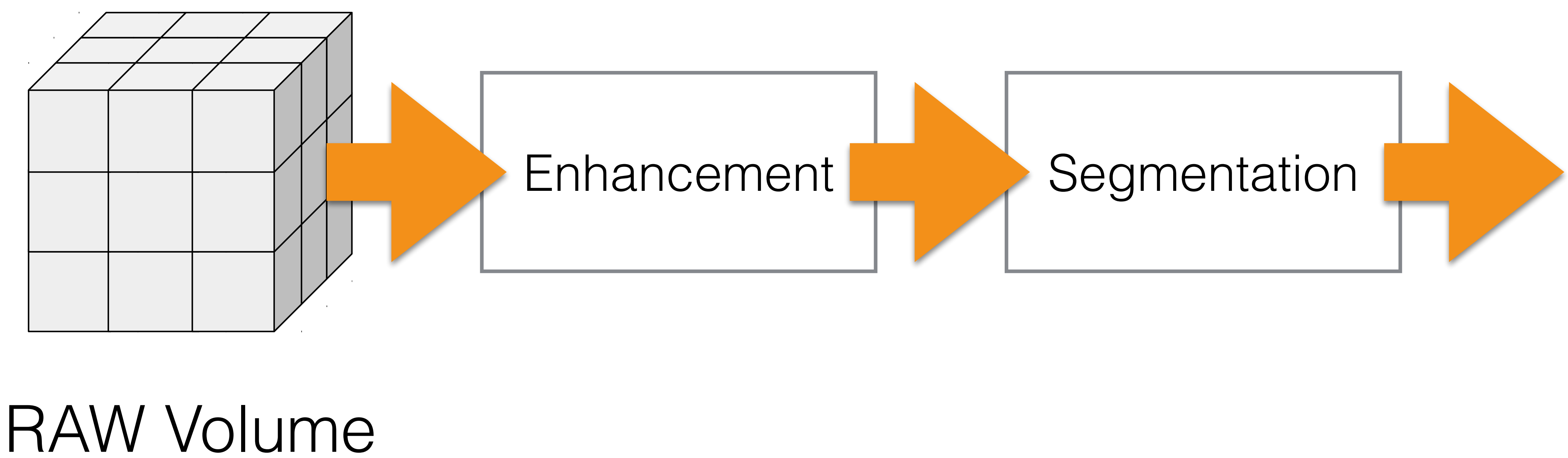
3D from Volume: Part I

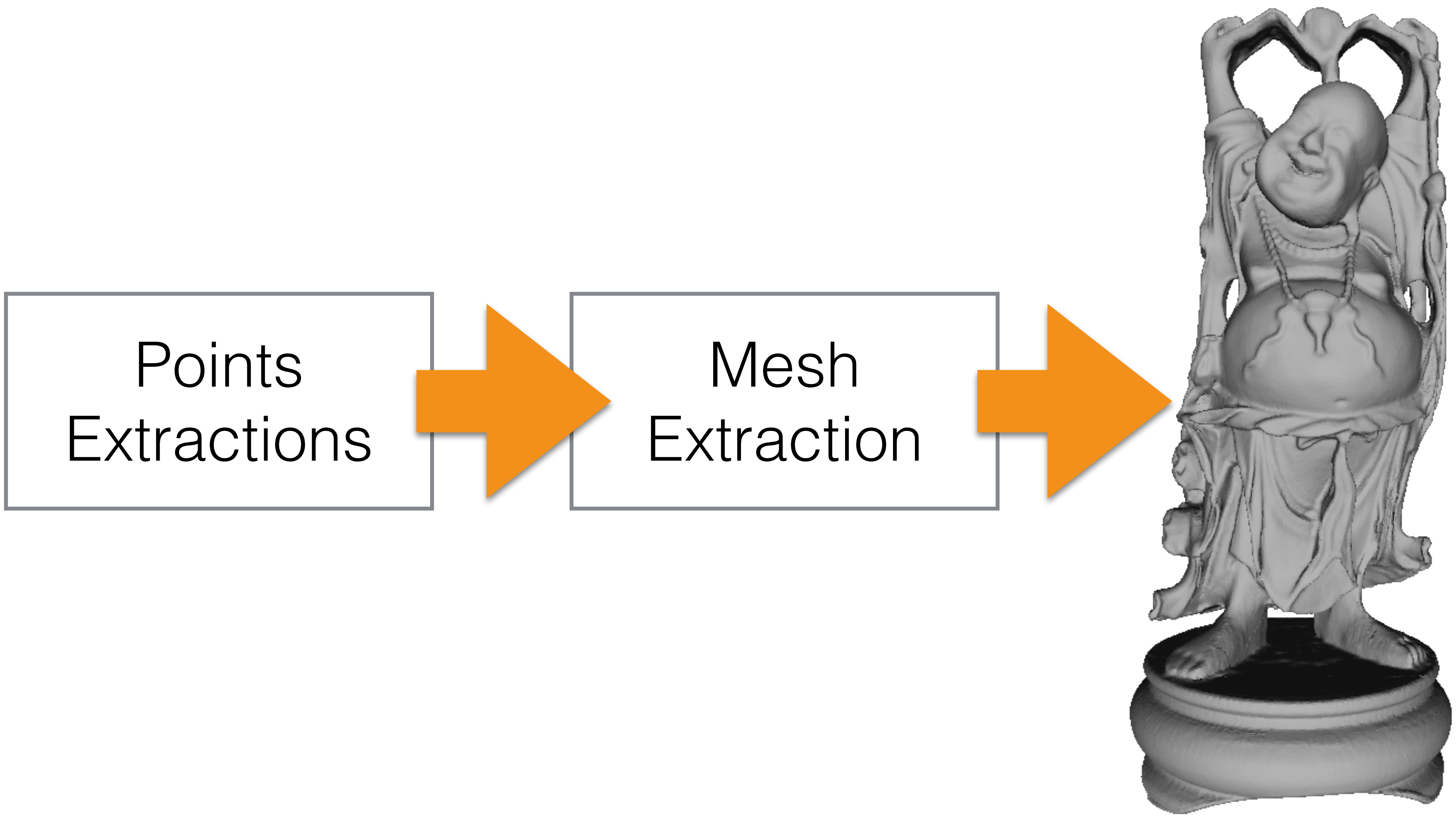
Francesco Banterle, Ph.D.
francesco.banterle@isti.cnr.it

The Main Pipeline

The Processing Pipeline



The Processing Pipeline



3D Mesh

The Processing Pipeline

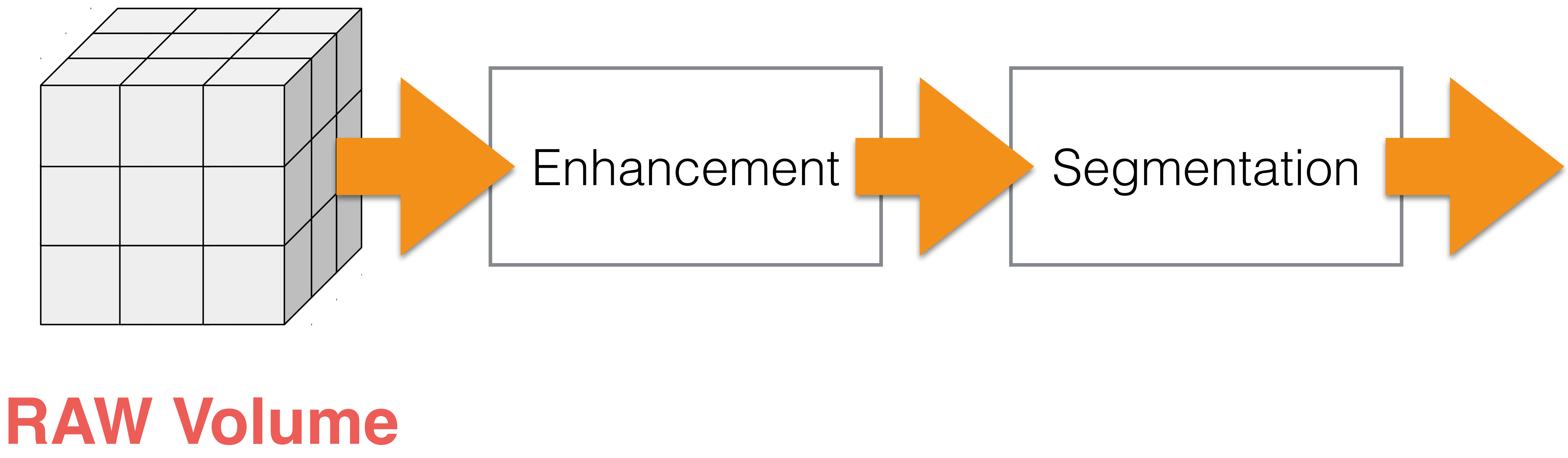


Image Enhancement

- **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

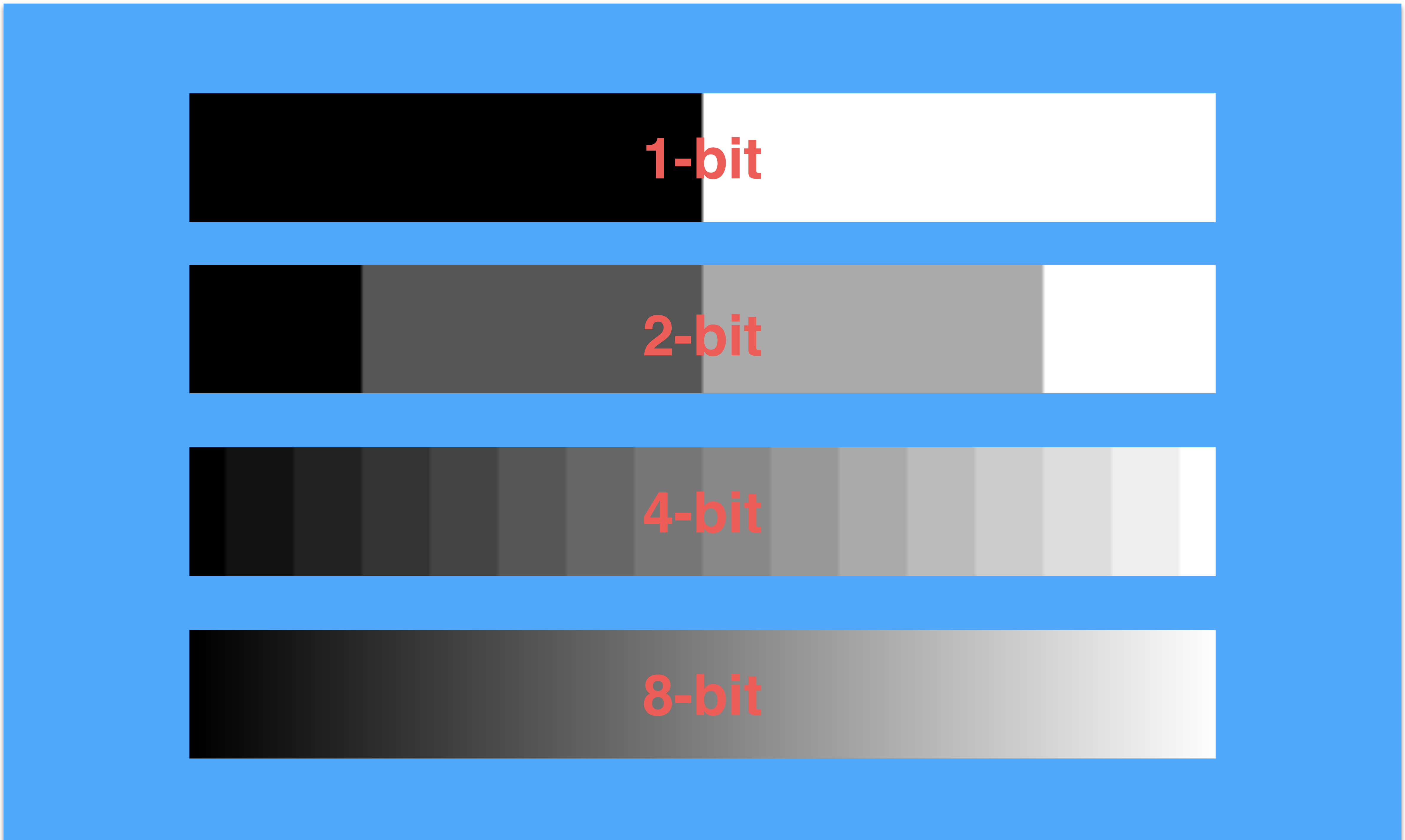
2D Images

Each square is called a pixel

2D Images

- Each pixel is a “square":
  - **Position**: (x, y)
  - **Size**: height and width —> the same for all pixels
  - **An attribute**: color (RGB) or ***intensity***:
    - Each intensity value is typically normalized in [0,1] —> integer values a different bit depth: 8-bit, 10-bit, 12-bit, 14-bit, and 16-bit.

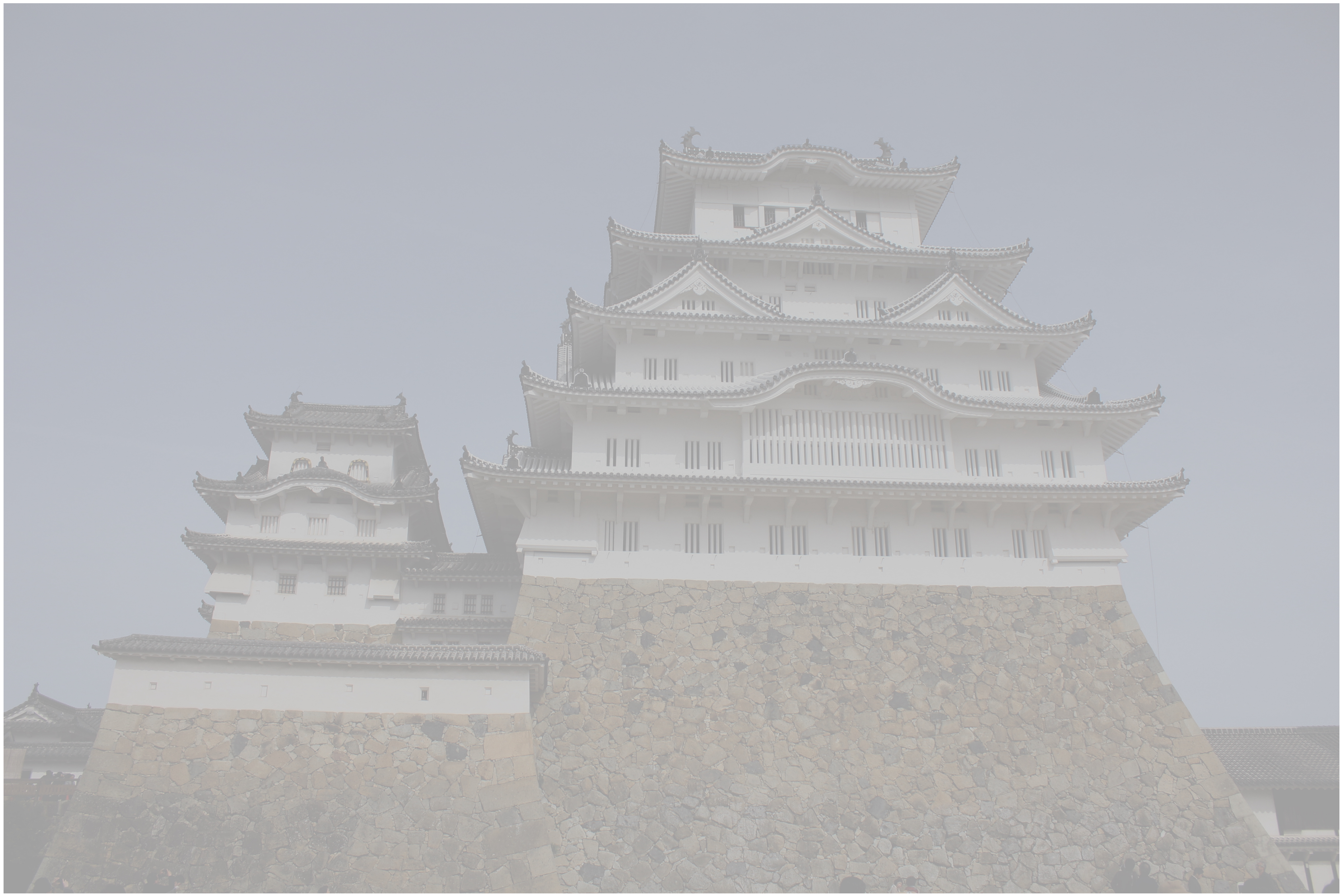
2D Images: Bit-Depth



2D Images: Contrast

- Contrast is the difference in intensity/color for making an object visible in a distinguishable way.
- If contrast is low, it is difficult to detect details
- If contrast is high, it is easier to detect different details

2D Images: Low Contrast



2D Images: High Contrast

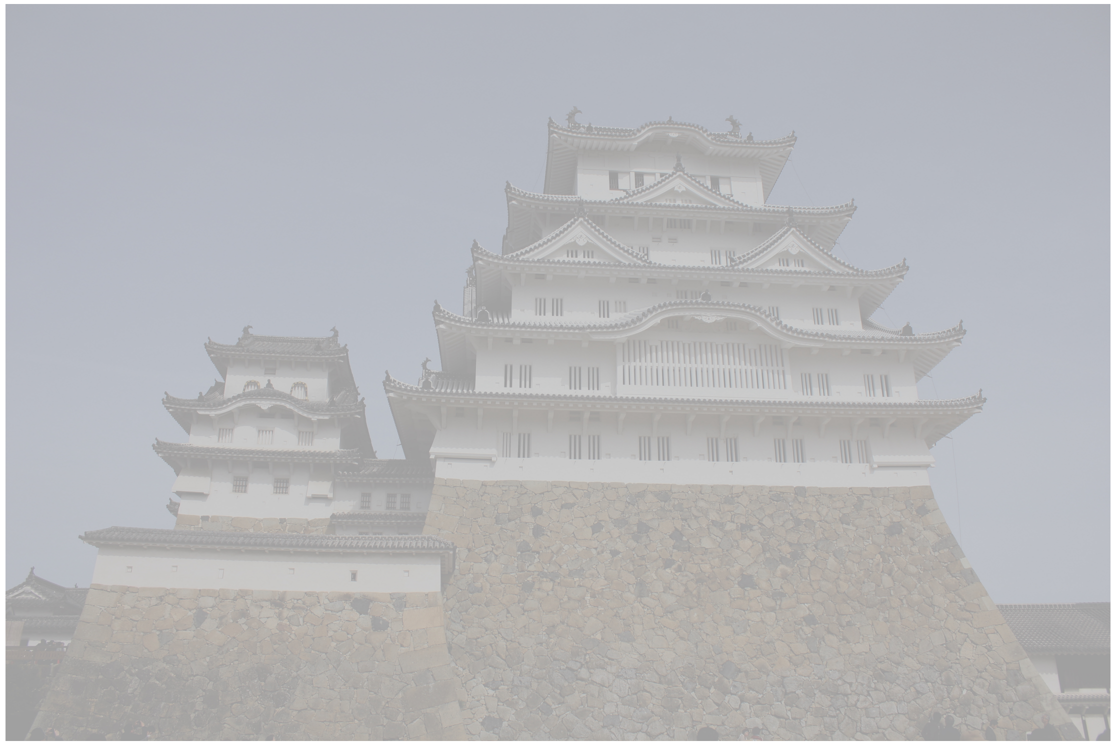
2D Images: How to Measure Contrast

$$C = max / min$$

$$C_{Weber} = (max - min) / min$$

$$C_{Michelson} = (max - min) / (max + min)$$

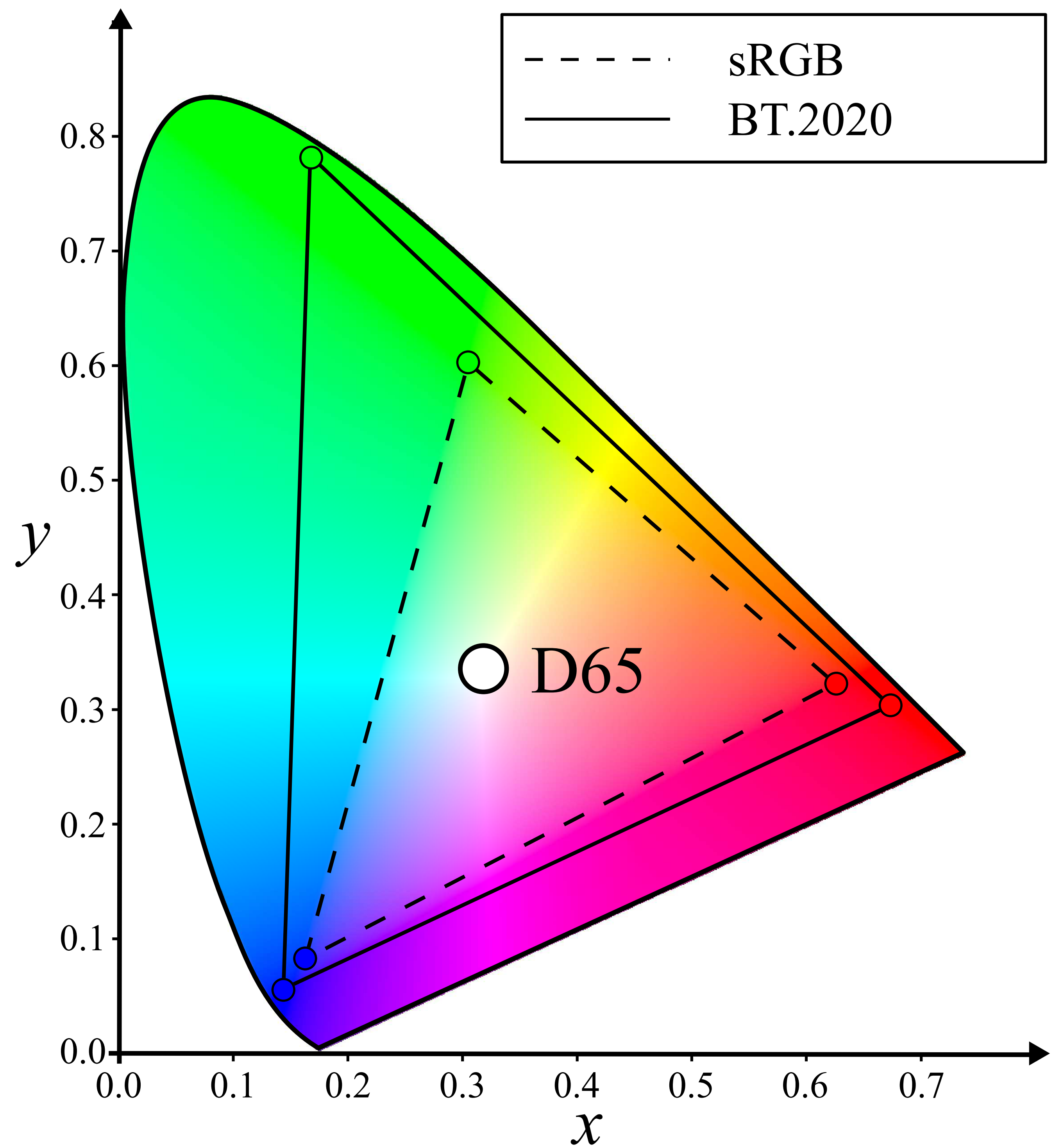
2D Images: Contrast



$C = 1.3574$
$C_{Weber} = 0.3574$
$C_{Michelson} = 0.1516$

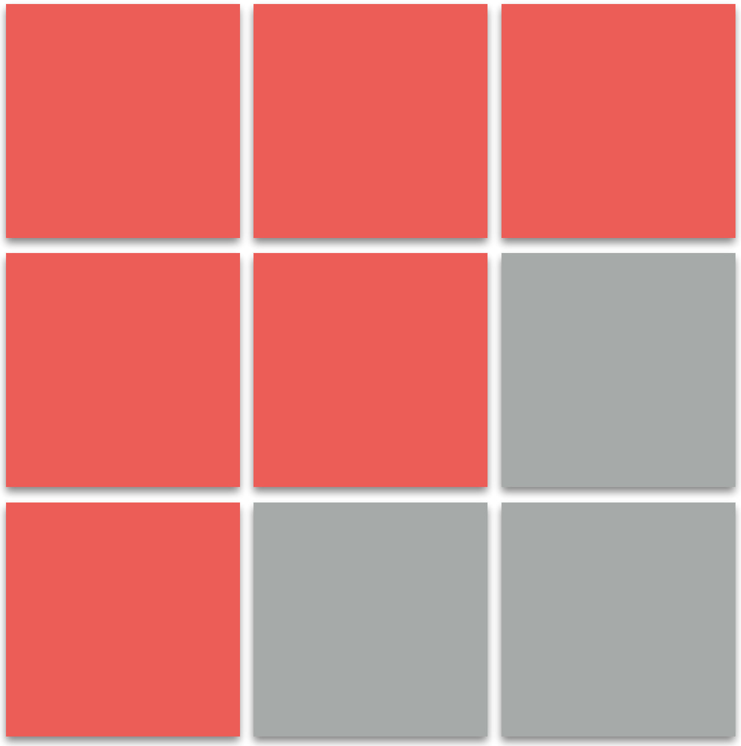
$C = 1118.4$
$C_{Weber} = 1117.4$
$C_{Michelson} = 0.9982$

2D Images: Colors



2D Images

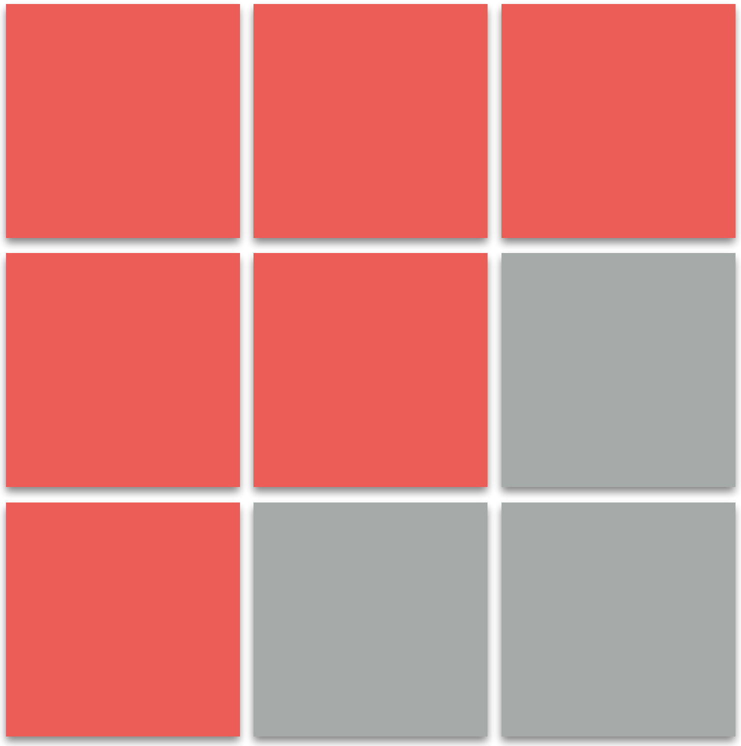
- A 2D image is a graph:



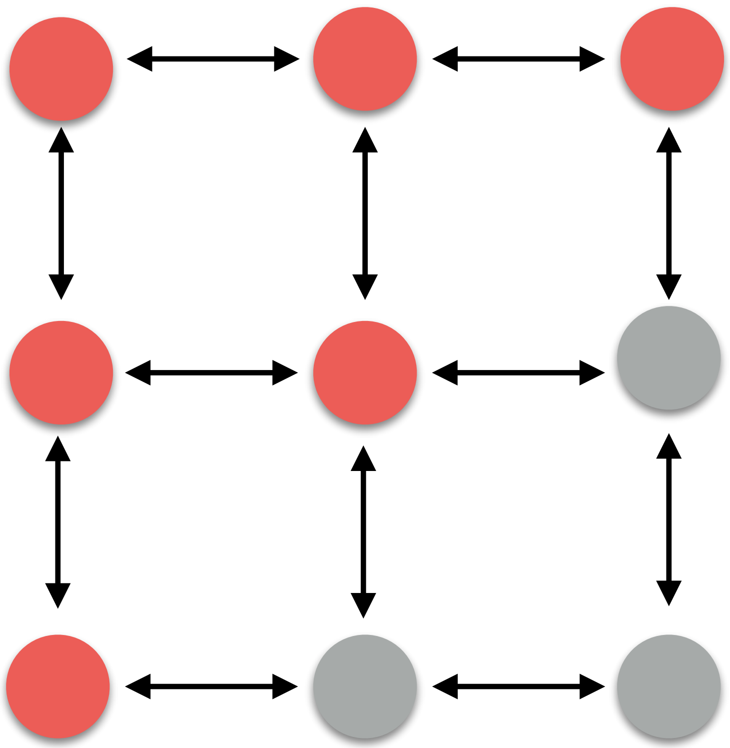
2D Image, 3x3 pixels

2D Images

- A 2D image is a graph:



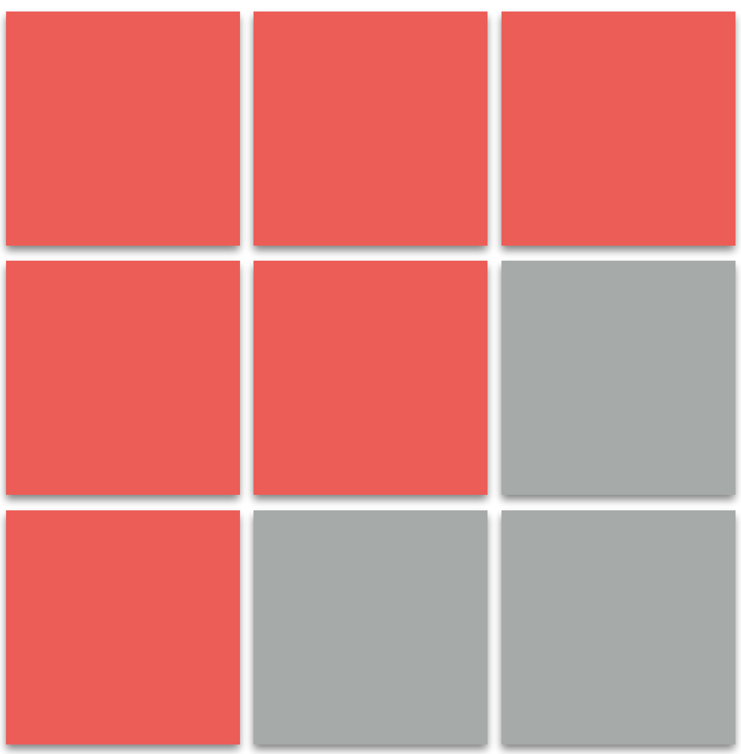
2D Image, 3x3 pixels



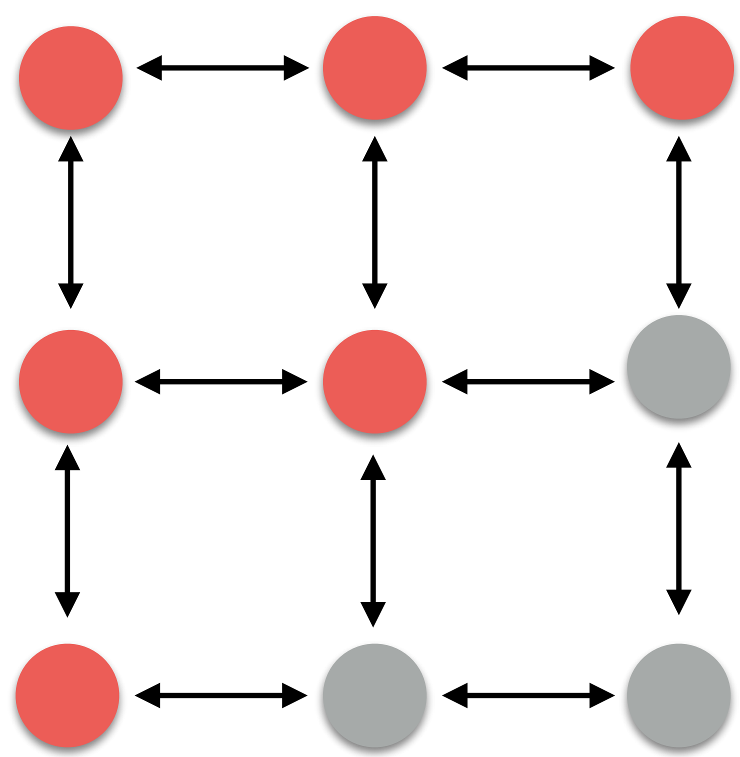
4-connected pixel
adjacency graph

2D Images

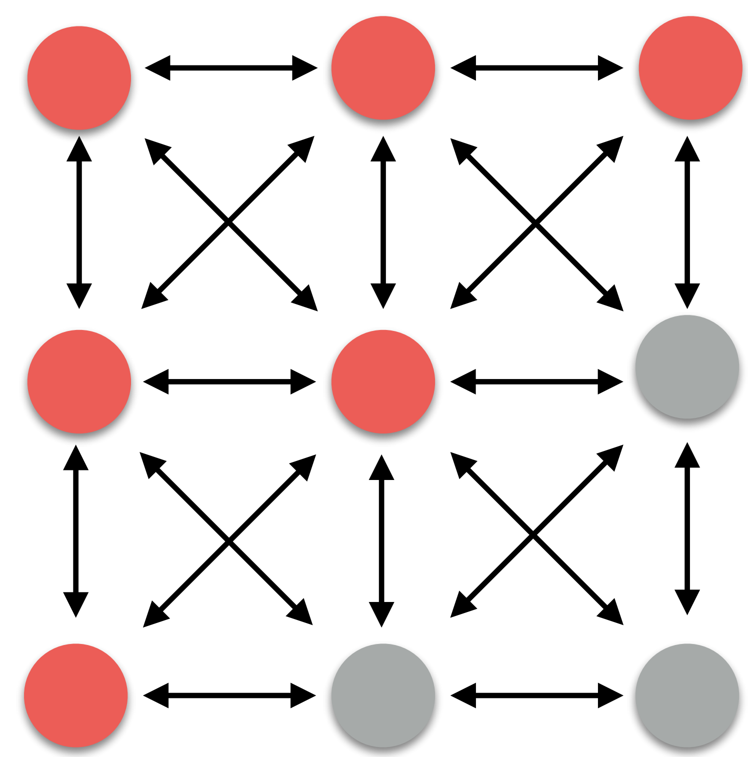
- A 2D image is a graph:



2D Image, 3x3 pixels



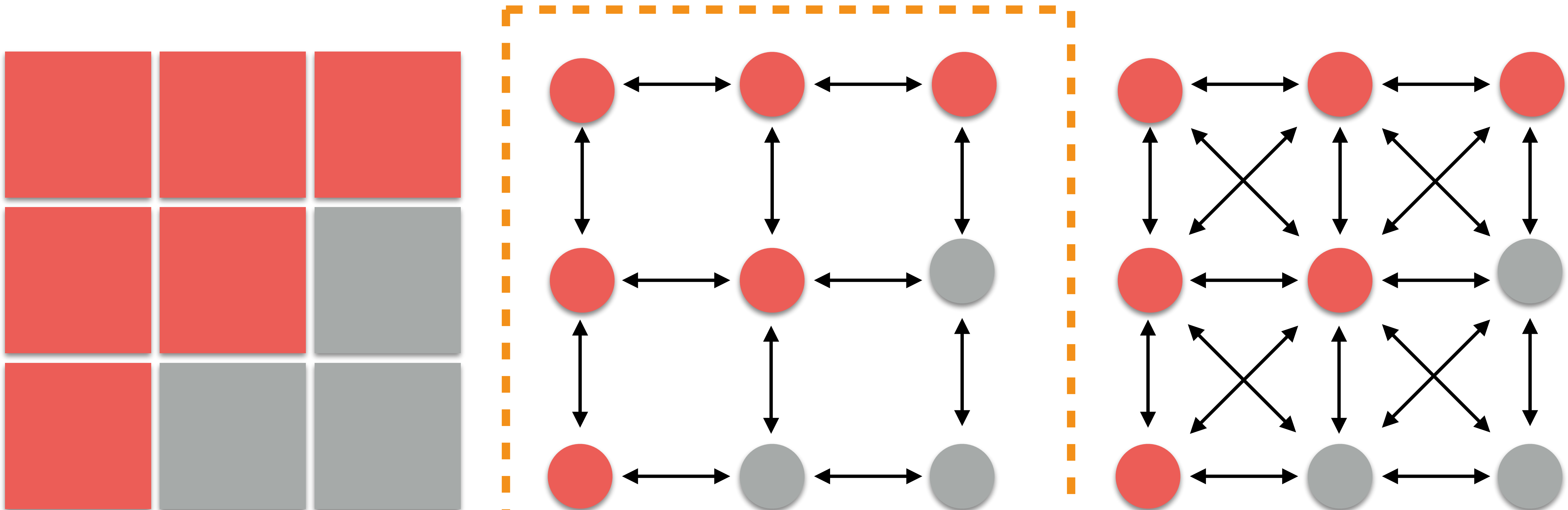
4-connected pixel adjacency graph



8-connected pixel adjacency graph

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

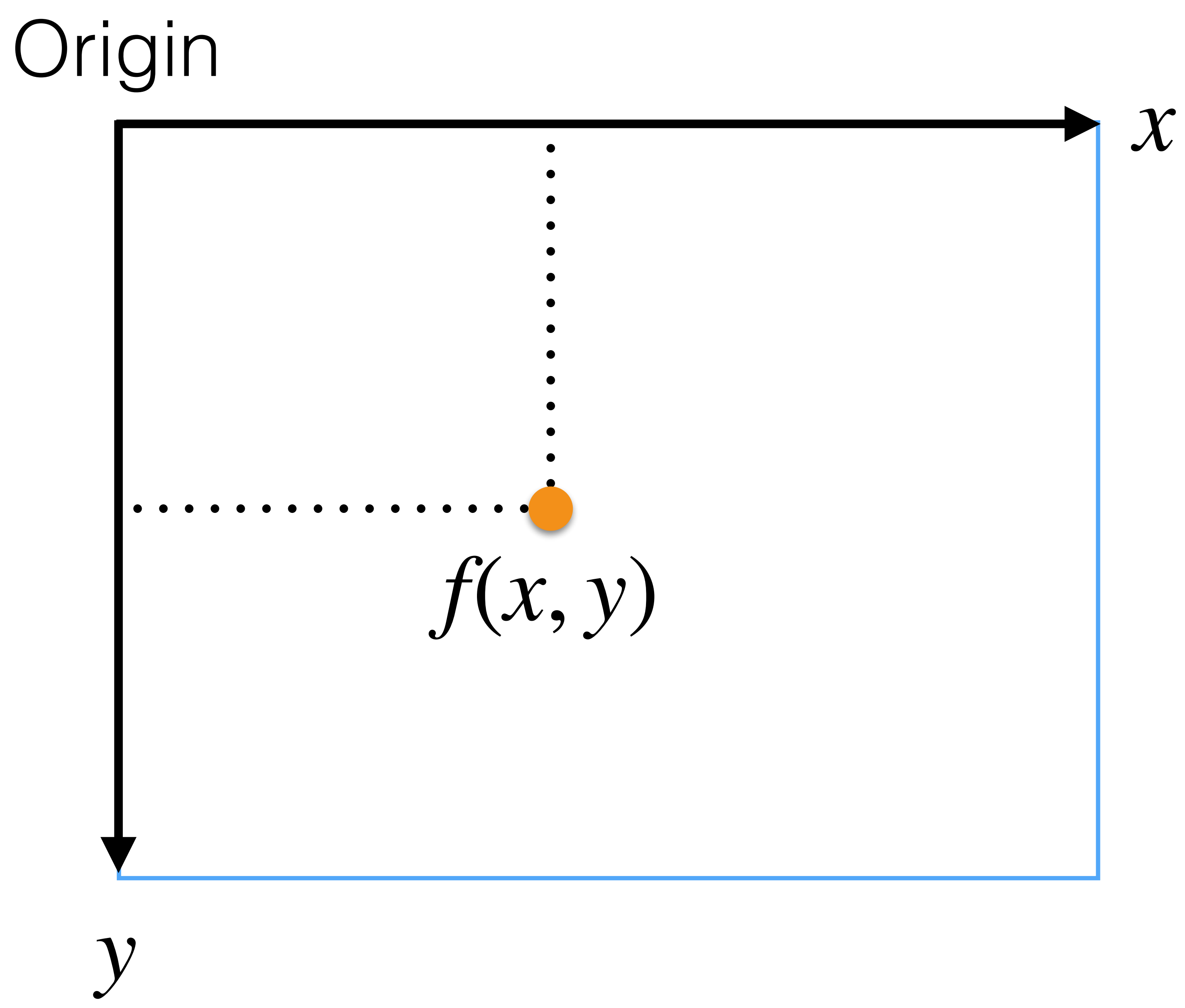


Image Coordinate System: MATLAB

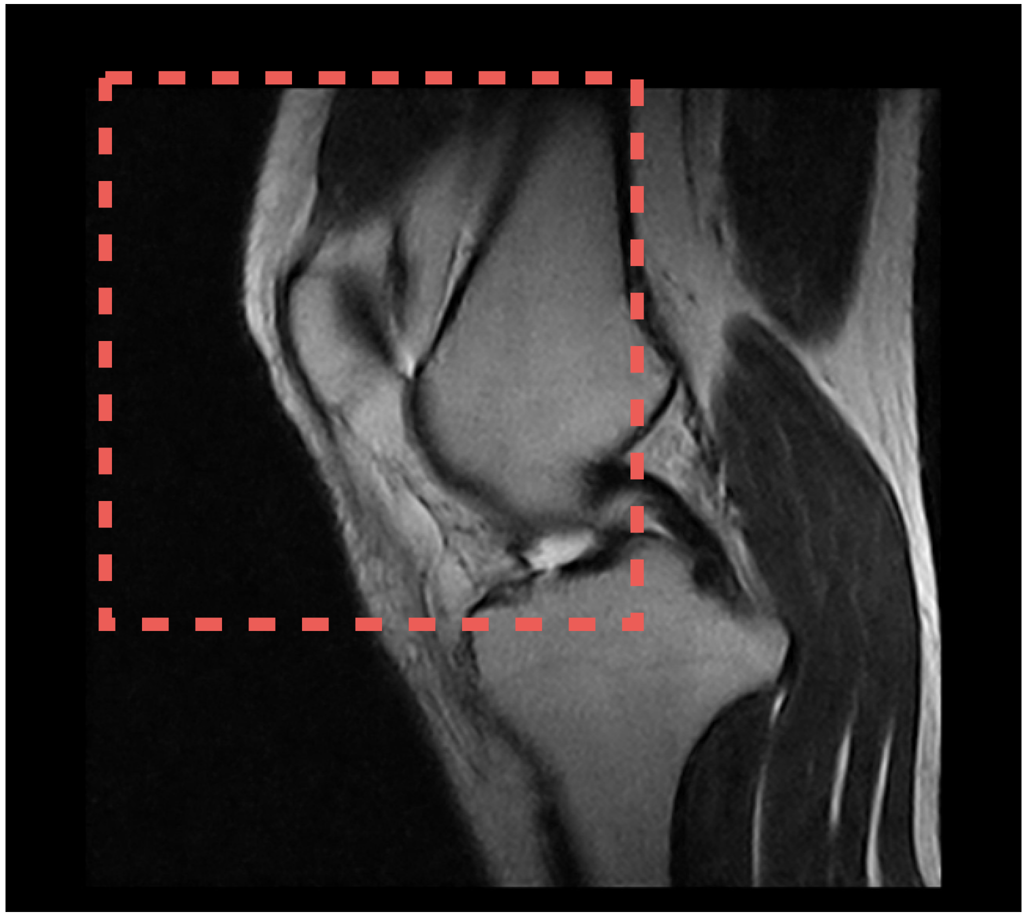
- MATLAB origin —> (1,1)
- Given an image, **img**, as m-n matrix to access:

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

- where m is the height of the image, and n is the width of it

Region Of Interest (ROI)

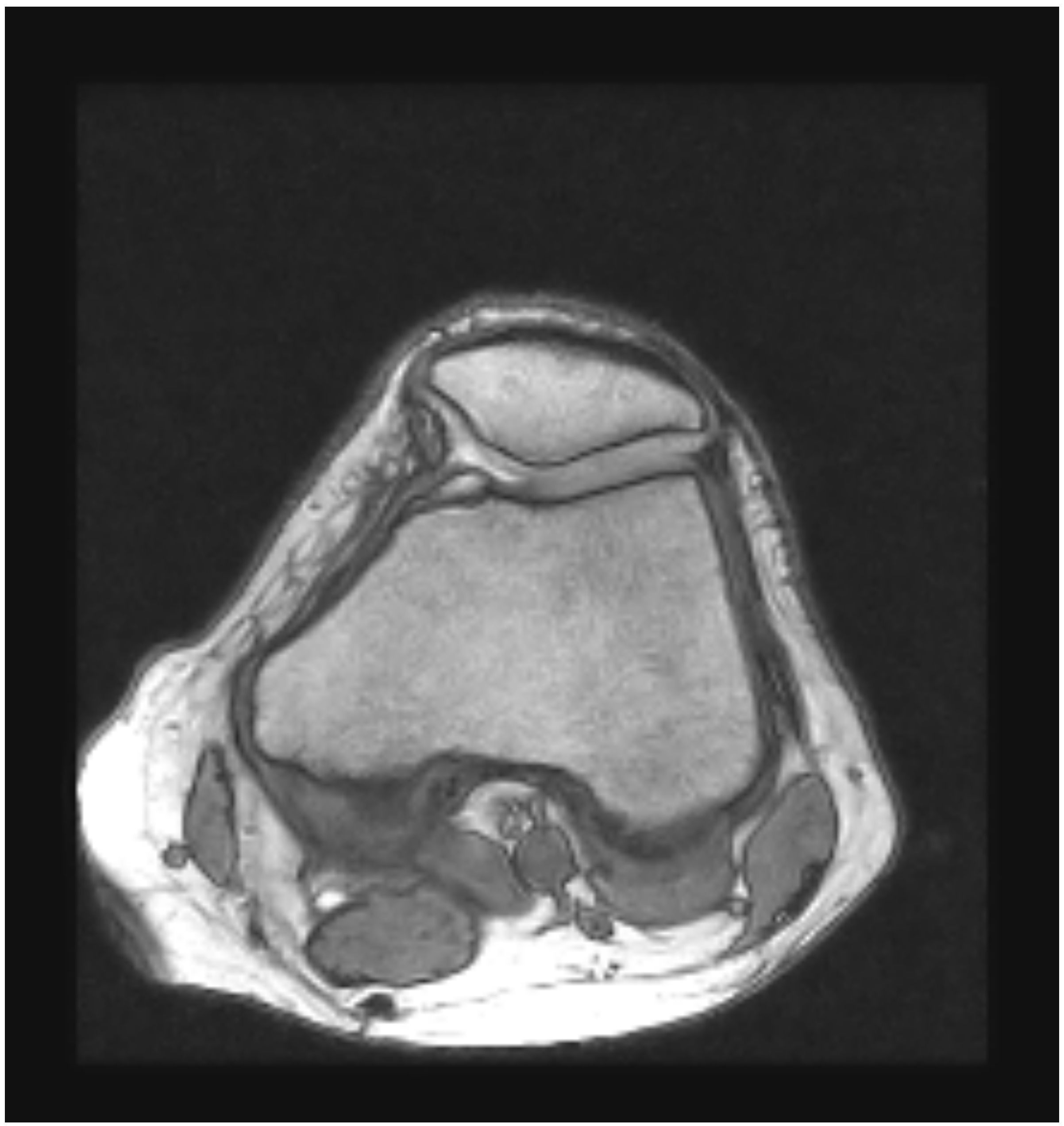
- We may be interested to process not the full image/volume but an area/volume.
- This area is typically called region of interest (**ROI**).

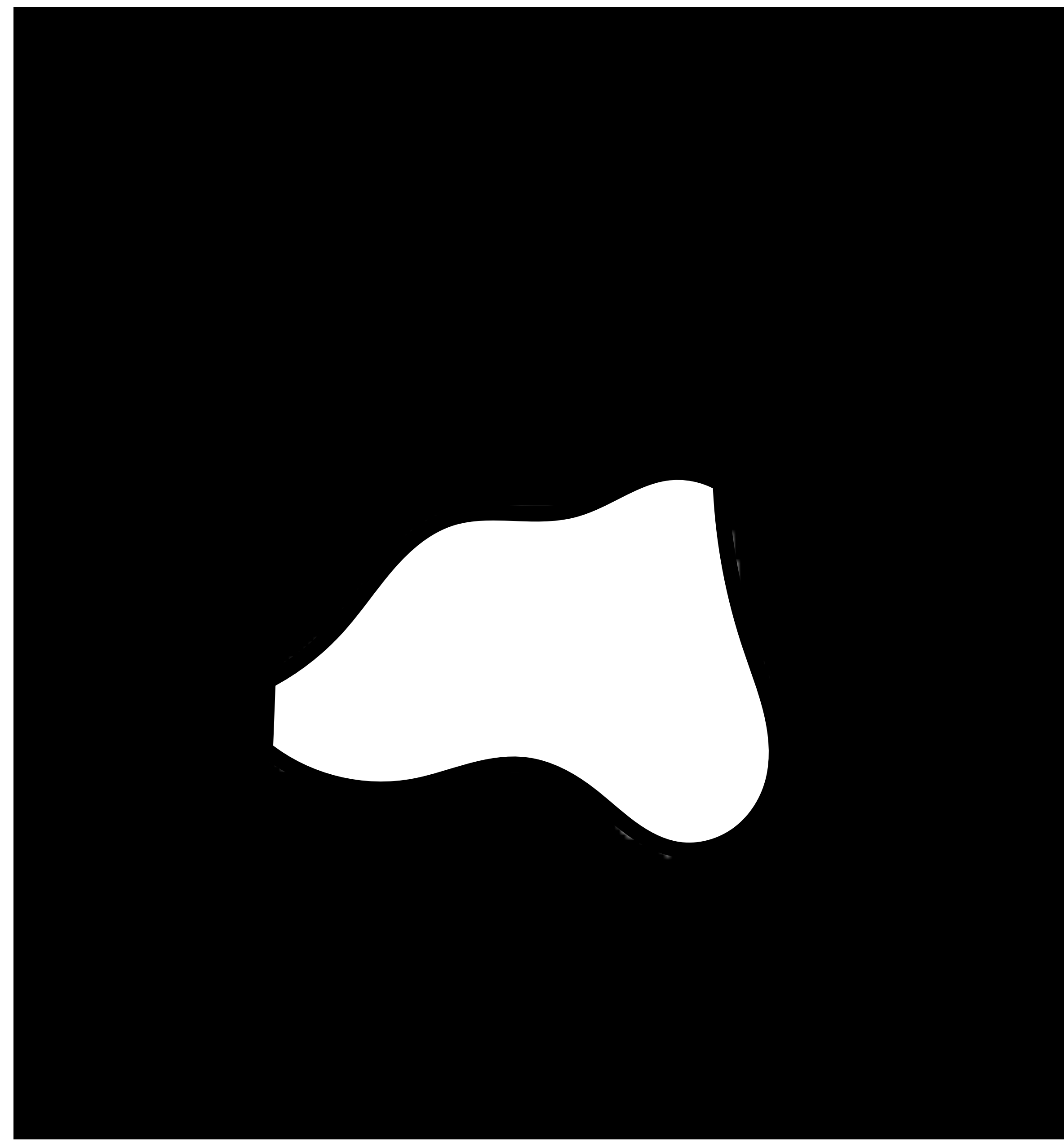


Region Of Interest (ROI)

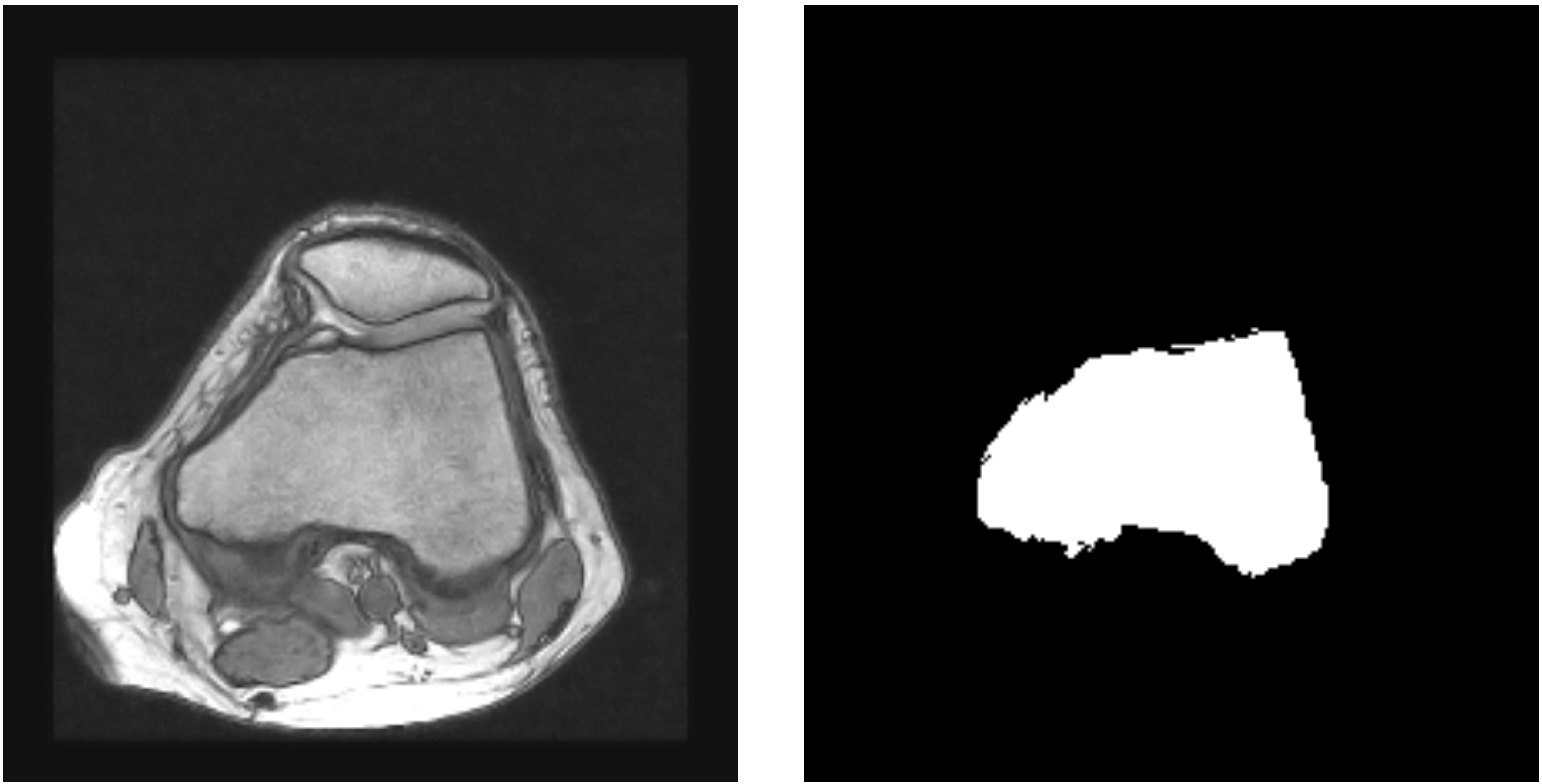
- A ROI can be defined as:
  - **Parametric region**: rectangles, circles, polygons, etc.
  - **Mask**: a binary image where:
    - 0 —> no interest area
    - 1 —> interest area

ROI: Polygons Example





ROI: Mask Example



Medical Images

Medical Images

- Main sources:
  - CAT
  - MRI
  - Ultrasound

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:

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

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

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:

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

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

Signal Damping

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: SNR

- Given a ROI i, a definition of signal-to-noise ratio (SNR) is:

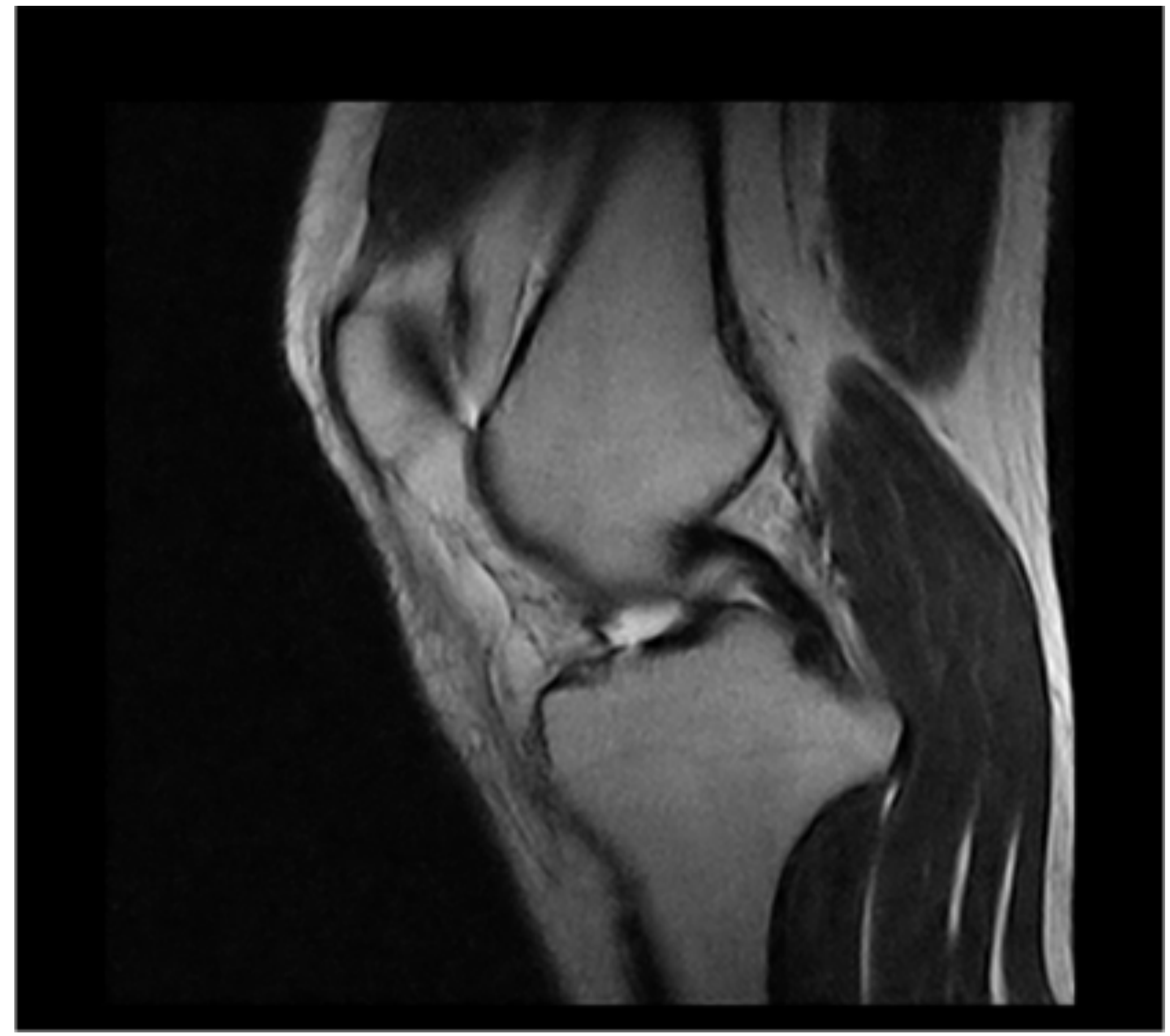
$$\text{SNR} = \frac{\mu_i}{\sigma_i}$$

- where μ_i is the mean of the signal in i, and σ_i is the standard deviation of the signal in the background.

- To have an estimate of noise, we compute σ_i in the background of the image (i.e., low intensity values) assuming that noise does not vary in different ROIs

Noise Measure: Example

$SNR_{db} = 3.43$



$SNR_{db} = 29.26$

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**: bit-depth, pixel's size (mm), thickness between slices (mm), 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:

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

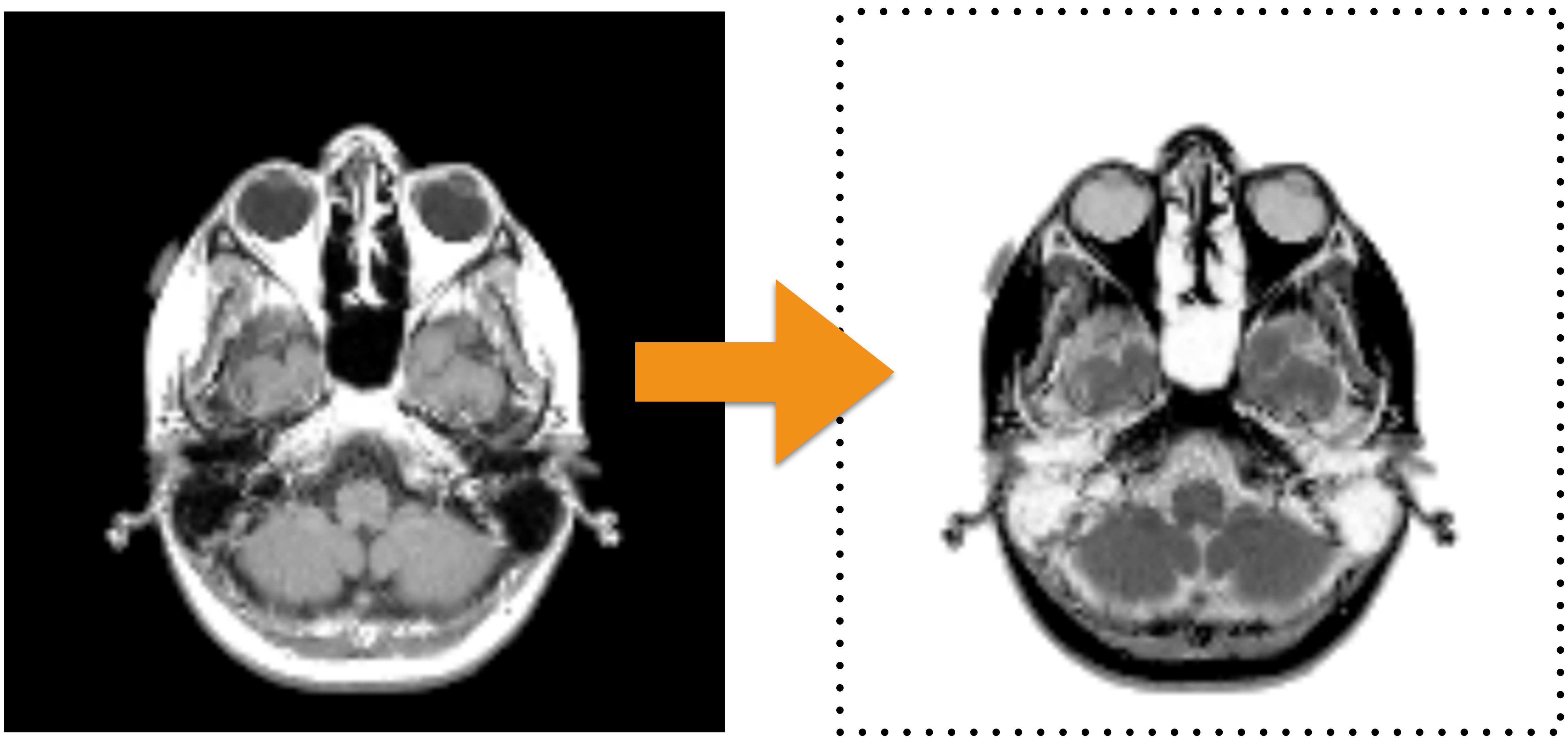
Unary Operators: Negative

- 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



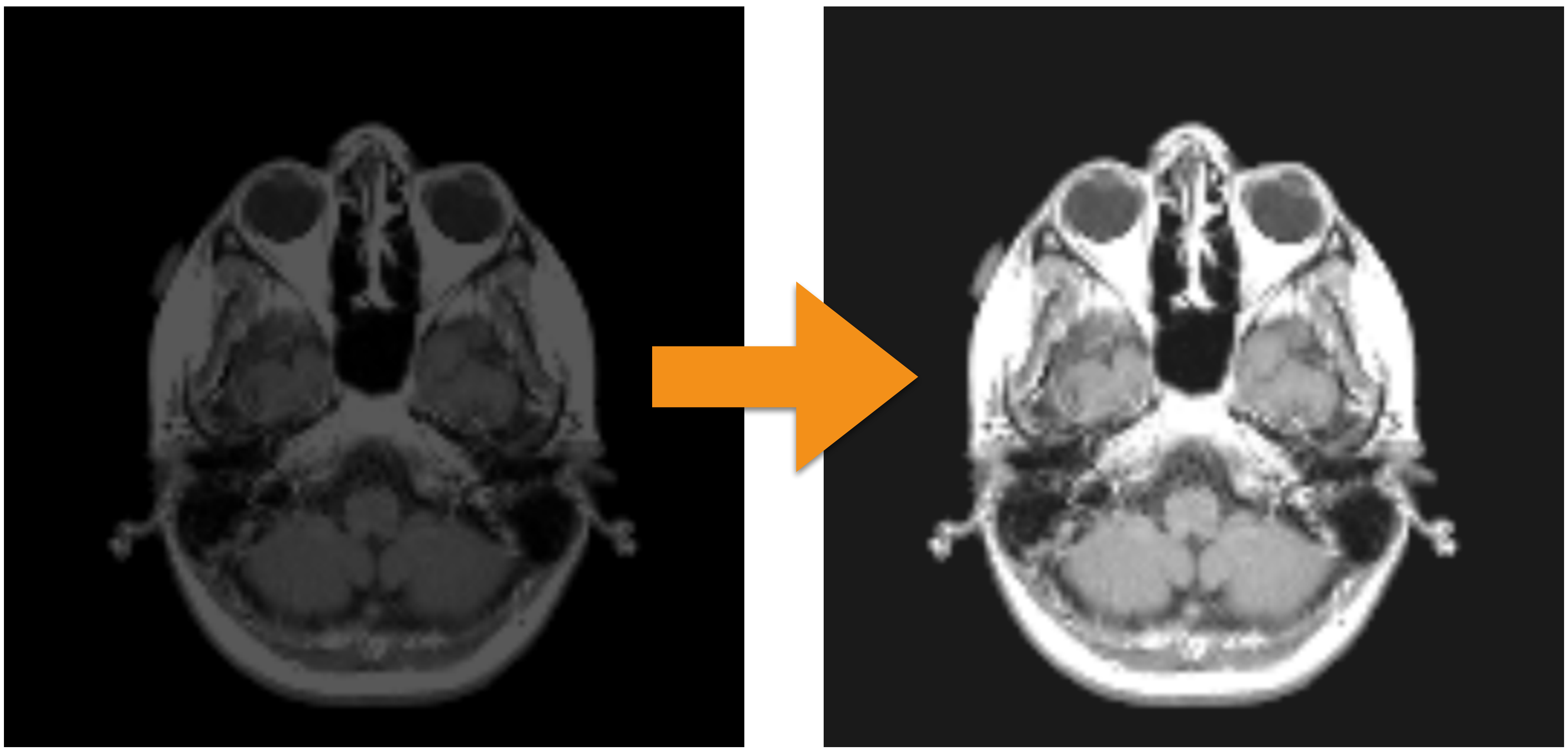
Unary Operators: Contrast Stretching

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

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

- It is useful when the contrast is low.

Contrast Stretching Example



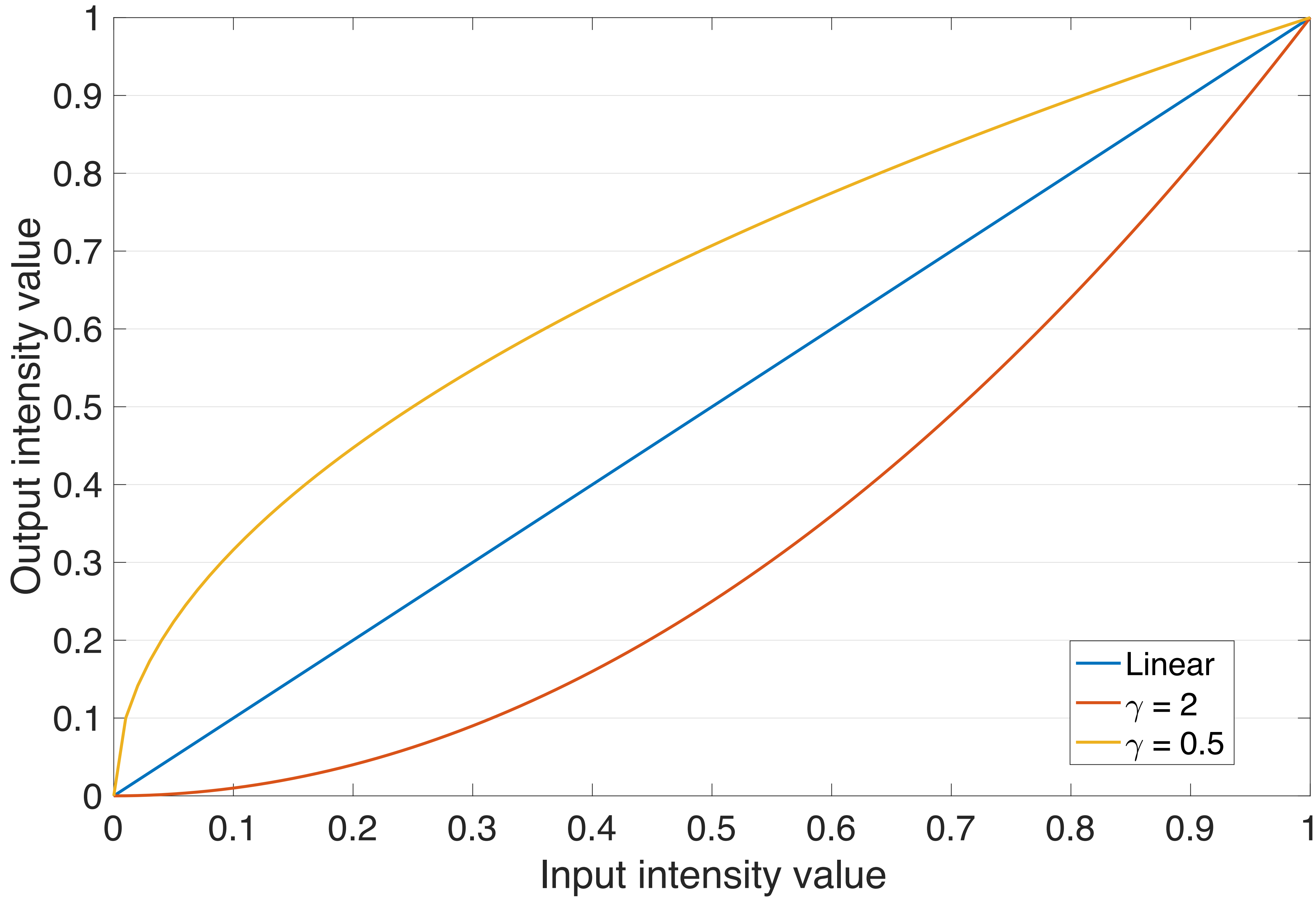
Unary Operators: Gamma

- Another method for increasing the dynamic range:

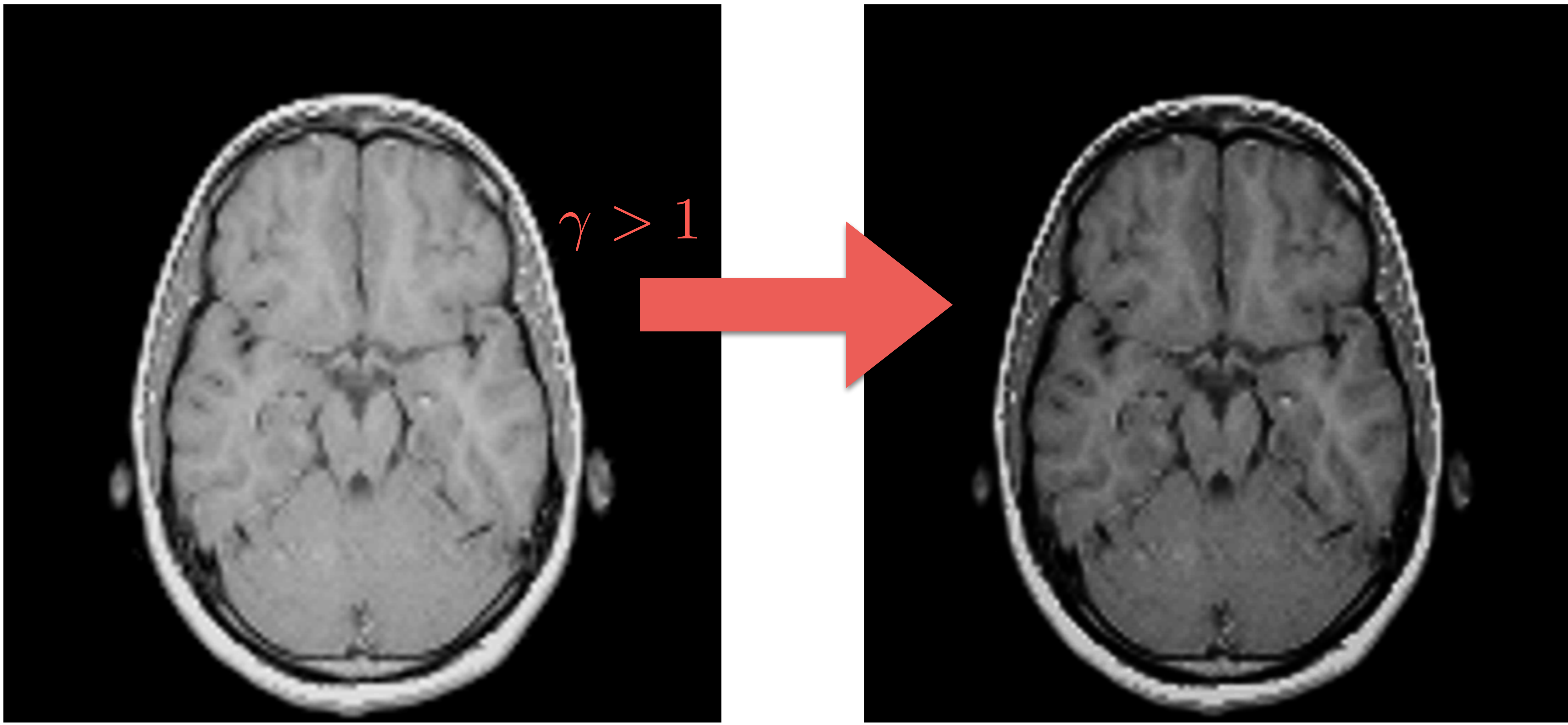
$$g(x,y) = \mathrm{G}\big[f(x,y); k; \gamma\big] =$$
$$= k \cdot f(x,y)^{\gamma}$$

- It is more intuitive.

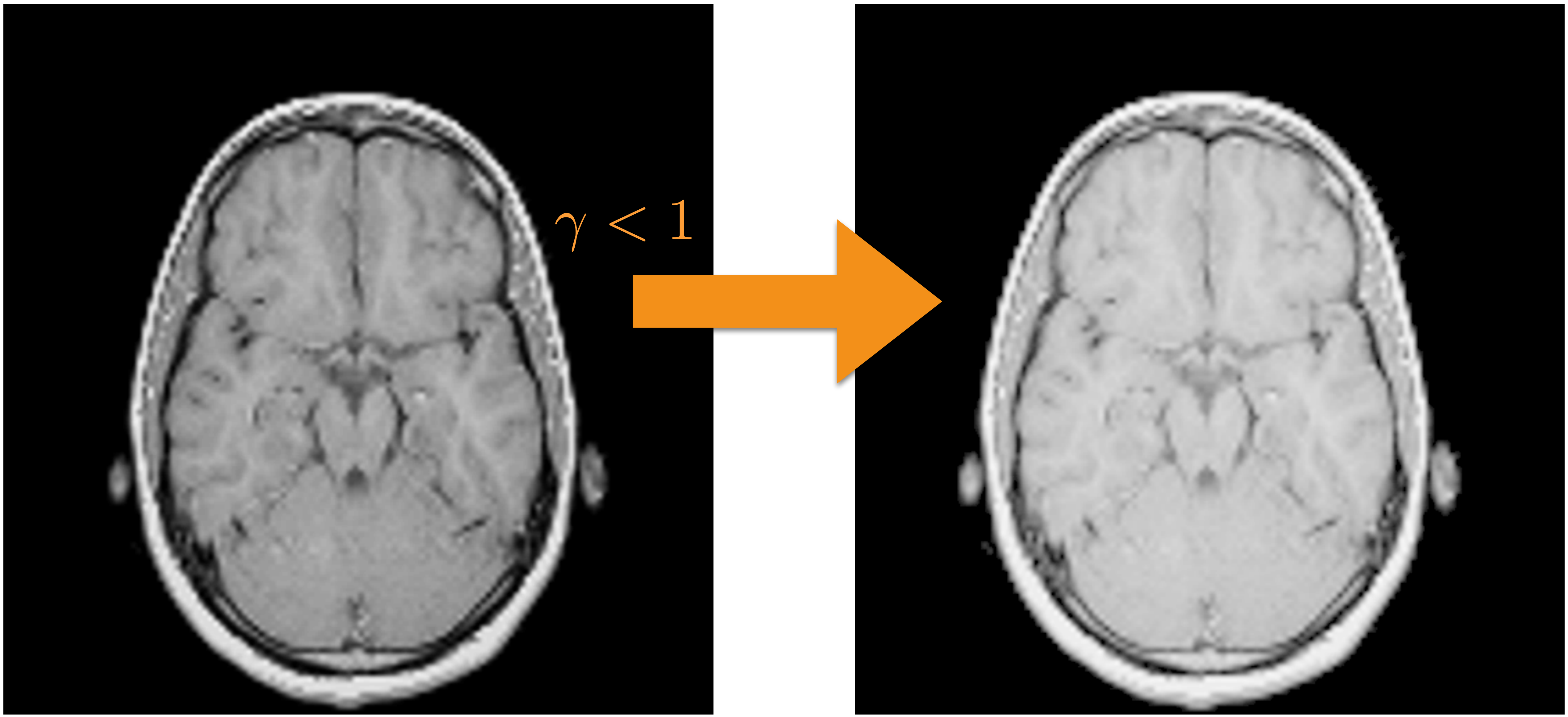
Unary Operators: Gamma



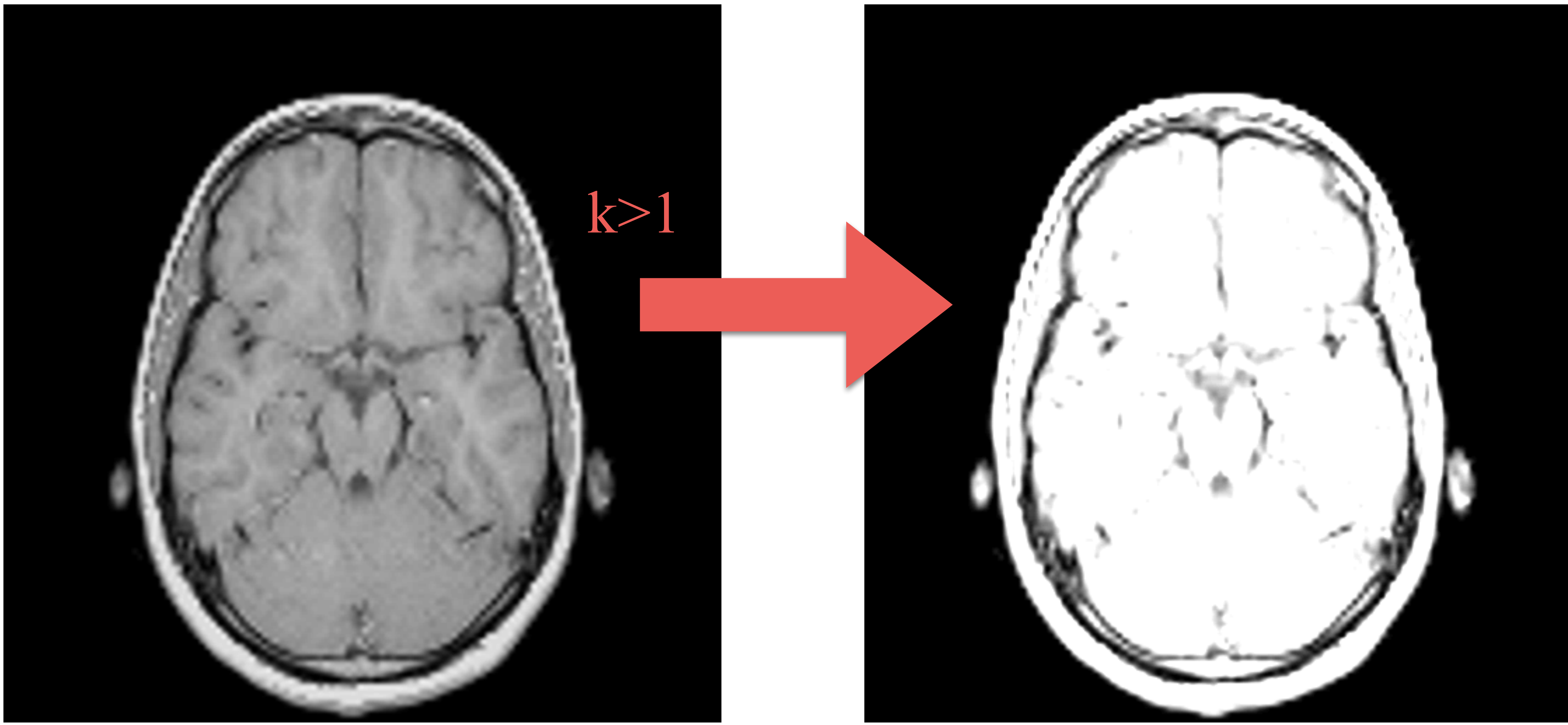
Gamma Example



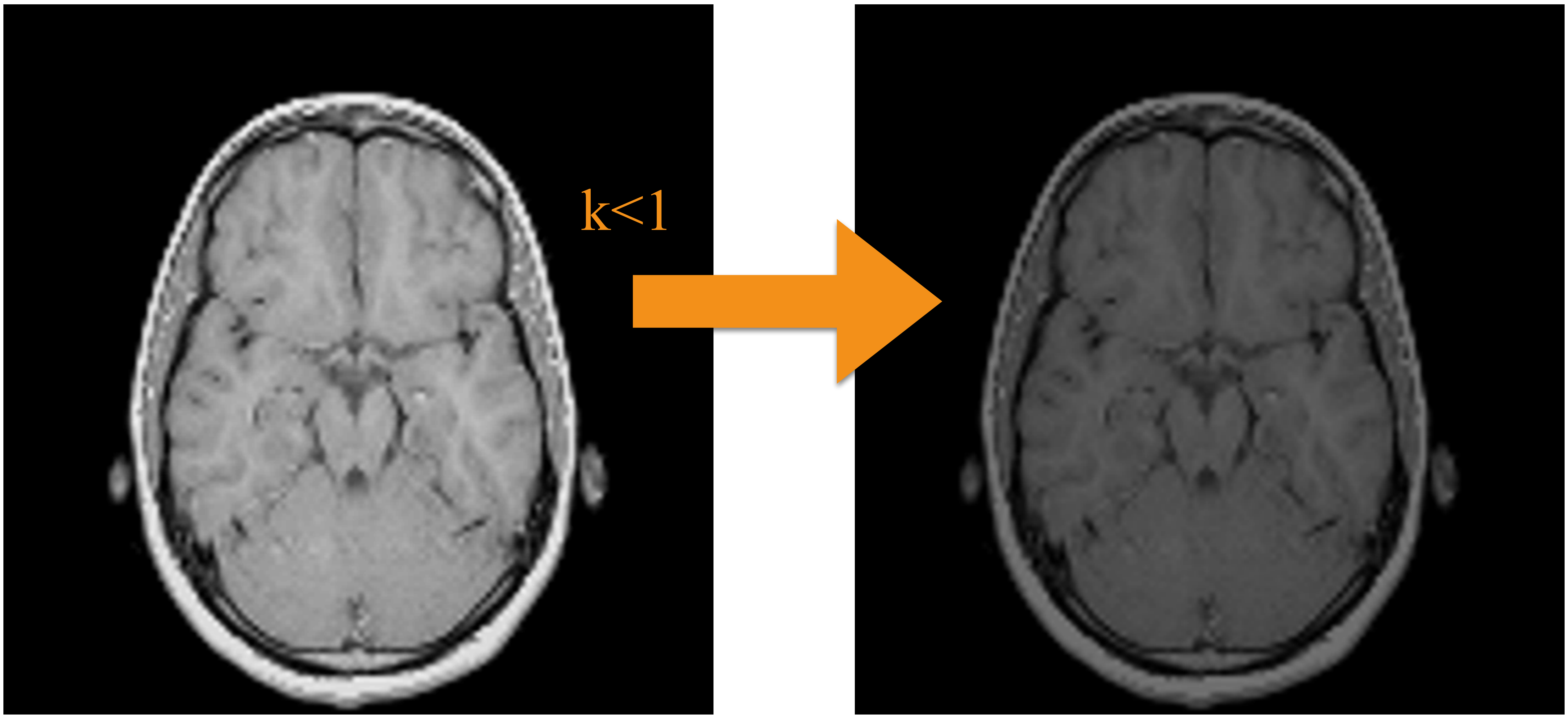
Gamma Example



k Example



k Example



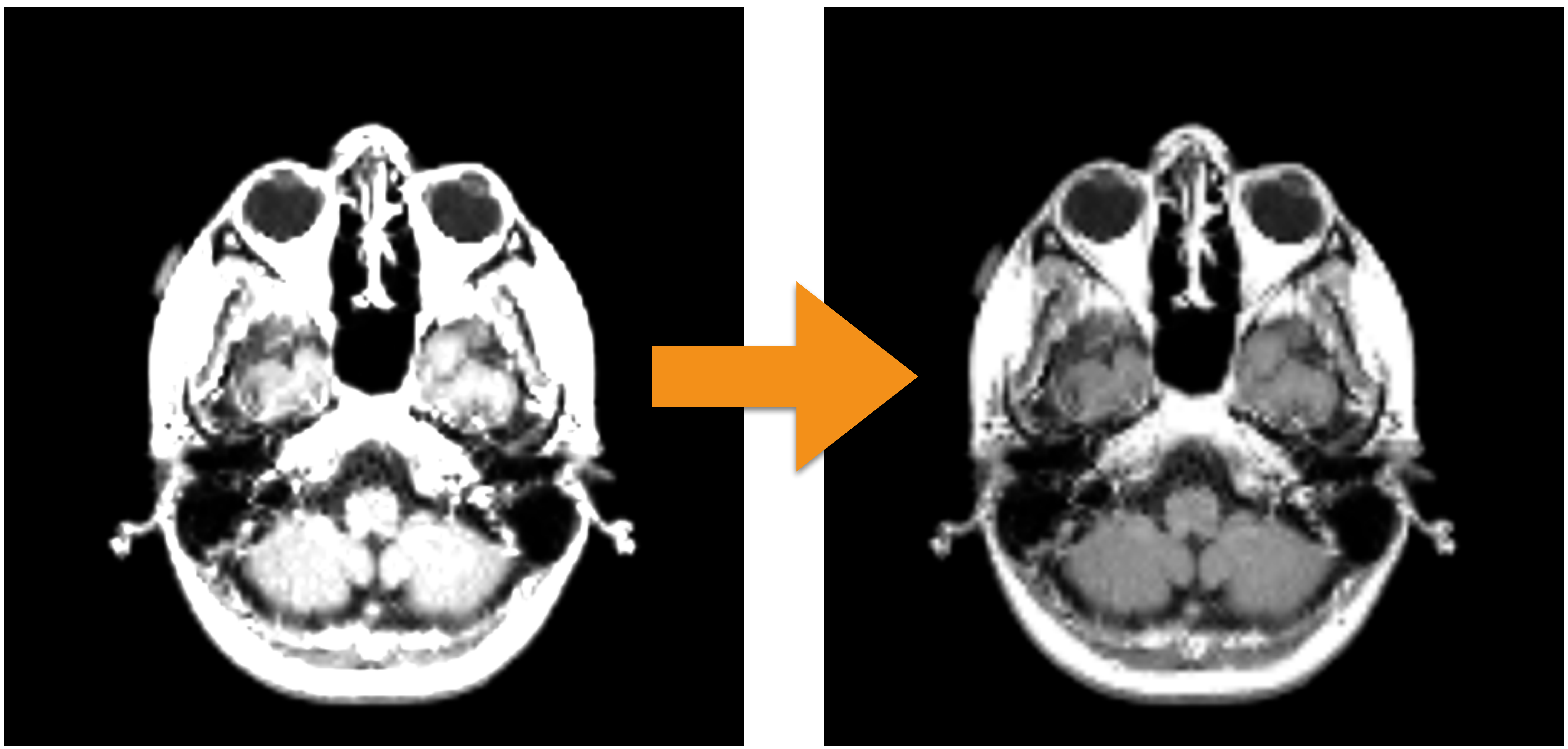
Unary Operators: Logarithmic Operator

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

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

$$= (E_{\max} - E_{\min}) \cdot \frac{\log(1 + f(x,y))}{\log(1 + \max(f))} + E_{\min}$$

Logarithmic Example



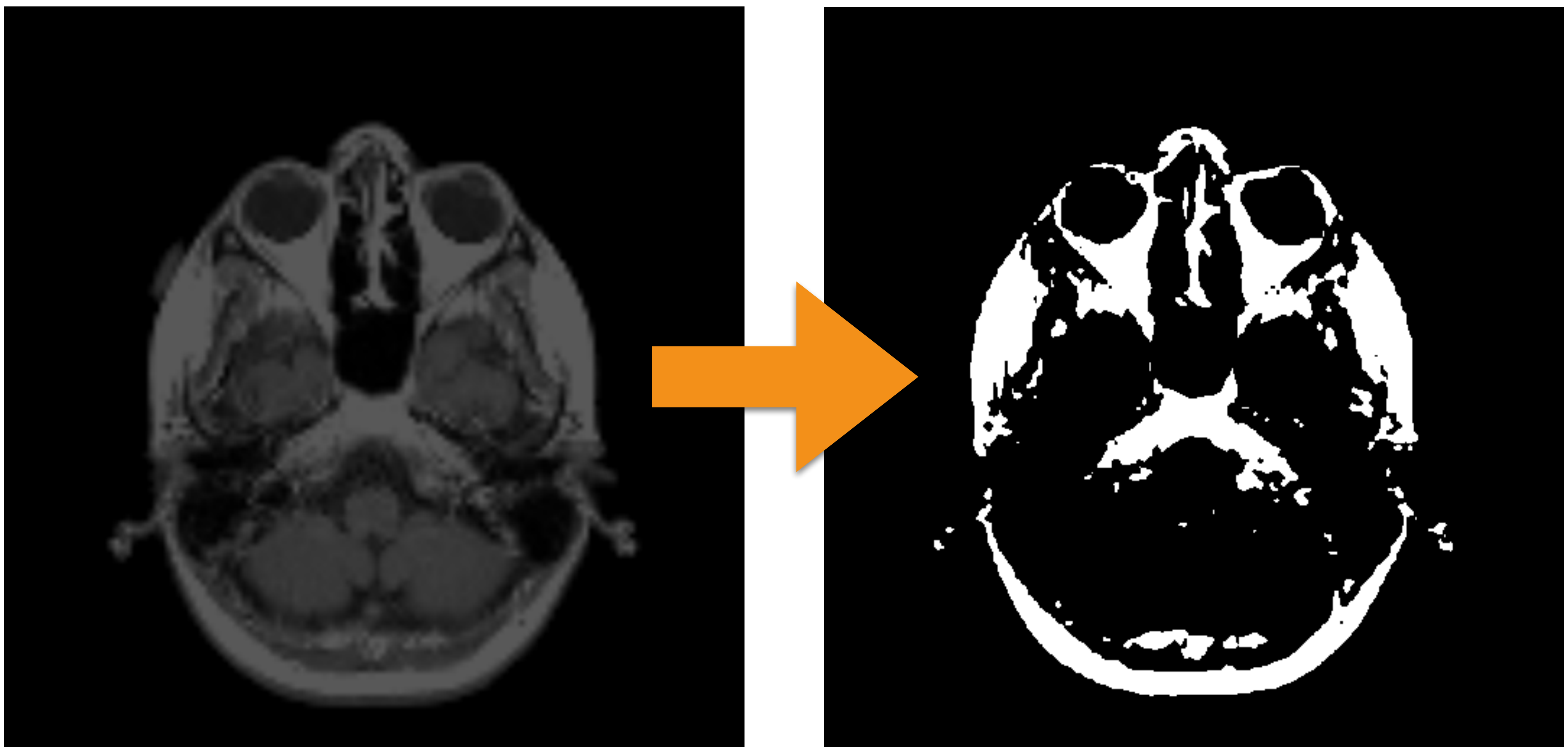
Unary Operators: Thresholding

- 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



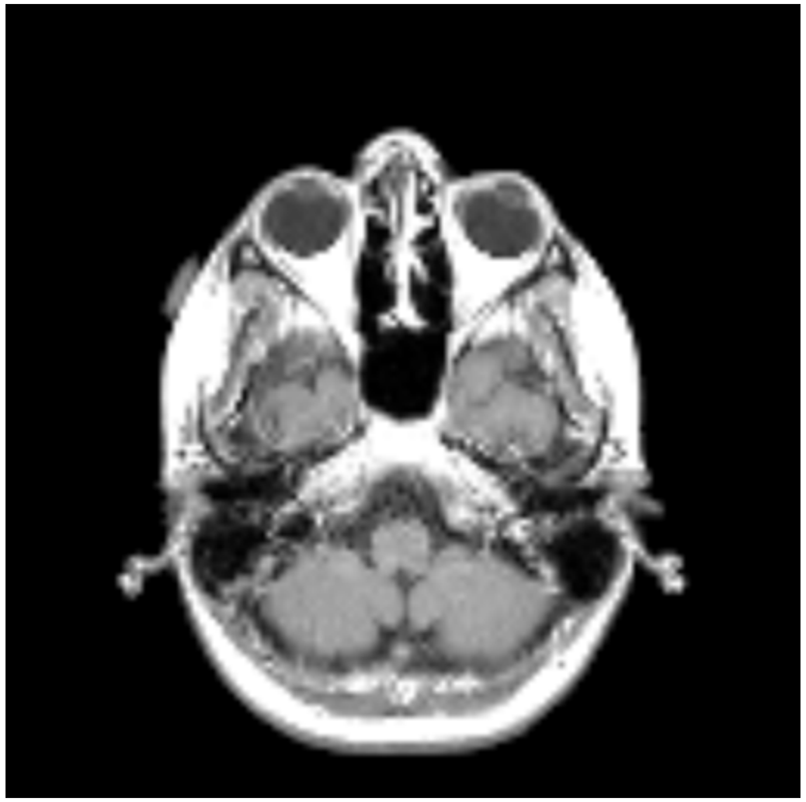
Binary Operators

- Binary operators are typically the classic arithmetic operators defined over images:
  - +, -, *, /
- Note that using +, -, and /, our dynamic range is not anymore in the range [0,1] (it can be negative!):
  - Linear scaling in [0,1]
  - Logarithmic operator

Histograms

Image Histogram

- A histogram H is the distribution of intensity values of pixels.



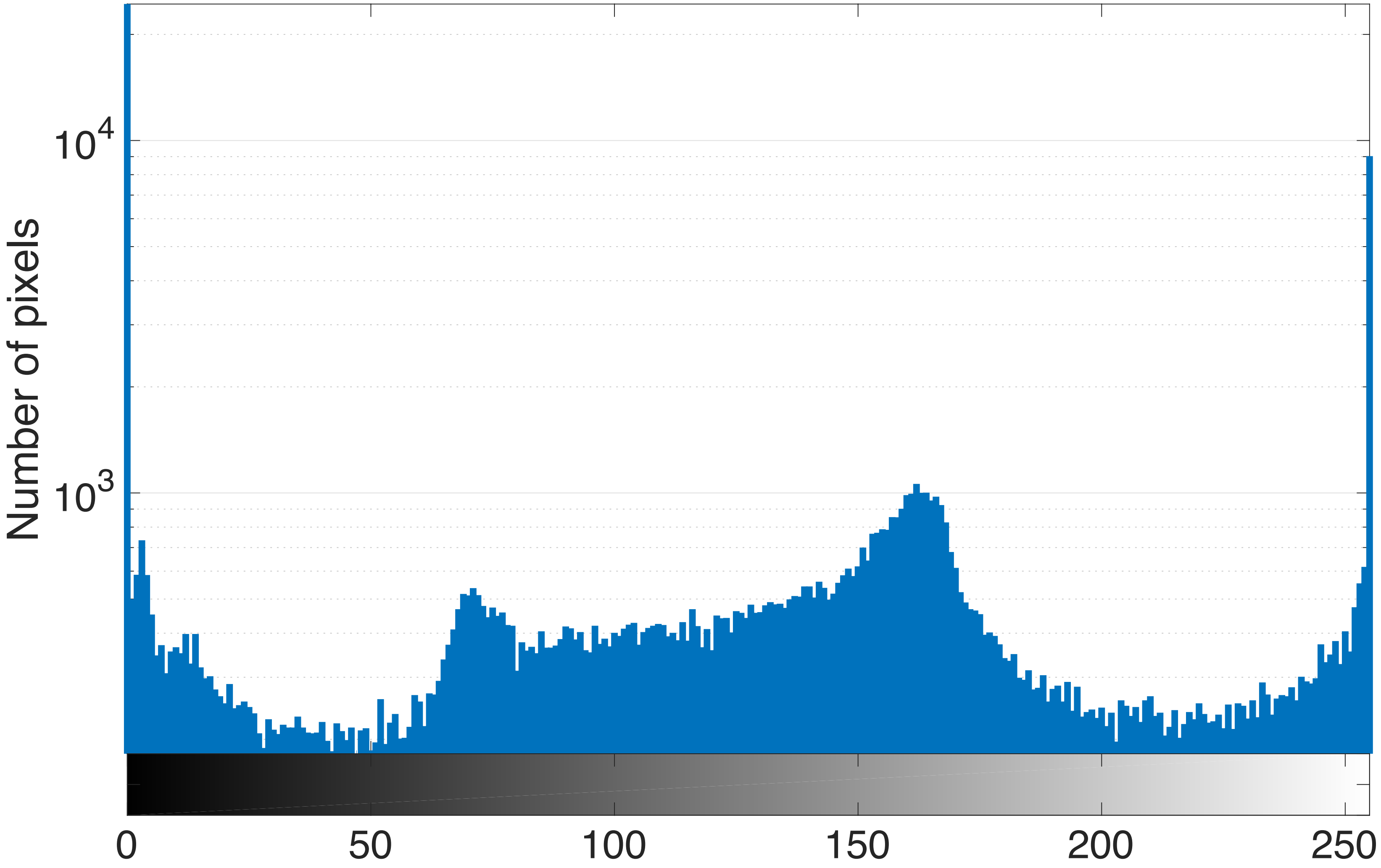


Image Histogram

- How do we compute it?
  - We divide the range of values into N bins
  - For each bin we count the number of pixels whose intensity values are in the range of that bin
  - MATLAB: **imhist** built-in function

Image Histogram: Example

12	16	16	12
12	12	16	12
4	8	16	8
4	4	8	8

Image Histogram: Example

12	16	16	12
12	12	16	12
4	8	16	8
4	4	8	8

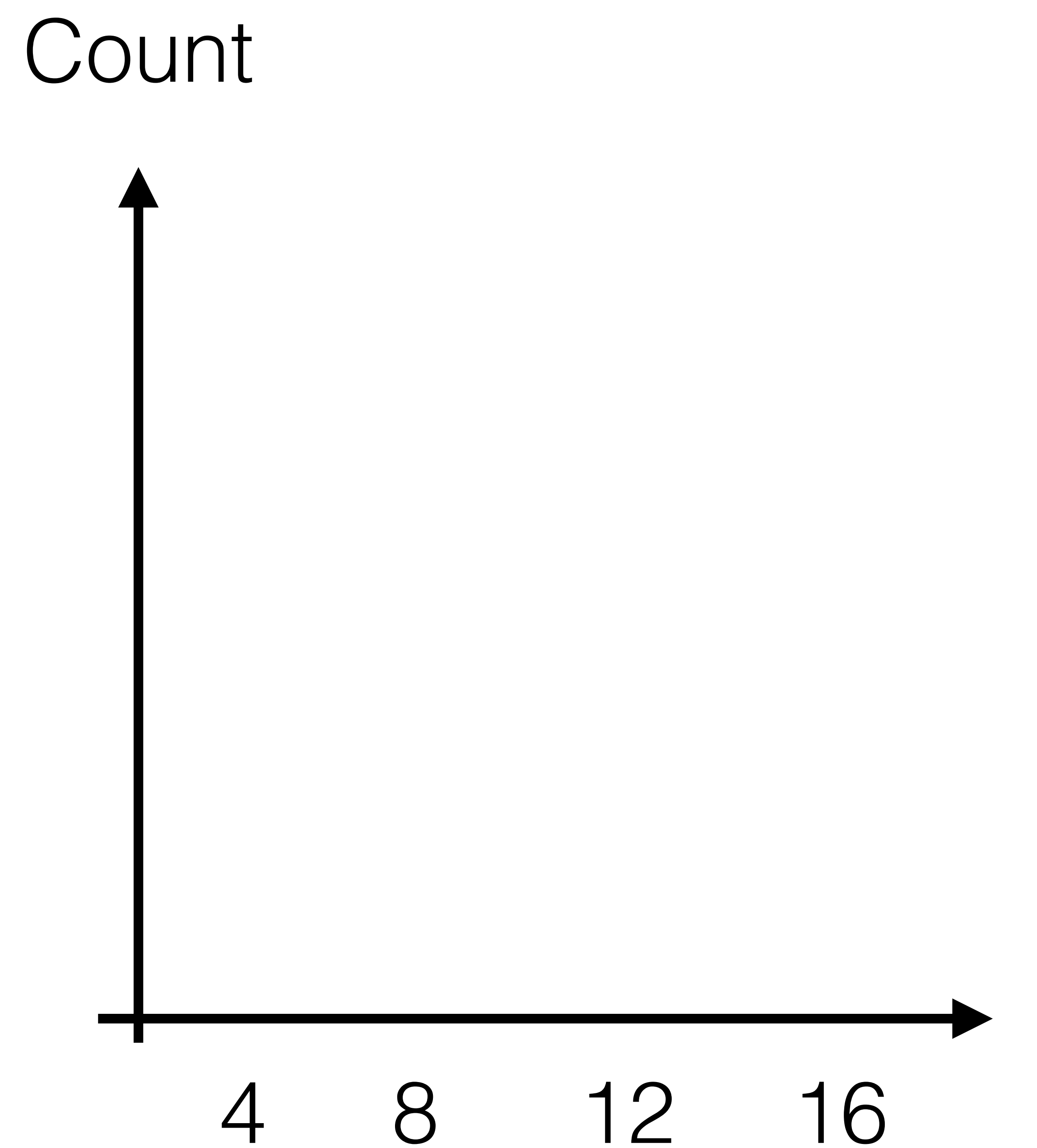


Image Histogram: Example

12	16	16	12
12	12	16	12
4	8	16	8
4	4	8	8

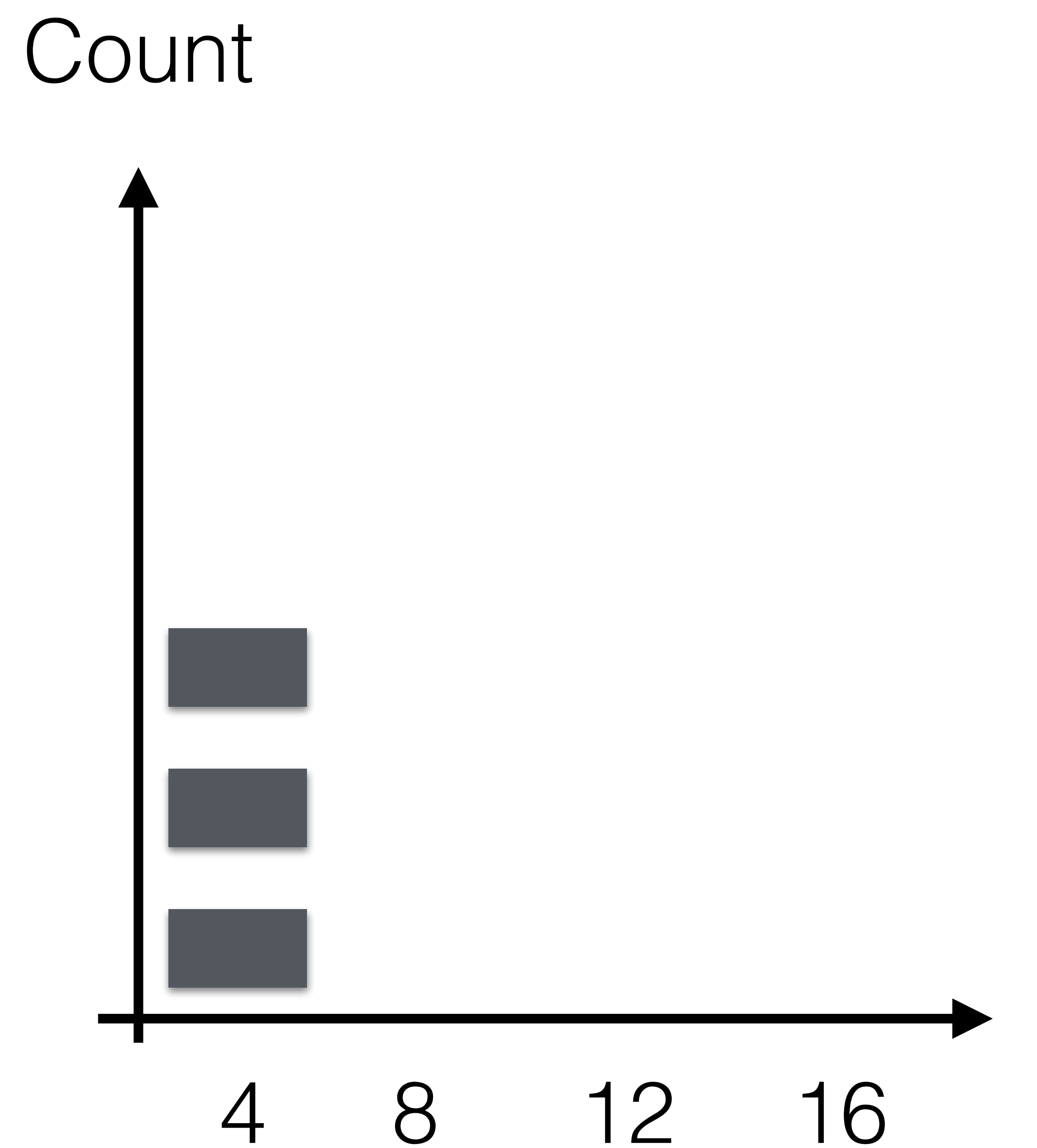


Image Histogram: Example

12	16	16	12
12	12	16	12
4	8	16	8
4	4	8	8

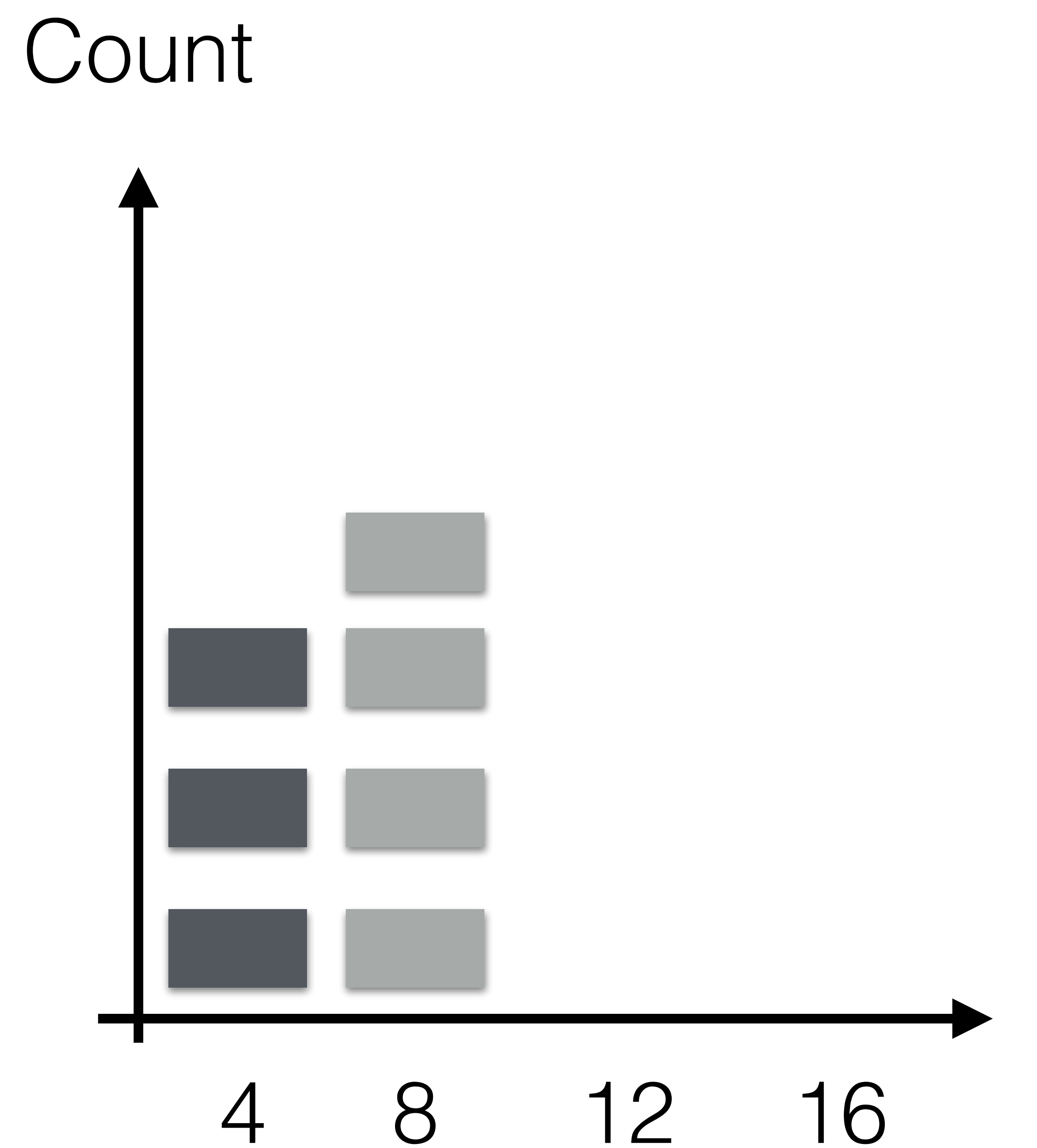


Image Histogram: Example

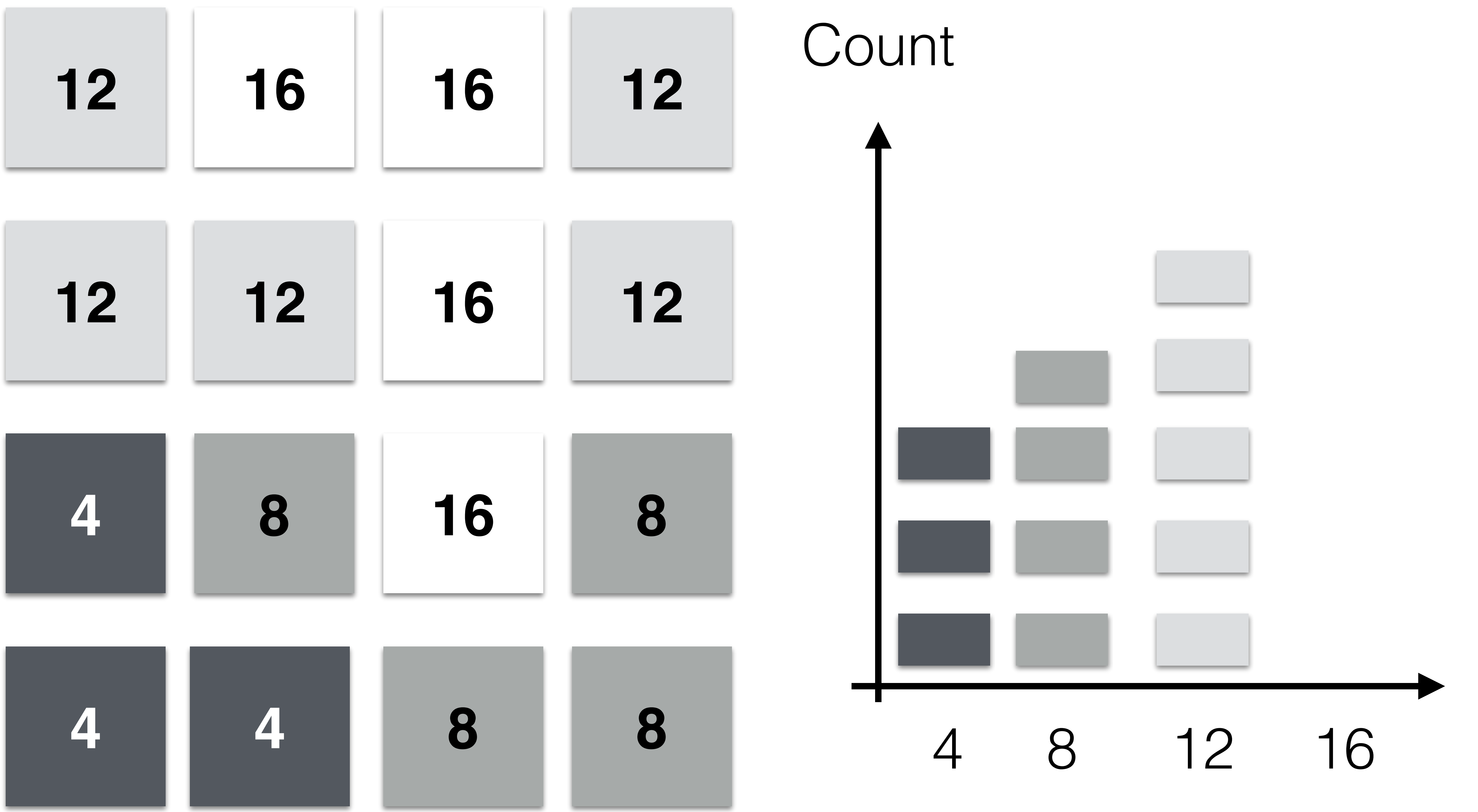
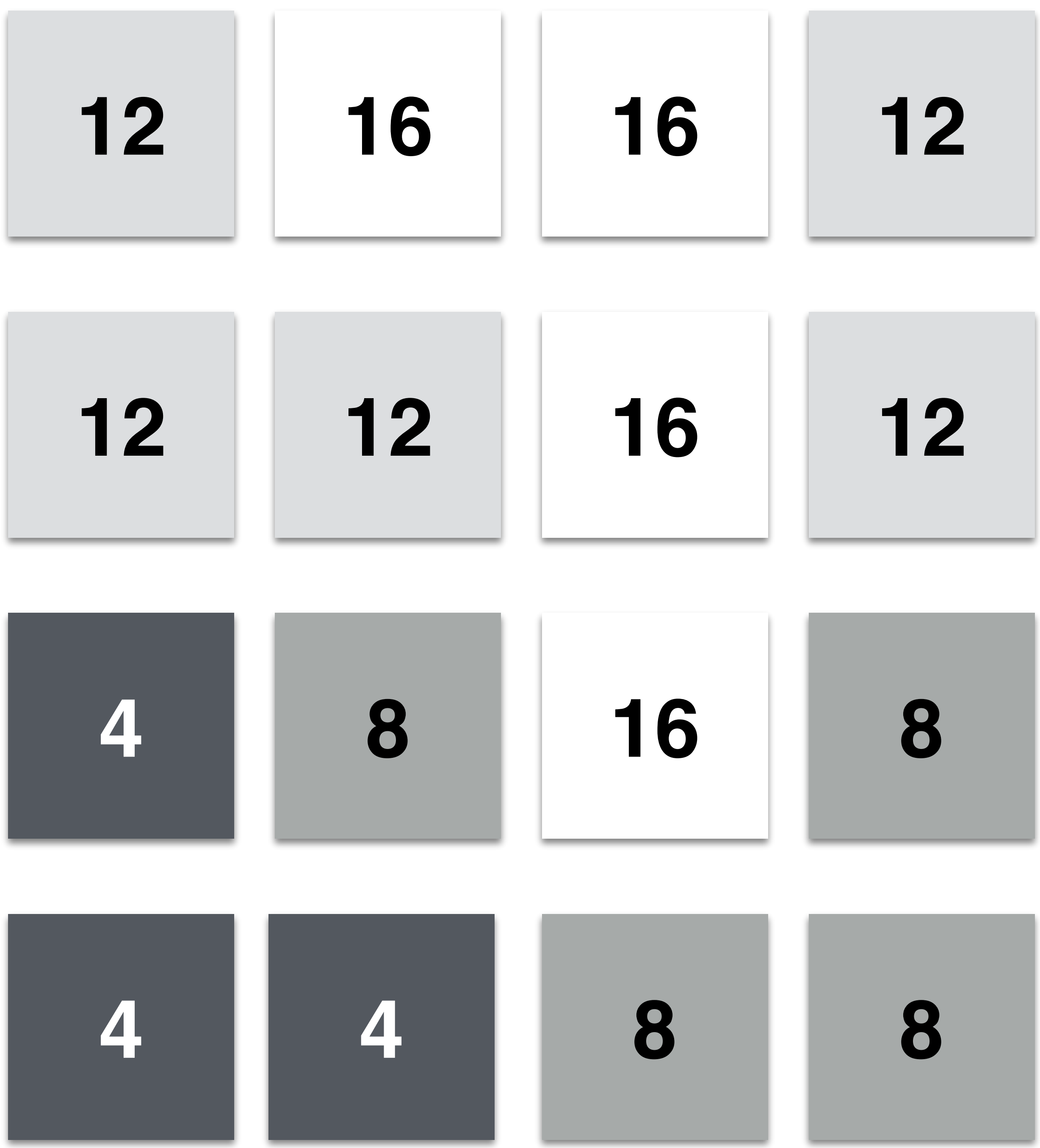


Image Histogram: Example



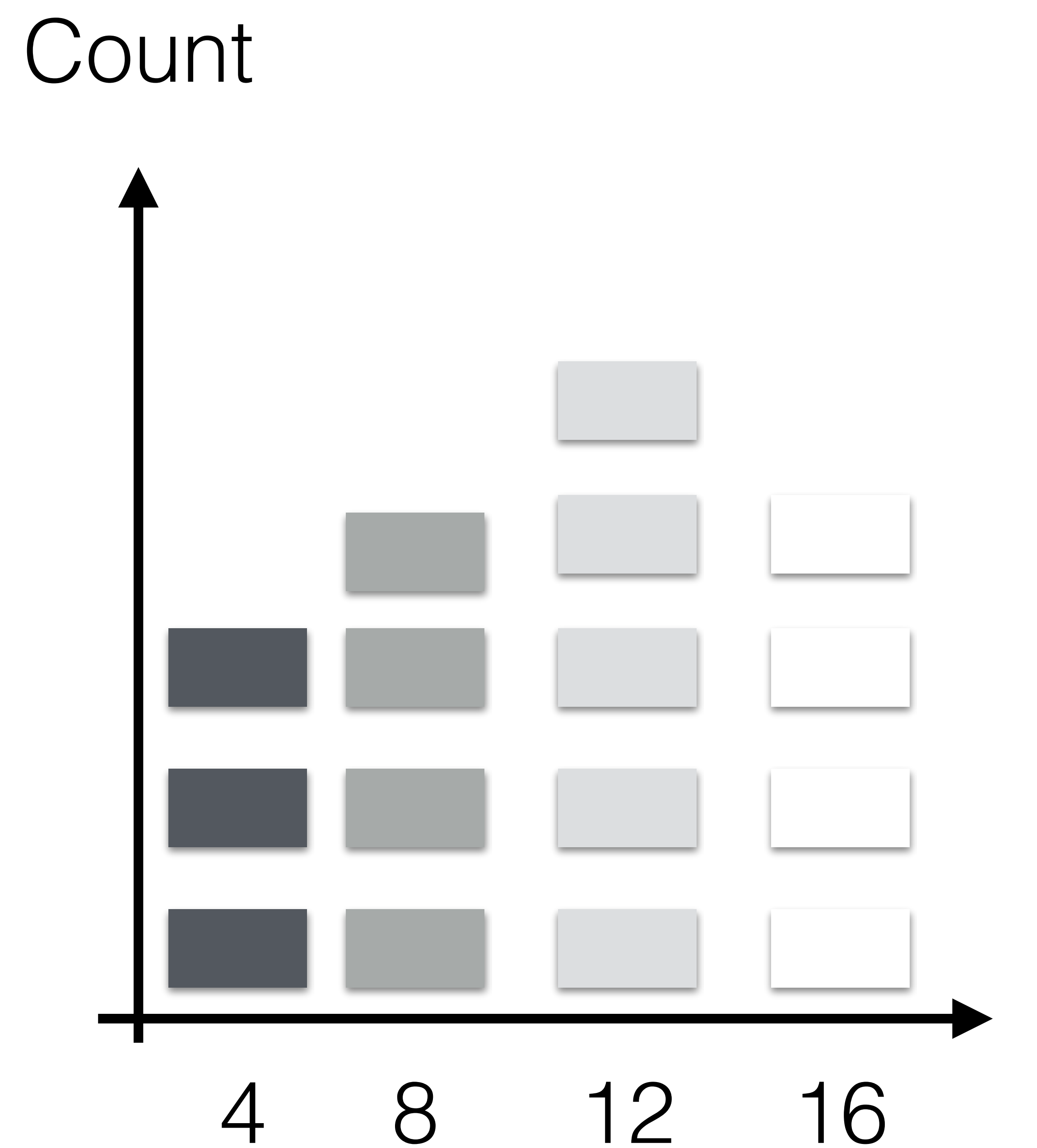


Image Histogram

- H can be seen as the ***probability*** of an intensity value to have to be present in the image I:

$$p(I(x, y)) = \frac{H[f(I(x, y))]}{N},$$

where:

- $I(x, y)$ is the intensity of a pixel in I at coordinate (x, y)
- N is the number of pixels of I
- f assigns intensity values to their bin in H

Histogram Equalization

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

- **IDEA**: we want to enforce pixels' value to have the same frequency.

Histogram Equalization

- This means that we want a histogram in which each intensity value j (or bin) has the **same** (more or less) number of pixels:

$$H(J) = \frac{N_p}{2^{\mathrm{n\_bit}}},$$

where N_p is the number of pixels of the image, and $\mathrm{n\_bit}$ is the bit-depth of all pixels.

- **NOTE**: we have a bin for each intensity value!

Histogram Equalization

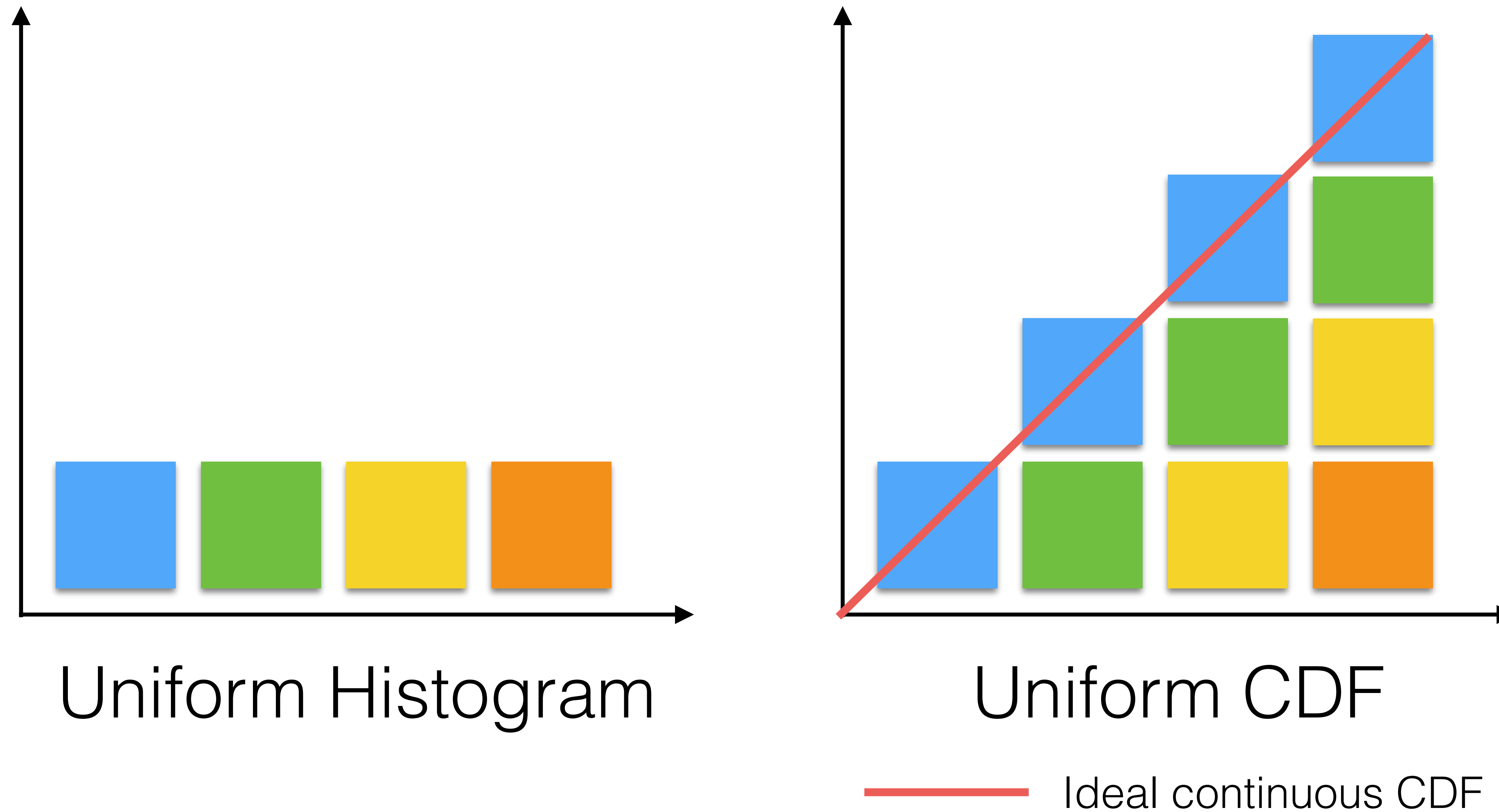
- How?
- Matching the CDF (cumulative distribution function) of the histogram with the CDF of a uniform histogram.
- A CDF is defined as:

$$F(x) = P(X \leq x) = \int_{-inf}^{x} p(x)dx$$

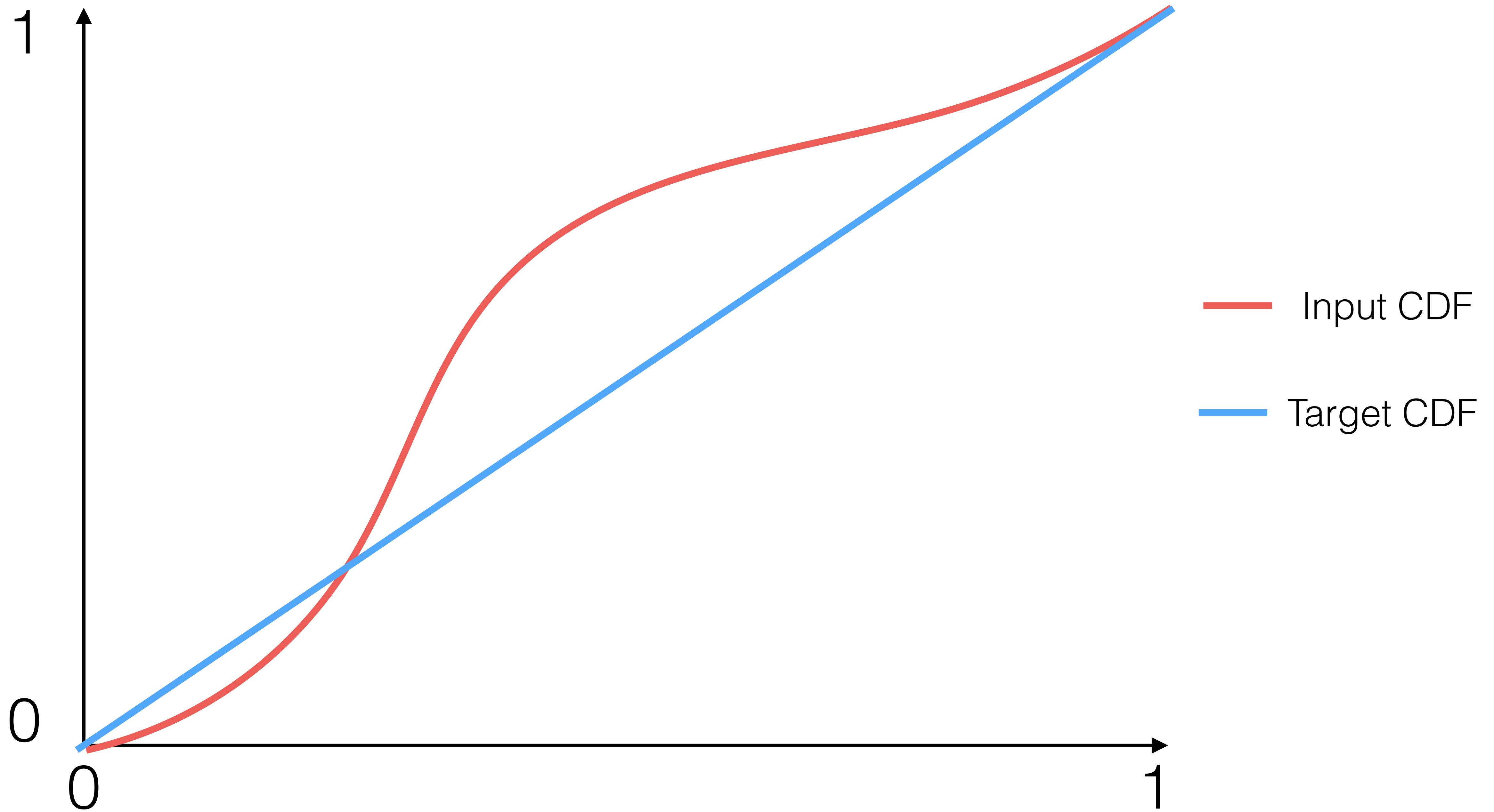
- A uniform CDF is defined as:

$$F(x) = x$$

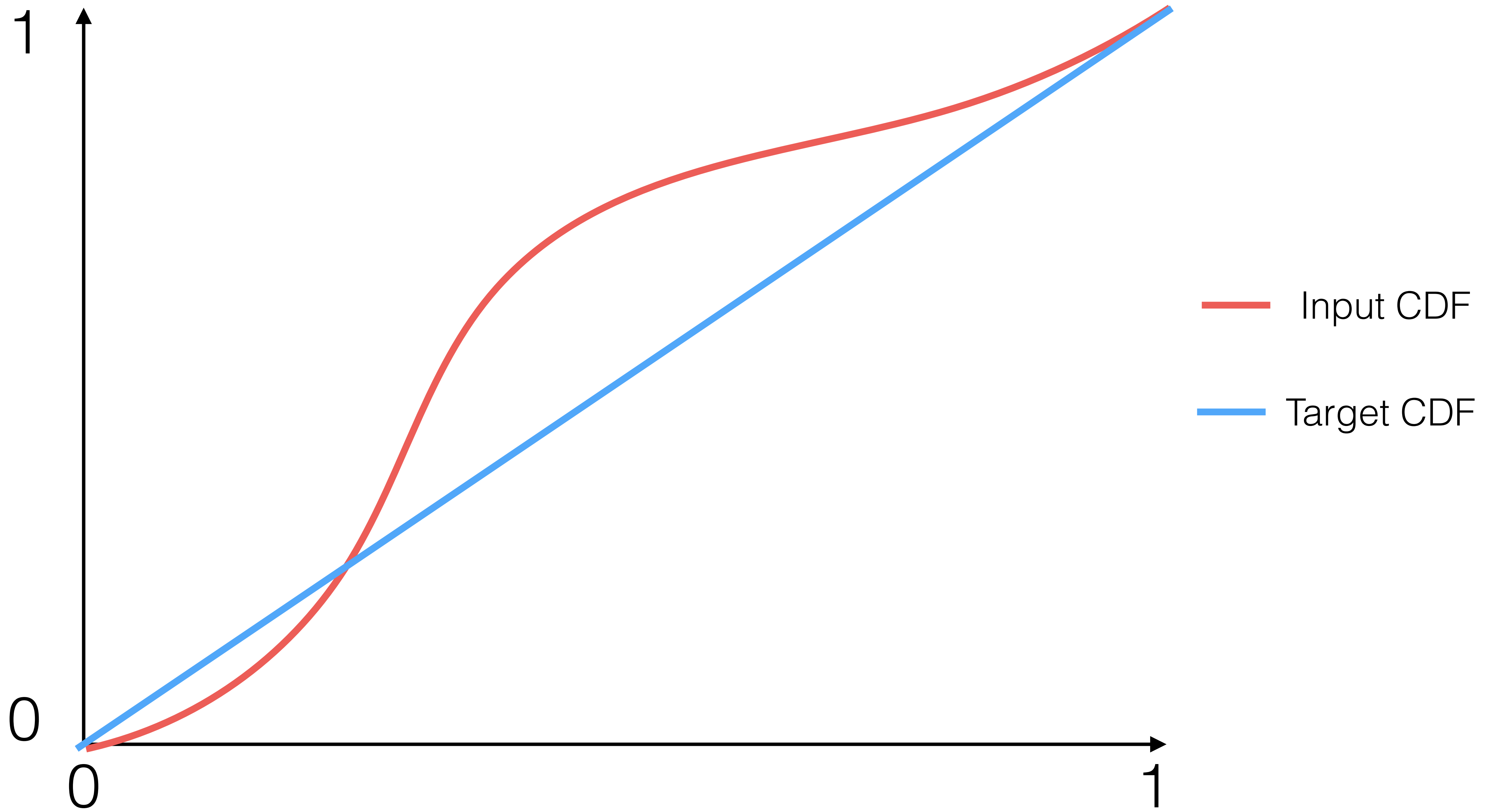
Histogram Equalization



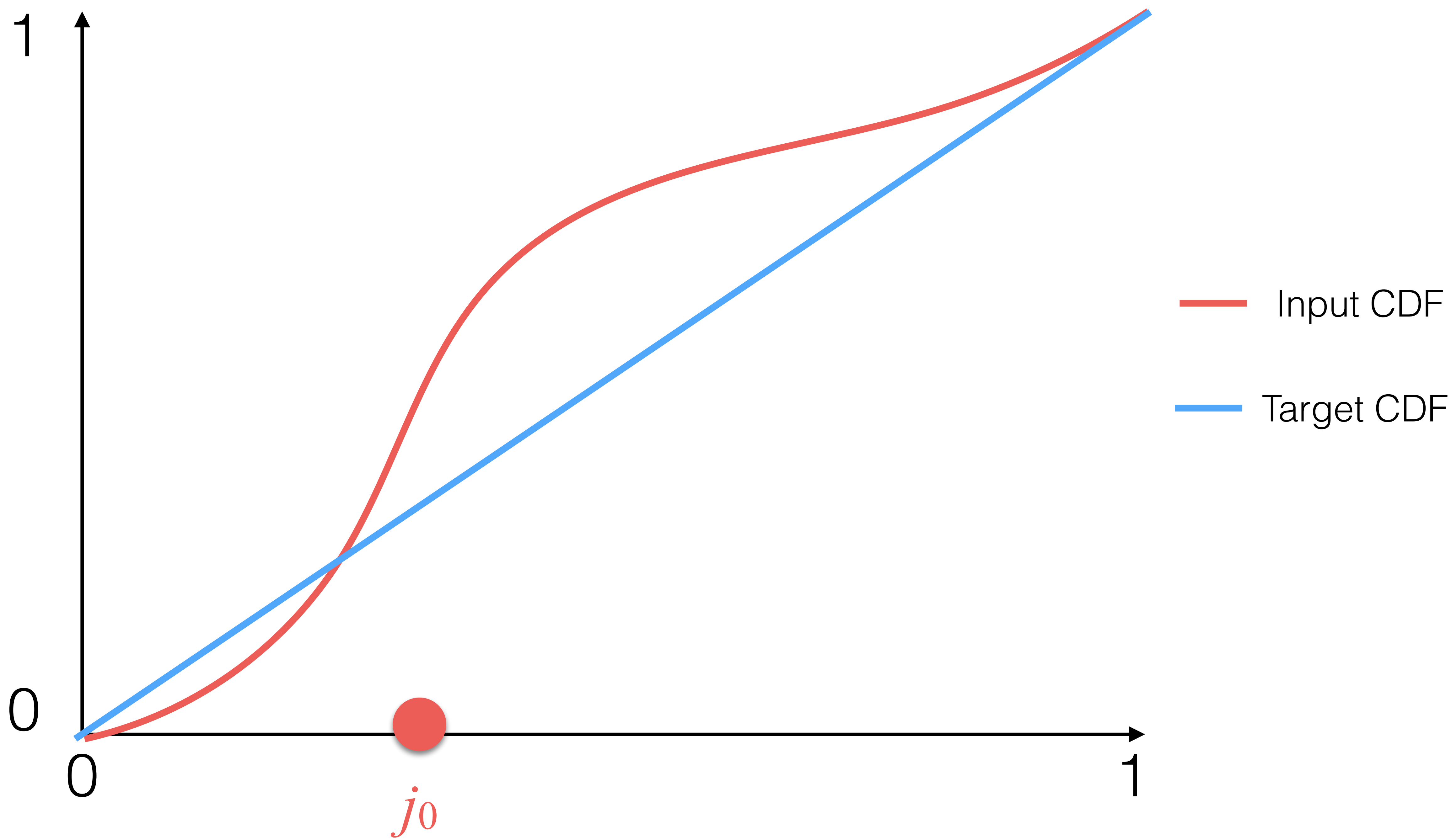
Histogram Equalization: Histogram Matching



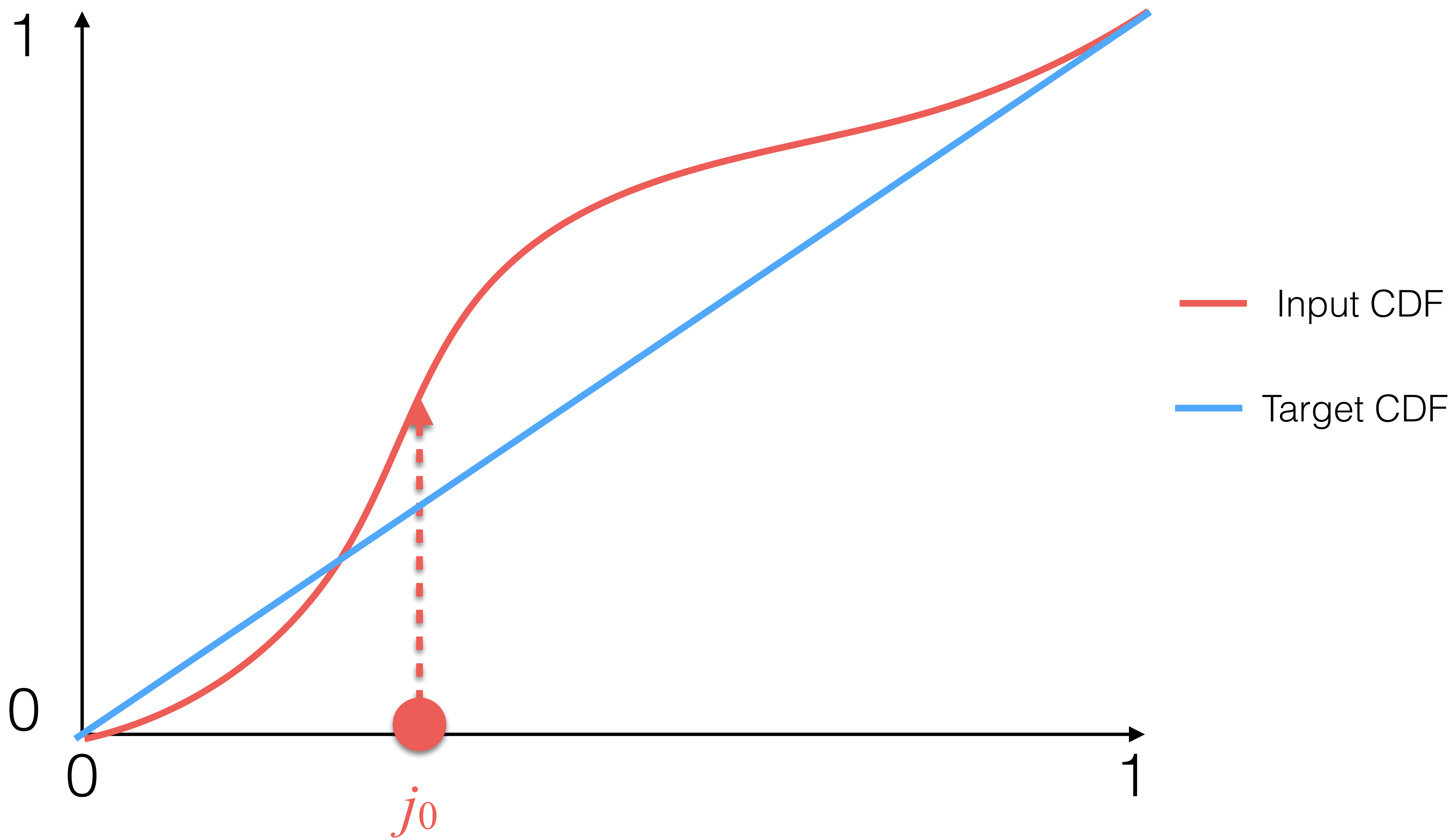
Histogram Equalization: Histogram Matching



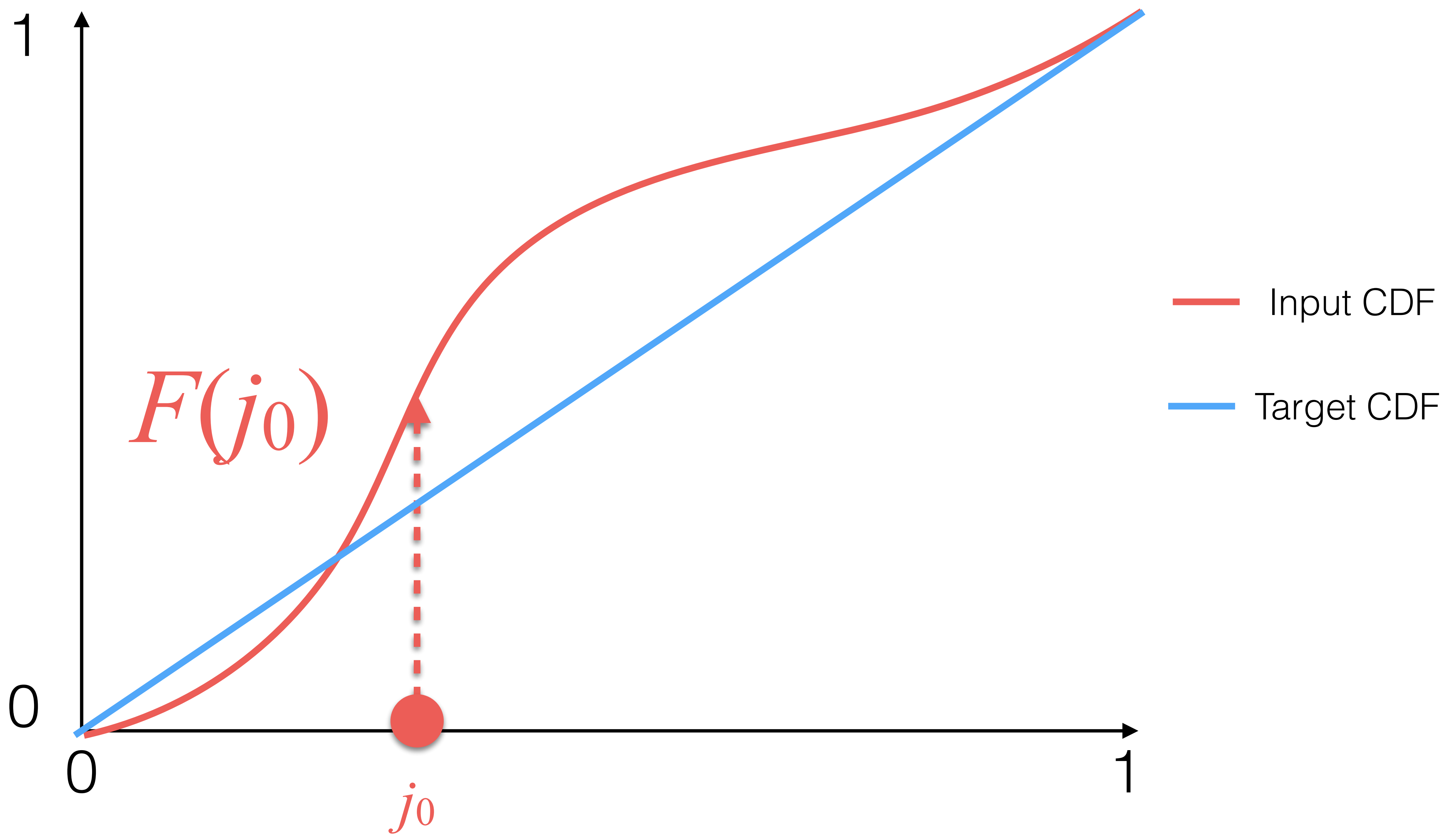
Histogram Equalization: Histogram Matching



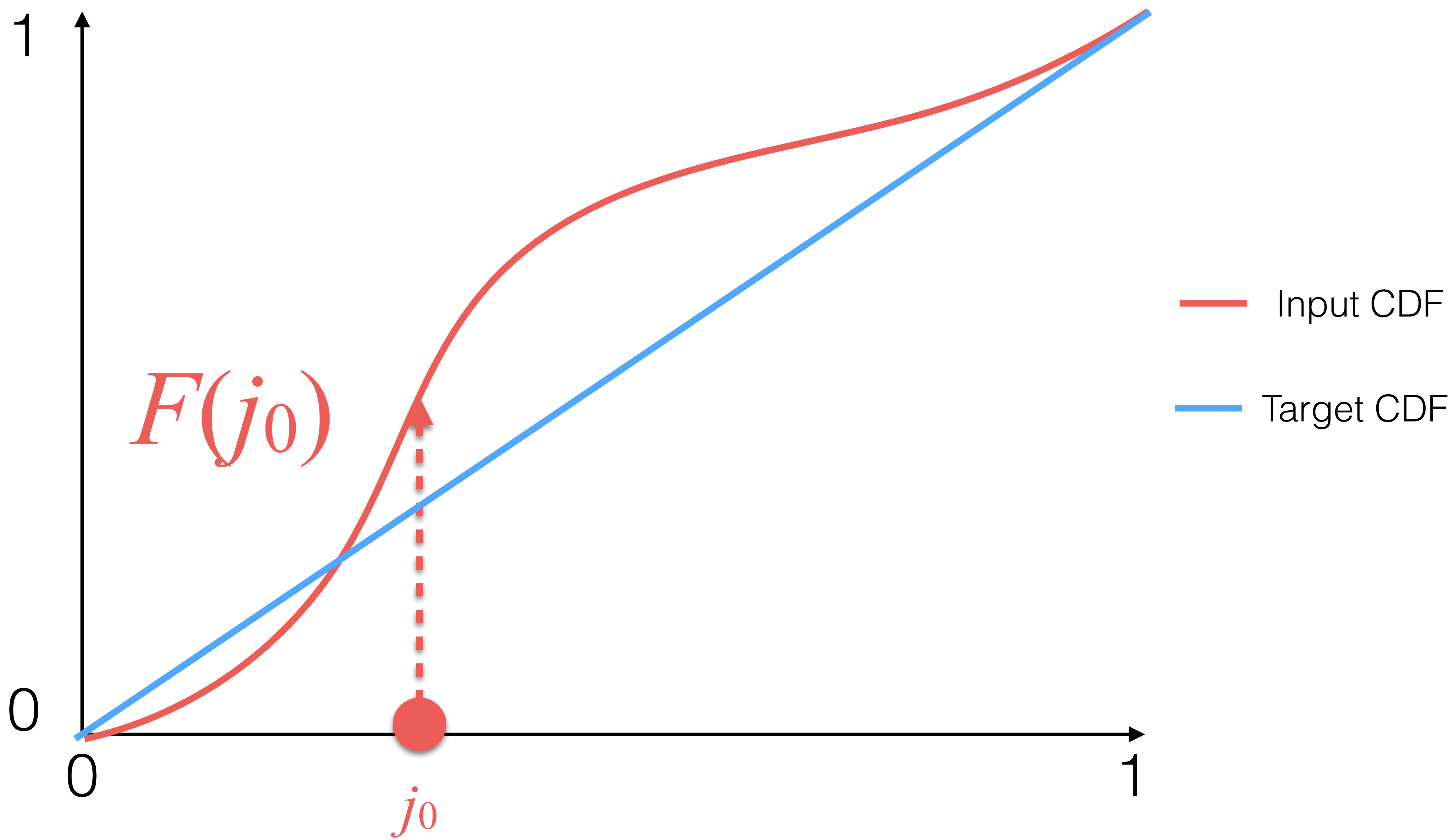
Histogram Equalization: Histogram Matching



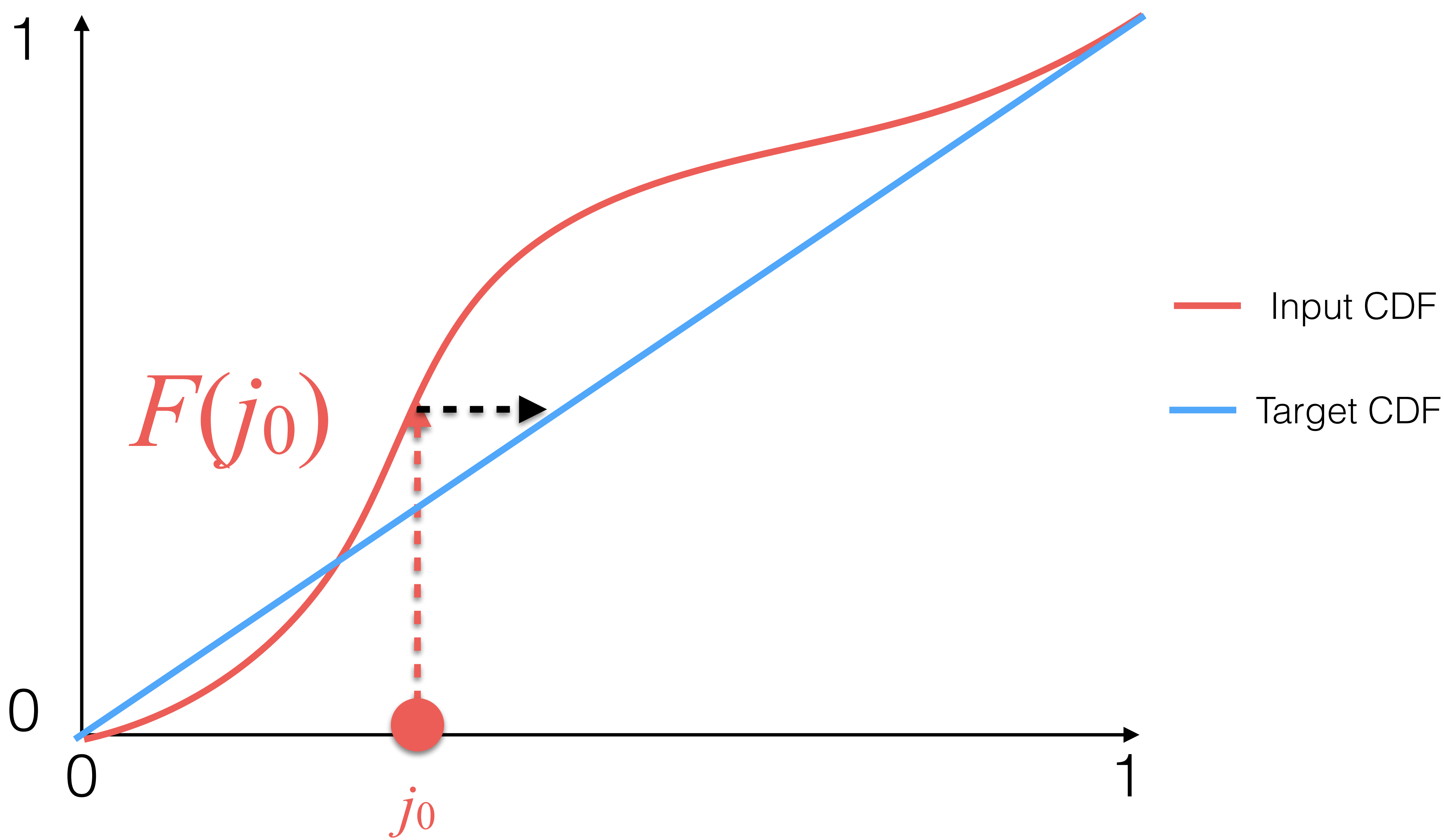
Histogram Equalization: Histogram Matching



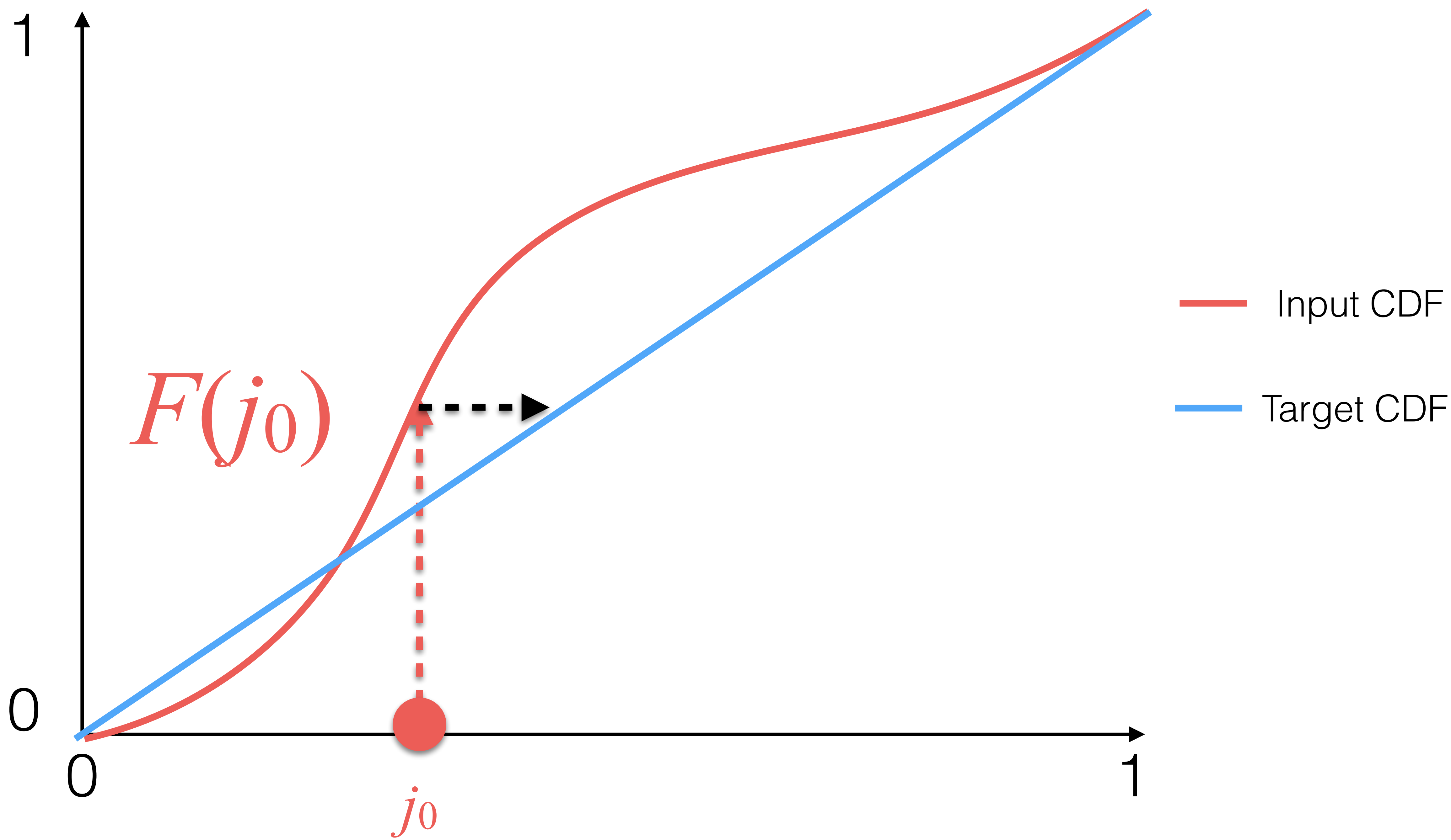
Histogram Equalization: Histogram Matching



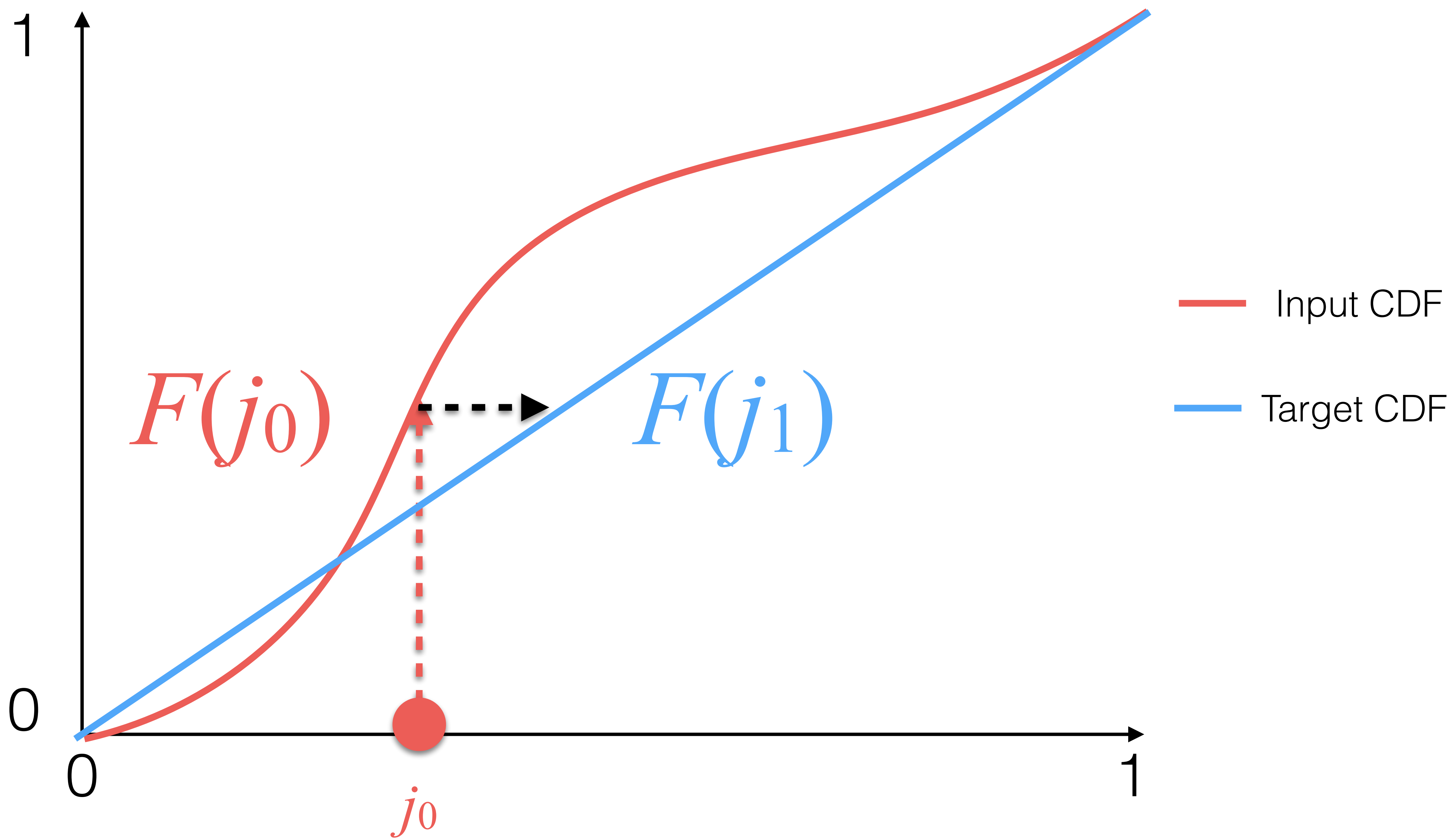
Histogram Equalization: Histogram Matching



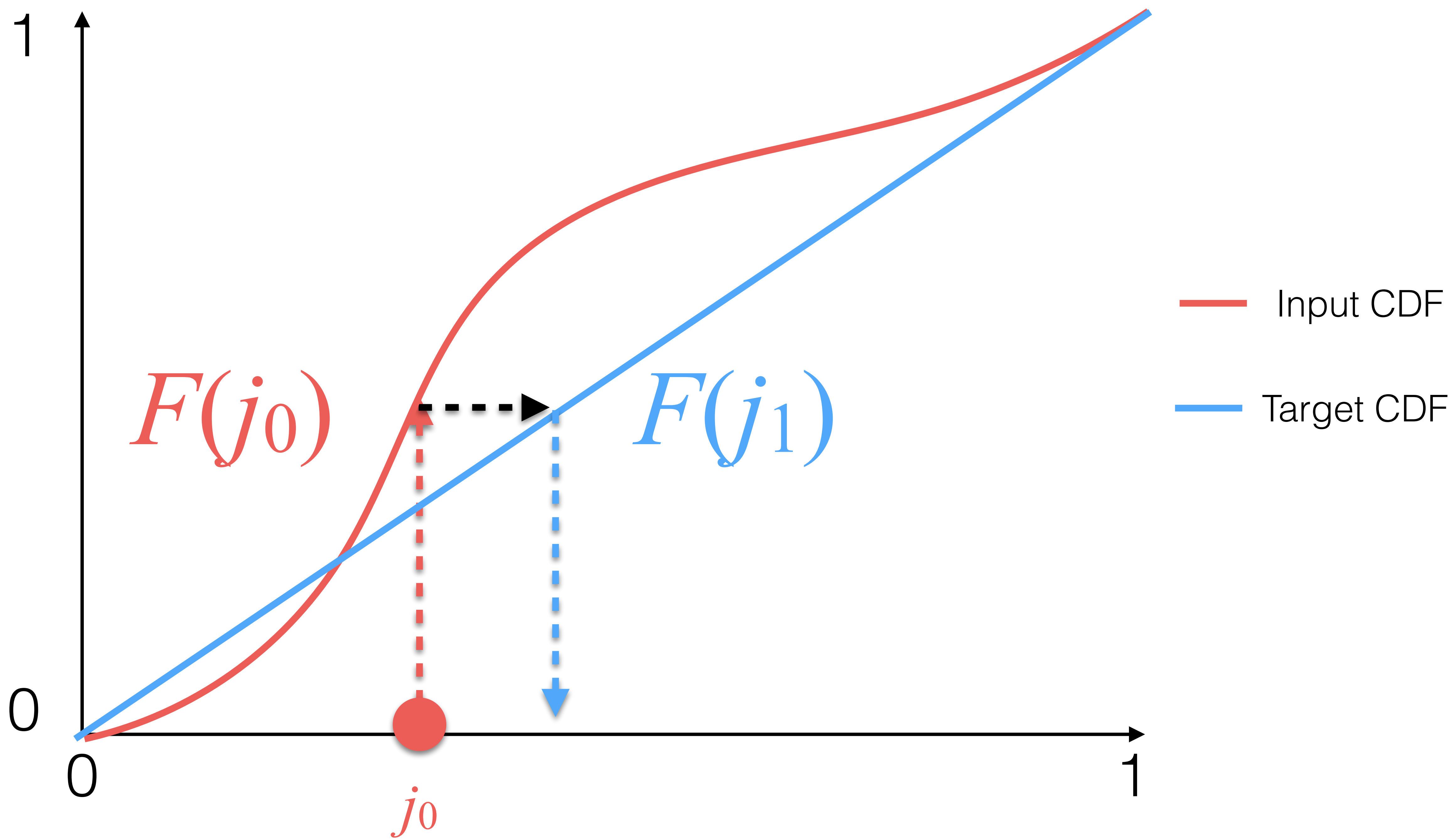
Histogram Equalization: Histogram Matching



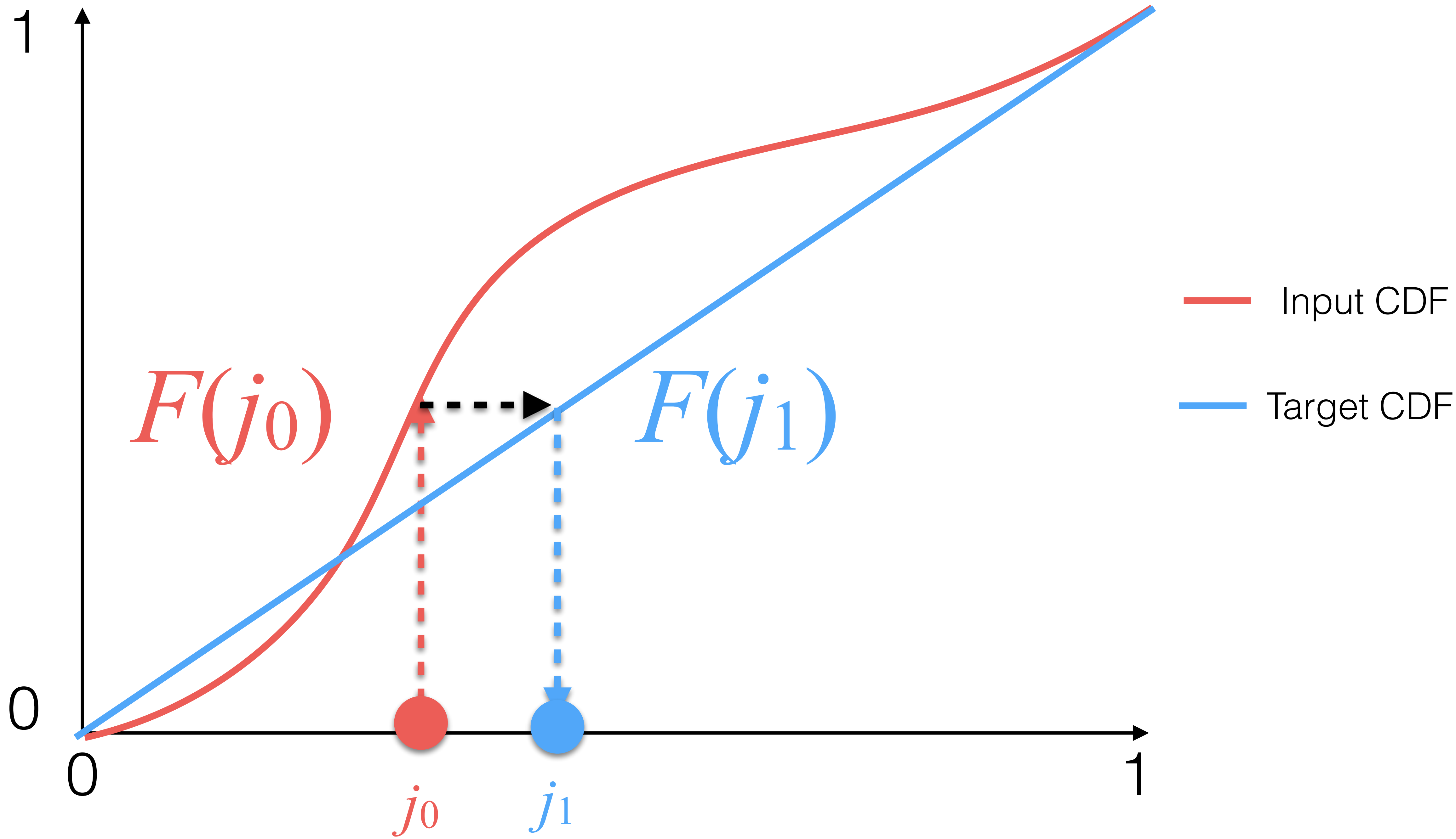
Histogram Equalization: Histogram Matching



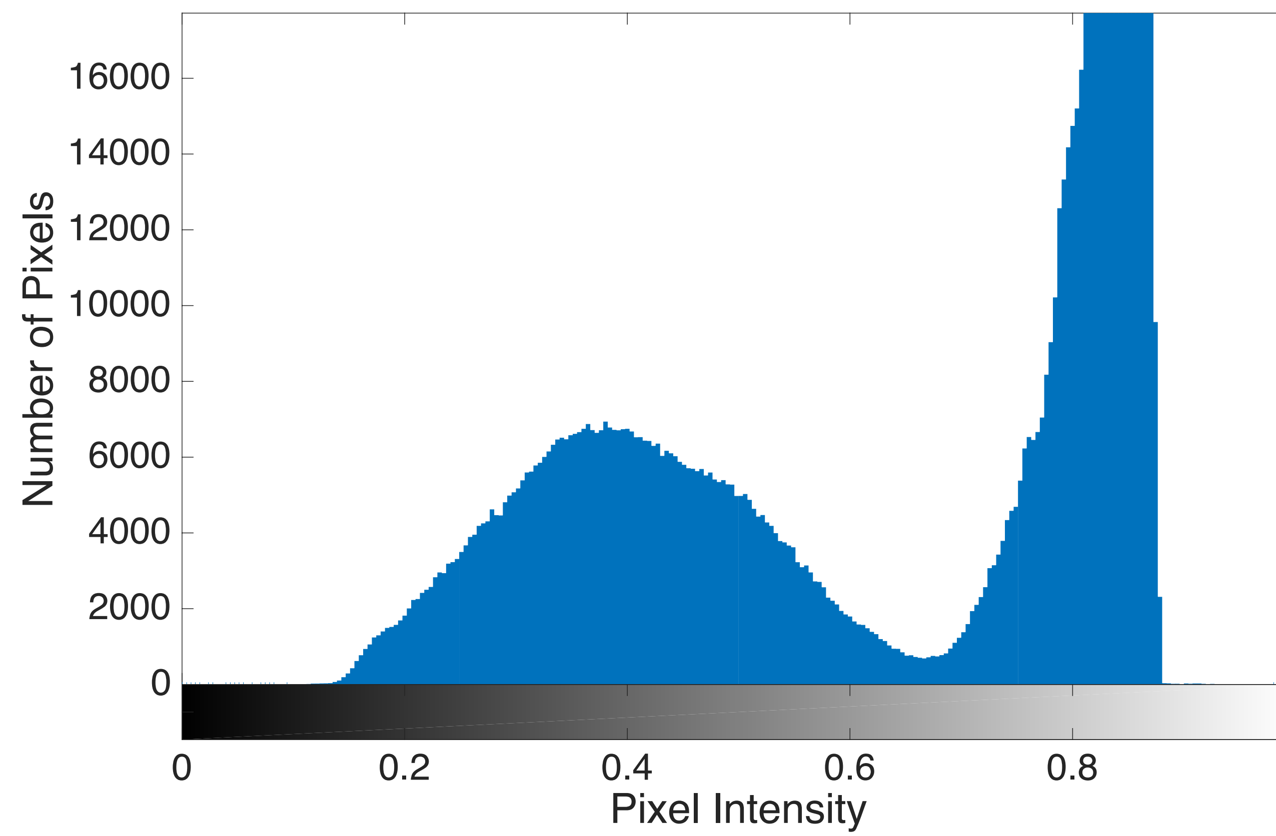
Histogram Equalization: Histogram Matching



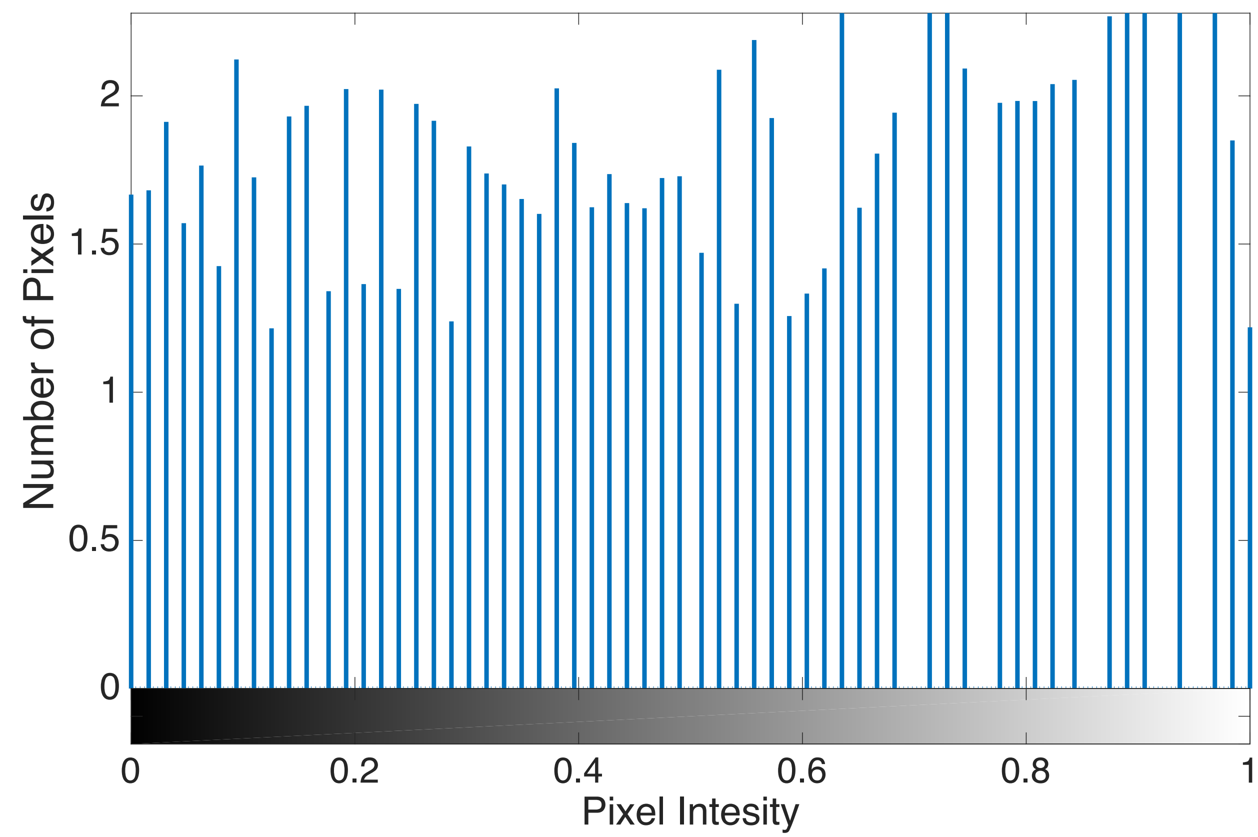
Histogram Equalization: Histogram Matching



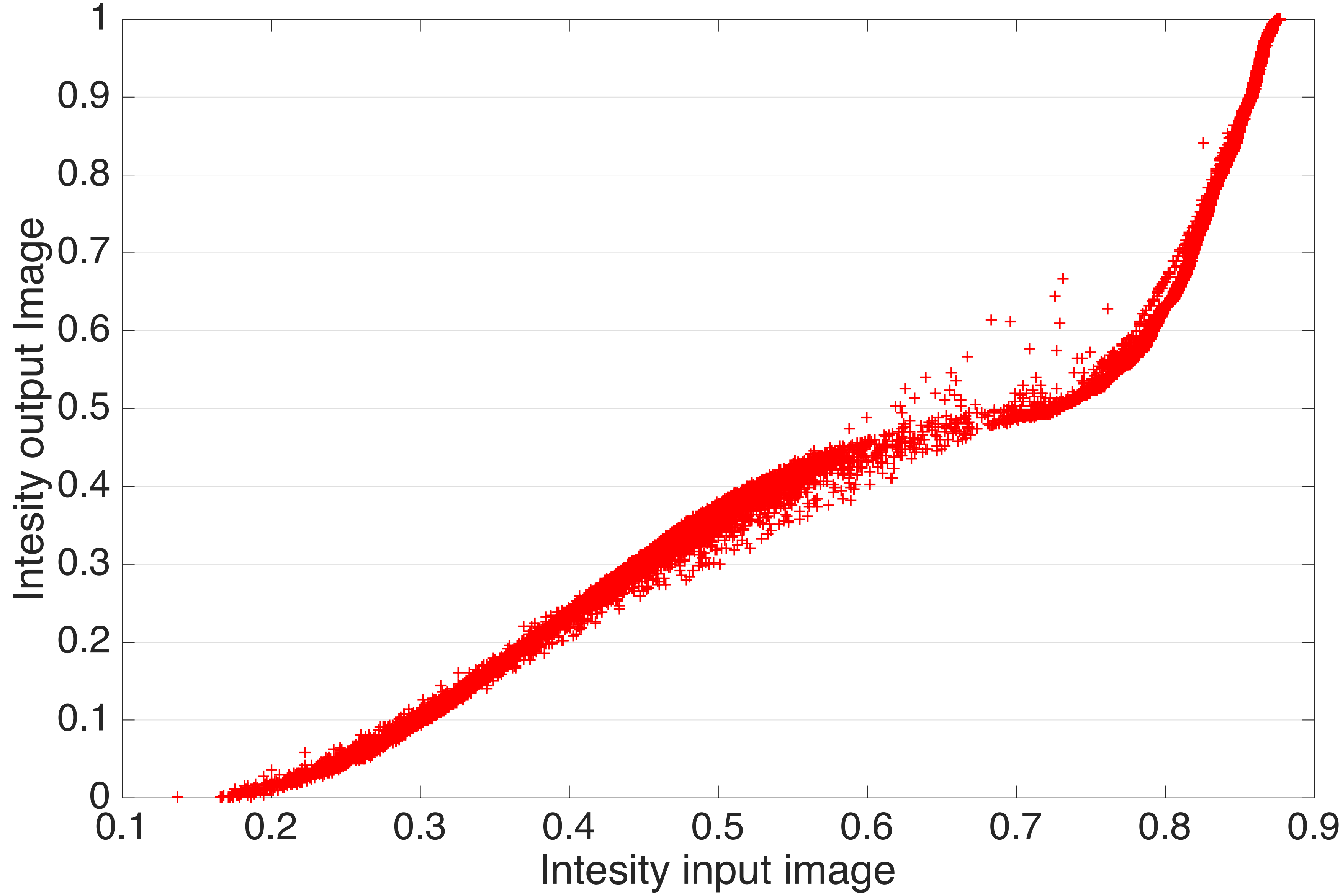
Histogram Equalization Example



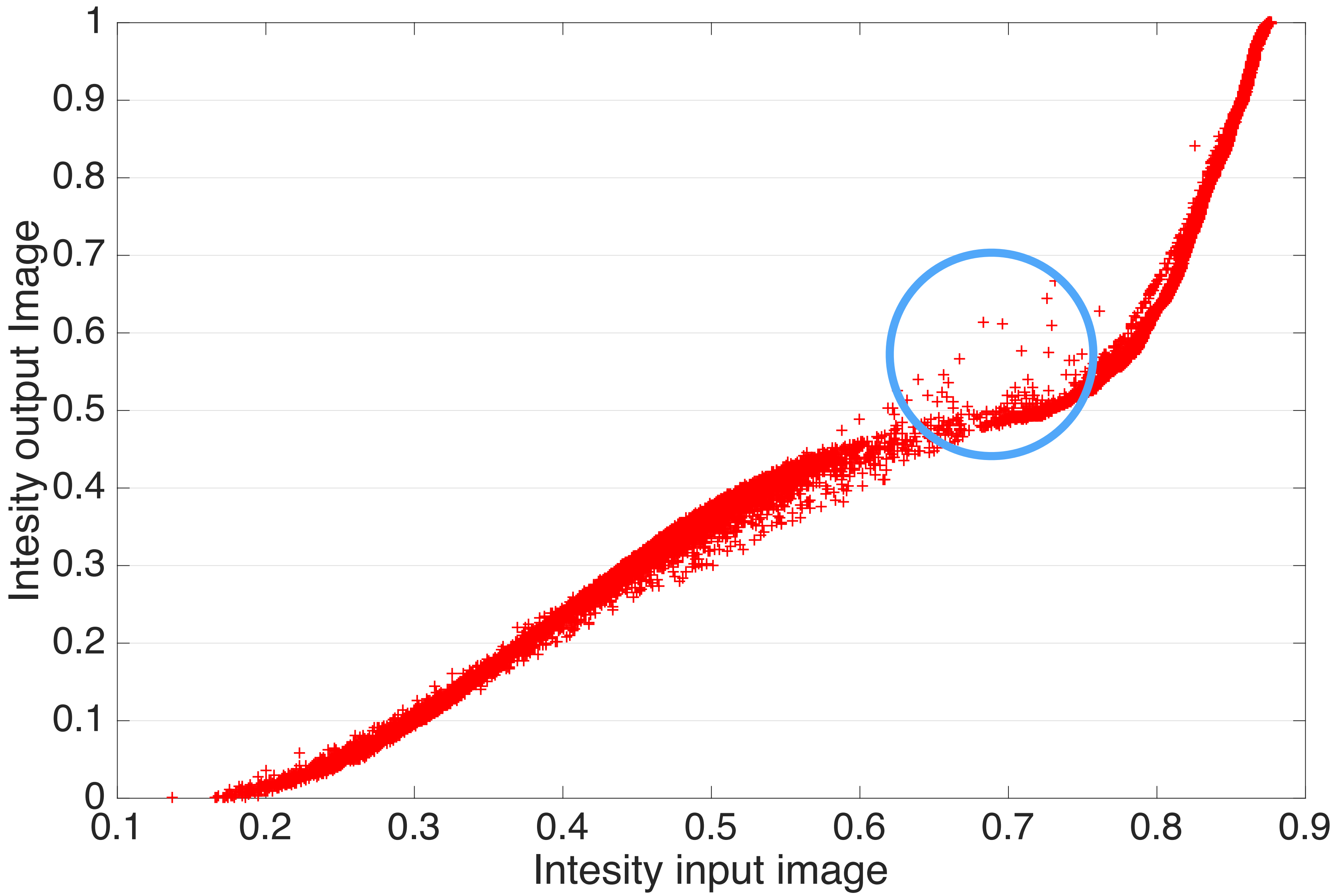
Histogram Equalization Example



Histogram Equalization Example

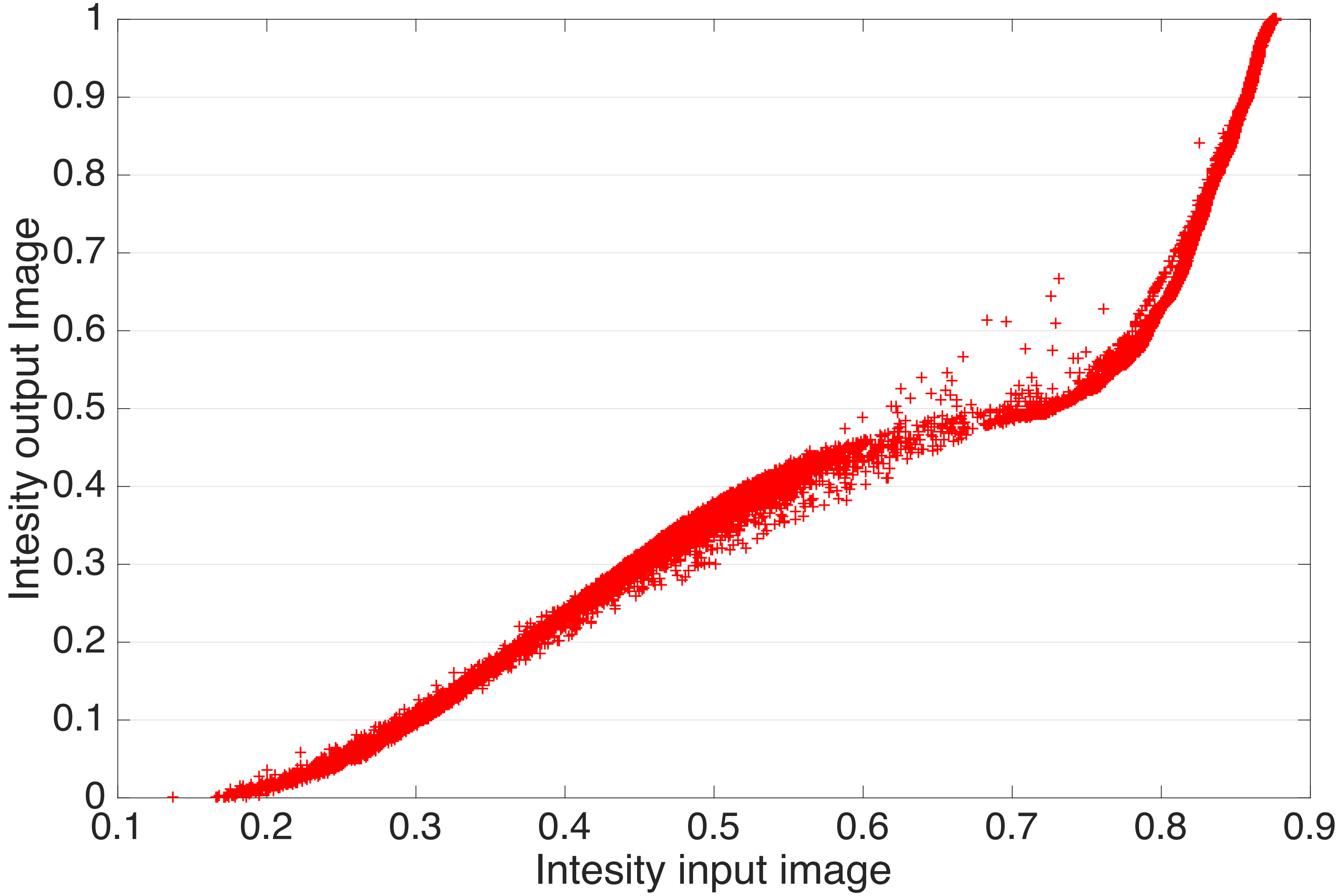


Histogram Equalization Example

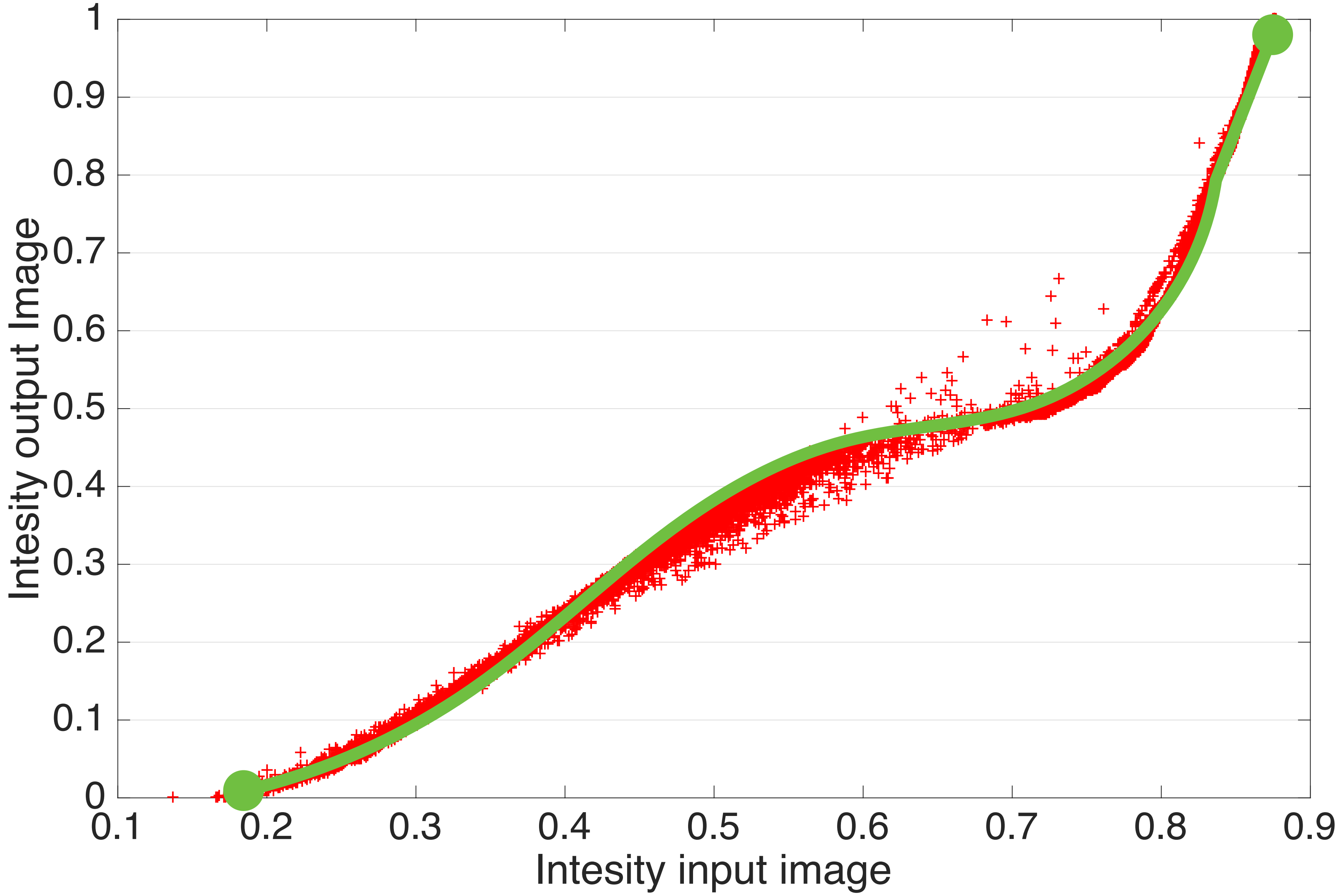


Histogram Equalization

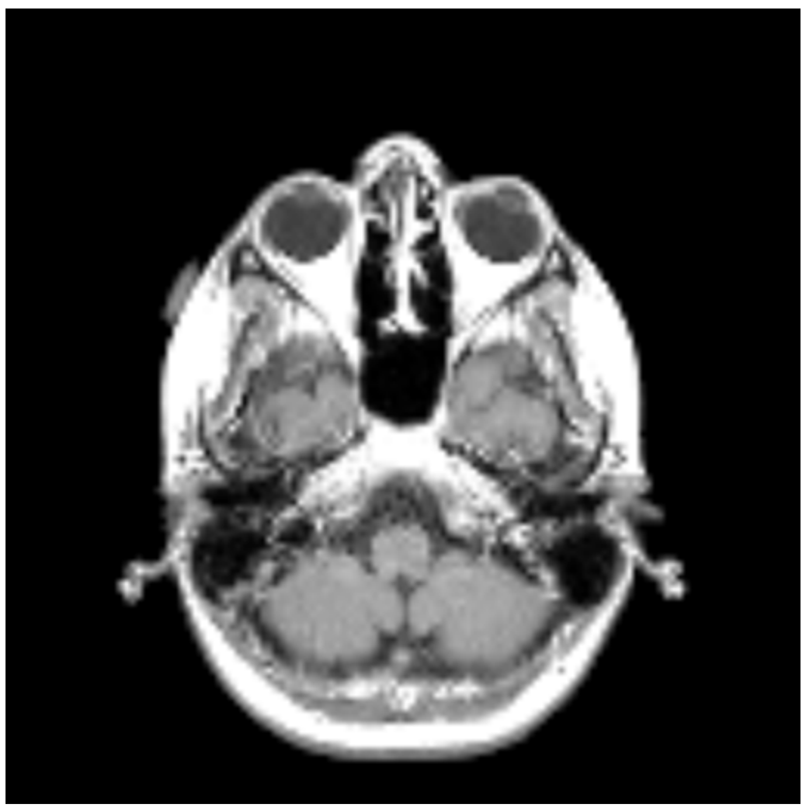
Histogram Equalization Example

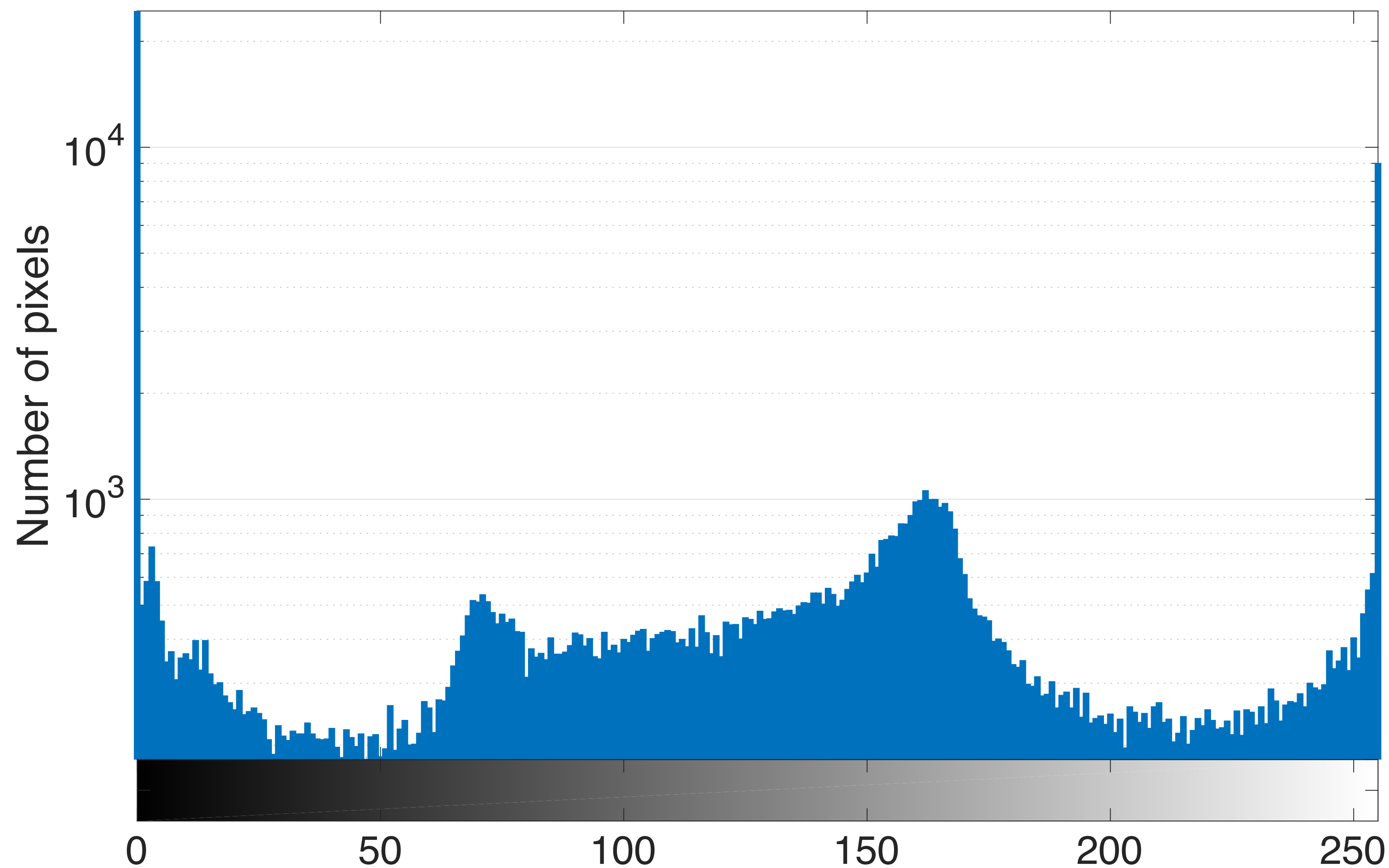


Histogram Equalization Example



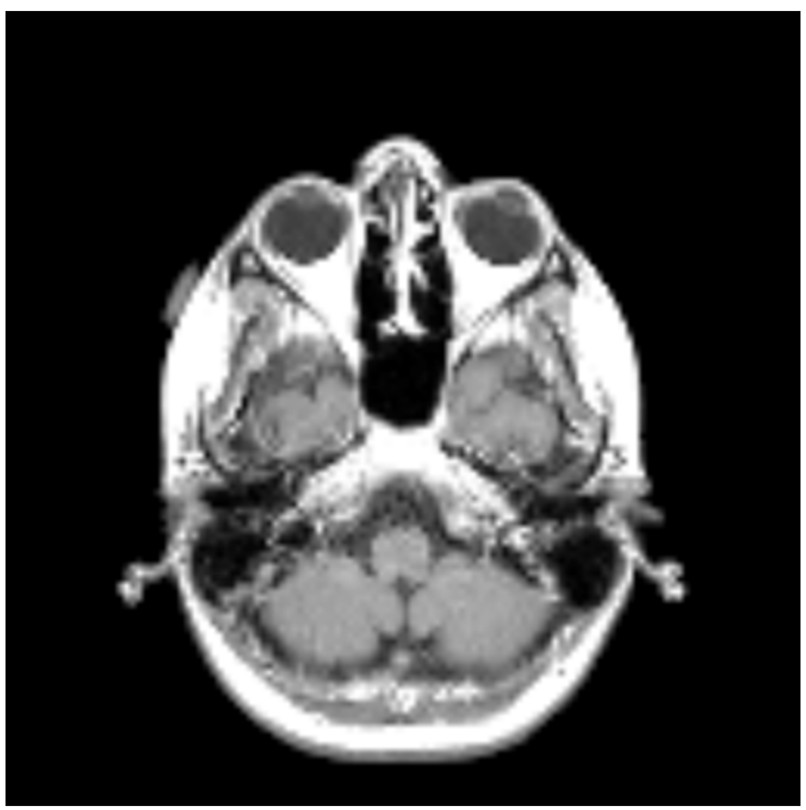
Histogram Equalization: Bias in Statistics

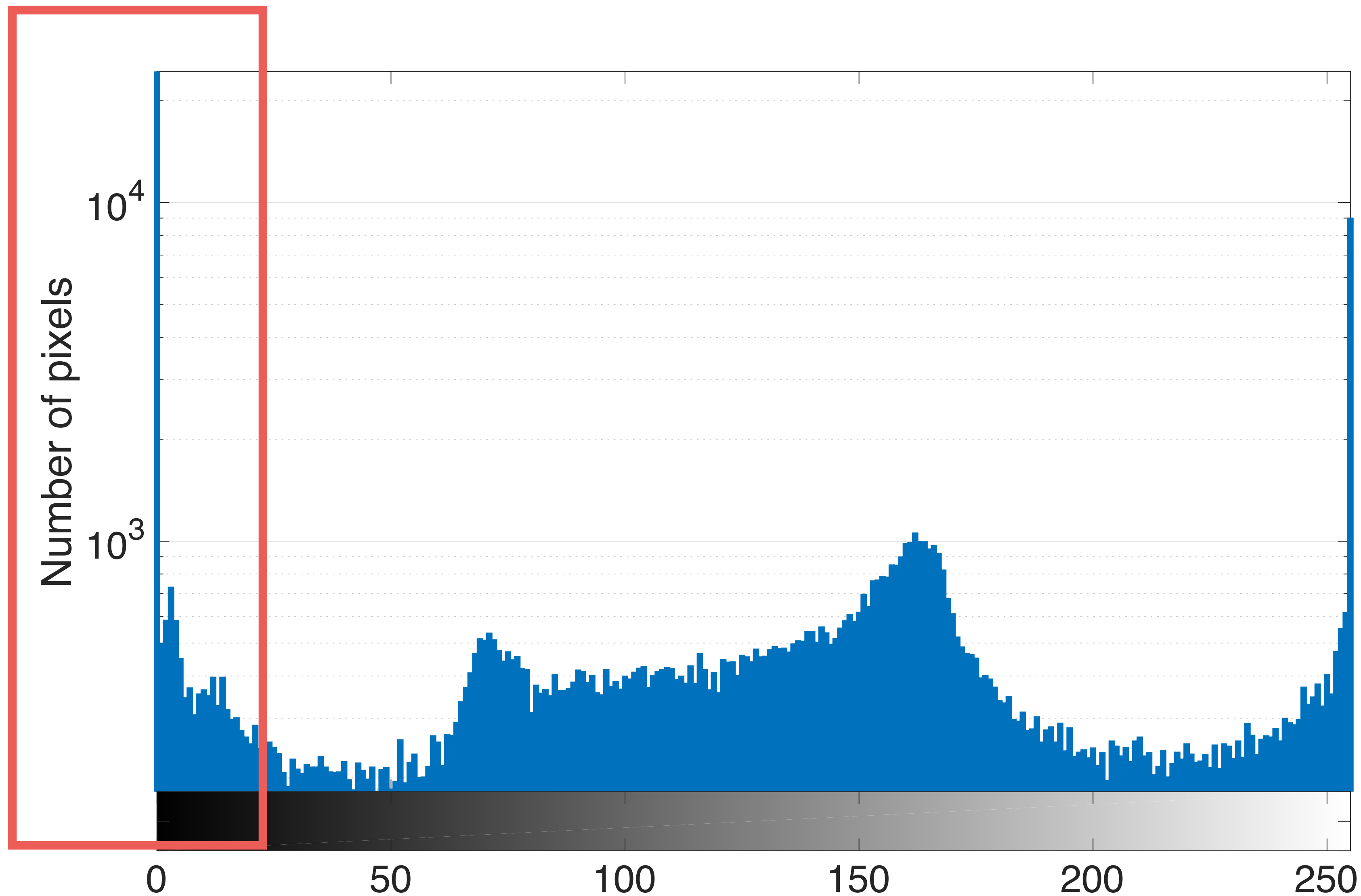




ROI helps in cases of huge peaks (see I=0)

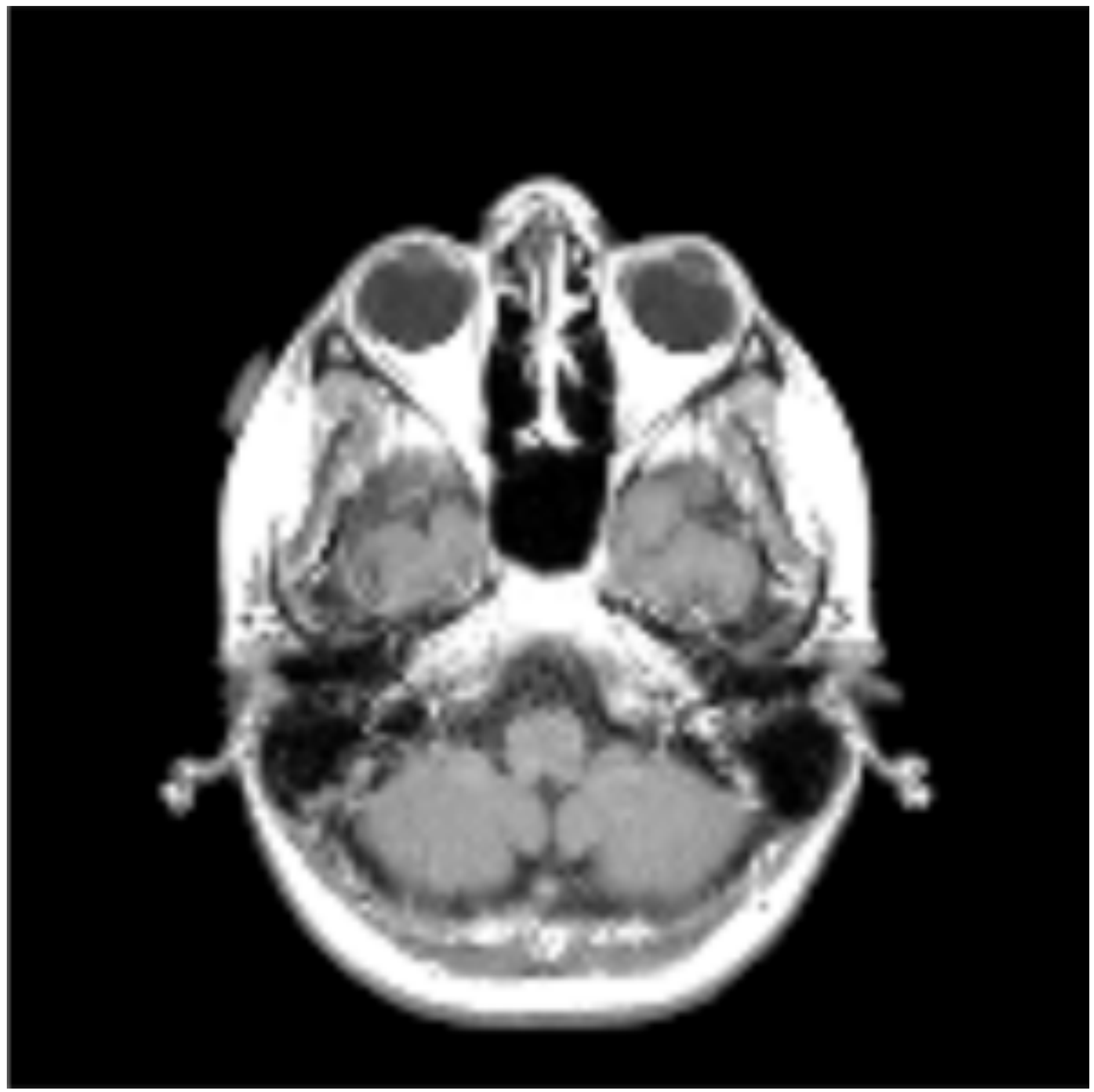
Histogram Equalization: Bias in Statistics





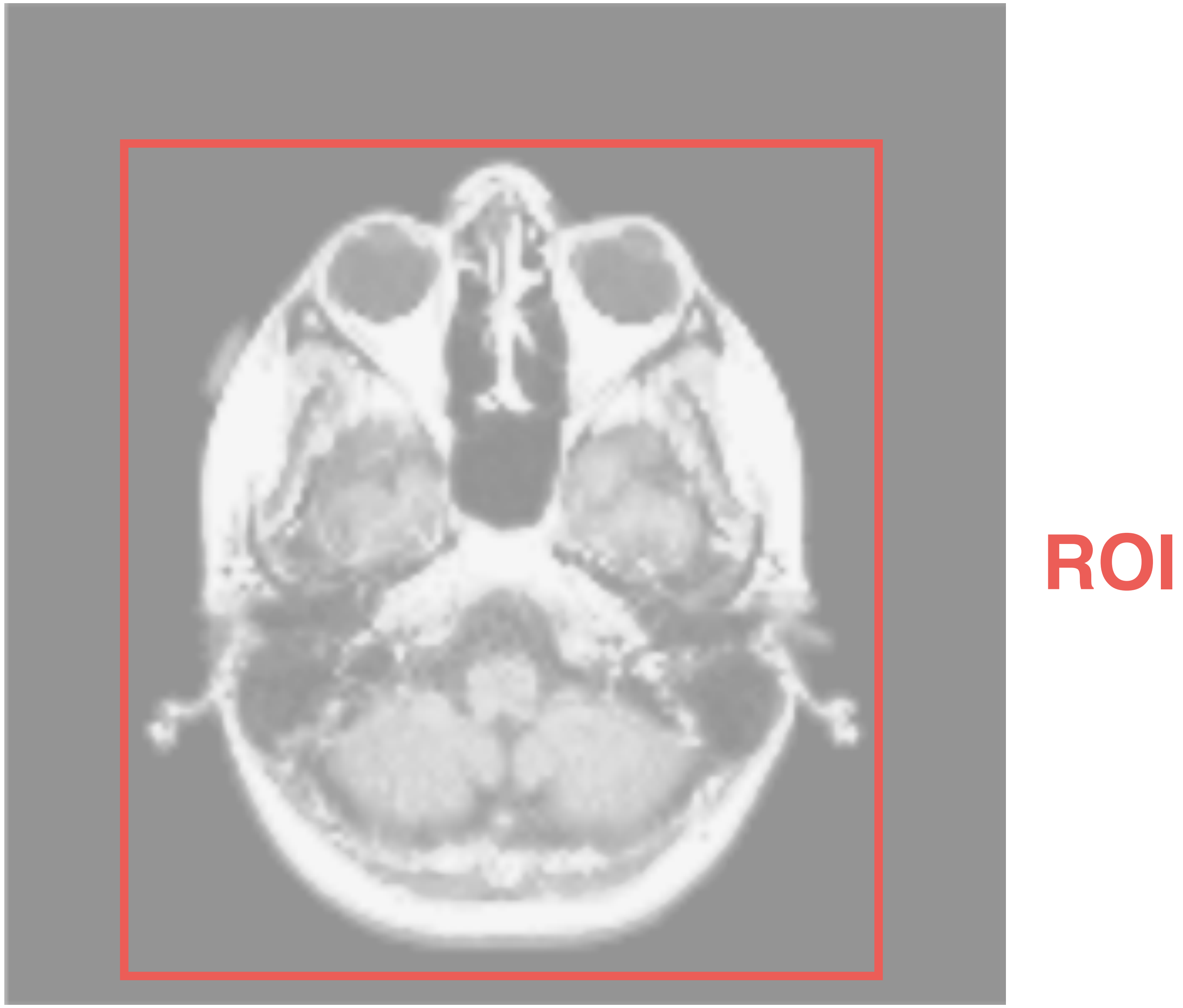
ROI helps in cases of huge peaks (see I=0)

Histogram Equalization: Bias in Statistics

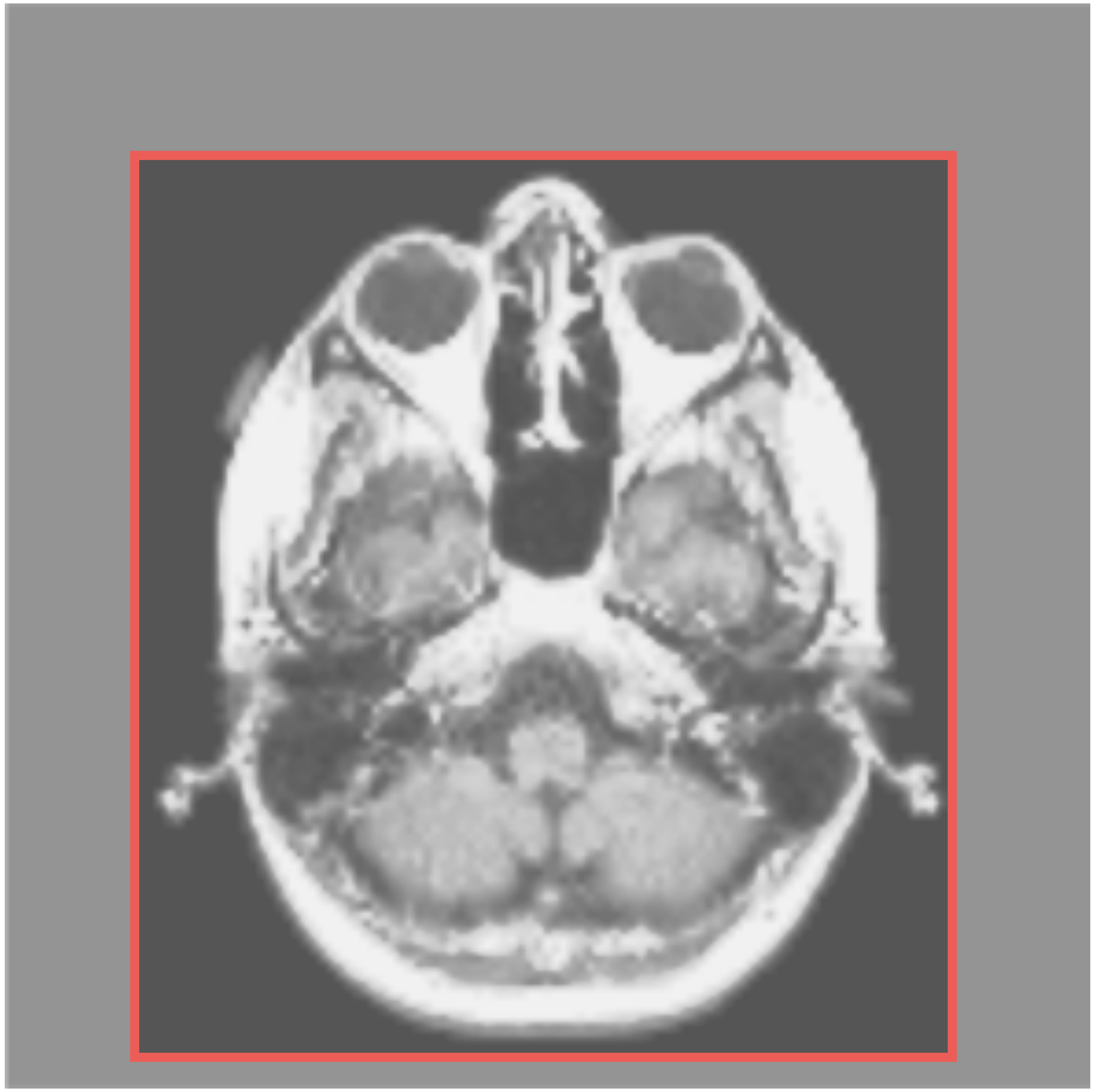


Histogram Equalization: Bias in Statistics

Histogram Equalization: Bias in Statistics



Histogram Equalization: Bias in Statistics



ROI

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 discrete 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 $2N \times 2M$ 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 using central differences:

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

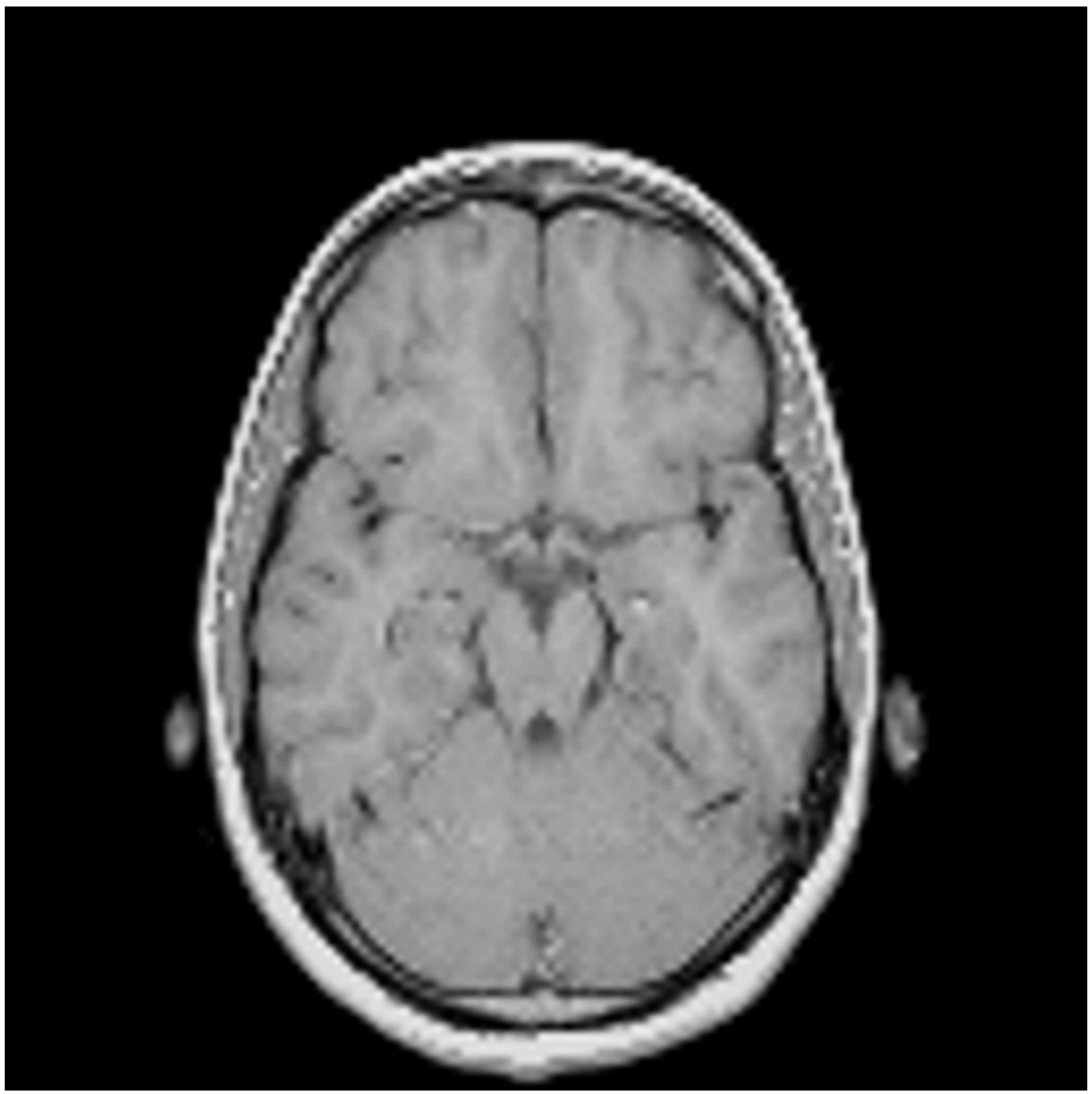
Sobel Gradient Operator

- Technically speaking, a more robust operator is Sobel operator that takes into account neighbors; at the end of the day it is just another discrete differential operator!

- It emphasizes more edges, which is good.

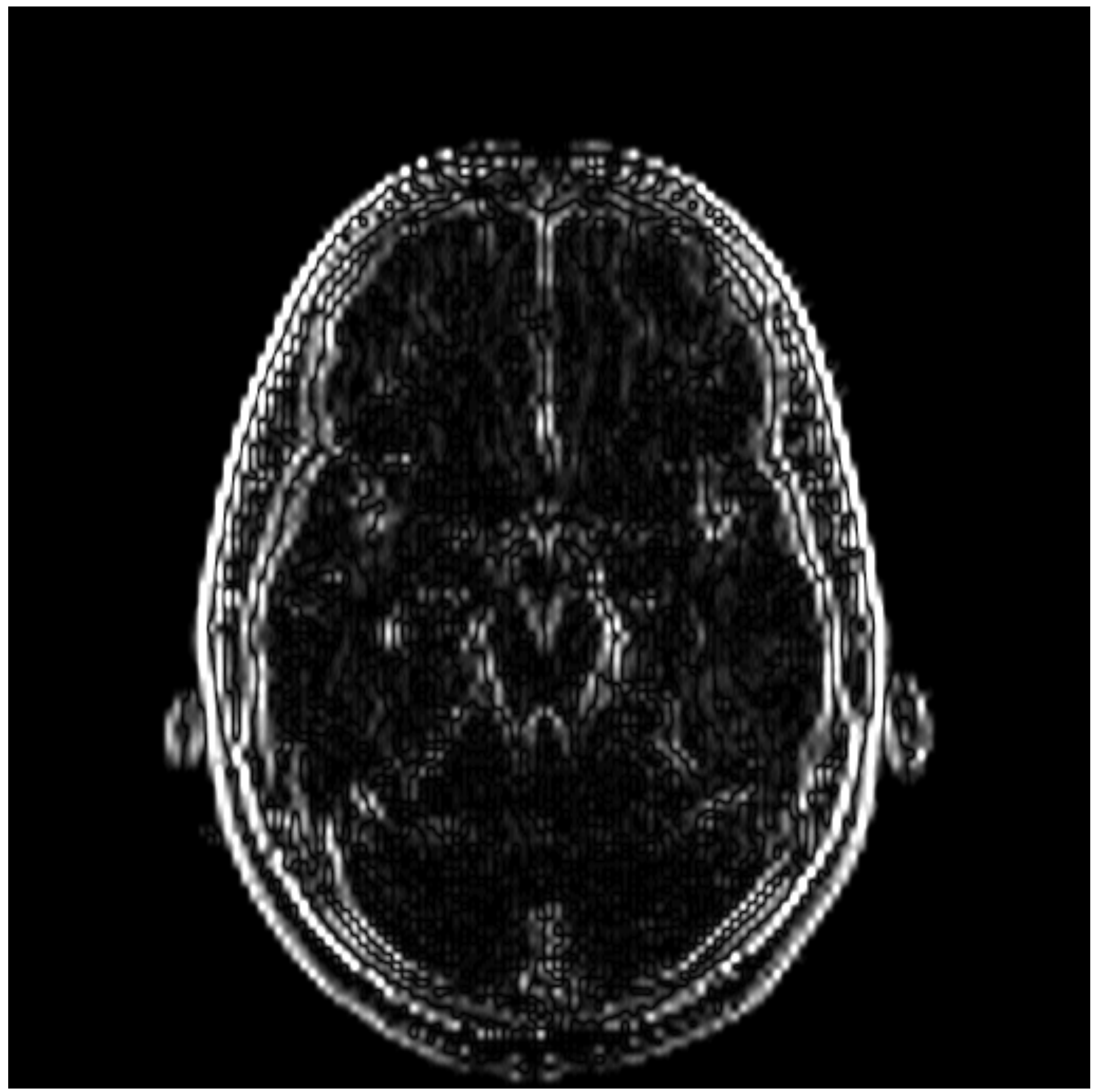
$$g_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad g_Y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & 1 \end{bmatrix}$$

Sobel Gradient Operator: X-gradient Example



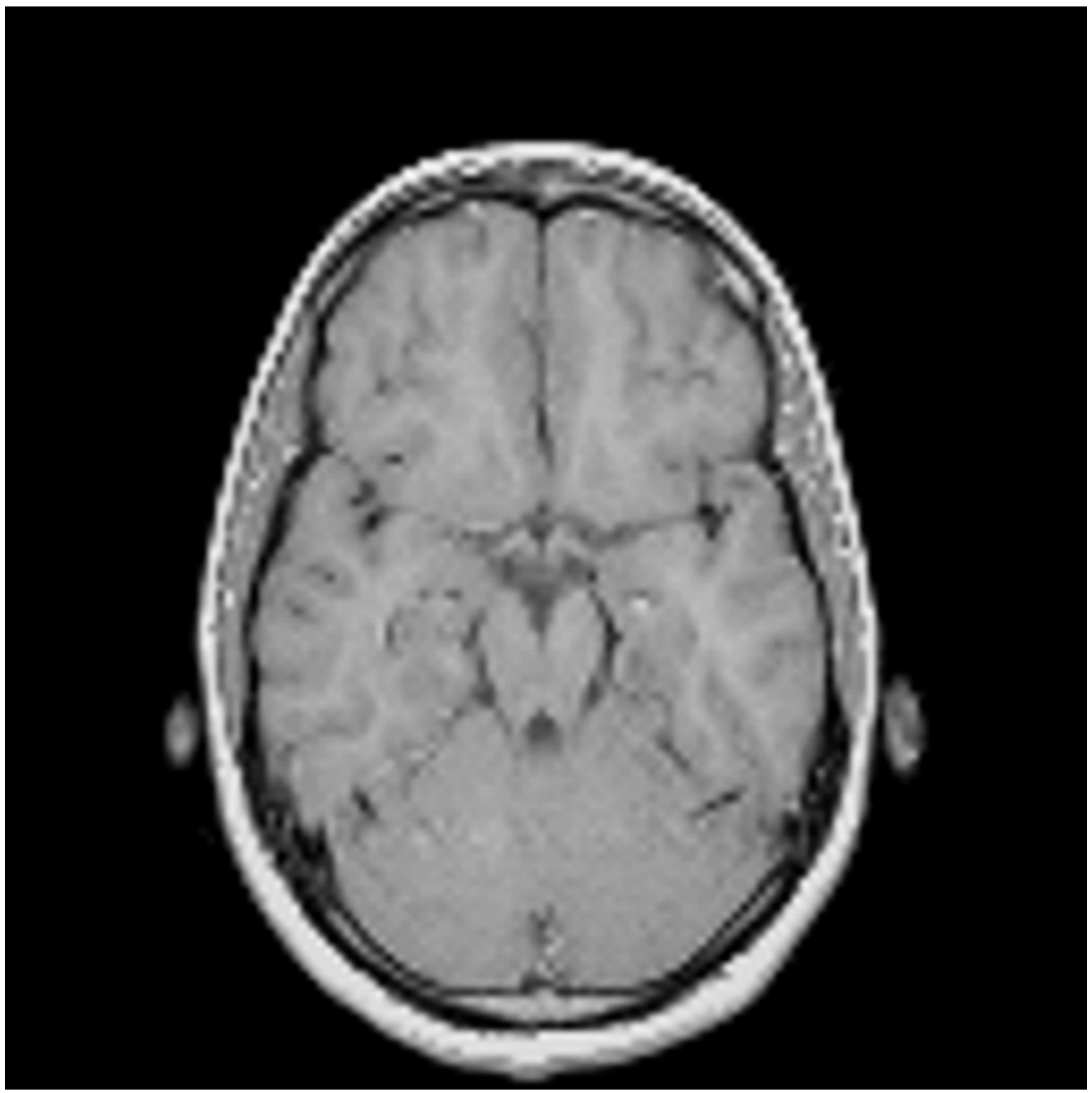
$$\otimes \quad g_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Sobel Gradient Operator: X-gradient Example



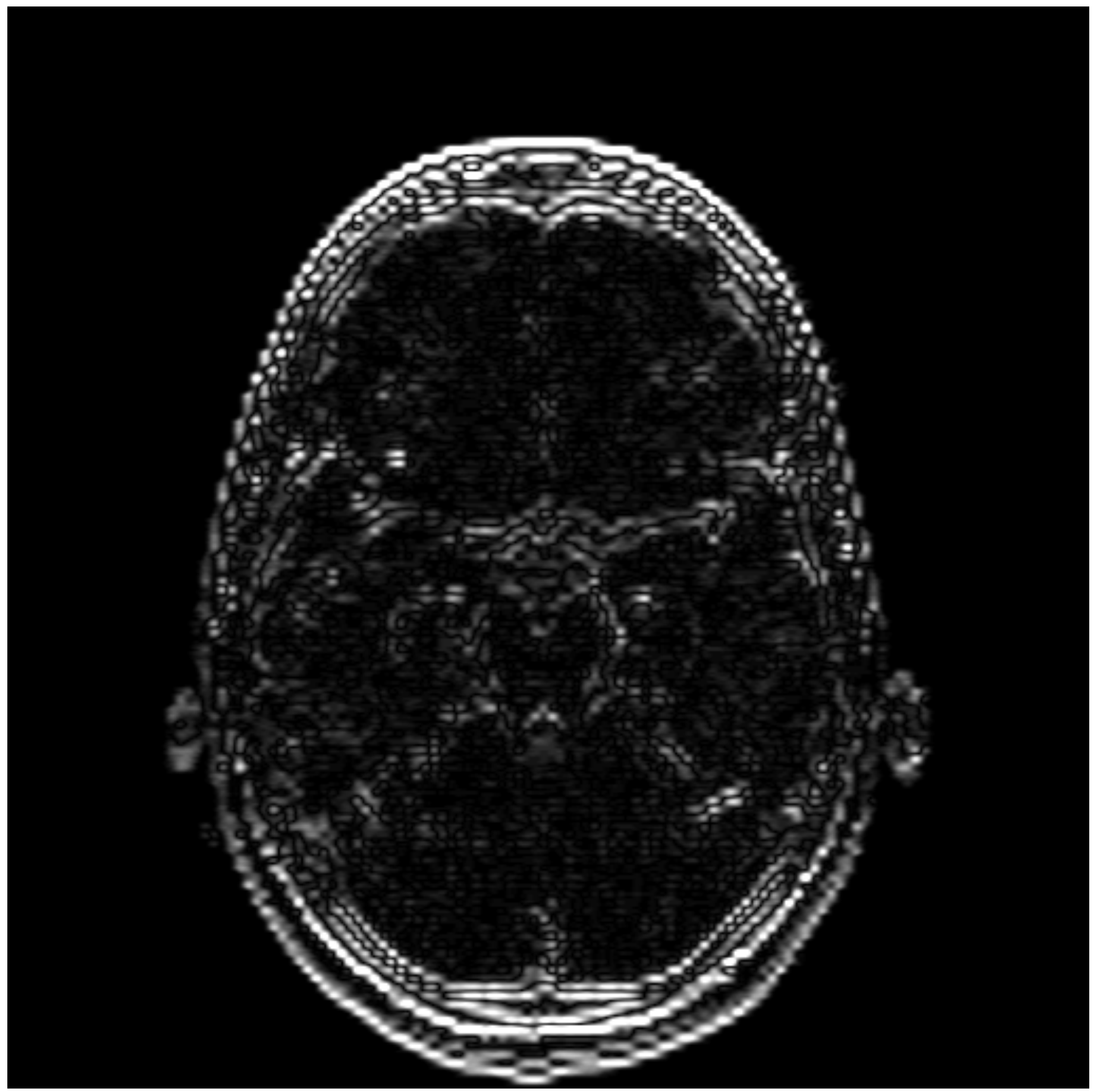
g_X

Sobel Gradient Operator: Y-gradient Example



$$\otimes \quad g_Y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & 1 \end{bmatrix}$$

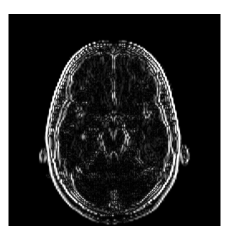
Sobel Gradient Operator: Y-gradient Example



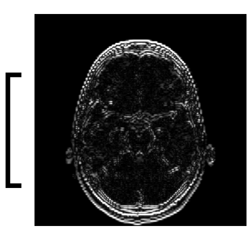
g_Y

Gradient Operator Example

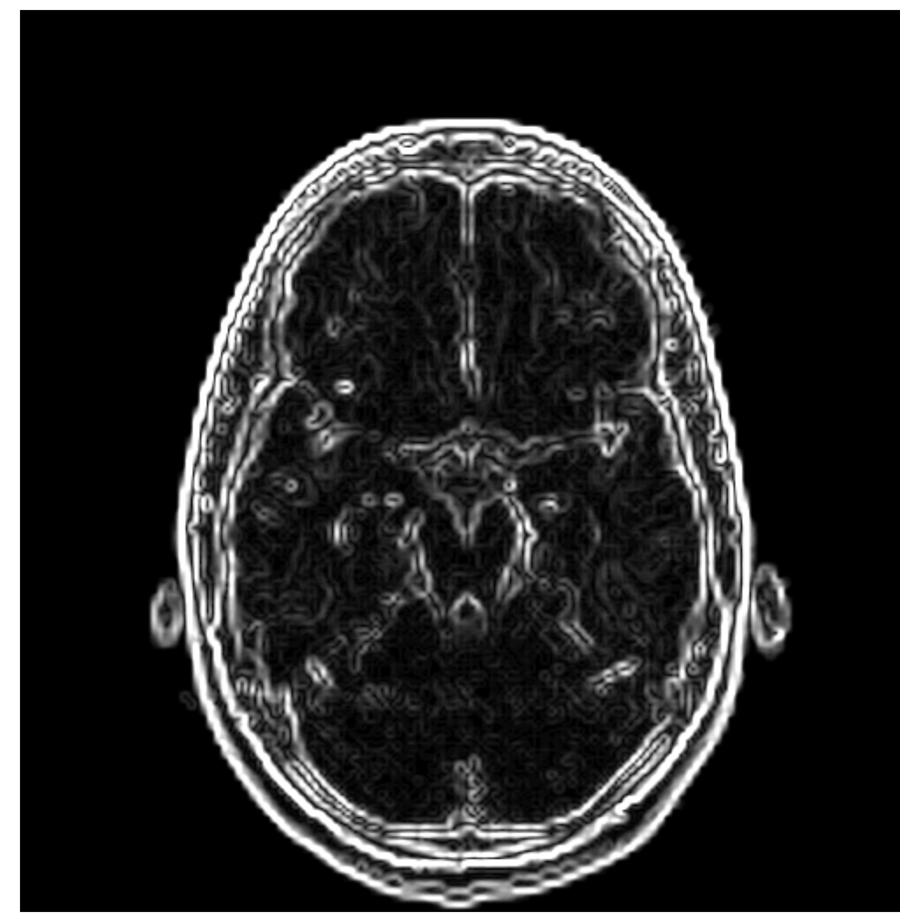
$$\|\nabla G\| = \sqrt{\left([\ \ ]^2 + [\ \ ]^2 \right)} =$$



g_X



g_Y



Edge Detectors

- Edges can be helpful for defining the borders of a region; e.g., a piece of tissue which may be of our interest!
- Furthermore, they give an aid for visualizing what we want to segment.

Edge Detectors: Canny

- 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 Detectors

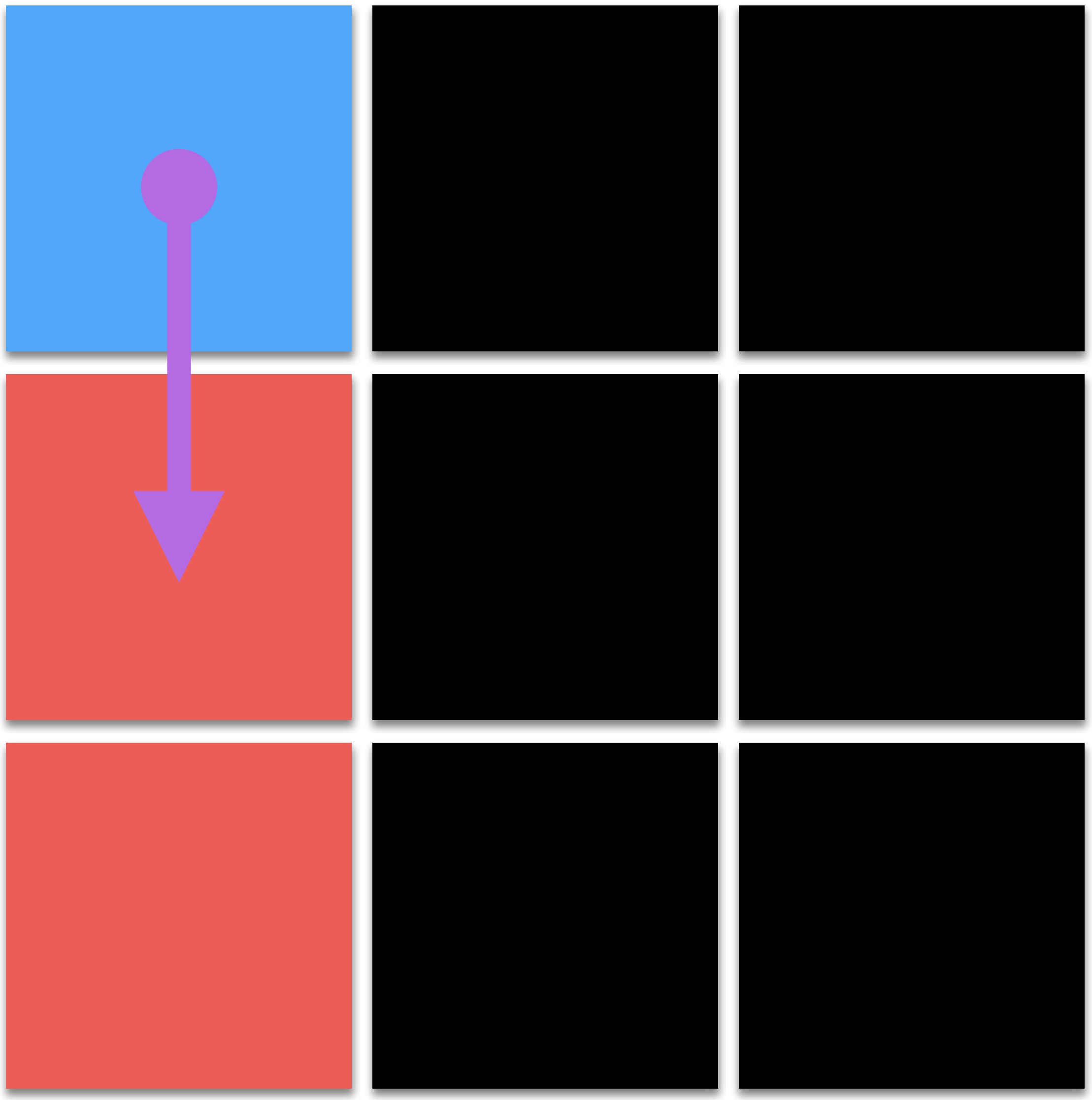
- Steps:
  - Compute gradients (magnitude and angle of orientation [**atan2**]).
  - Non-maximum suppression —> remove low power stuff.
  - Apply double thresholding; classification: strong (1.0), weak (0.5), and no edge (0.0):

$$E(x,y) = \begin{cases} 1 & \text{if } I(x,y) > t_2 \\ 0.5 & \text{if } I(x,y) > t_1 \wedge I(x,y) \leq t_2 \\ 0 & \text{otherwise} \end{cases}$$

  - Edge tracking; a weak edge is a strong one if it is connected to a strong edge!

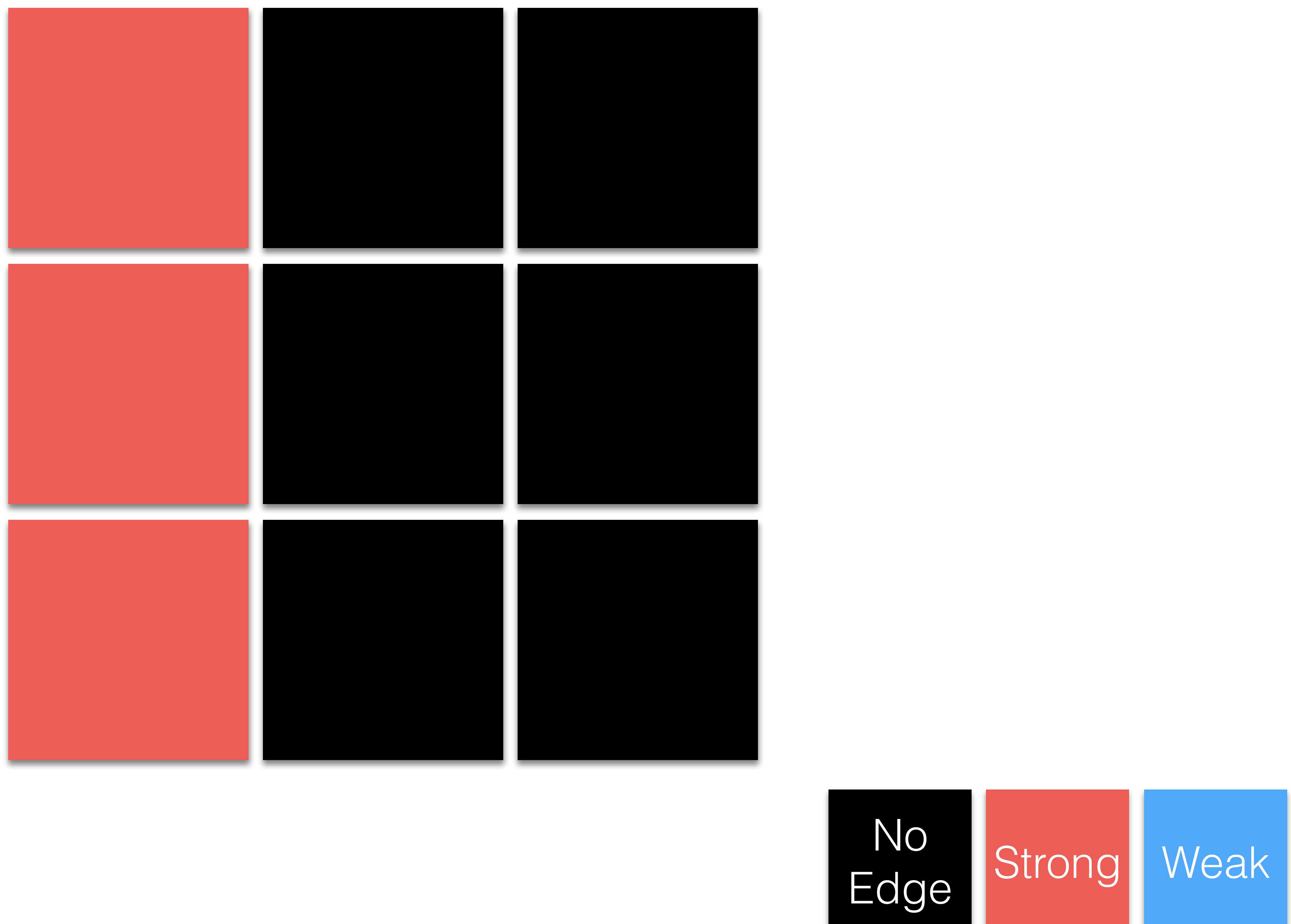
Edge Detectors:
Edge Tracking Example 1

The **gradient** (purple arrow) points towards a pixel that is a strong edge!



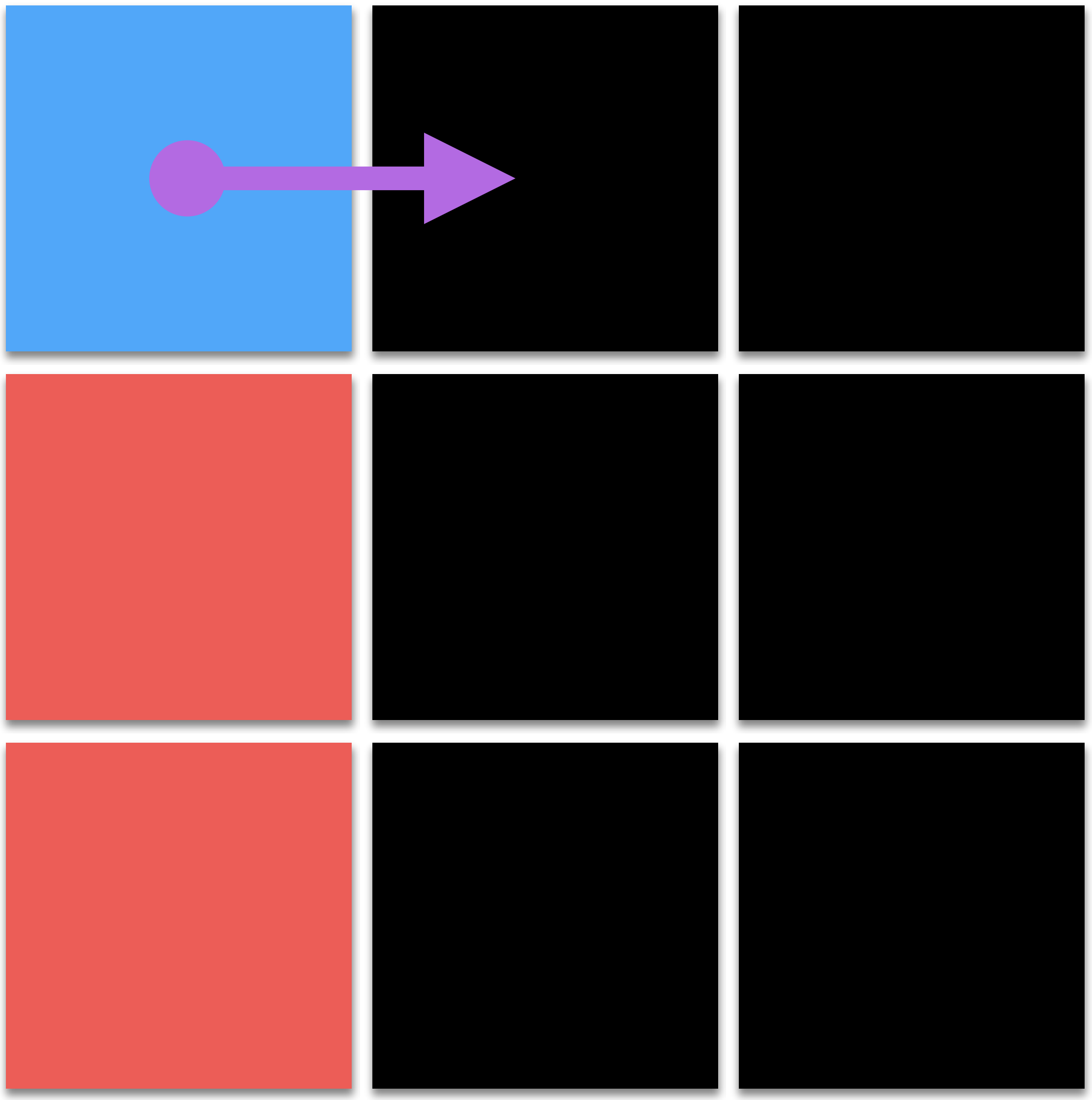
Edge Detectors: Edge Tracking Example 1

This means that strong and weak edges needs to be connected!



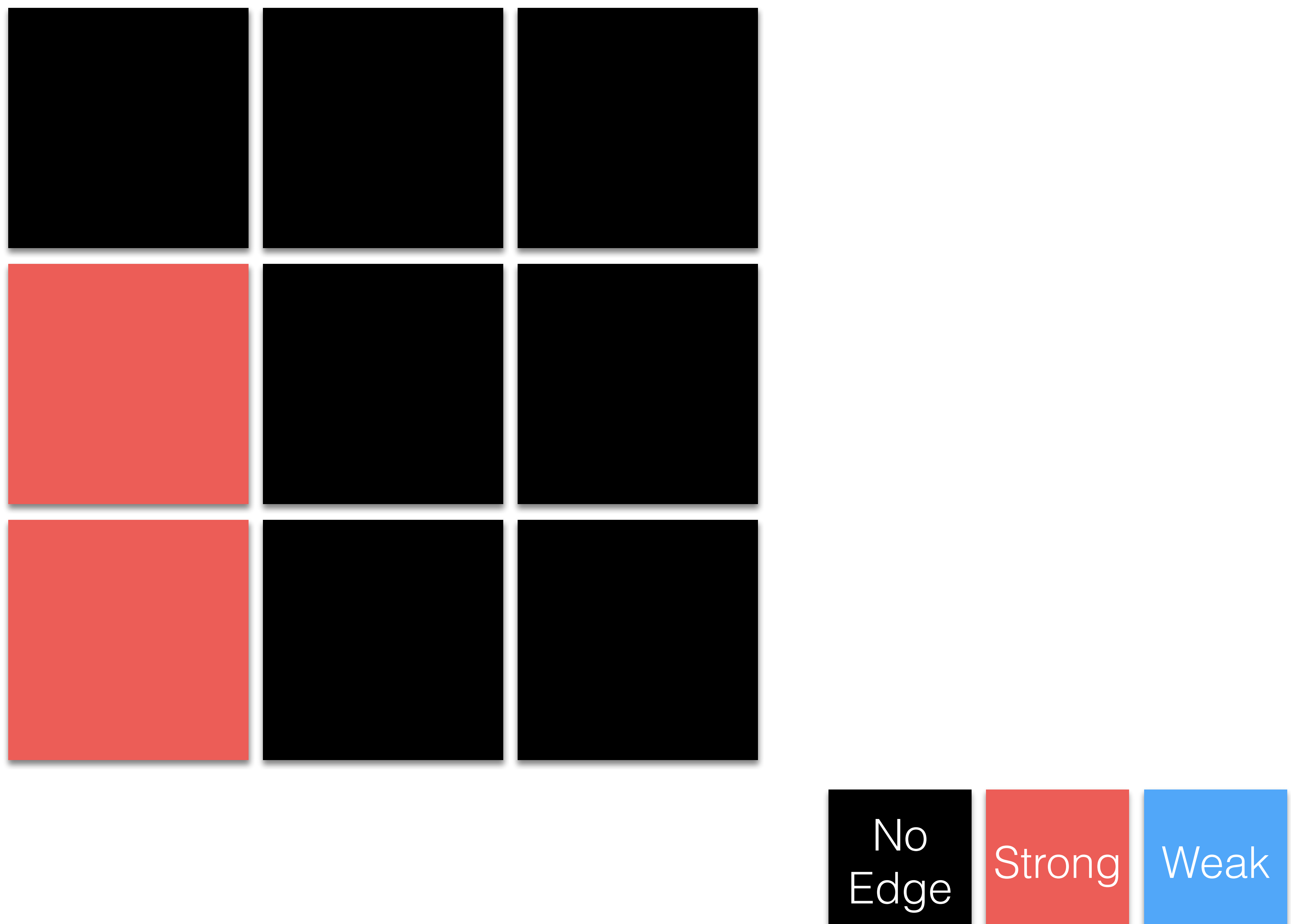
Edge Detectors: Edge Tracking Example 2

The **gradient** points towards a pixel which is not an edge

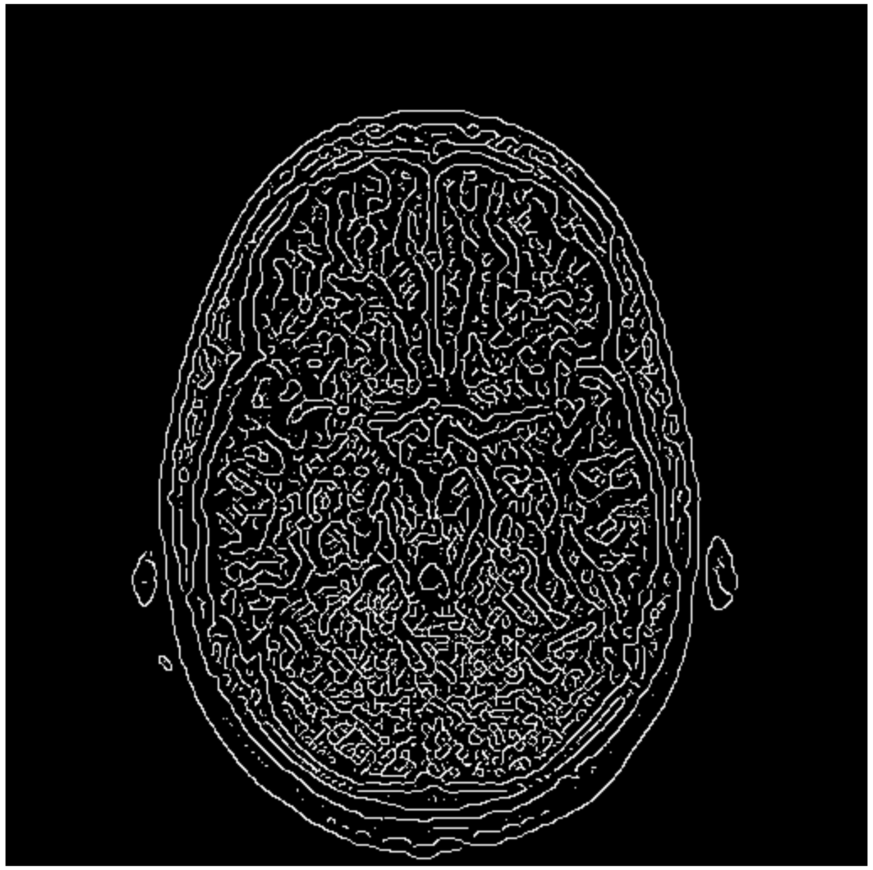


Edge Detectors:
Edge Tracking Example 2

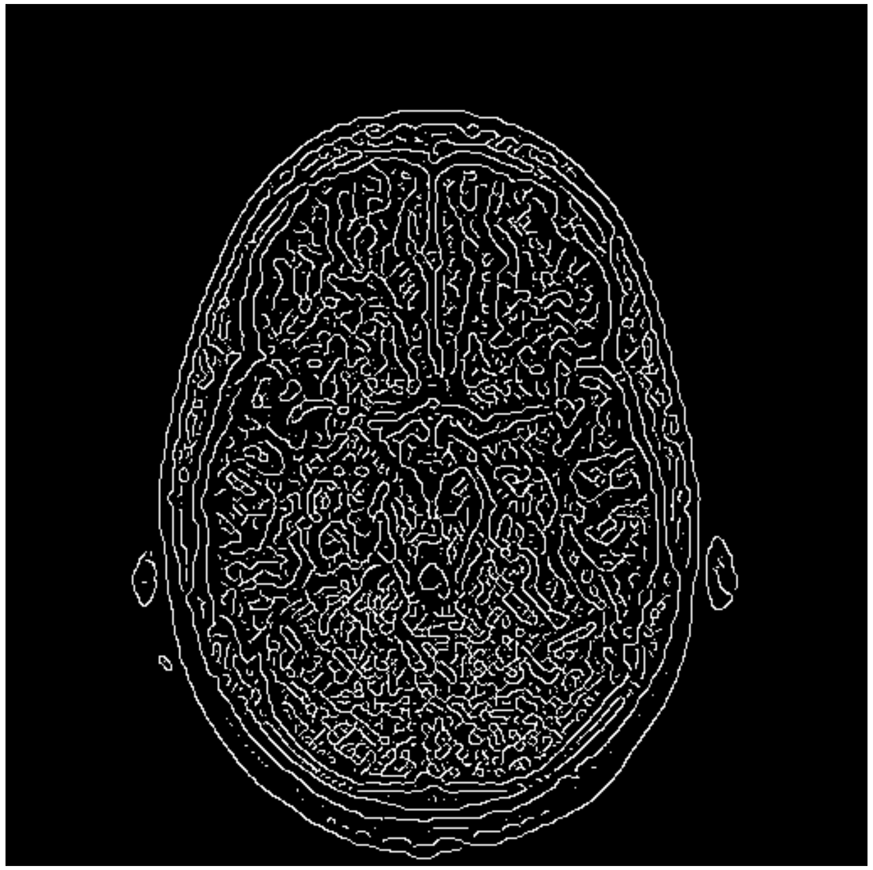
This means that the strong and the weak edges are not belonging to the same path, and the weak edge is set to “no edge”!



Edge Detector Example



$t_2 = 0.001$



$t_2 = 0.01$

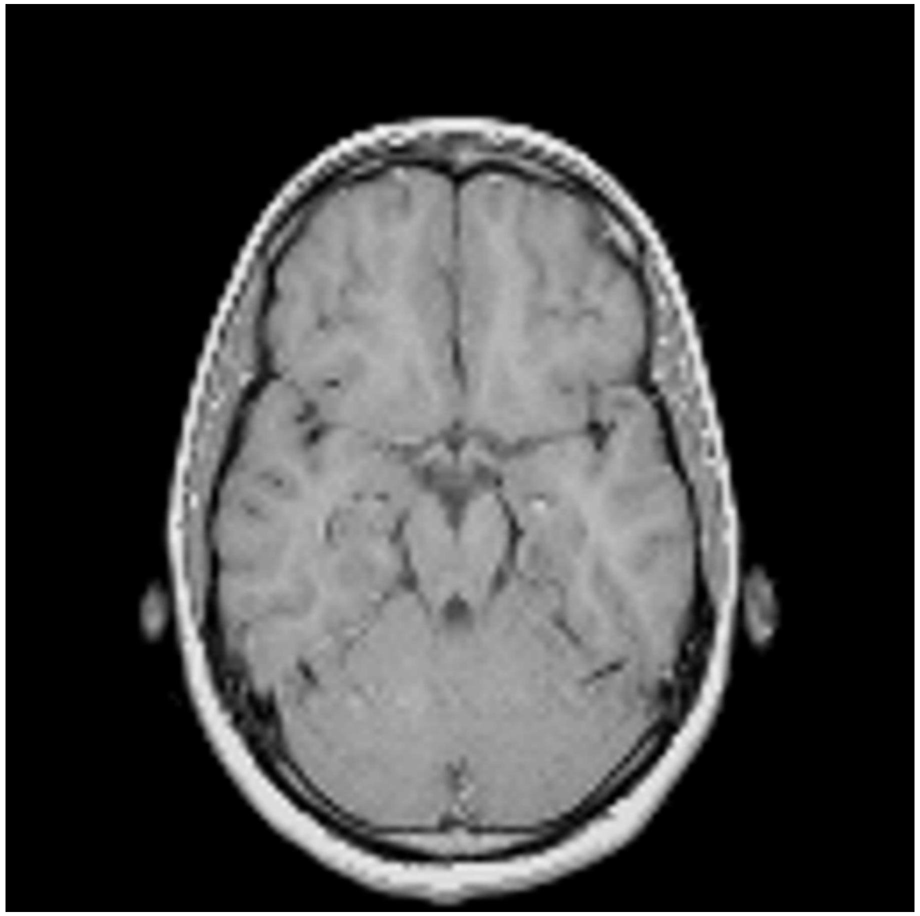
$t_2 = 0.1$

Laplacian Filter

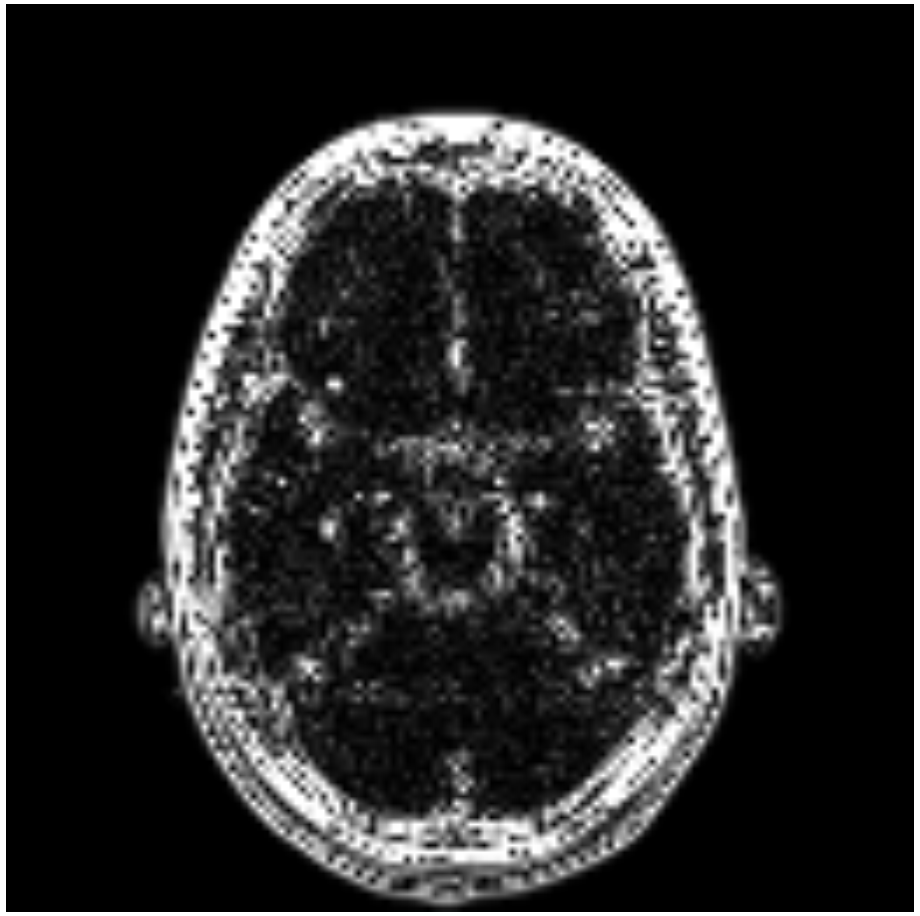
- If you really want… we can also define a Laplacian operator… Why?
  - The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection
  - We can have two kernels (for 4 or 8 connected neighbors):

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

Laplacian Filter Example

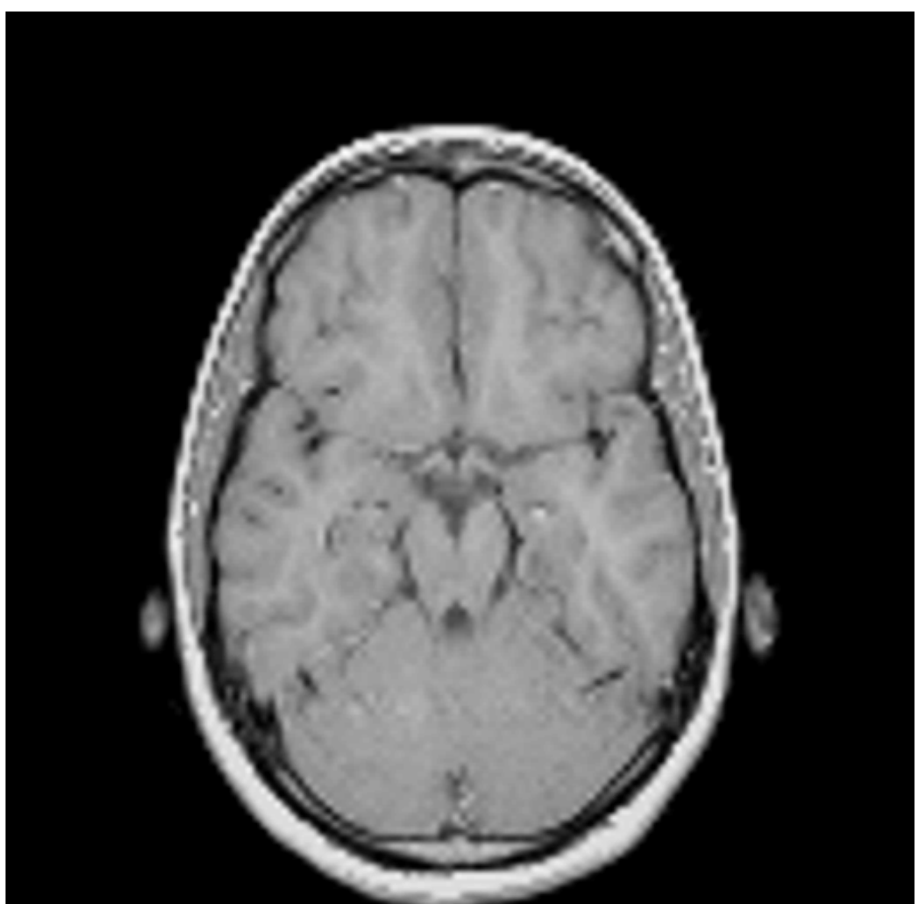


$$\otimes \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

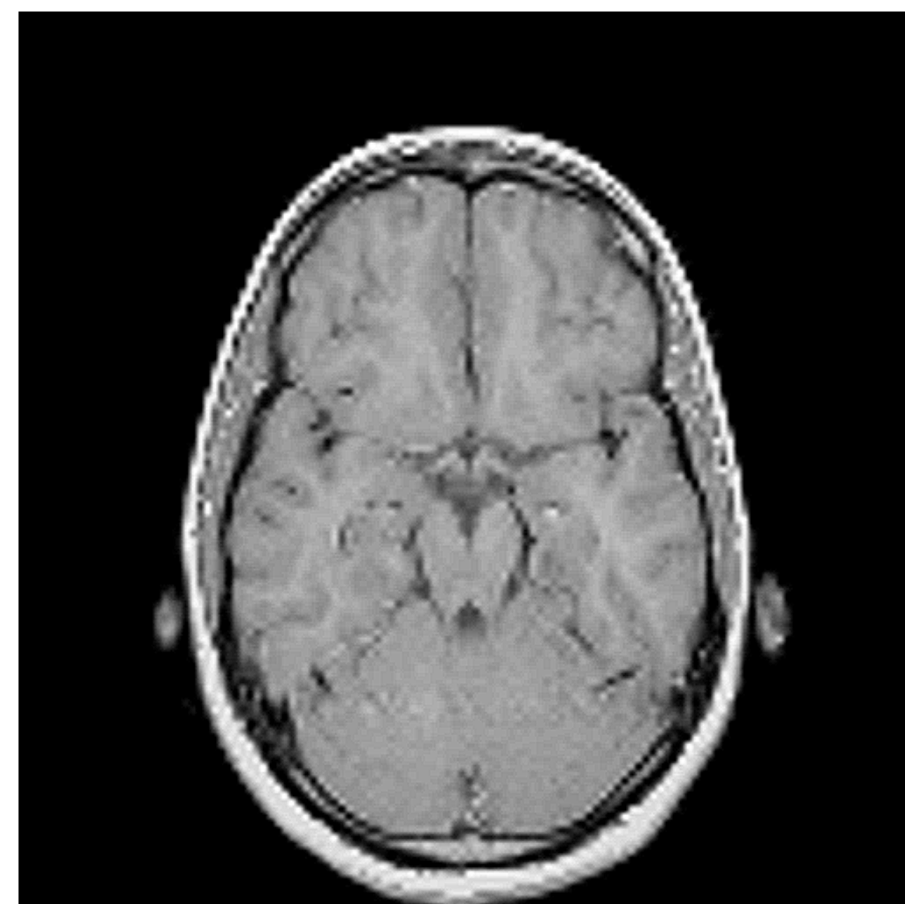


Laplacian Filter Bonus

- With a small change ($g_{L4}(2,2) = 5$) we can increase sharpness in the image:

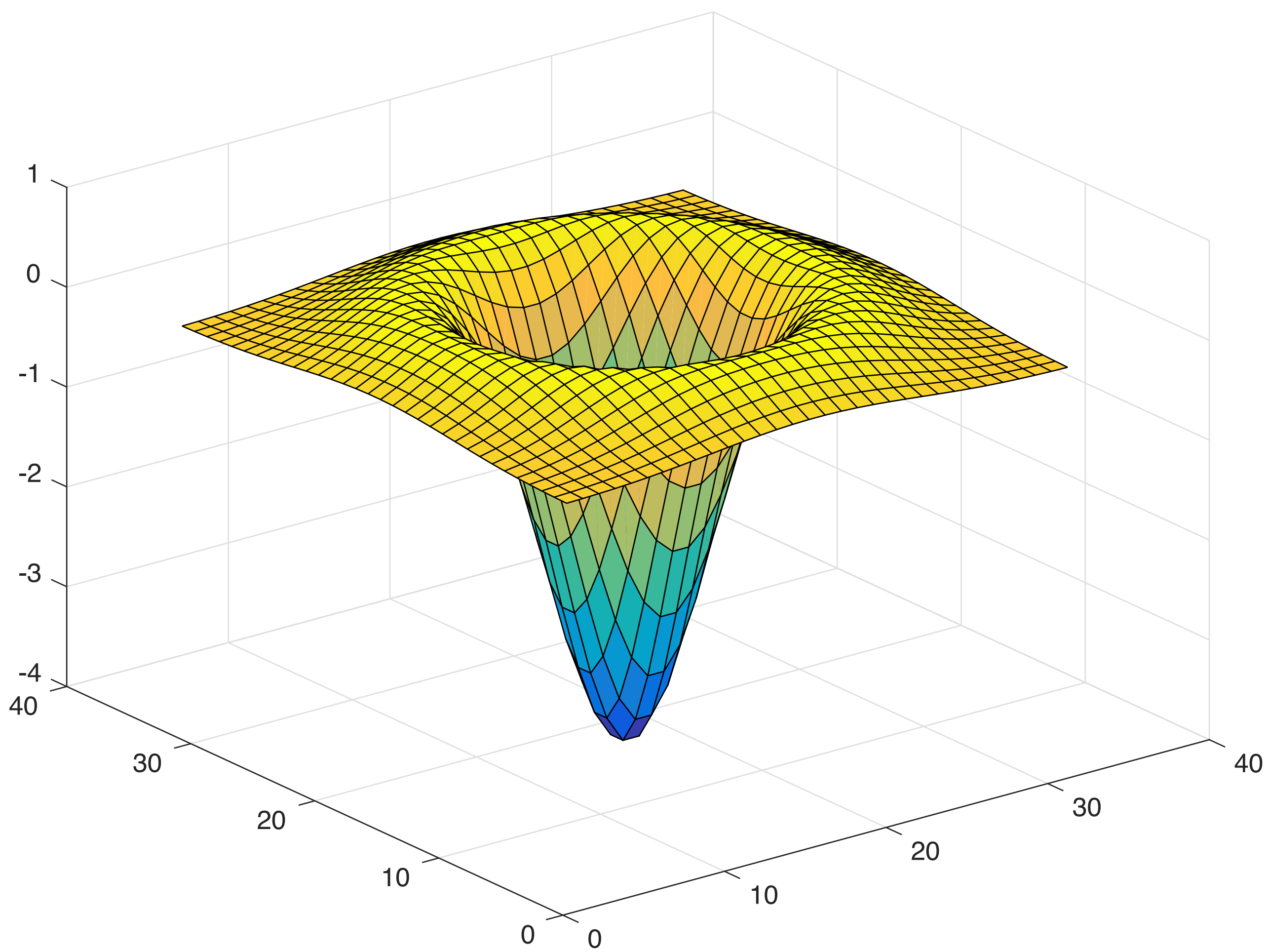


$$\otimes \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$



Laplacian Filter Extra

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



Box Filter

- This is a very simple filter low-pass filter of size NxM:

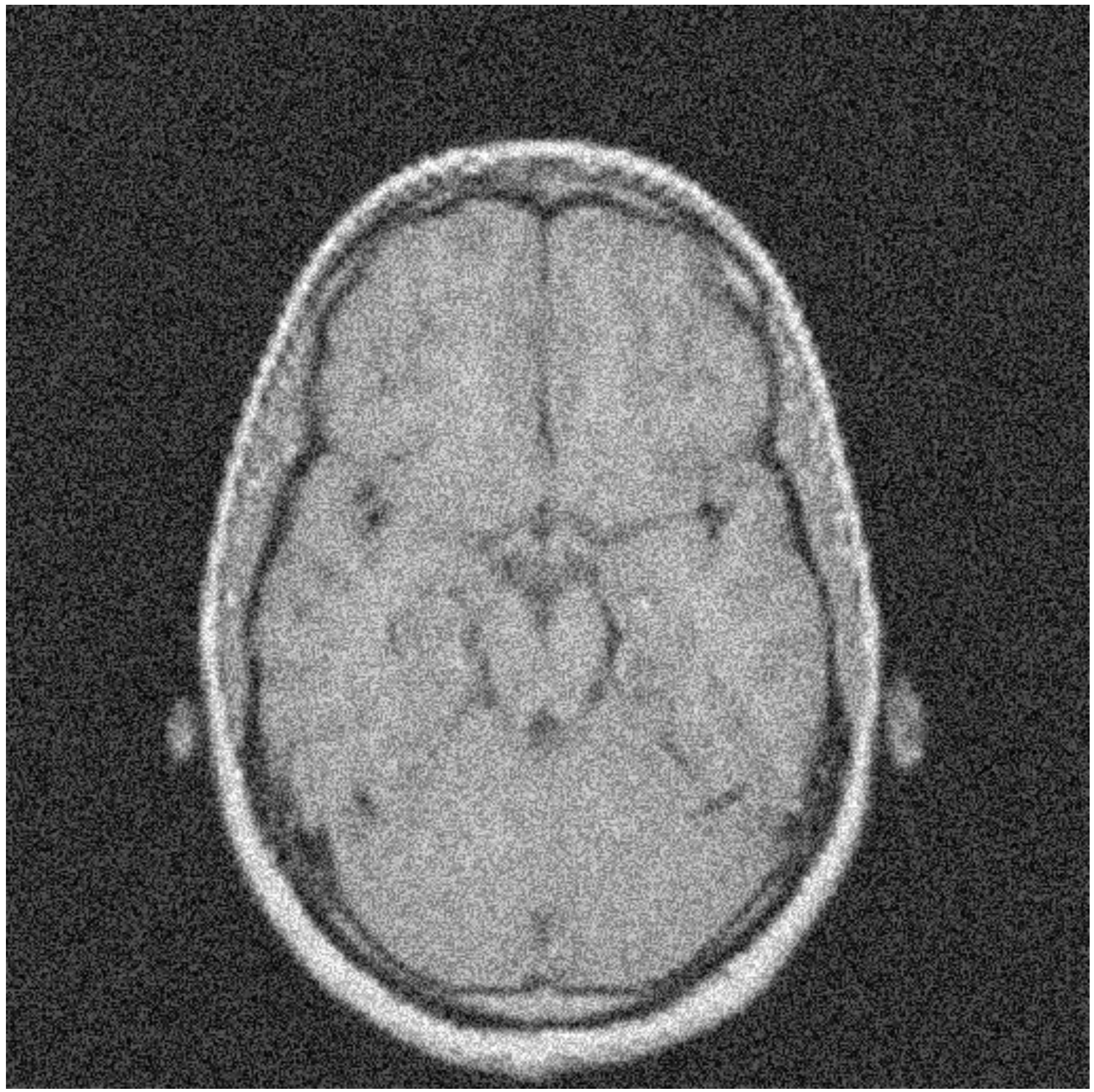
$$g[k, l] = 1 \quad \forall k \wedge \forall l$$

- What does it do? It blurs the signal!

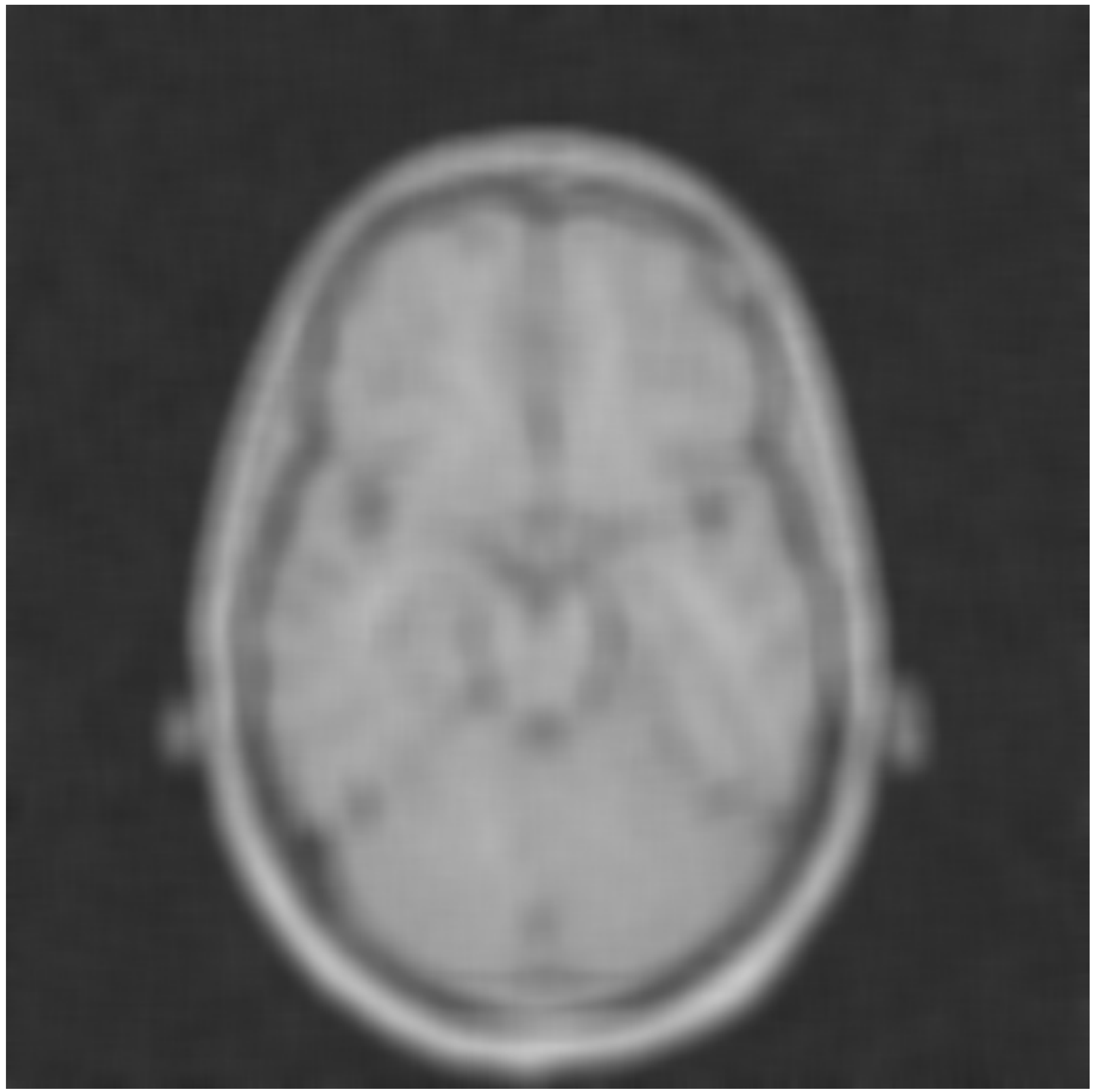
- This kernel has to be normalized:

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

Box Filter Example



Box Filter Example



Gaussian Filter

- We use a Gaussian kernel defined as

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

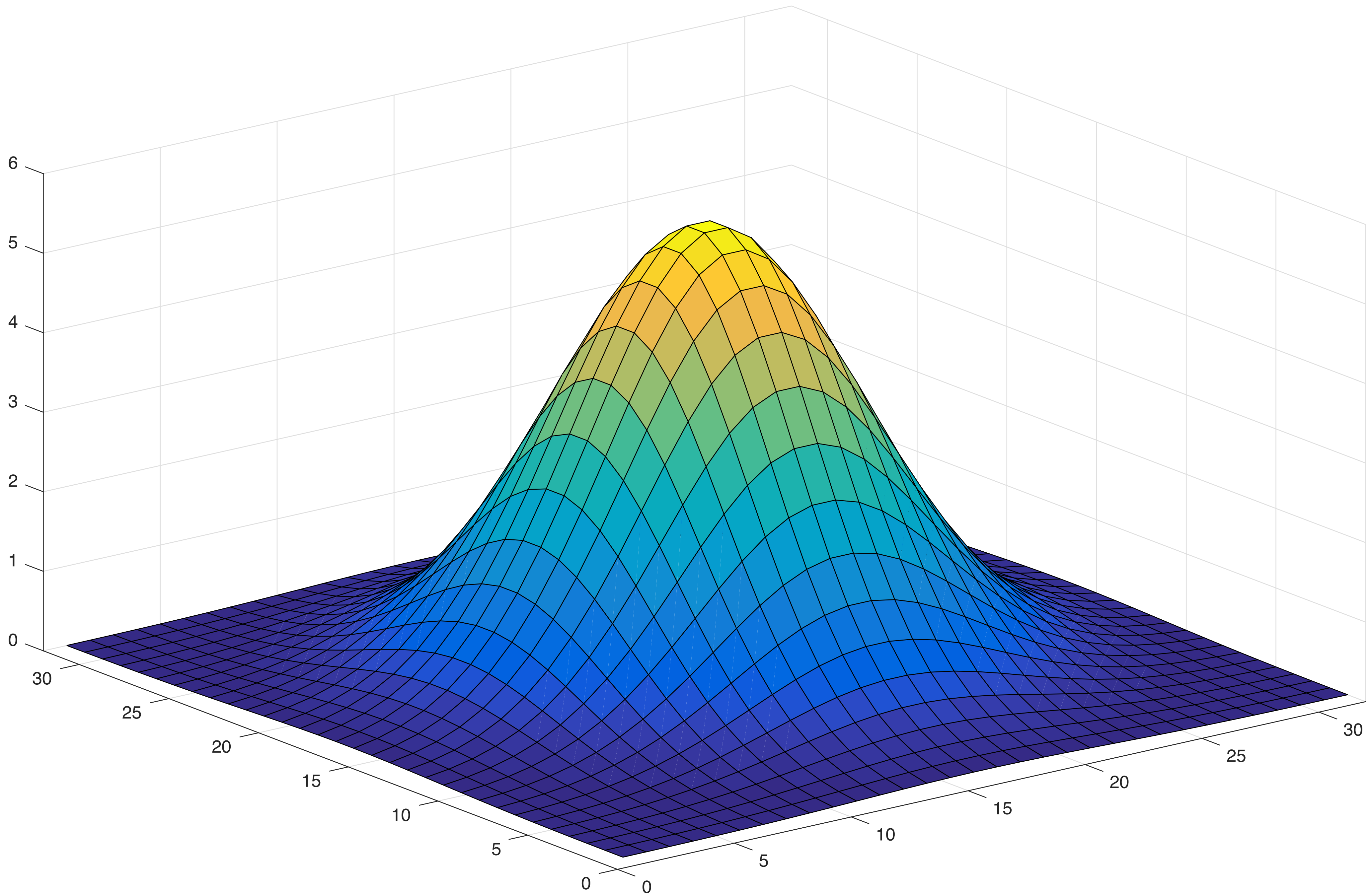
- where G is a classic Gaussian function:

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

- 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: Example of a 2D Kernel



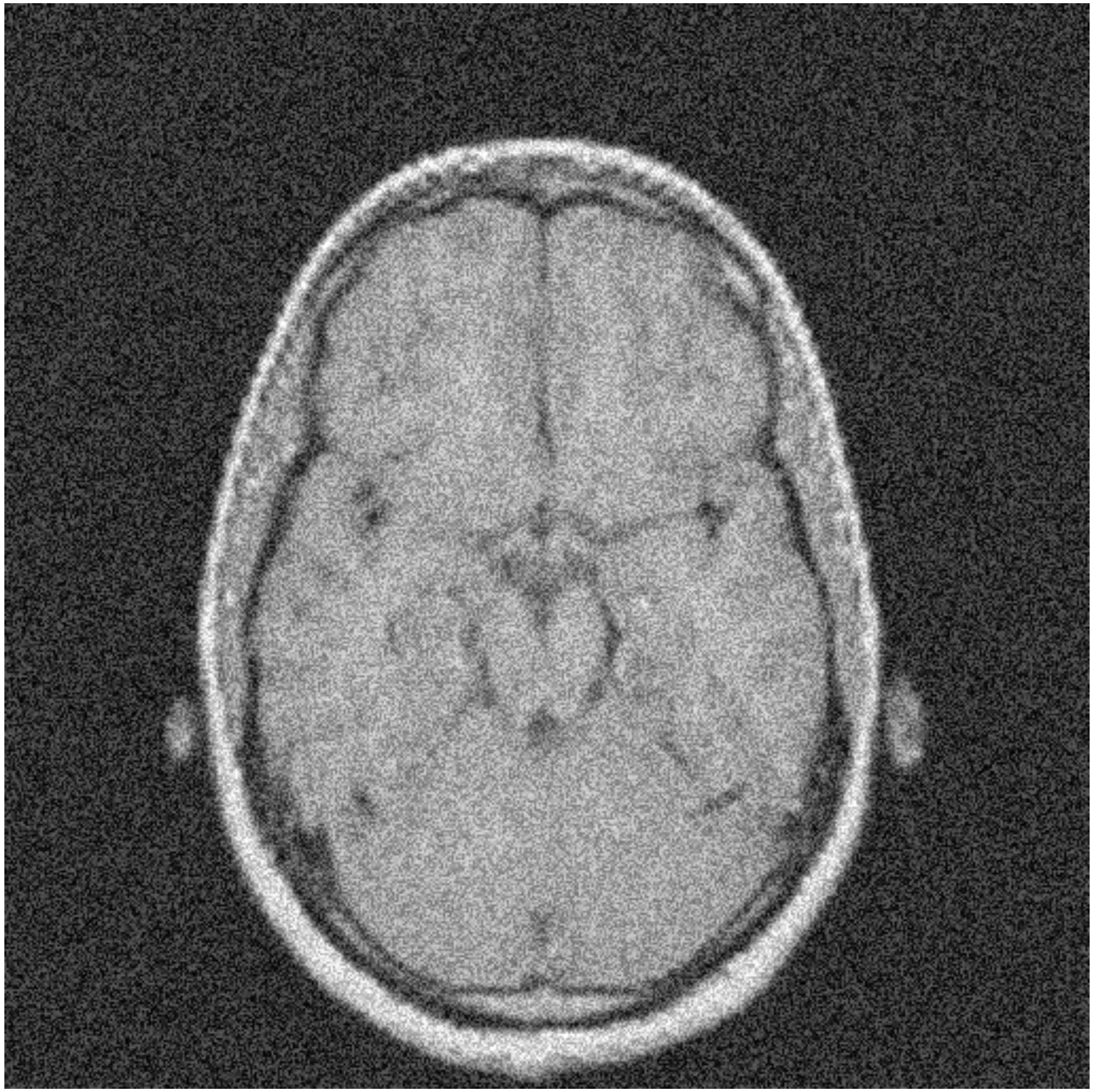
Gaussian Filter: how large?

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

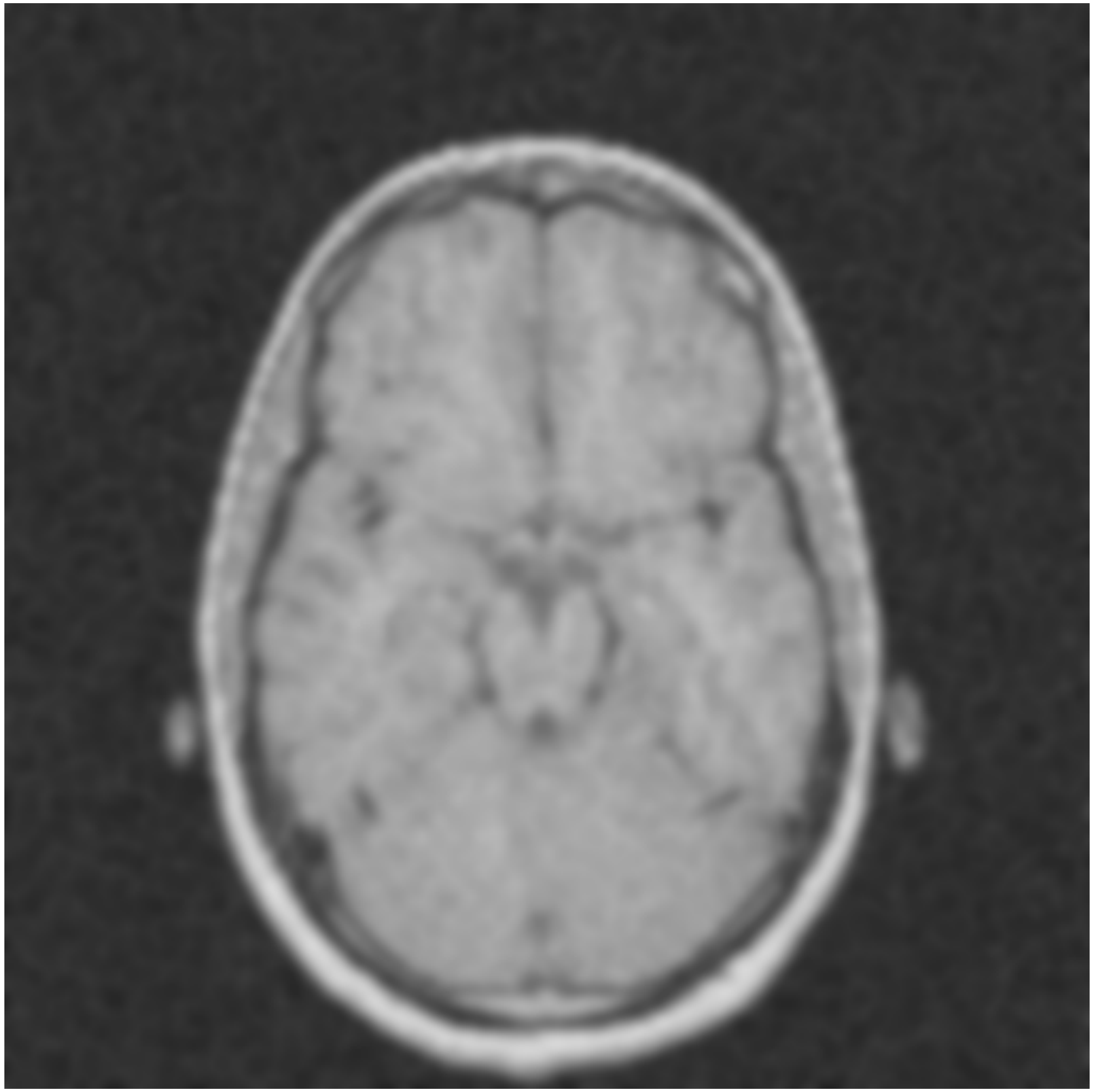
$$N = M = \frac{5}{2}\sigma \longrightarrow \text{98\% of energy}$$

- 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



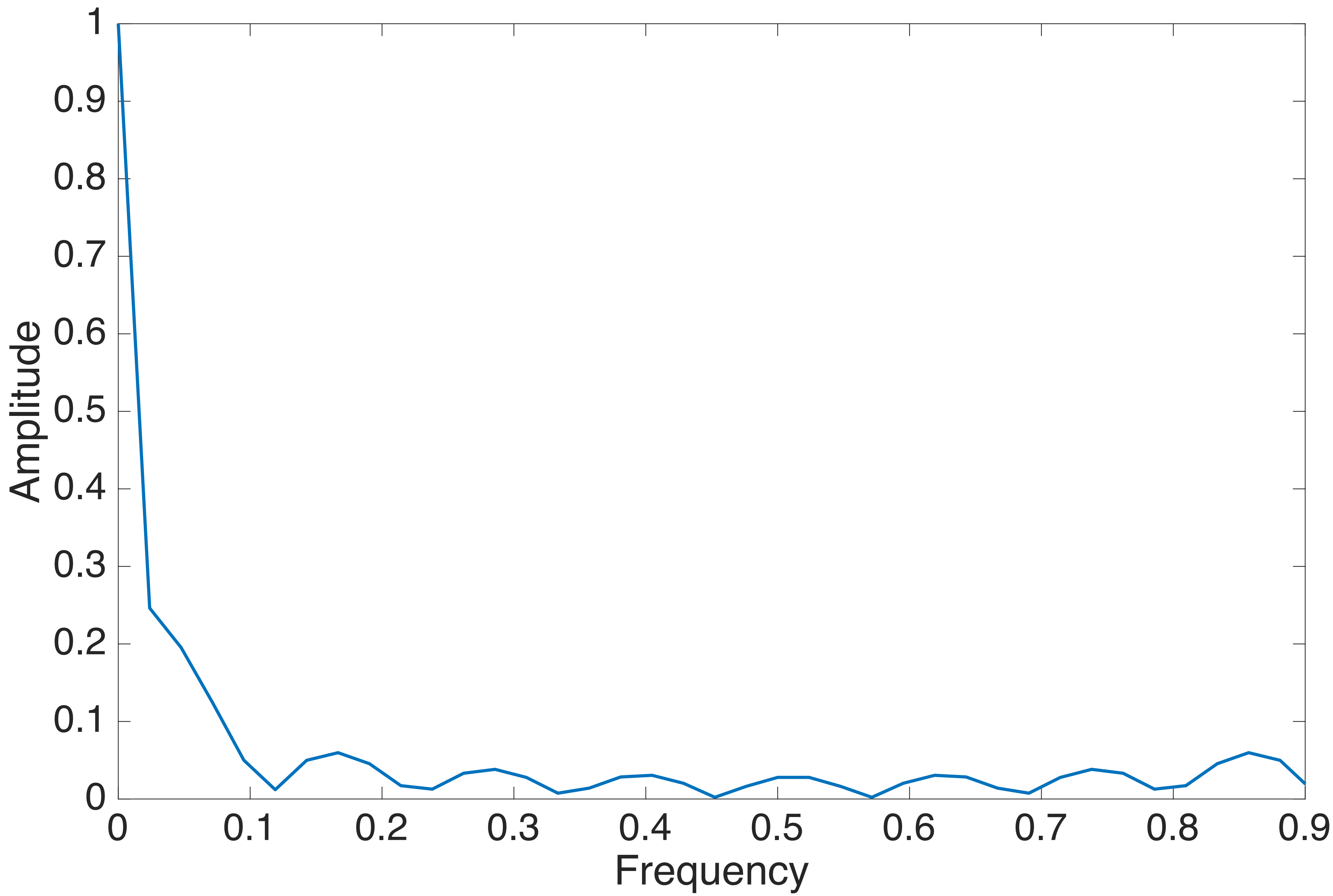
Gaussian Filter Example



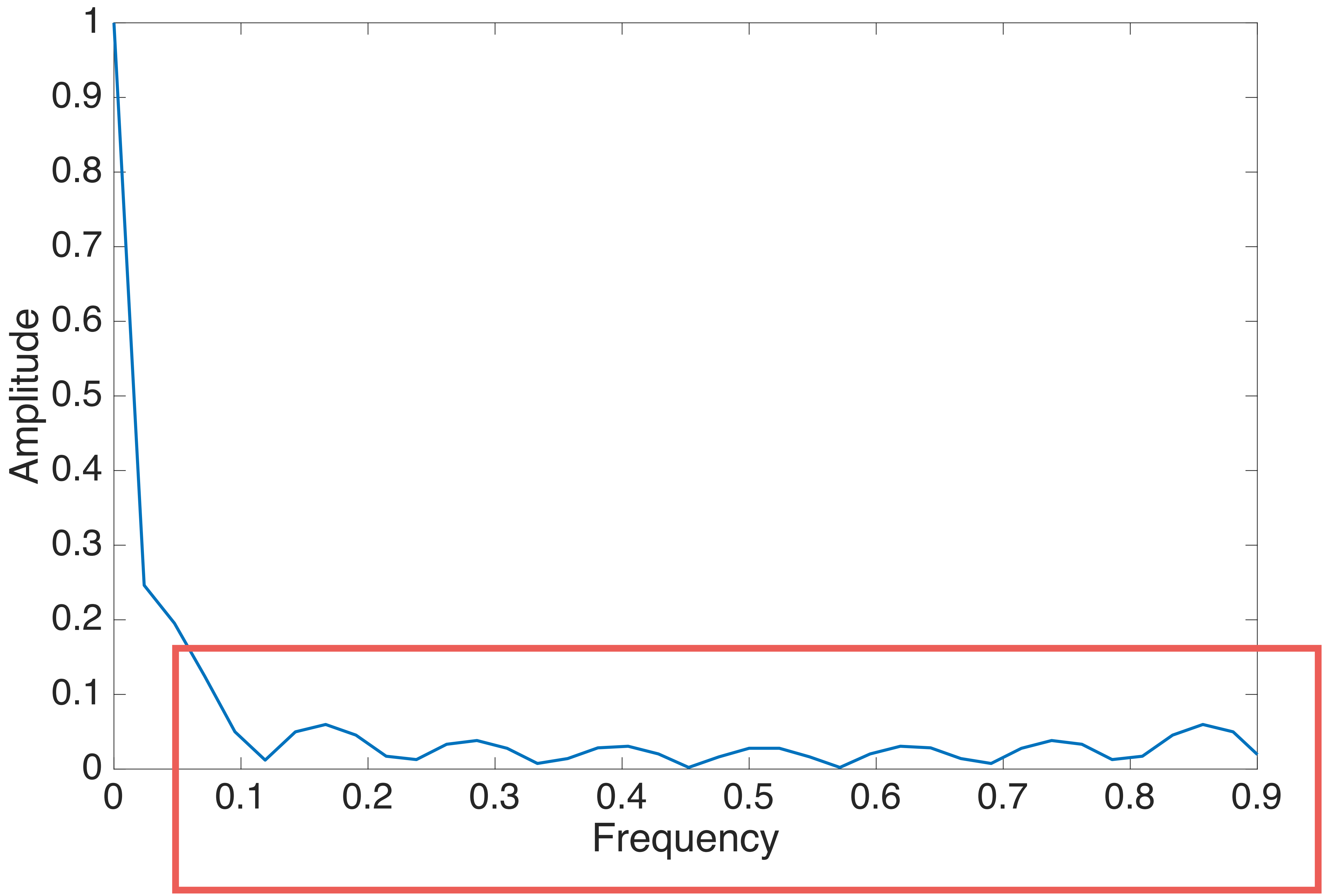
Box vs Gaussian

- 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!**

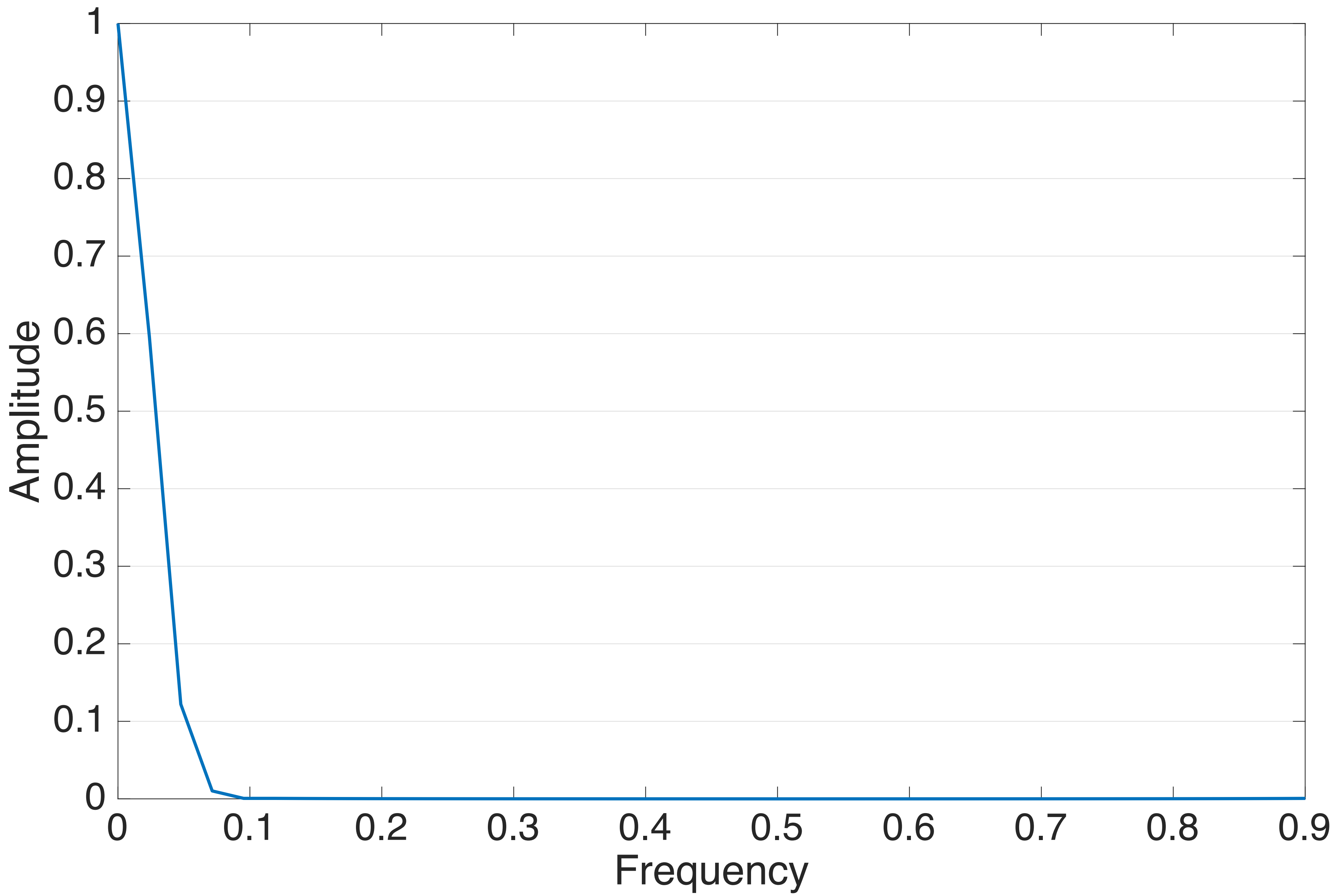
Box vs Gaussian Example



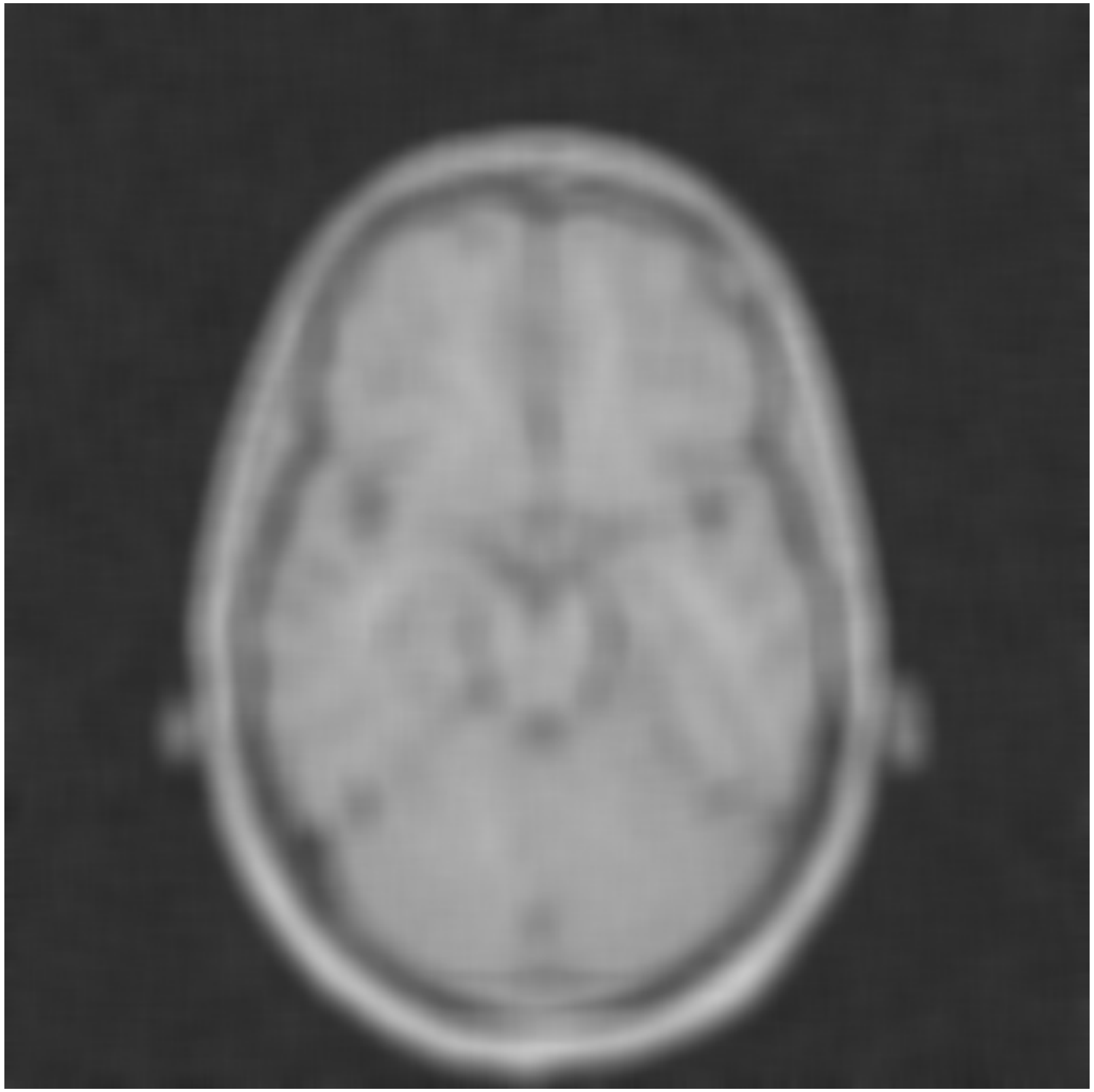
Box vs Gaussian Example



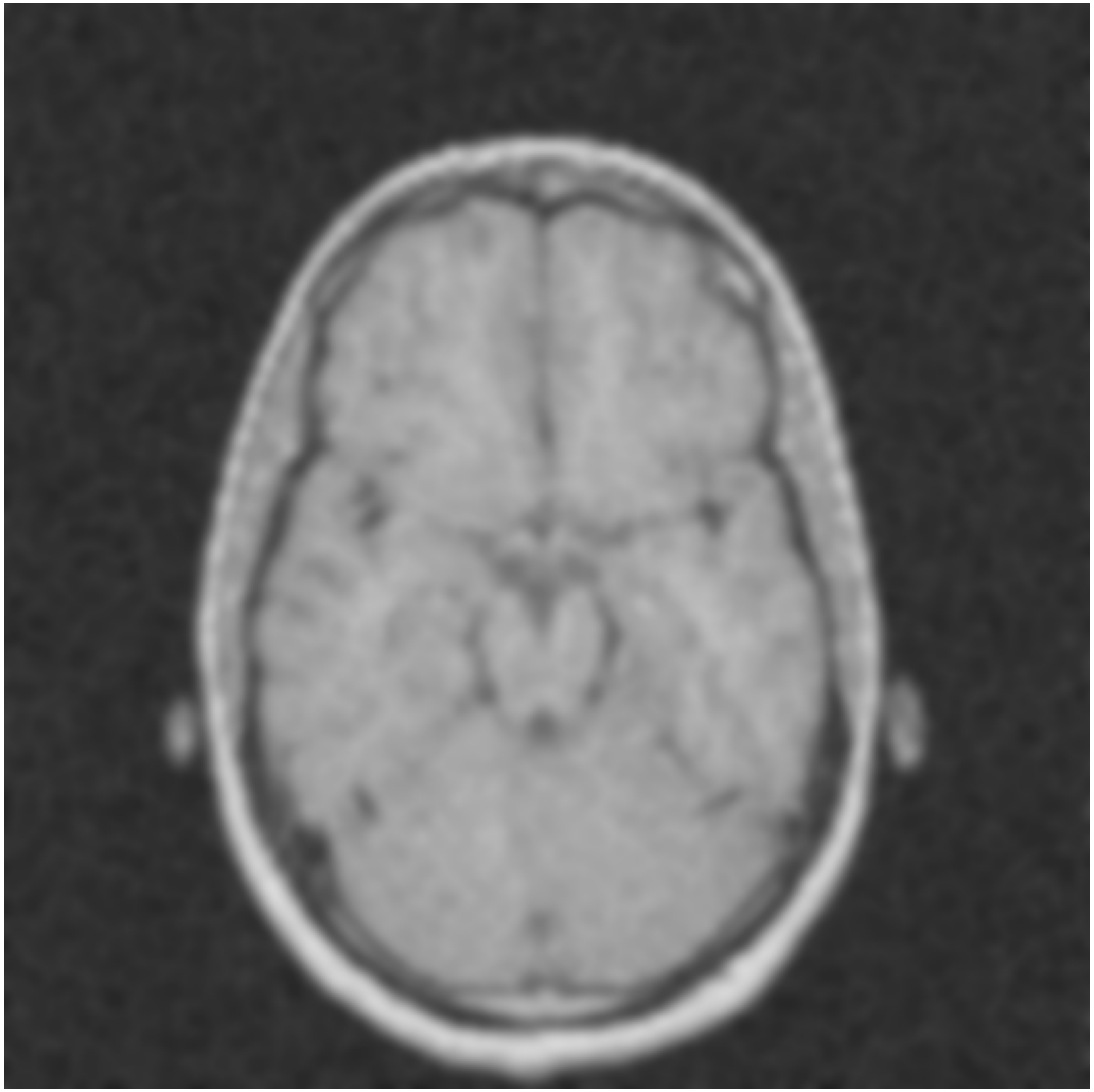
Box vs Gaussian Example



Box vs Gaussian Example

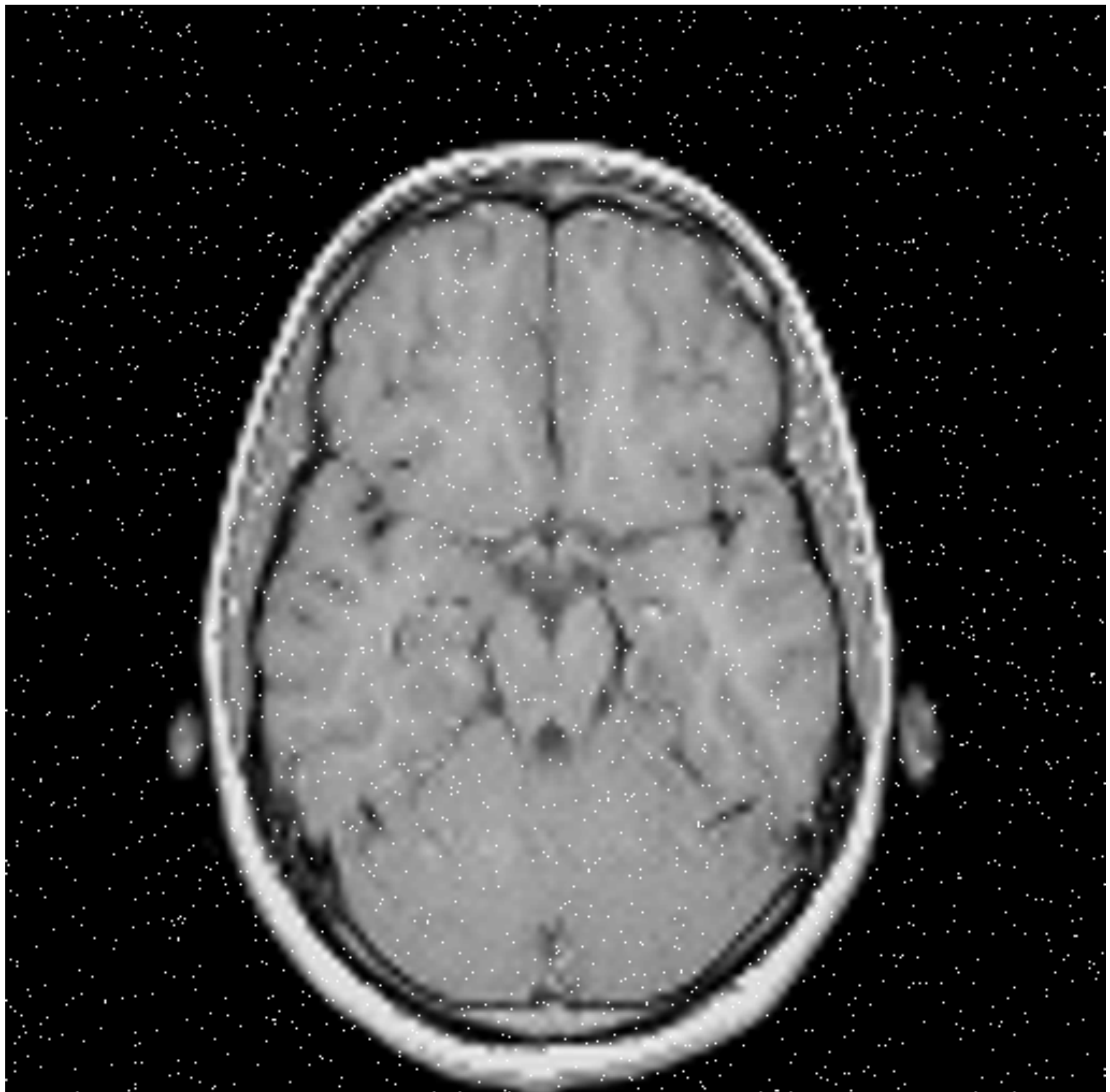


Box vs Gaussian Example

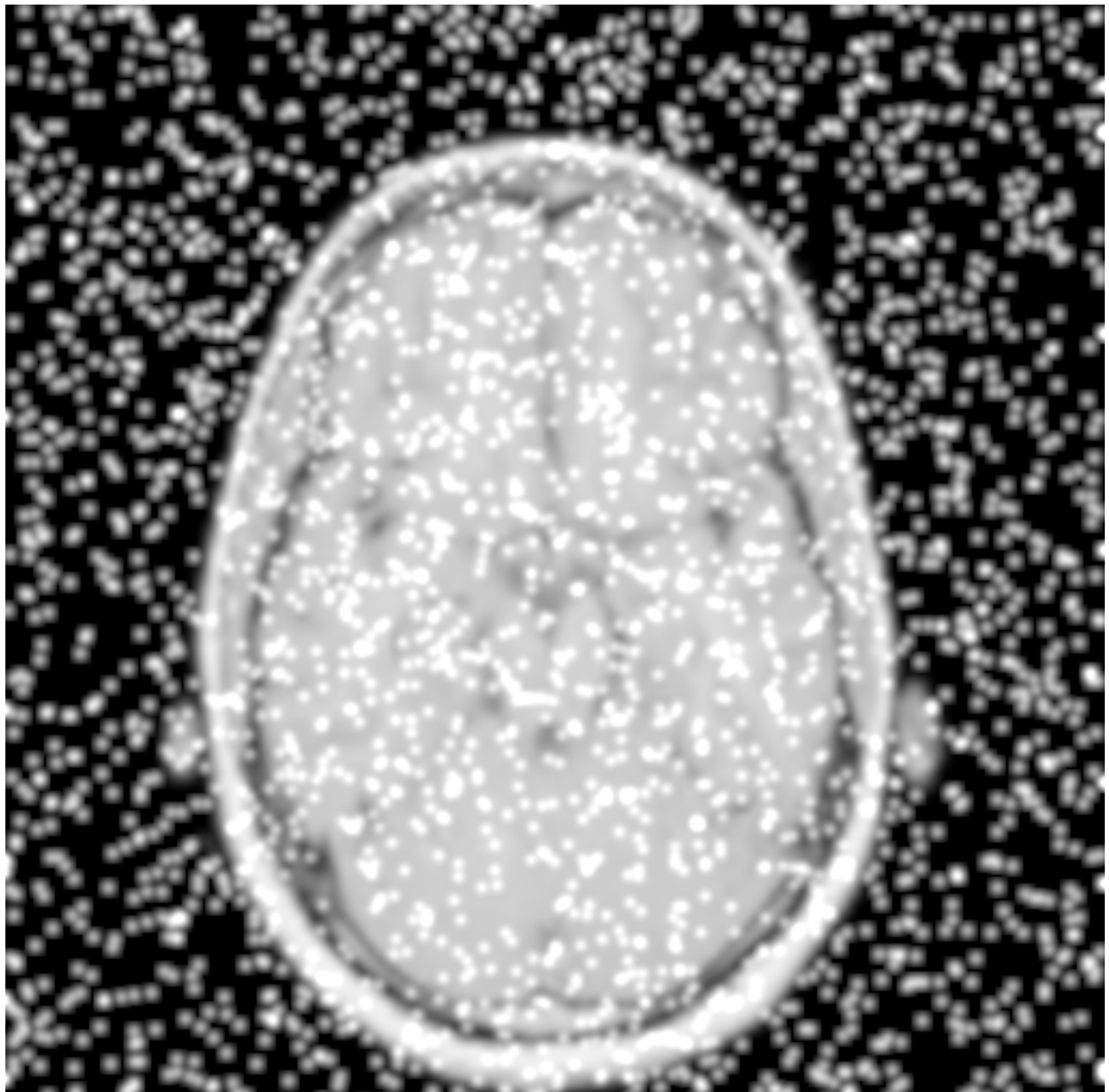


Non-Linear Filters

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

- How does it work?
  - We define the size of the filter; e.g., 9×9
  - For each pixel (i, j):
    - We collect all pixel values around (i, j)
    - We sort pixel values
    - We take the median value

Median Filter: Example

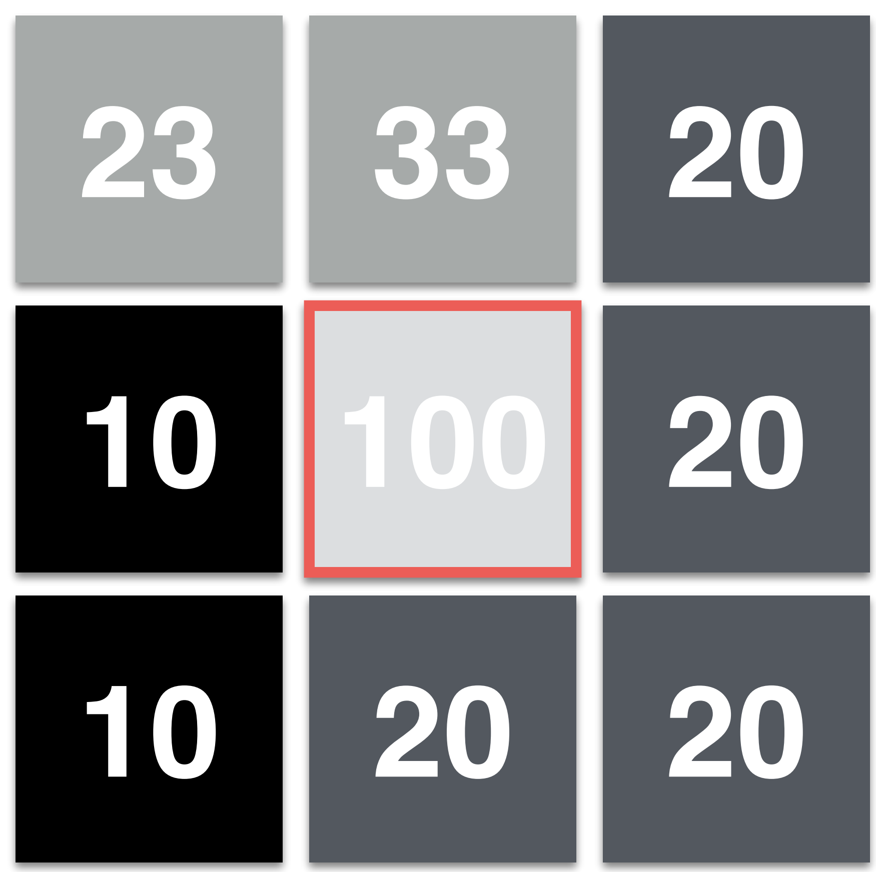
23	33	20
10	100	20
10	20	20

Let's consider the neighborhood of the central pixel; which has value 100

Median Filter: Example

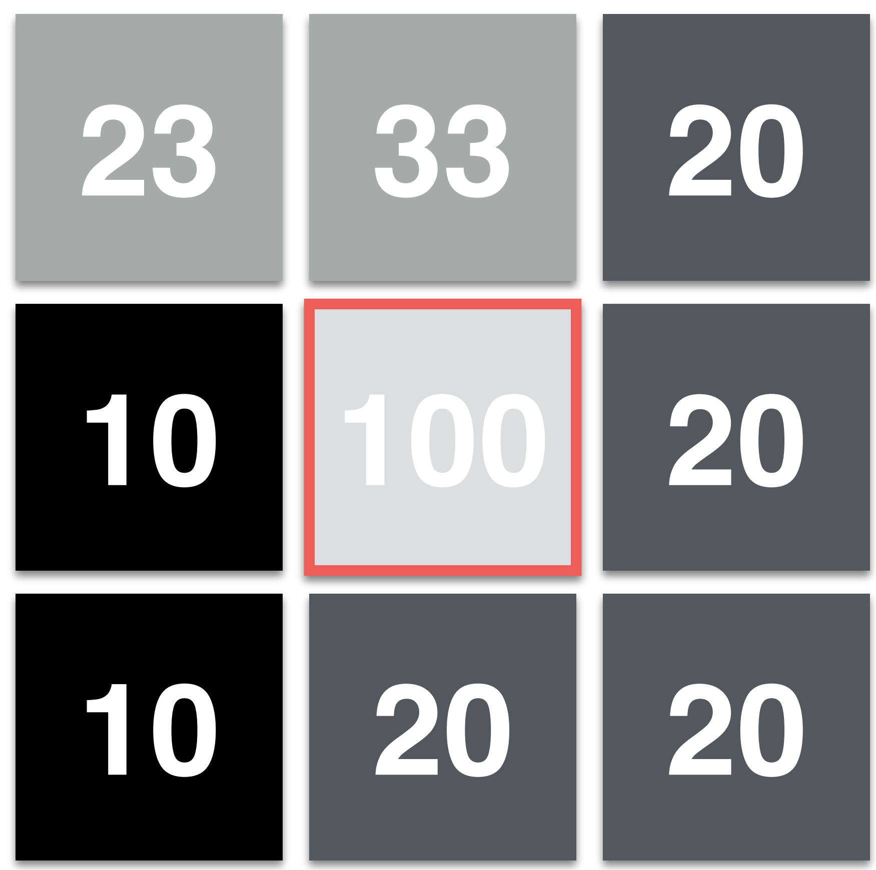
23	33	20
10	100	20
10	20	20

Median Filter: Example



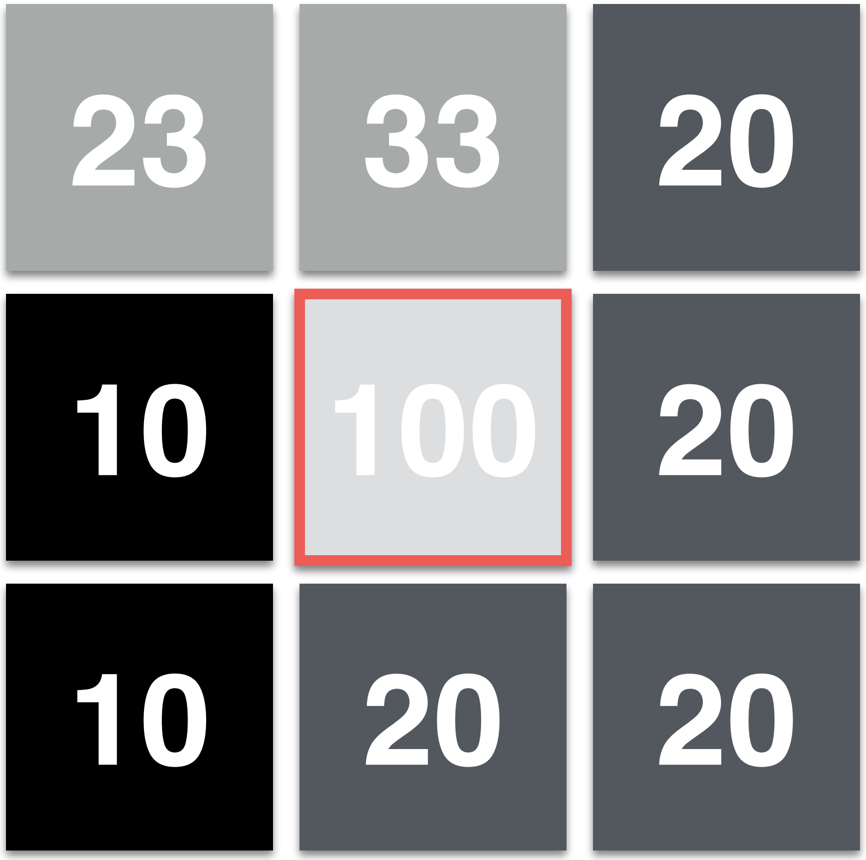
[23, 33, 20, 10, 100, 20, 10, 20, 20]

Median Filter: Example



[10, 10, 20, 20, 20, 20, 23, 33, 100]

Median Filter: Example



[10, 10, 20, 20, 20, 20, 23, 33, 100]

Median Value!

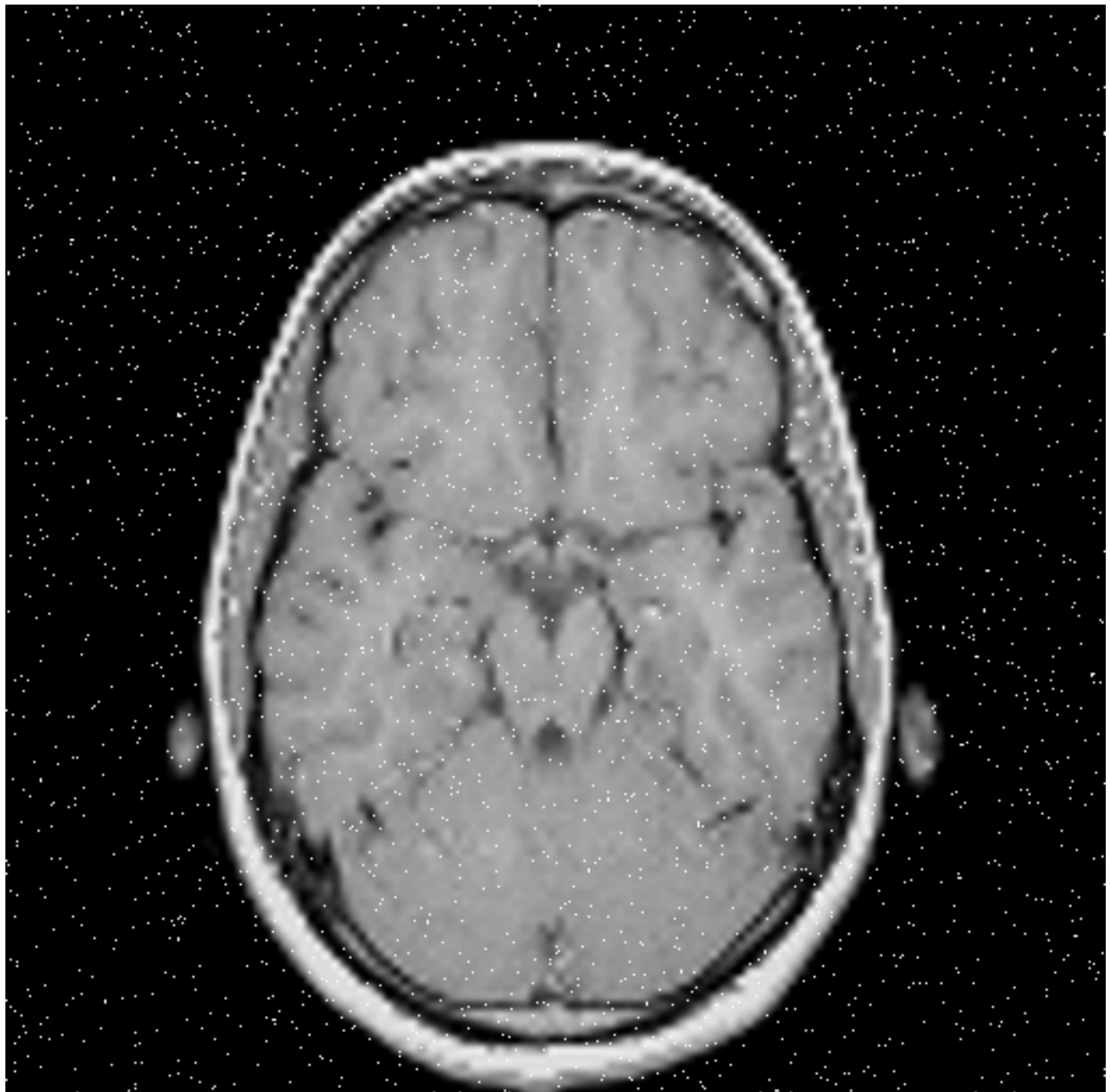
Median Filter: Example

23	33	20
10	20	20
10	20	20

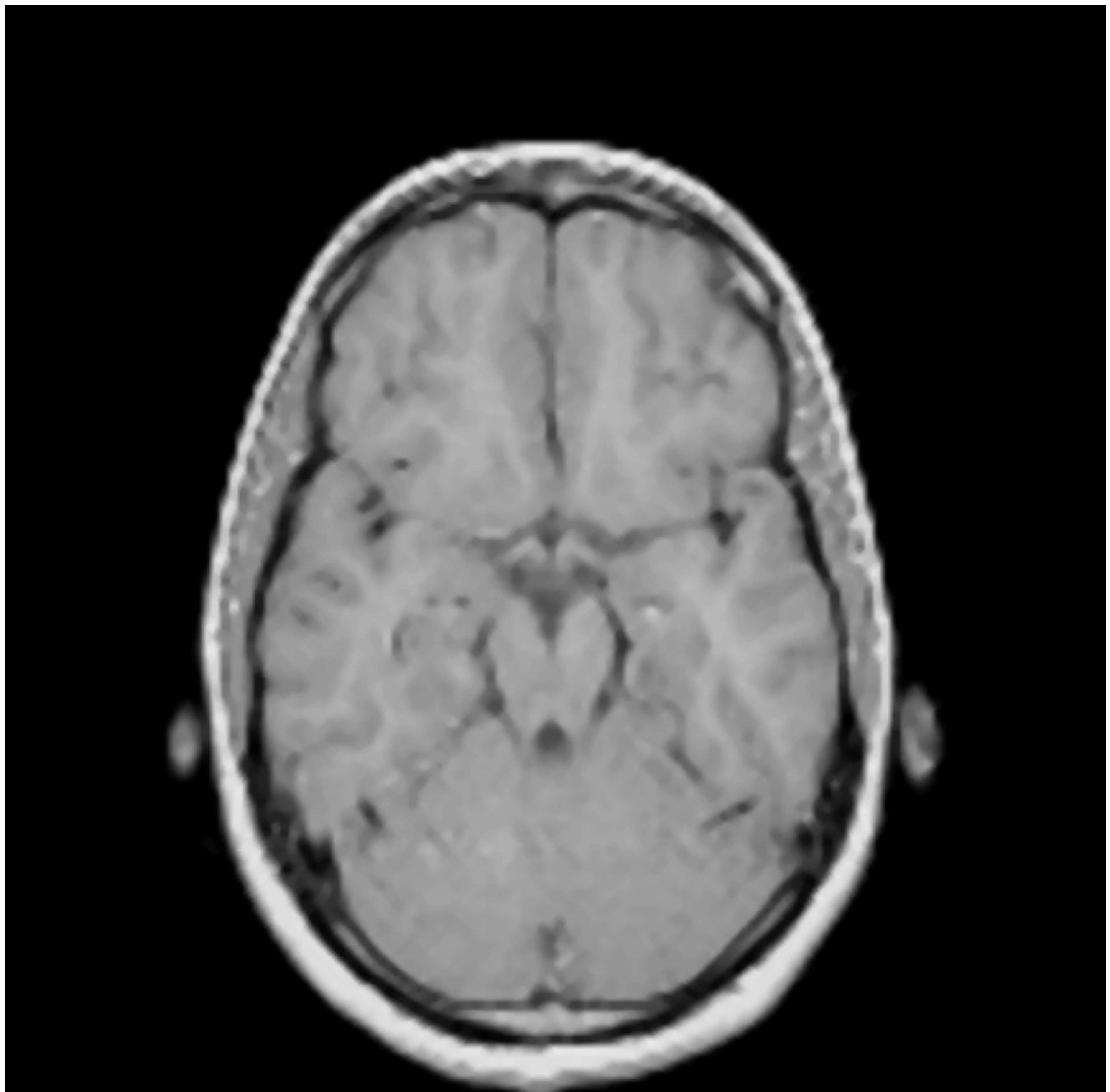
[10, 10, 20, 20, 20, 20, 23, 33, 100]

Median Value!

Median Filter Example



Median Filter Example



Min/Max Filter

- How does it work?
  - We define the size of the filter; e.g., 9×9
  - For each pixel (i, j):
    - We collect all pixel values around (i, j)
    - We check for the min/max pixel values
      - This is faster than sorting values!
    - We take the median value

Min/Max Filter: Example

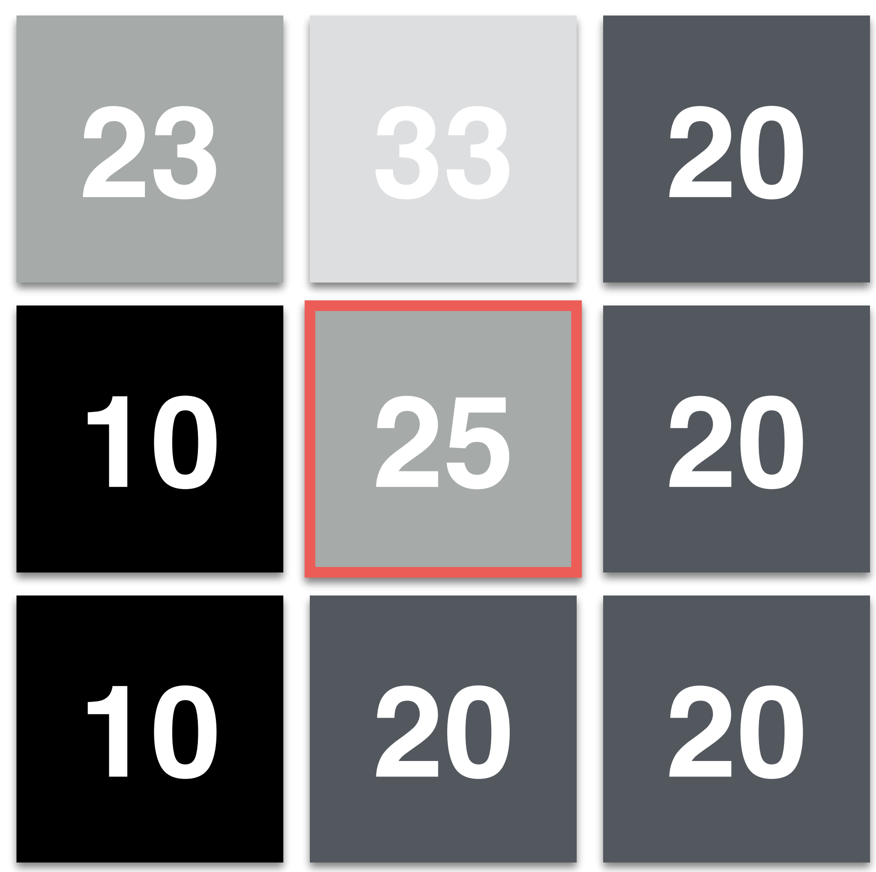
23	33	20
10	25	20
10	20	20

Let's consider the neighborhood of the central pixel; which has value 25.

Min/Max Filter: Example

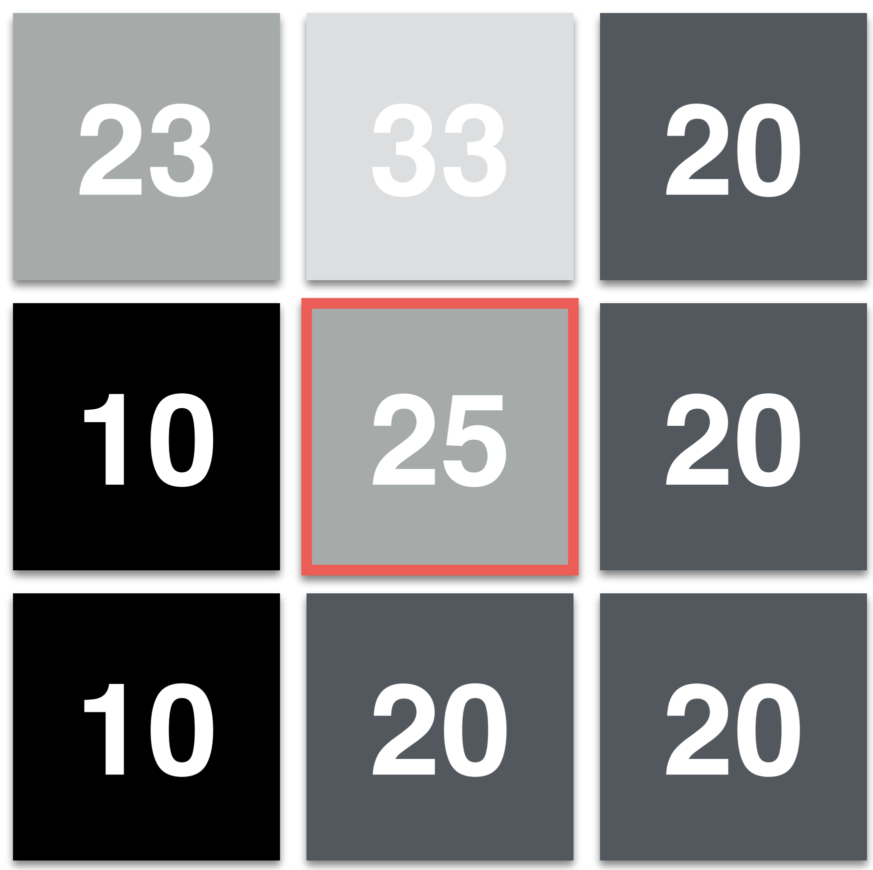
23	33	20
10	25	20
10	20	20

Min/Max Filter: Example



[23, 33, 20, 10, 25, 20, 10, 20, 20]

Min/Max Filter: Min Example



[23, 33, 20, 10, 25, 20, 10, 20, 20]

Min/Max Filter: Example

23	33	20
10	10	20
10	20	20

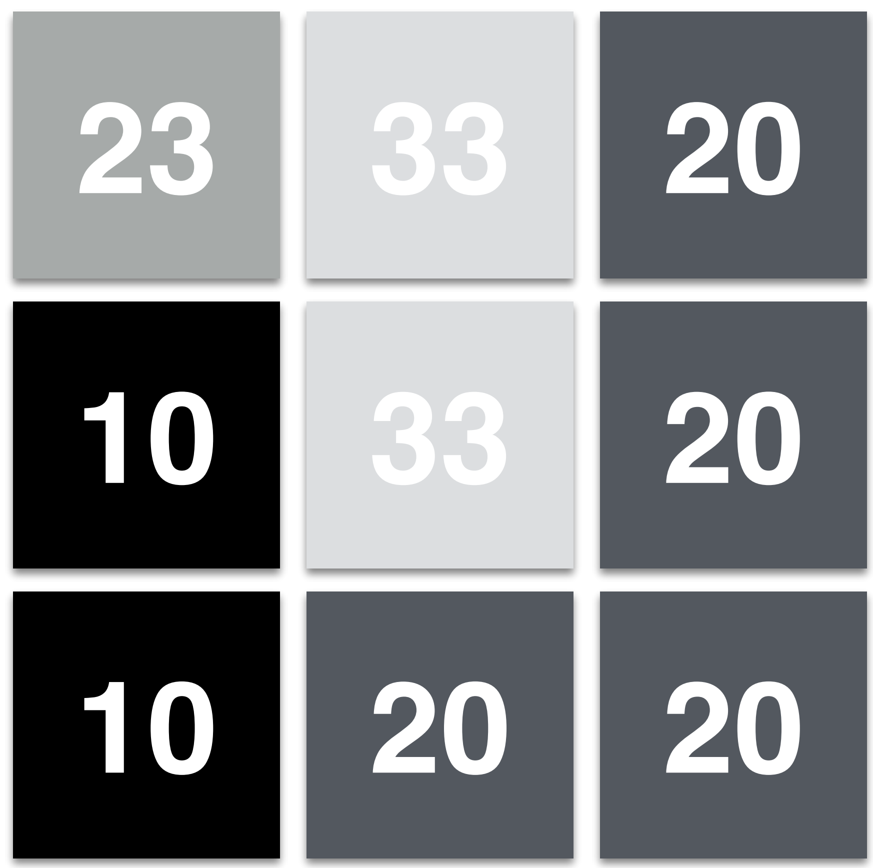
[23, 33, 20, 10, 25, 20, 10, 20, 20]

Min/Max Filter: Max Example

23	33	20
10	25	20
10	20	20

[23, 33, 20, 10, 25, 20, 10, 20, 20]

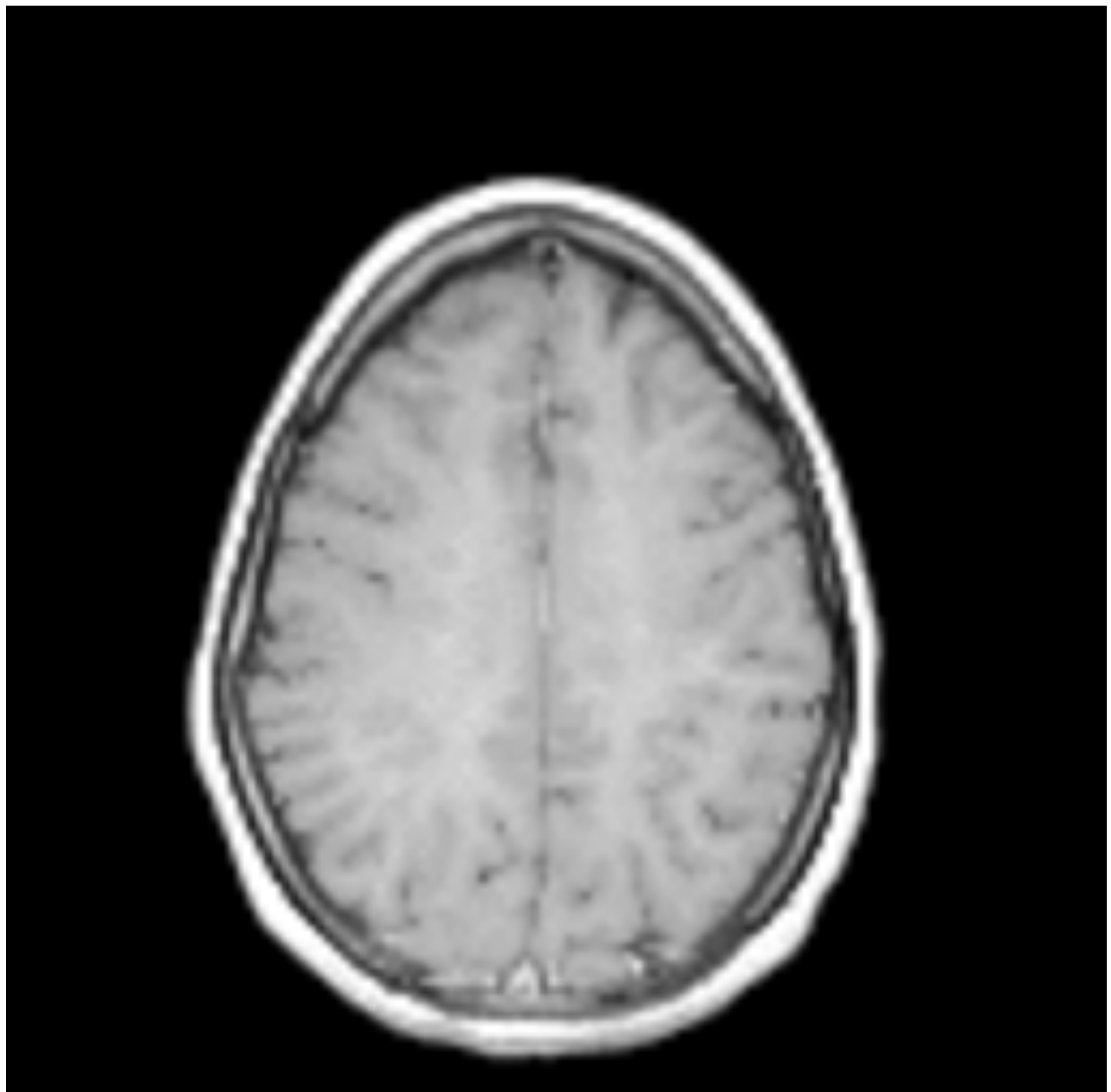
Min/Max Filter: Max Example



[23, 33, 20, 10, 25, 20, 10, 20, 20]

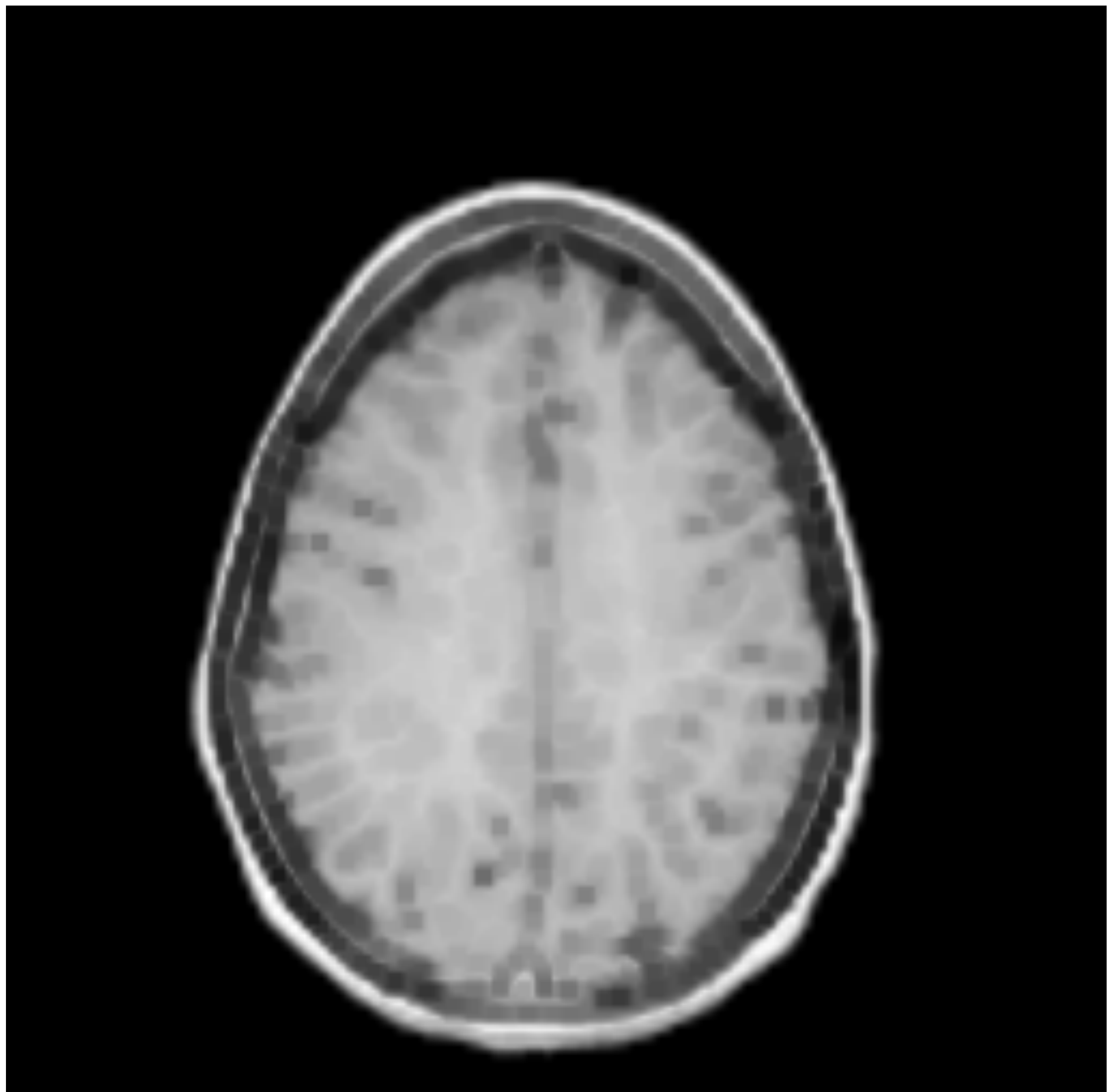
Median Filter: Min+Max Example

Input



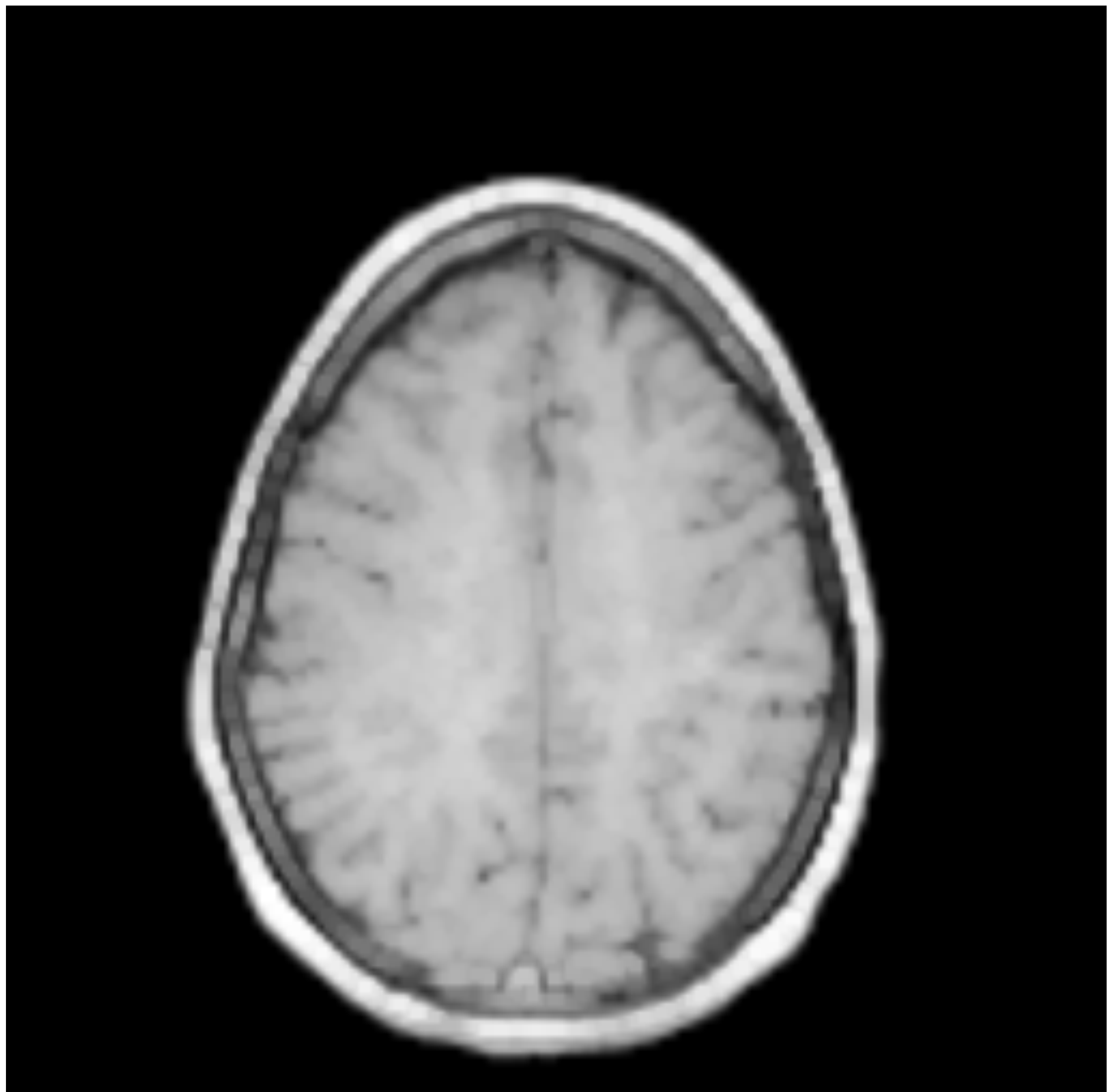
Median Filter: Min+Max Example

Min Filter



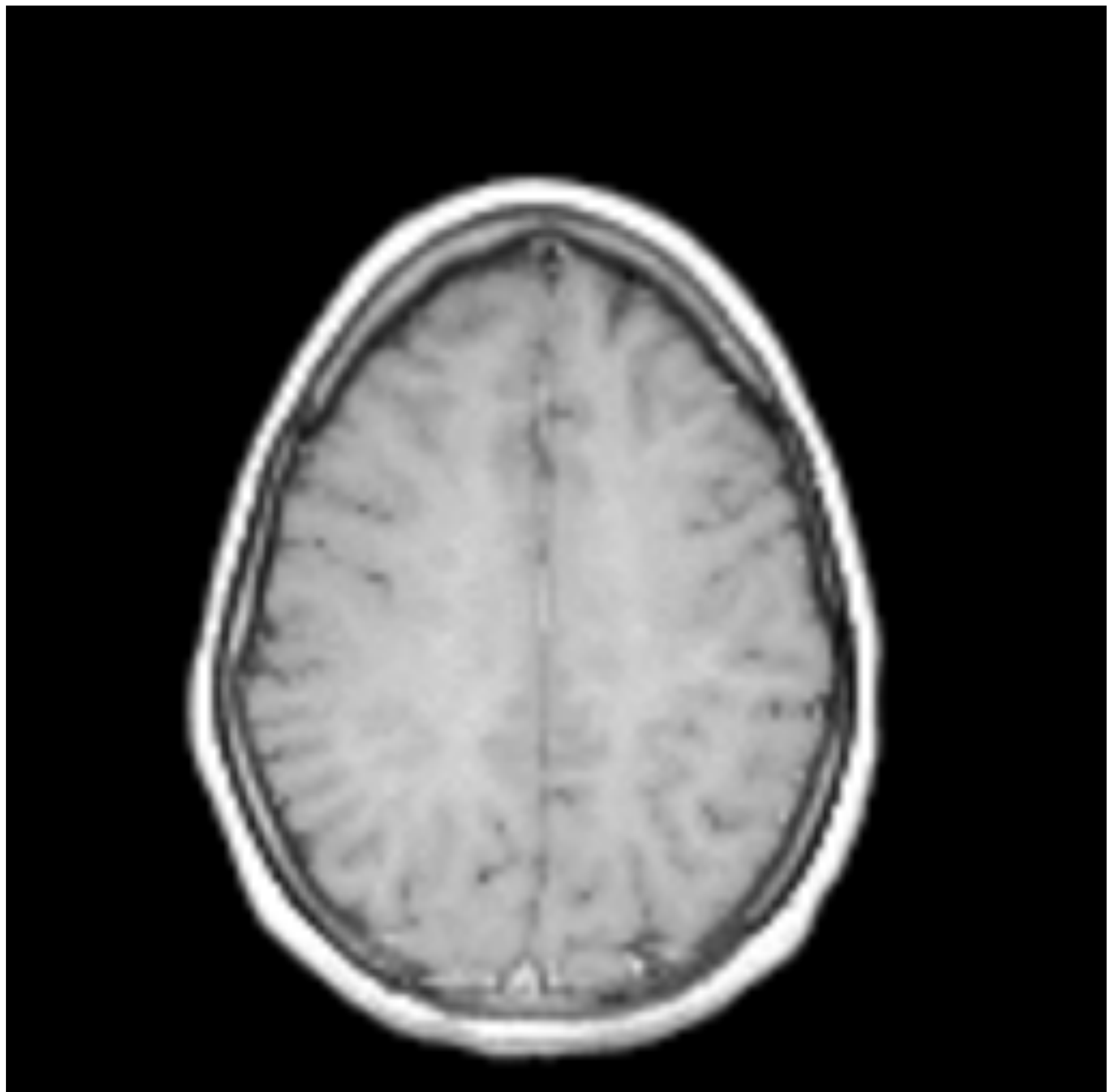
Median Filter: Min+Max Example

Max Filter



Median Filter: Min+Max Example

Input



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

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

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

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

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

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

The Bilateral Filter

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

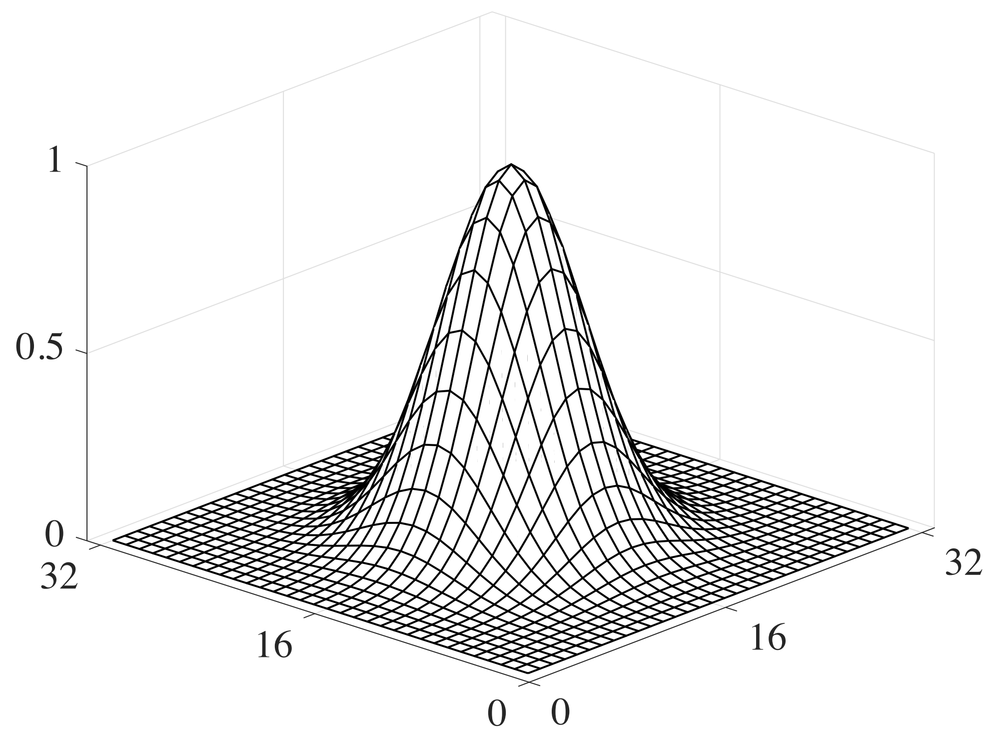
$$K[I](\mathbf{x}, f_s, g_r) = \sum_{\mathbf{y}\in\Omega(\mathbf{x})} f_s(\|\mathbf{x}-\mathbf{y}\|) g_r(\|I(\mathbf{y}) - I(\mathbf{x})\|),$$

The Bilateral Filter

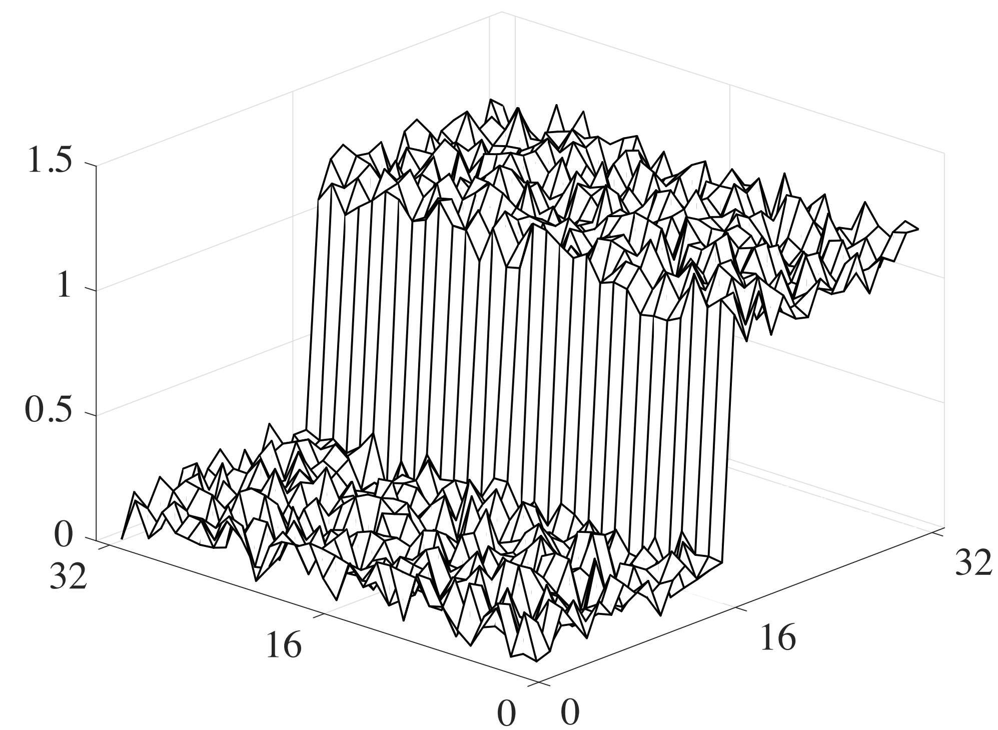
- f_s (Spatial function): a Gaussian function
- g_r (Range function): a Gaussian function
- How large is the kernel?
  - If the spatial function is a Gaussian:

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

The Bilateral Filter Explained

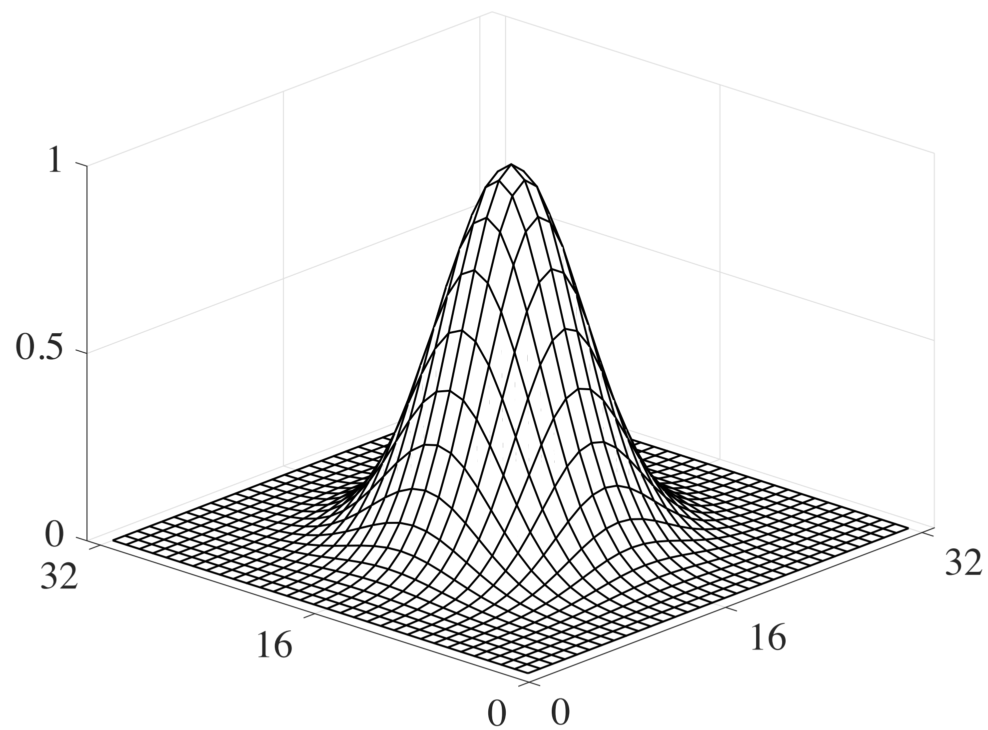


The Kernel

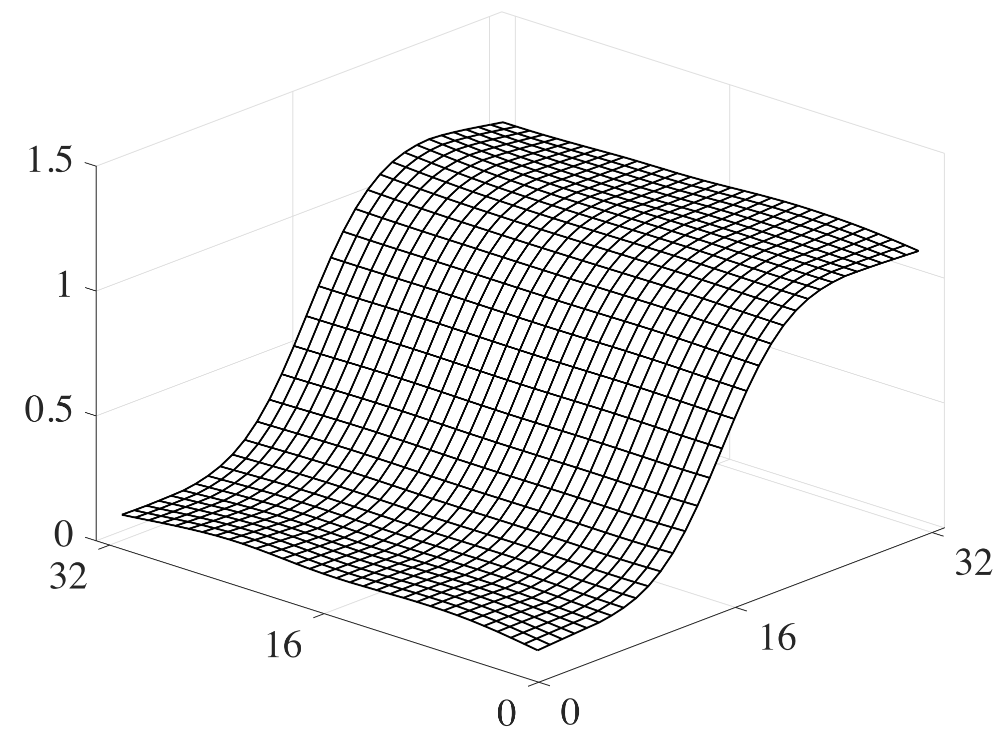


Image

The Bilateral Filter Explained

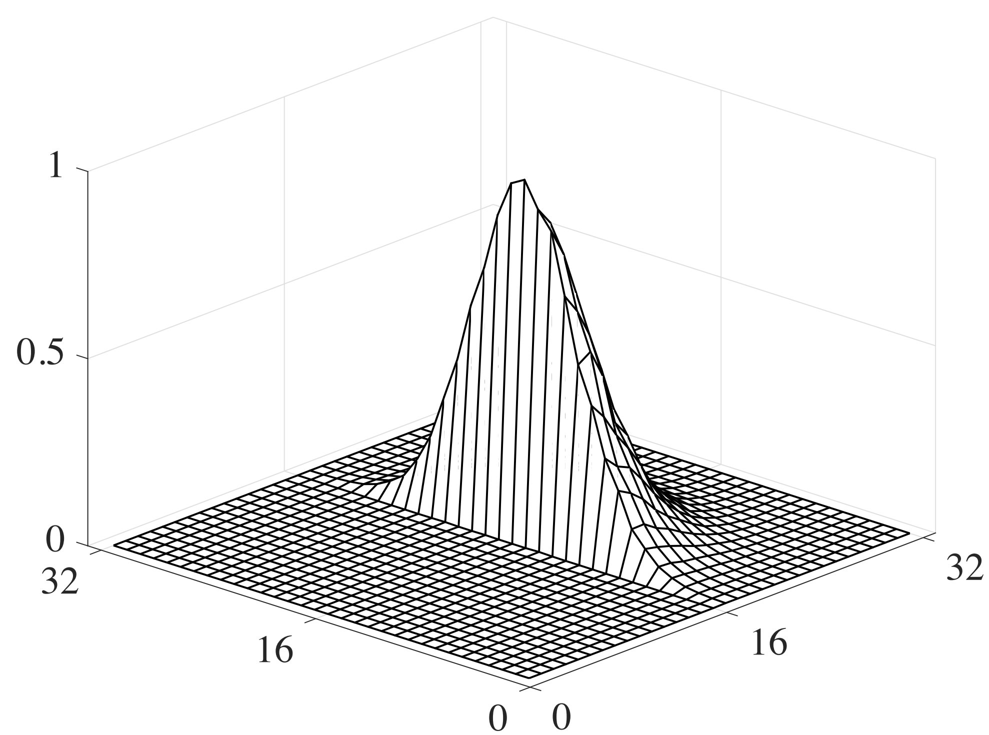


The Kernel

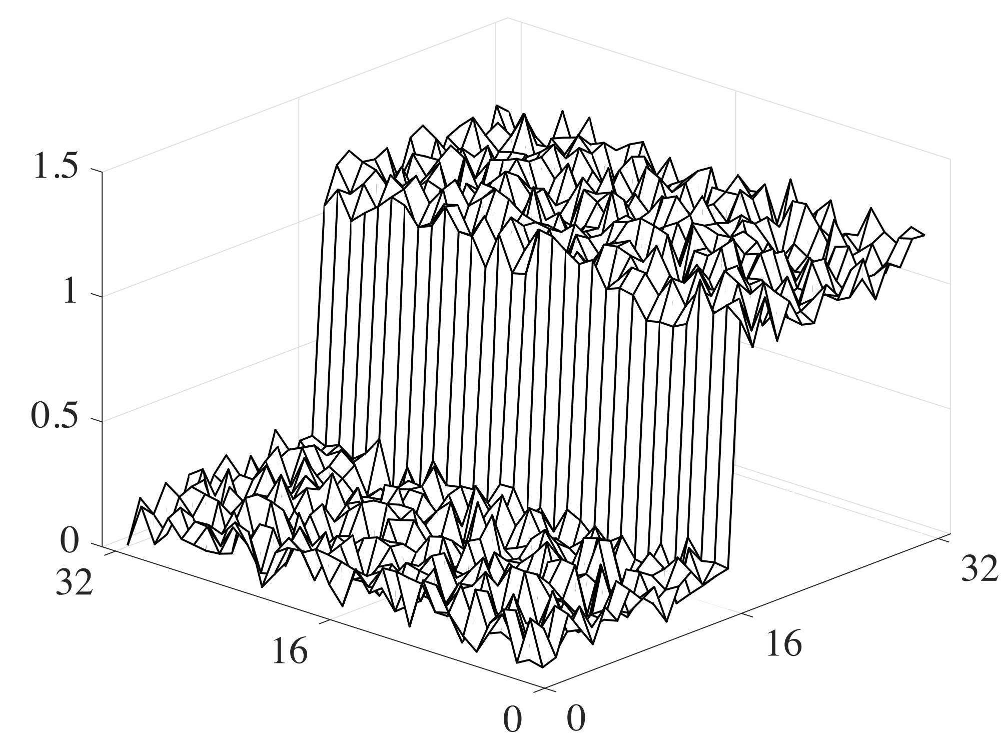


Image

The Bilateral Filter Explained

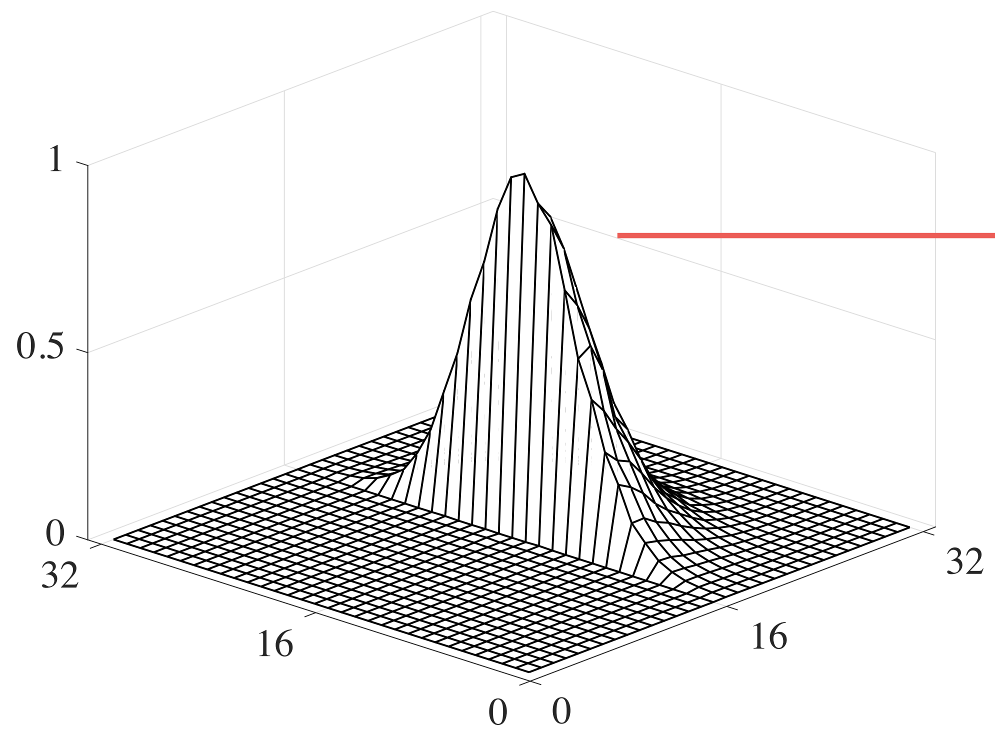


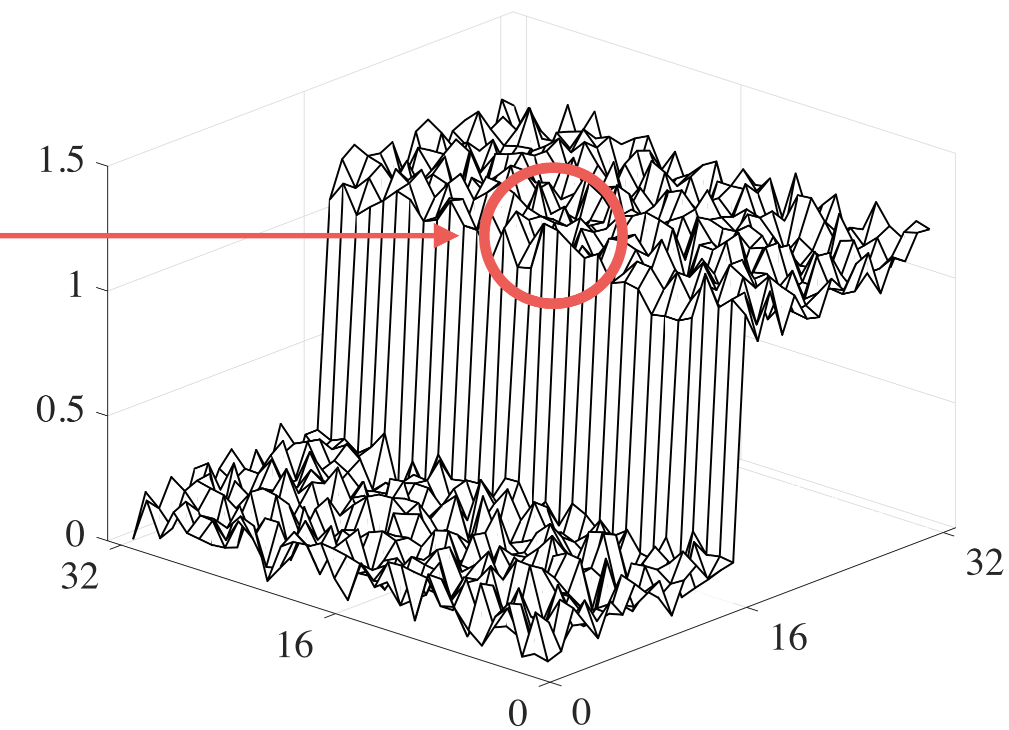
The Kernel
(change for each pixel!!)



Image

The Bilateral Filter Explained

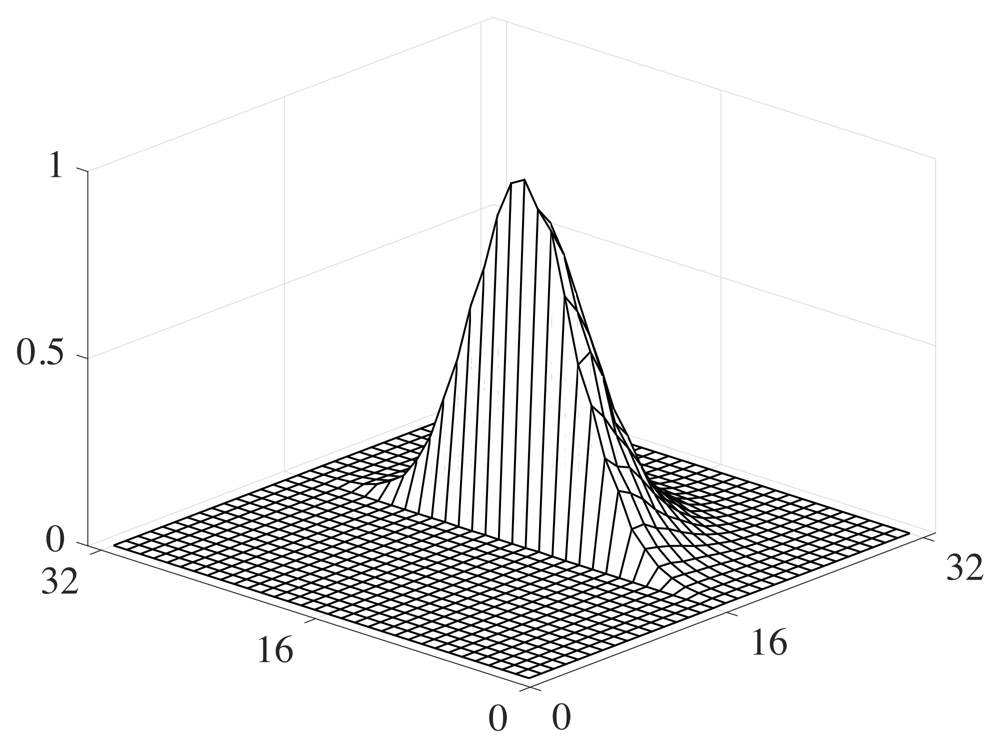




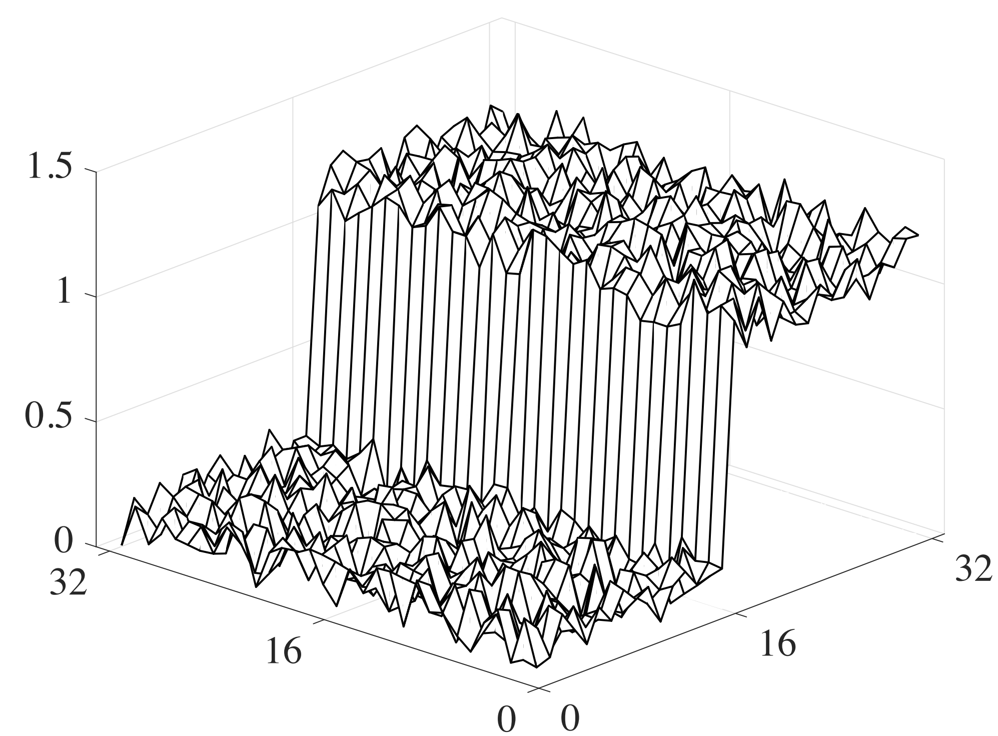
The Kernel
(change for each pixel!!)

Image

The Bilateral Filter Explained

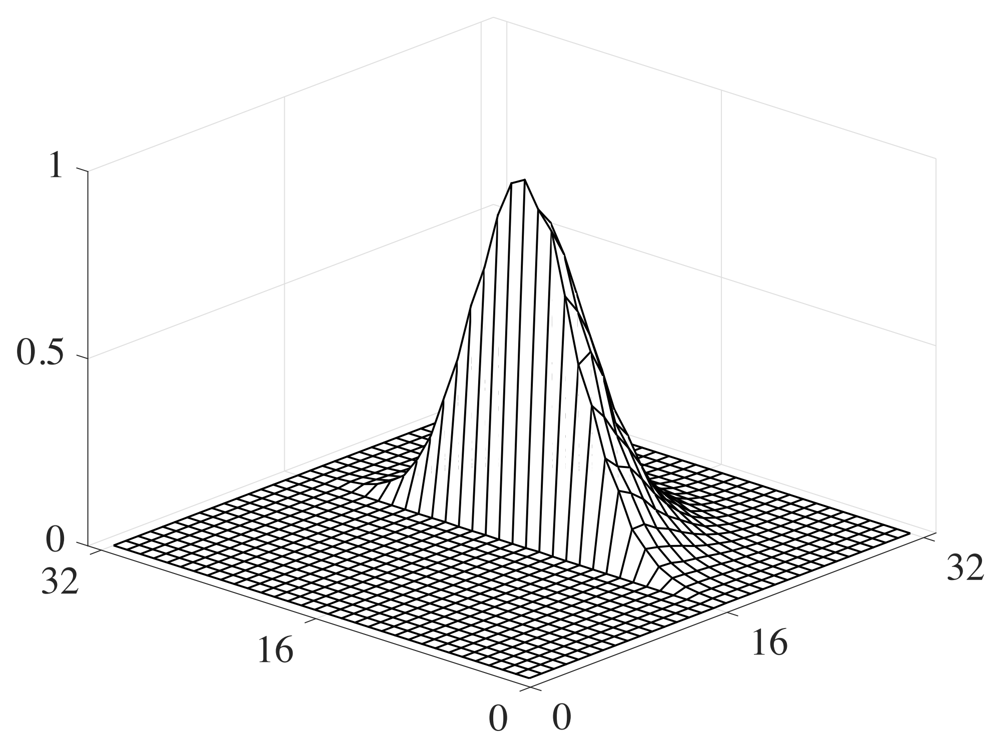


The Kernel
(change for each pixel!!)

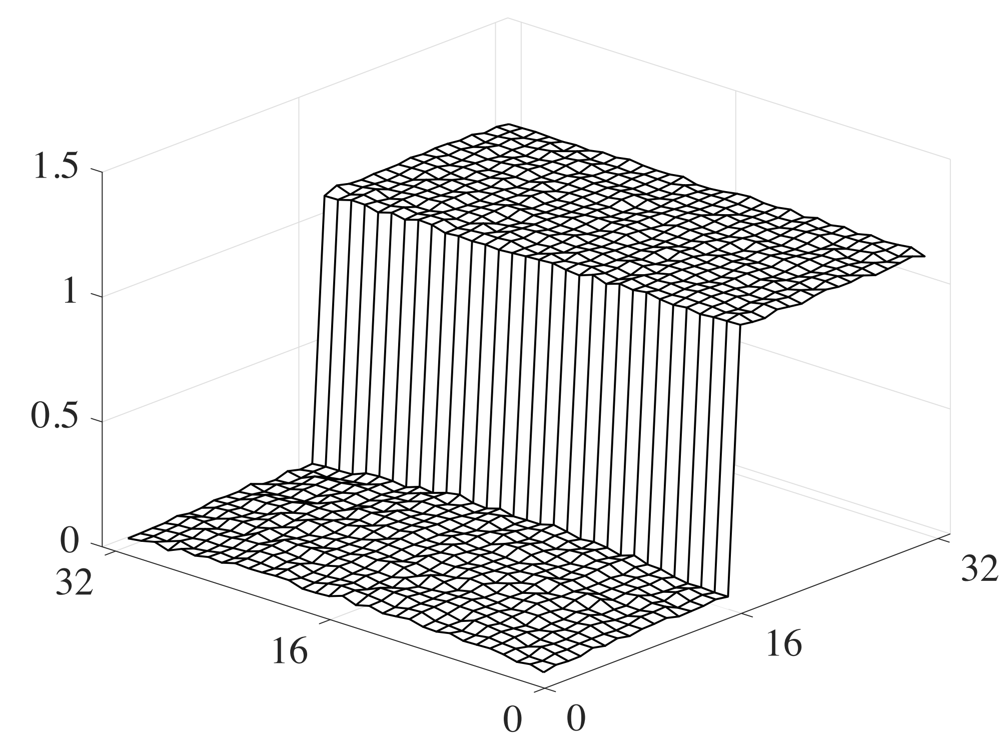


Image

The Bilateral Filter Explained

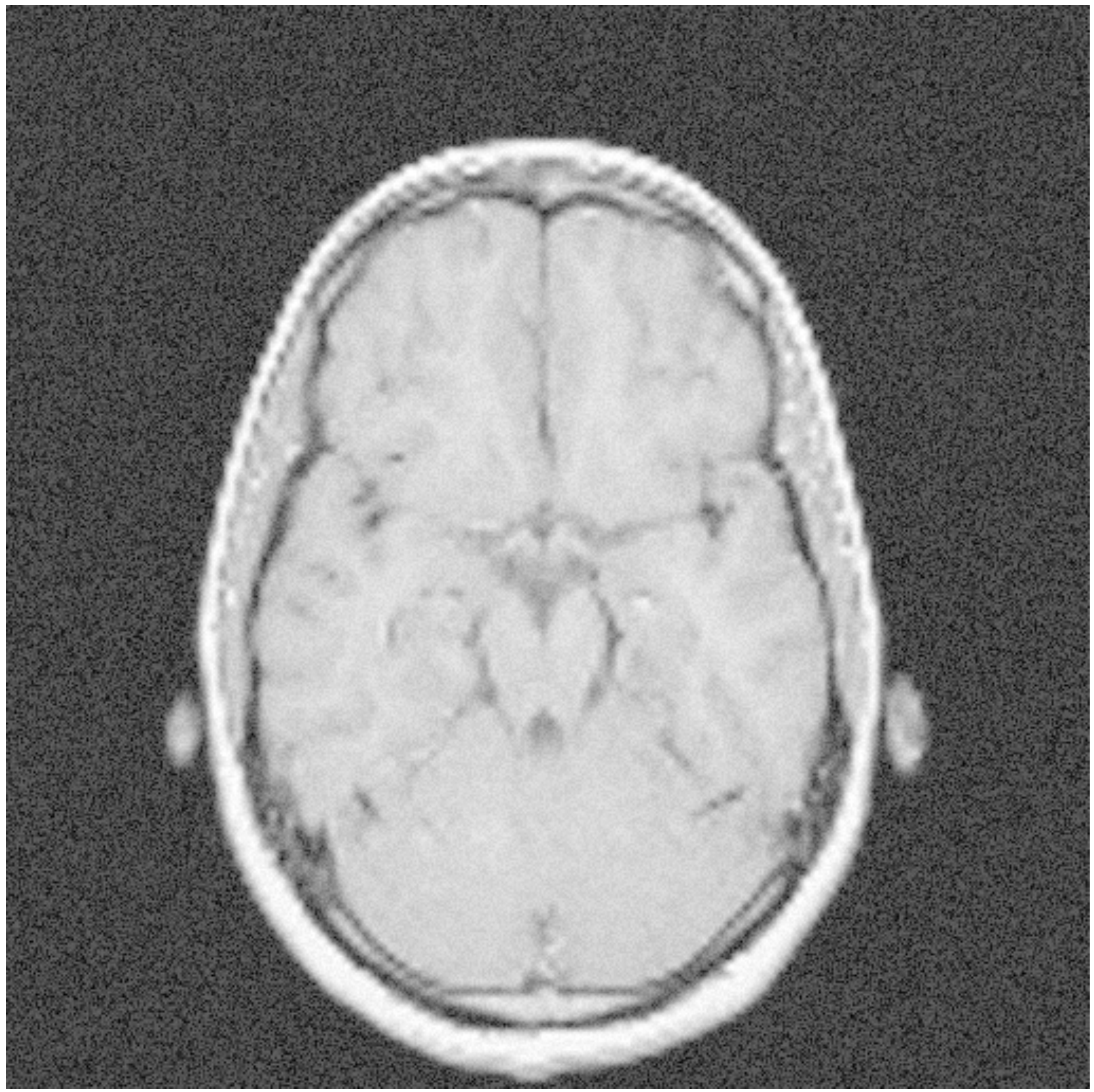


The Kernel
(change for each pixel!!)

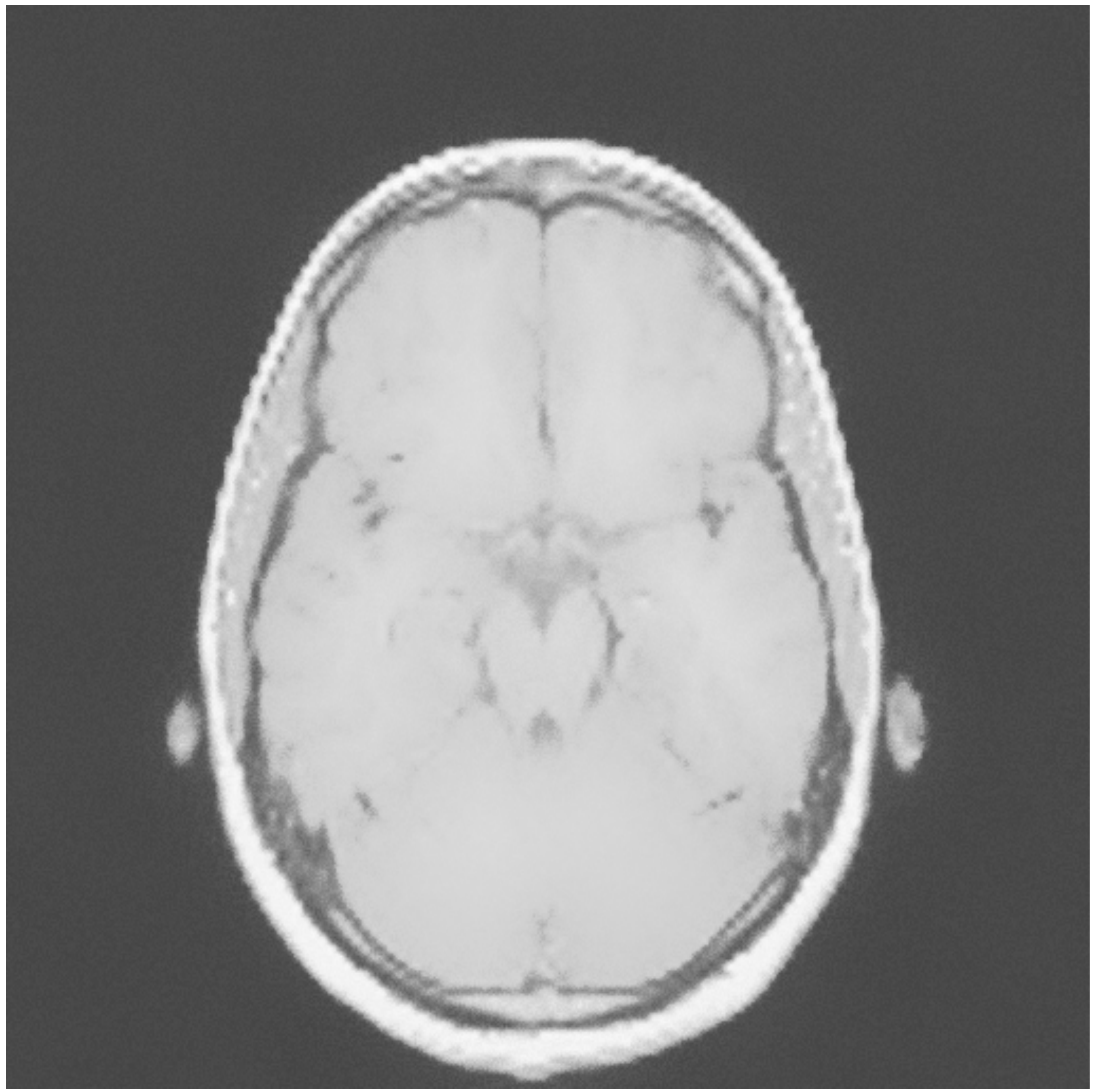


Image

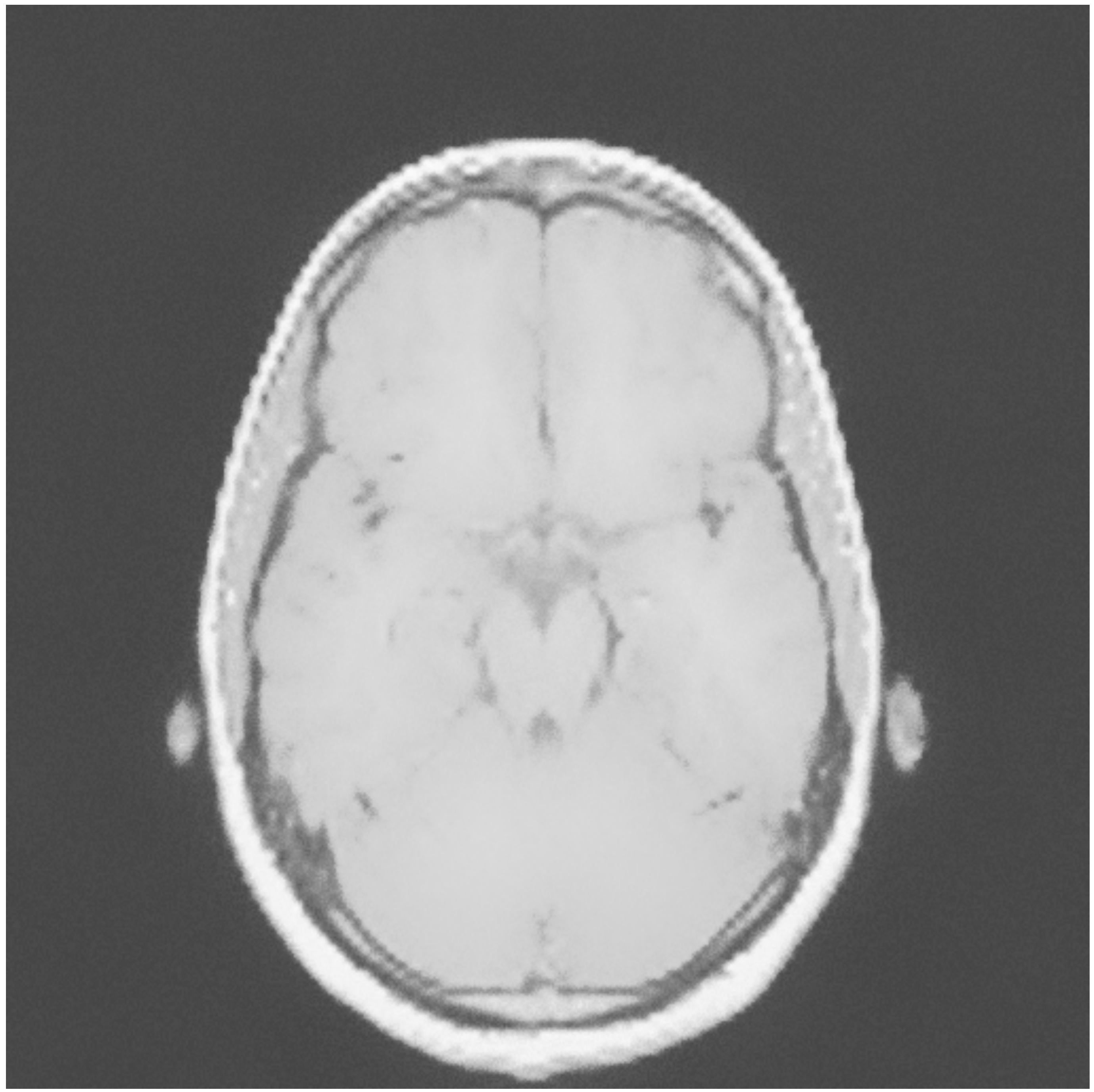
The Bilateral Filter Example



The Bilateral Filter Example



The Bilateral Filter Example



Result of using a Gaussian filter with
the same kernel size of previous slide

The Bilateral Filter Example

Result of using a Gaussian filter with
the same kernel size of previous slide

Local Contrast Enhancement

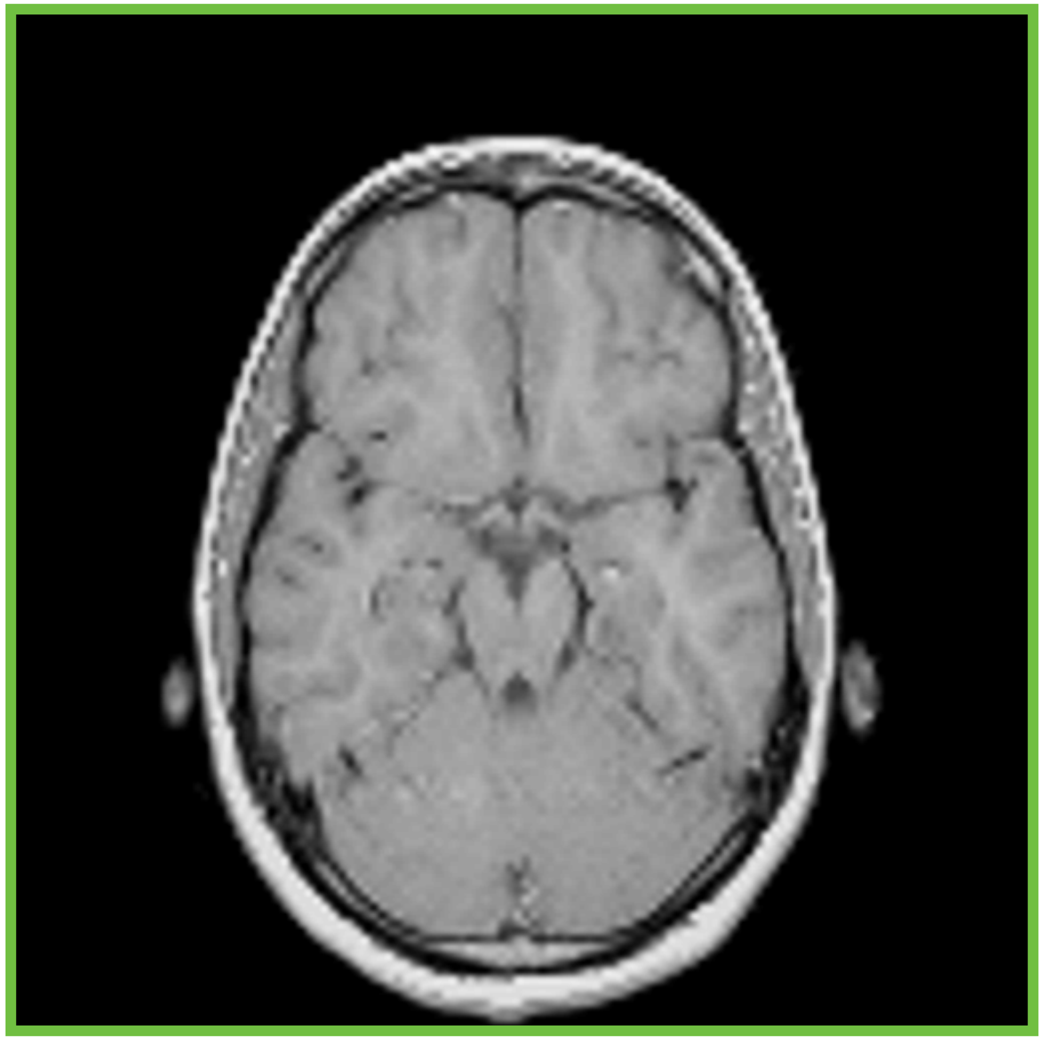
- Before, we have seen how to increase local contrast using the sharpening operator (or modified Laplacian).
- We can achieve better results using a more general framework

Local Contrast Enhancement

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

Local Contrast Enhancement

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

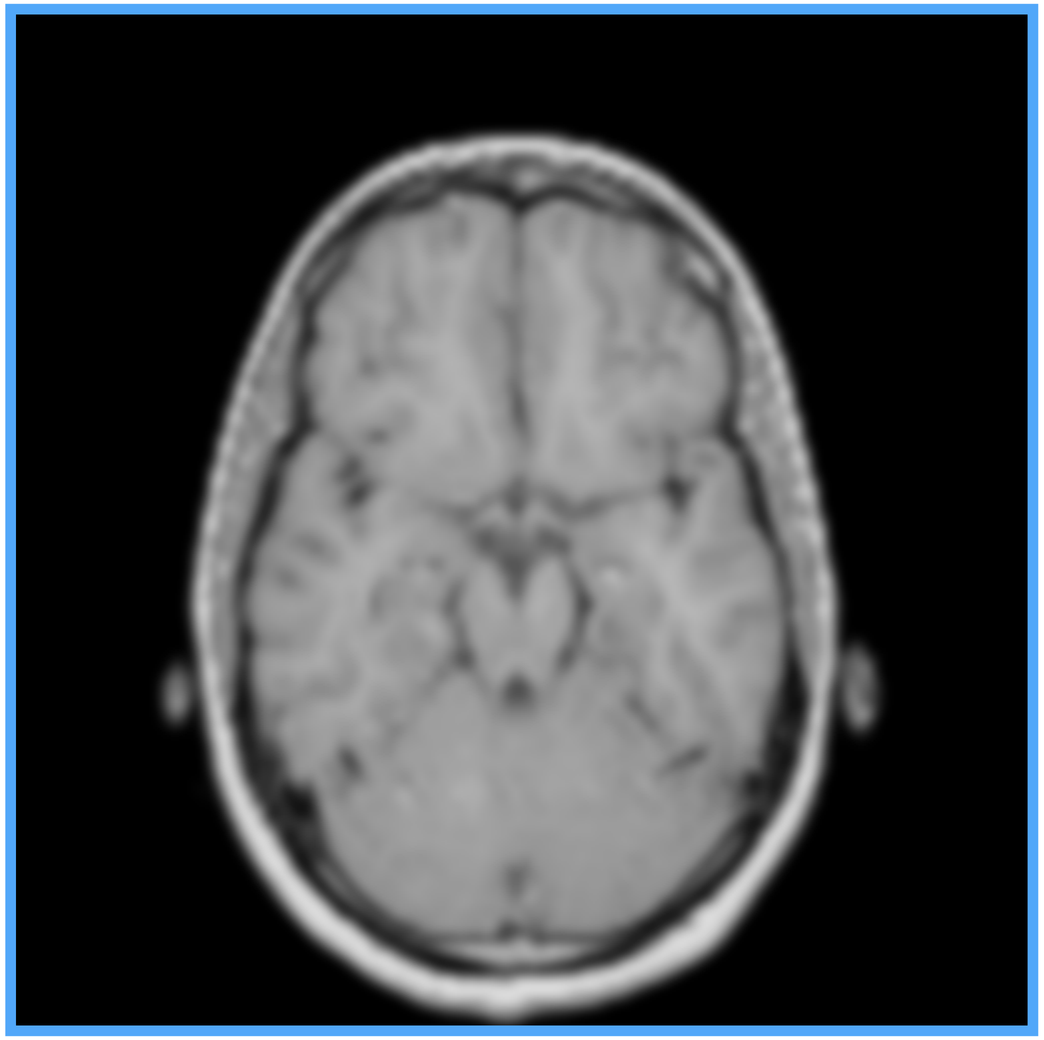


Local Contrast Enhancement

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

Local Contrast Enhancement

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

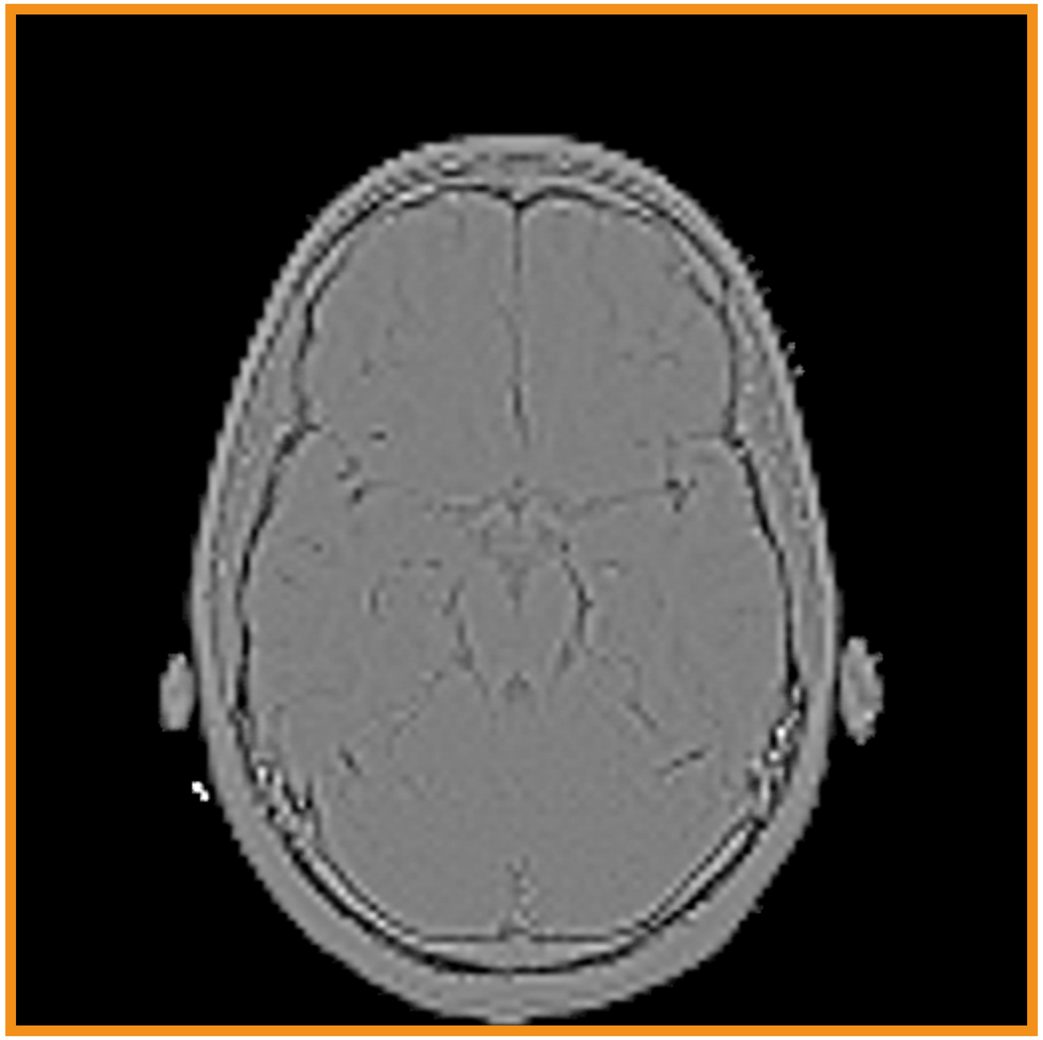


Local Contrast Enhancement

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

Local Contrast Enhancement

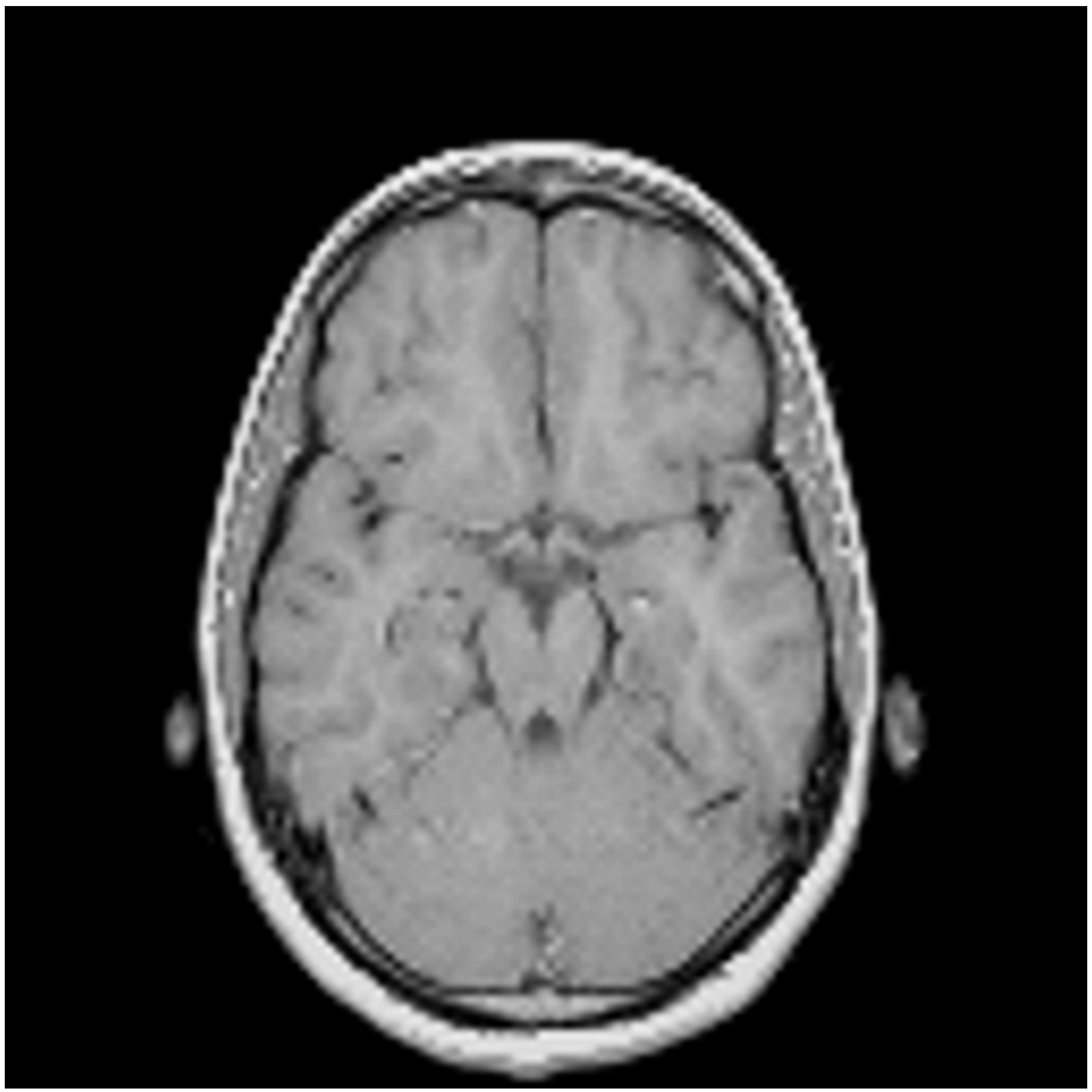
$$O[i,j] = f[i,j] \cdot \left(\frac{f[i,j]}{(f \otimes g)[i,j])} \right)$$



Local Contrast Enhancement

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

Local Contrast Enhancement Example



Input Image

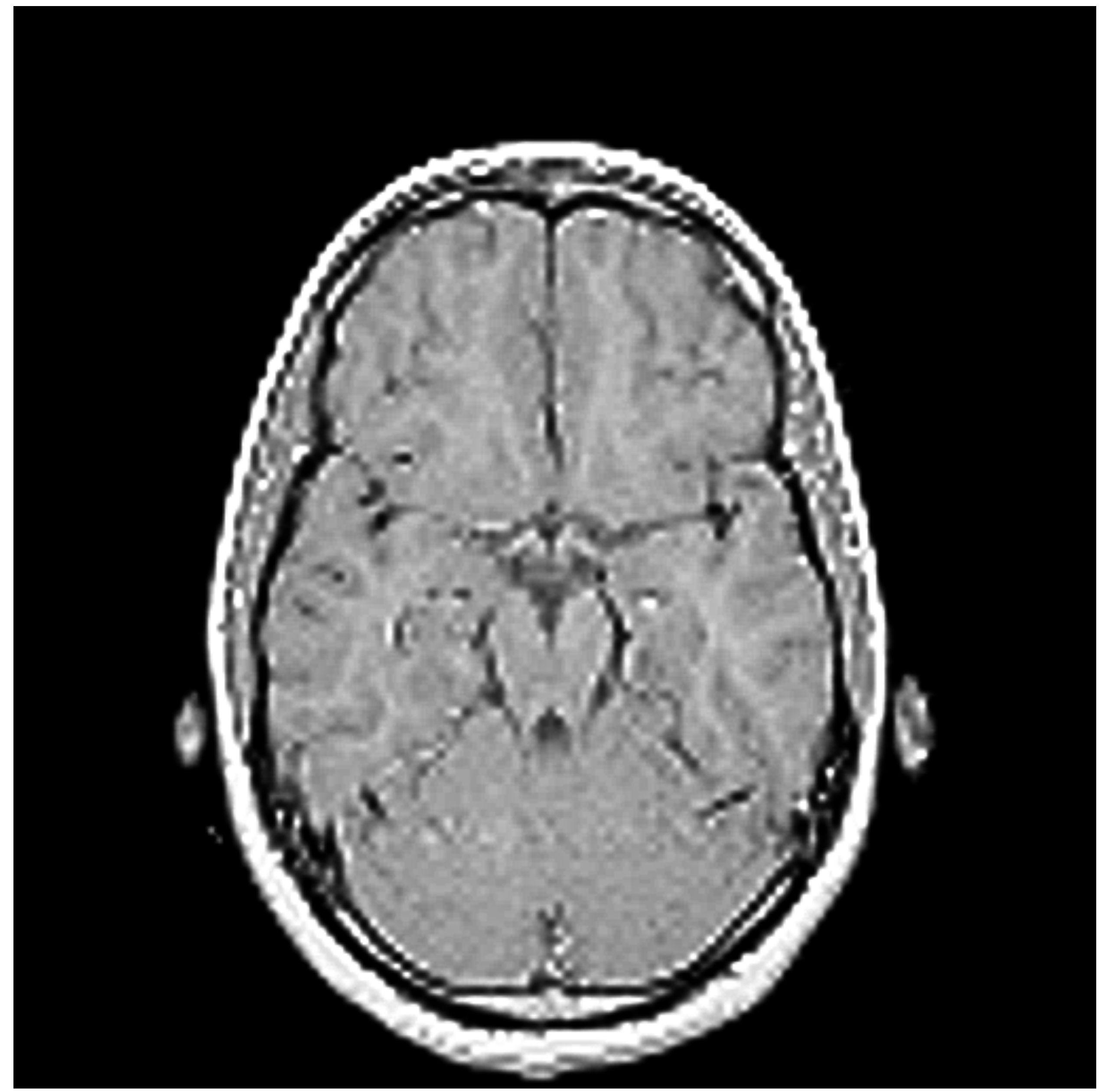


Image After Enhancement

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.

Deconvolution

- In some cases, we have to reduce “blur” (e.g., the patient moved during the scan):

$$I \otimes K = J$$

- where I is ideal image, K is a convolution kernel, and J is the input blurred image.

- **NOTE**: the real case is:

$$(I \otimes K) + n = J$$

- where n is noise

Deconvolution

- Assuming that we know K:
  - We need to find an I such that:

$$I \otimes K = J$$

  - which means:

$$\arg\min_{I} \Big(I \otimes K - J \Big)^2$$

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's method

Deconvolution

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

Input Blurred Image

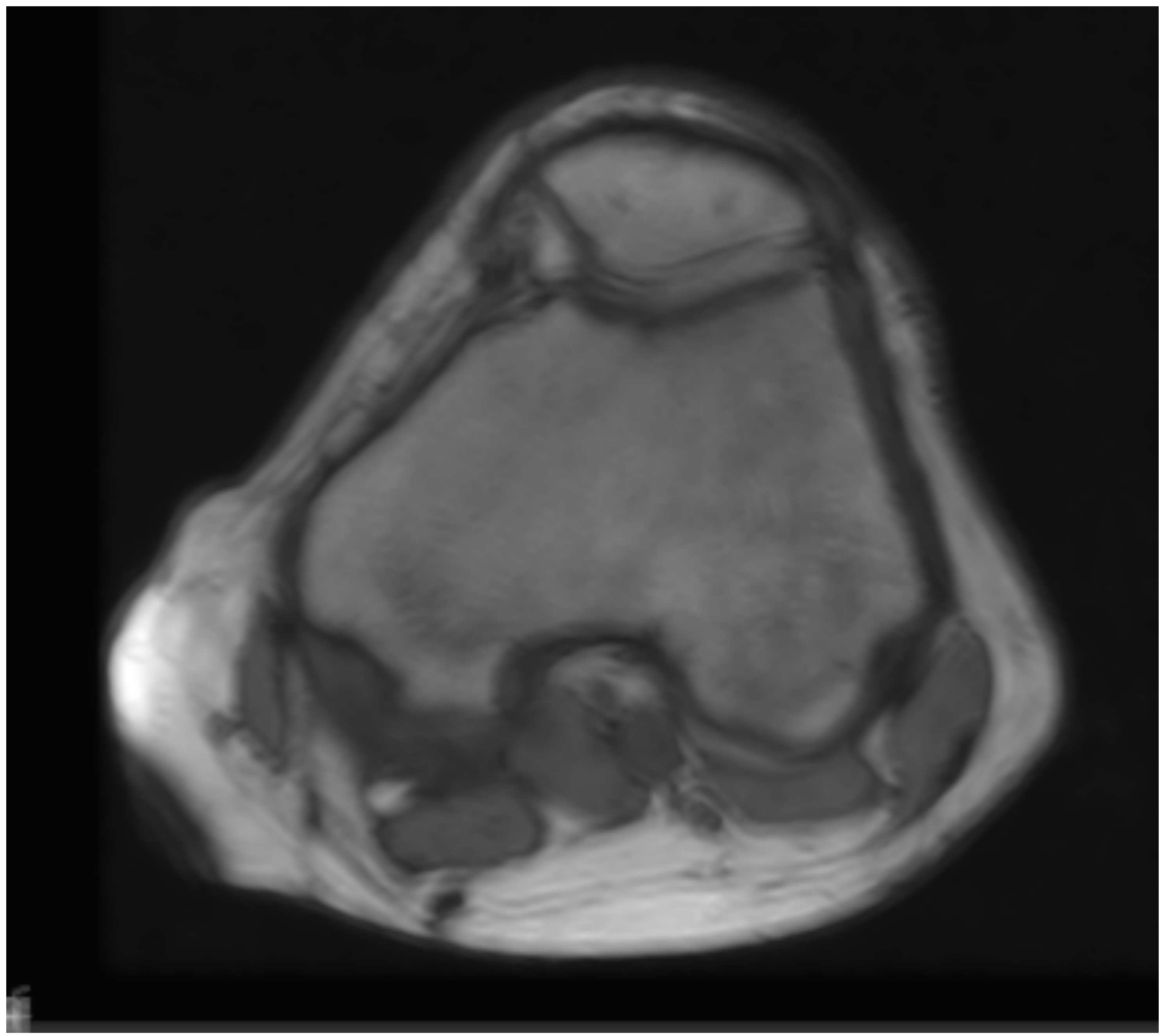
Richardson–Lucy's method

Deconvolution

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

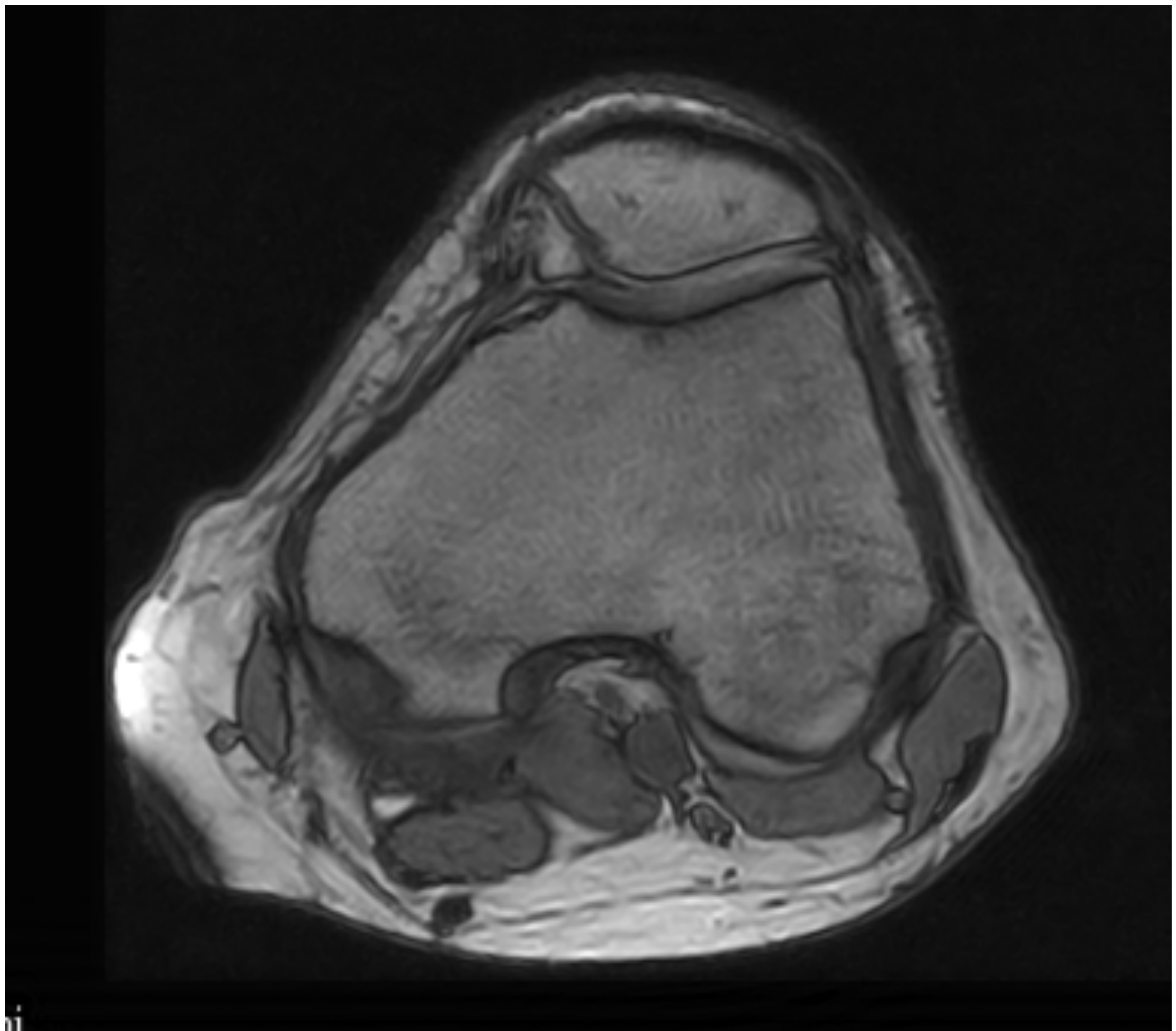
Richardson–Lucy's method

Deconvolution Example



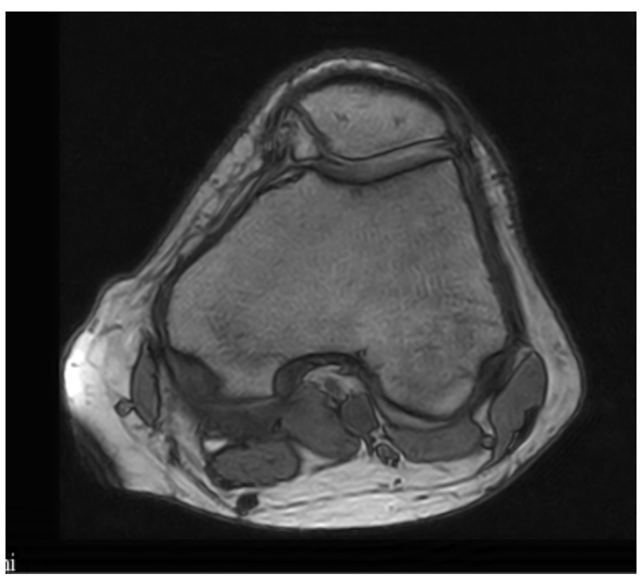
After 1000 iterations

Deconvolution Example

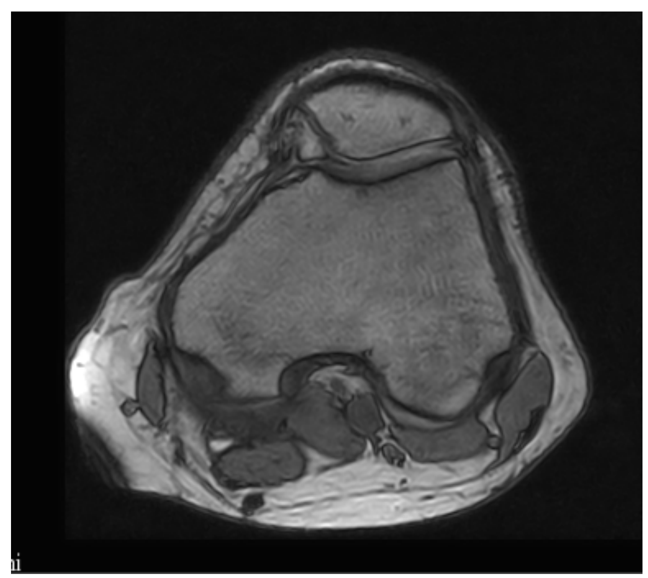


After 1000 iterations

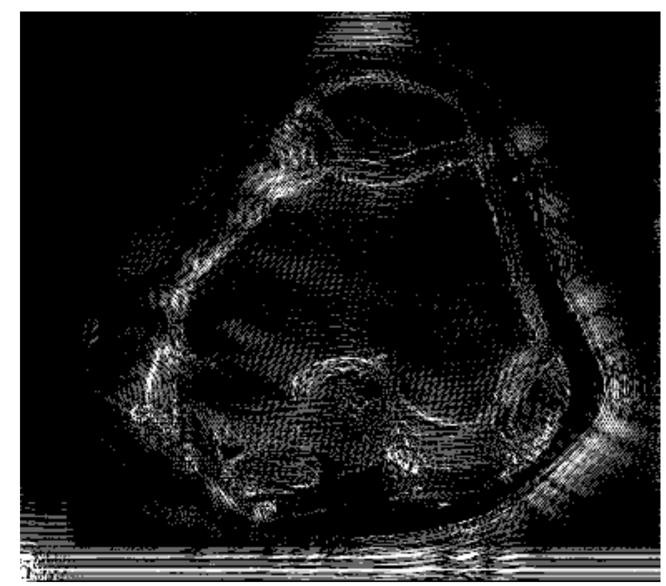
Deconvolution Example



-



=



original

deconvolution

difference

Deconvolution

- To make it work, we need to:
  - Run many iterations (more than thousands)!
  - Known exactly the size and shape of K; we can estimate it. This may create artifacts!

Image Upsampling

Why Upsampling?

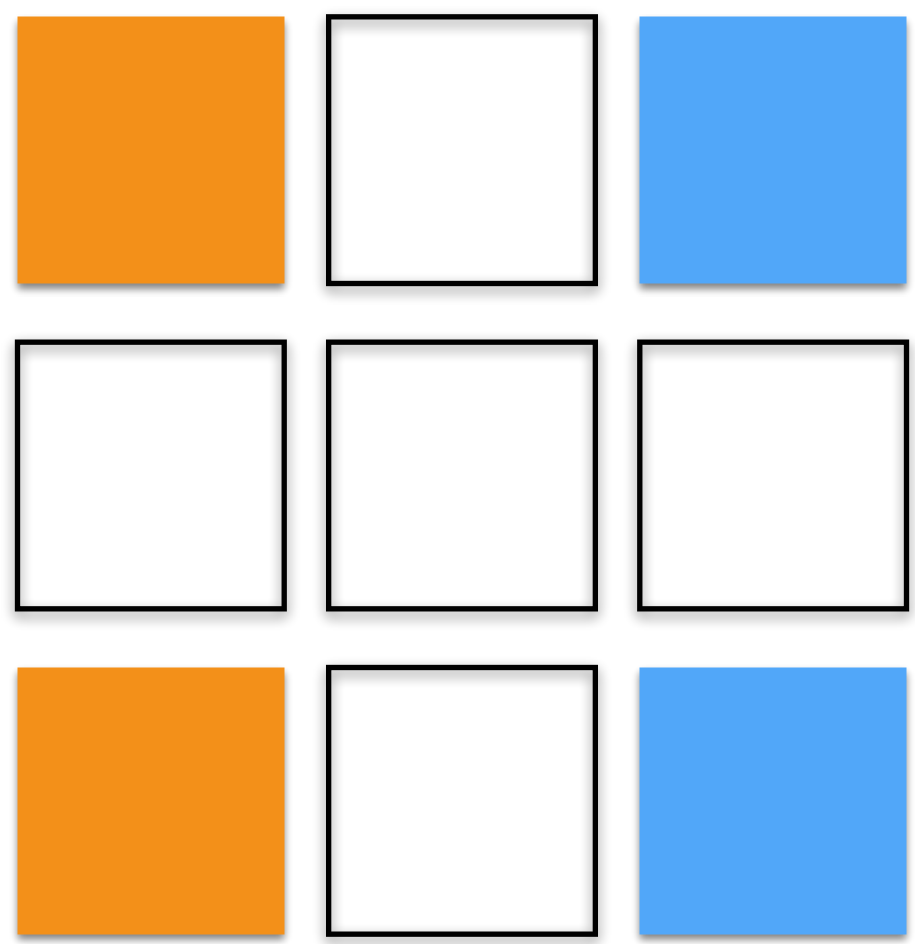
- The main reason why we want to upsample (we invent data basically) our input data is that they have a very low resolution

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

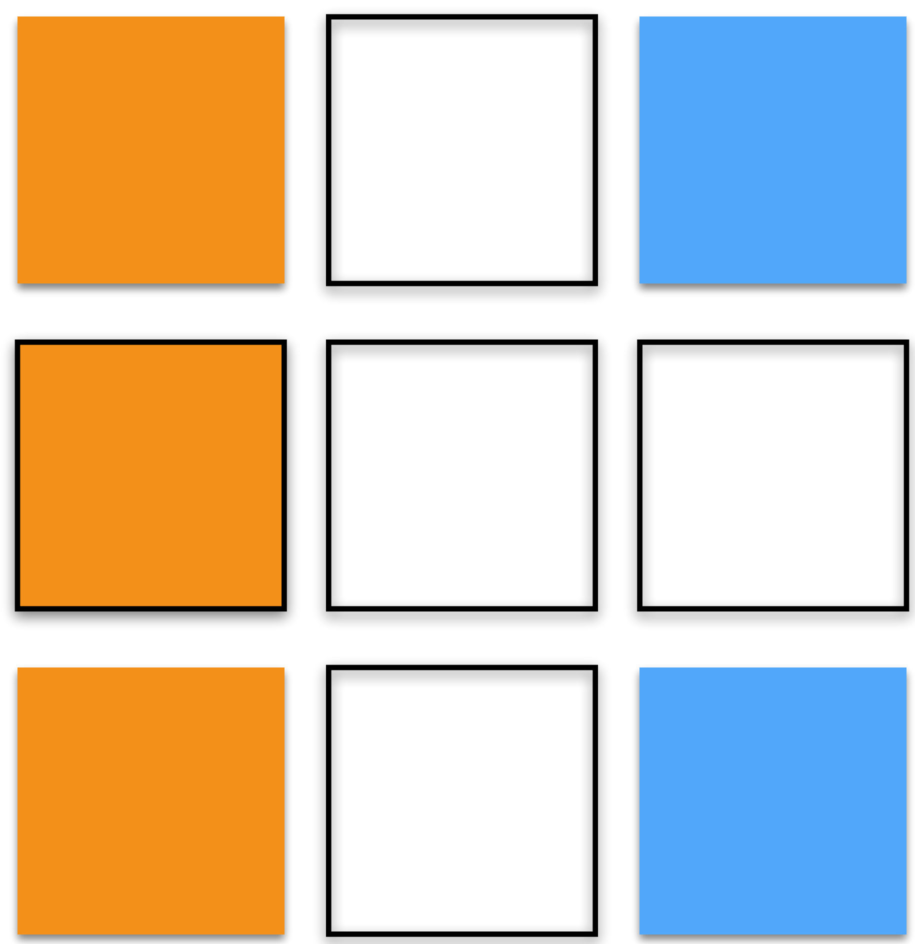
Upsampling

- 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

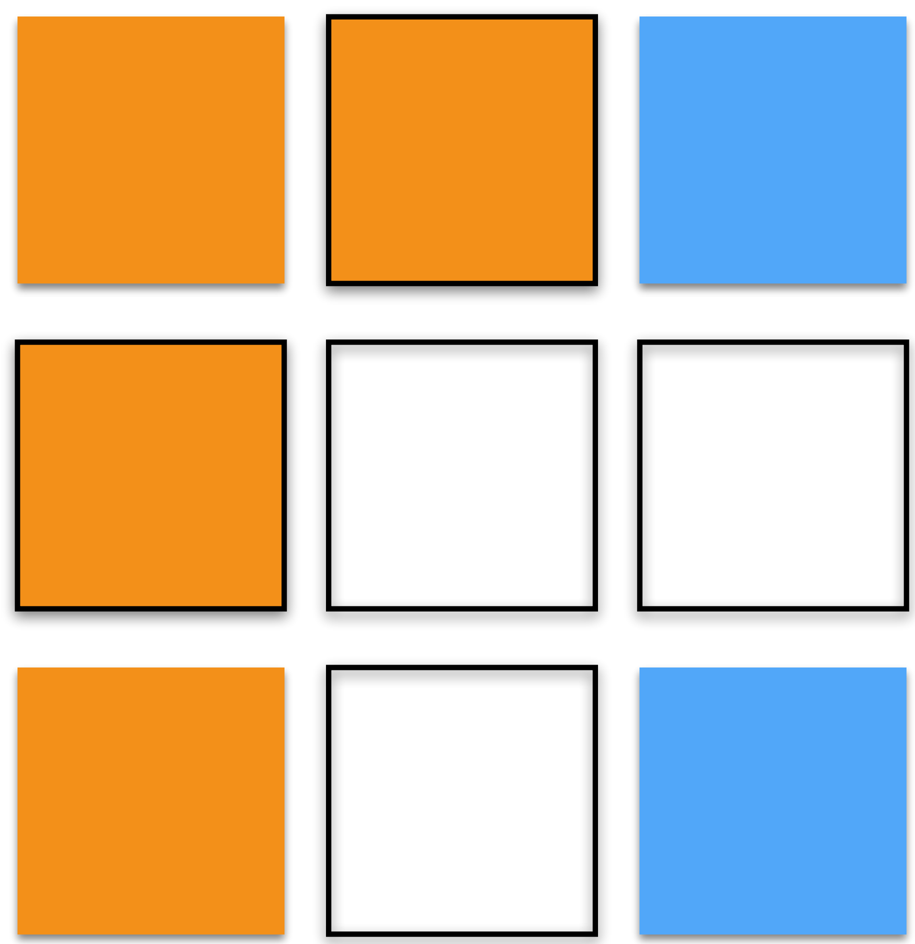
Upsampling:
Nearest Neighbors



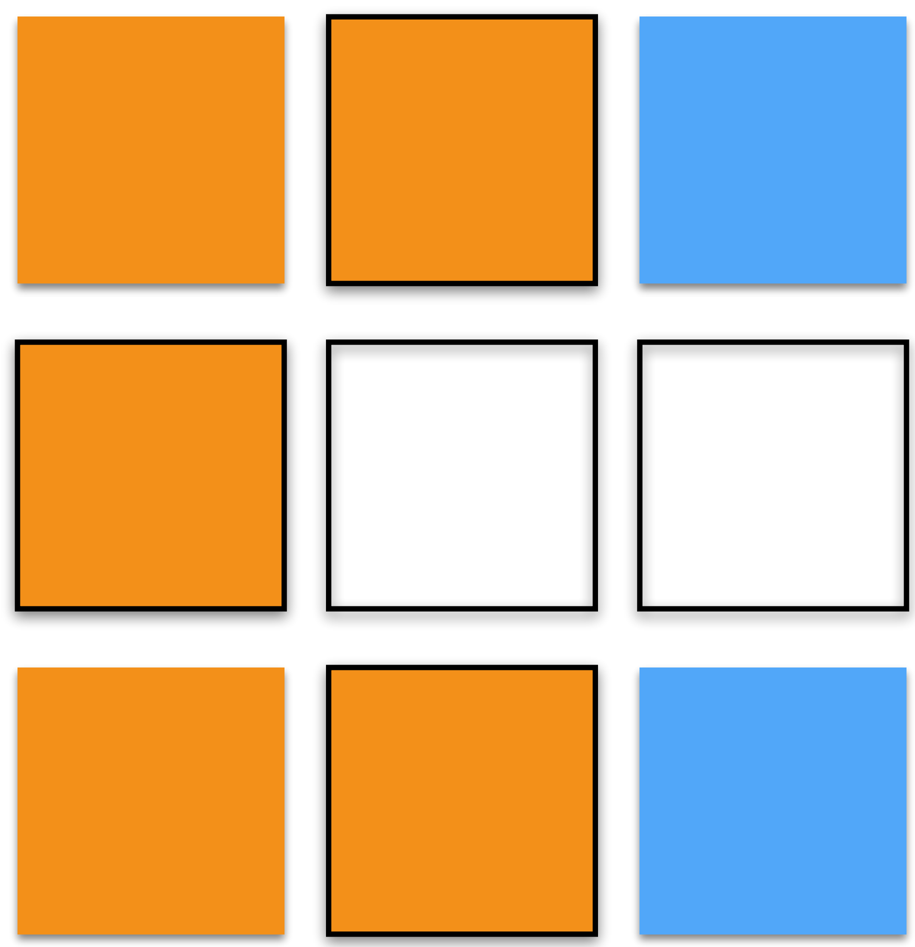
Upsampling: Nearest Neighbors



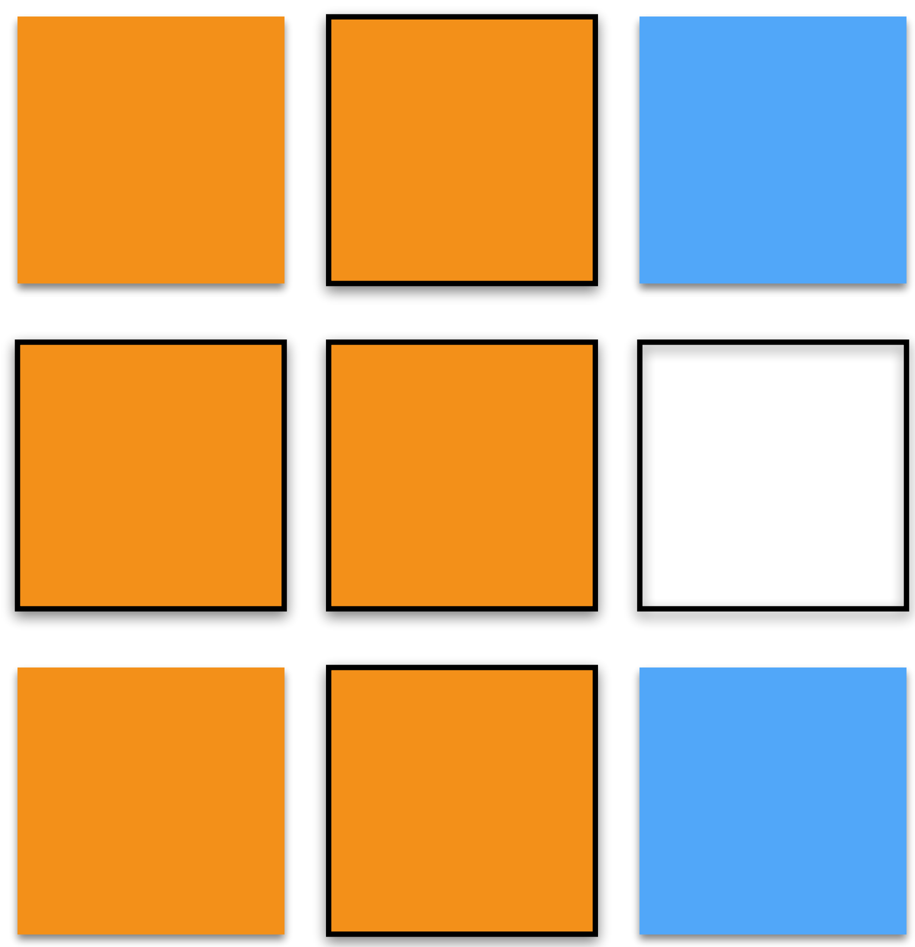
Upsampling: Nearest Neighbors



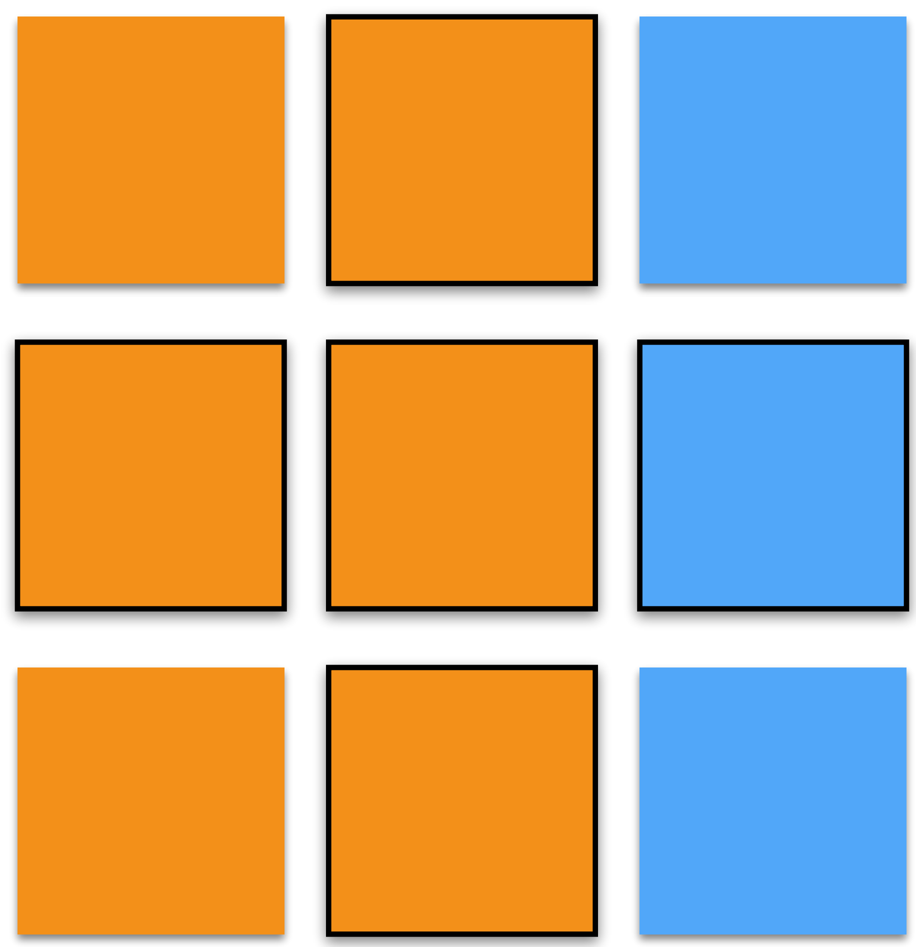
Upsampling: Nearest Neighbors



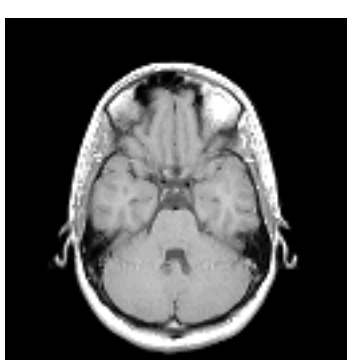
Upsampling: Nearest Neighbors



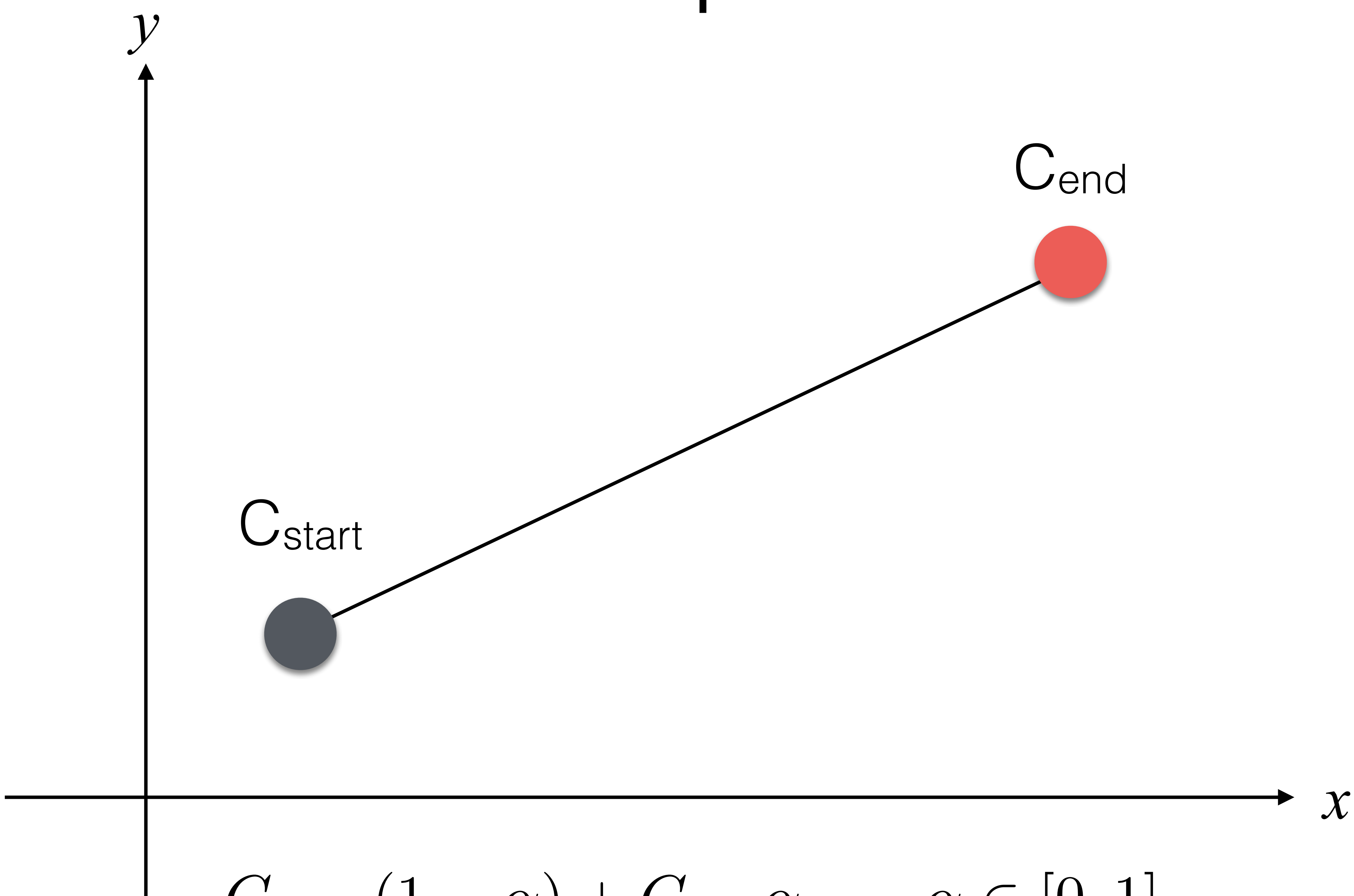
Upsampling: Nearest Neighbors



Upsampling: Nearest Neighbors

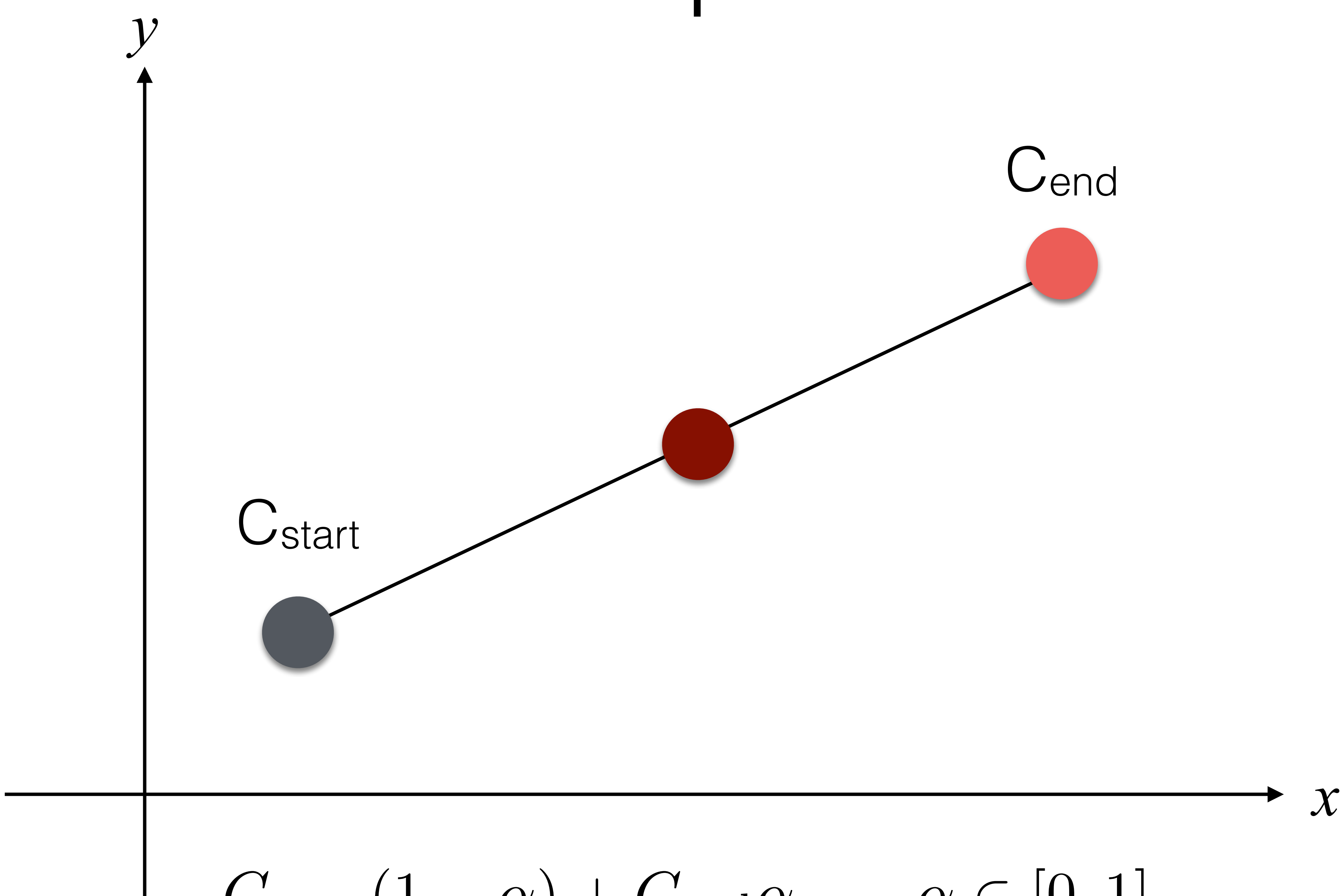


Upsampling 1D: Linear Interpolation



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

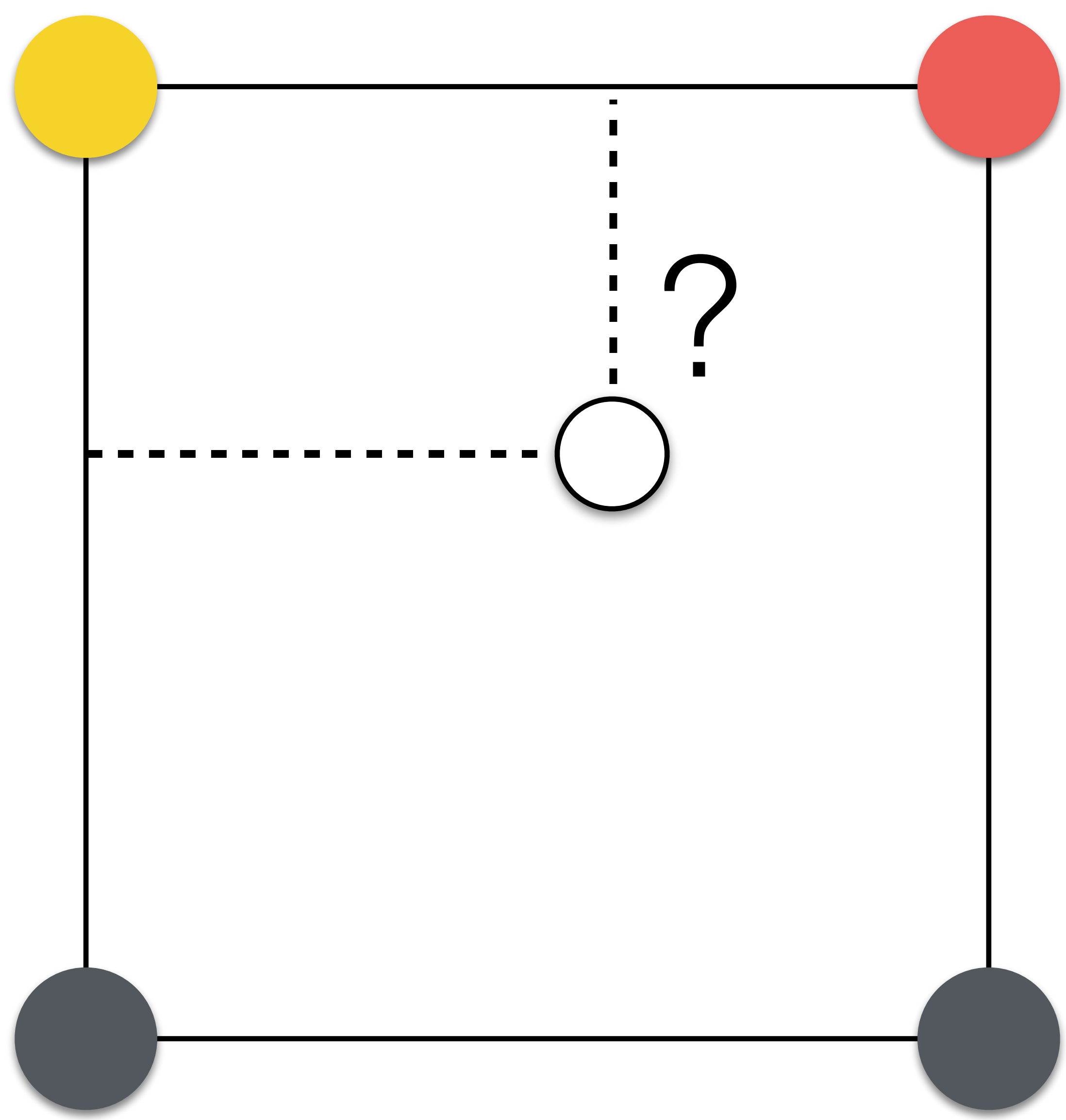
Upsampling 1D: Linear Interpolation



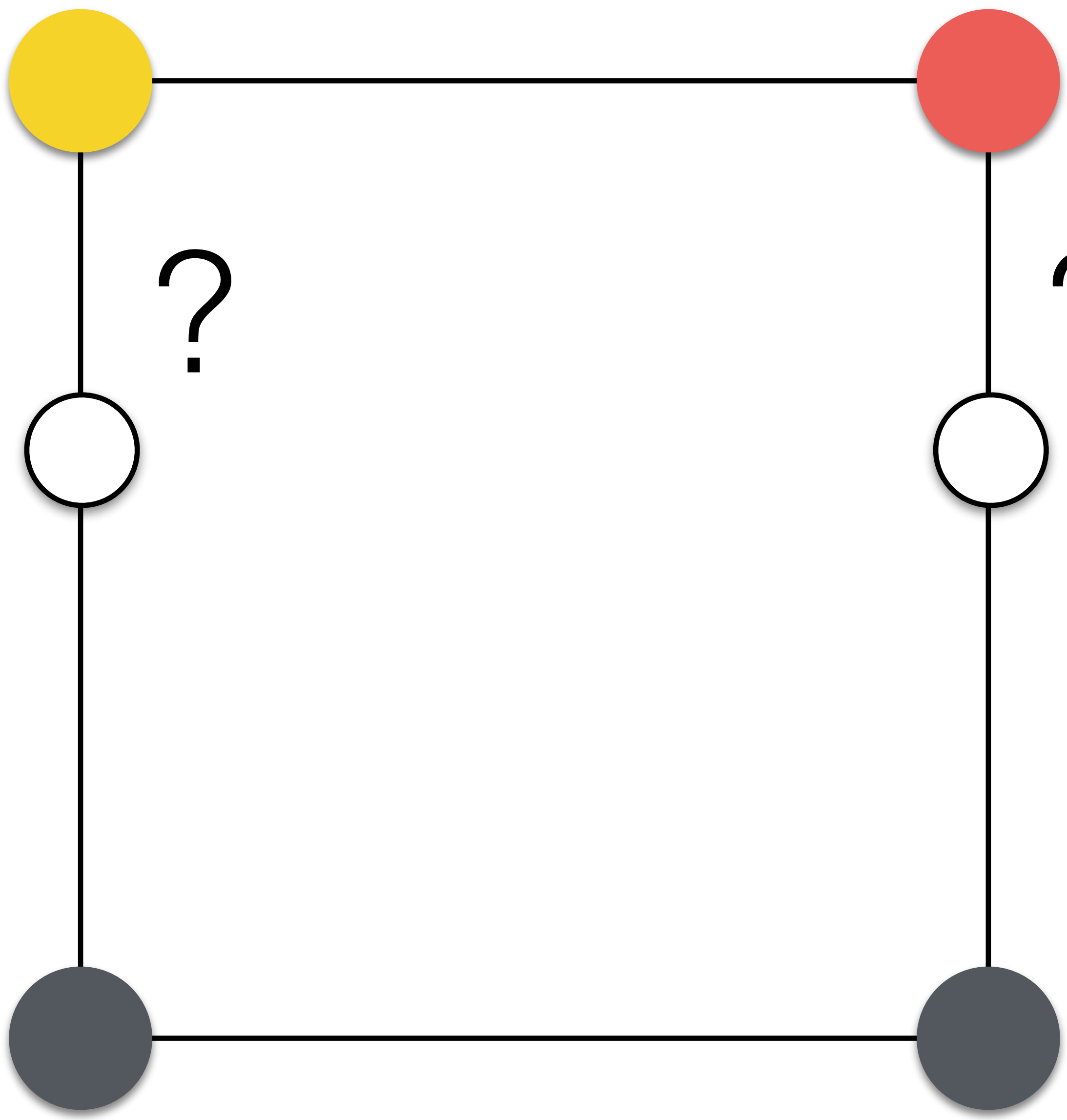
$$C_{\text{start}}(1-\alpha) + C_{\text{end}}\alpha \qquad \alpha \in [0, 1]$$

This becomes a bi-linear interpolation in 2D!

Bilinear Upsampling 2D

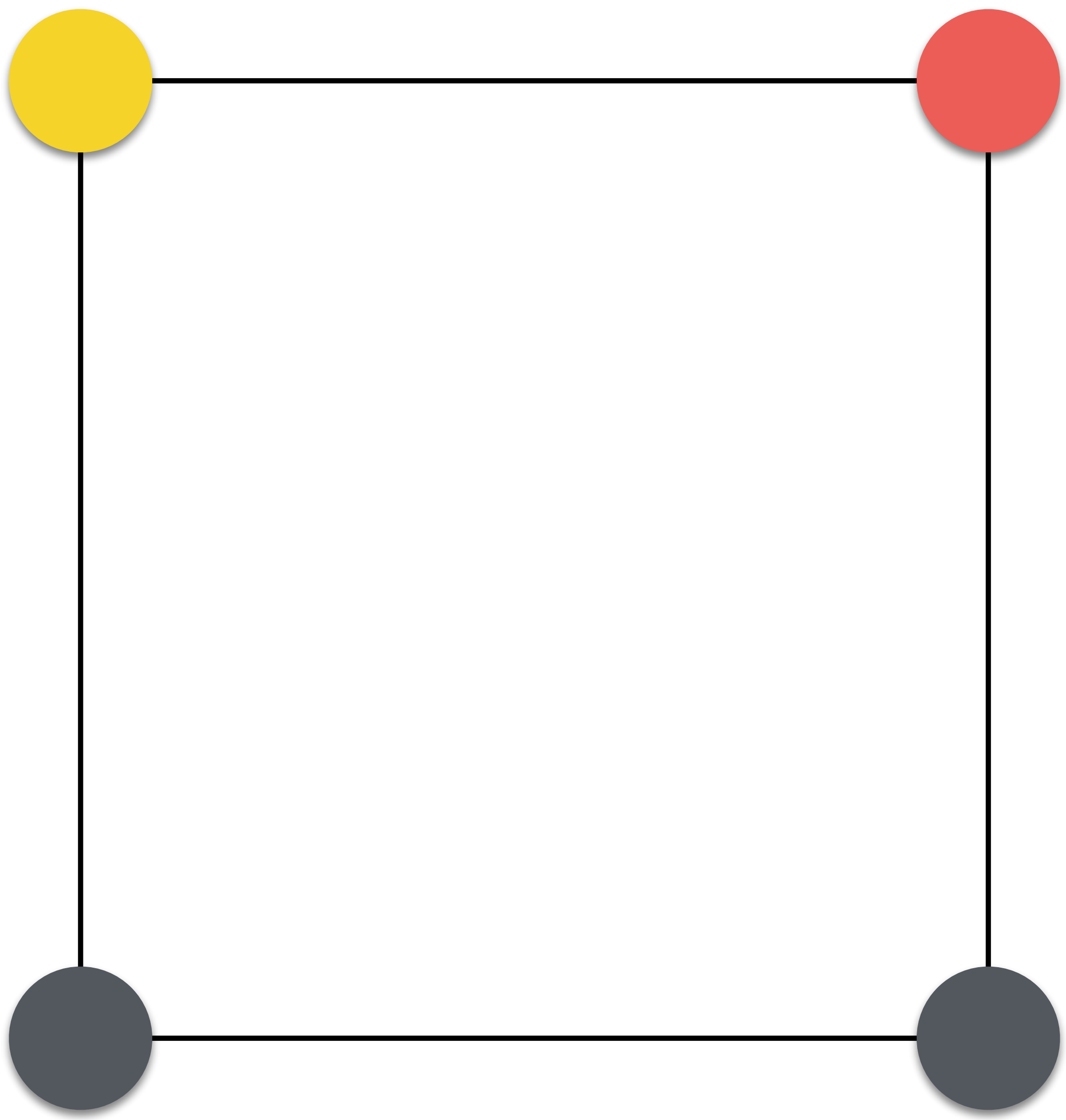


Bilinear Upsampling 2D

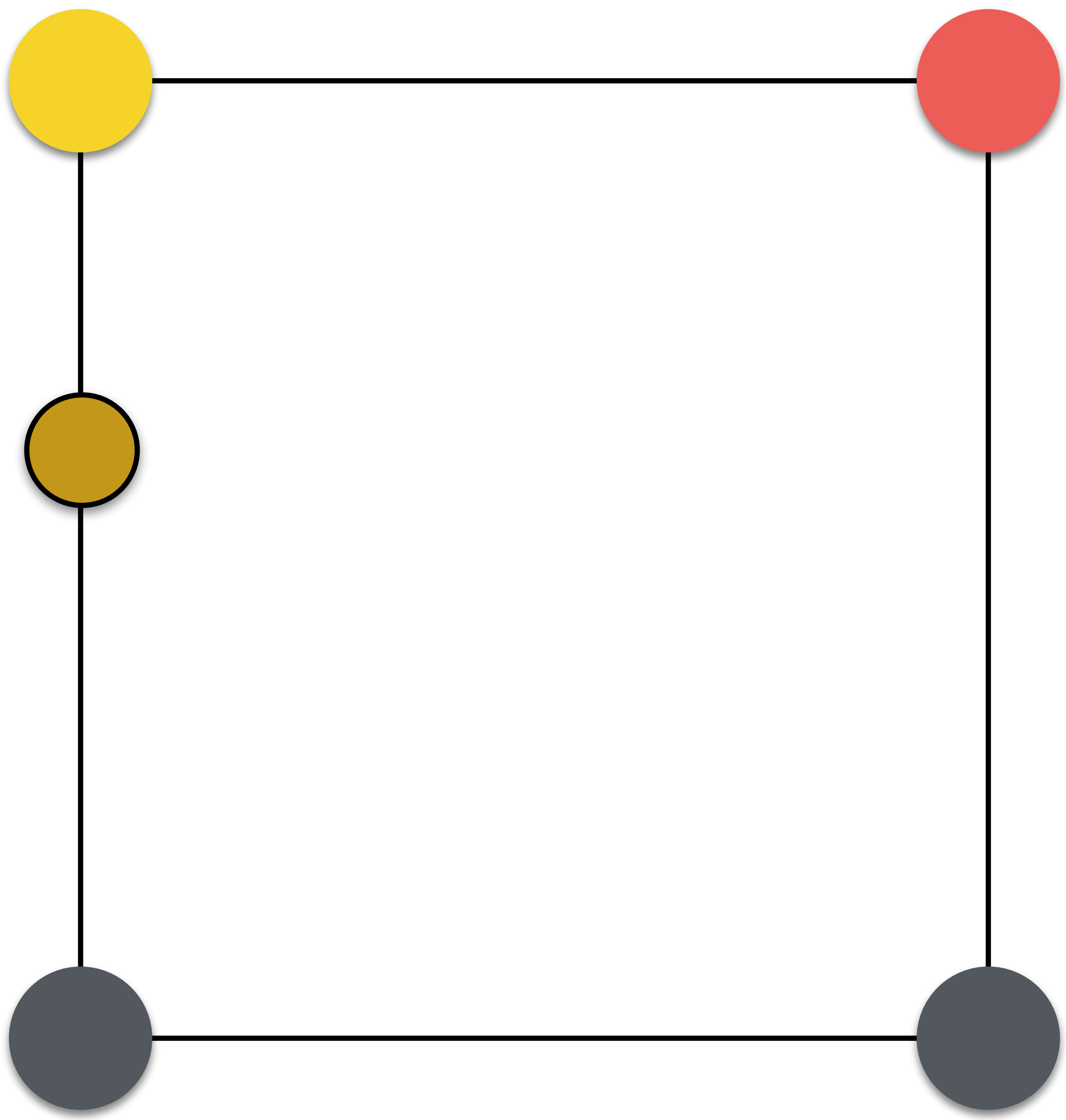


Two 1D Linear Interpolations for both!

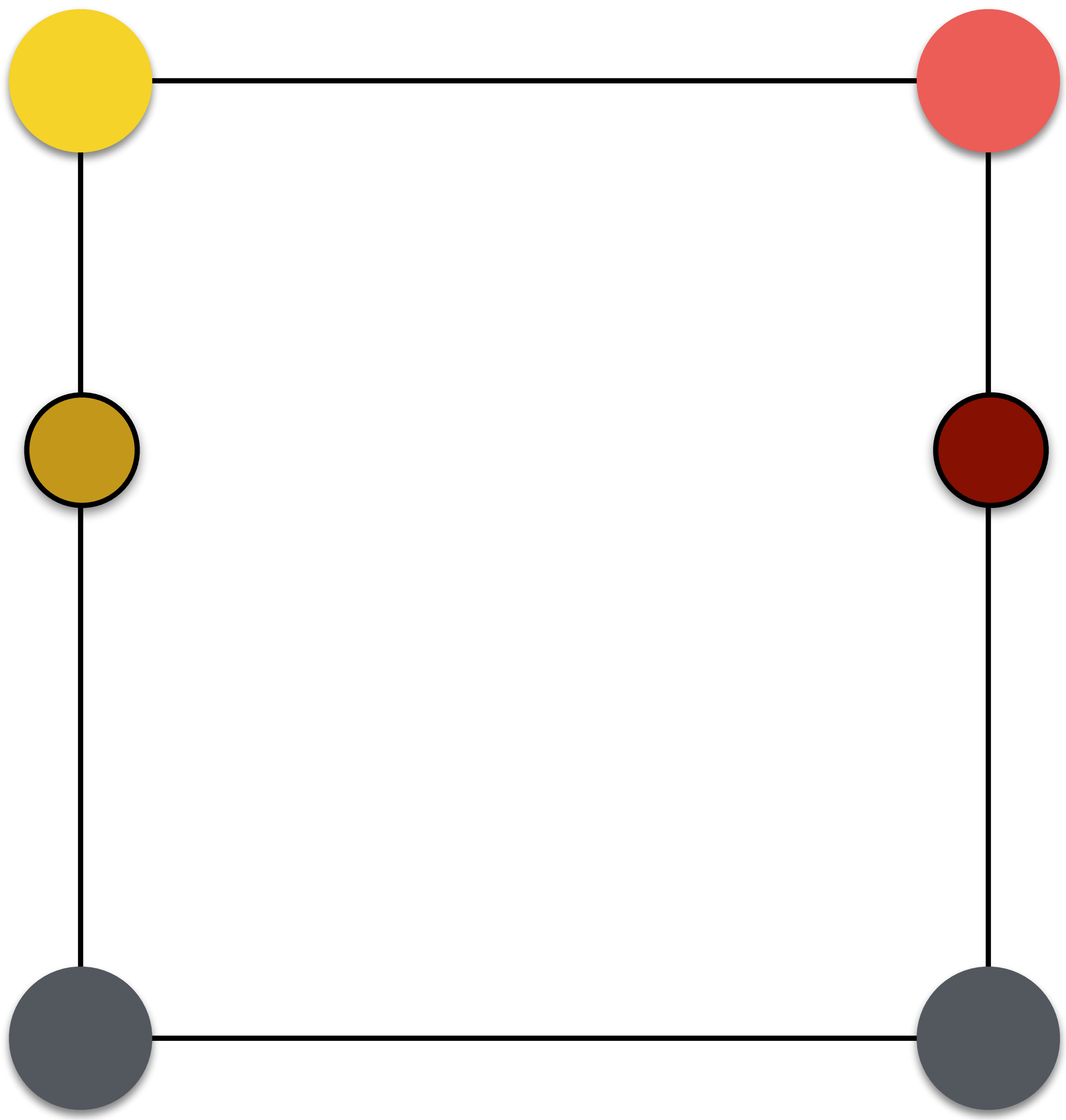
Bilinear Upsampling 2D



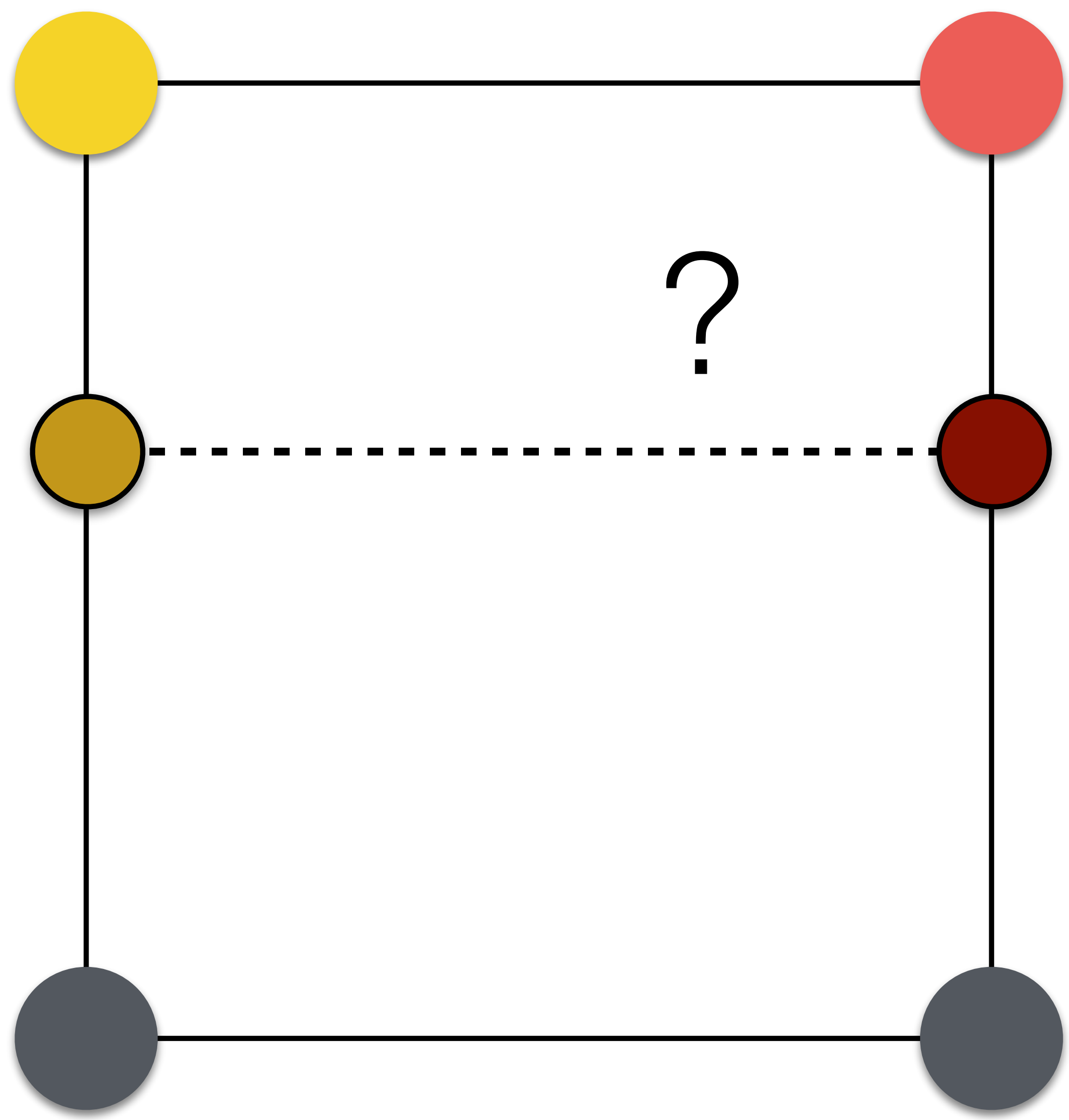
Bilinear Upsampling 2D



Bilinear Upsampling 2D

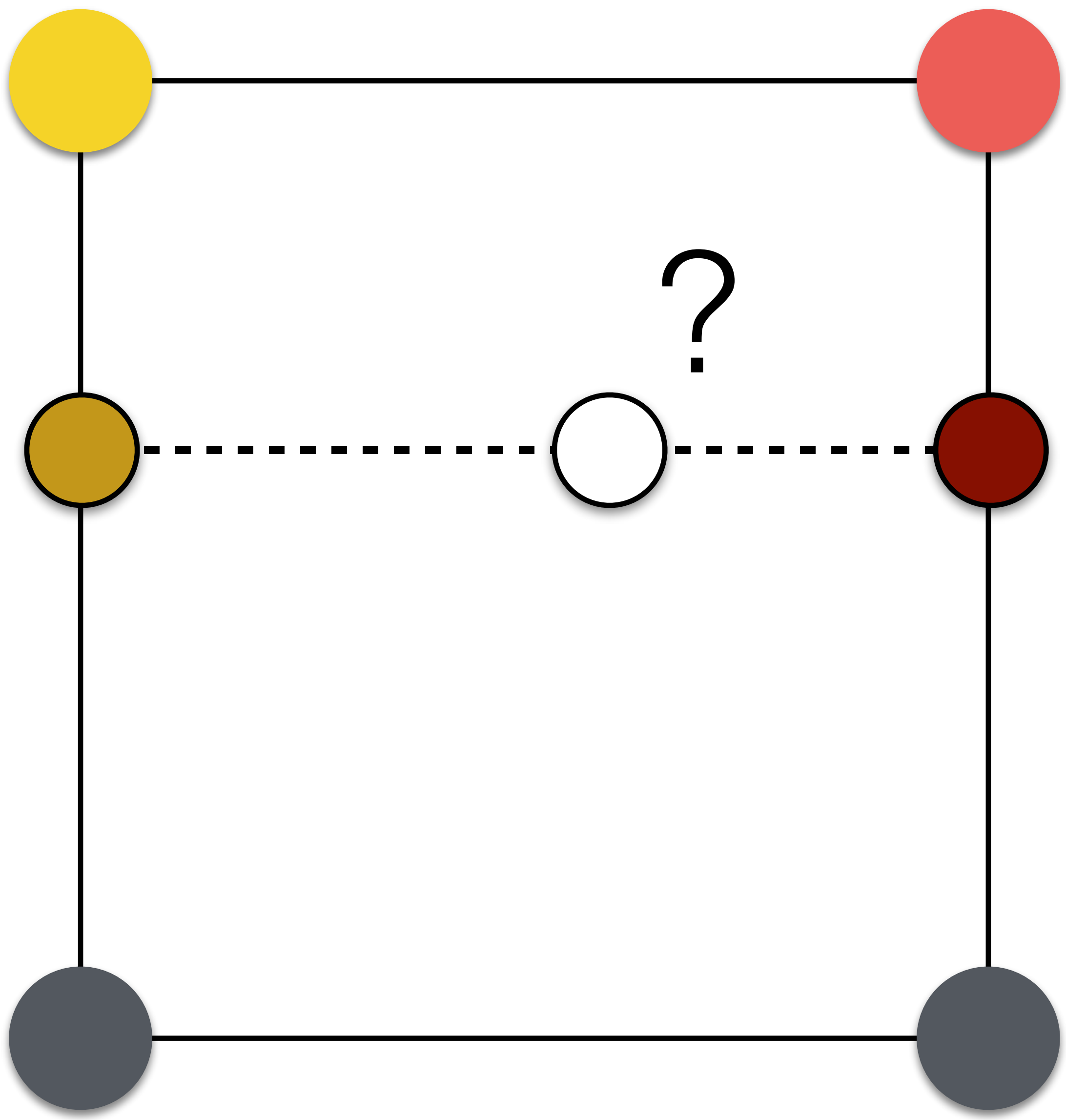


Bilinear Upsampling 2D



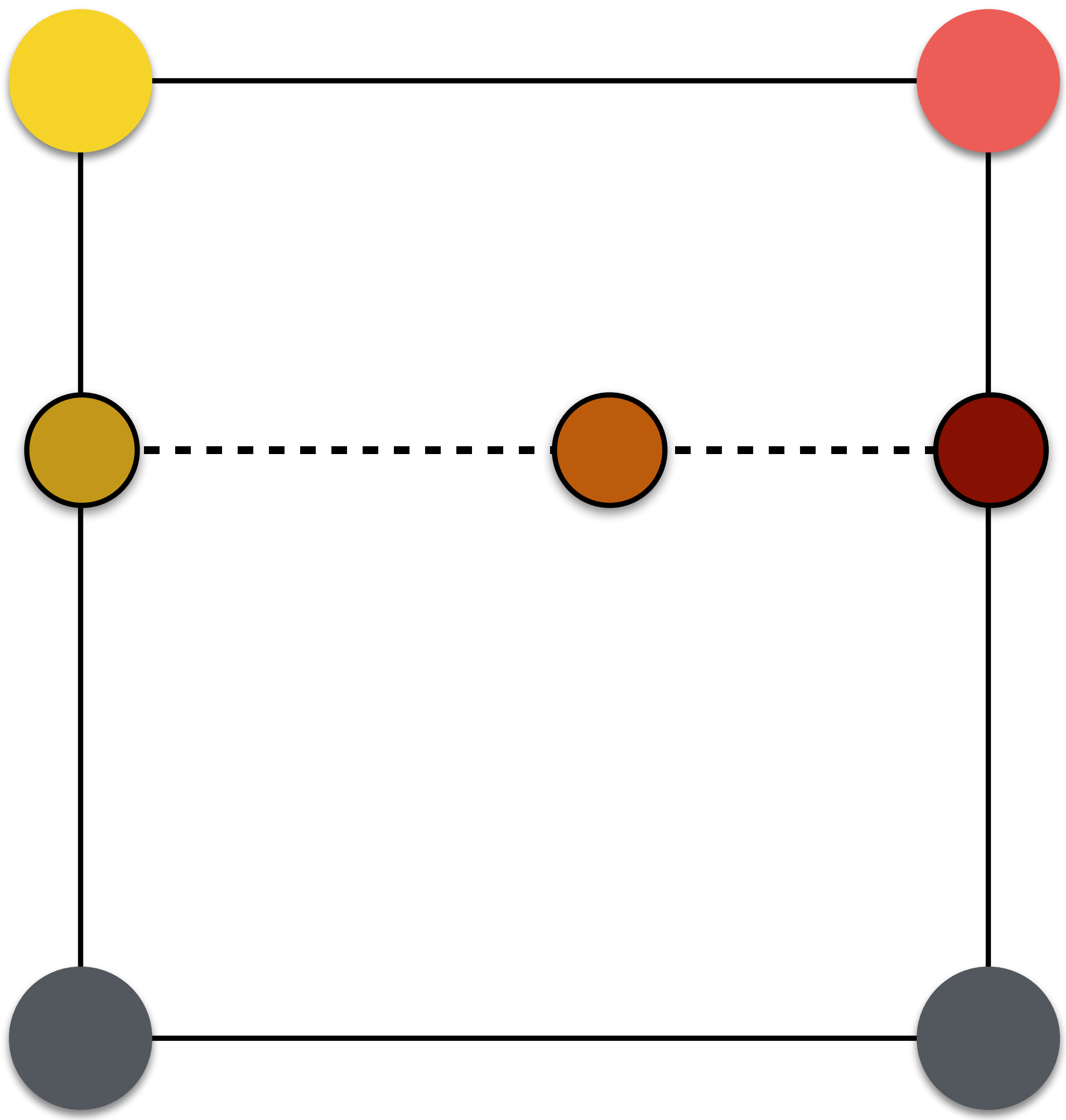
One 1D Linear Interpolation

Bilinear Upsampling 2D

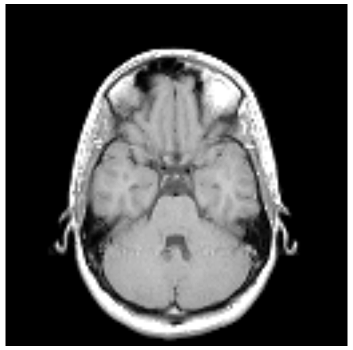


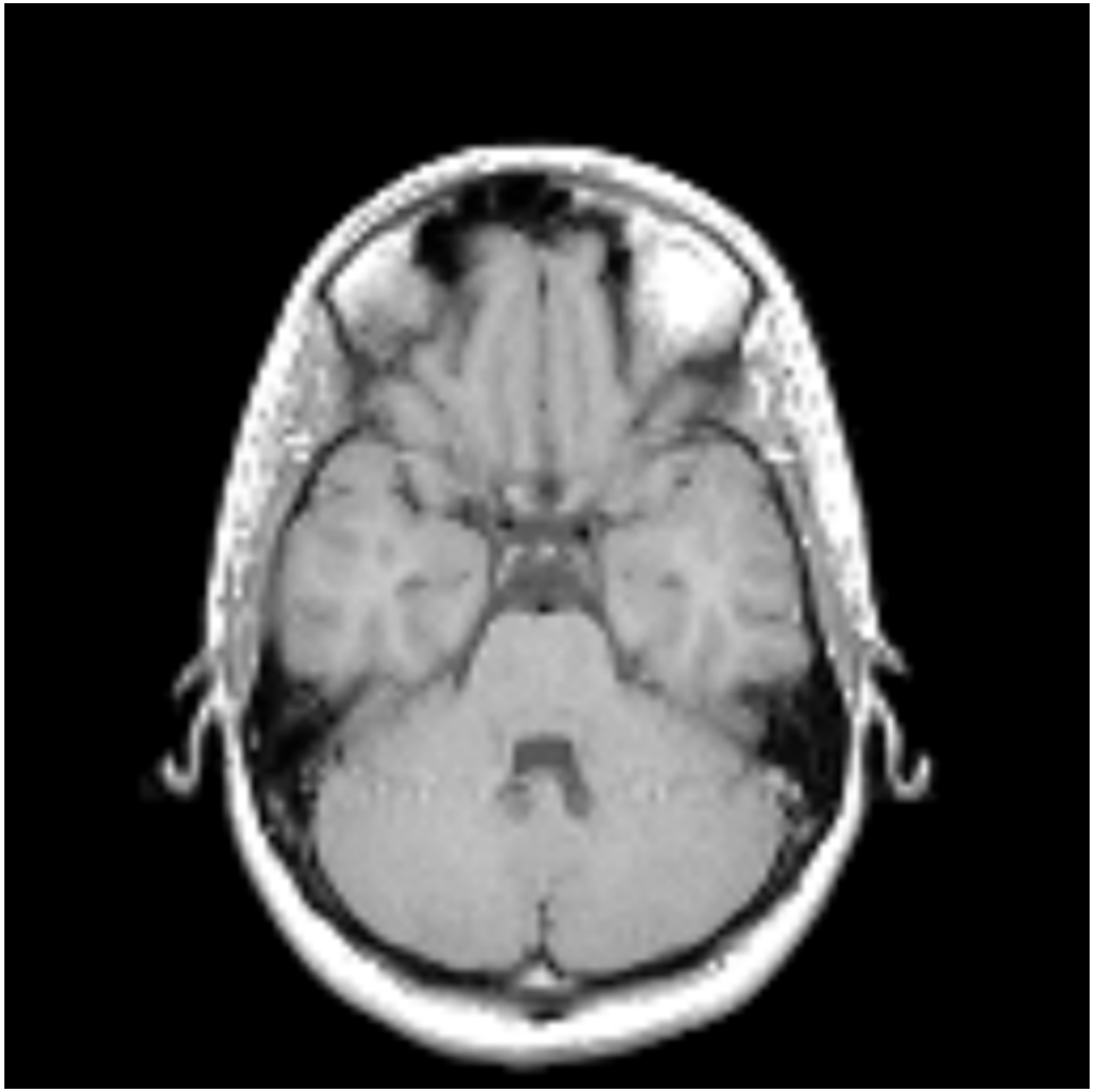
One 1D Linear Interpolation

Bilinear Upsampling 2D



Bilinear Upsampling Example



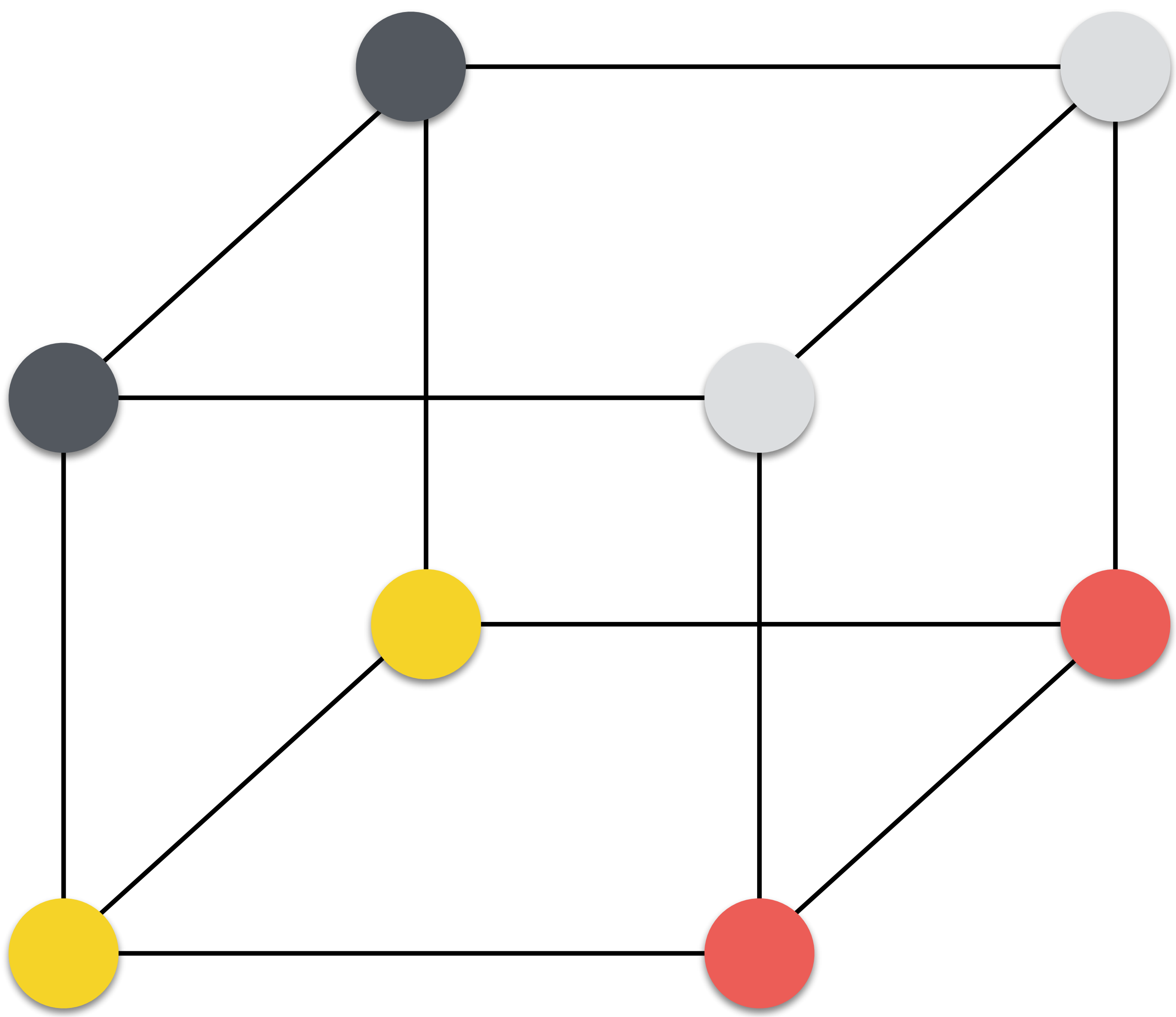


What’s about 3D?

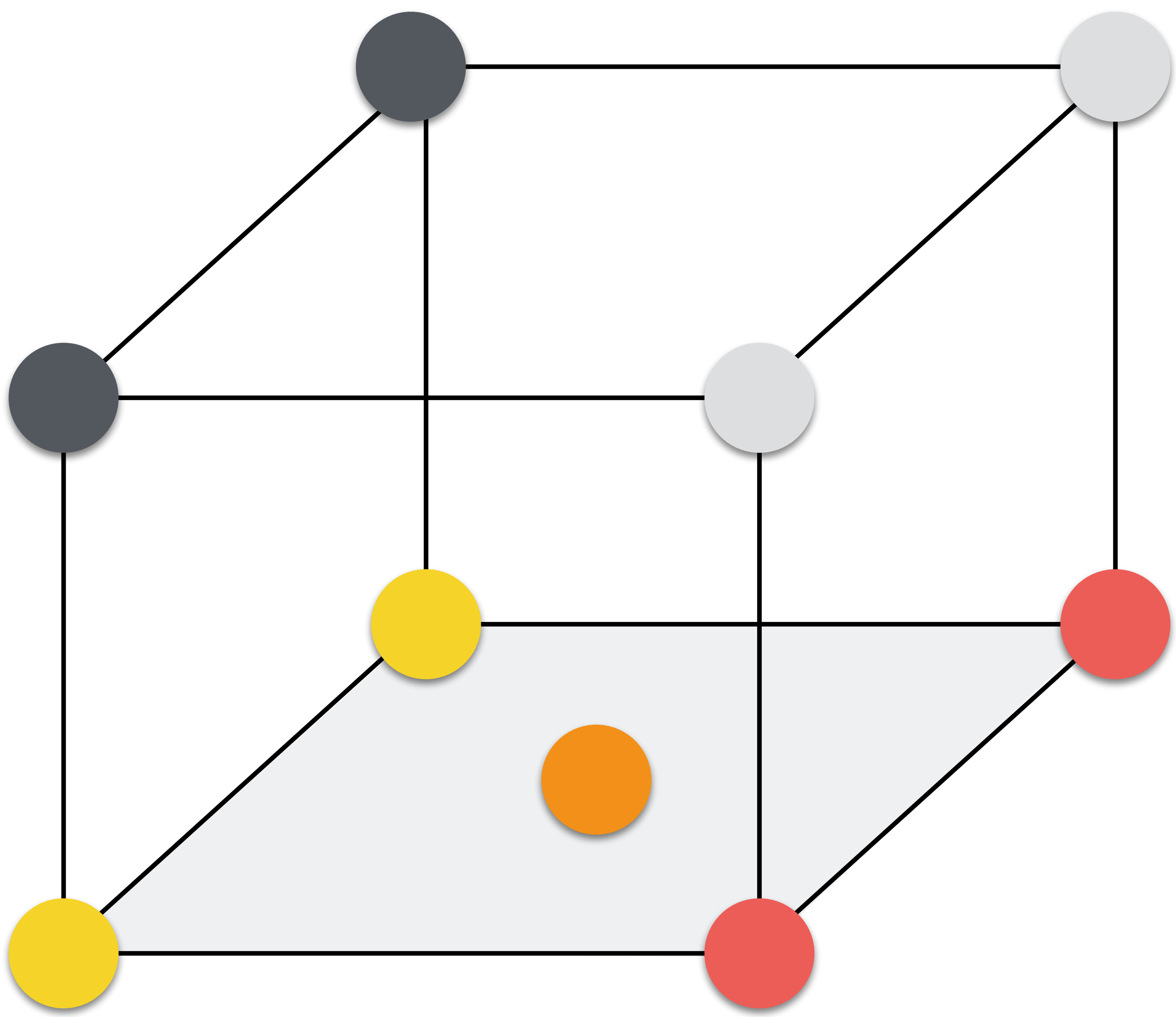
Trilinear Upsampling

- We want to upsample the whole volume:
  - First, we apply bilinear upsampling to all slices of the volume; i.e., 2D images
  - Second, we linearly interpolate between slices to obtain a new slice

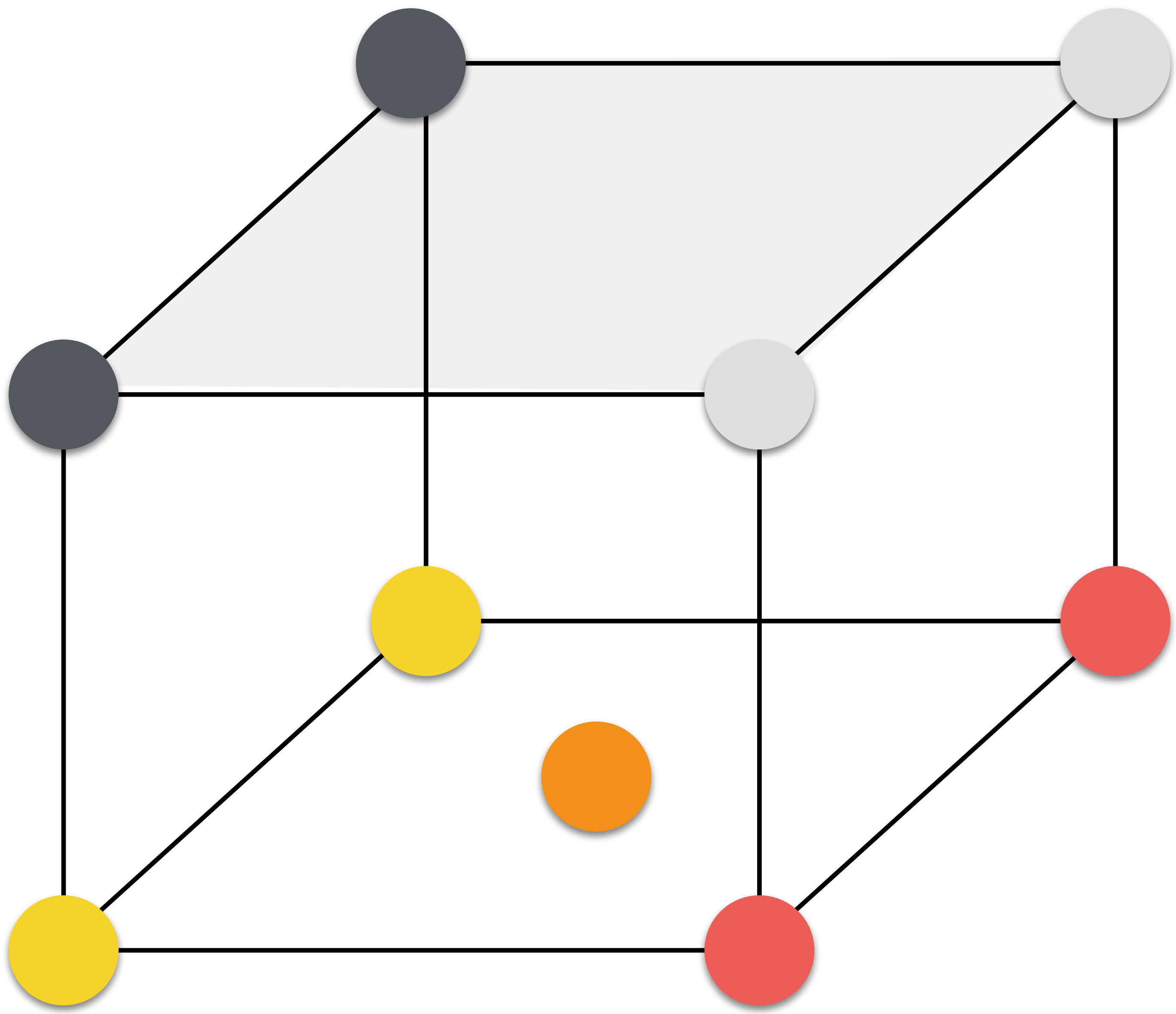
Trilinear Upsampling



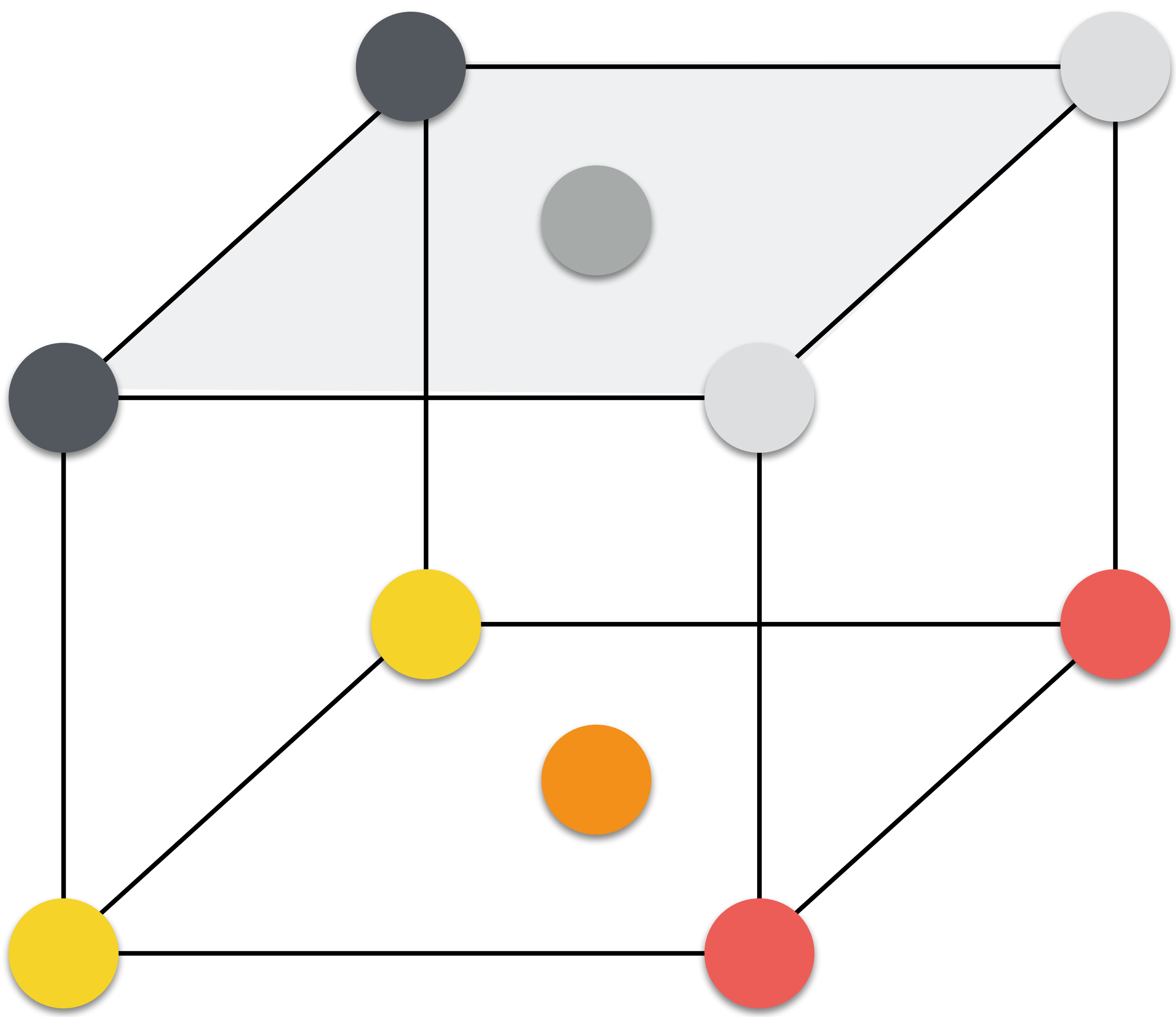
Trilinear Upsampling



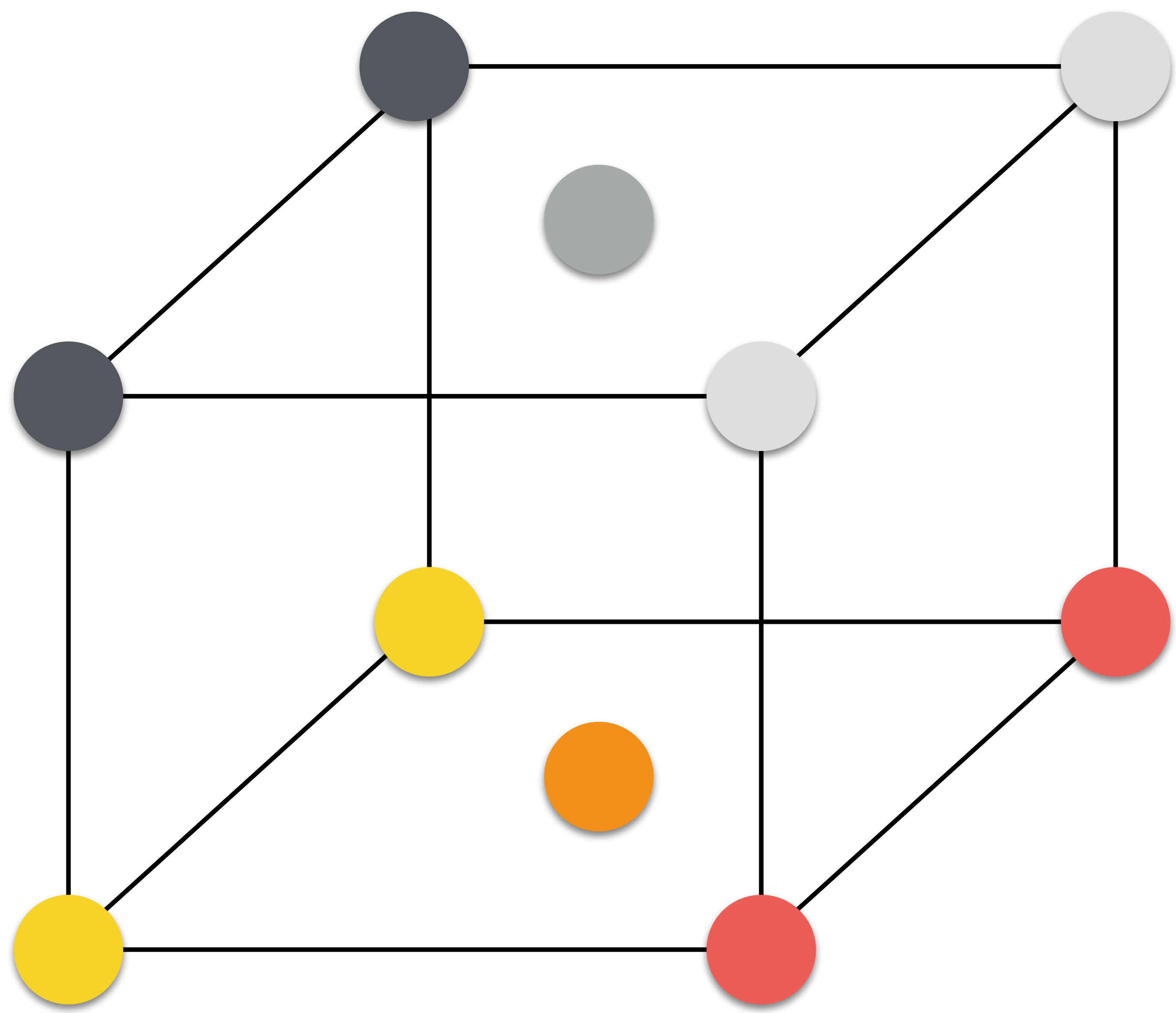
Trilinear Upsampling



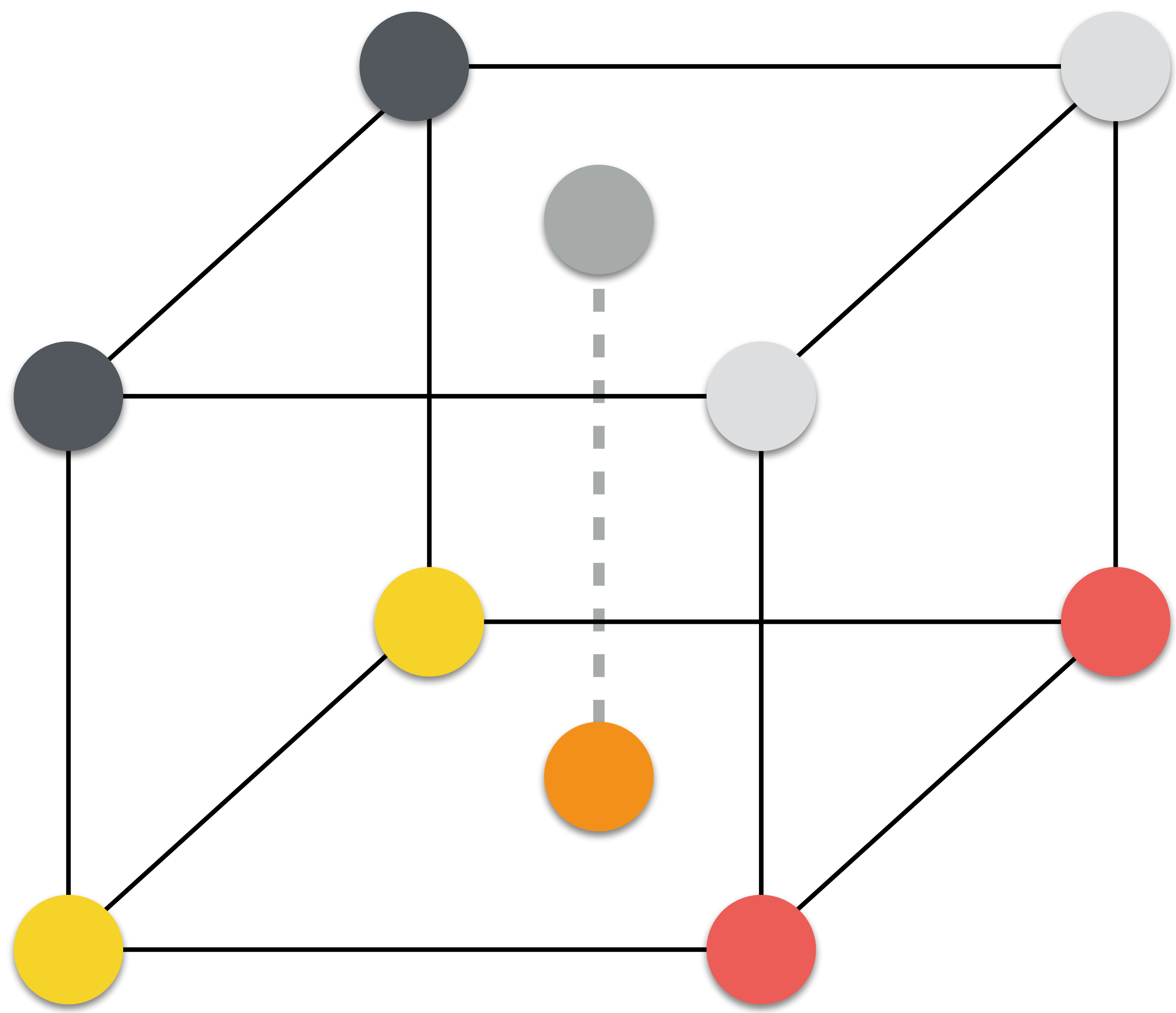
Trilinear Upsampling



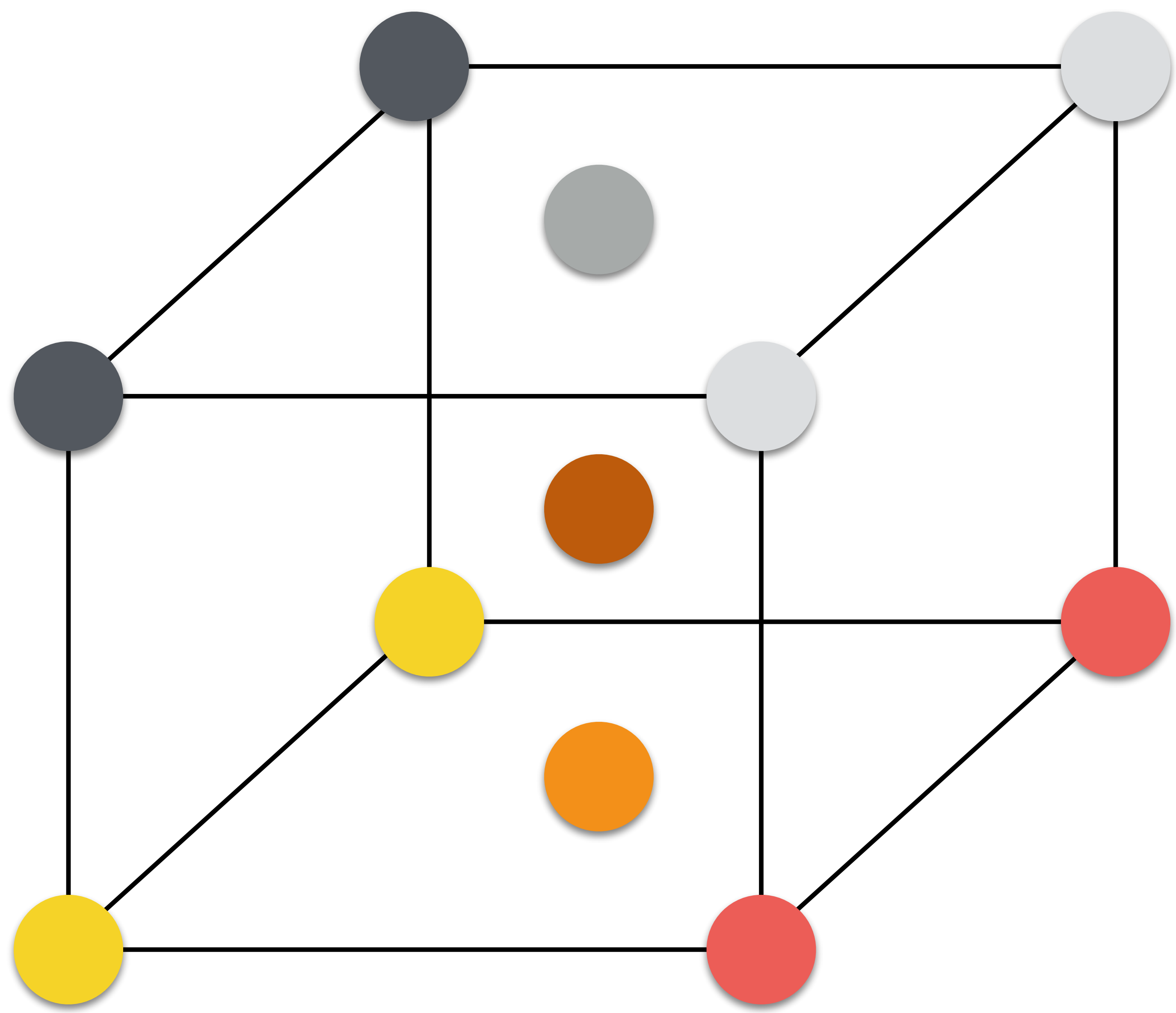
Trilinear Upsampling



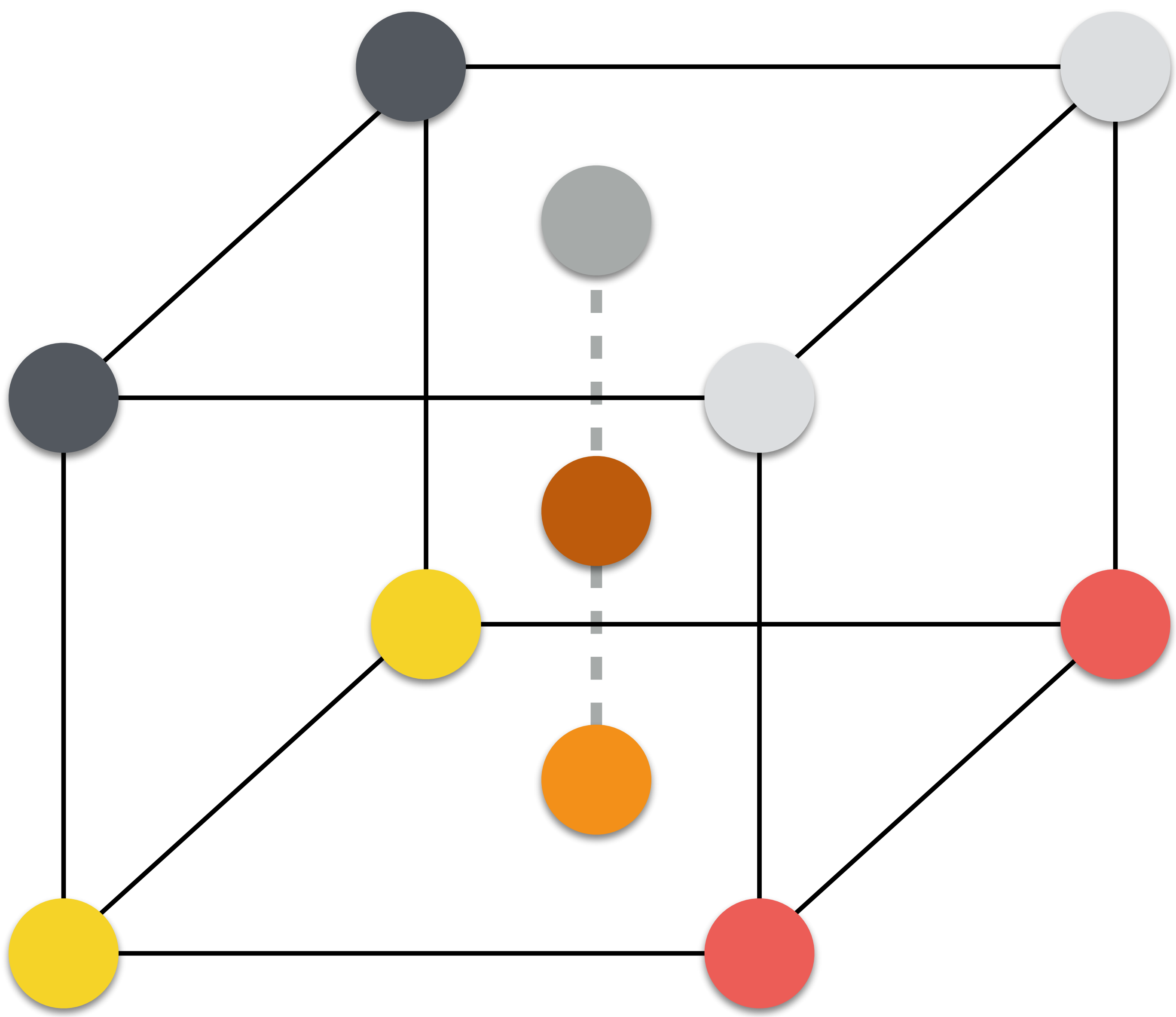
Trilinear Upsampling



Trilinear Upsampling

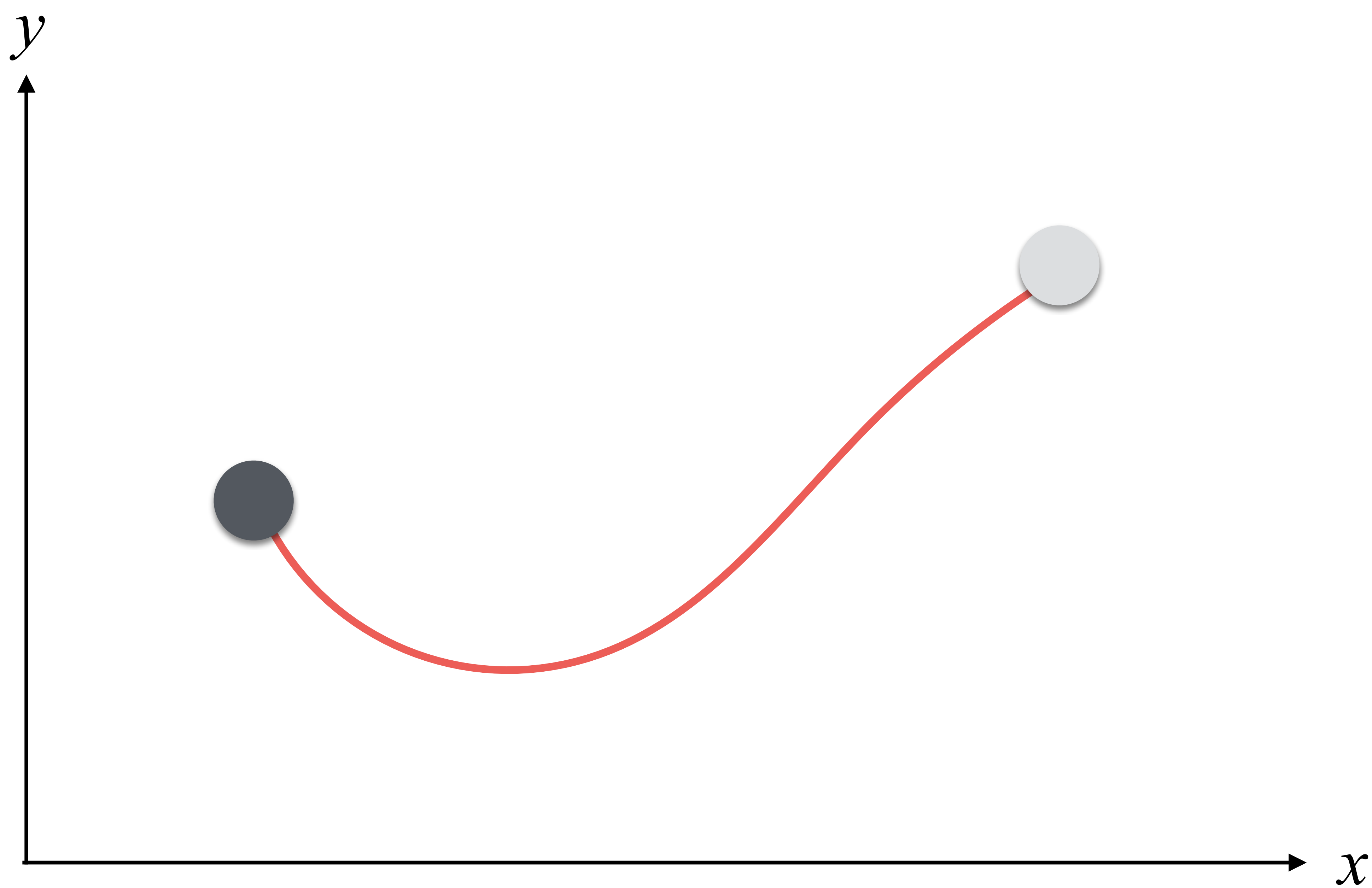


Trilinear Upsampling



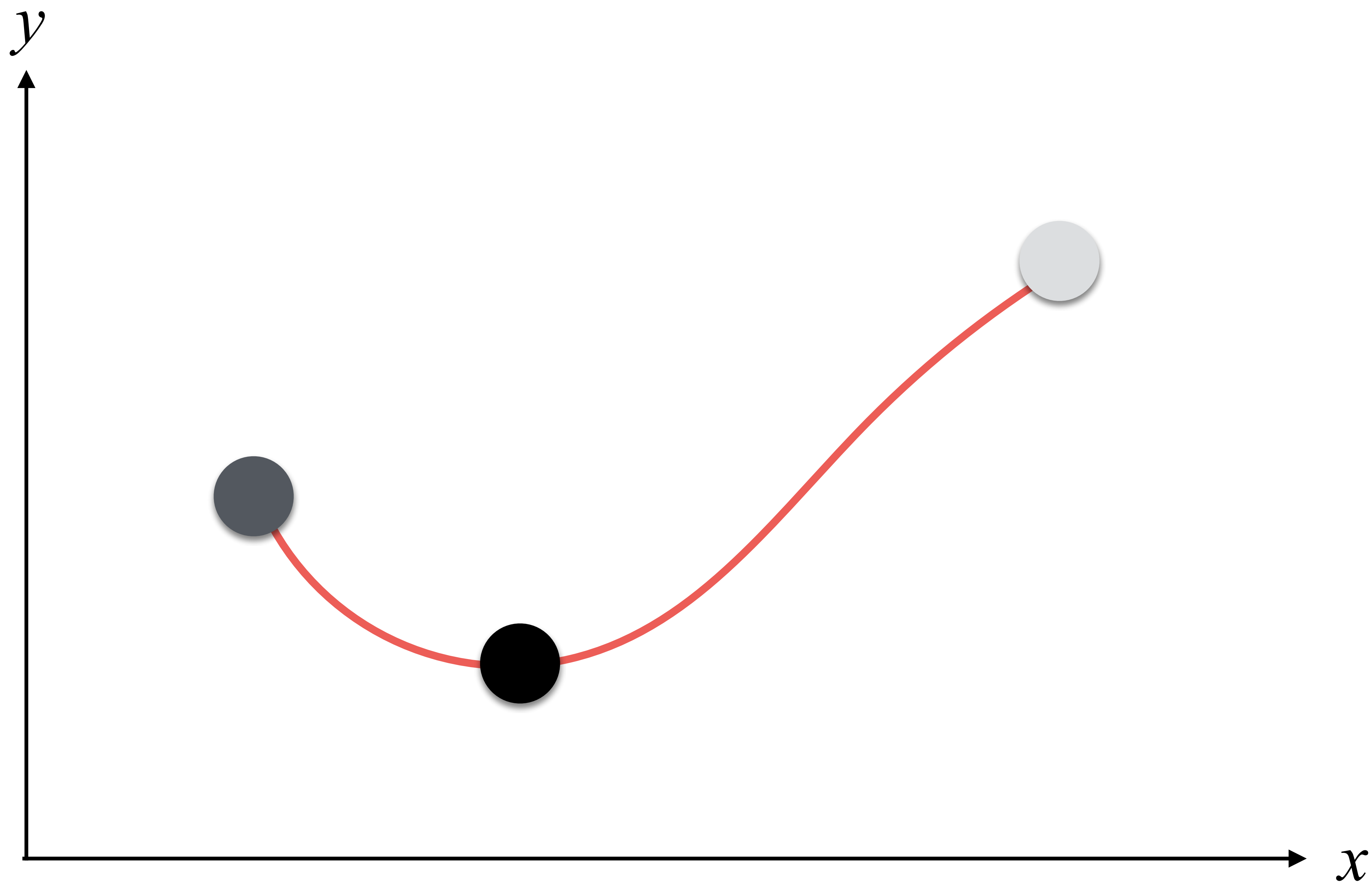
Can we do it better?

Bicubic Usampling (2D)



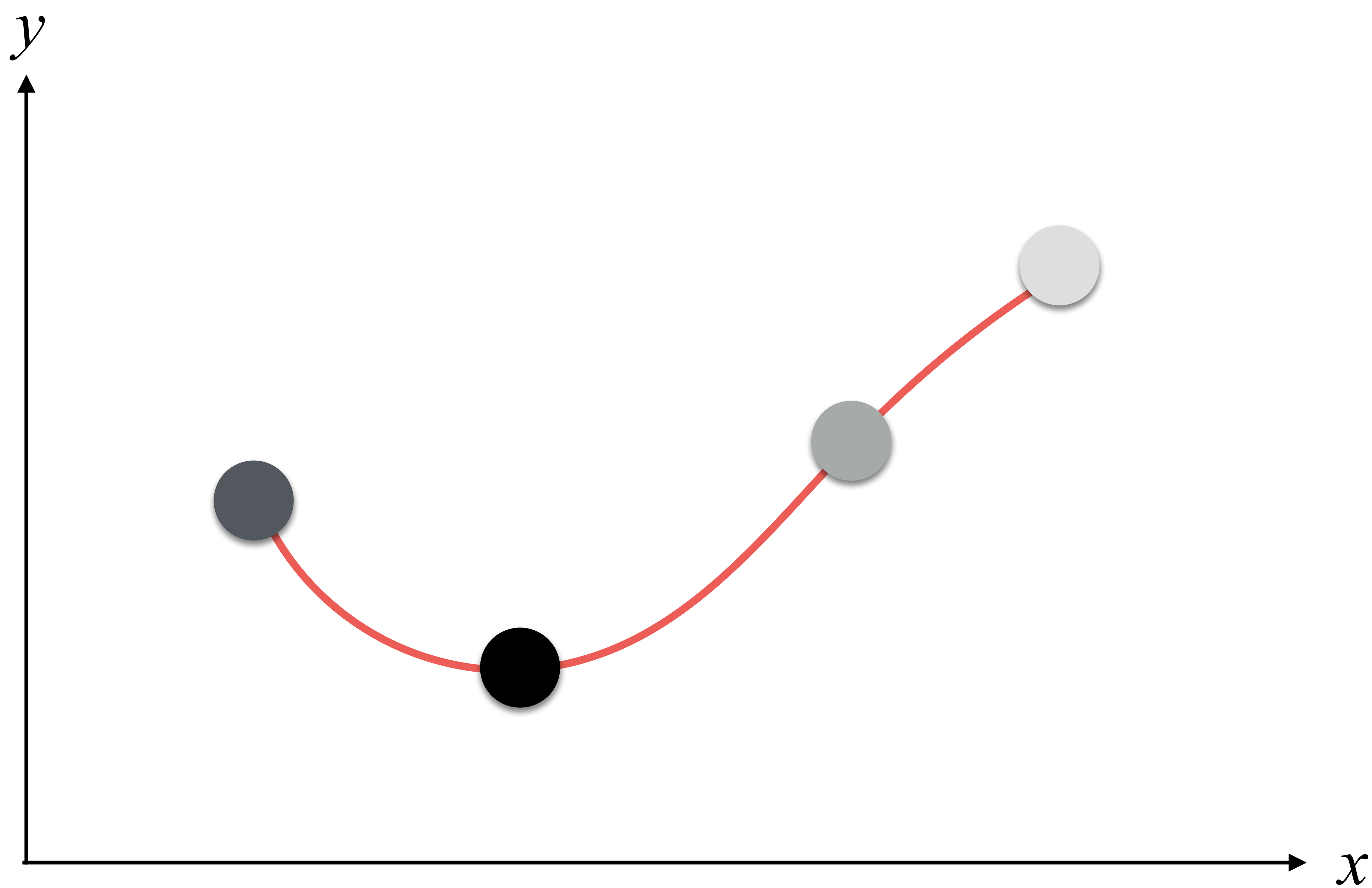
$$C(x) = \sum_{i=0}^{3} a_i x^i \quad \Rightarrow \quad C(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{i,j} x^i y^j$$

Bicubic Usampling (2D)



$$C(x) = \sum_{i=0}^{3} a_i x^i \quad \Rightarrow \quad C(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{i,j} x^i y^j$$

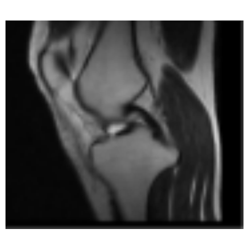
Bicubic Usampling (2D)



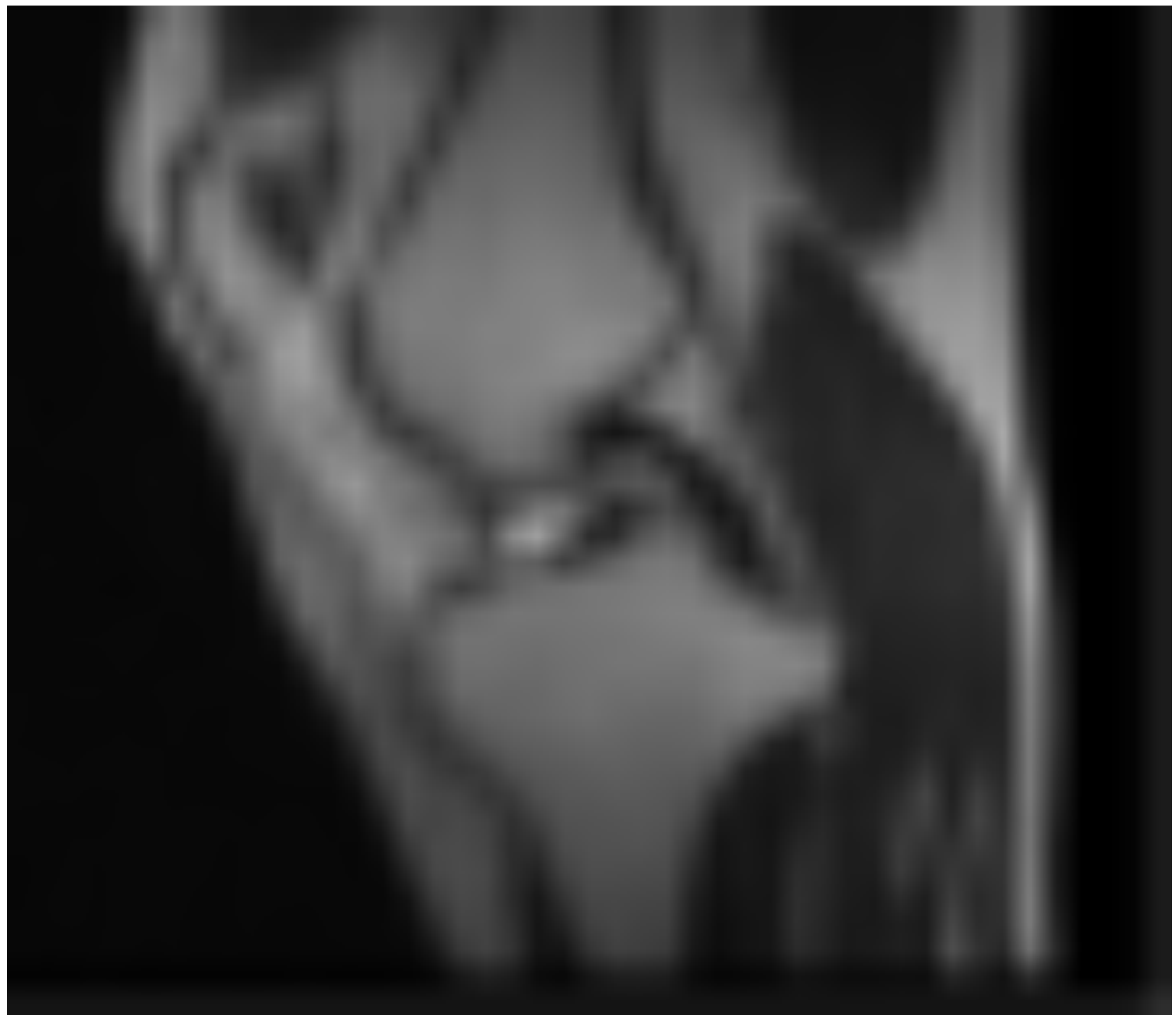
$$C(x) = \sum_{i=0}^{3} a_i x^i$$

$$C(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{i,j} x^i y^j$$

Bicubic Upsampling

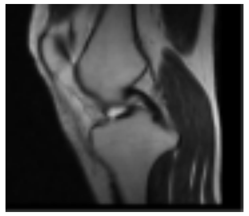


Input

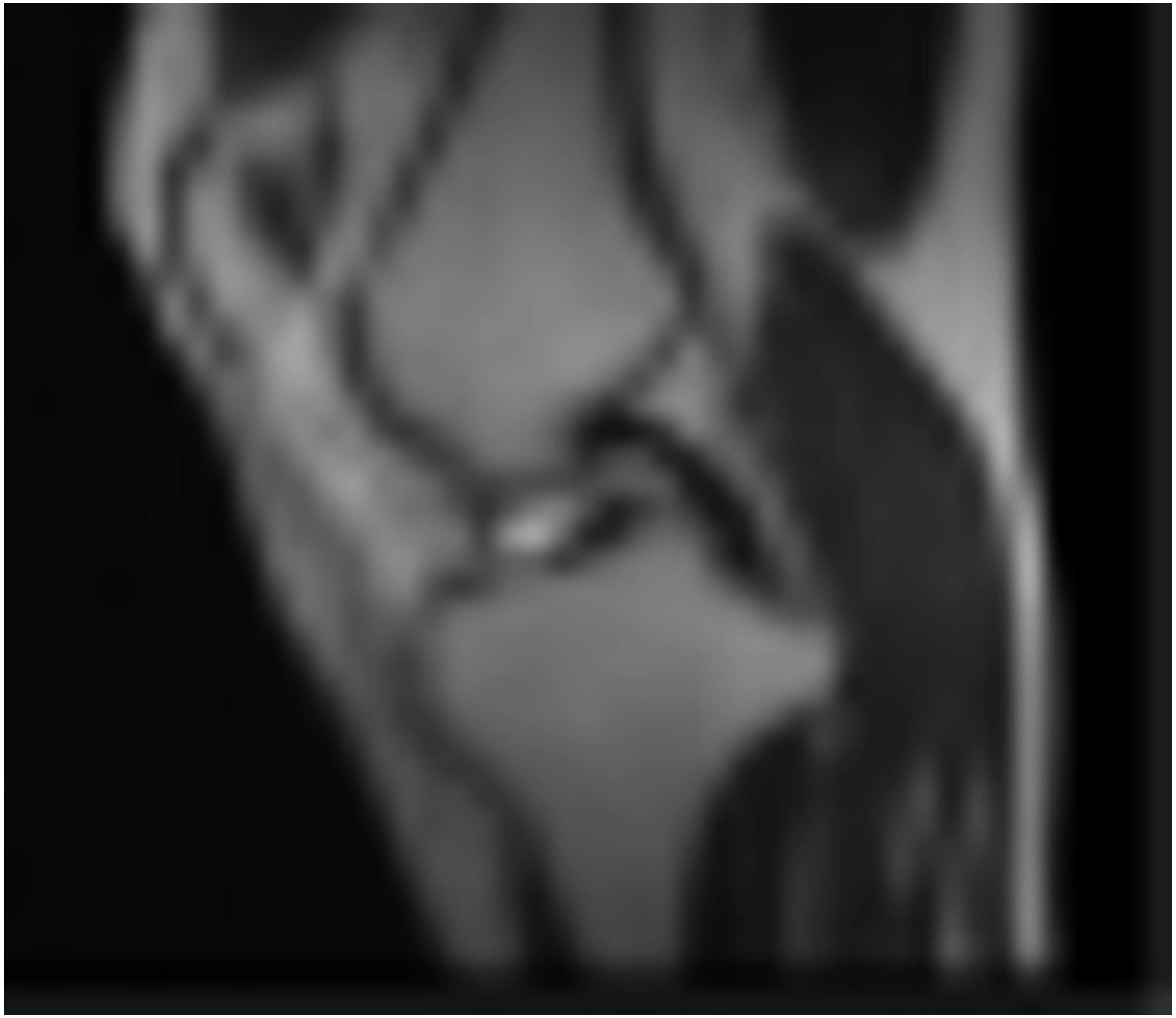


Upsampling

Bicubic Upsampling

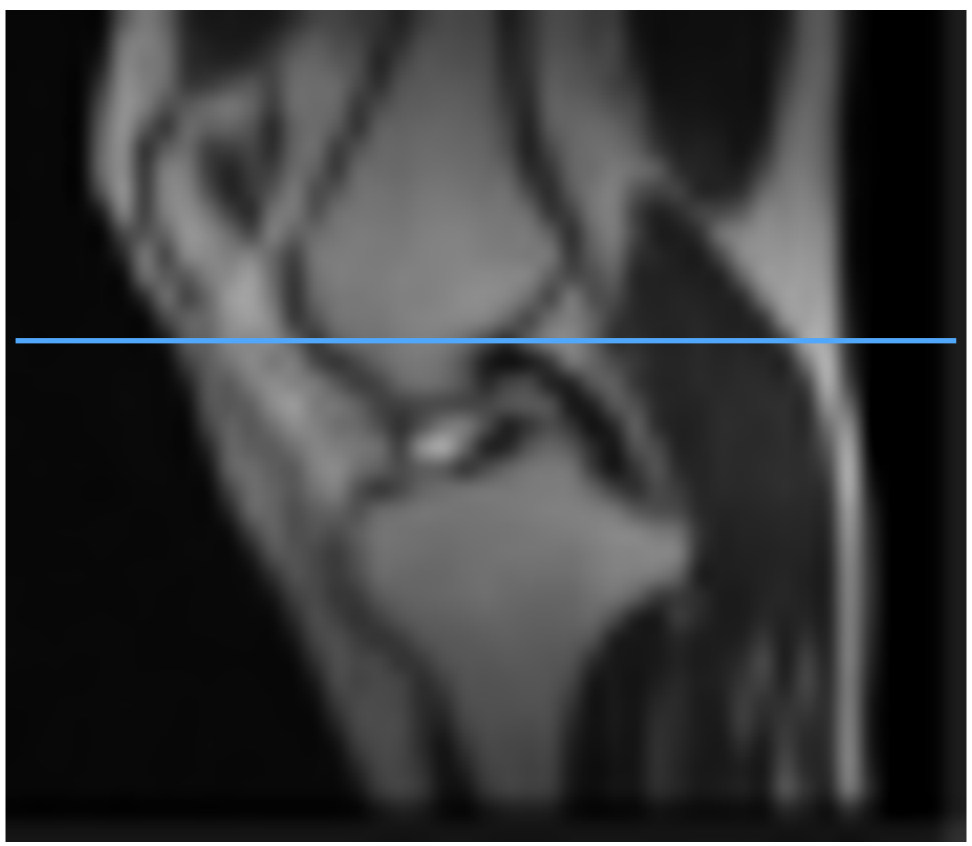


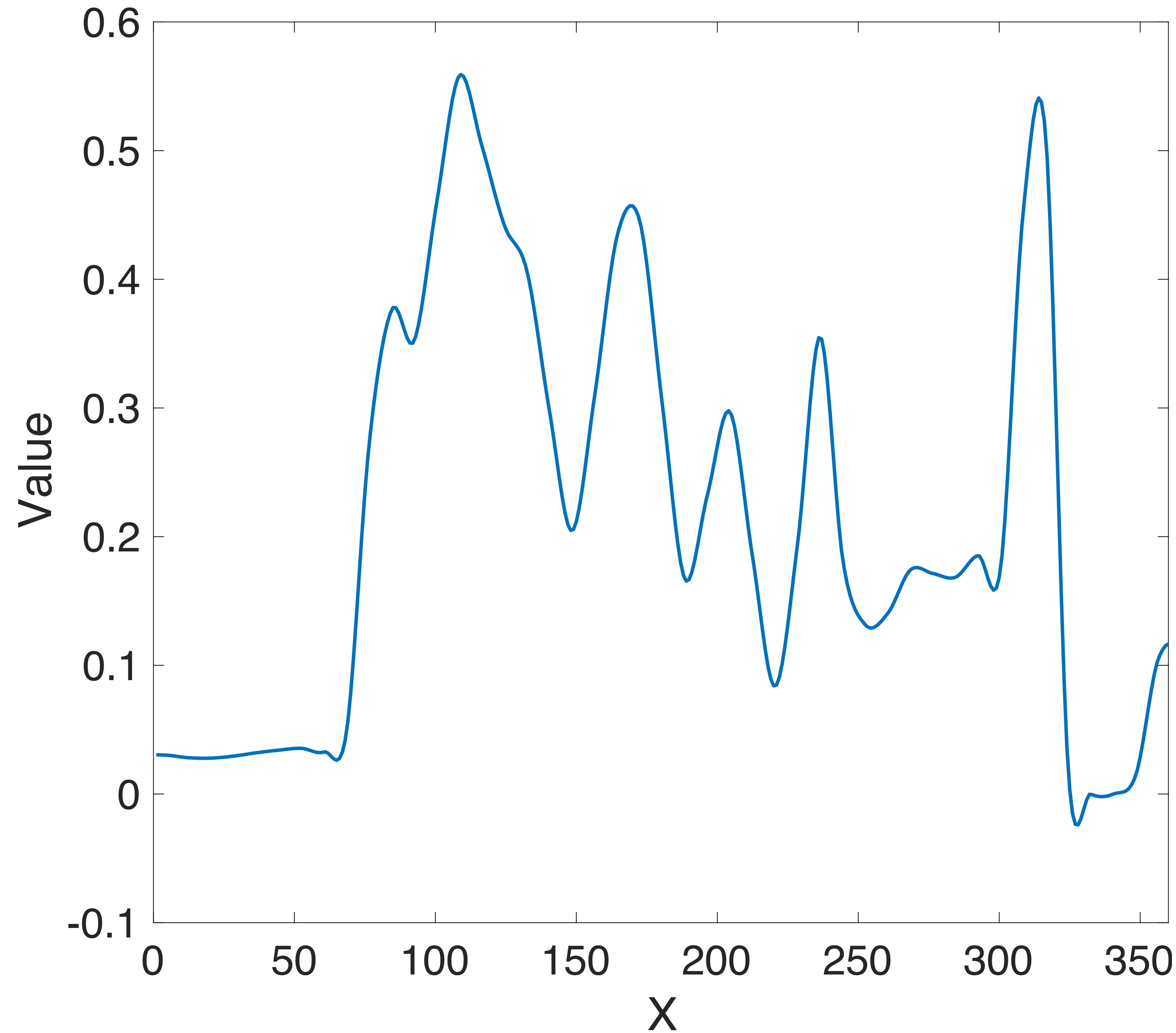
Input



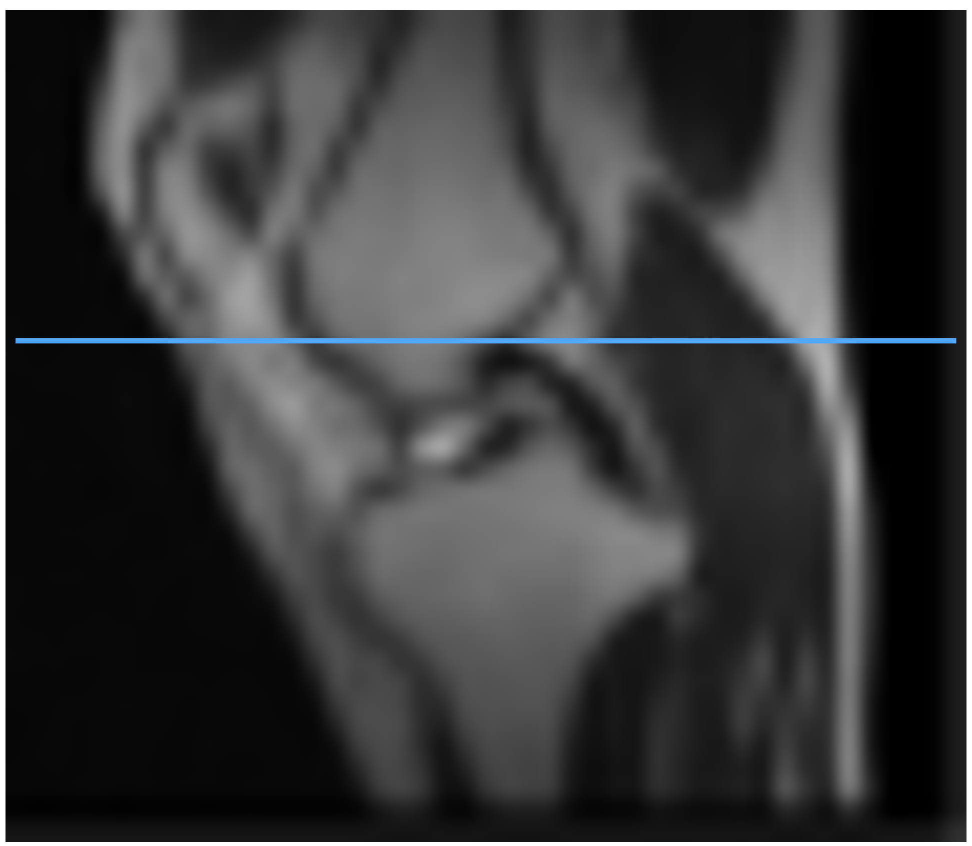
Upsampling

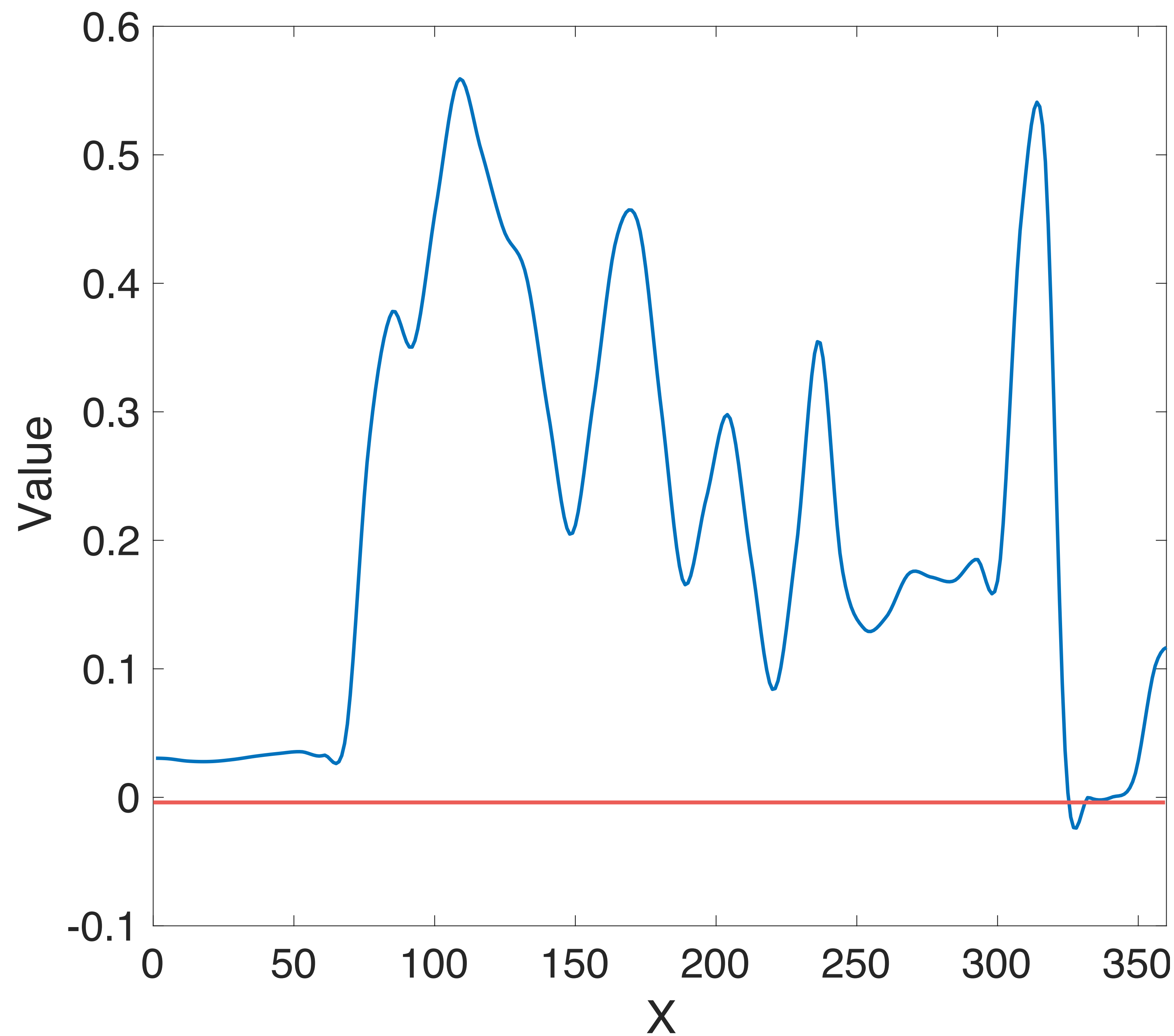
Bicubic Upsampling: Negative Values at Strong Edges





Bicubic Upsampling: Negative Values at Strong Edges





that’s all folks!