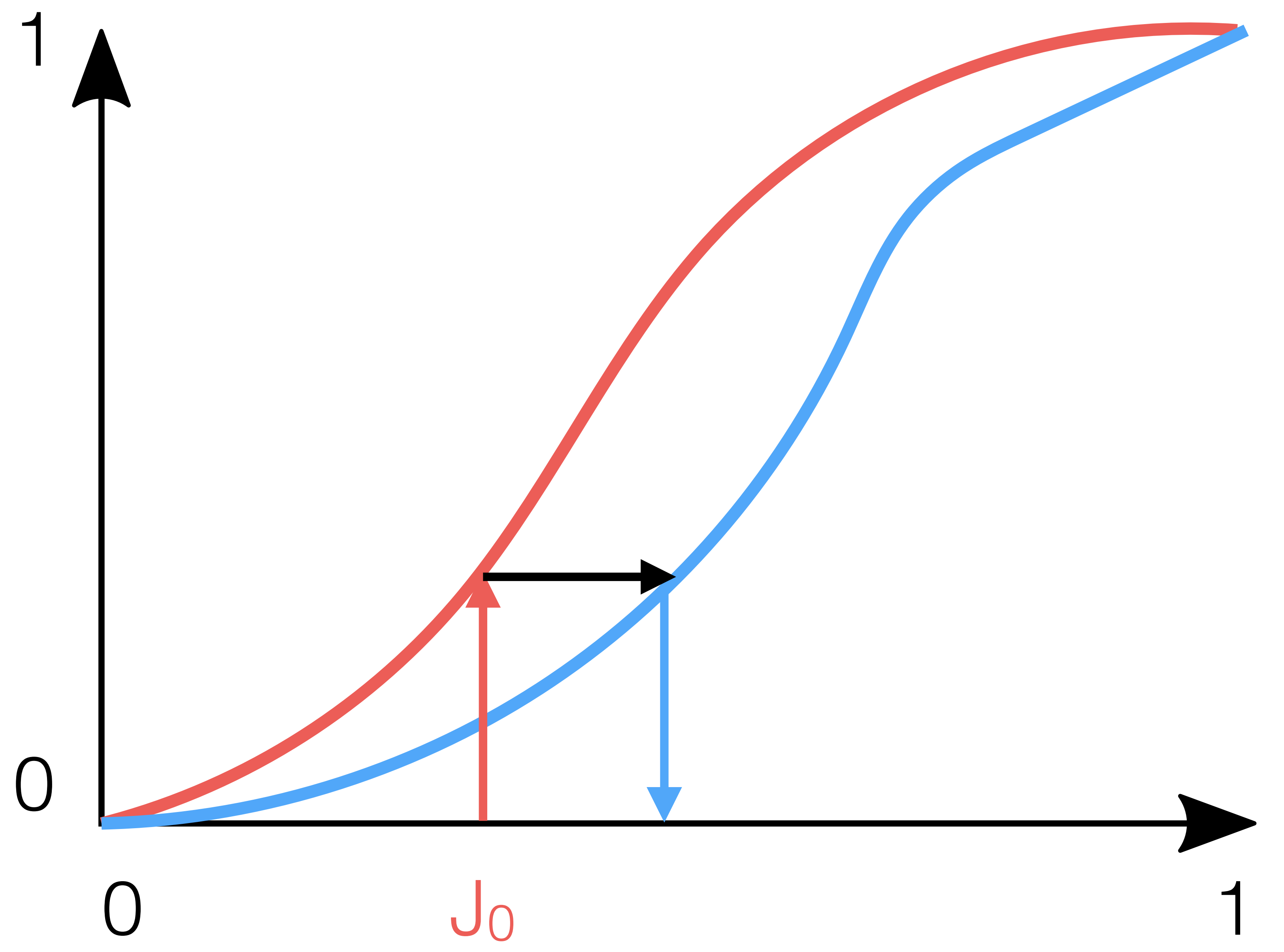
Histogram Equalization: Histogram Matching



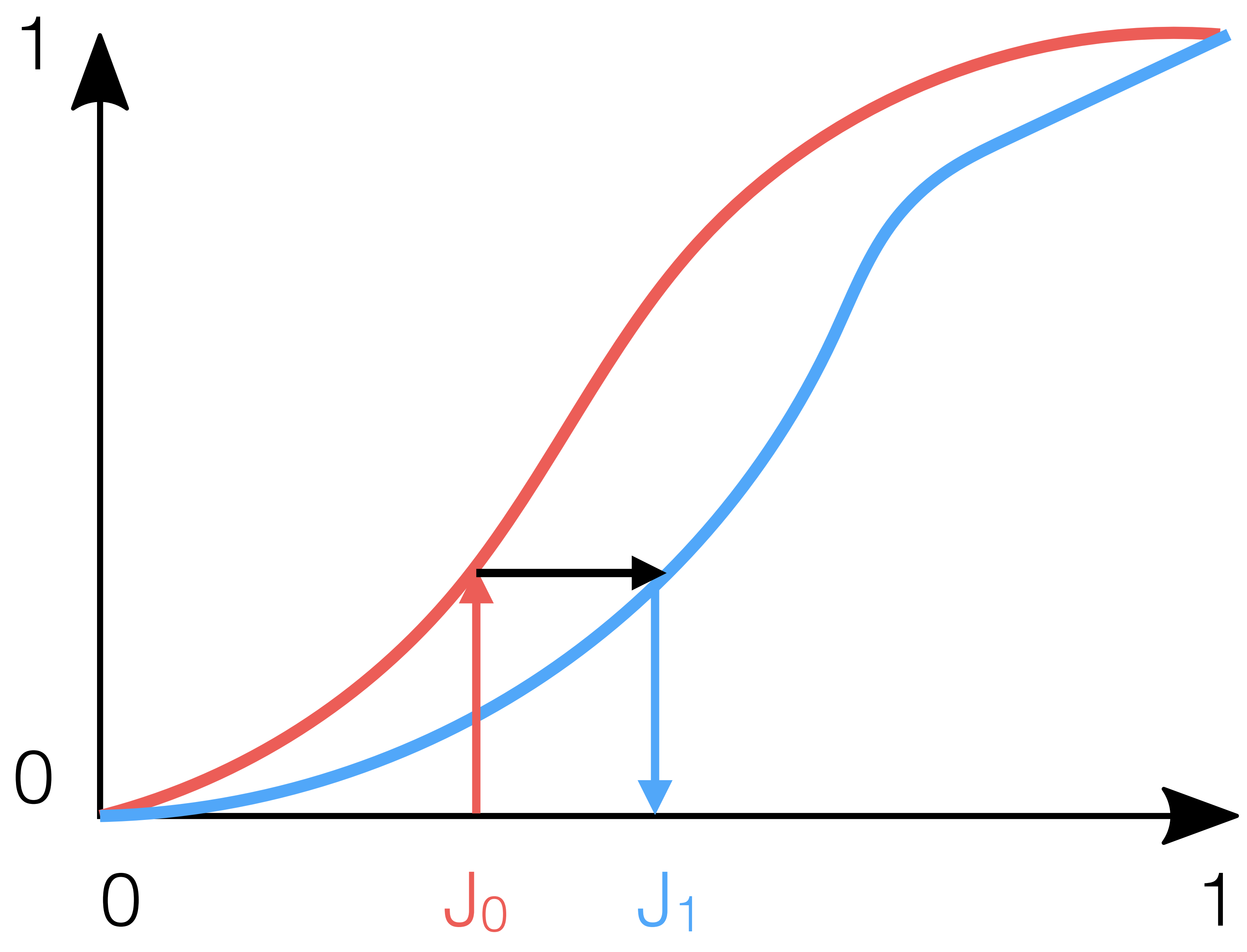
Histogram Equalization: Histogram Matching



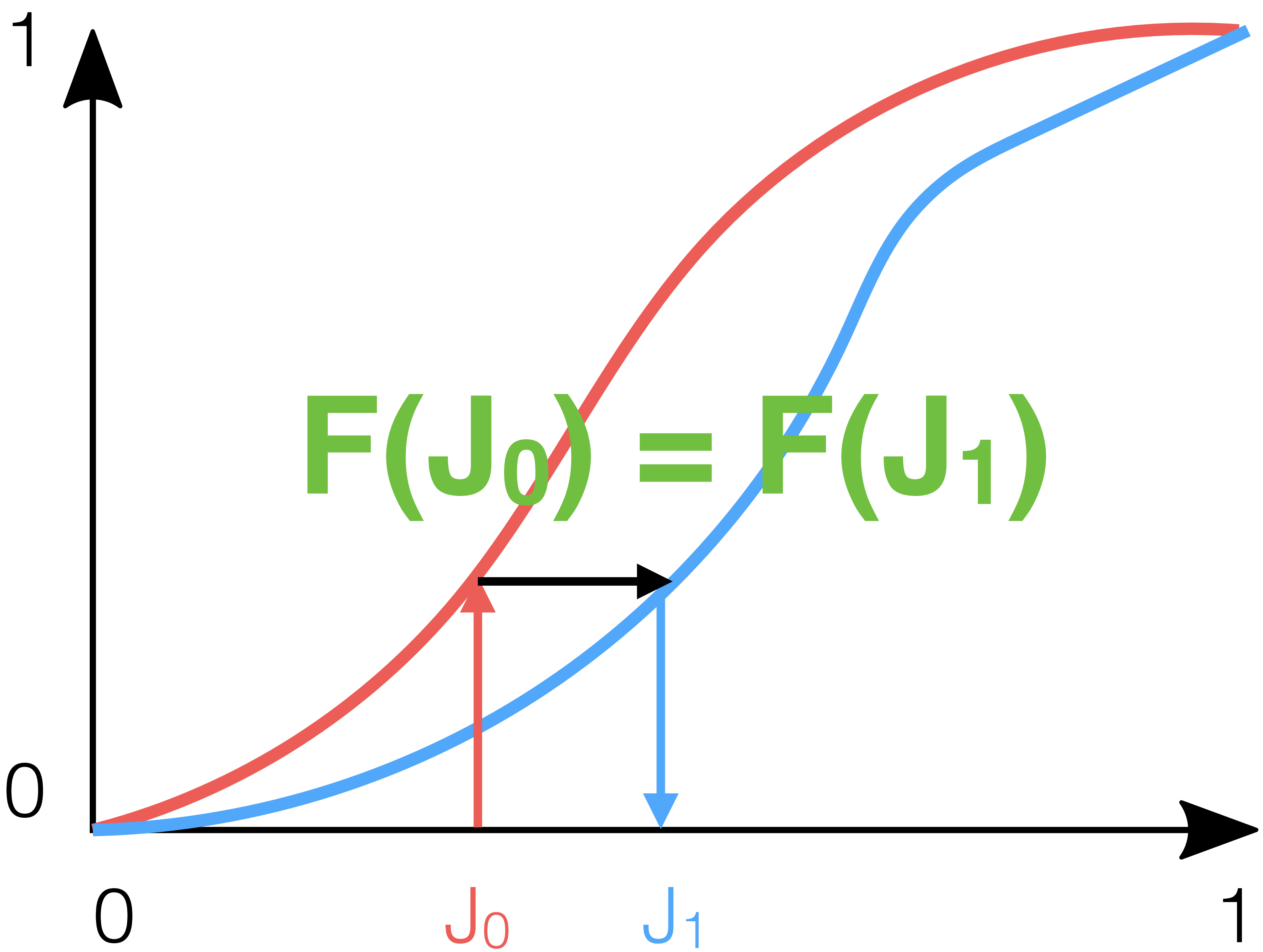
Histogram Equalization: Histogram Matching



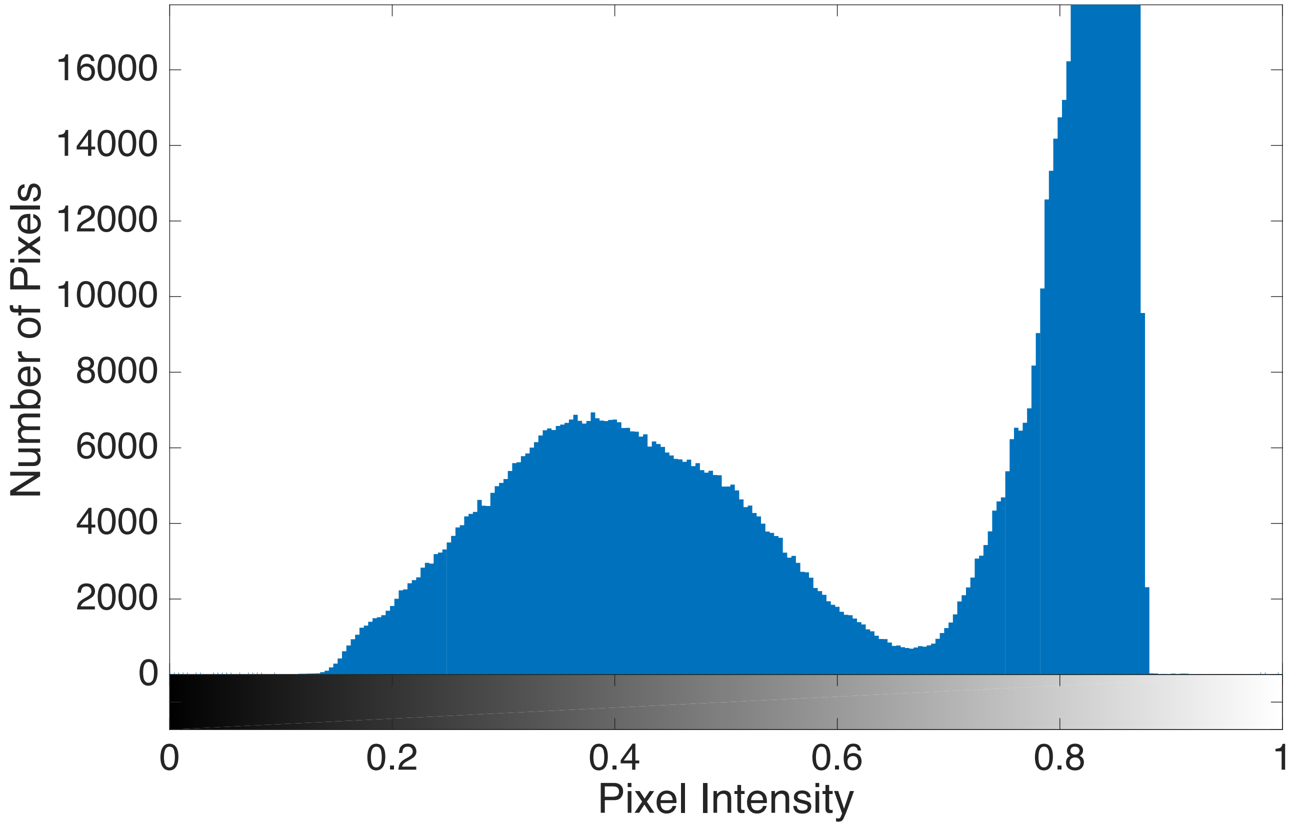
Histogram Equalization: Histogram Matching



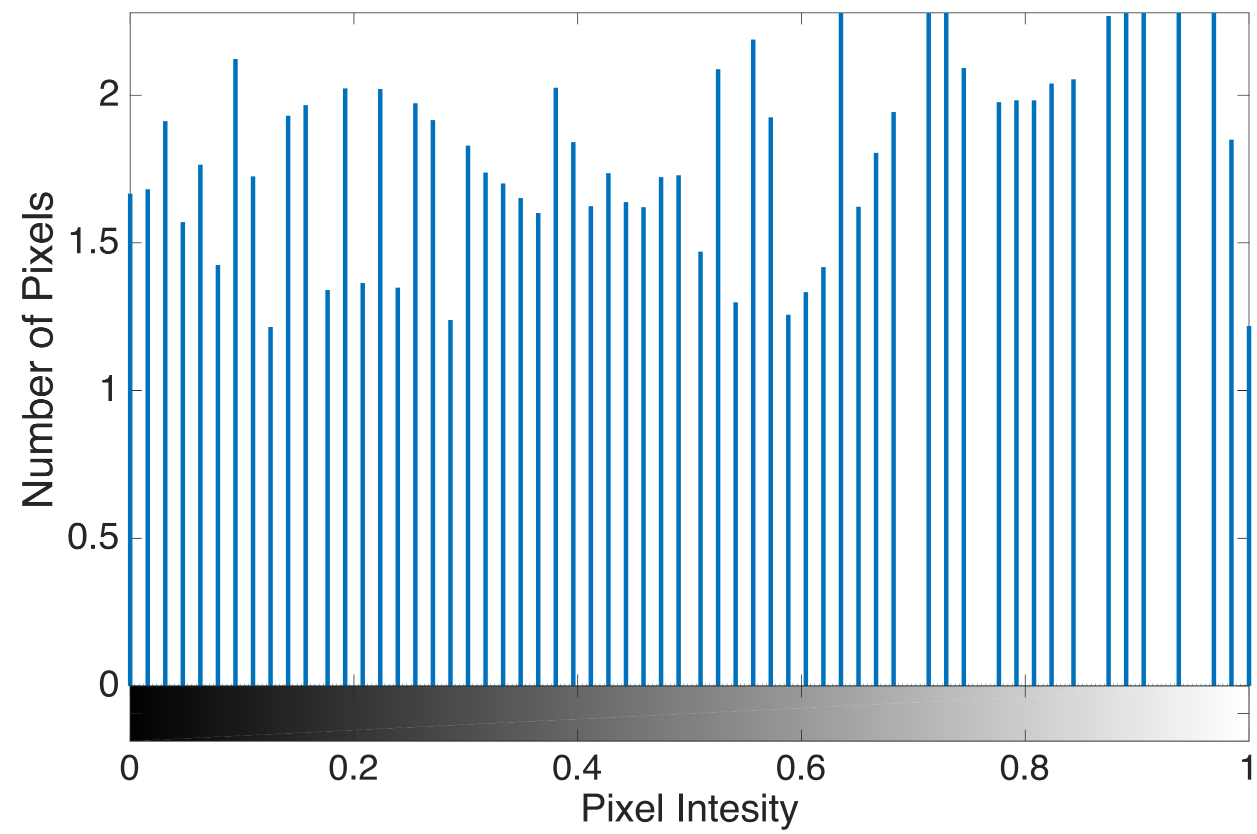
Histogram Equalization: Histogram Matching



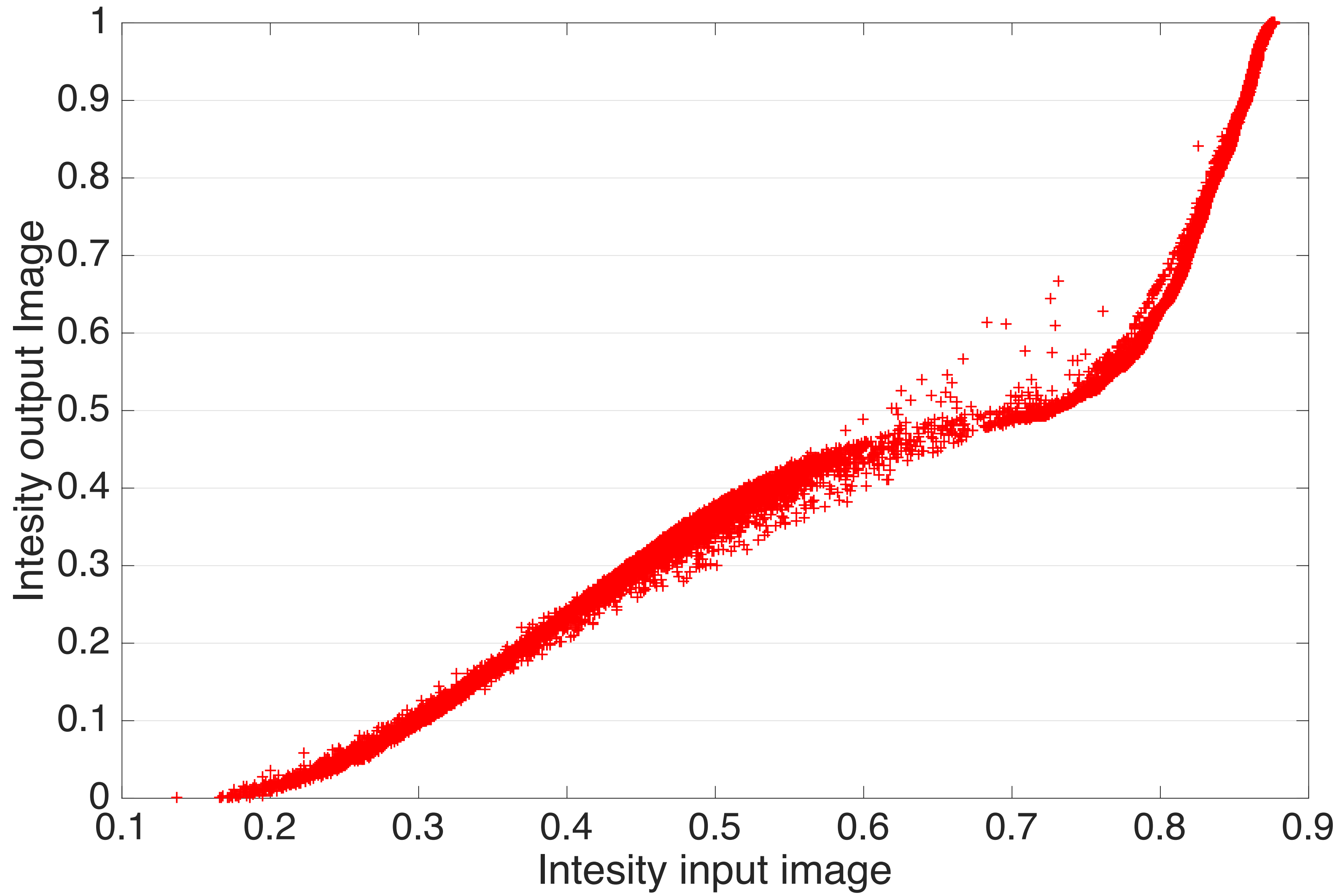
Histogram Equalization Example



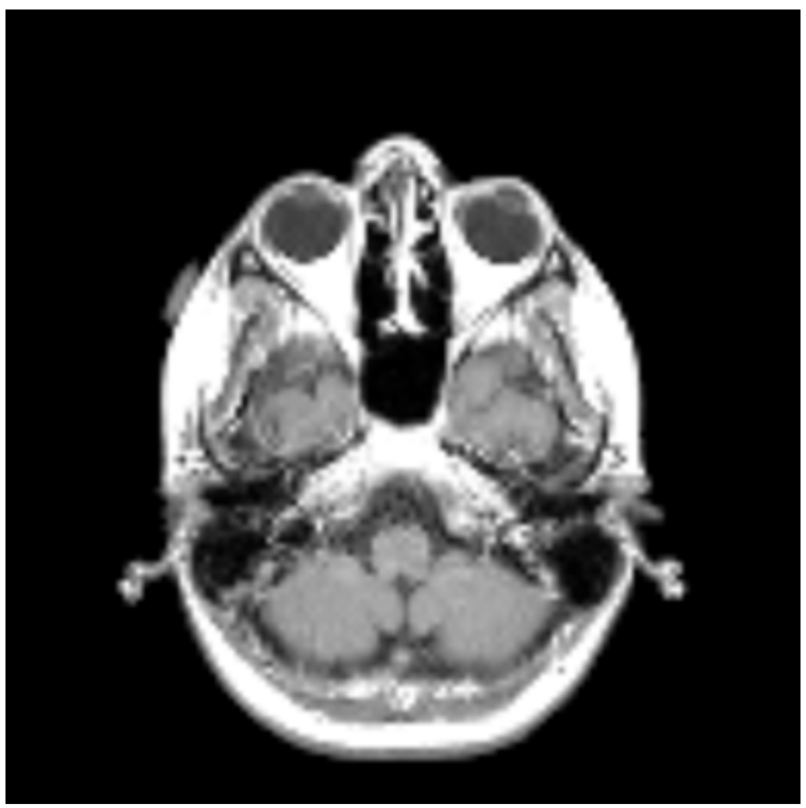
Histogram Equalization Example

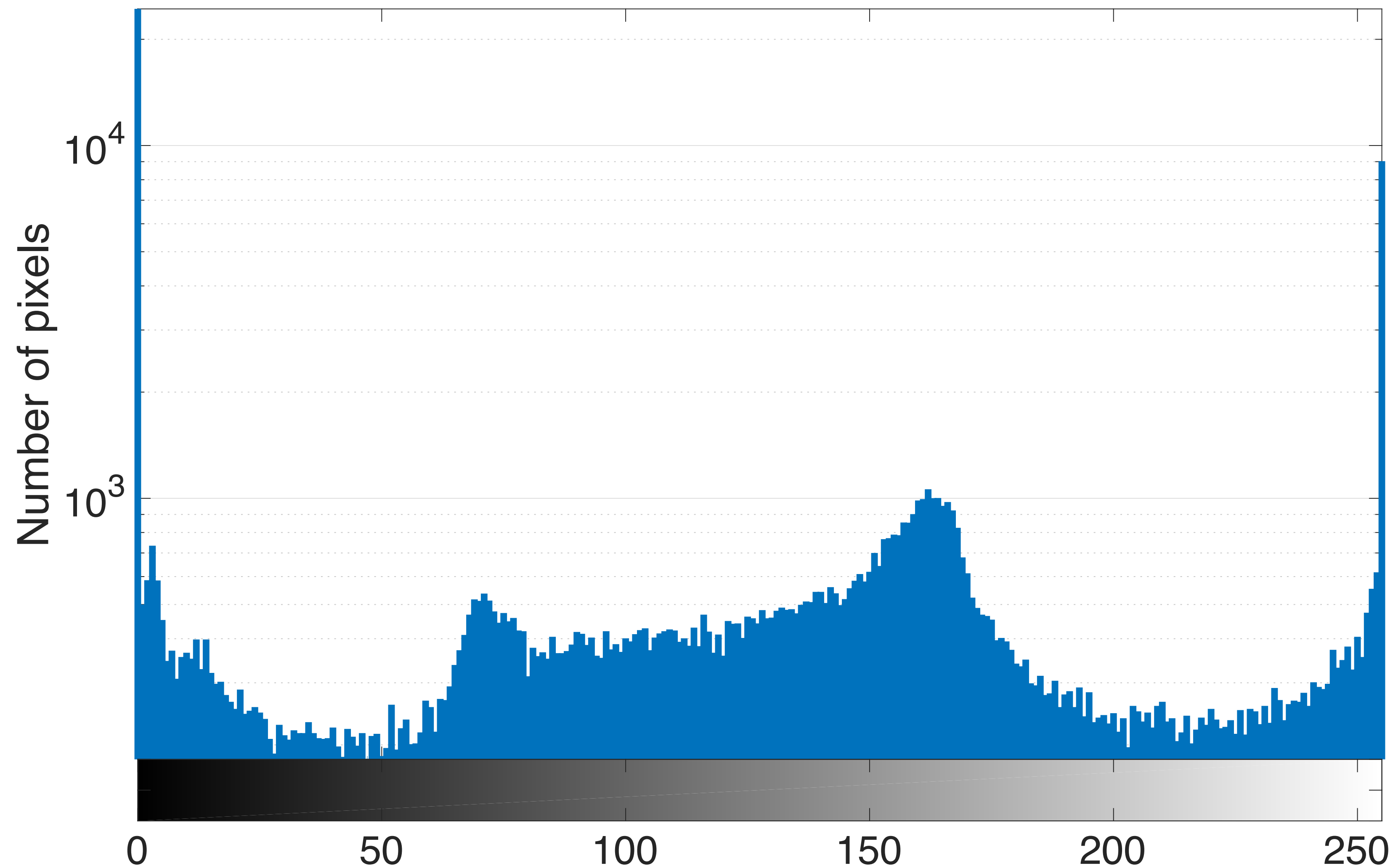


Histogram Equalization Example



Histogram Equalization Example





ROI helps in cases of huge peaks (see I=0)

Linear Filters

1D Convolution

- Given two functions f and g, f convolved g is defined as

$$(f \otimes g)[x] = \int_{-\infty}^{+\infty} f(t) \cdot g(t-x)dx =$$
$$= \int_{-\infty}^{+\infty} f(x-t) \cdot g(x)dx$$

- In the discrete world, this leads to:

$$(f \otimes g)[i] = \sum_{j=-N}^{N} f[i-j] \cdot g[j]$$

2D Convolution

- In the 2D world, this leads to:

$$(f \otimes g)[i,j] = \sum_{k=-N}^{N} \sum_{l=-M}^{M} f[i-k, j-l] \cdot g[k,l]$$

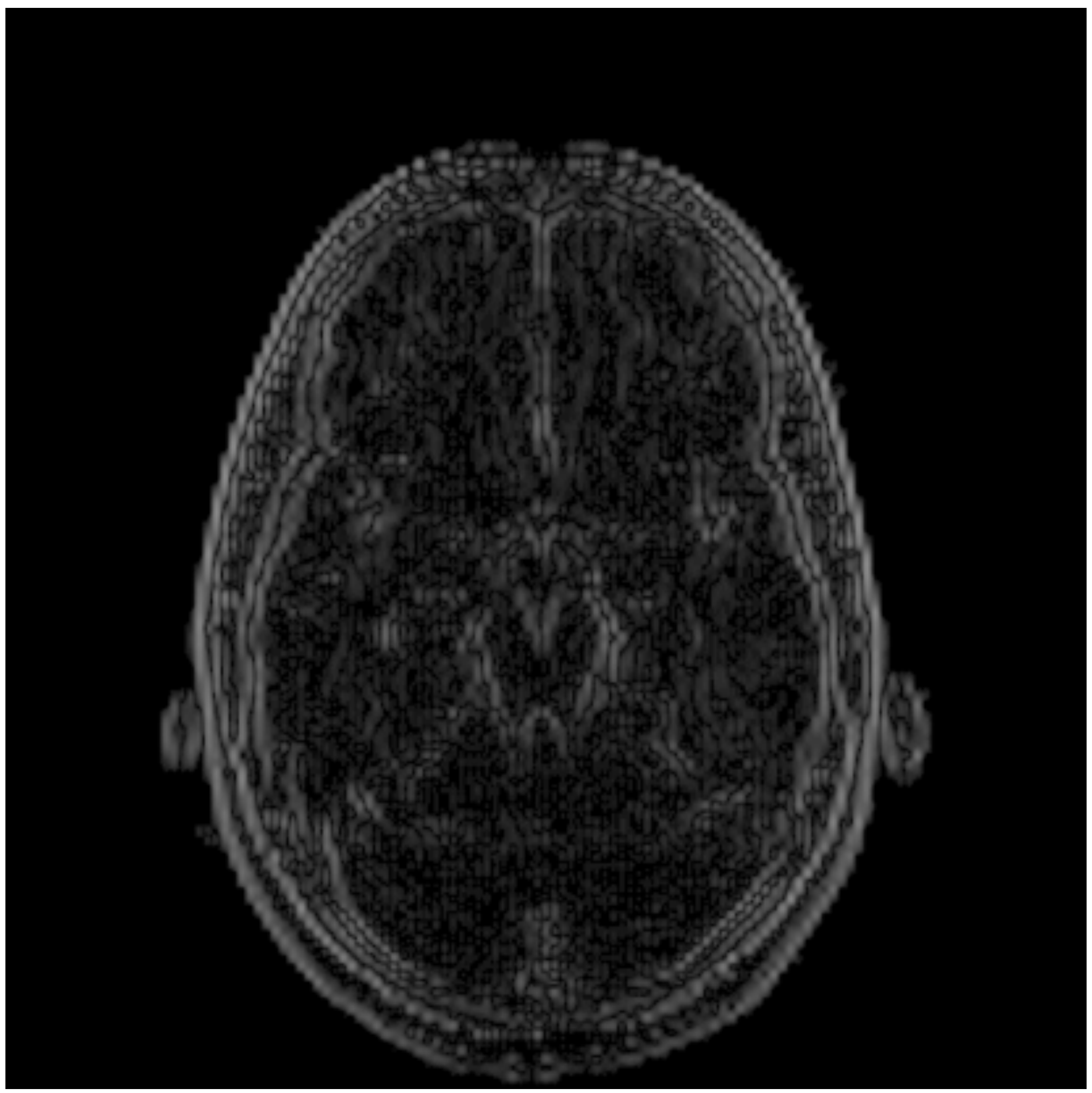
- where g is a (2n)-(2m) matrix, called kernel.
  - For sake of simplicity, let’s assume negative addresses!
- MATLAB: **conv** (1D convolution), and **conv2** (2D convolution) built-in functions

Gradient Filter

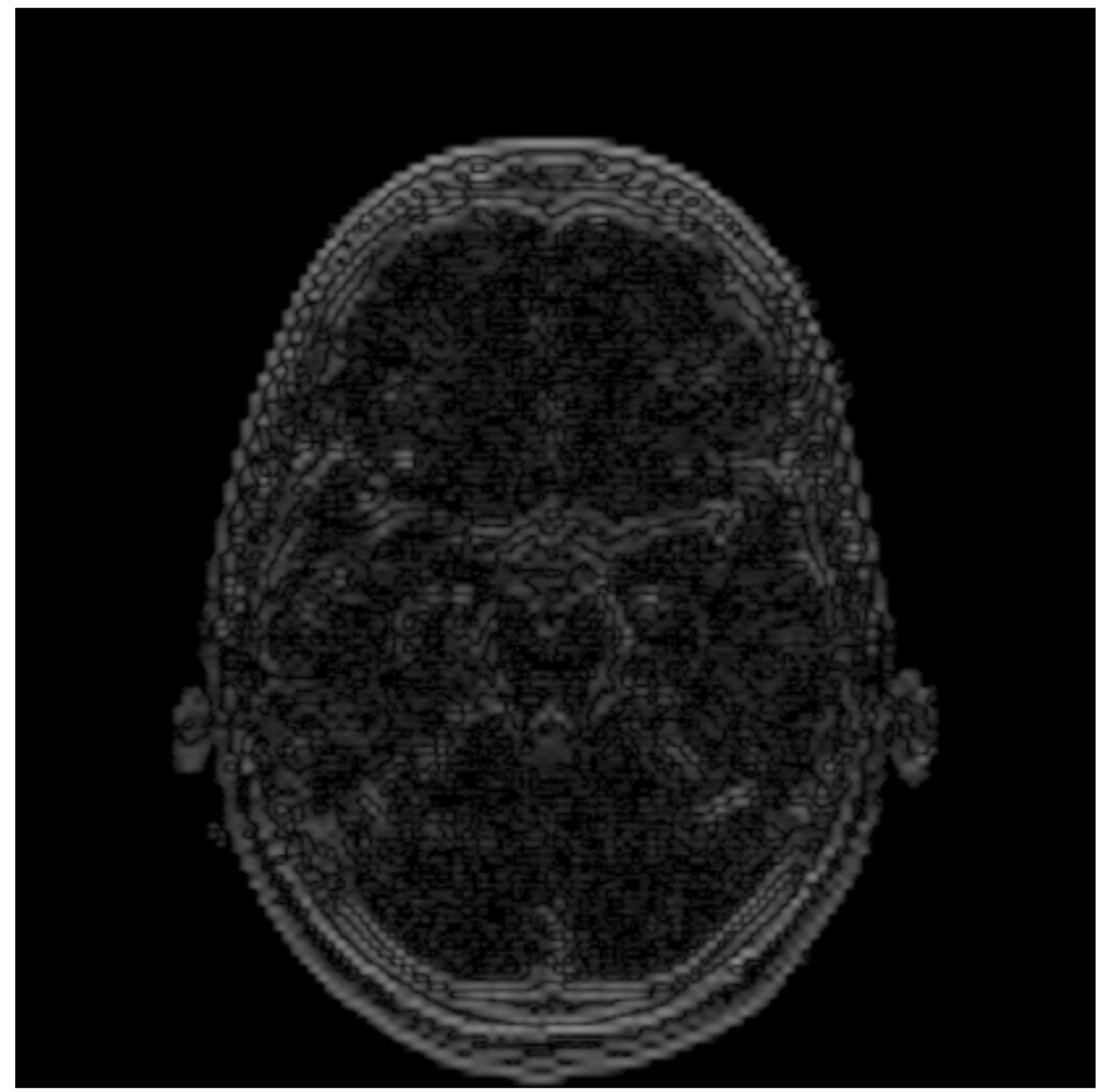
- The gradient of an image is an important piece of information:
  - Where it is high implies we may have an edge; i.e., a boundary between two different regions.
- Typically, kernels for computing gradients are defined as

$$g_X = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad g_Y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

Gradient Operator Example

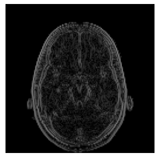
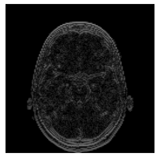


g_X



g_Y

Gradient Operator Example

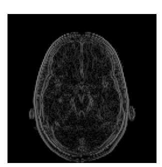
$\|\nabla G\| = \sqrt{\Bigg(}$ $[$  $]^2 +$ $[$  $]^2$ $\Bigg) =$

gX

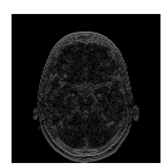
gY

Gradient Operator Example

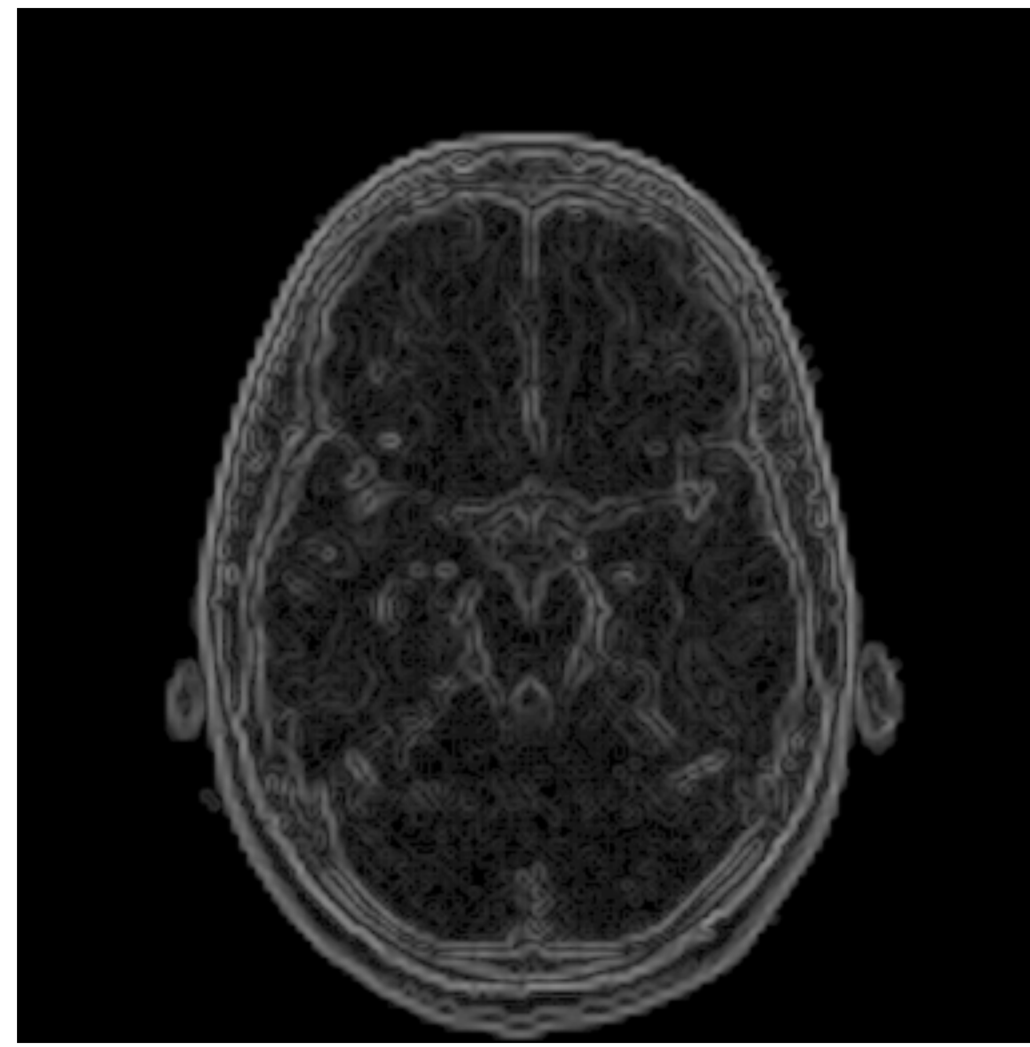
$$\|\nabla G\| = \sqrt{\left([\quad]^2 + [\quad]^2 \right)} =$$



gX



gY

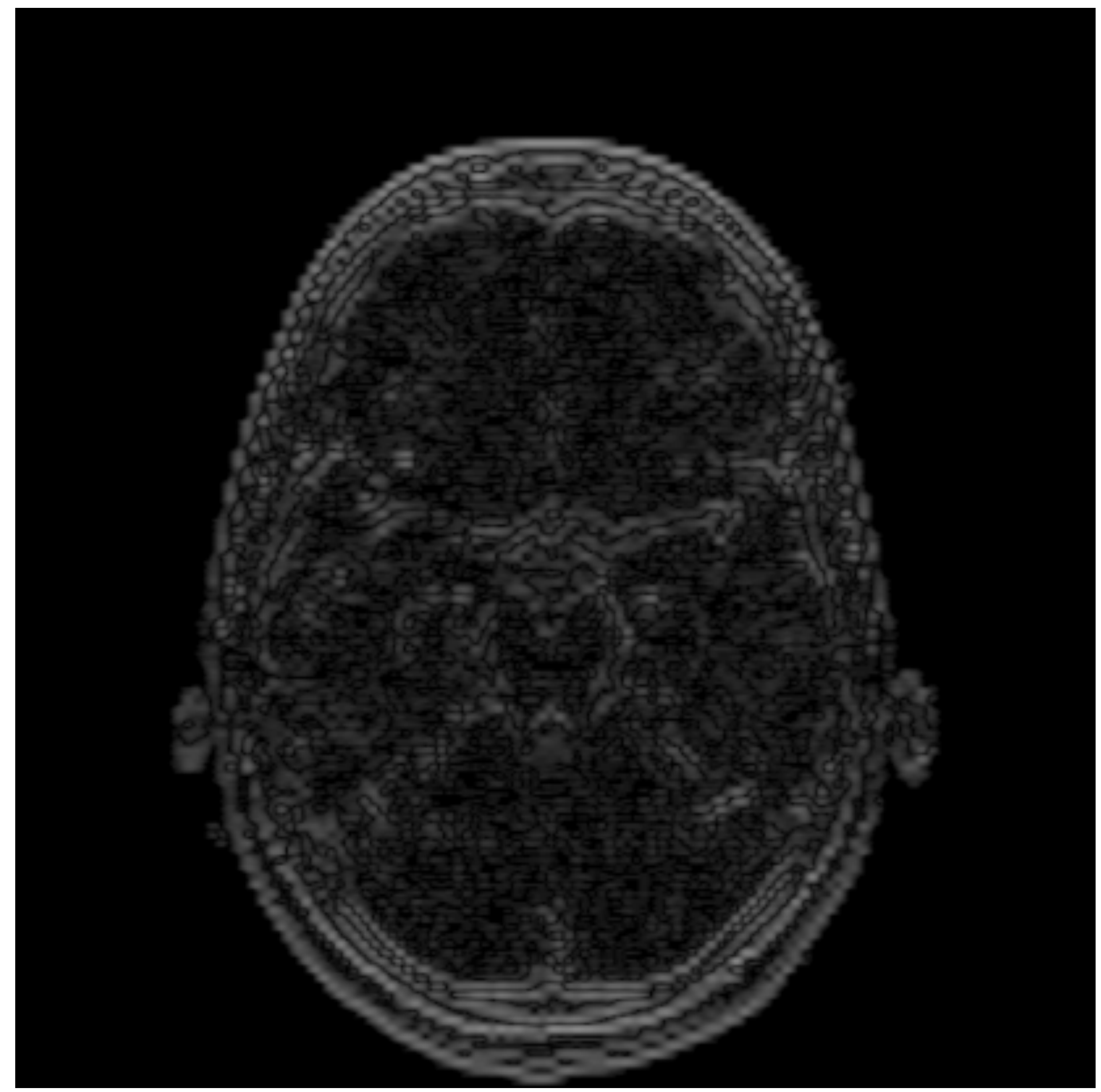


Sobel Gradient Operator

- Technically speaking, it is just another discrete differential operator!
- It emphasizes more edges, which is good.

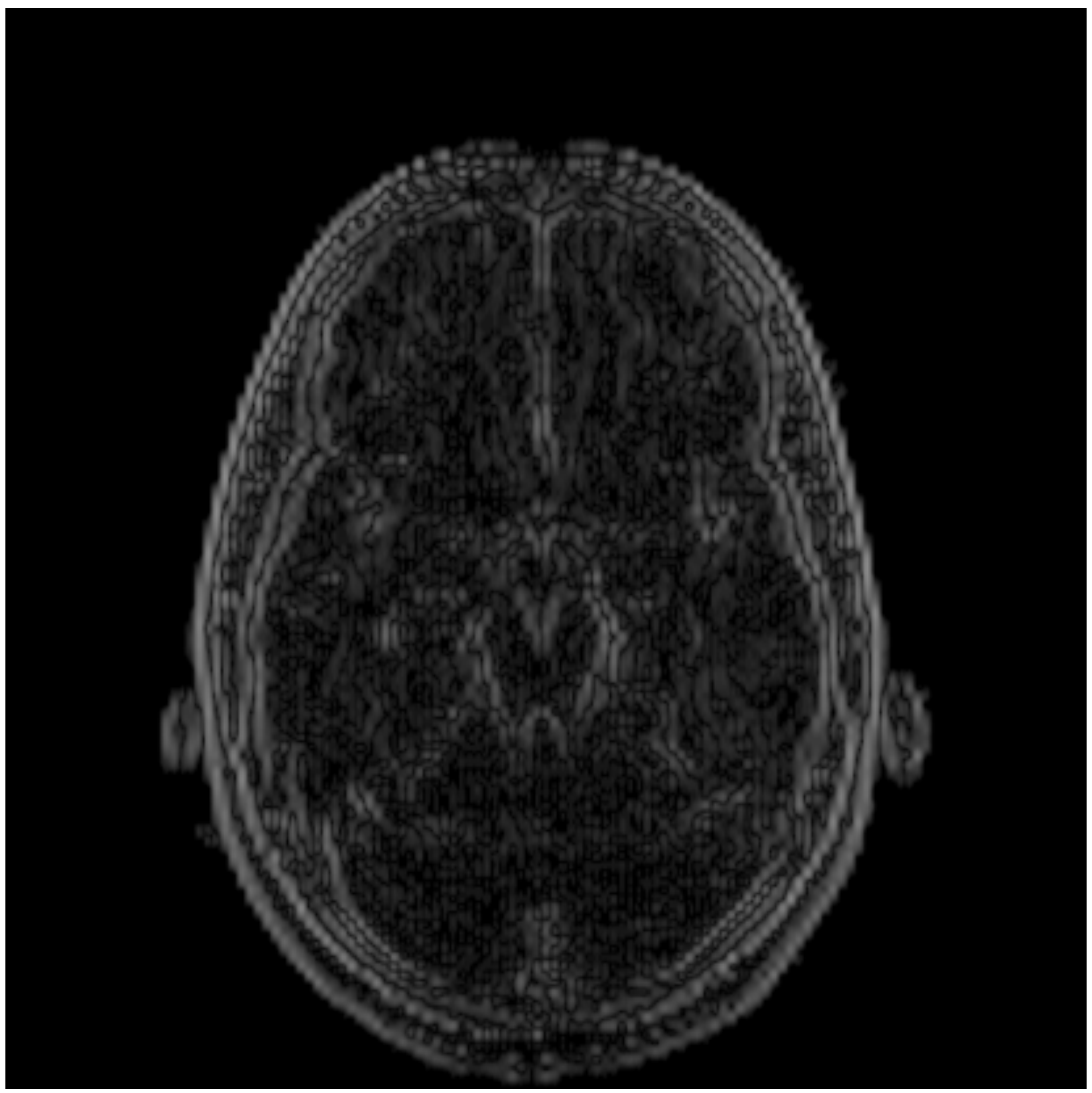
$$g_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad g_Y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & 1 \end{bmatrix}$$

Sobel Gradient Operator Example

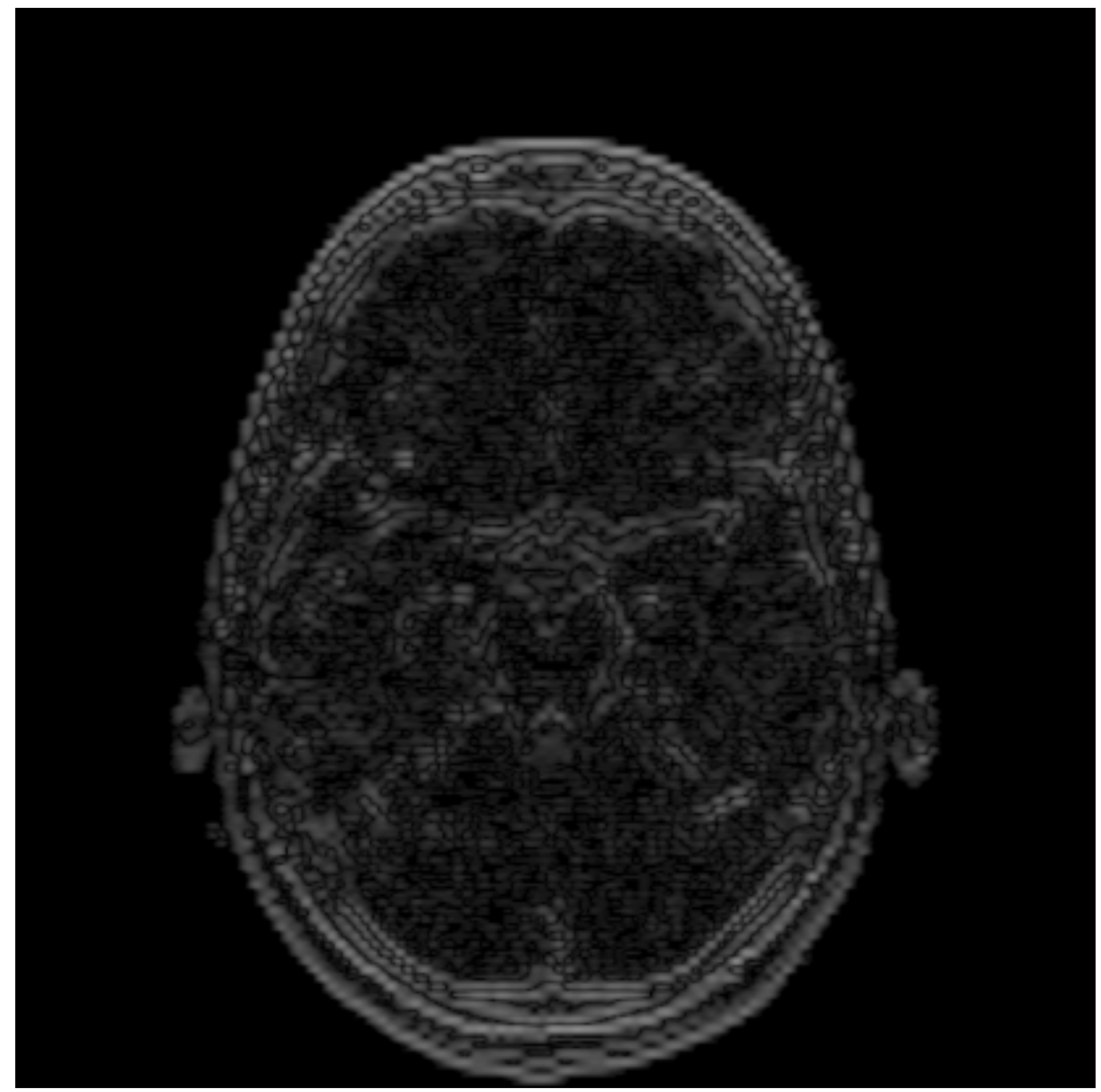


g_X g_Y

Sobel Gradient Operator Example

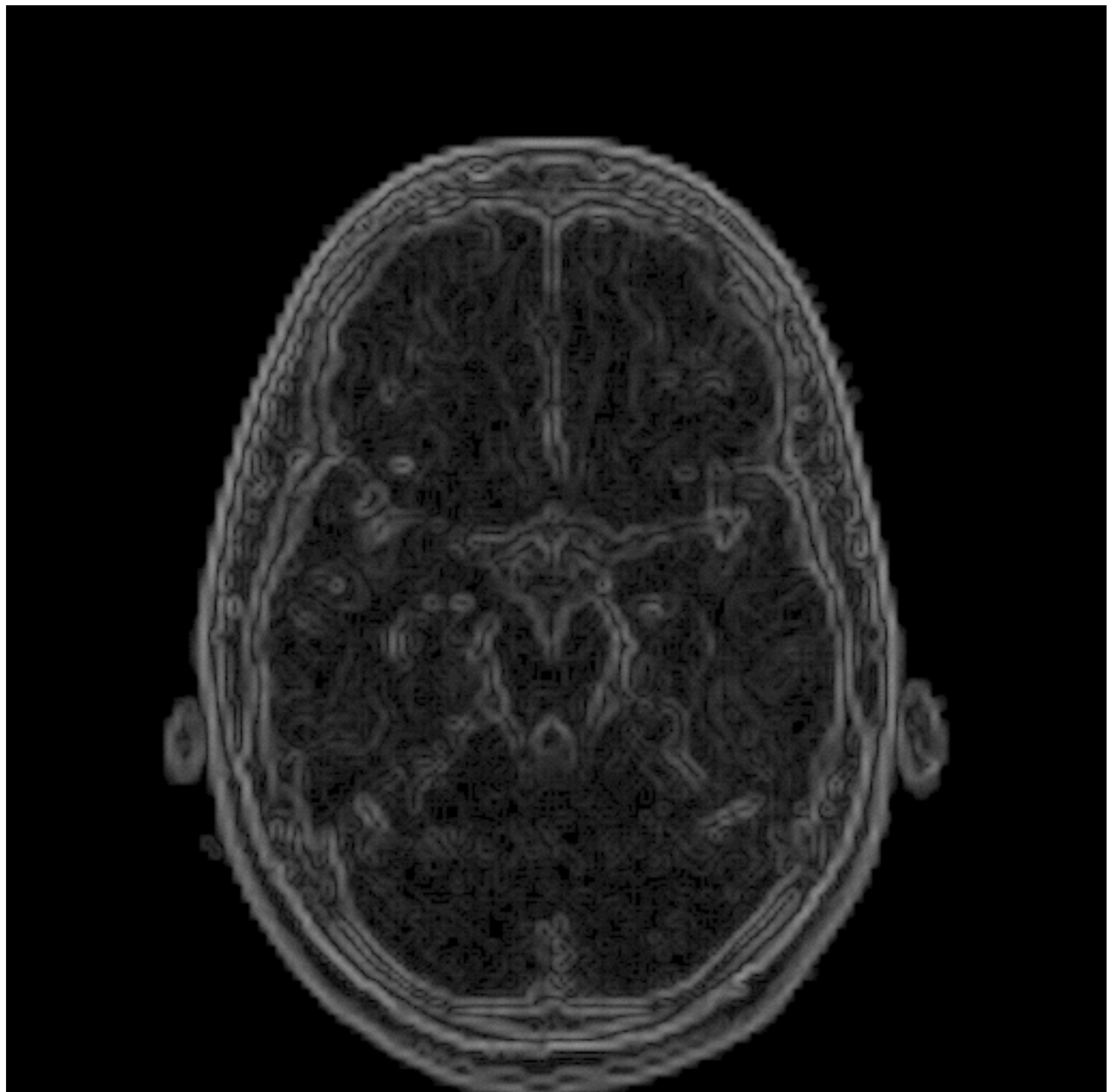


g_X



g_Y

Sobel Gradient Operator Example



Edge Detectors

- Edges are can be helpful for defining regions
- They help in the visualization of what we want to segment

Edge Detectors

- Steps:
  - Compute gradients (magnitude and angle of orientation [**atan2**])
  - Non-maximum suppression —> remove low power stuff
  - Apply double thresholding; classification: strong, weak, and no edge
  - Edge tracking; a weak edge is a strong one if it is connected to a strong edge!

Edge Detector Example

thr = 0.001

thr = 0.01

thr = 0.1

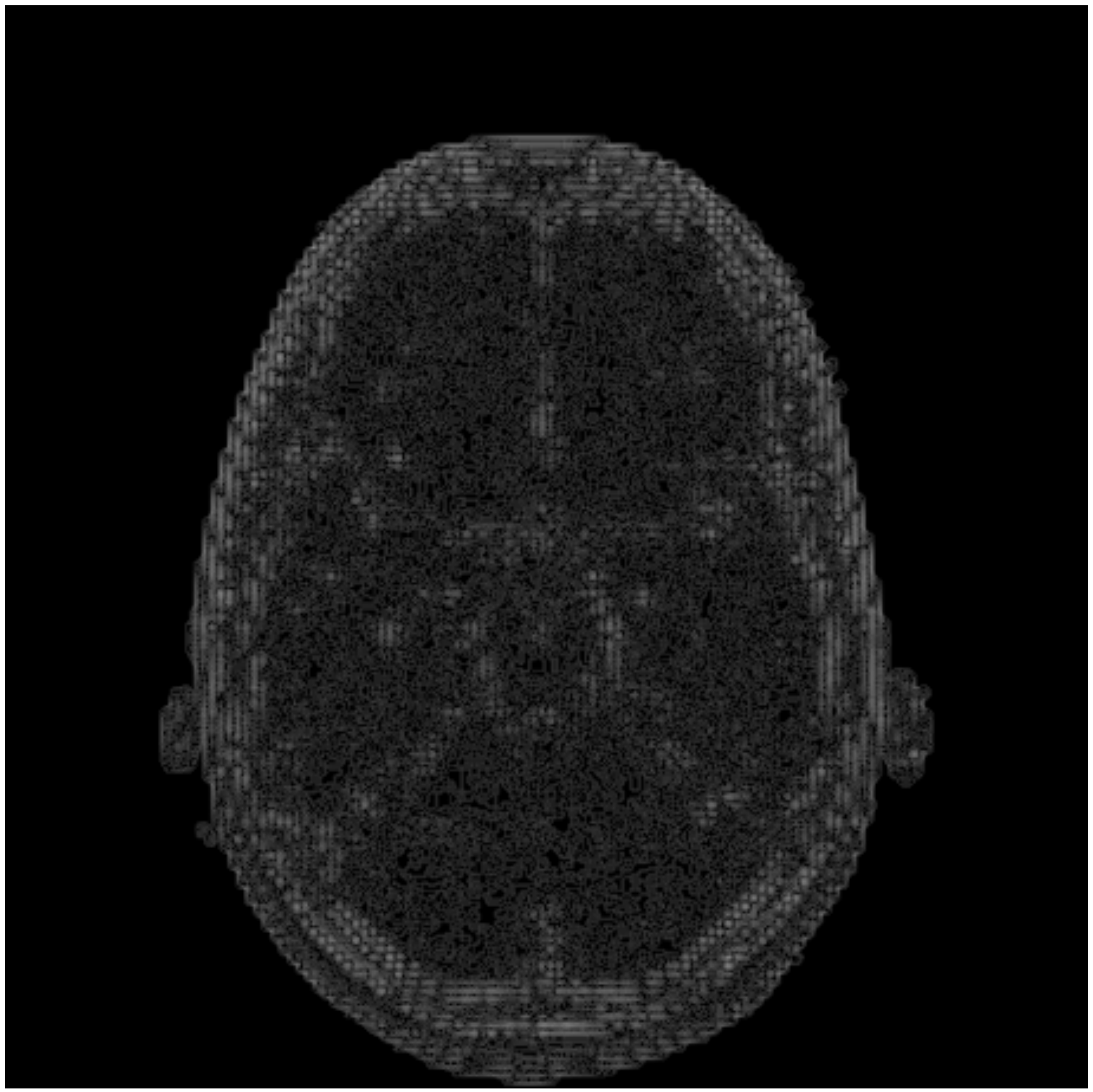
Laplacian Filter

- If you really want… we can also define a Laplacian operator… Why?
  - The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection
  - oh, jolly good!

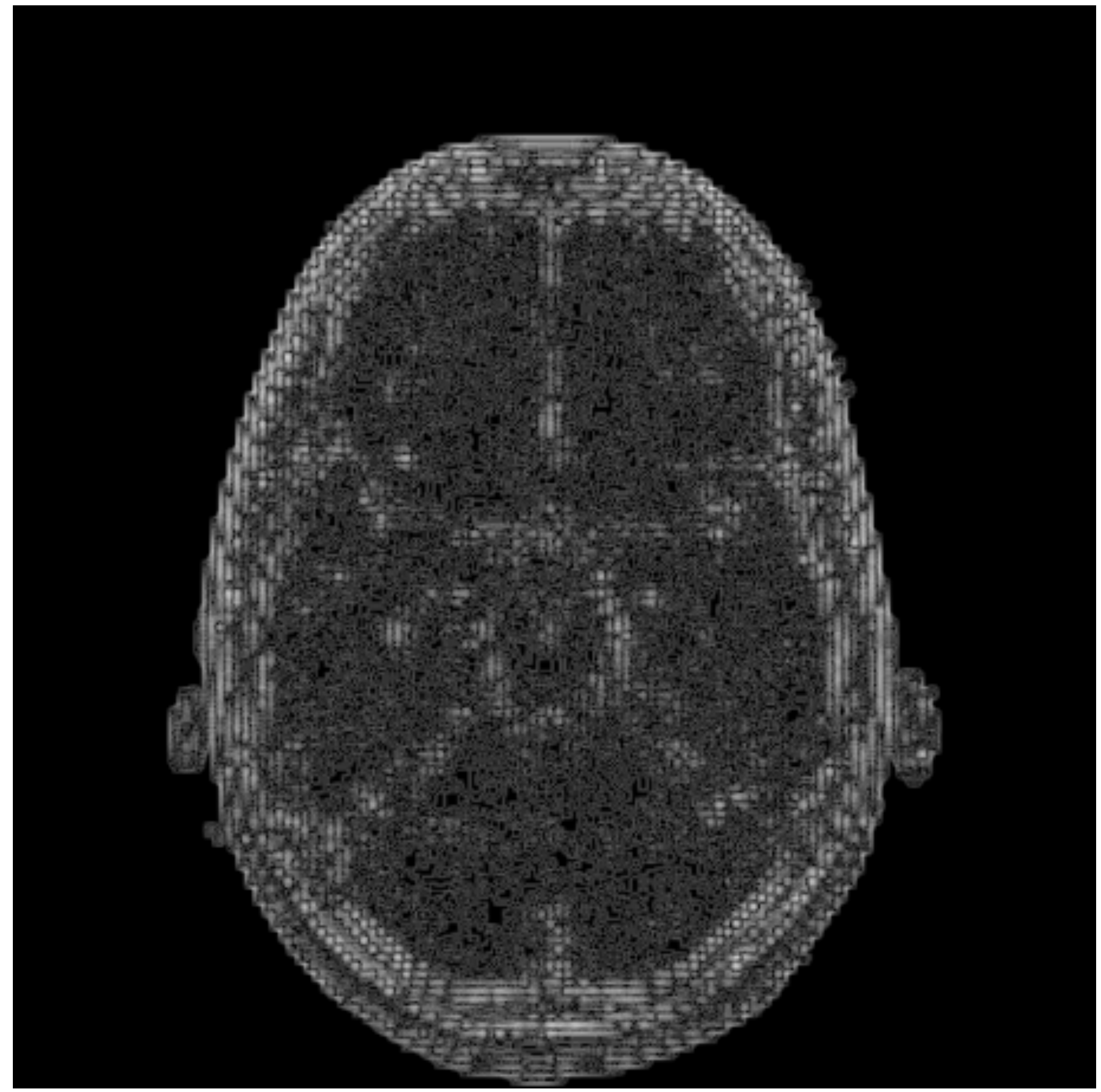
Laplacian Filter

$$g_{L_4} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad g_{L_8} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Laplacian Filter Example



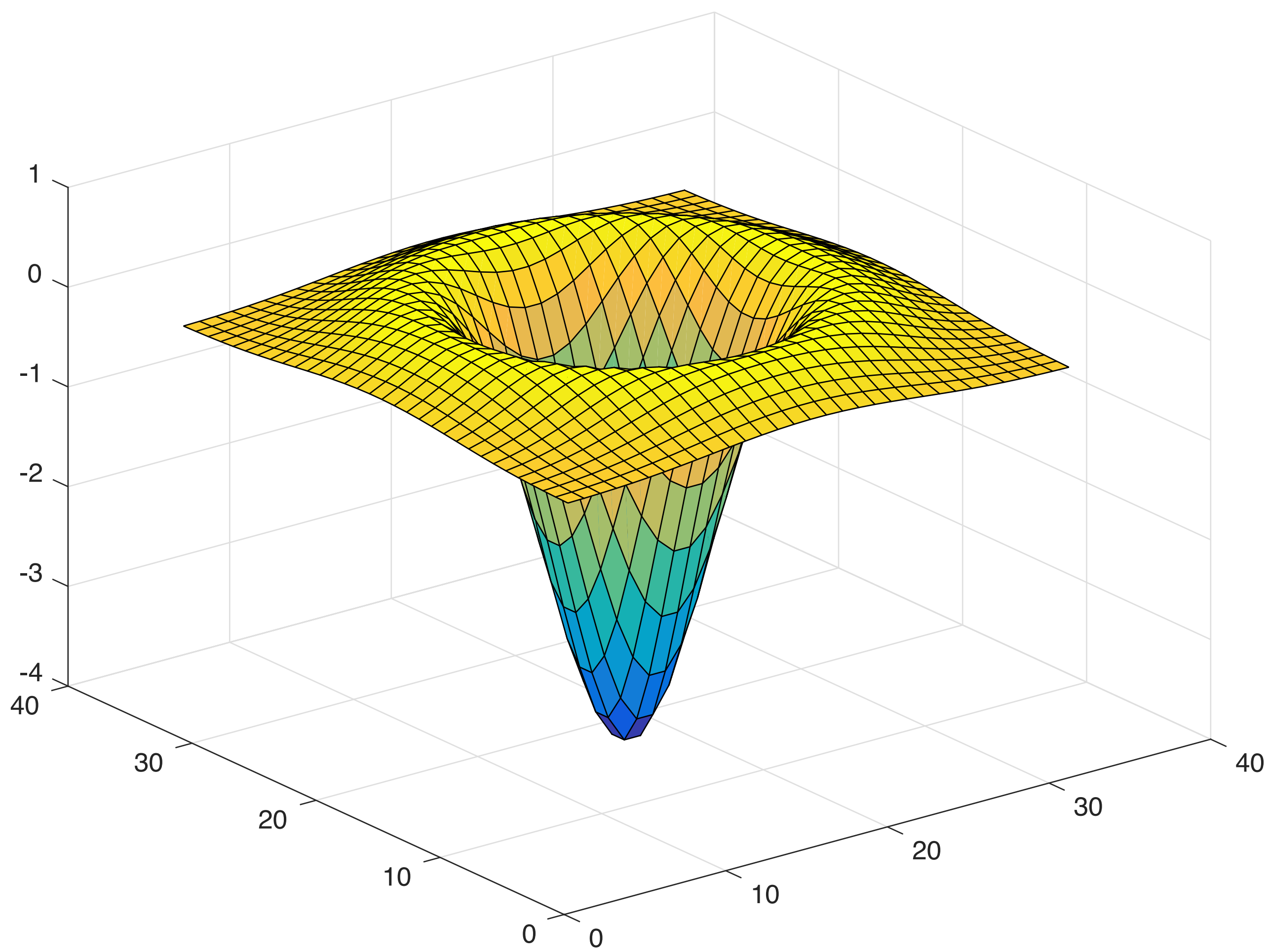
g_{L4}



g_{L8}

Laplacian Filter Extra

$$g_{LoG}[k,l] = -\frac{1}{\pi\sigma^4}\left(1 - \frac{k^2+l^2}{2\sigma^2}e^{-\frac{k^2+l^2}{2\sigma^2}}\right)$$

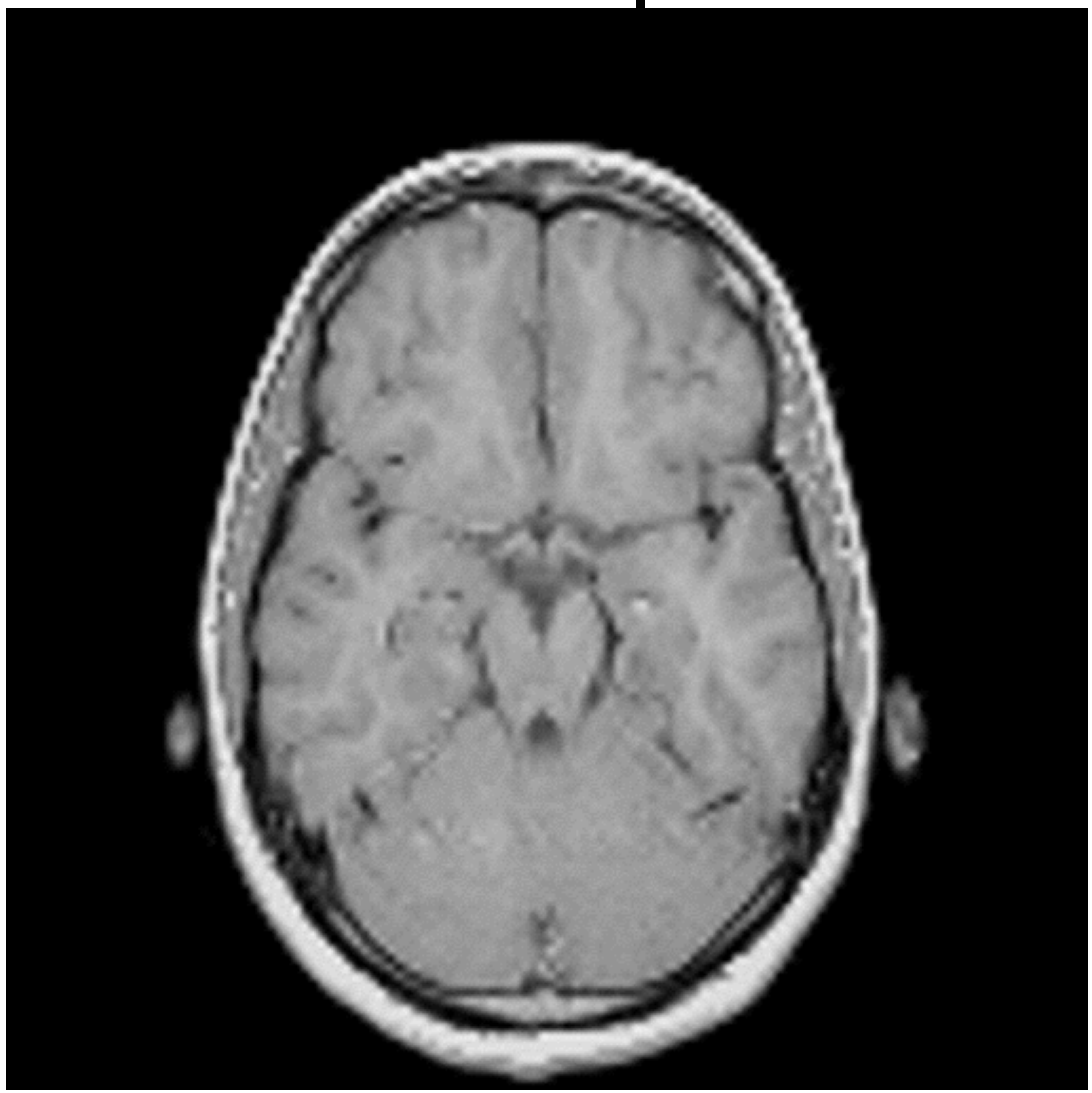


Laplacian Filter Bonus

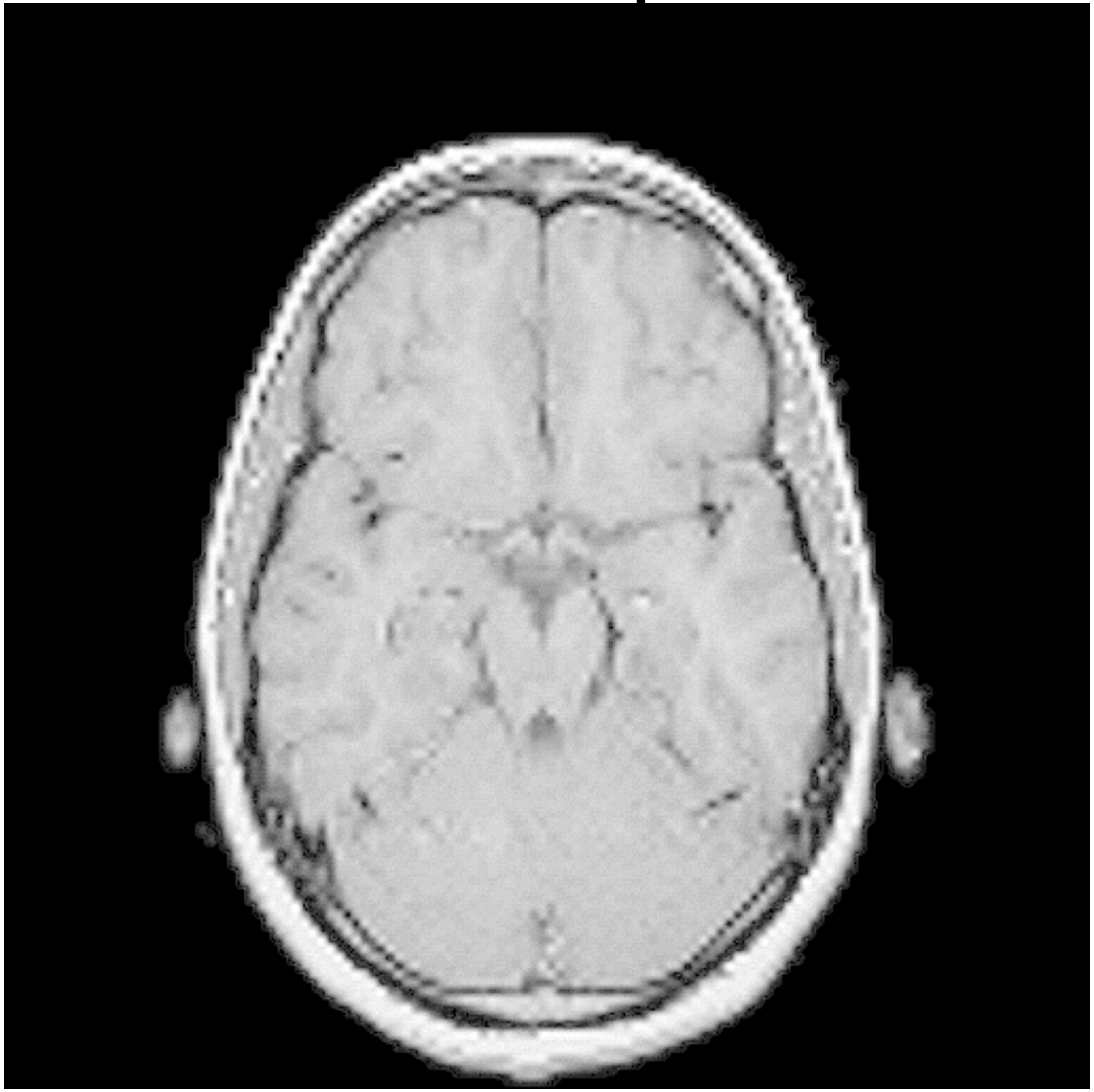
- With a small change (+1) we can increase sharpness in the image:

$$g_{\text{sharp}} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Laplacian Filter Bonus Example



Laplacian Filter Bonus Example



Box Filter

- This is a very simple filter low-pass filter:

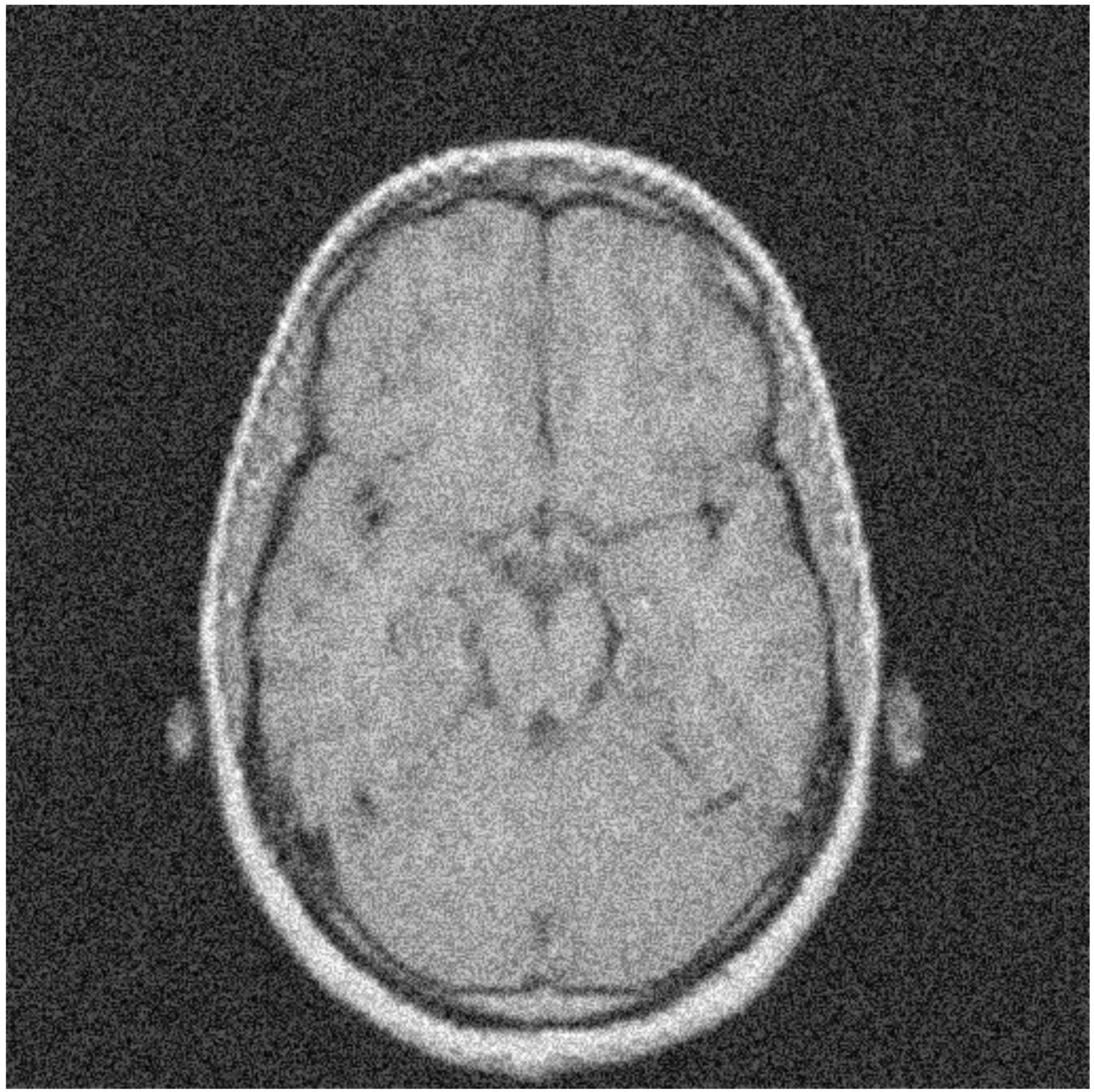
$$g[k,l] = 1 \quad \forall k \wedge \forall l$$

- What does it do? It blurs the signal!

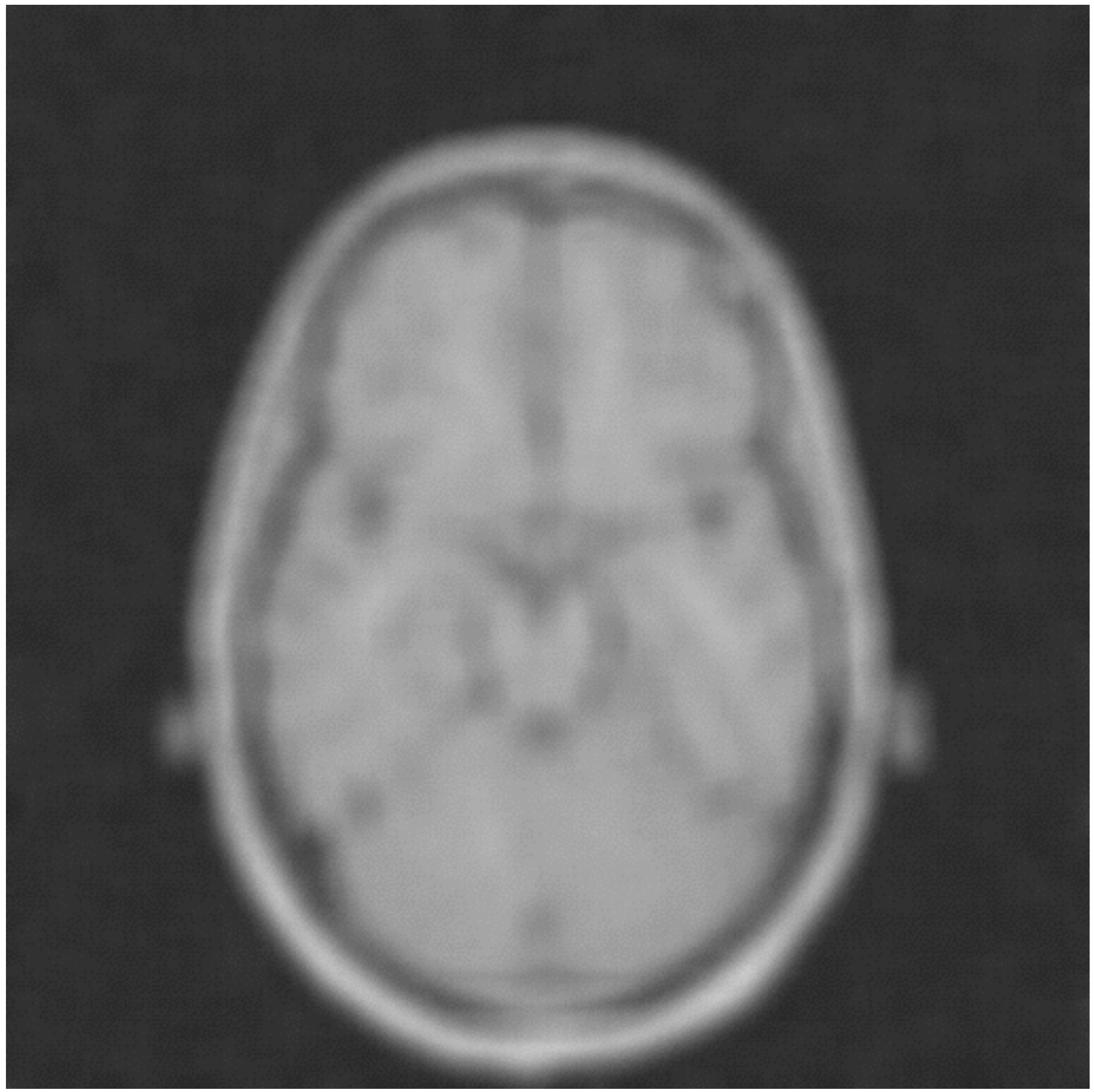
- This kernel has to be normalized:

$$g[k,l] = \frac{g[k,l]}{\sum_{k=-N}^{N} \sum_{l=-M}^{M} g[k,l]}$$

Box Filter Example



Box Filter Example



Gaussian Filter

- We use a Gaussian kernel defined as

$$g[k,l] = G\left(\sqrt{k^2 + l^2}\right)$$

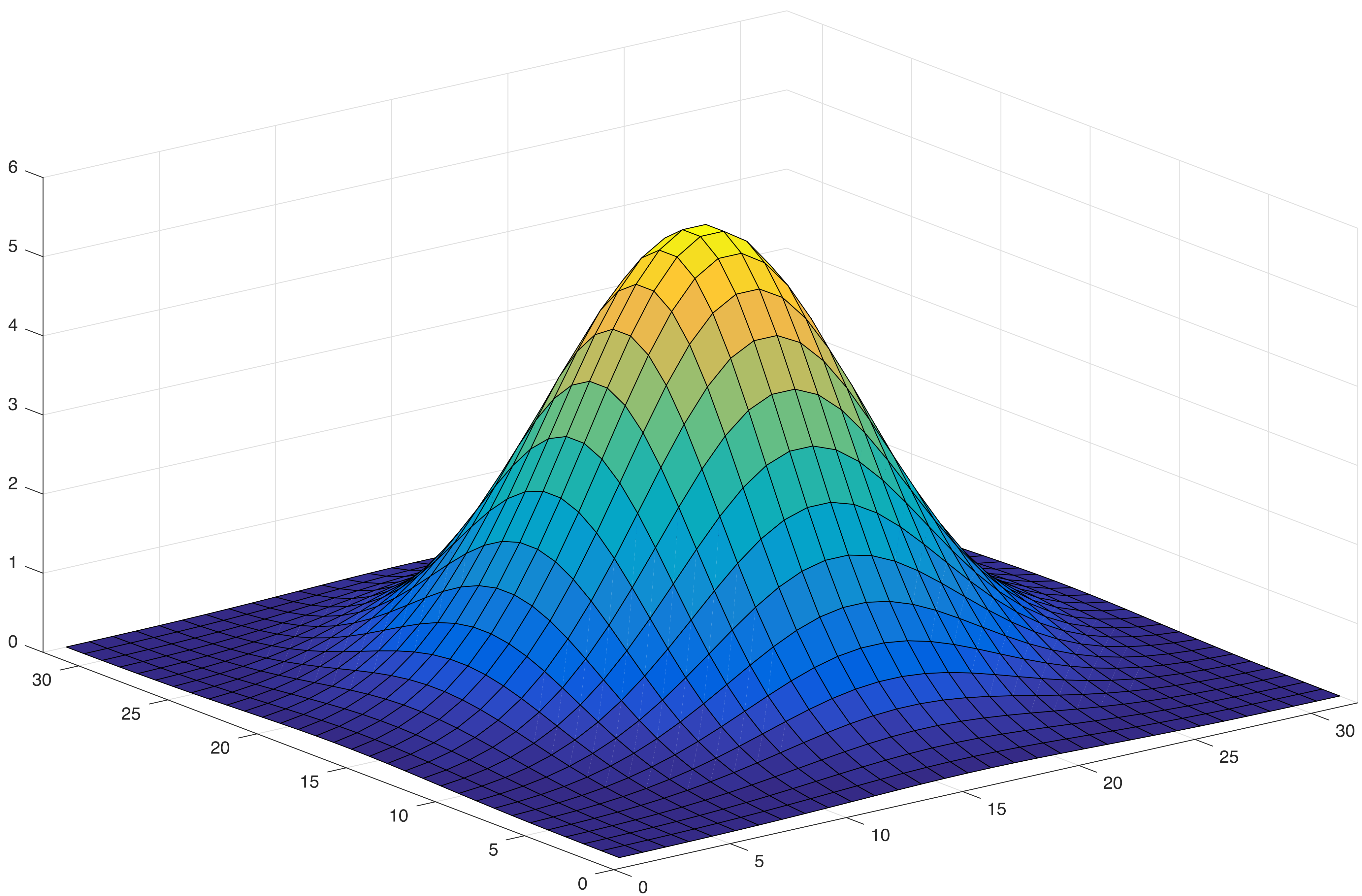
- where G is:

$$G(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

- Note that g has to be normalized:

$$g[k,l] = \frac{g[k,l]}{\sum_{k=-N}^{N} \sum_{l=-M}^{M} g[k,l]}$$

Gaussian Filter



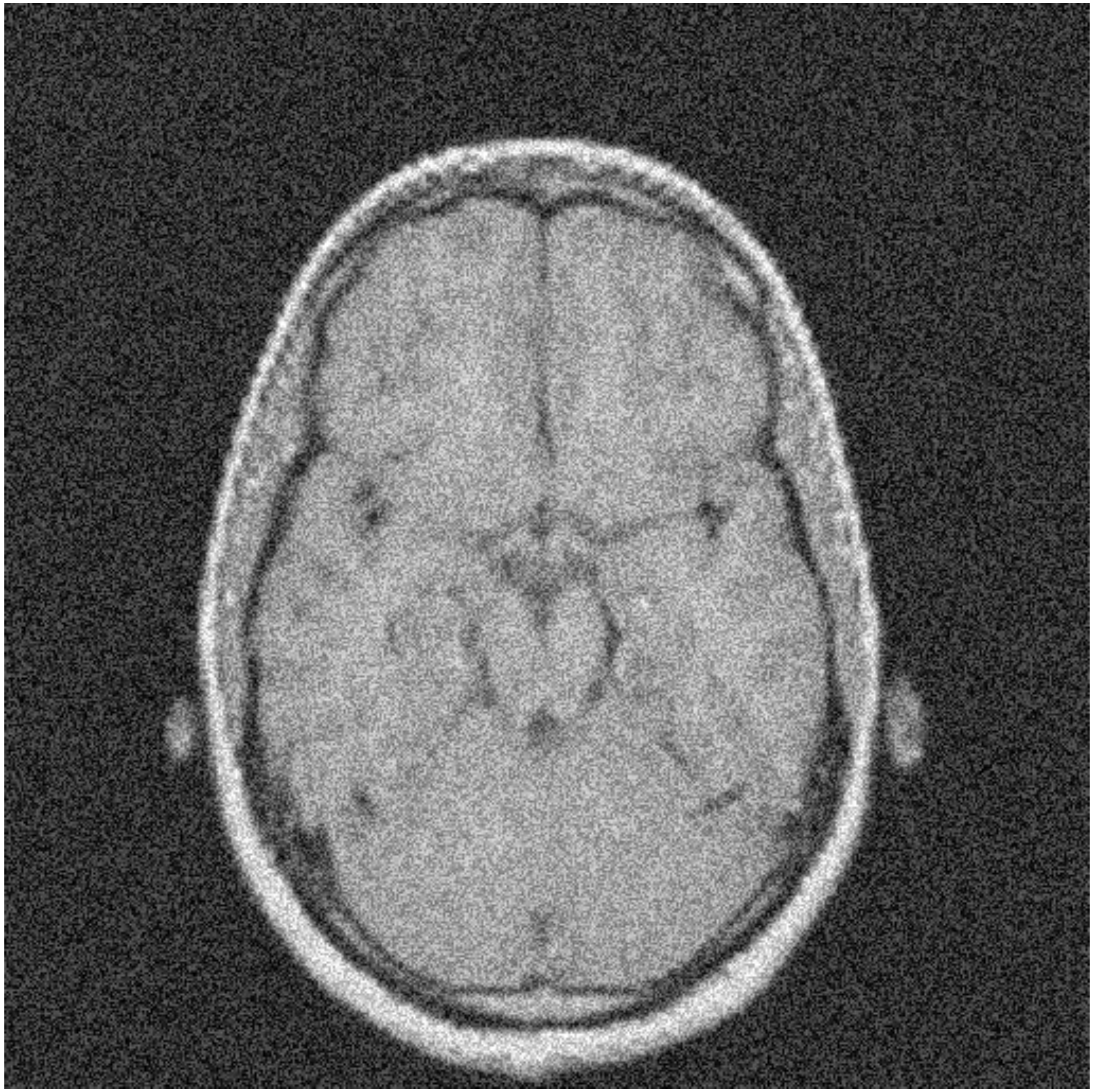
Gaussian Filter: how large?

- Typically, we have N = M;
- N and M depends on the sigma parameter:

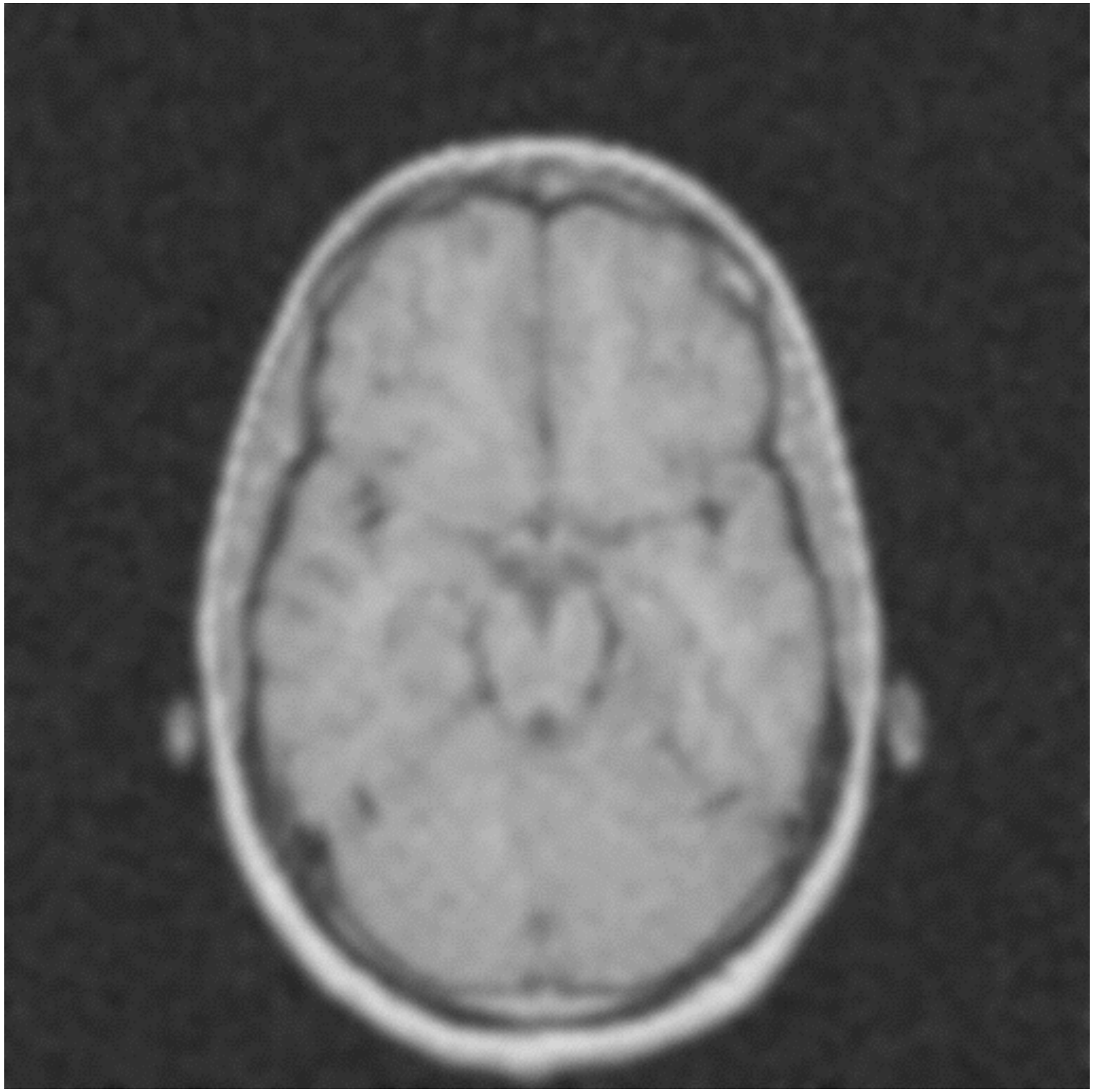
$$N = M = \frac{5}{2} \cdot \sigma \longrightarrow \text{98\% of energy}$$

- Larger sigma the better but the slower!
- Note: when sigma is too large (e.g., more than 128 pixels) it is better to work in the Fourier domain!

Gaussian Filter Example



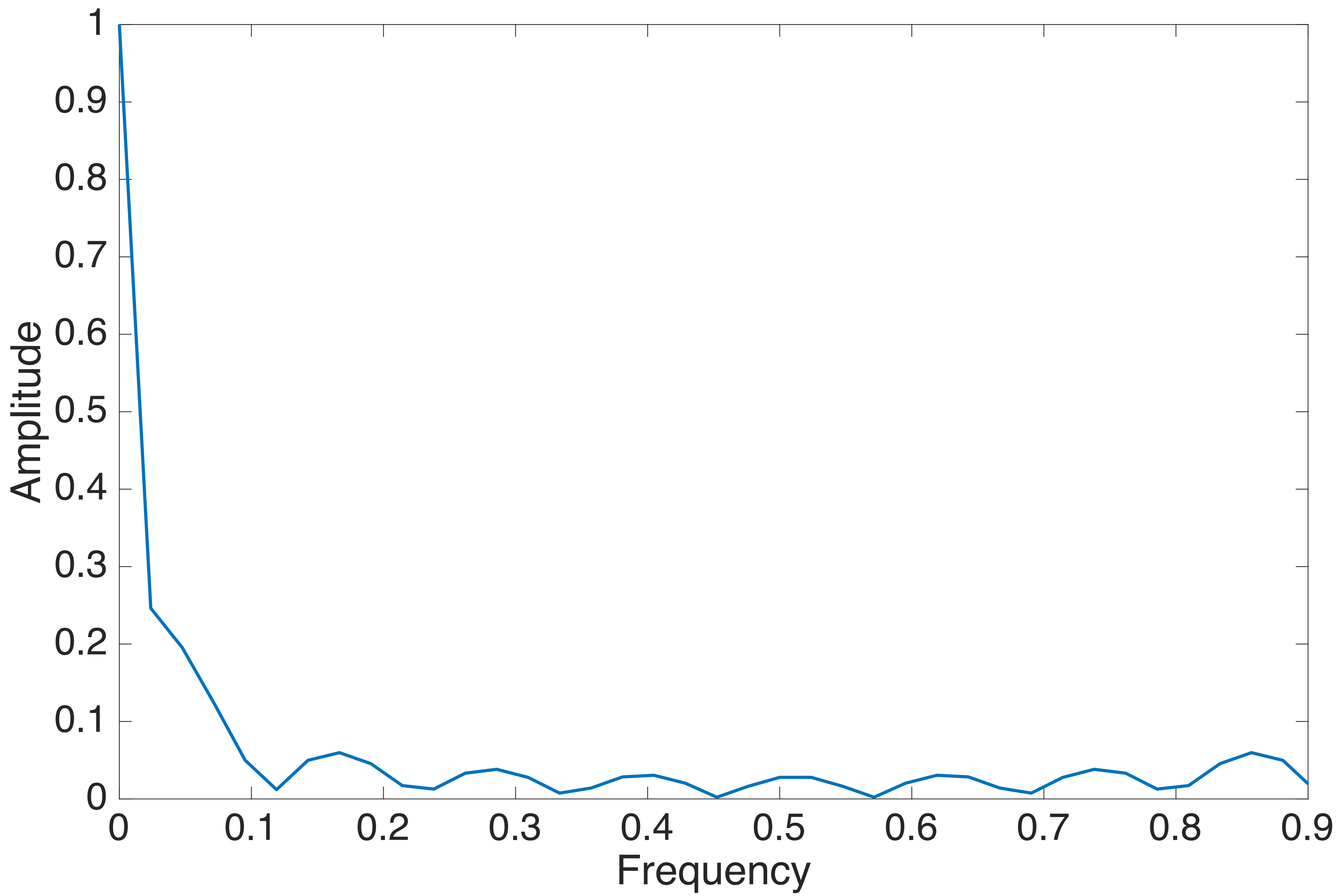
Gaussian Filter Example



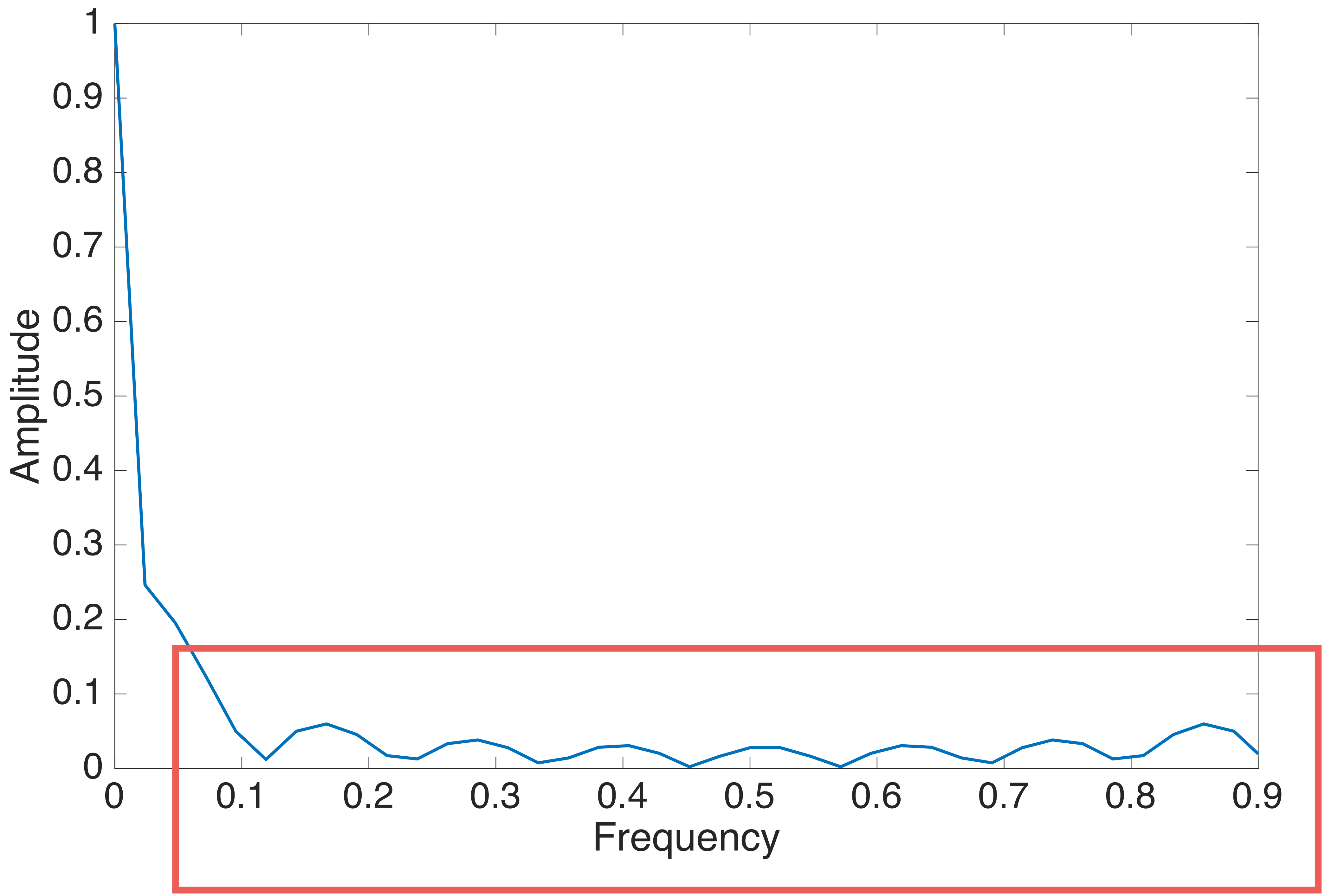
Box vs Gaussian

- As you probably know…
- The box filter cuts primarily high frequencies but it has oscillations for some low frequencies.
  - What does it mean? **That is BAD!**
- The Gaussian filter cuts mostly high frequencies!
  - **That is GOOD!**

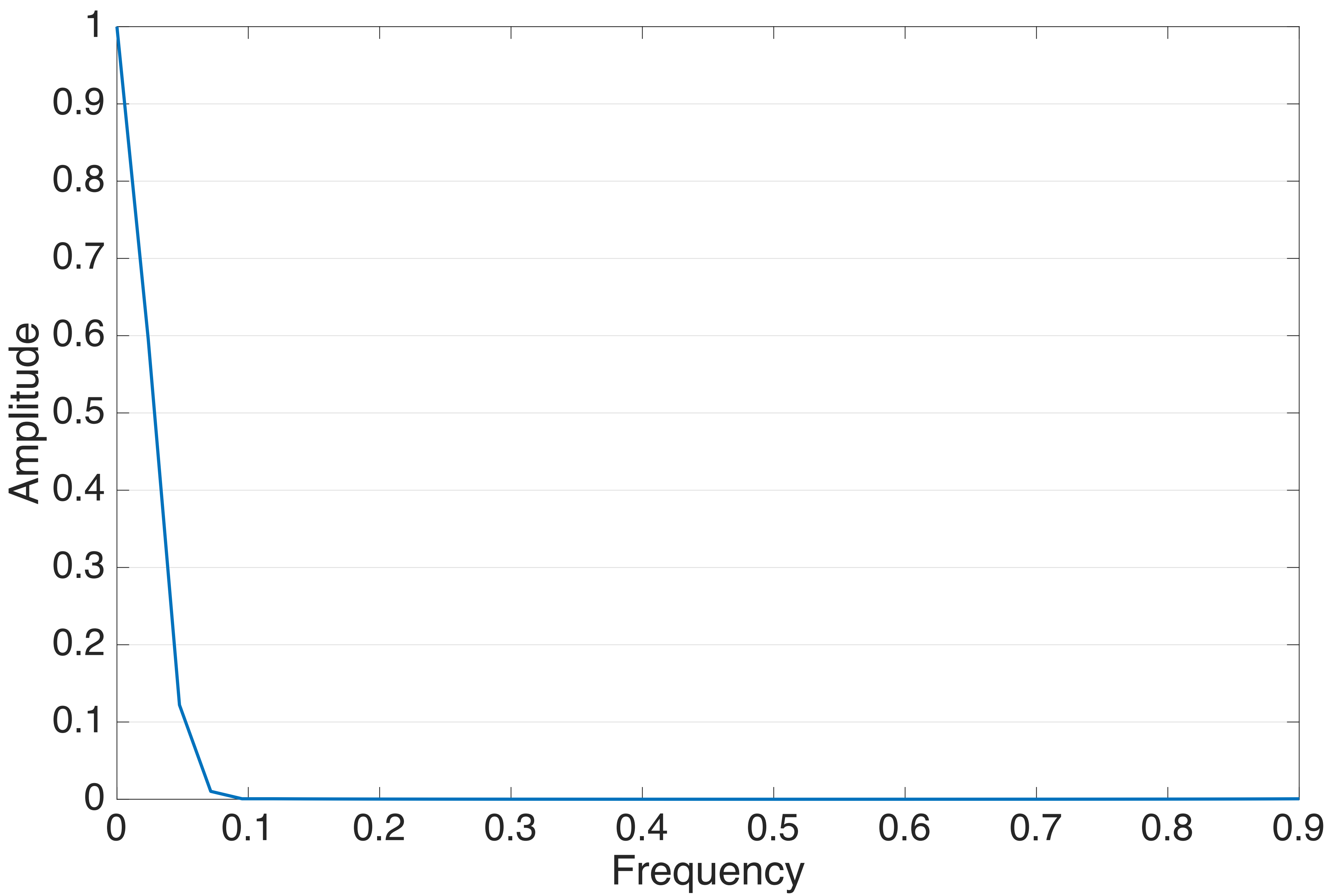
Box vs Gaussian Example



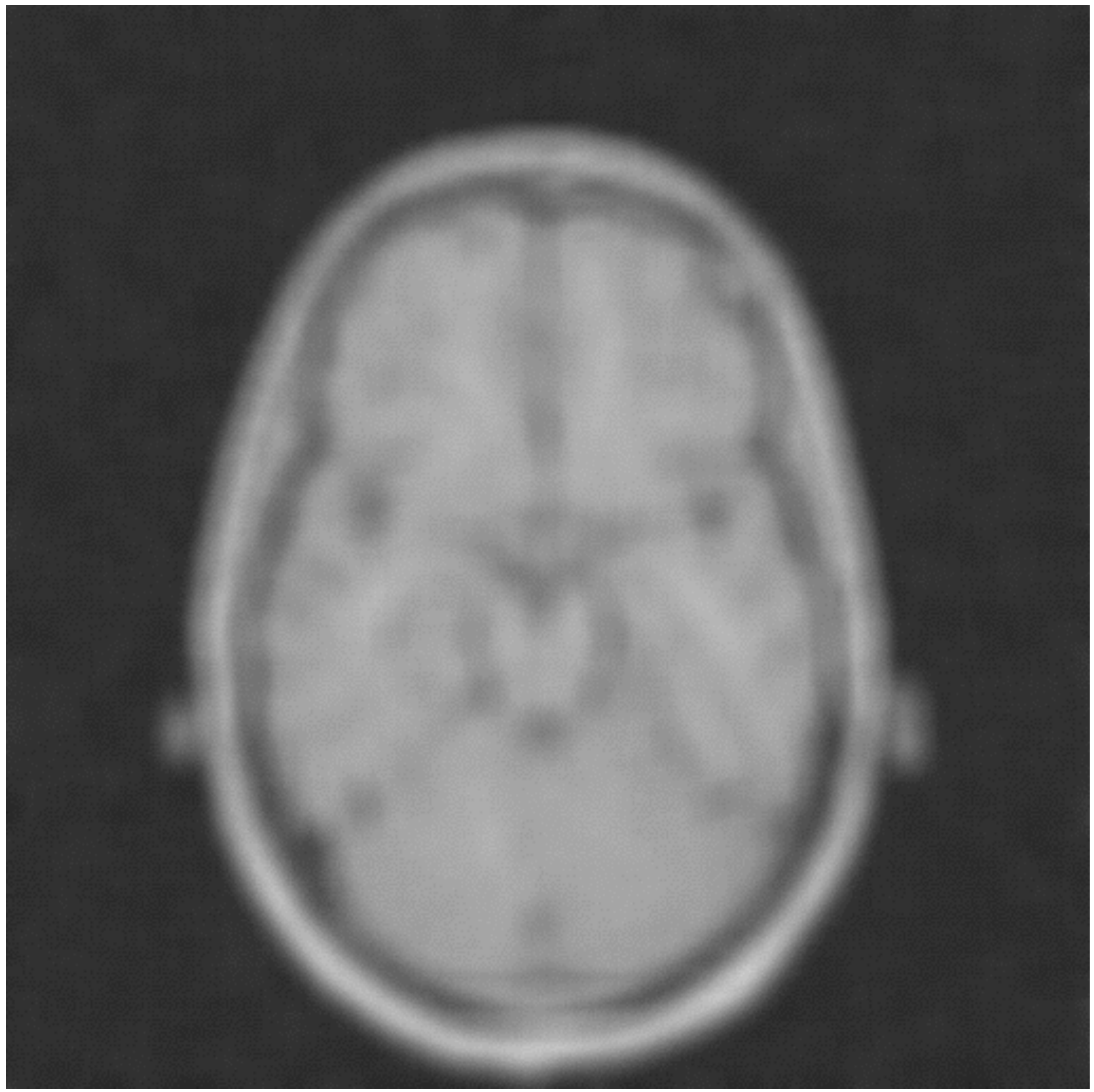
Box vs Gaussian Example



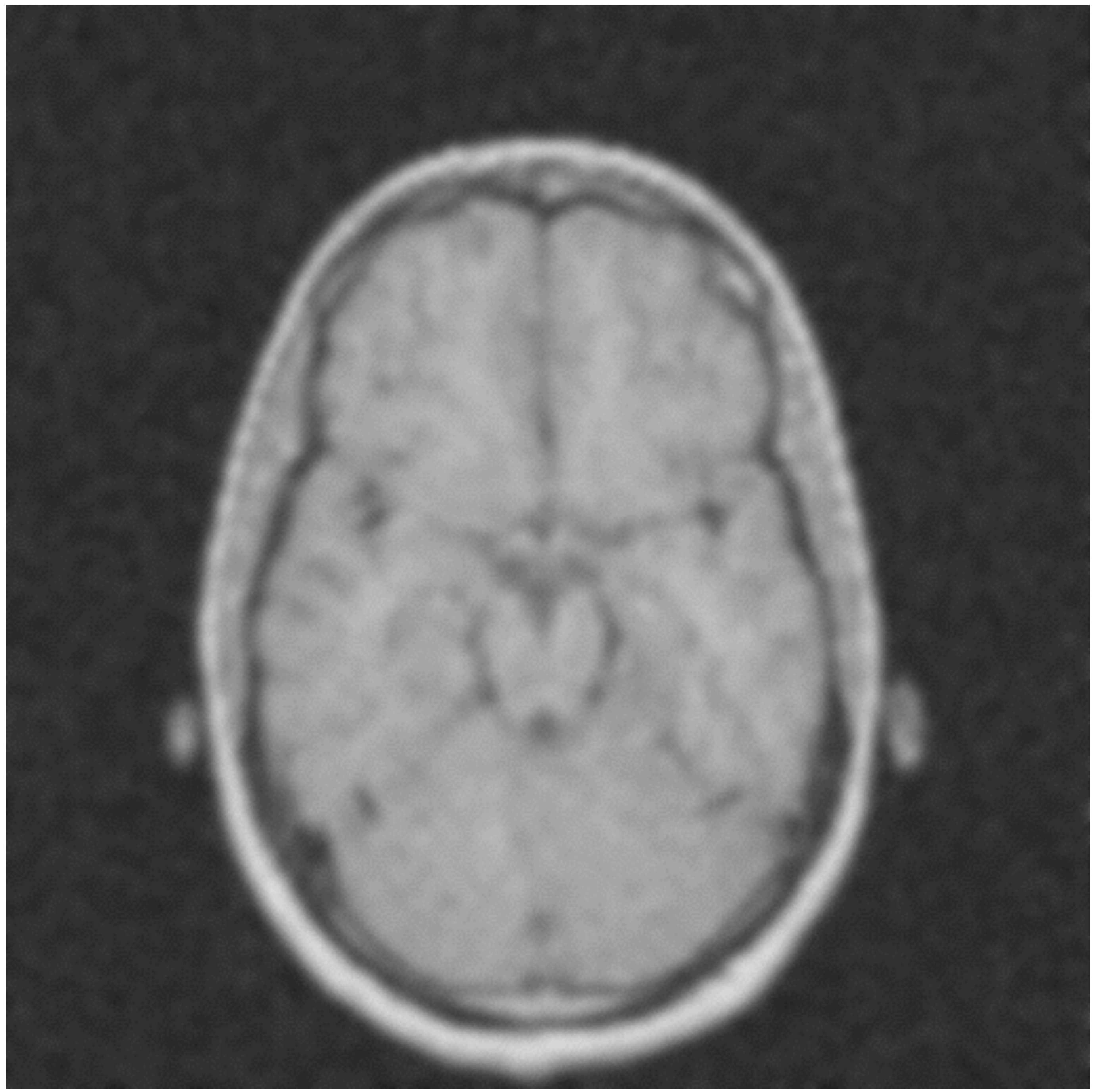
Box vs Gaussian Example



Box vs Gaussian Example

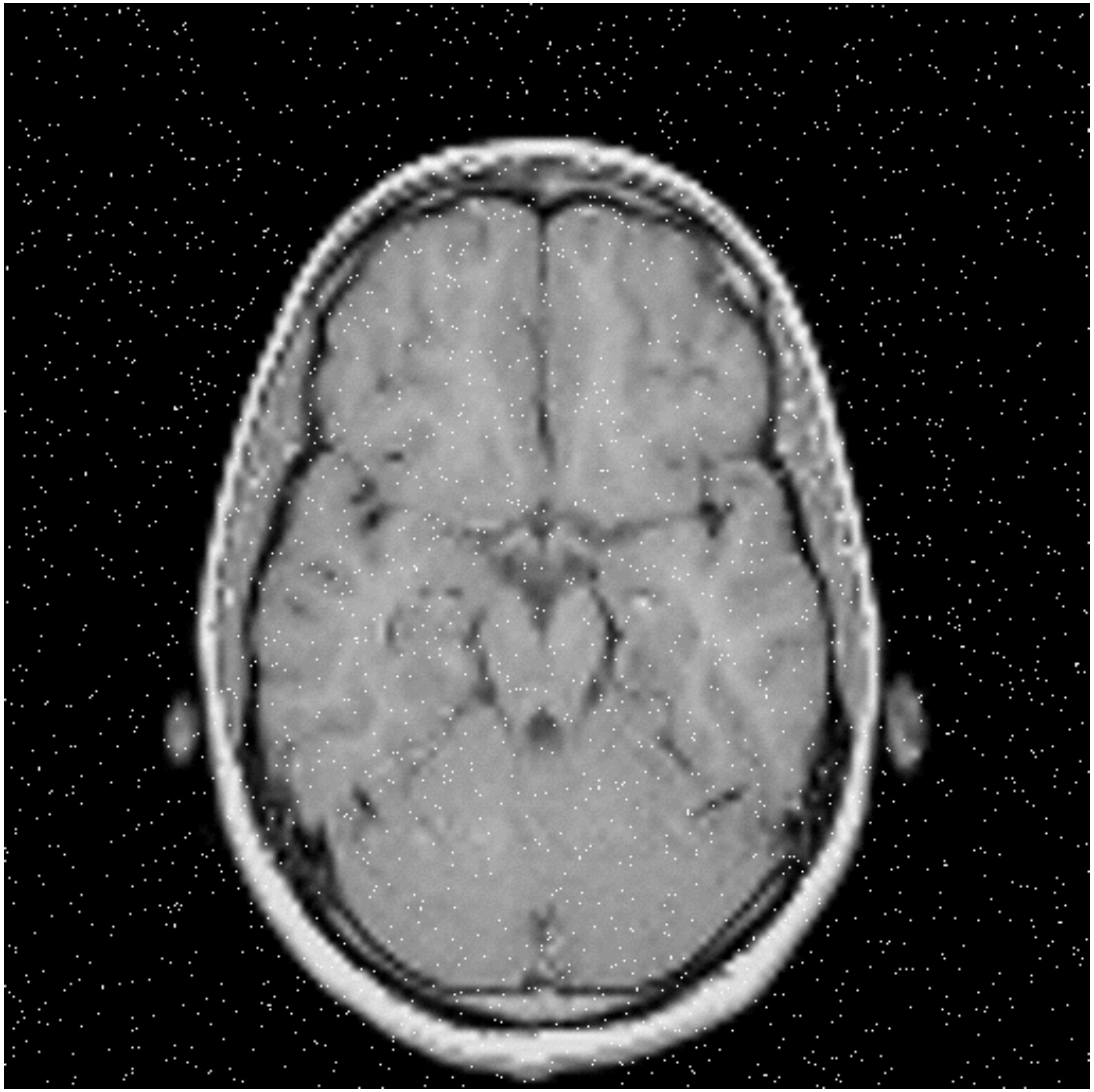


Box vs Gaussian Example

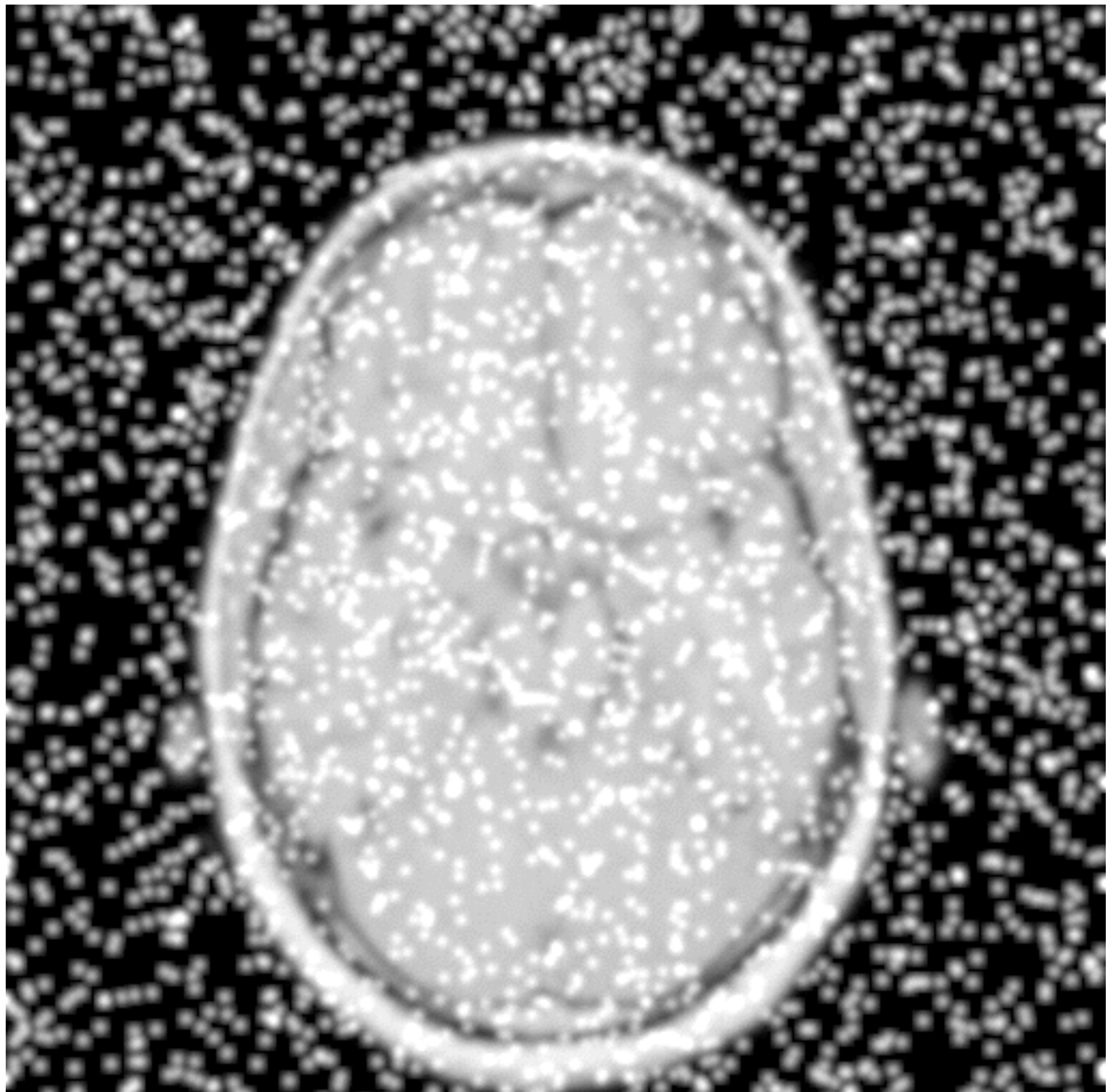


Non-Linear Filters

Salt and Pepper Noise



Salt and Pepper Noise

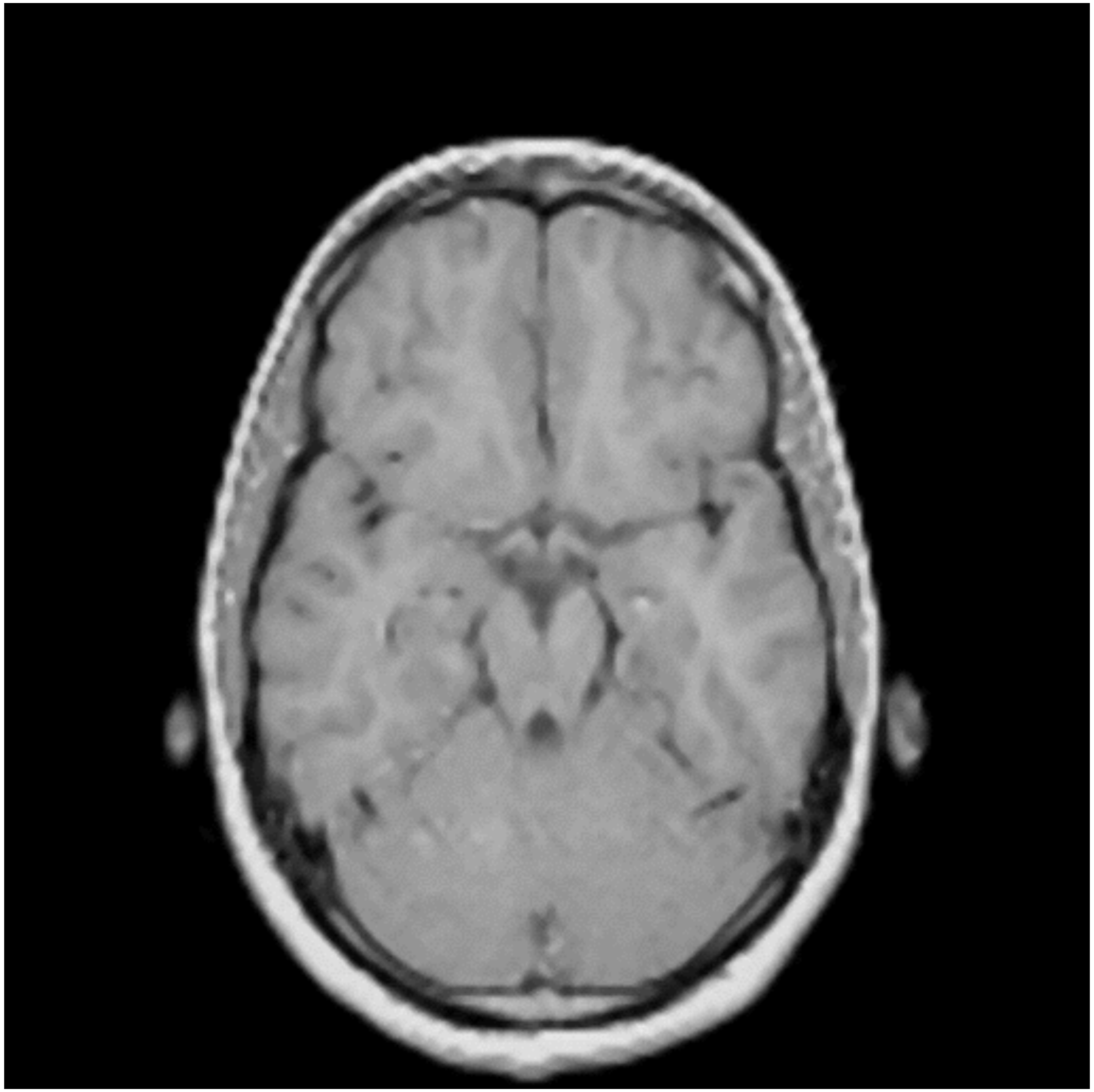


Median Filter

- This filter is mostly meant for tackling salt-and-pepper noise!
    - Linear filters do a mess with salt-and-pepper!
- It exploits the fact that median is robust in separating the higher half of data sample from the lower part! Classist isn't it?

Median Filter Example

Median Filter Example



The Bilateral Filter

- It is a non-linear filter, oh really?
- It works both spatial domain and intensity/range domain of the image.
- Basically, it is an adaptive linear filter:
  - It behaves as a linear filter in flat regions;
  - At strong edges (step-edge), filtering is “limited”.

The Bilateral Filter

$$BF[I](\mathbf{x}, f_s, g_r) = \frac{1}{K(\mathbf{x}, f_s, g_r)} \sum_{\mathbf{y} \in \Omega(\mathbf{x})} I(\mathbf{y}) f_s(\|\mathbf{x} - \mathbf{y}\|) g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

$$K[I](\mathbf{x}, f_s, g_r) = \sum_{\mathbf{y} \in \Omega(\mathbf{x})} f_s(\|\mathbf{x} - \mathbf{y}\|) g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

The Bilateral Filter

Spatial Function

$$BF[I](\mathbf{x}, f_s, g_r) = \frac{1}{K(\mathbf{x}, f_s, g_r)} \sum_{\mathbf{y} \in \Omega(\mathbf{x})} I(\mathbf{y}) \boxed{f_s(\|\mathbf{x} - \mathbf{y}\|)} g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

$$K[I](\mathbf{x}, f_s, g_r) = \sum_{\mathbf{y} \in \Omega(\mathbf{x})} f_s(\|\mathbf{x} - \mathbf{y}\|) g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

The Bilateral Filter

$$BF[I](\mathbf{x}, f_s, g_r) = \frac{1}{K(\mathbf{x}, f_s, g_r)} \sum_{\mathbf{y} \in \Omega(\mathbf{x})} I(\mathbf{y}) f_s(\|\mathbf{x} - \mathbf{y}\|) g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

$$K[I](\mathbf{x}, f_s, g_r) = \sum_{\mathbf{y} \in \Omega(\mathbf{x})} f_s(\|\mathbf{x} - \mathbf{y}\|) g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

The Bilateral Filter

Range Function

$$BF[I](\mathbf{x}, f_s, g_r) = \frac{1}{K(\mathbf{x}, f_s, g_r)} \sum_{\mathbf{y} \in \Omega(\mathbf{x})} I(\mathbf{y}) f_s(\|\mathbf{x} - \mathbf{y}\|) \boxed{g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|)},$$

$$K[I](\mathbf{x}, f_s, g_r) = \sum_{\mathbf{y} \in \Omega(\mathbf{x})} f_s(\|\mathbf{x} - \mathbf{y}\|) g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

The Bilateral Filter

$$BF[I](\mathbf{x}, f_s, g_r) = \frac{1}{K(\mathbf{x}, f_s, g_r)} \sum_{\mathbf{y} \in \Omega(\mathbf{x})} I(\mathbf{y}) f_s(\|\mathbf{x} - \mathbf{y}\|) g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

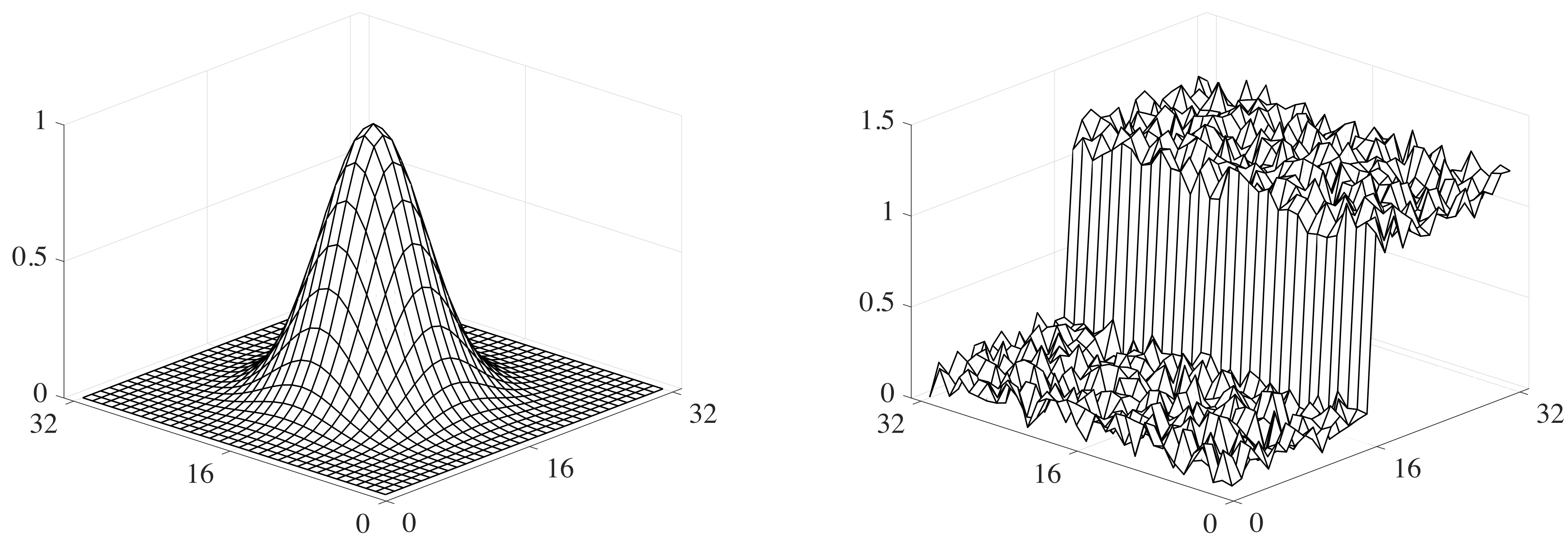
$$K[I](\mathbf{x}, f_s, g_r) = \sum_{\mathbf{y} \in \Omega(\mathbf{x})} f_s(\|\mathbf{x} - \mathbf{y}\|) g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

The Bilateral Filter

- Spatial function: a Gaussian function
- Range function: a Gaussian function
- How large is the kernel?
  - If the spatial function is a Gaussian:

$$N = M = \frac{5}{2}\sigma_s$$

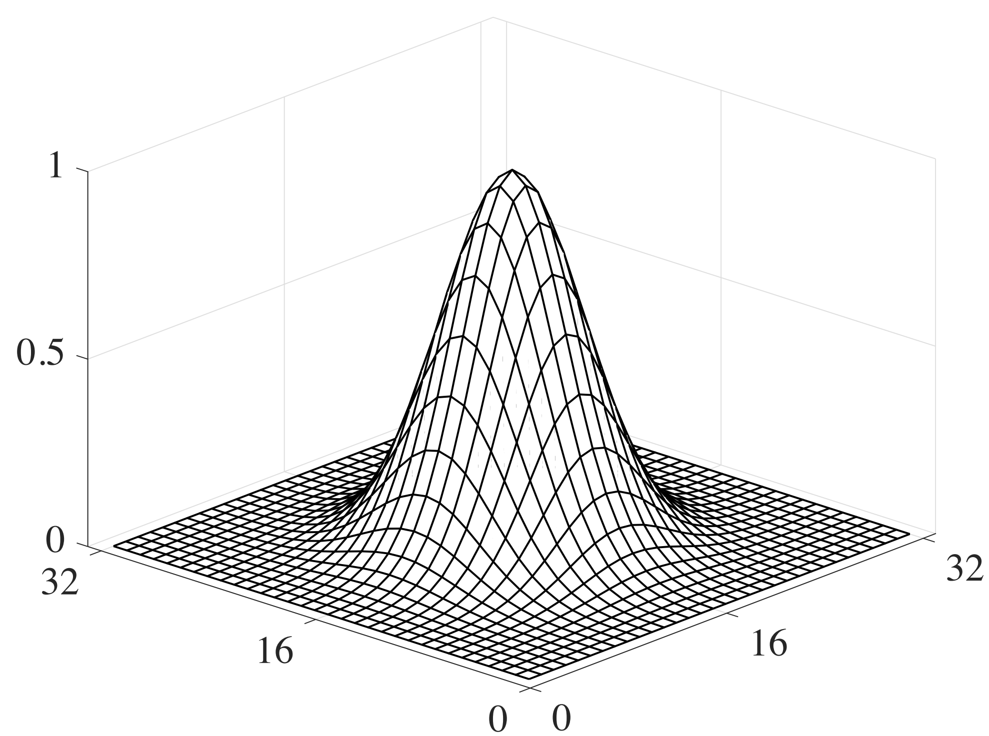
The Bilateral Filter Explained



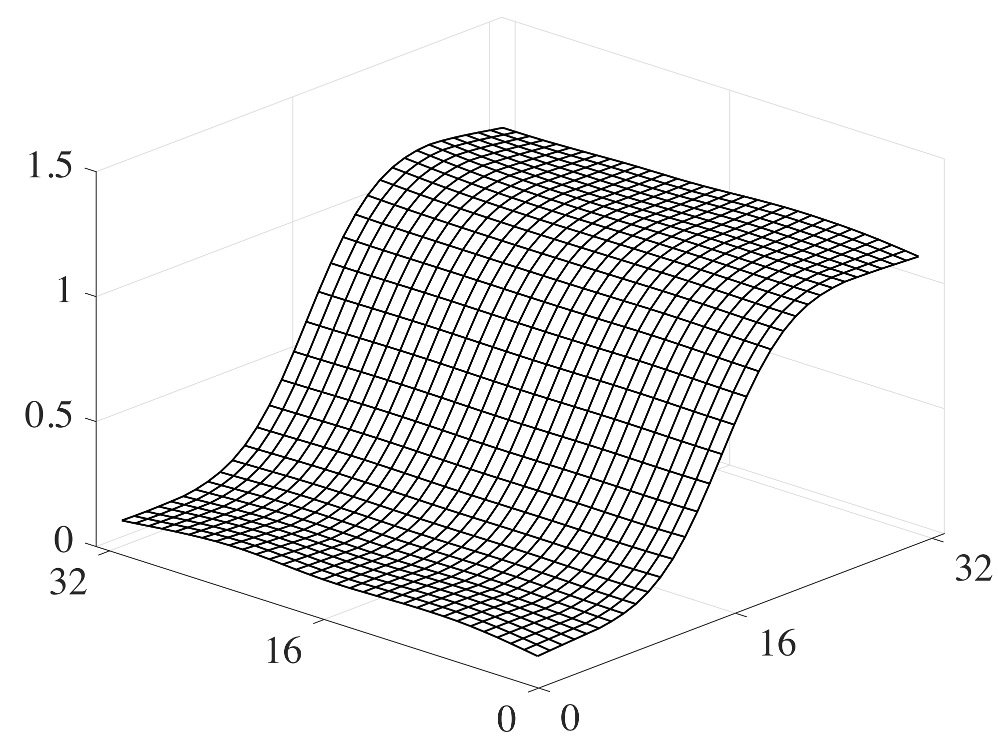
Kernel

Image

The Bilateral Filter Explained

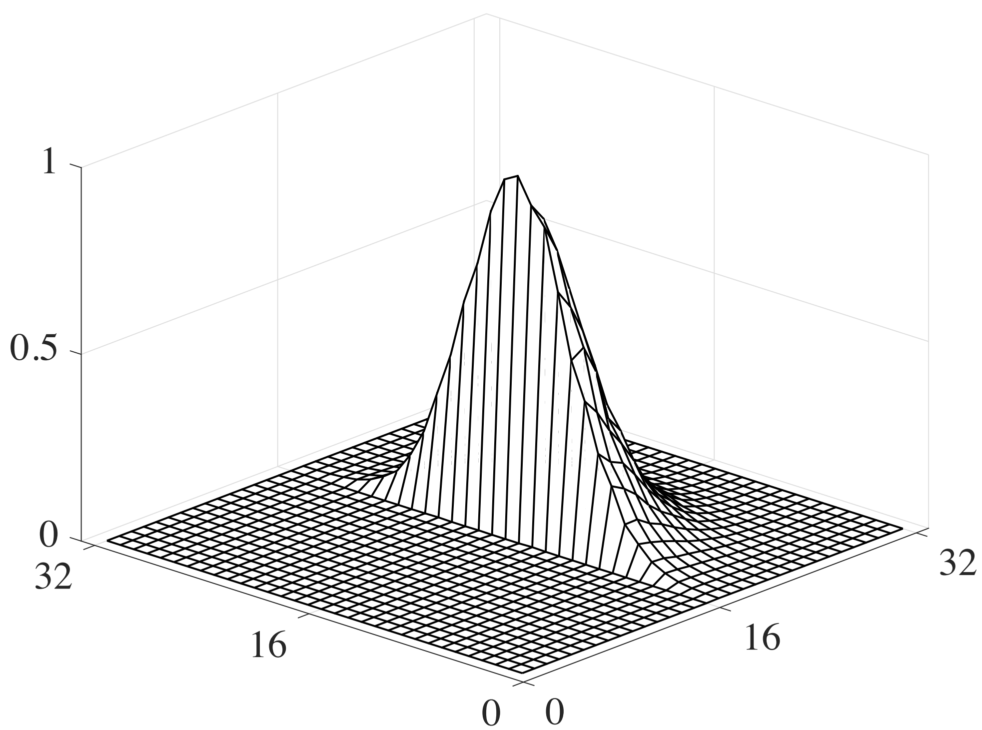


Kernel

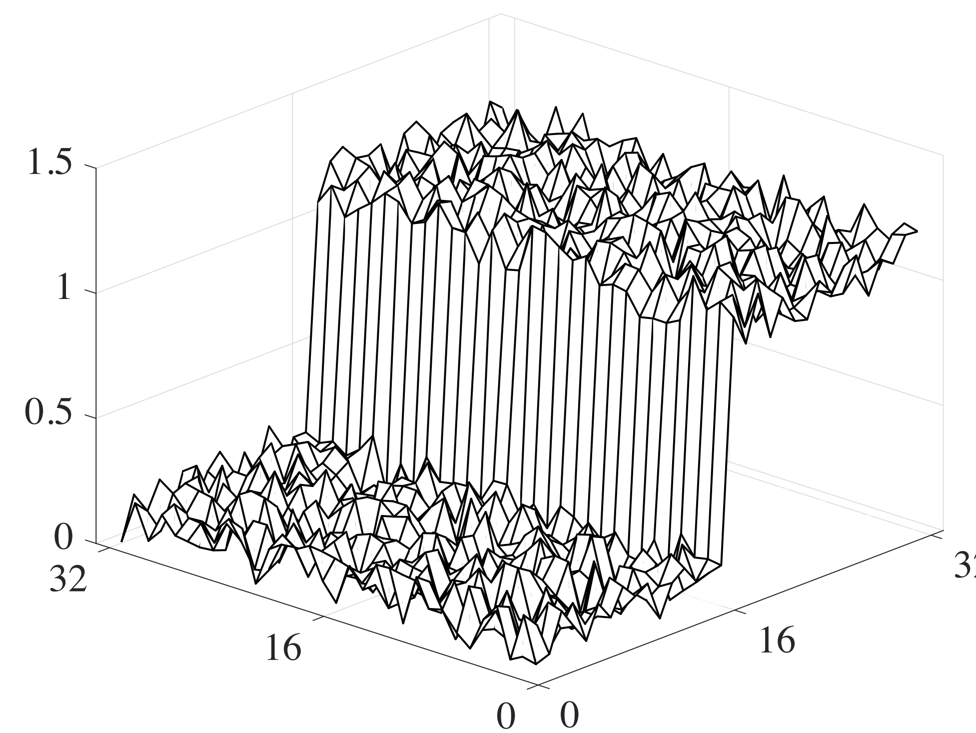


Image

The Bilateral Filter Explained

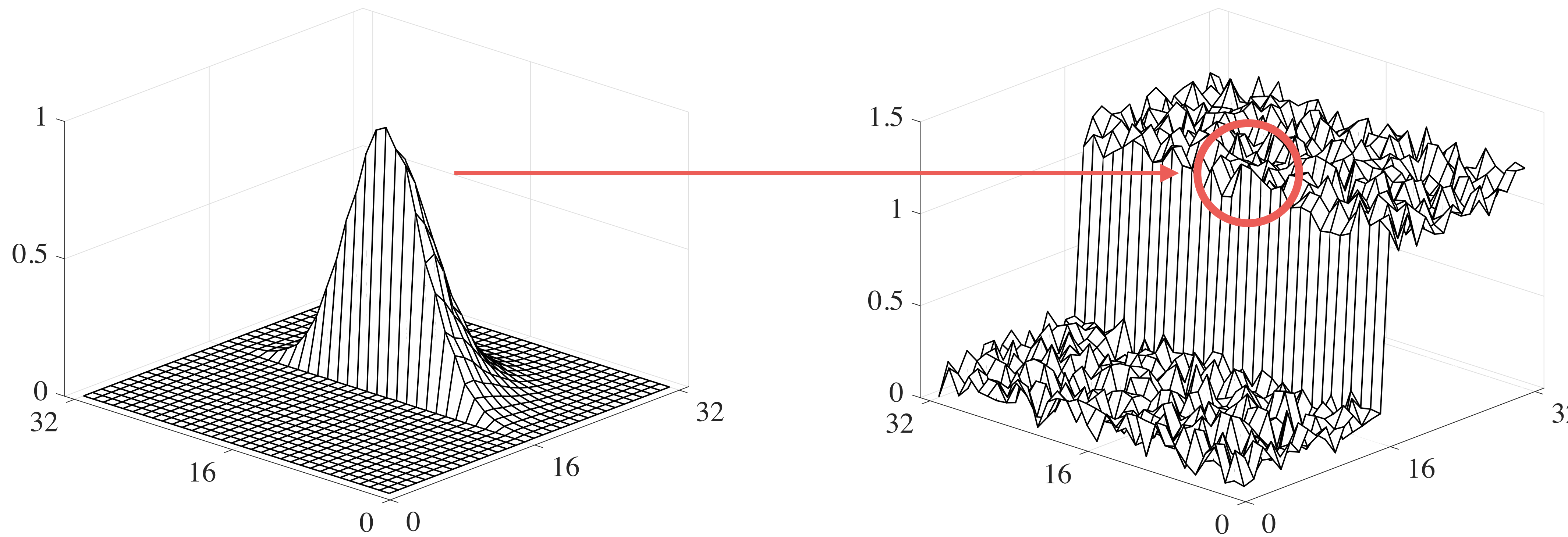


Kernel
(change for each pixel!!)



Image

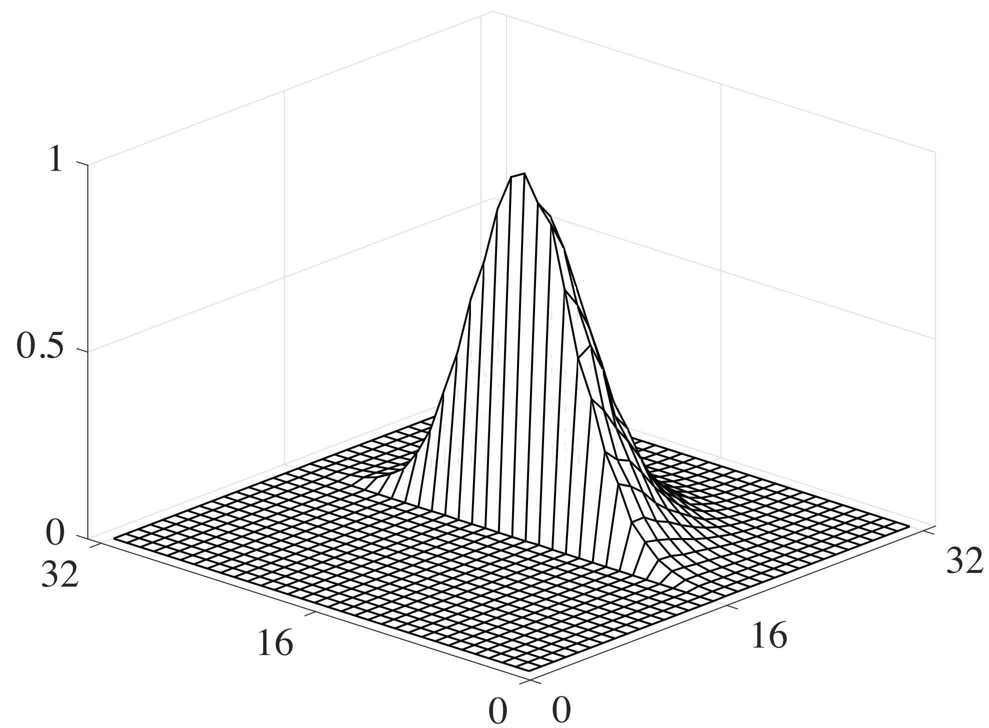
The Bilateral Filter Explained



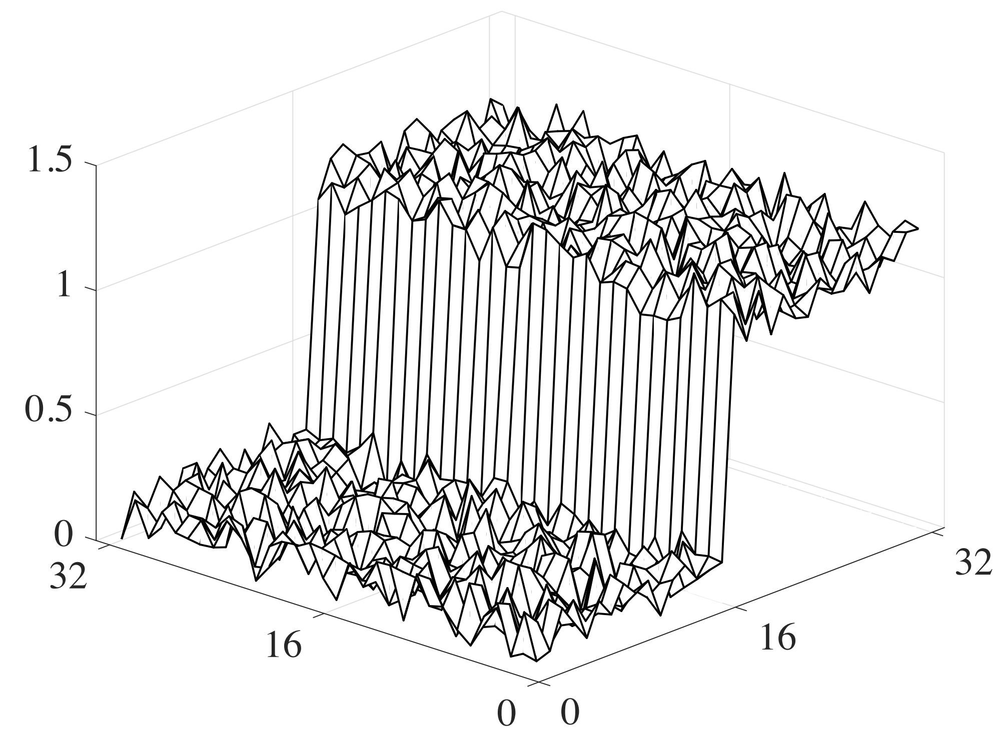
Kernel
(change for each pixel!!)

Image

The Bilateral Filter Explained

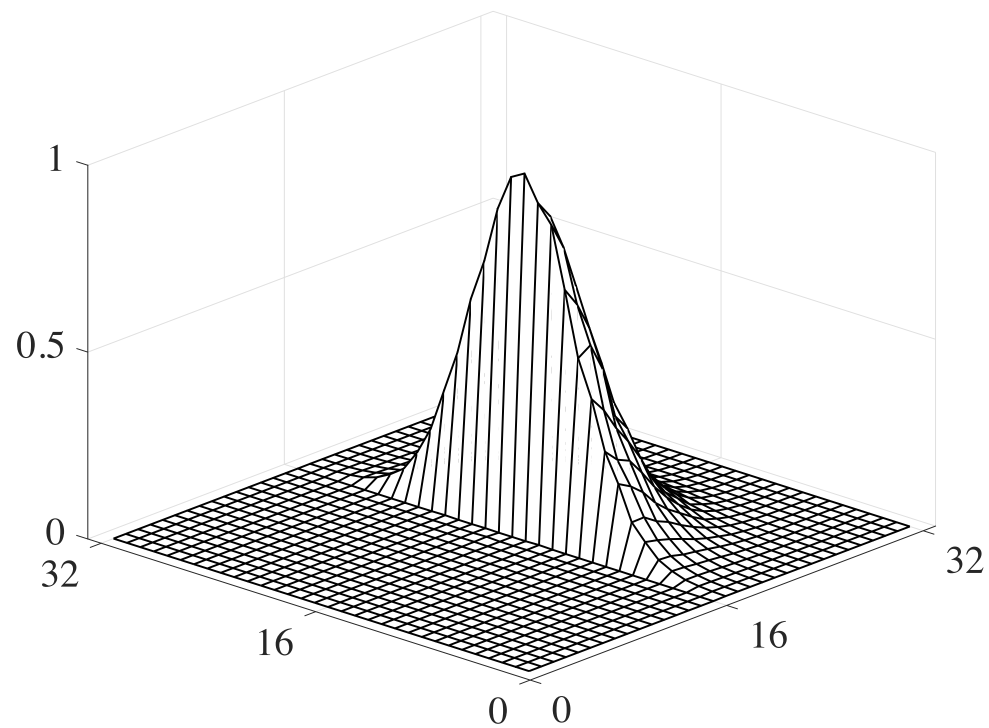


Kernel
(change for each pixel!!)

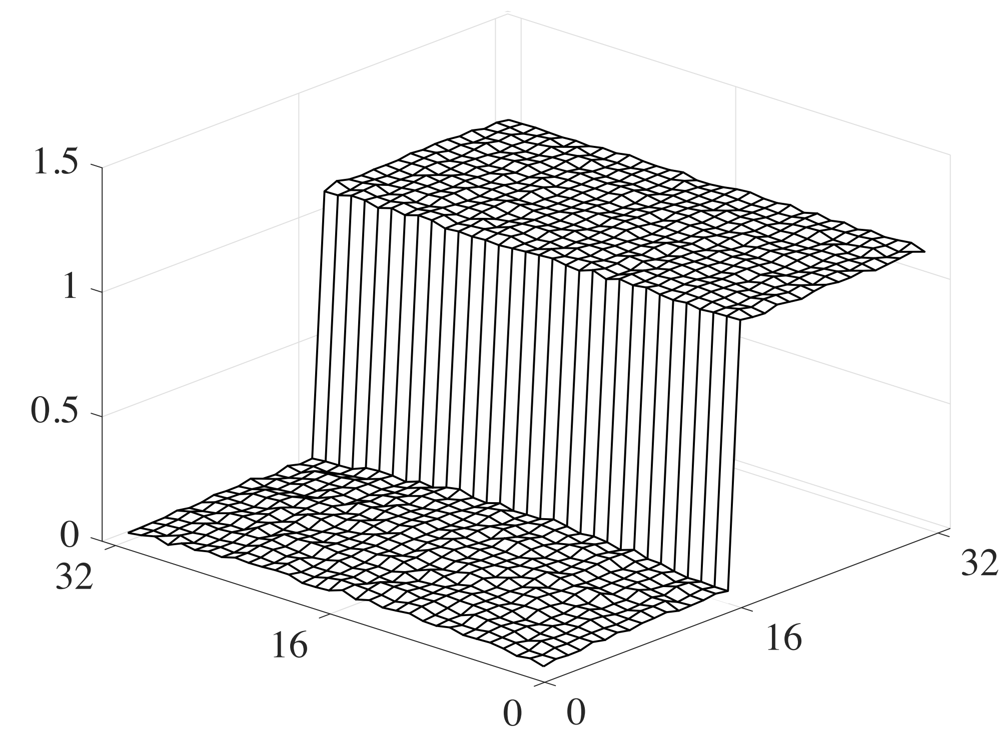


Image

The Bilateral Filter Explained

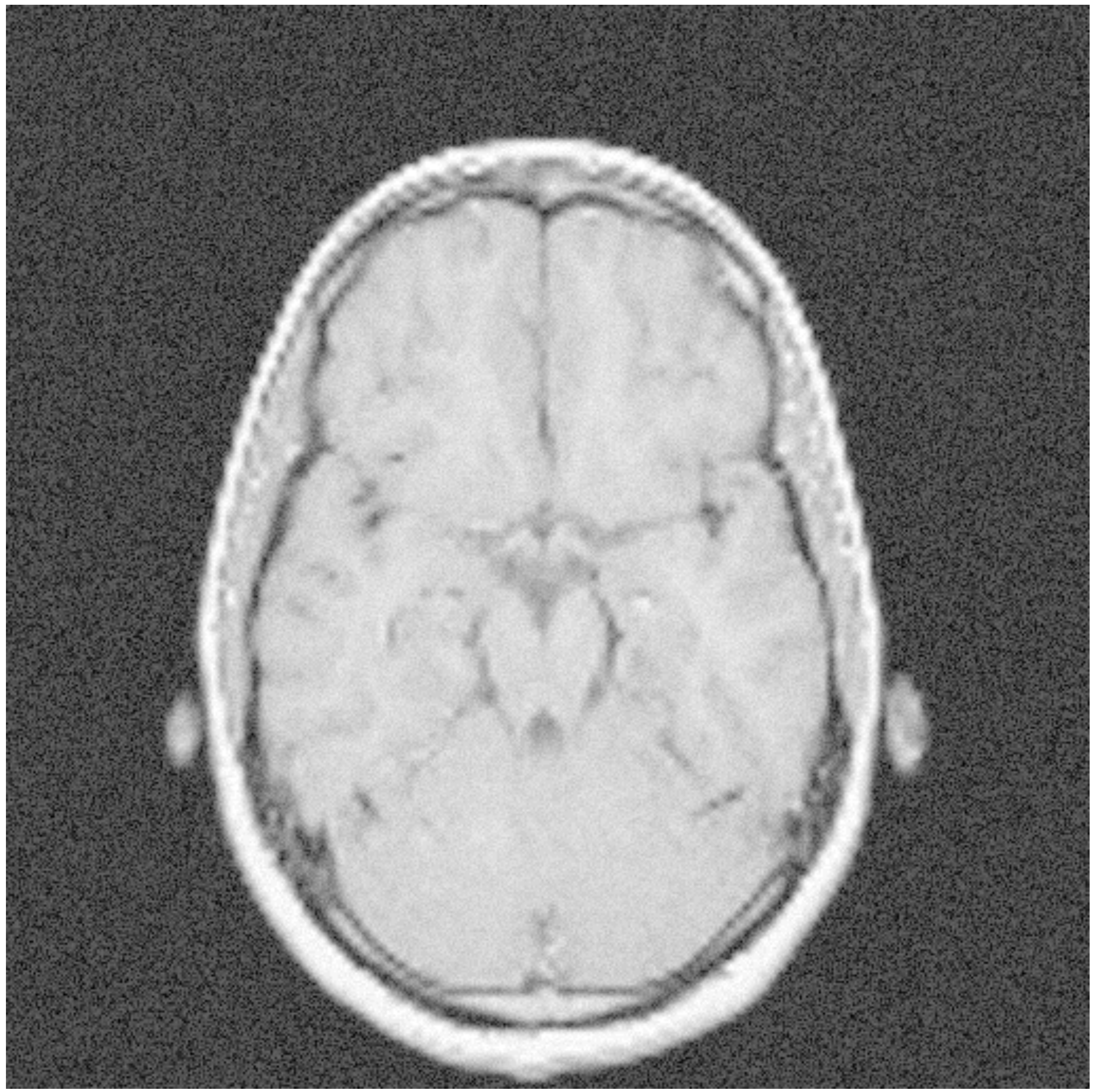


Kernel
(change for each pixel!!)

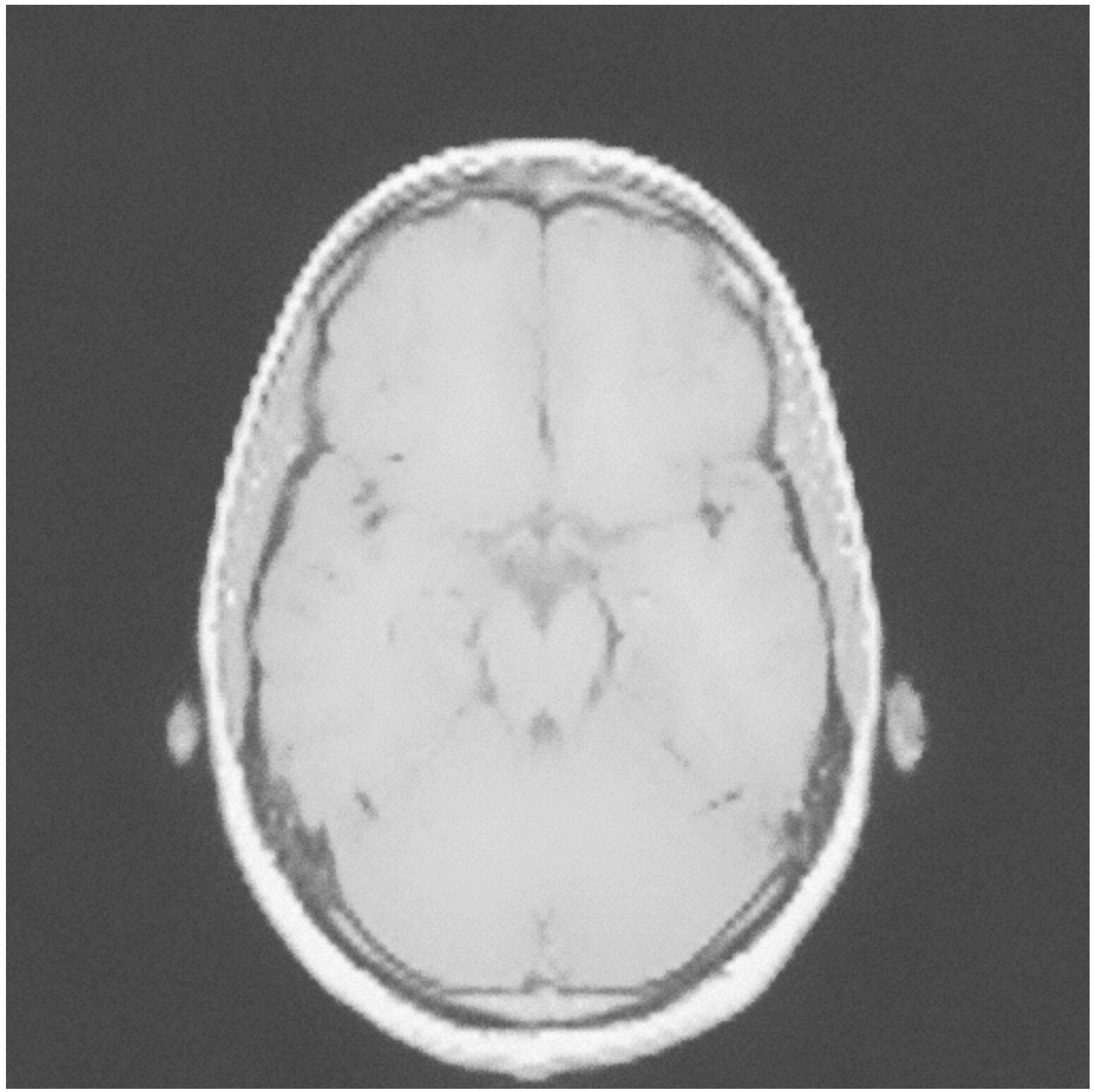


Image

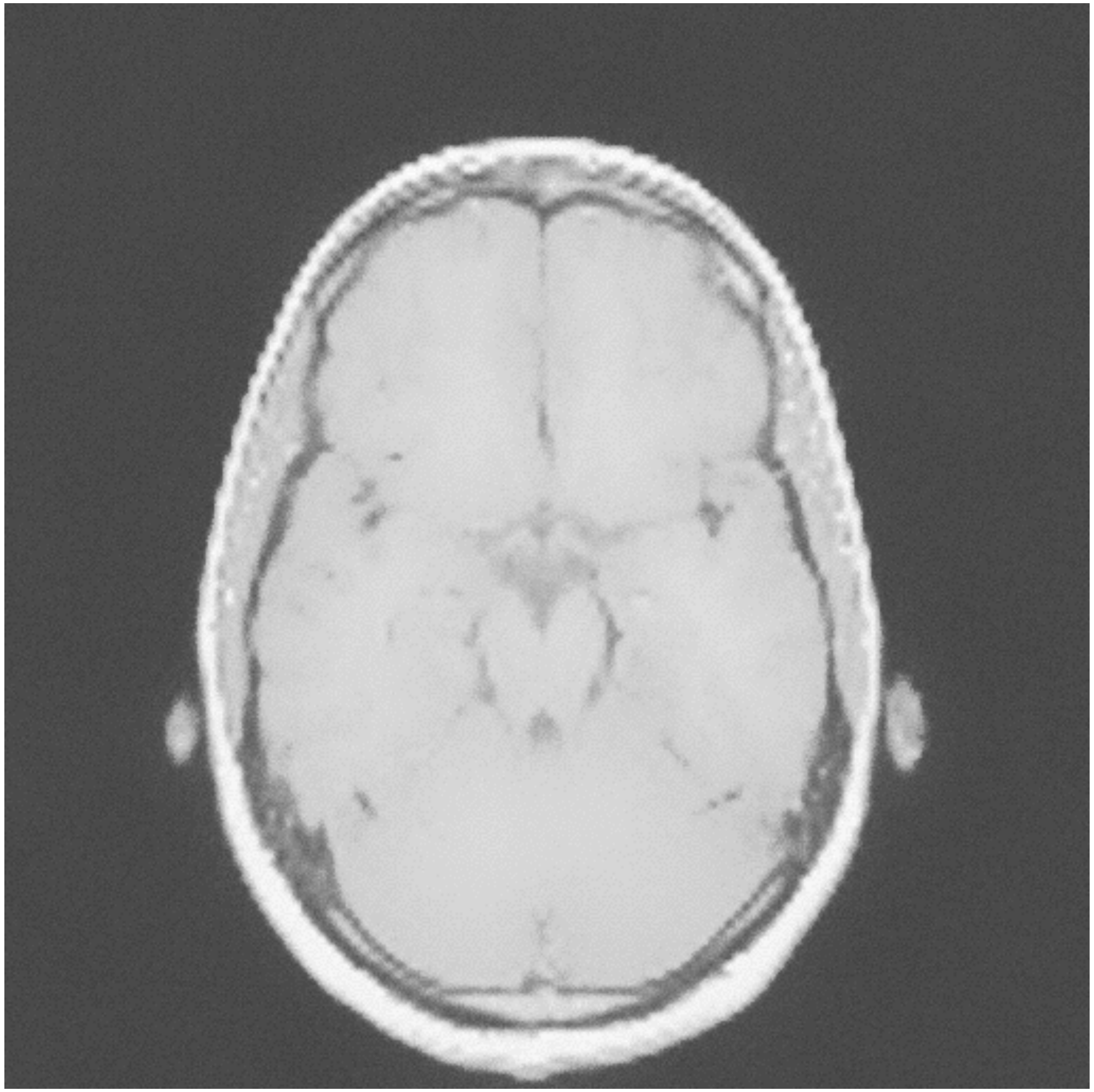
The Bilateral Filter Example



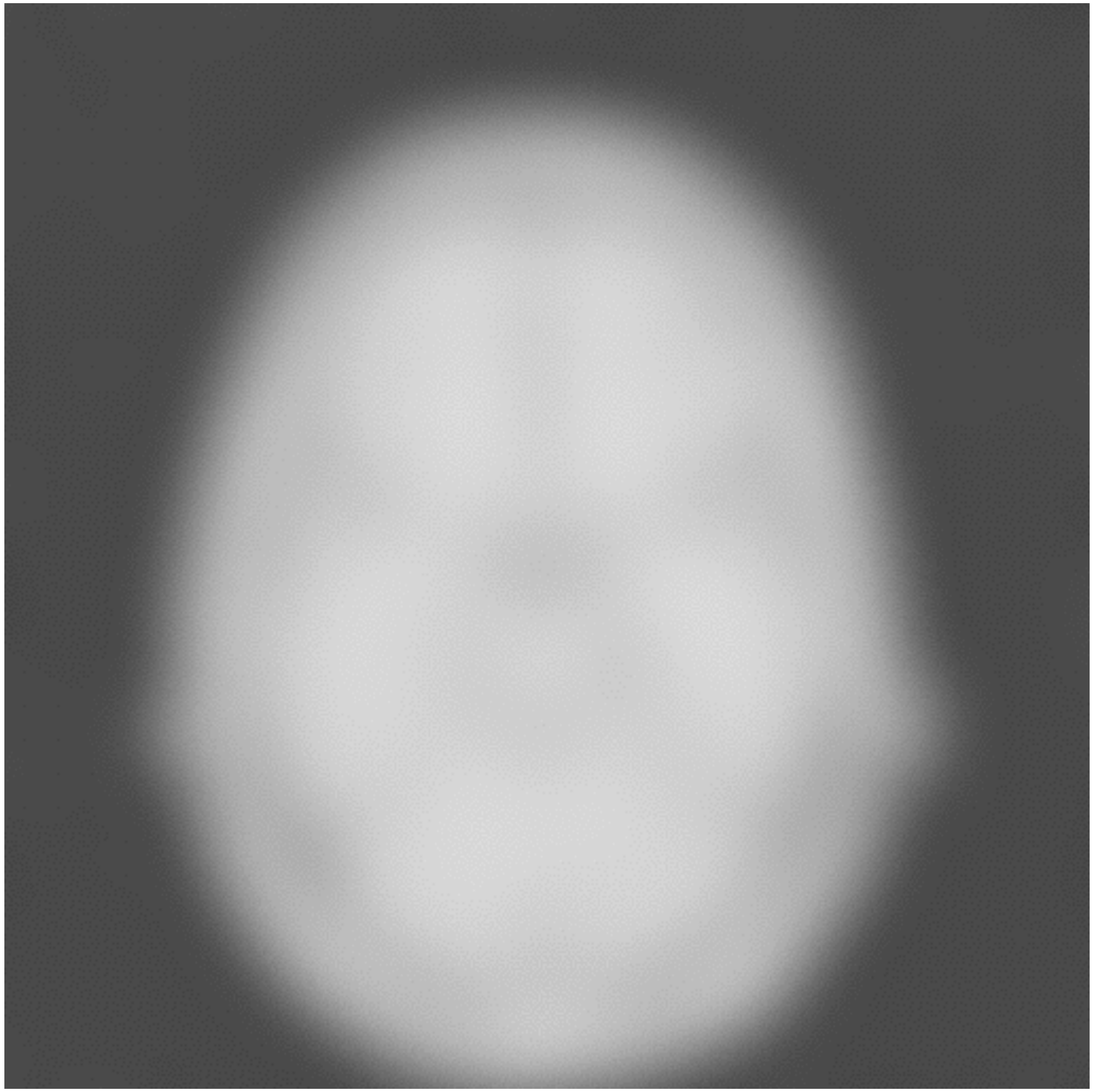
The Bilateral Filter Example



The Bilateral Filter Example



The Bilateral Filter Example



Local Contrast Enhancement

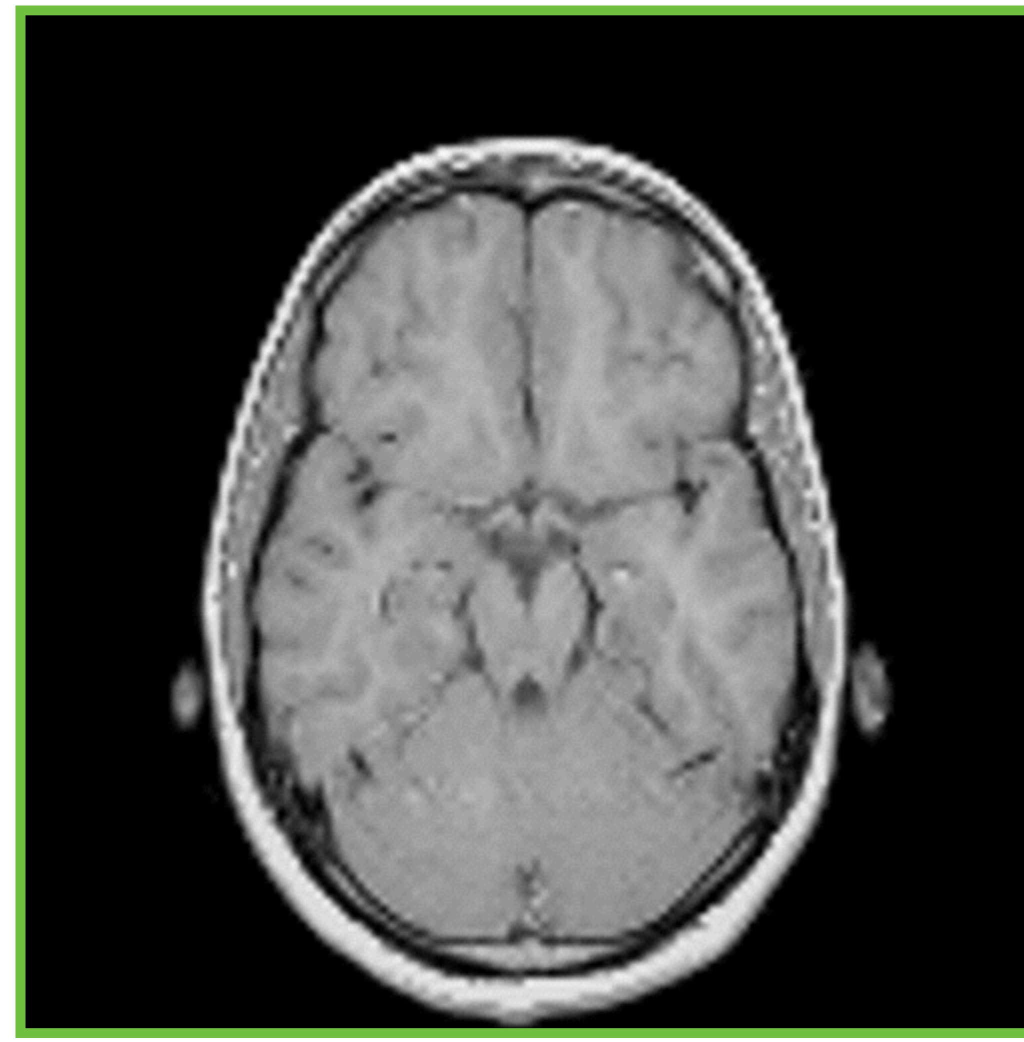
- Before, we have seen how to increase local contrast using the sharpening operator.
- We can achieve better results using a more general framework!

Local Contrast Enhancement Explained

$$O[i,j] = f[i,j] \cdot \left(\frac{f[i,j]}{(f \otimes g)[i,j])} \right)$$

Local Contrast Enhancement Explained

$$O[i,j] = \boxed{f[i,j] \cdot} \left(\frac{f[i,j]}{(f \otimes g)[i,j])} \right)$$

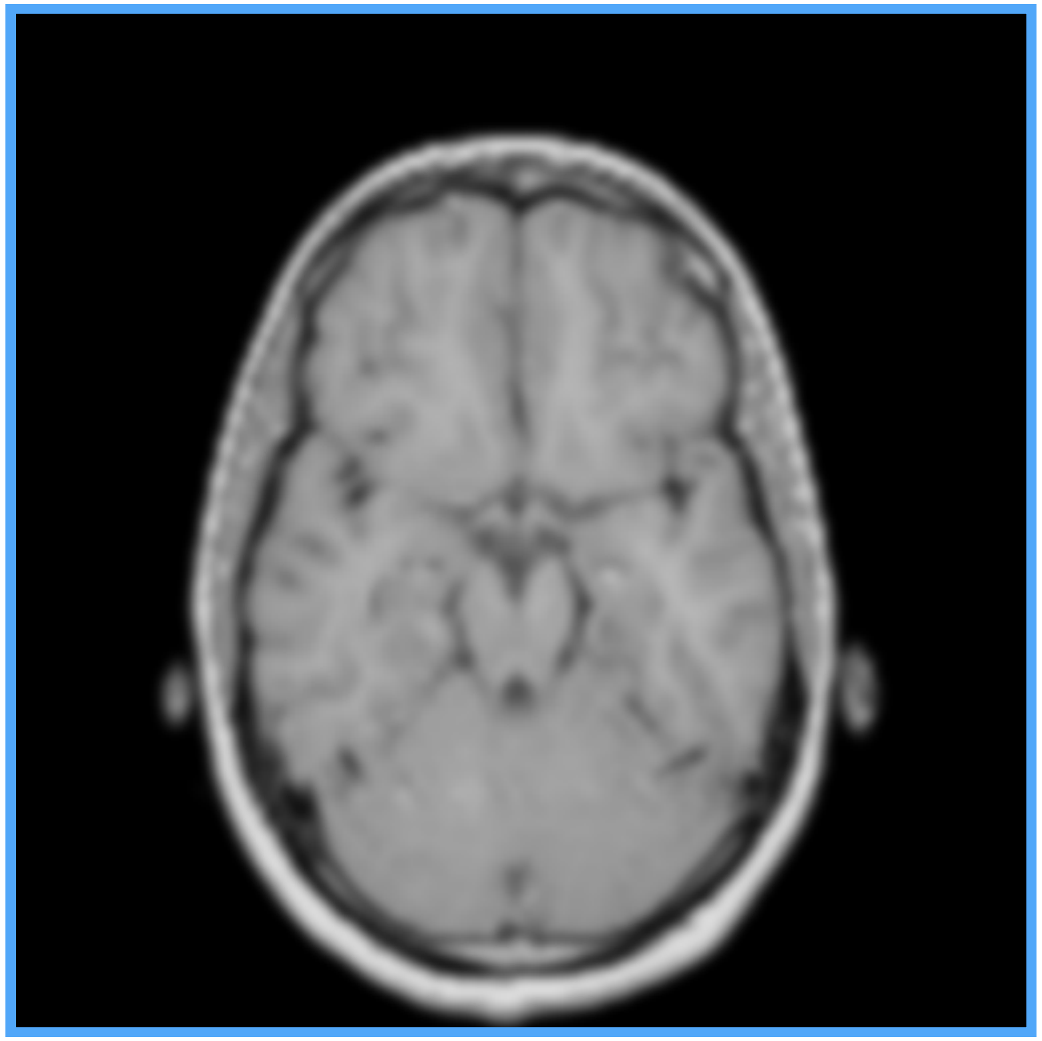


Local Contrast Enhancement Explained

$$O[i,j] = f[i,j] \cdot \left(\frac{f[i,j]}{(f \otimes g)[i,j])} \right)$$

Local Contrast Enhancement Explained

$$O[i,j] = f[i,j] \cdot \left(\frac{f[i,j]}{\boxed{(f \otimes g)[i,j])}} \right)$$



Local Contrast Enhancement Explained

$$O[i,j] = f[i,j] \cdot \left(\frac{f[i,j]}{(f \otimes g)[i,j])} \right)$$

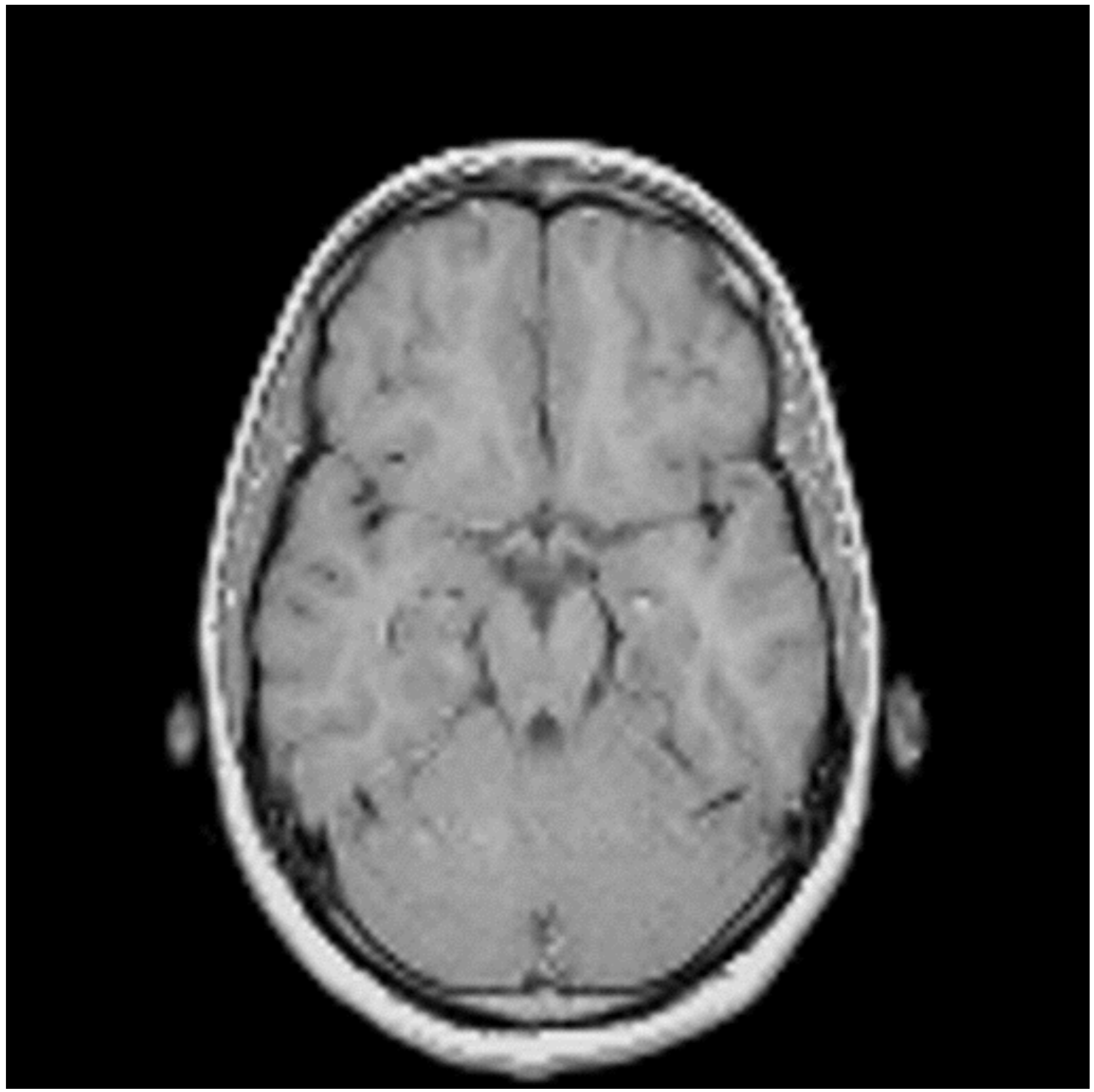
Local Contrast Enhancement Explained

$$O[i,j] = f[i,j] \cdot \boxed{\left(\frac{f[i,j]}{(f \otimes g)[i,j])} \right)}$$

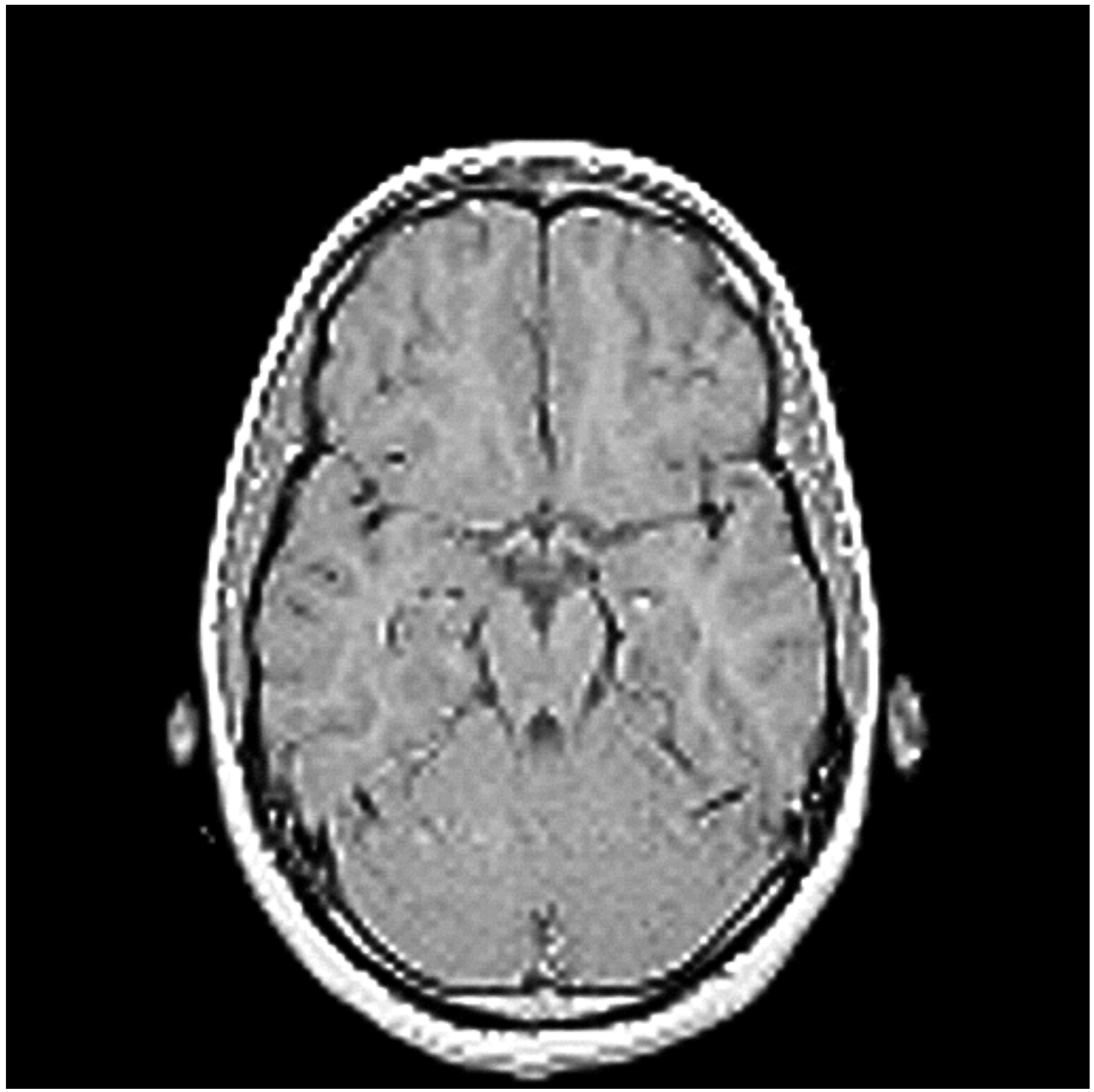
Local Contrast Enhancement Explained

$$O[i,j] = f[i,j] \cdot \left(\frac{f[i,j]}{(f \otimes g)[i,j])} \right)$$

Local Contrast Enhancement Example



Local Contrast Enhancement Example



Bonus: Deconvolution

$$\begin{cases} I_0 = 0.5 \\ I_{i+1} = I_i \cdot \left(\frac{J}{I_i \otimes K} \otimes K^\top \right) \end{cases}$$

Richardson–Lucy deconvolution

Bonus: Deconvolution

$$\begin{cases} I_0 = 0.5 \\ I_{i+1} = I_i \cdot \left(\dfrac{\boxed{J}}{I_i \otimes K} \otimes K^\top \right) \end{cases}$$

Input Blurred Image

Richardson–Lucy deconvolution

Bonus: Deconvolution

$$\begin{cases} I_0 = 0.5 \\ I_{i+1} = I_i \cdot \left(\frac{J}{I_i \otimes K} \otimes K^\top \right) \end{cases}$$

Richardson–Lucy deconvolution

Bonus: Deconvolution Example

Bonus: Deconvolution Example

Local Contrast Enhancement

- When using linear filters we may introduce halos!
    - halos —> BIAS!
- It is better to use non-linear filters such as the bilateral filter, the guided filter, WLS, etc.

Image Upsampling

Why Upsampling?

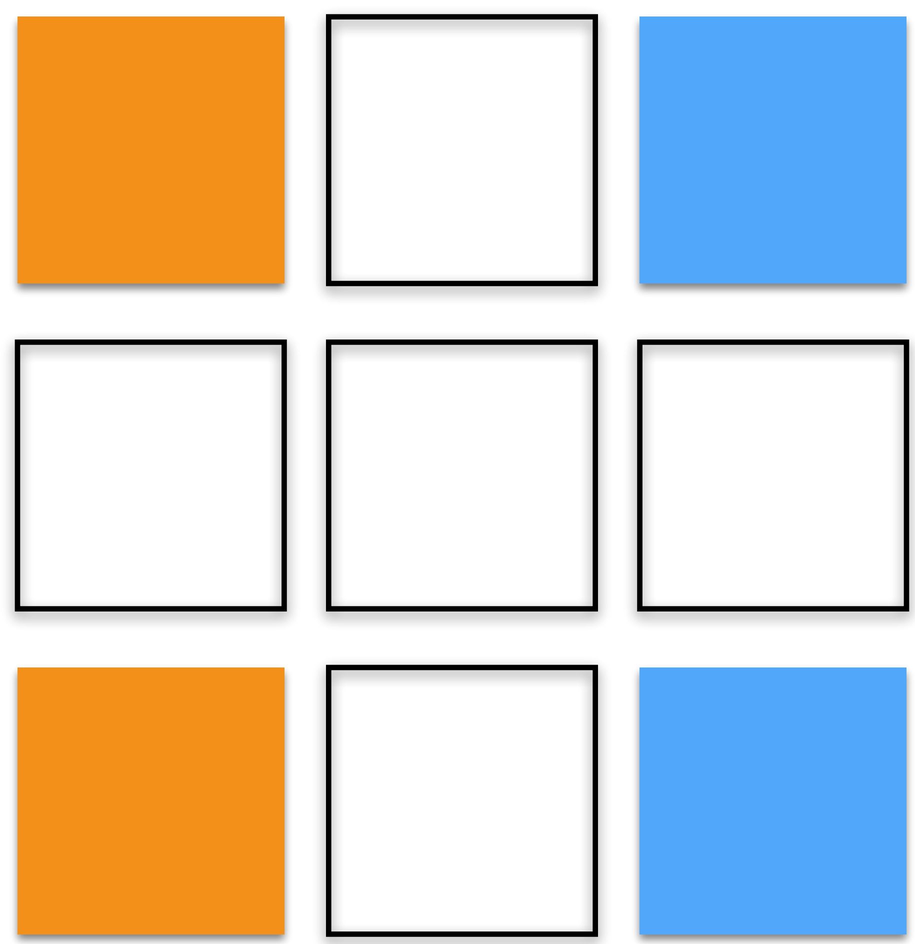
- The main reason why we want to upsample (we invent data basically) our input data is that they have a very low resolution

- Forget 4K for your flicks, we have 512x512 resolution in happy days

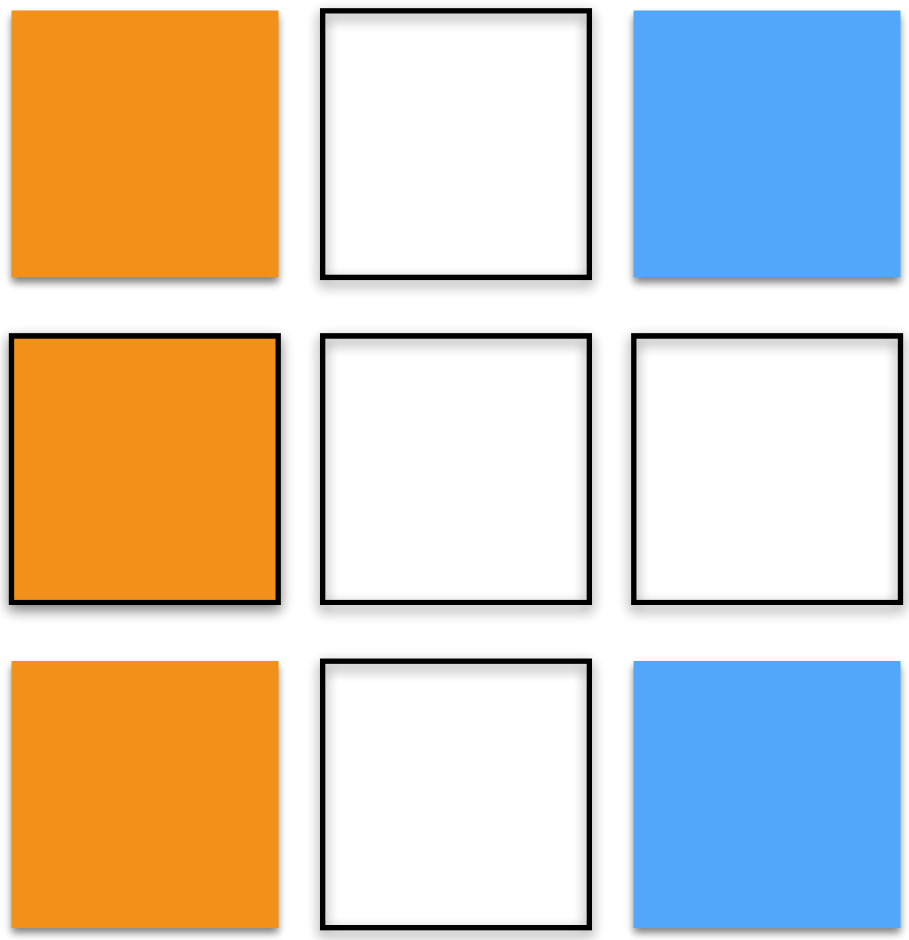
Upsampling

- When we upsample we need to invent the pixel in between the original ones…
- Basic solution:
  - For each missing pixel:
    - find the closest (norm 1, 2, whatever) “real” pixel with intensity/color C_n
    - Set the intensity/color of the missing pixel equals to C_n

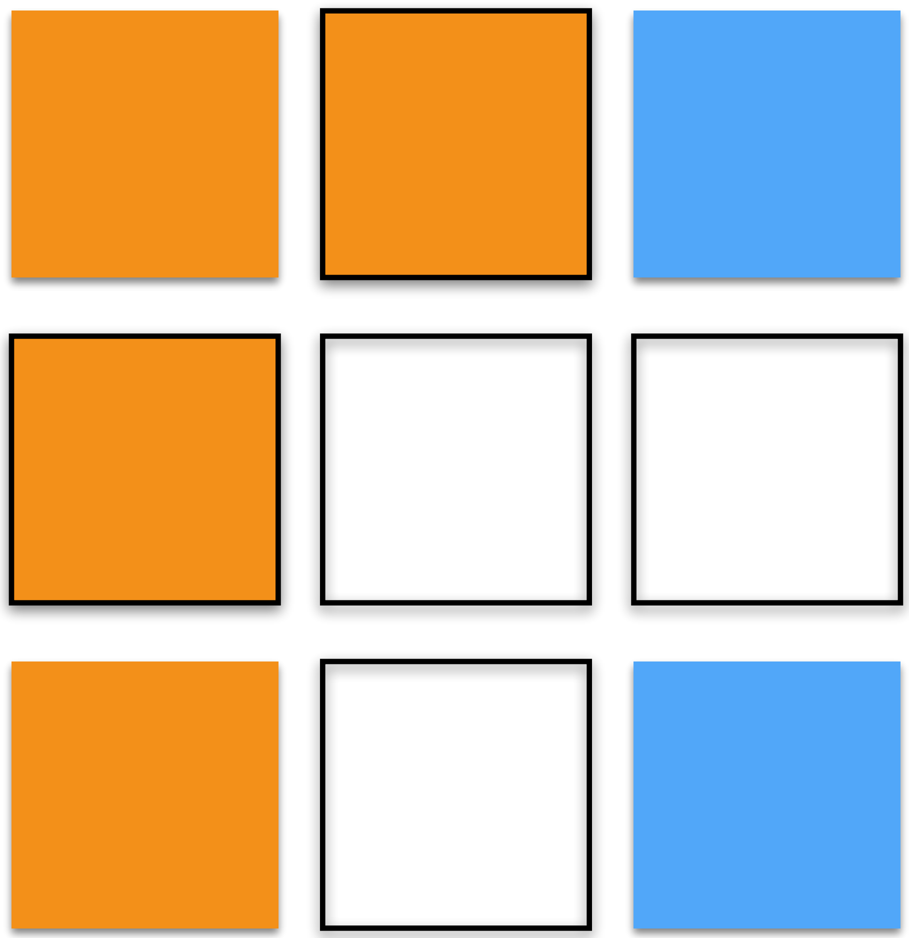
Upsampling: Nearest Neighbors



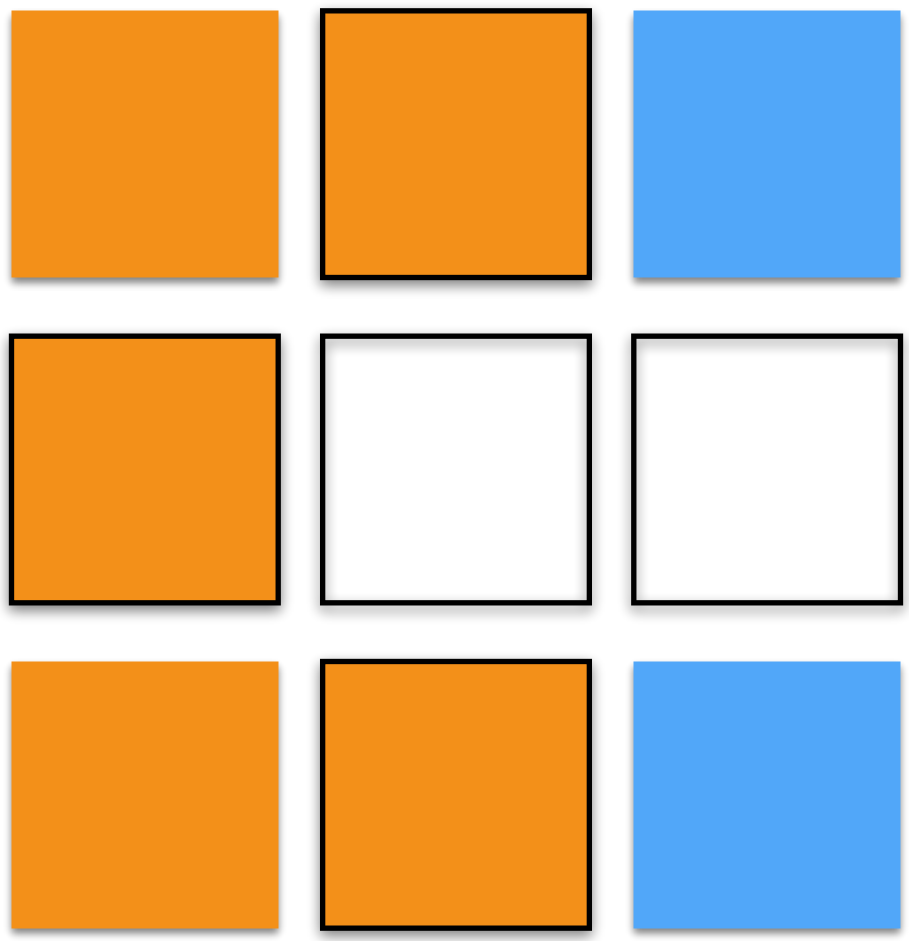
Upsampling:
Nearest Neighbors



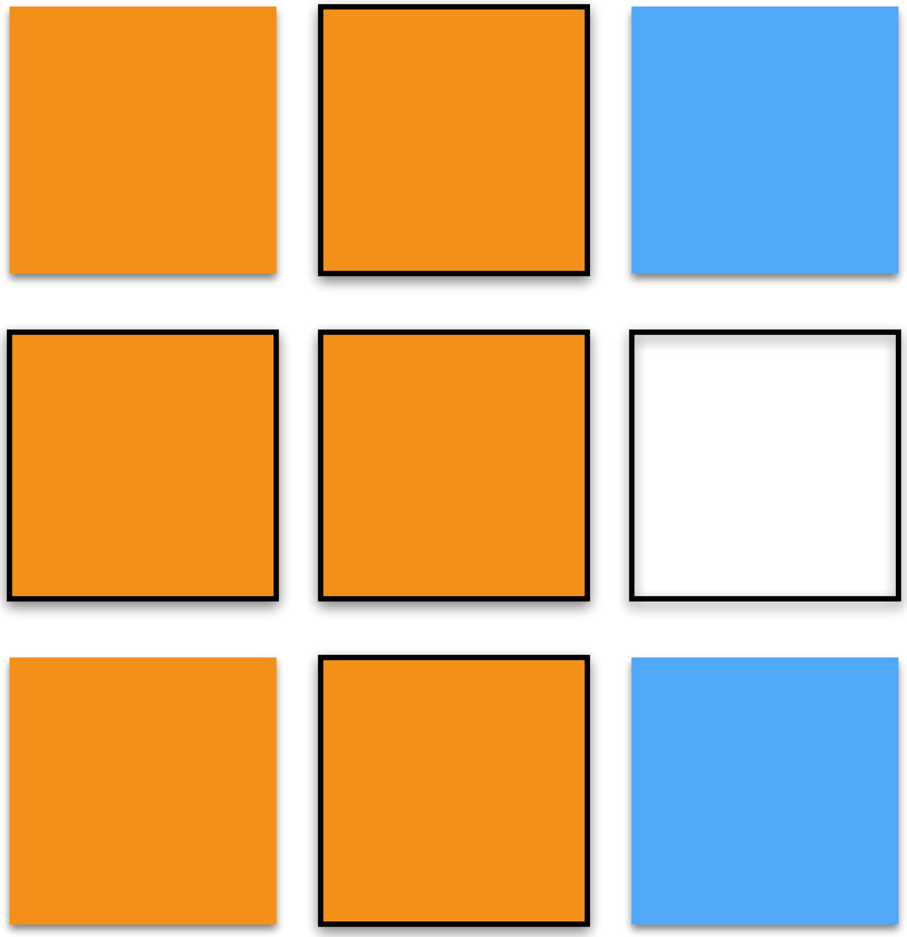
Upsampling: Nearest Neighbors



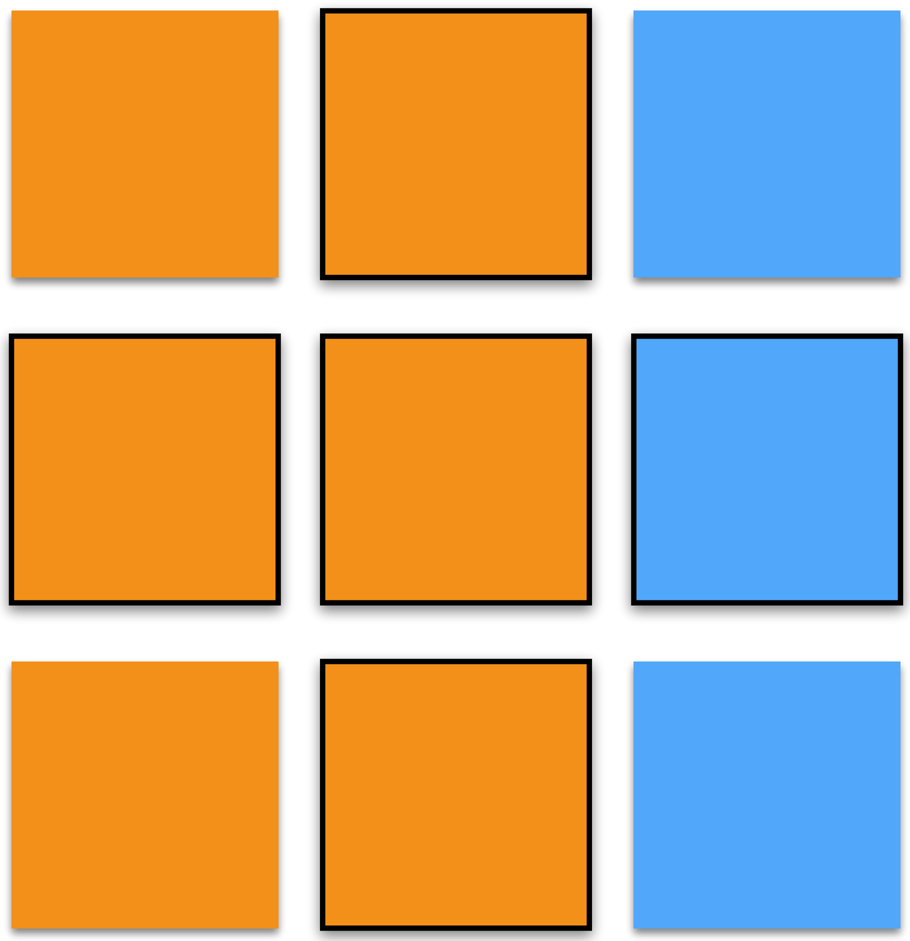
Upsampling:
Nearest Neighbors



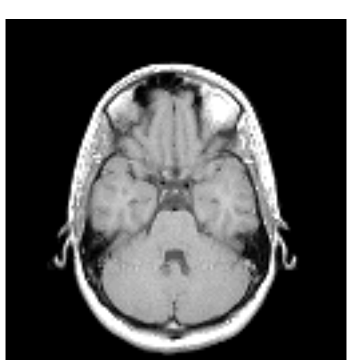
Upsampling:
Nearest Neighbors



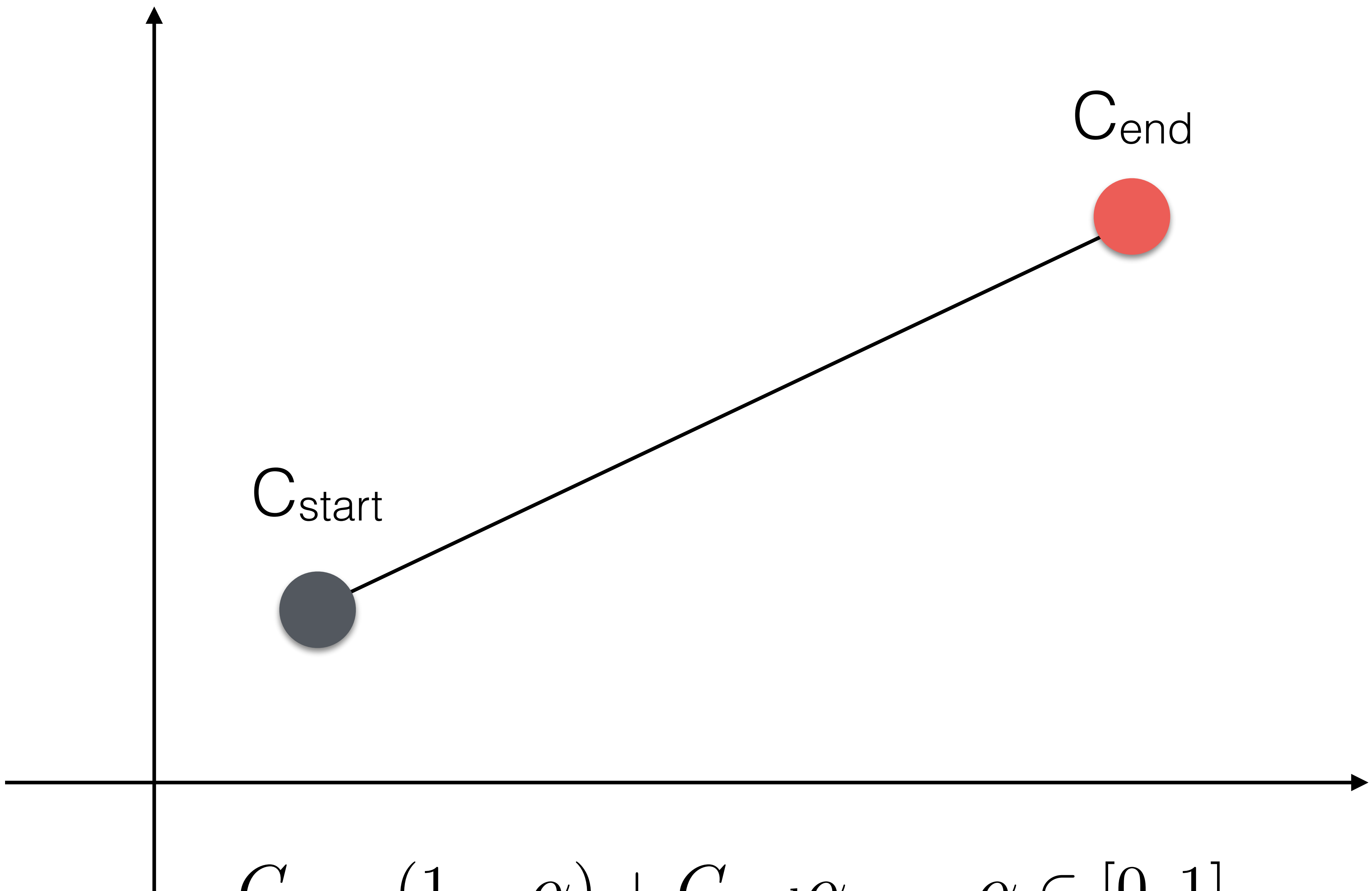
Upsampling:
Nearest Neighbors



Upsampling: Nearest Neighbors

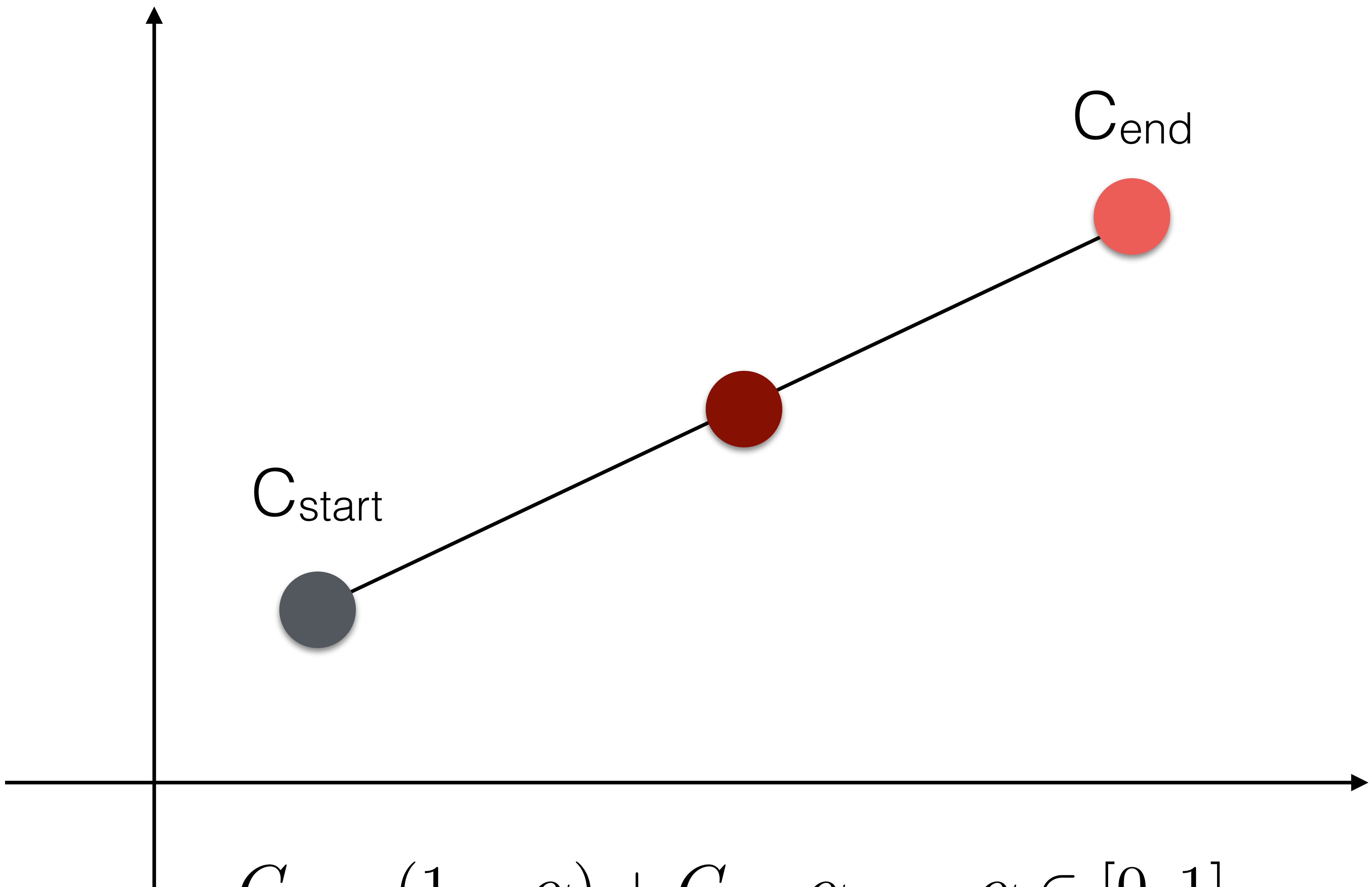


Upsampling 1D: Linear Interpolation



$$C_{\text{start}}(1-\alpha) + C_{\text{end}}\alpha \qquad \alpha \in [0, 1]$$

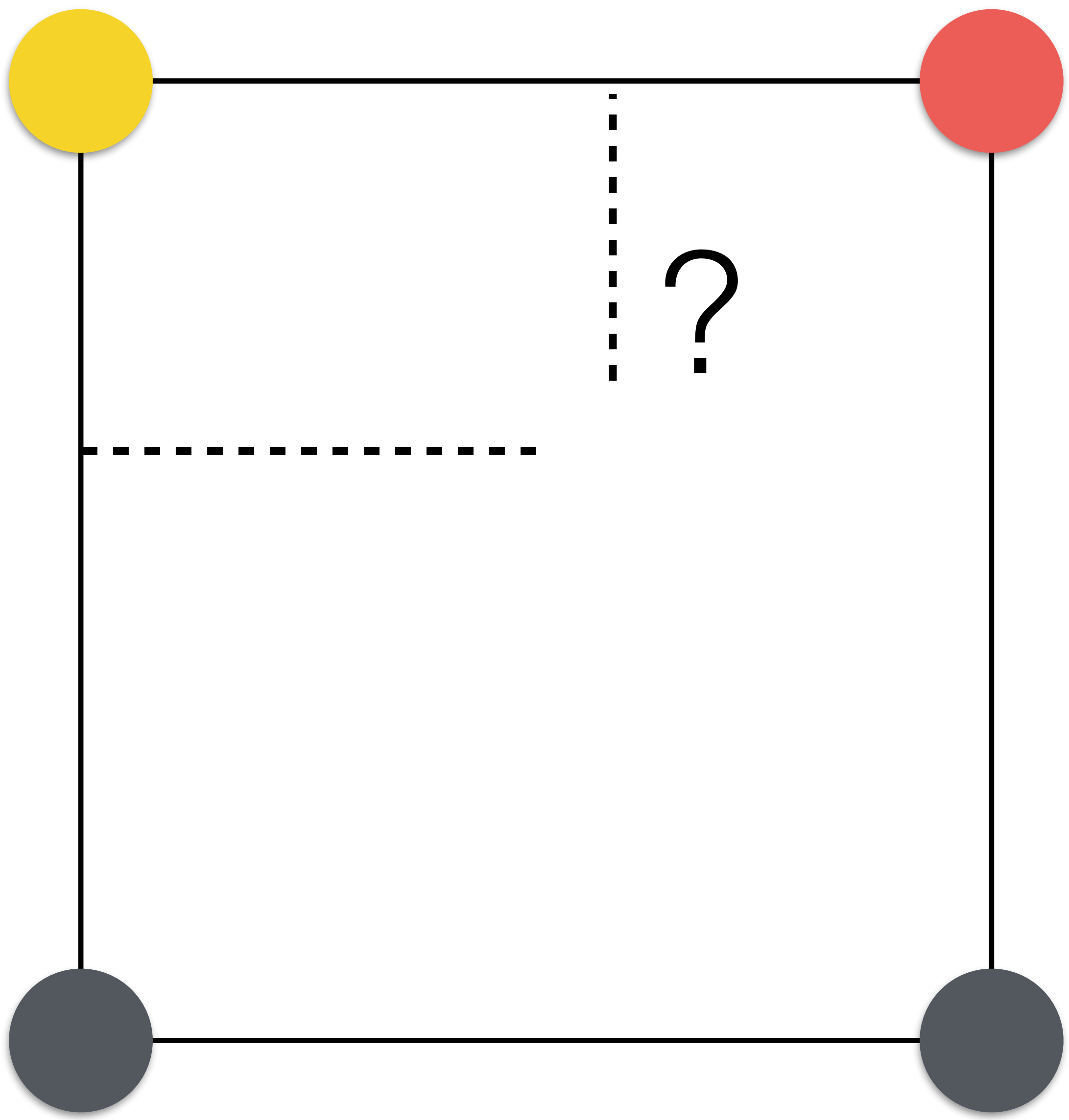
Upsampling 1D: Linear Interpolation



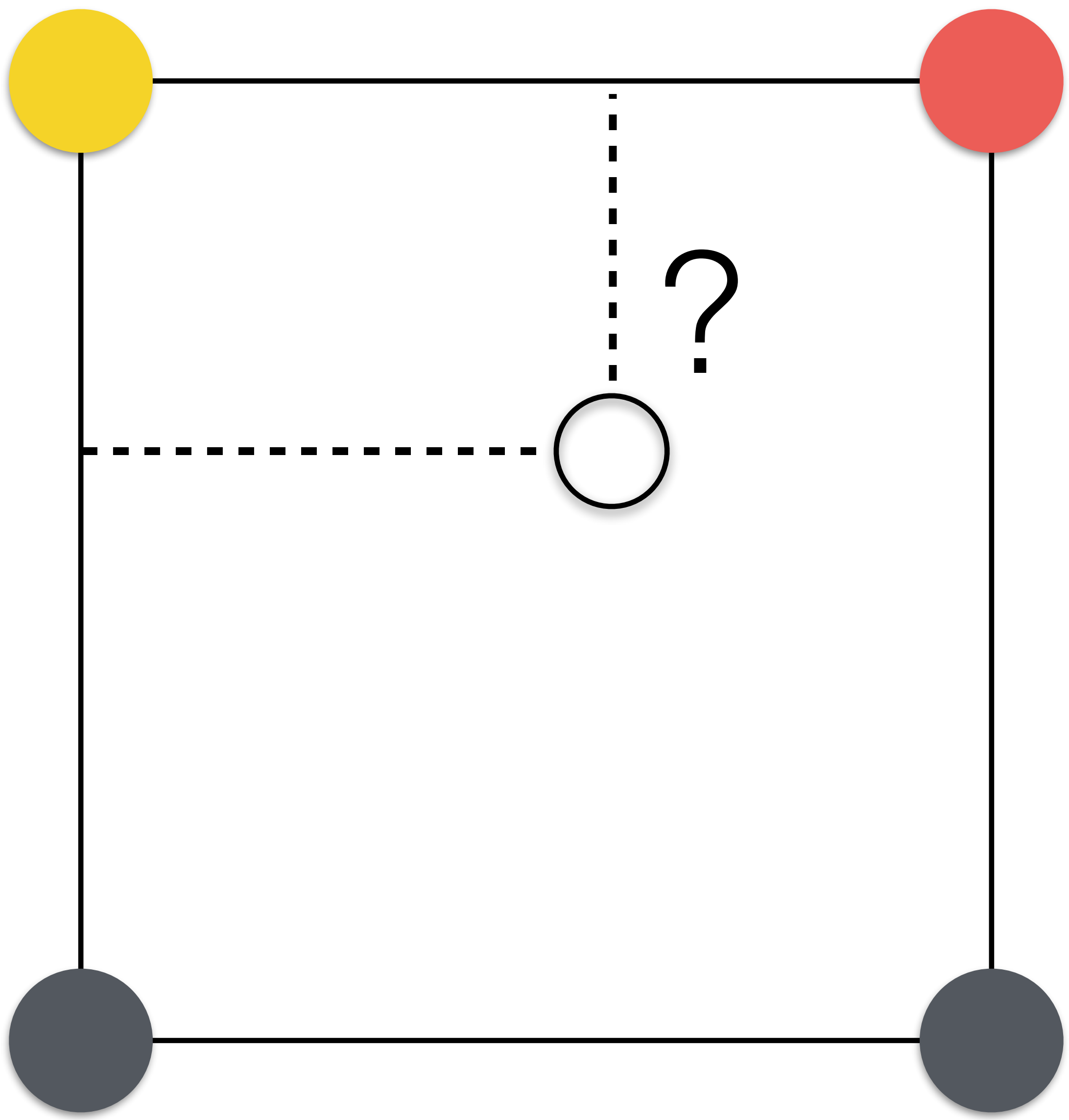
$$C_{\text{start}}(1-\alpha) + C_{\text{end}}\alpha \qquad \alpha \in [0,1]$$

This becomes a bi-linear interpolation in 2D!

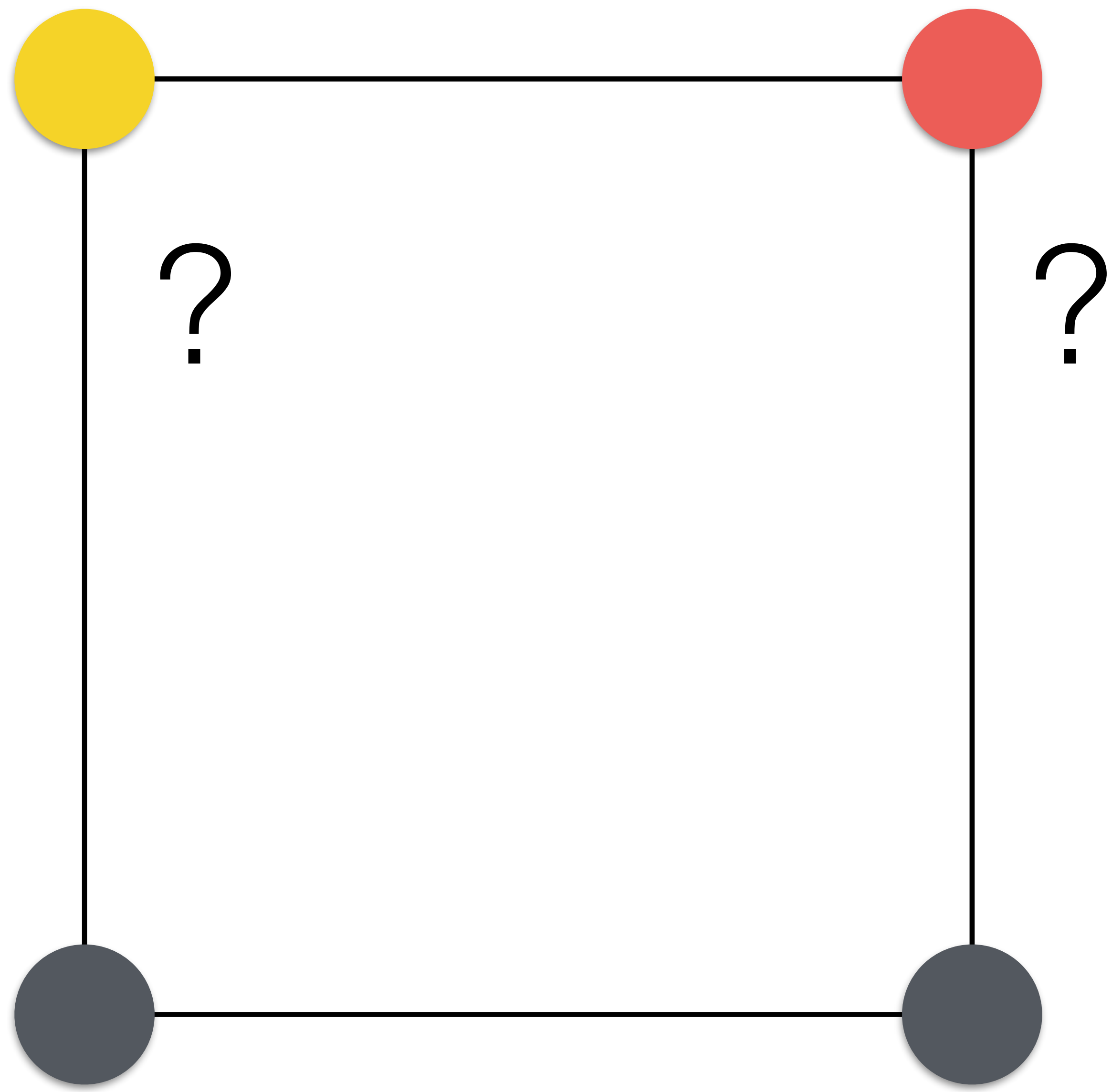
Bilinear Upsampling 2D



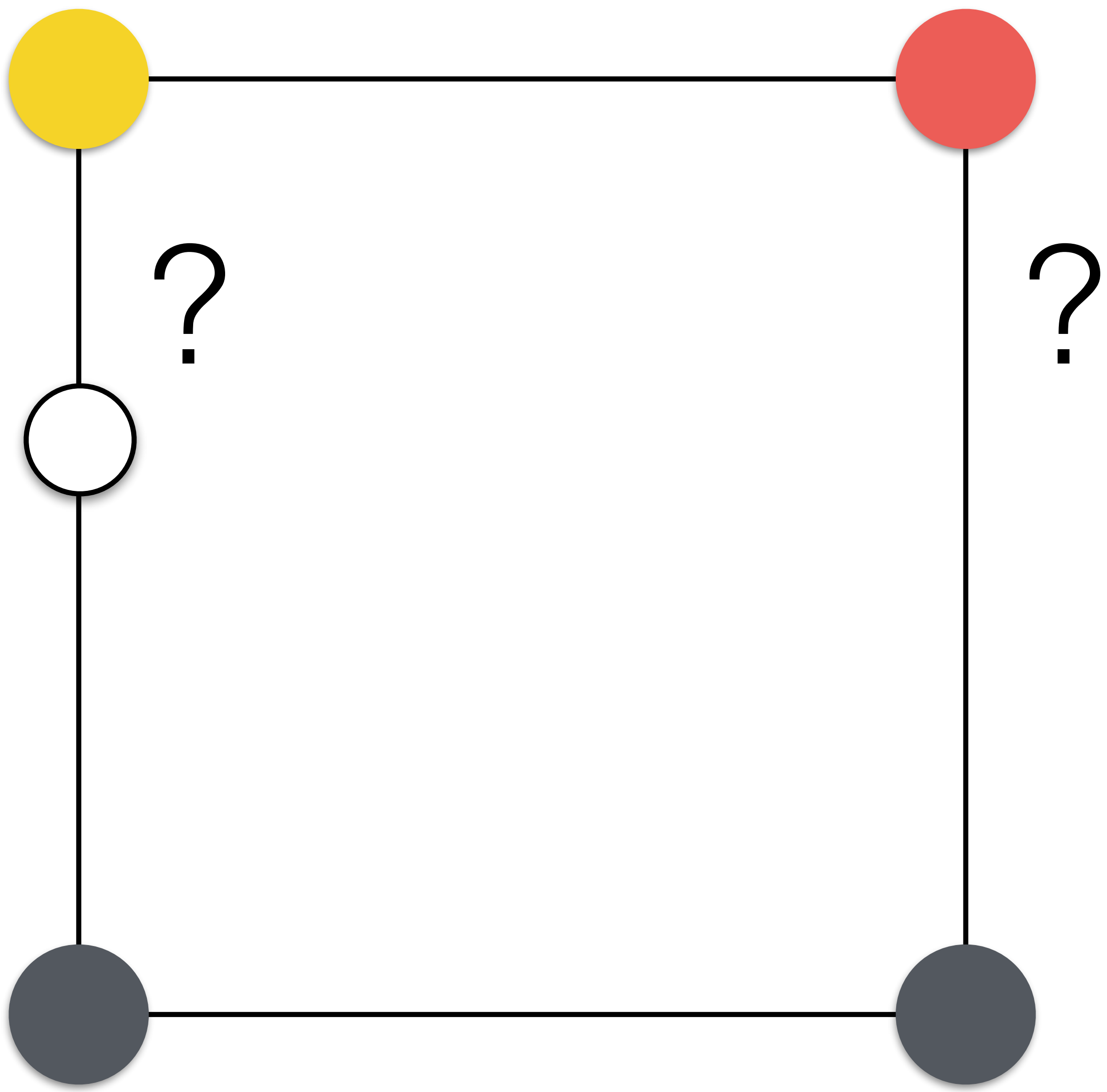
Bilinear Upsampling 2D



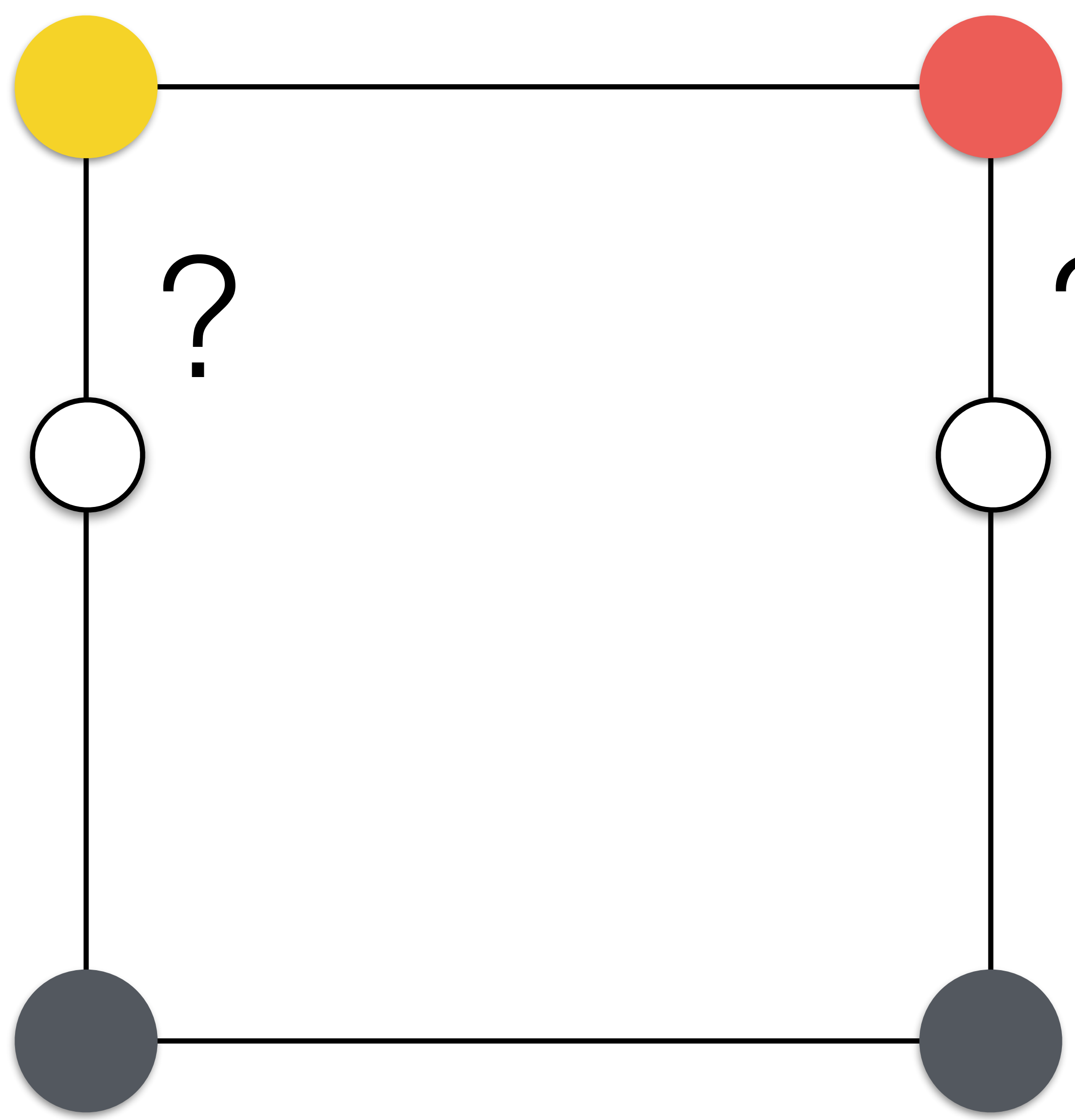
Bilinear Upsampling 2D



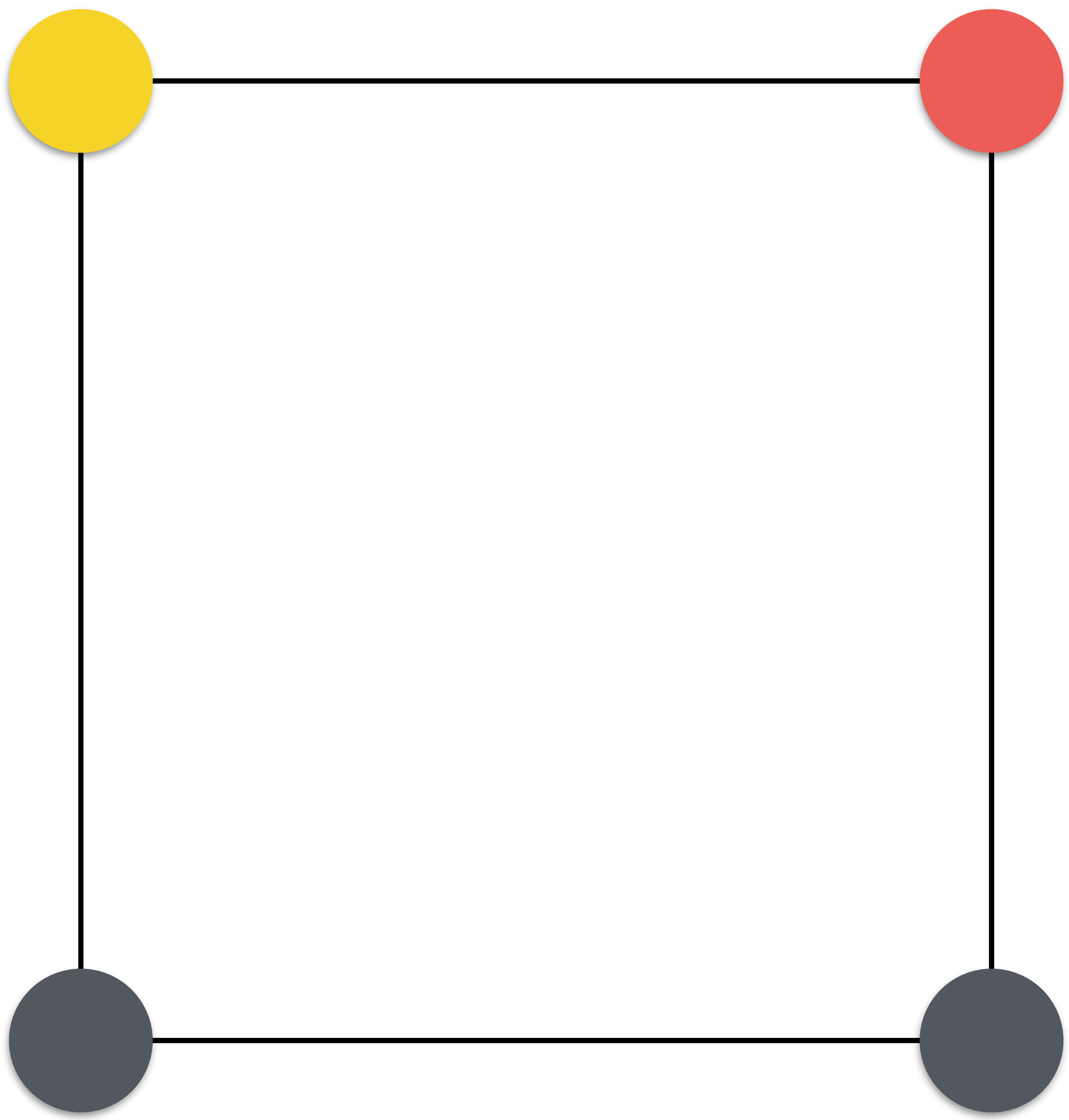
Bilinear Upsampling 2D



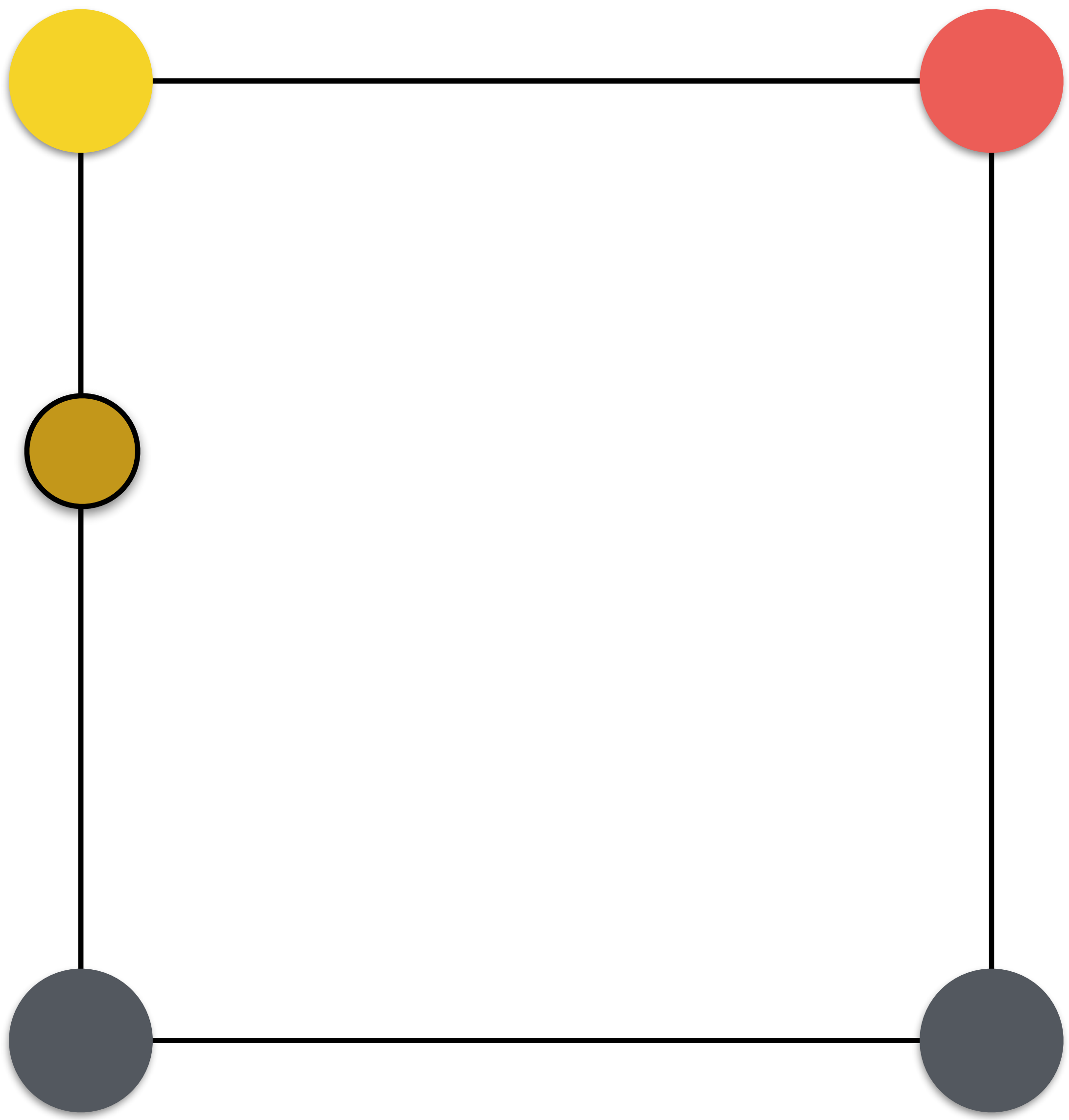
Bilinear Upsampling 2D



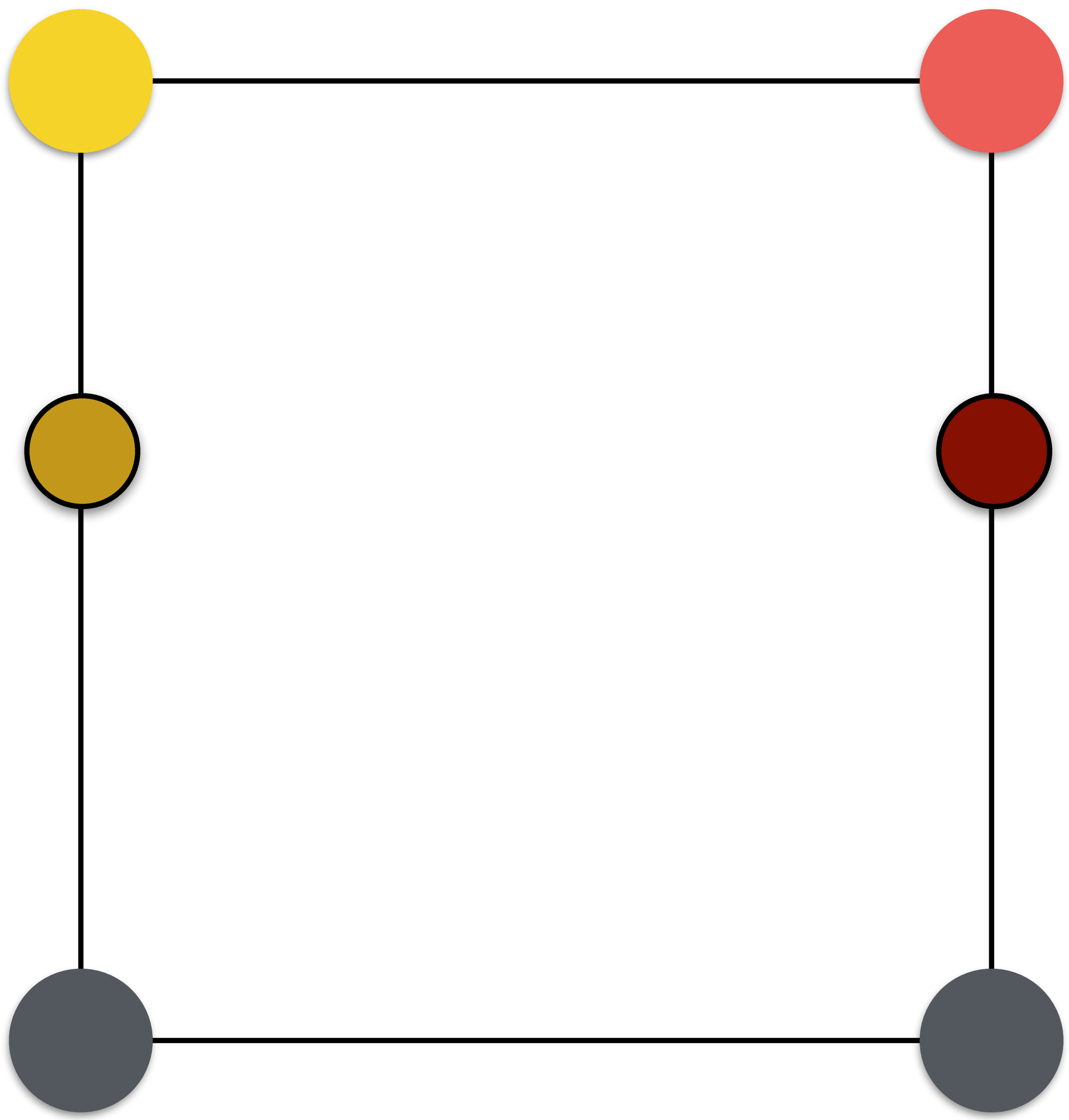
Bilinear Upsampling 2D



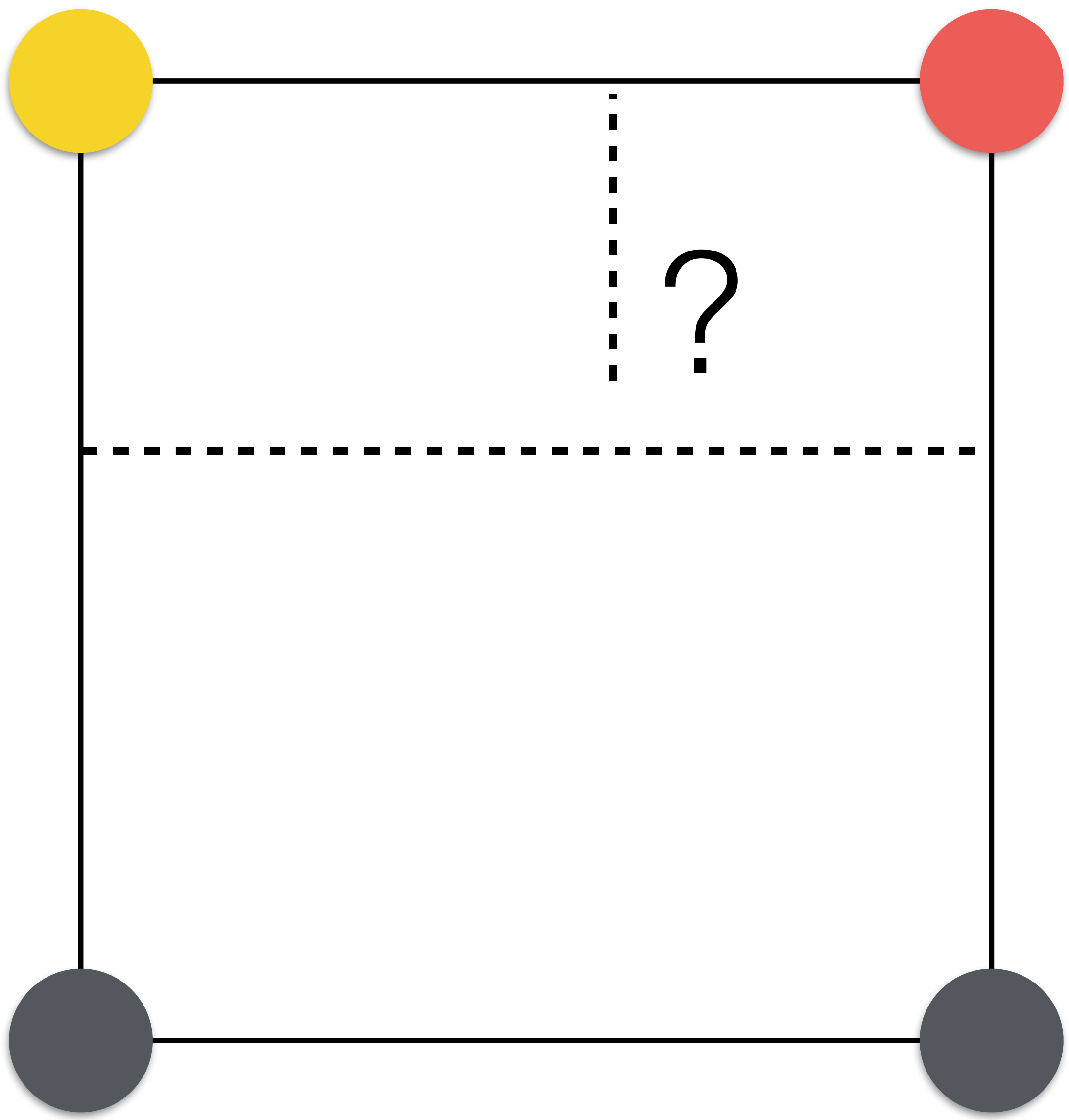
Bilinear Upsampling 2D



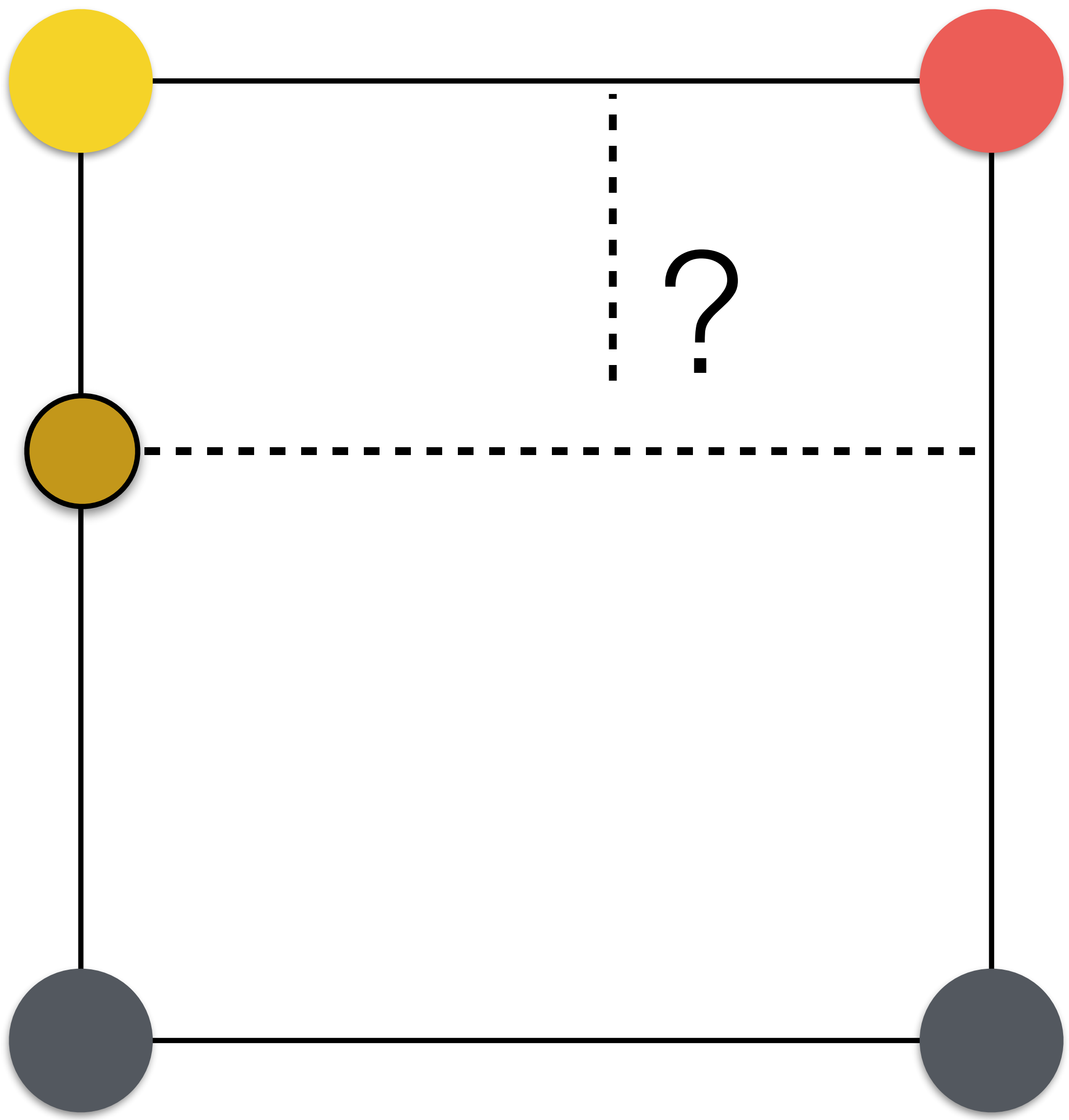
Bilinear Upsampling 2D



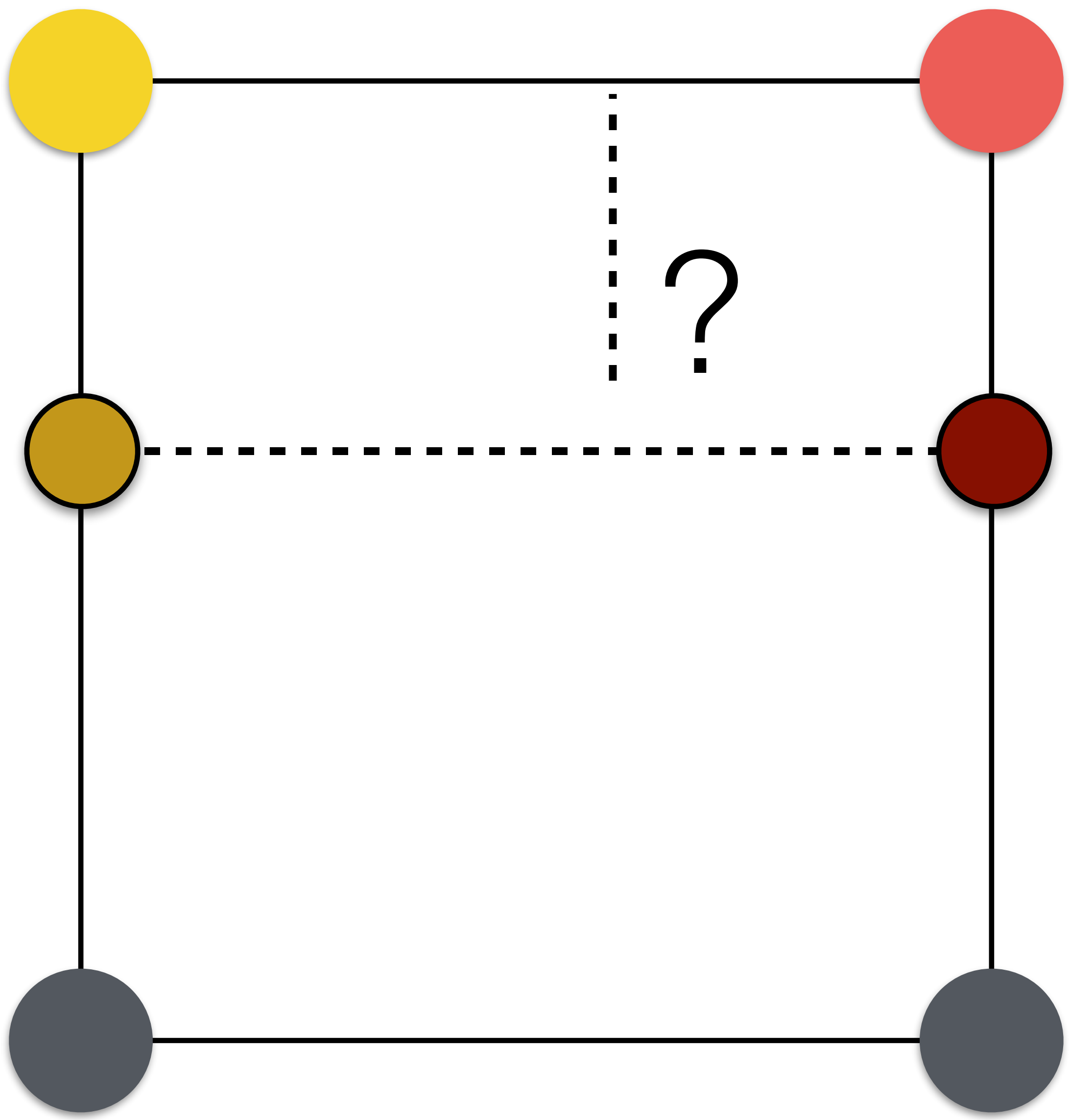
Bilinear Upsampling 2D



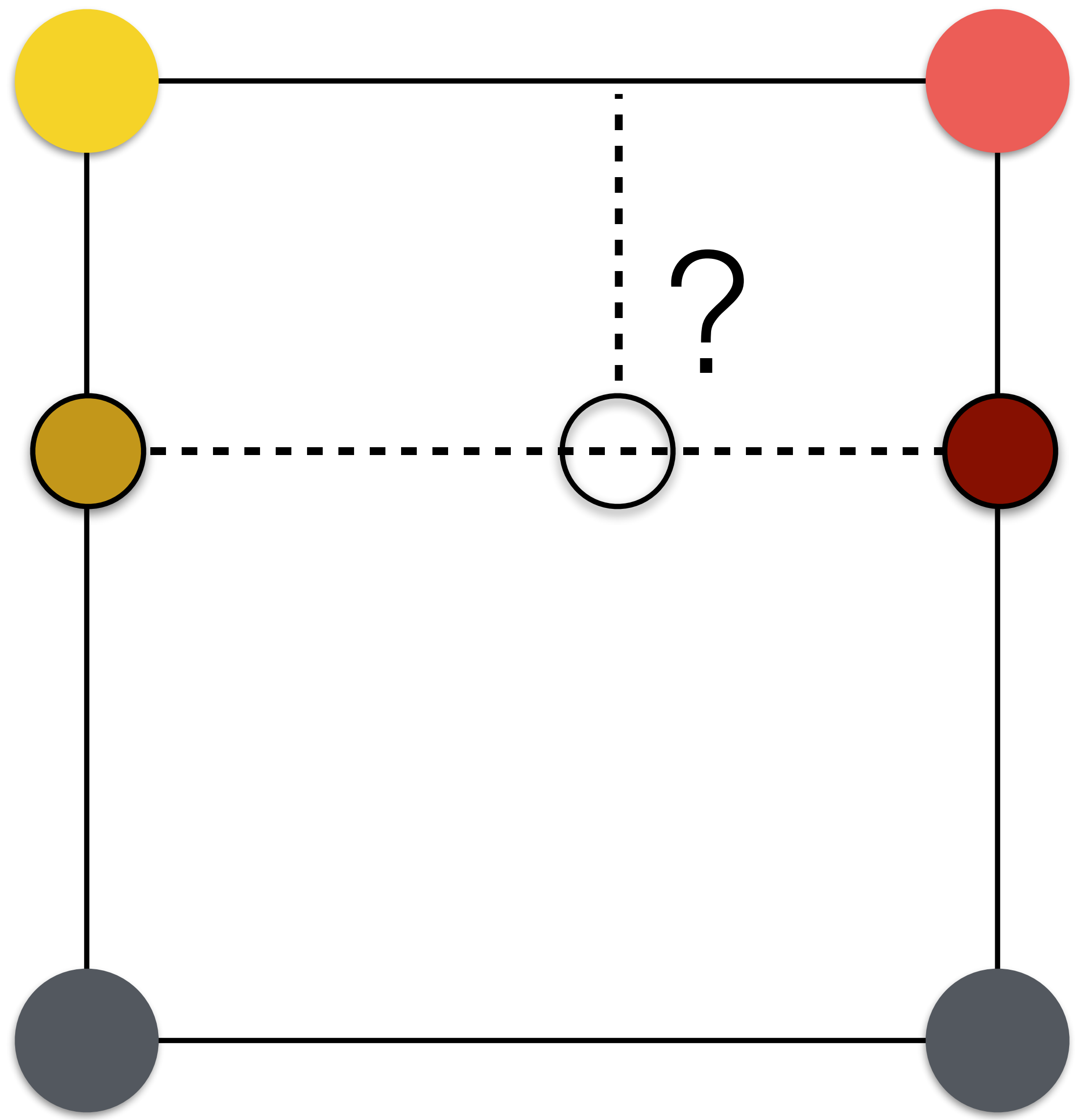
Bilinear Upsampling 2D



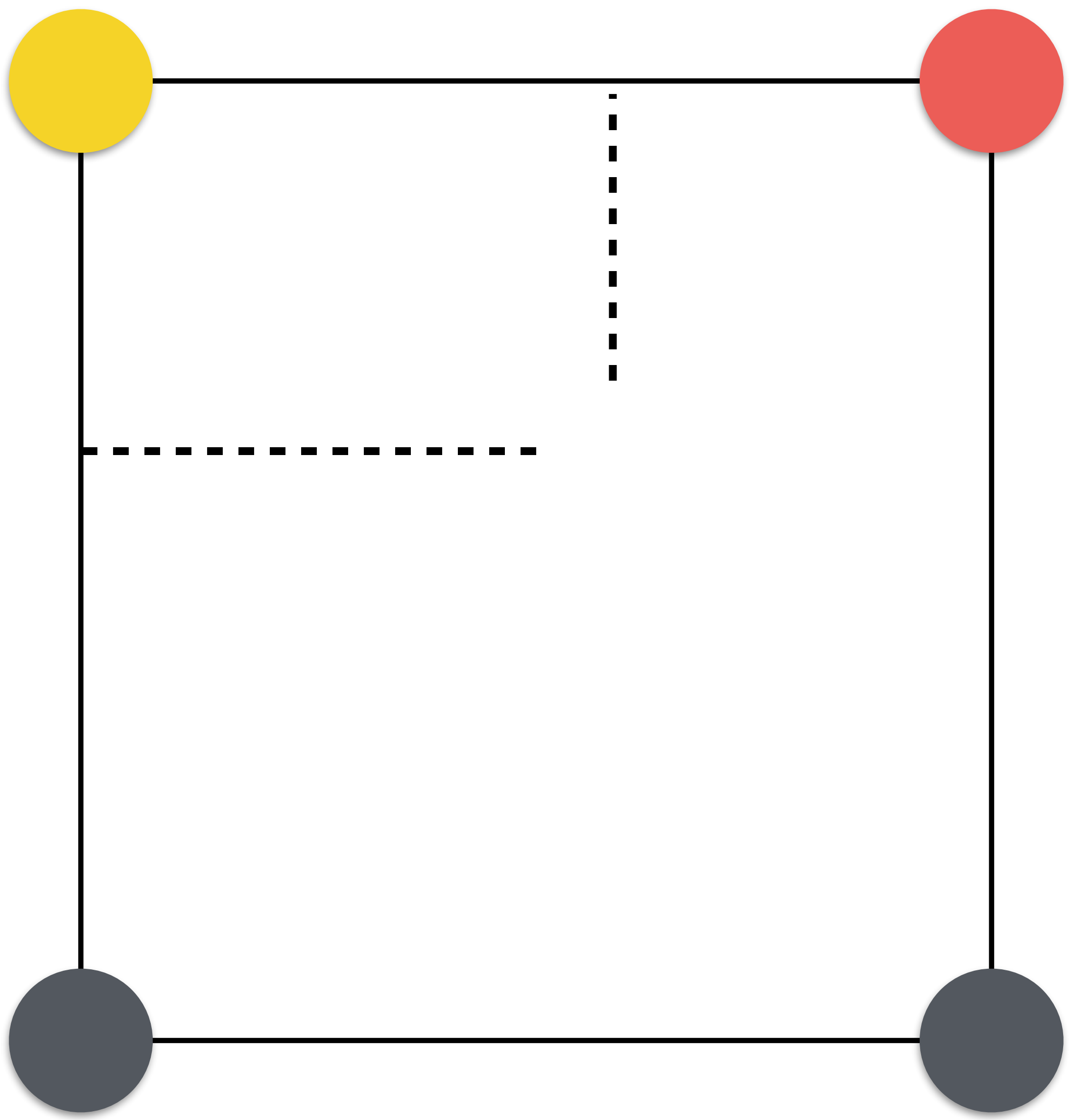
Bilinear Upsampling 2D



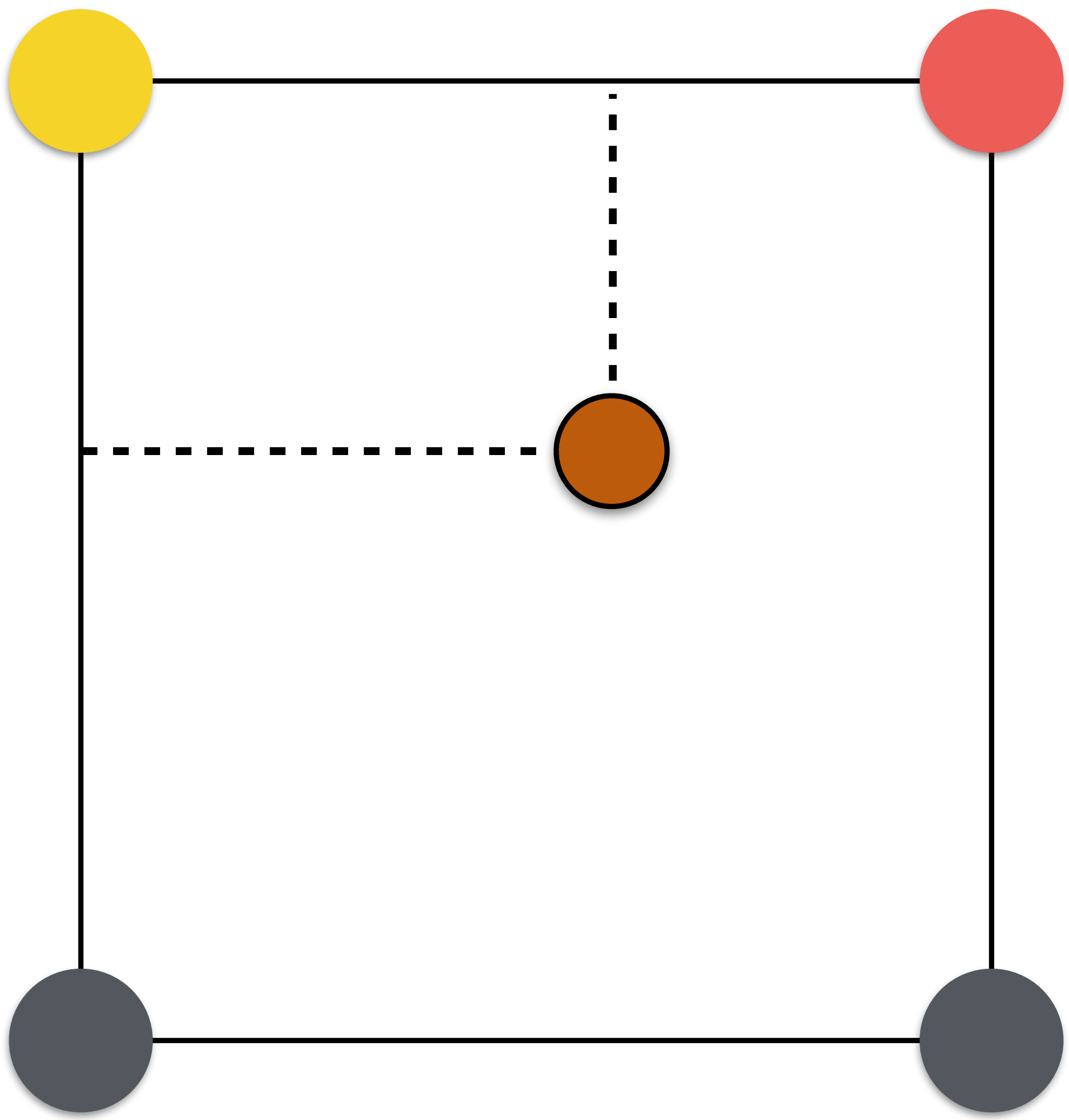
Bilinear Upsampling 2D



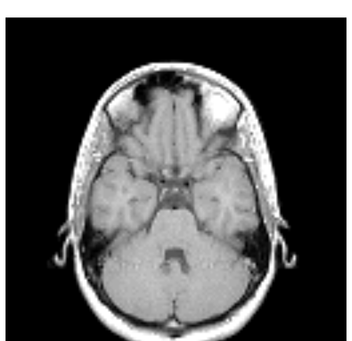
Bilinear Upsampling 2D

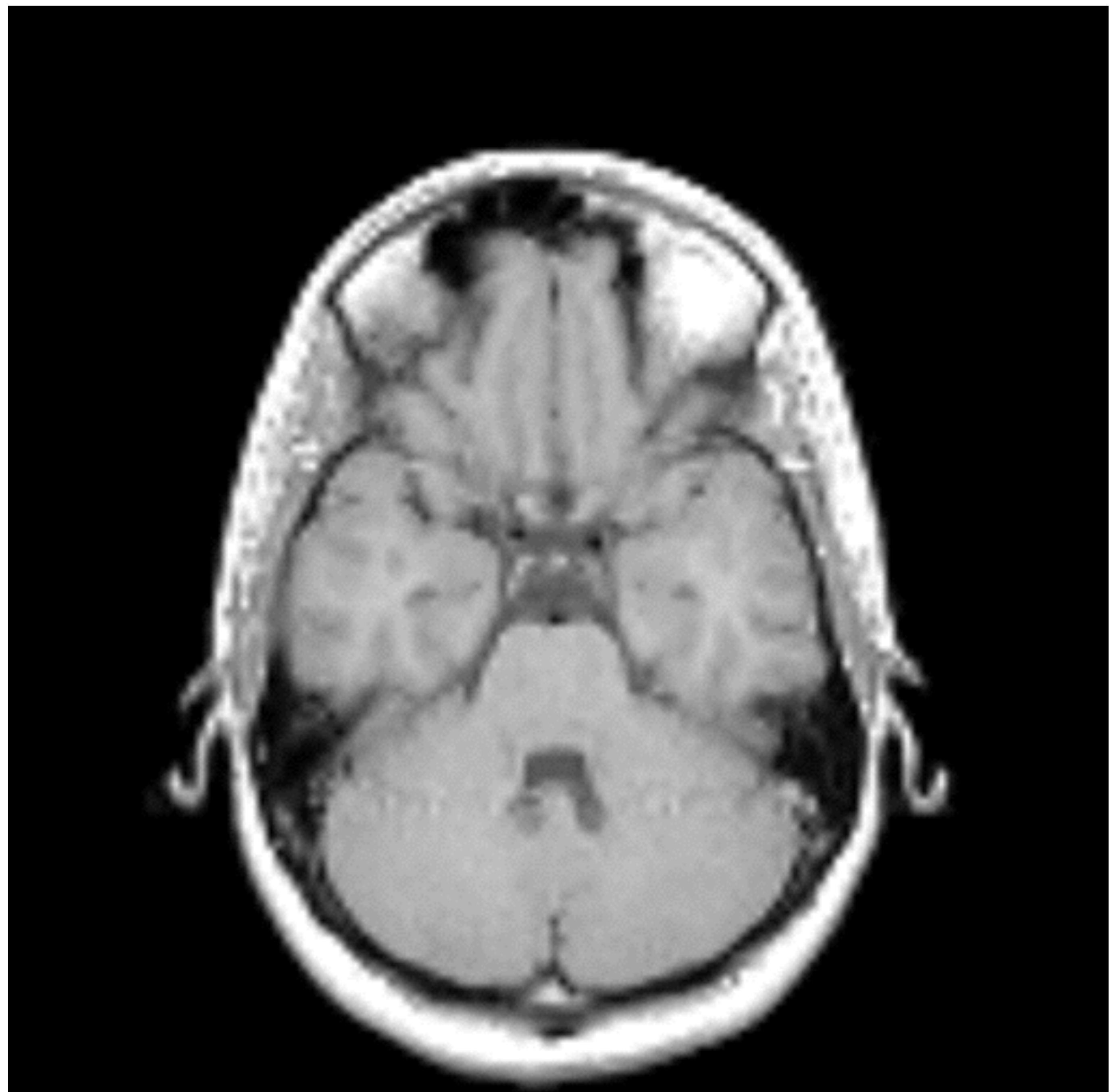


Bilinear Upsampling 2D



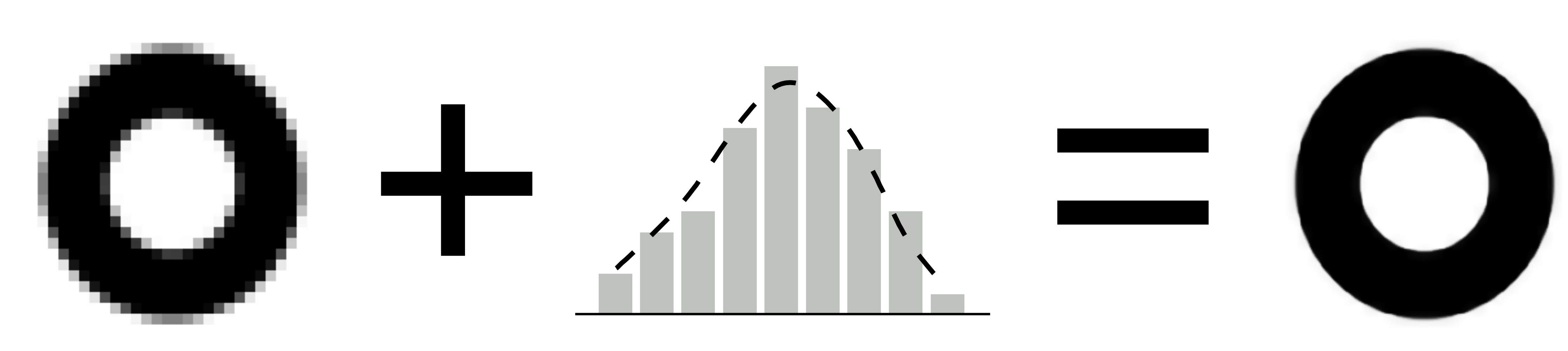
Bilinear Upsampling Example





Can we do it better?

Upsampling



that’s all folks!