Fundamentals & Filtering

CS 482, Prof. Stein

Lecture 02

Motivation: Building a Library of Tools

How we represent images will define how we are capable of manipulating them.

Motivation: Building a Library of Tools

How we represent images will define how we are capable of manipulating them.








[Building Rome in a Day]

Motivation: Building a Library of Tools

How we represent images will define how we are capable of manipulating them.








Today, we will define some fundamental terms and start building a library of tools that allow us to manipulate images.

[Building Rome in a Day]

Reading & Slide Credits

Readings:

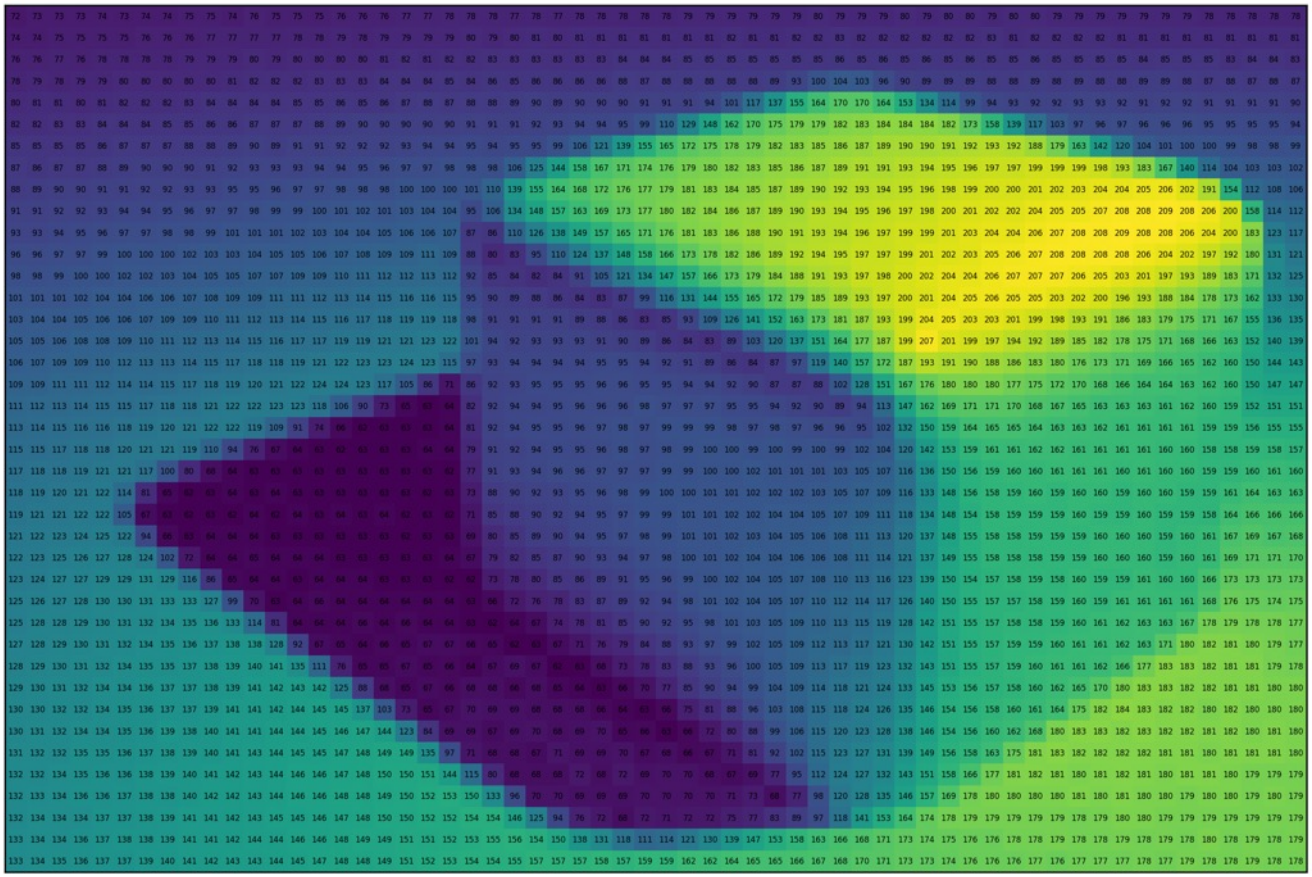
- Klette Ch 1 & Ch 2.0–2.2
- Szeliski Ch 3.1–3.3

Slide Credits (from which many of these slides are either directly taken or adapted)

- CMU Computer Vision Course (Yannis Gkioulekas)
- Cornell Tech Computer Vision Course (Noah Snavely)

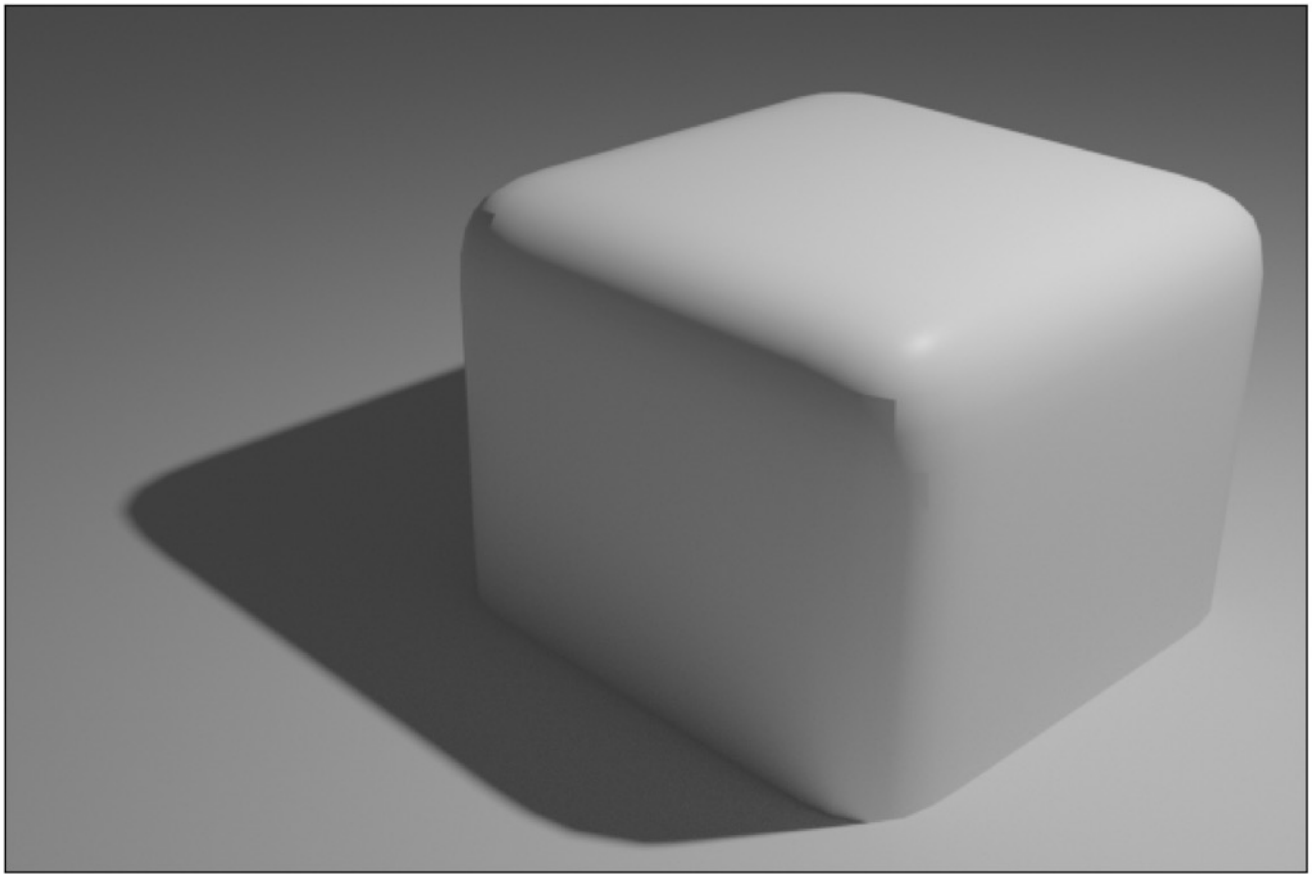
So what *is* an image?

For the purposes of this class, we will most often think of images as a grid of numbers:

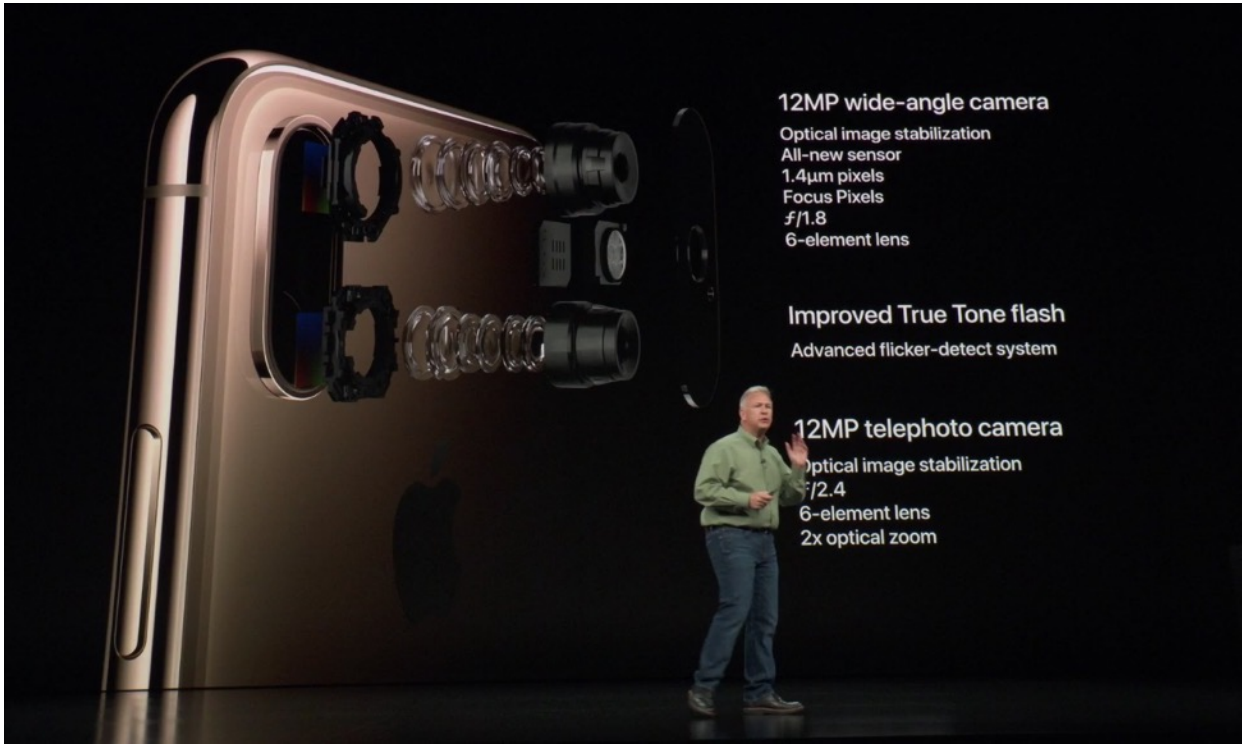


So what *is* an image?

For the purposes of this class, we will most often think of images as a grid of numbers:



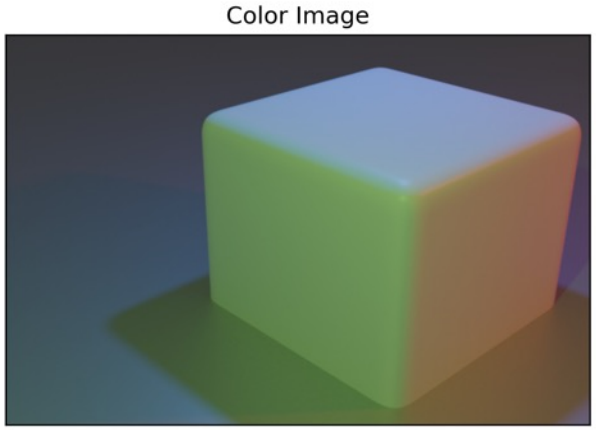
Pixel arrays are relatively easy to obtain

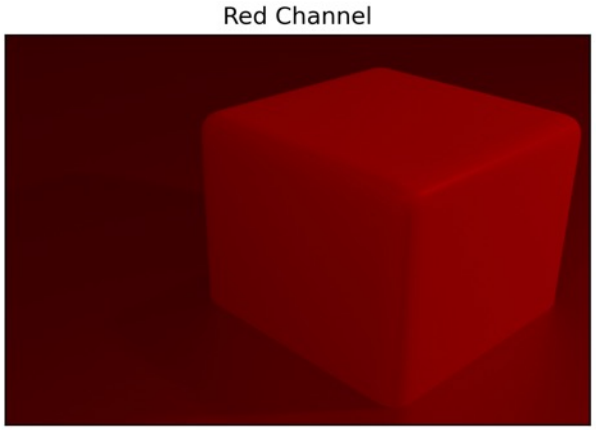


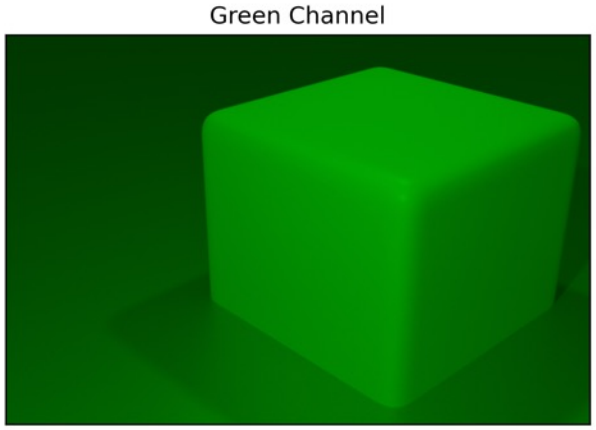
We will discuss more about how cameras work later on in the class: understanding how the lens of a camera images the world will be fundamental to inverting that operation for scene understanding.

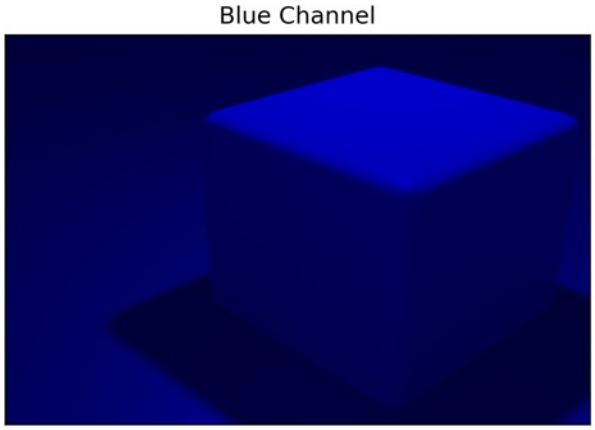
Image from Apple/TechCrunch

Color images have multiple *color channels*







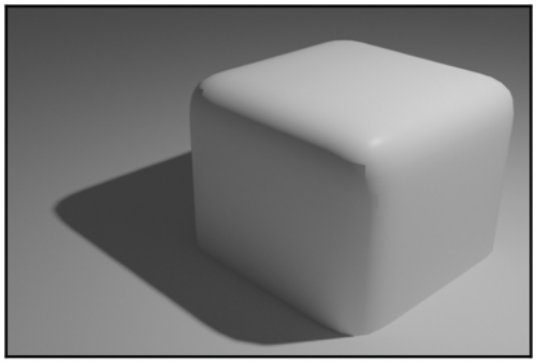



```
red_channel = image[:, :, 0]
```

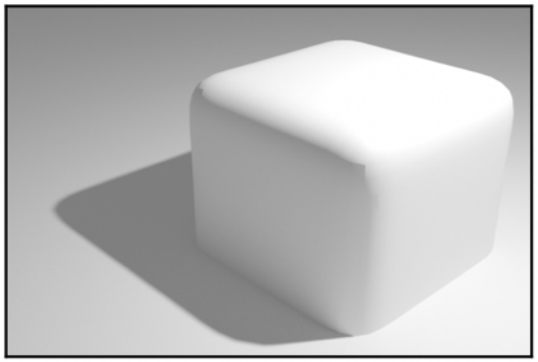
It is also helpful to think of an image as a *function* that returns intensity values

Like any other function, we can apply transformations to modify the outputs, or modify the inputs to effectively change its domain:

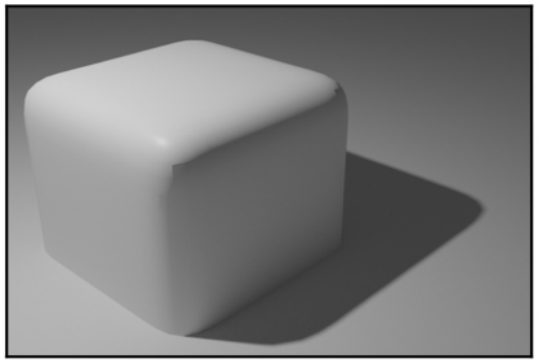
f(x, y)



f(x, y) + 64



f(-x, y)



The function f(x,y) represents the intensity at point (x,y). The digital image I is simply a discrete version of this function.

Definitions of some key terms

- *digital image* A rectangular array I of *pixels*
- *pixel* A pixel (x, y, u) is an element of the image array; it combines a sample u and a location $p = (x, y) \in \mathbb{Z}^2$, where $\mathbb{Z}$ is the set of integers, and $u \in [0, 1]$ (or $\in \{0, \cdots, 255\}$.

Computing image statistics

$$\text{Mean}(I) = \frac{1}{|\Omega|} \sum_{(x,y)\in\Omega} I(x,y) = \frac{1}{N_{\text{cols}} N_{\text{rows}}} \sum_{(x,y)\in\Omega} I(x,y)$$

or, if you want to use Python:

```
n = img.shape[0] * img.shape[1]
mean = np.sum(img)/n
var = np.sum((img - mean) ** 2)/n
print(f"mean={mean}\nvar={var}")
```

```
mean=0.4819119934640523
var=0.026983677793744925
```

What are the different ways in which we might manipulate images?

Much of this course will be devoted to manipulating images so we can understand them computationally.

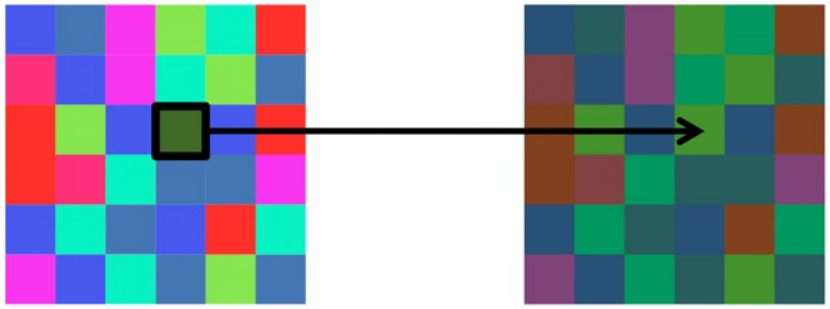
Point Processing

Warping

Filtering

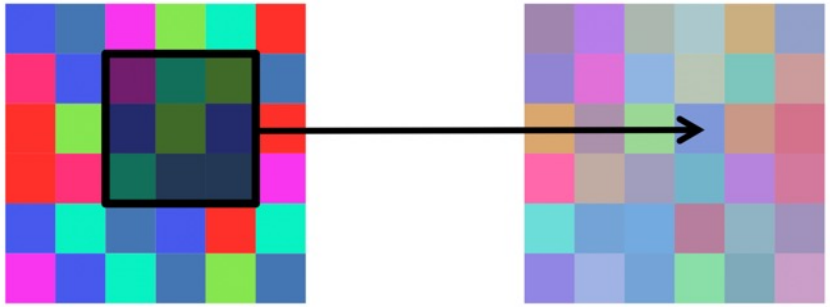
Point Processing versus Filtering

Point Operation



point processing

Neighborhood Operation

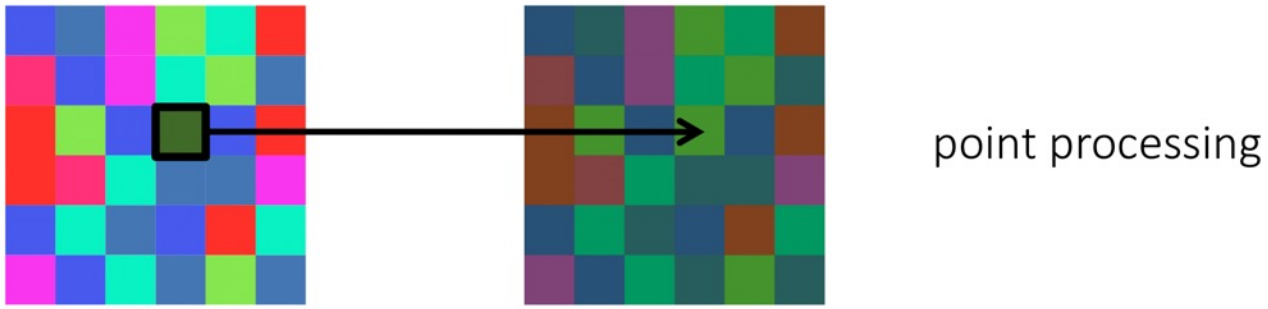


“filtering”

Point processing transforms single pixels.

Filtering is broader, and relies on context.

Point Processing takes in a single pixel as input and outputs another pixel or value



1 pixel in. 1 pixel out.

Examples of Point Processing

Can you identify these functions?

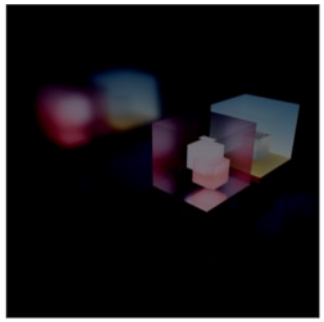
Base Image

$f(x, y)$

Examples of Point Processing

Can you identify these functions?

Base Image



$f(x, y)$

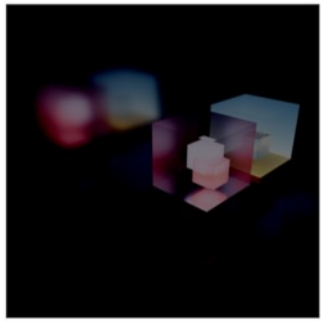
Examples of Point Processing

Can you identify these functions?

Base Image

$f(x, y)$

Darken



$f(x, y) - 0.5$

Examples of Point Processing

Can you identify these functions?

Base Image

$f(x, y)$

Darken

$f(x, y) - 0.5$

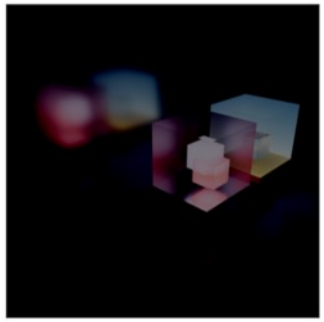
Examples of Point Processing

Can you identify these functions?

Base Image

$f(x, y)$

Darken



$f(x, y) - 0.5$

Lower Contrast

$f(x, y)/2$

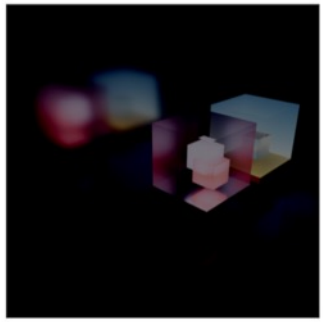
Examples of Point Processing

Can you identify these functions?

Base Image

$f(x, y)$

Darken



$f(x, y) - 0.5$

Lower Contrast

$f(x, y)/2$

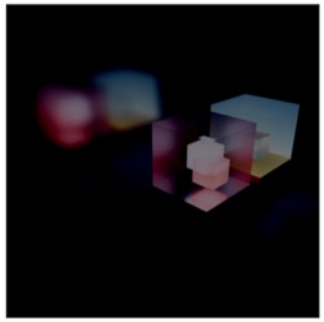
Examples of Point Processing

Can you identify these functions?

Base Image

$f(x, y)$

Darken



$f(x, y) - 0.5$

Lower Contrast

$f(x, y)/2$

Lower Contrast (nonlinear)

$f(x, y)^{1/3}$

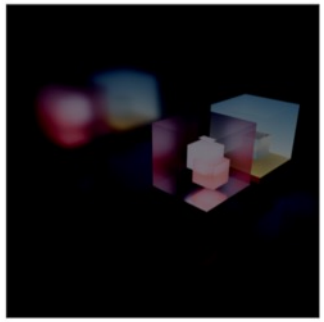
Examples of Point Processing

Can you identify these functions?

Base Image

$f(x, y)$

Darken



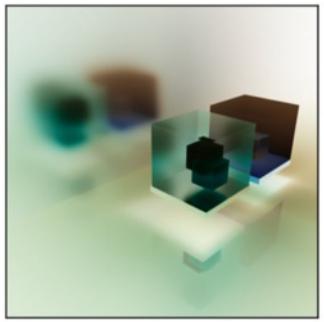
$f(x, y) - 0.5$

Lower Contrast

$f(x, y)/2$

Lower Contrast (nonlinear)

$f(x, y)^{1/3}$



Examples of Point Processing
Can you identify these functions?

Base Image

$f(x, y)$

Darken

$f(x, y) - 0.5$

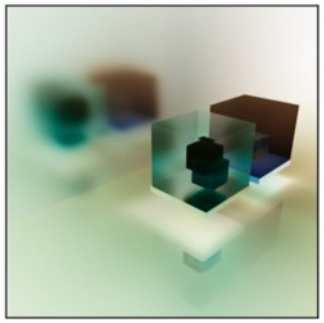
Lower Contrast

$f(x, y)/2$

Lower Contrast (nonlinear)

$f(x, y)^{1/3}$

Invert



$1 - f(x, y)$

Examples of Point Processing
Can you identify these functions?

Base Image

$f(x, y)$

Darken

$f(x, y) - 0.5$

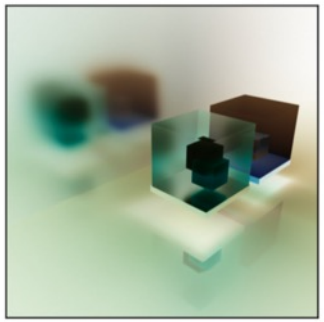
Lower Contrast

$f(x, y)/2$

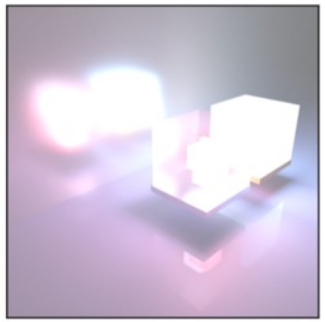
Lower Contrast (nonlinear)

$f(x, y)^{1/3}$

Invert



$1 - f(x, y)$



Examples of Point Processing
Can you identify these functions?

Base Image

$f(x, y)$

Darken

$f(x, y) - 0.5$

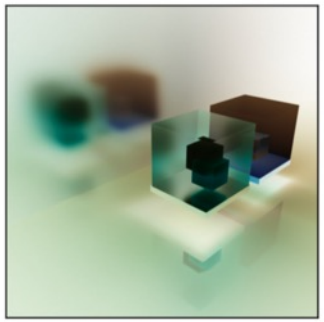
Lower Contrast

$f(x, y)/2$

Lower Contrast (nonlinear)

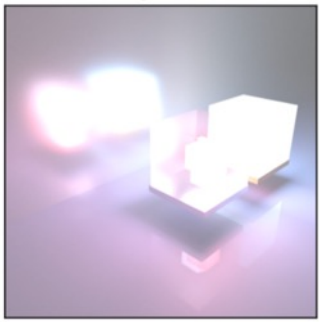
$f(x, y)^{1/3}$

Invert



$1 - f(x, y)$

Lighten



$f(x, y) - 0.5$

Examples of Point Processing
Can you identify these functions?

Base Image

$f(x, y)$

Darken

$f(x, y) - 0.5$

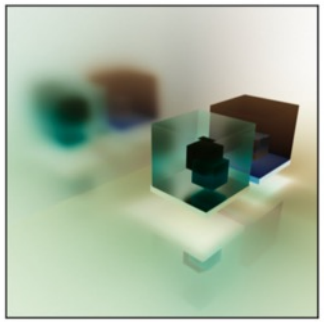
Lower Contrast

$f(x, y)/2$

Lower Contrast (nonlinear)

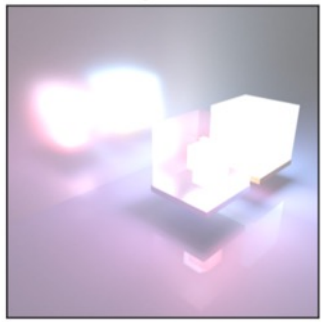
$f(x, y)^{1/3}$

Invert

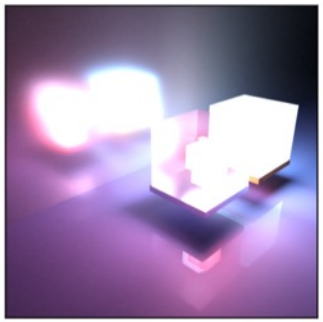


$1 - f(x, y)$

Lighten



$f(x, y) - 0.5$



Examples of Point Processing
Can you identify these functions?

Base Image

$f(x, y)$

Darken

$f(x, y) - 0.5$

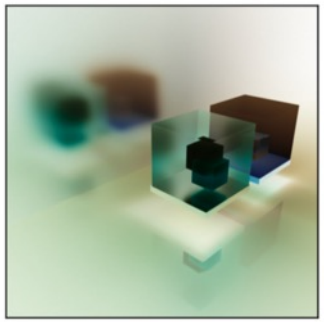
Lower Contrast

$f(x, y)/2$

Lower Contrast (nonlinear)

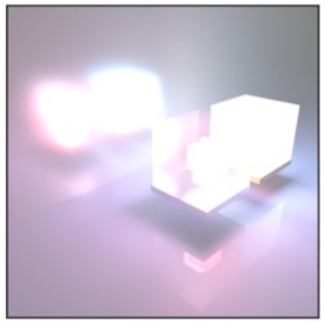
$f(x, y)^{1/3}$

Invert



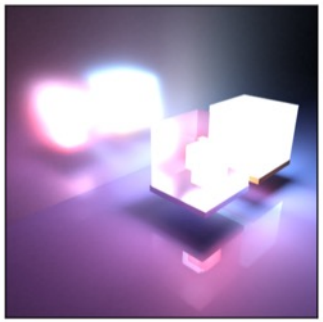
$1 - f(x, y)$

Lighten



$f(x, y) - 0.5$

Raise Contrast



$f(x, y) * 2$

Examples of Point Processing
Can you identify these functions?

Base Image

$f(x, y)$

Darken

$f(x, y) - 0.5$

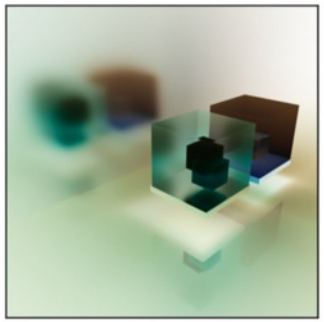
Lower Contrast

$f(x, y)/2$

Lower Contrast (nonlinear)

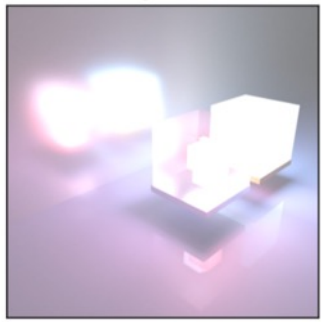
$f(x, y)^{1/3}$

Invert



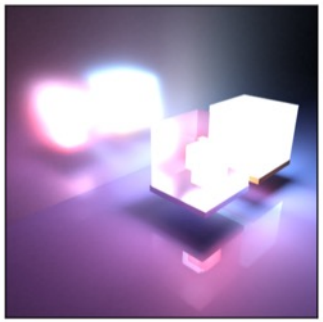
$1 - f(x, y)$

Lighten

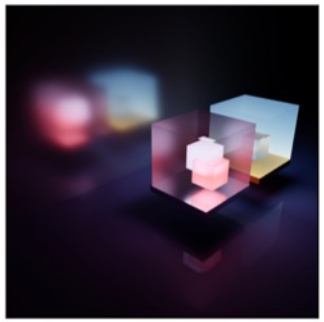


$f(x, y) - 0.5$

Raise Contrast



$f(x, y) * 2$



Examples of Point Processing

Can you identify these functions?

Base Image

$f(x, y)$

Darken

$f(x, y) - 0.5$

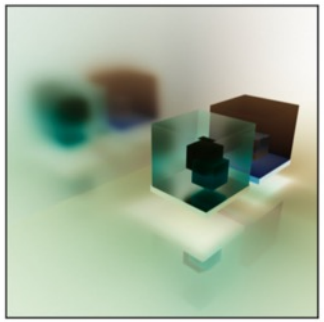
Lower Contrast

$f(x, y)/2$

Lower Contrast (nonlinear)

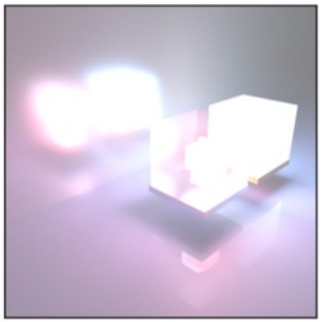
$f(x, y)^{1/3}$

Invert



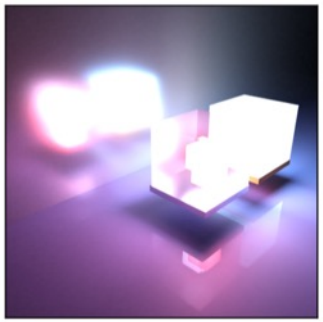
$1 - f(x, y)$

Lighten



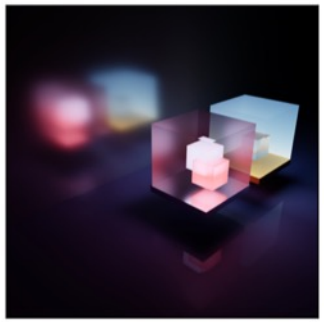
$f(x, y) - 0.5$

Raise Contrast



$f(x, y) * 2$

Raise Contrast (nonlinear)



$f(x, y)^2$

Other examples of point processing

Image from Bae et al.

(c) direct histogram transfer

(d) our result

Other examples of point processing

Image from Bae et al.

(a) input image (1200x900)

(b) our result

Other examples of point processing

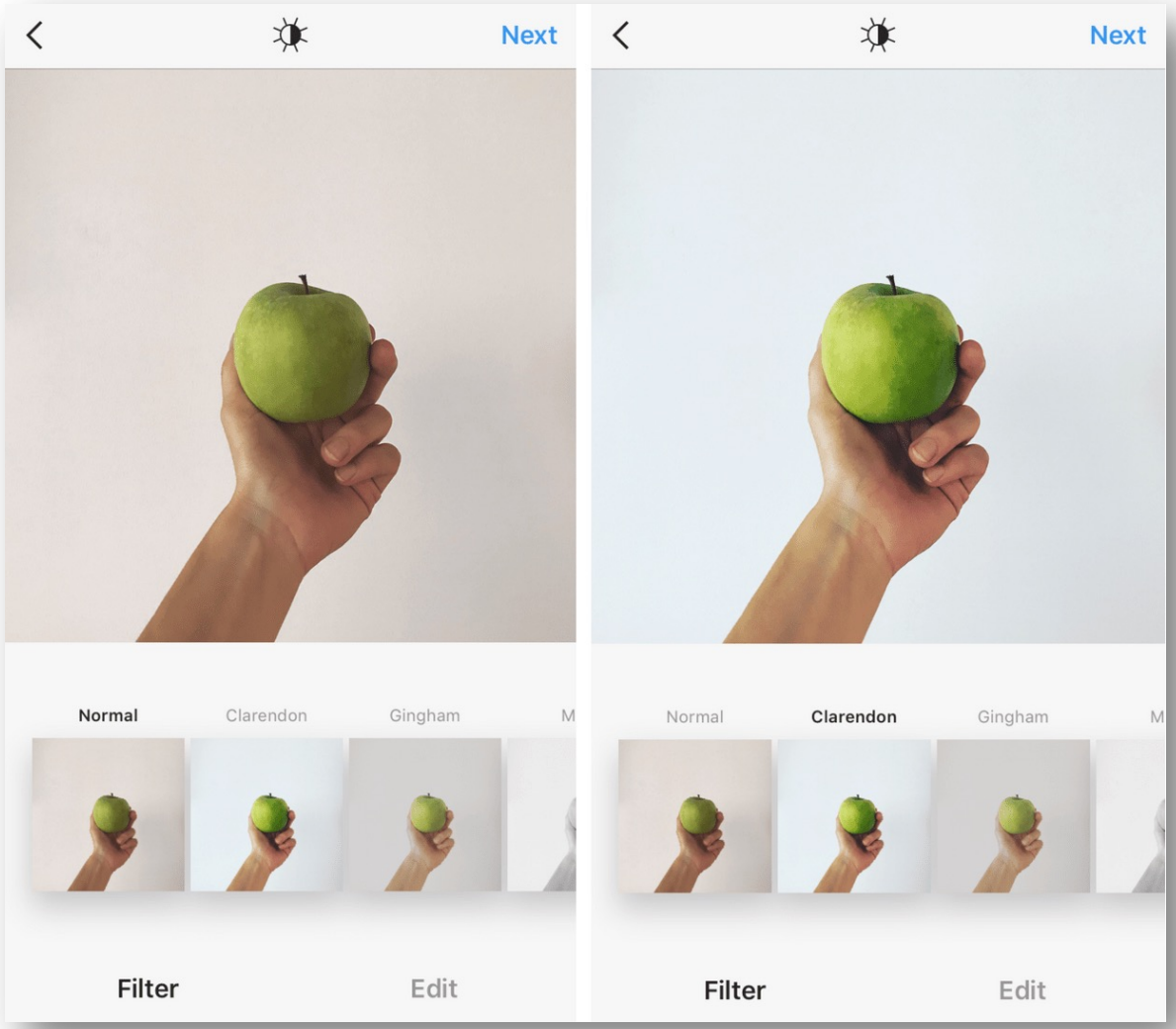


Image from blog.iconosquare (Instagram)

Aside: I care *a lot* about assumptions

Our assumptions reflect judgments about the way we think the world works.

Sometimes those assumptions are good, and help us accomplish our goals. Sometimes they limit us, and we need to develop new tools and ideas that reflect that the world is more complicated.

What assumptions do we make when point processing?

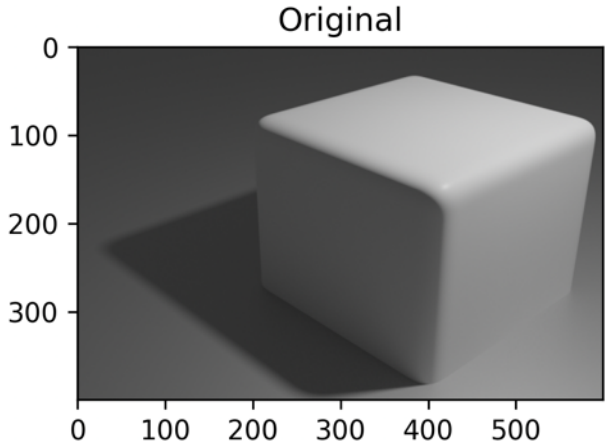
Thoughts?

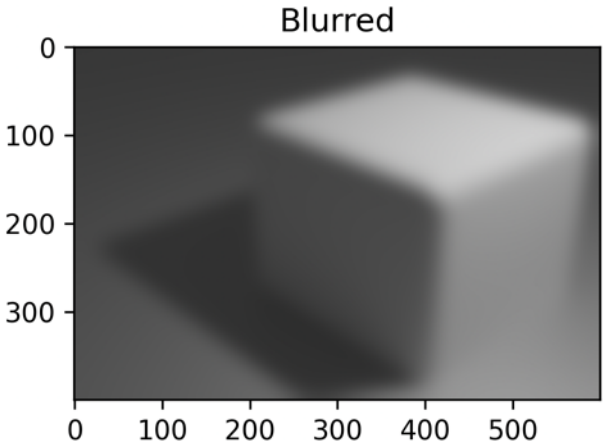
What assumptions do we make when point processing?

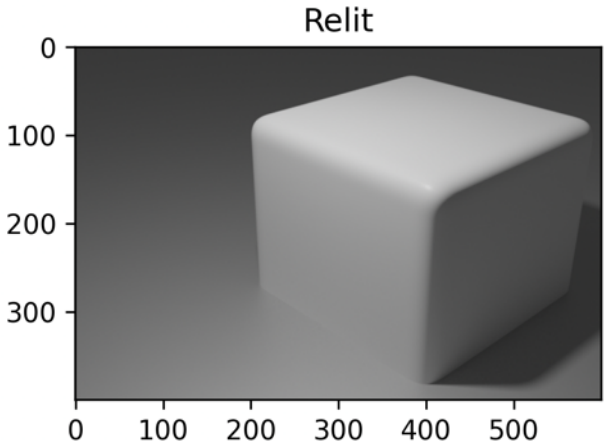
That an individual pixel is enough information to manipulate the images; that the pixel's context is unimportant.

What assumptions do we make when point processing?

That an individual pixel is enough information to manipulate the images; that the pixel's context is unimportant.







Simple pixel operations cannot map between these images: we need more context, more understanding.

Now let's look at *filtering*, which can involve multiple input pixels.

Perhaps the simplest "filter" is a *mean filter* (or *box filter*).

Mean Filter: We want to process an image such that each pixel in the output image is the average of the surrounding pixels in the input image.

Now let's look at *filtering*, which can involve multiple input pixels.

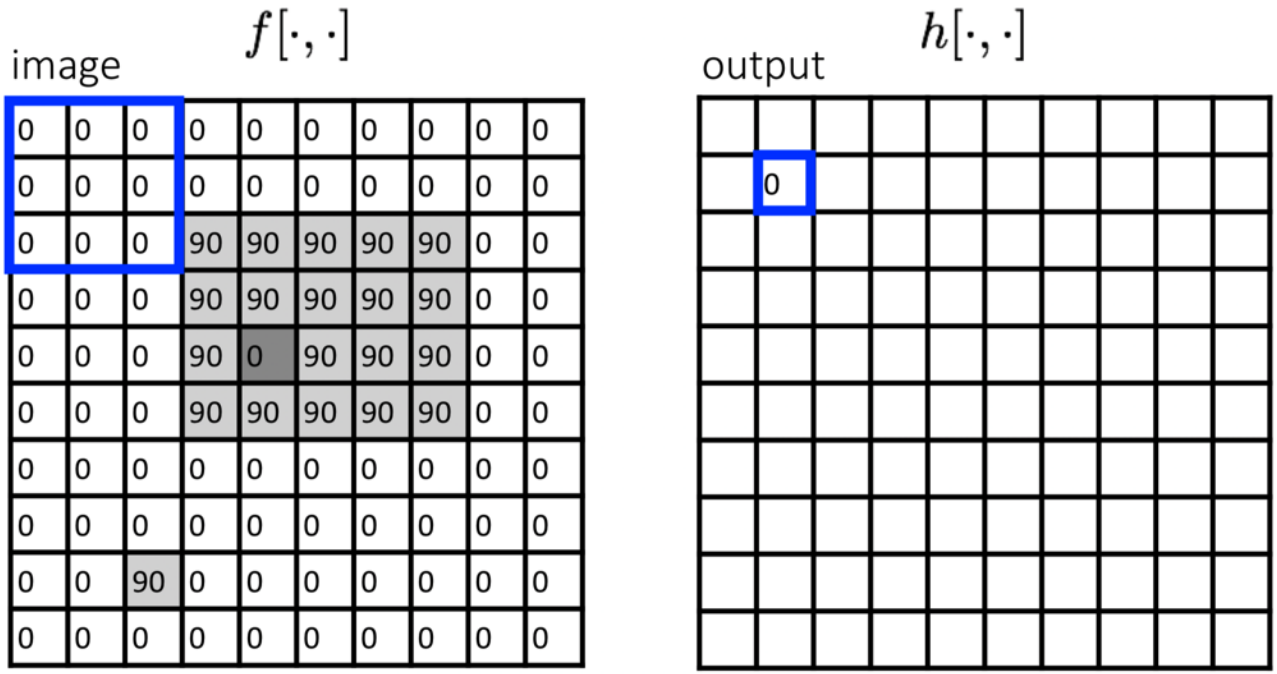
Perhaps the simplest "filter" is a *mean filter* (or *box filter*).

Mean Filter: We want to process an image such that each pixel in the output image is the average of the surrounding pixels in the input image.

What does this look like in practice? Let's try an example

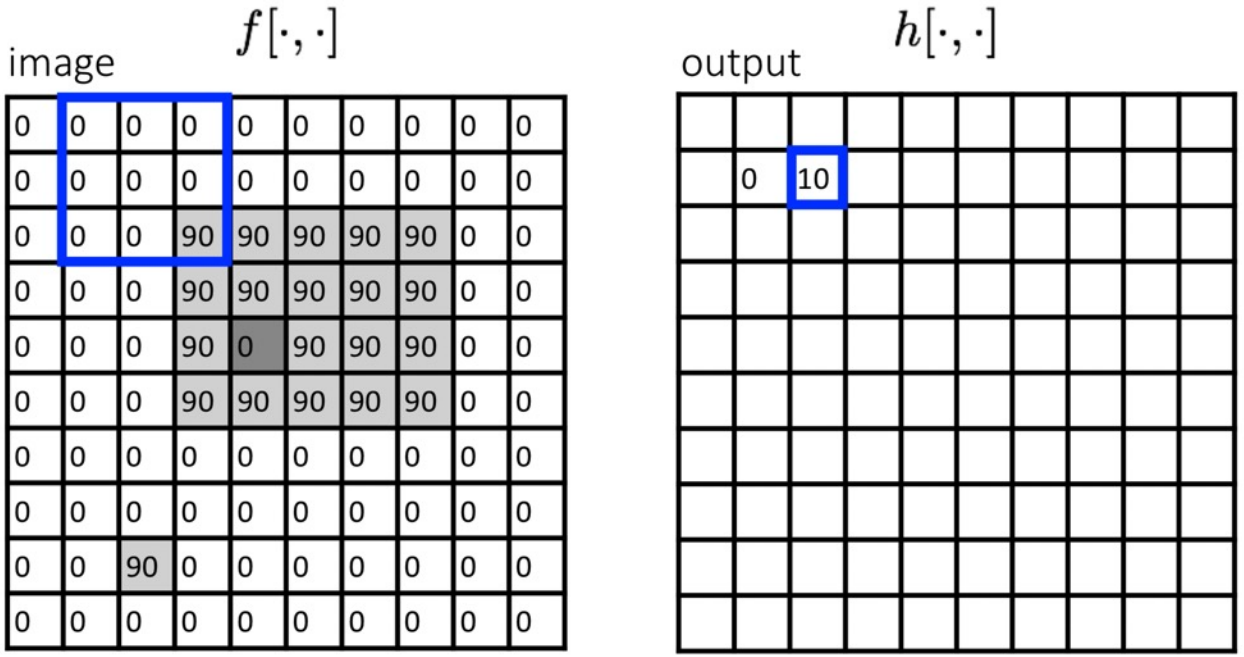
Box Filter Worked Example

Mean Filter / Box Filter : We take a 3x3 window of pixels and average those values to get the value of the pixel at the output:



Box Filter Worked Example

Mean Filter / Box Filter : We take a 3x3 window of pixels and average those values to get the value of the pixel at the output:



Box Filter Worked Example

Mean Filter / Box Filter : We take a 3x3 window of pixels and average those values to get the value of the pixel at the output:

image $f[\cdot,\cdot]$

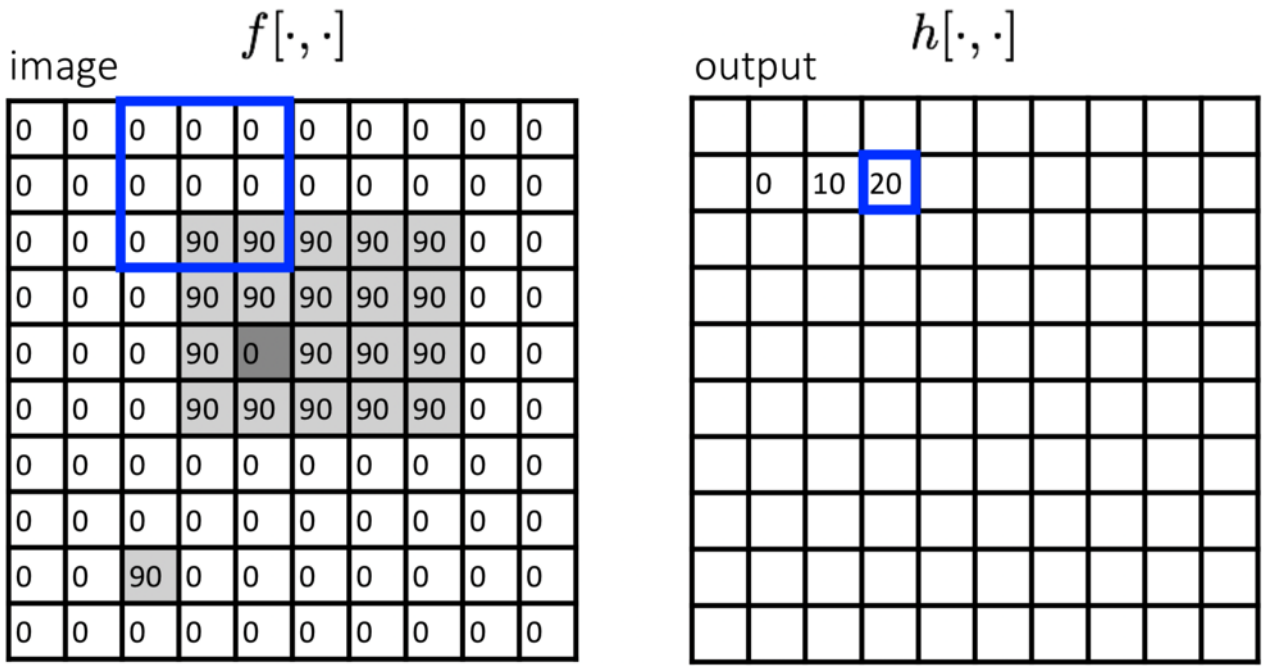
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

output $h[\cdot,\cdot]$

	0	10							

Box Filter Worked Example

Mean Filter / Box Filter : We take a 3x3 window of pixels and average those values to get the value of the pixel at the output:



Box Filter Worked Example

Mean Filter / Box Filter : We take a 3x3 window of pixels and average those values to get the value of the pixel at the output:

image $f[\cdot, \cdot]$

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

output $h[\cdot, \cdot]$

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

How might we implement the box filter in Python code?

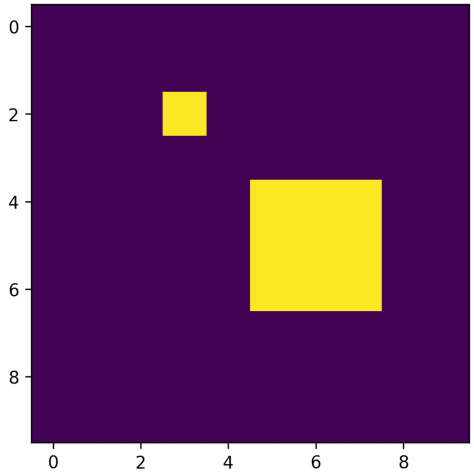
How might we implement the box filter in Python code?

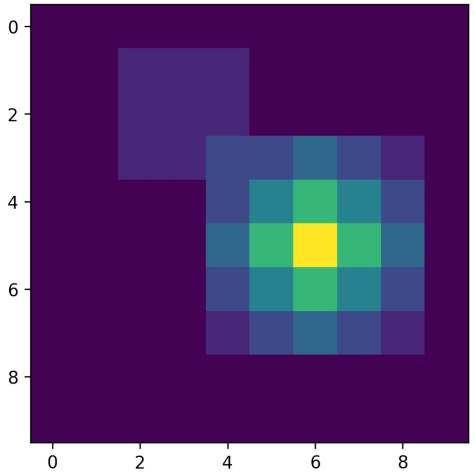
Easy enough!

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

N = 10
data = np.zeros((N, N))
data[2, 3] = 1.0
data[4:7, 5:8] = 1.0

data_filtered = np.zeros_like(data)
for ii in range(1, N-1):
    for jj in range(1, N-1):
        data_filtered[ii, jj] = data[ii-1:ii+2, jj-1:jj+2].sum()/9
```





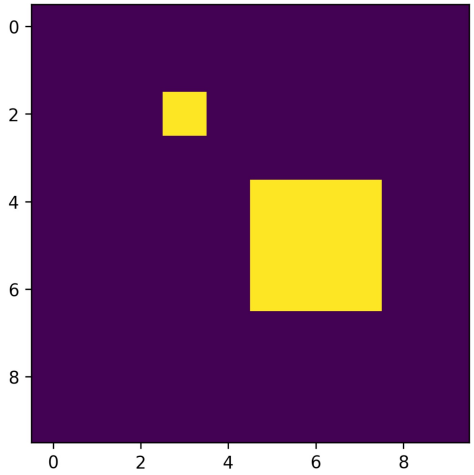
How might we implement the box filter in Python code?

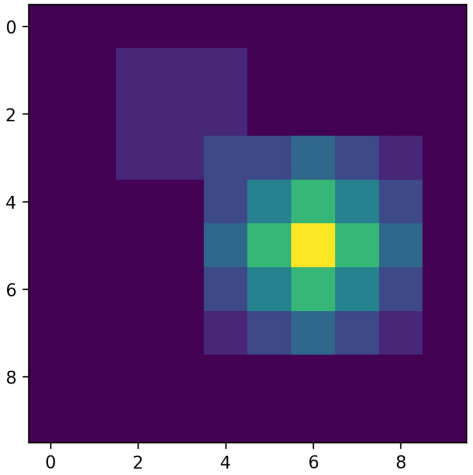
Easy enough!

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

N = 10
data = np.zeros((N, N))
data[2, 3] = 1.0
data[4:7, 5:8] = 1.0

data_filtered = np.zeros_like(data)
for ii in range(1, N-1):
    for jj in range(1, N-1):
        data_filtered[ii, jj] = data[ii-1:ii+2, jj-1:jj+2].sum()/9
```





Claim: The box filter (mean filter) is really just a simple example of a more general class of function.

How might we implement the box filter in Python code?

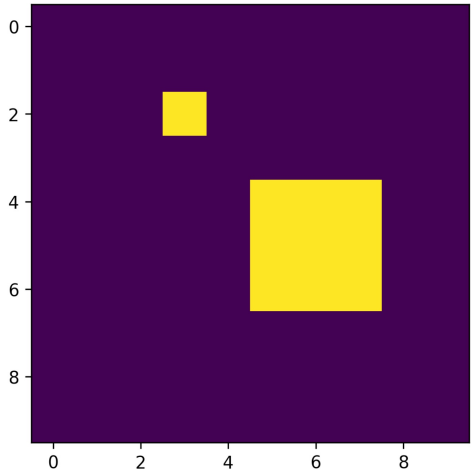
Easy enough!

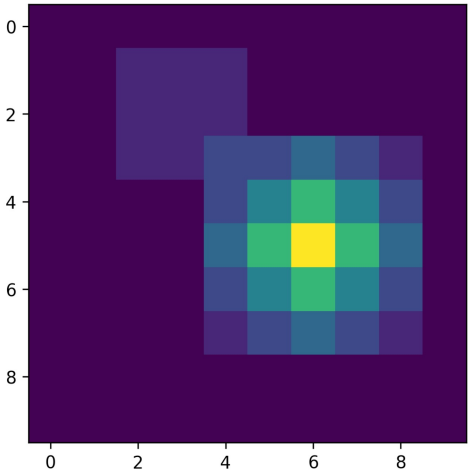
```python
import numpy as np
import matplotlib.pyplot as plt

N = 10
data = np.zeros((N, N))
data[2, 3] = 1.0
data[4:7, 5:8] = 1.0

data_filtered = np.zeros_like(data)
for ii in range(1, N-1):
    for jj in range(1, N-1):
        data_filtered[ii, jj] = data[ii-1:ii+2, jj-1:jj+2].sum()/9
```

Can we replace this with something a bit more general?





Claim: The box filter (mean filter) is really just a simple example of a more general class of function.

Who remembers the dot product?

Element-wise multiplication followed by a sum.

Who remembers the dot product?

Element-wise multiplication followed by a sum.

For vector data:

$$\begin{bmatrix} a & b & c \end{bmatrix} \text{ and } \begin{bmatrix} A & B & C \end{bmatrix} \text{ become } aA + bB + cC$$

For matrix-shaped data:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \text{ and } \begin{bmatrix} A & B & C \\ D & E & F \end{bmatrix} \text{ become } aA + bB + cC + dD + eE + fF$$

Who remembers the dot product?

Element-wise multiplication followed by a sum.

We can represent the mean filter using a similar sort of operation.

Who remembers the dot product?

Element-wise multiplication followed by a sum.

We can represent the mean filter using a similar sort of operation. We can use this matrix:

$$\text{kernel} = g[\cdot, \cdot] = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

How is this code different from before?

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

N = 10
data = np.zeros((N, N))
data[2, 3] = 1.0
data[4:7, 5:8] = 1.0

data_filtered = np.zeros_like(data)
box_kernel = np.ones((3, 3))/9.0
for ii in range(1, N-1):
    for jj in range(1, N-1):
        data_filtered[ii, jj] = (
          box_kernel * data[ii-1:ii+2, jj-1:jj+2]).sum()
```

Does it produce the same result?

Old Version:

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

N = 10
data = np.zeros((N, N))
data[2, 3] = 1.0
data[4:7, 5:8] = 1.0

data_filtered = np.zeros_like(data)
for ii in range(1, N-1):
    for jj in range(1, N-1):
        data_filtered[ii, jj] = data[ii-1:ii+2, jj-1:jj+2].sum()/9
```

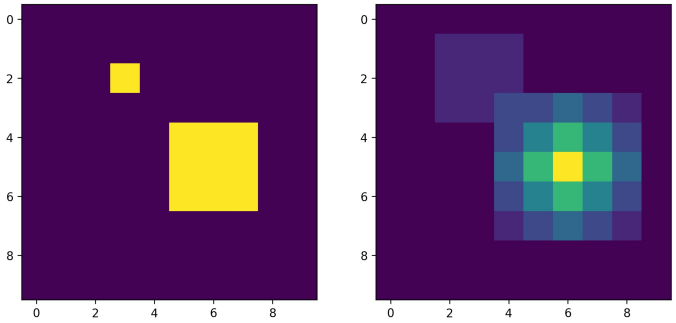
How is this code different from before?

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

N = 10
data = np.zeros((N, N))
data[2, 3] = 1.0
data[4:7, 5:8] = 1.0

data_filtered = np.zeros_like(data)
box_kernel = np.ones((3, 3))/9.0
for ii in range(1, N-1):
    for jj in range(1, N-1):
        data_filtered[ii, jj] = (
          box_kernel * data[ii-1:ii+2, jj-1:jj+2]).sum()
```

Does it produce the same result? **Yes!**



So why does this matter?

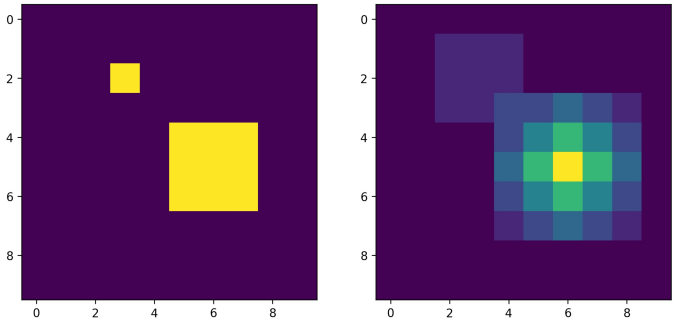
How is this code different from before?

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

N = 10
data = np.zeros((N, N))
data[2, 3] = 1.0
data[4:7, 5:8] = 1.0

data_filtered = np.zeros_like(data)
box_kernel = np.ones((3, 3))/9.0
for ii in range(1, N-1):
    for jj in range(1, N-1):
        data_filtered[ii, jj] = (
          box_kernel * data[ii-1:ii+2, jj-1:jj+2]).sum()
```

Does it produce the same result? **Yes!**



So why does this matter? **Now we have a new mathematical tool we can use!**

Box Filter Worked Example

$g[\cdot,\cdot]$

kernel

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

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0								

Box Filter Worked Example

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10							

Box Filter Worked Example

$g[\cdot,\cdot]$

kernel

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

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	0	10							

Box Filter Worked Example

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10	20						

Box Filter Worked Example

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

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

A mean filter is an example of a *linear shift independent (invariant) filter.*

The most common form of filtering is Linear Shift-Independent Filtering.

- Each pixel at the output is a linear combination of nearby pixels at the input.
- The **kernel** is the matrix that stores the weights for computing the linear combination.

The kernel is swept across the image to process all pixels at the output.

The Box Filter is perhaps the simplest smoothing operation

Perhaps the simplest (non-identity) kernel is the box filter:

$$\text{kernel} = g[\cdot,\cdot] = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Each pixel at the output is a local average of the pixels at the input. This filter has a smoothing effect.

The box filter is also known as the mean filter, the rect filter, and I'm sure there are more names.

The Box Filter is perhaps the simplest smoothing operation

Perhaps the simplest (non-identity) kernel is the box filter:

$$\text{kernel} = g[\cdot,\cdot] = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

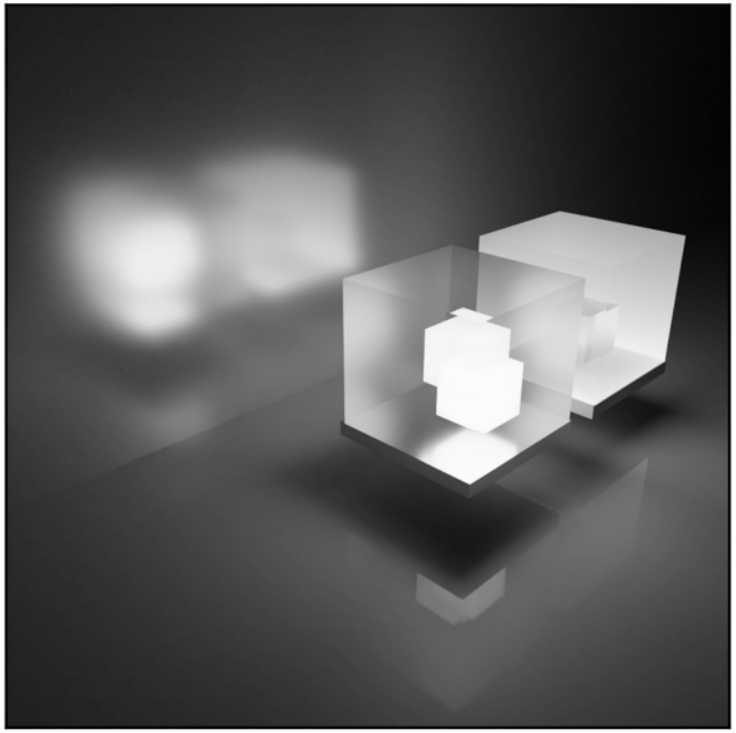
Why divide by 9?

Each pixel at the output is a local average of the pixels at the input. This filter has a smoothing effect.

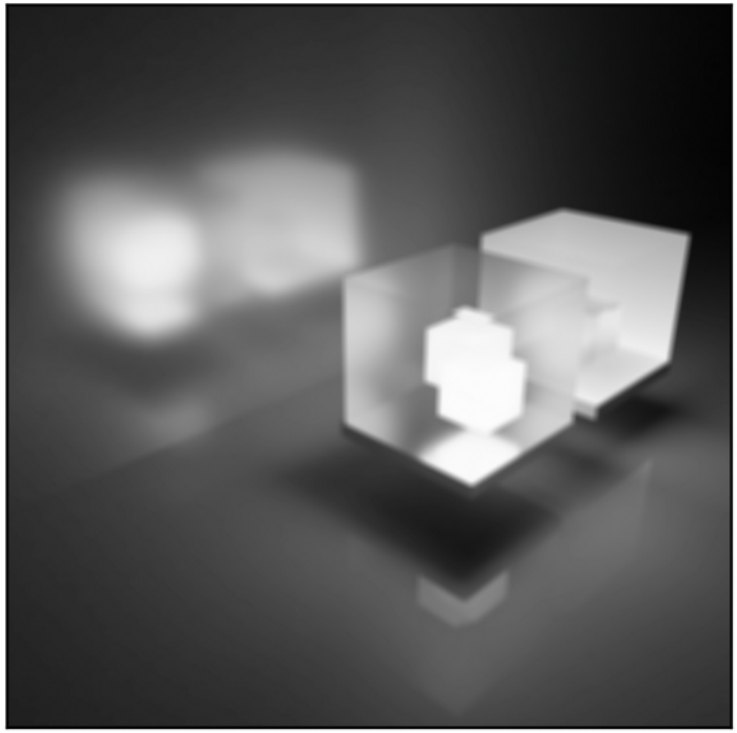
The box filter is also known as the mean filter, the rect filter, and I'm sure there are more names.

Box Filter Examples

Original

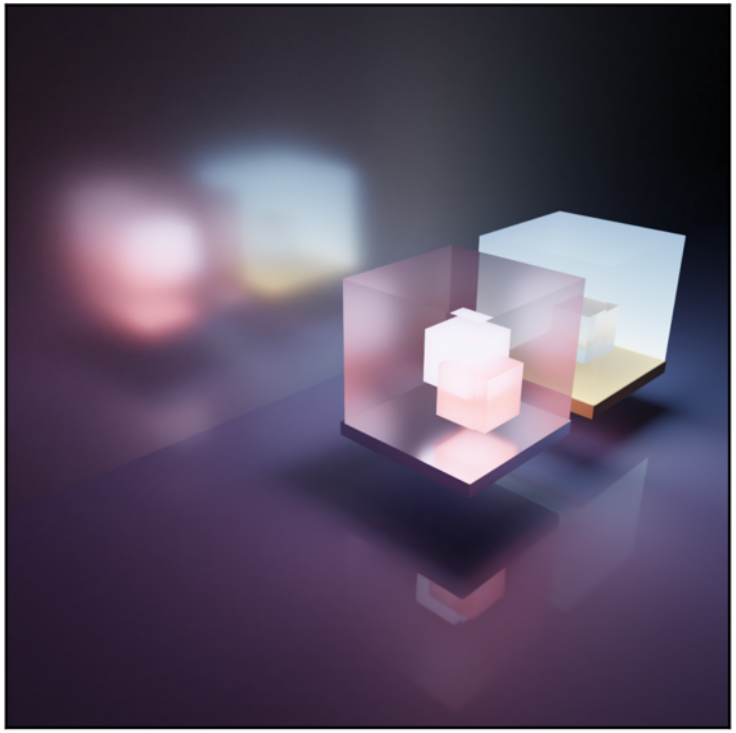


Box Filter



Box Filter Examples

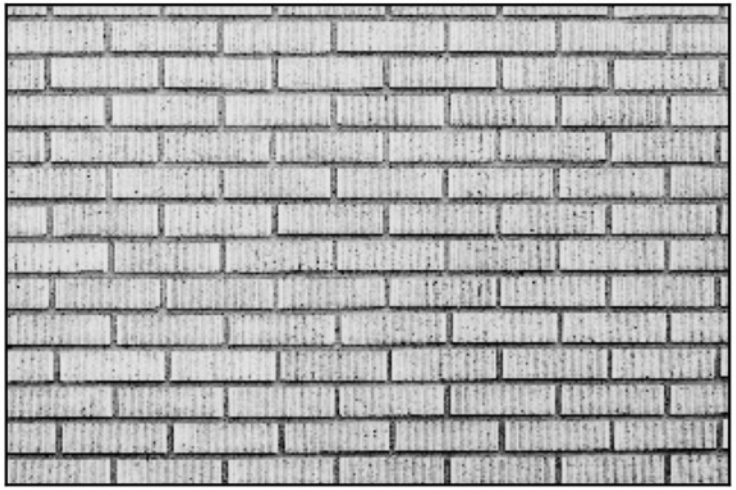
Original



Box Filter

Box Filter Examples

Original



Box Filter

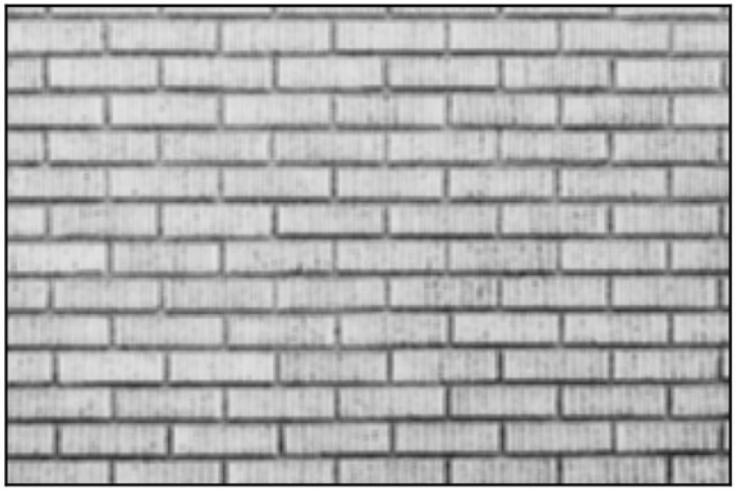


Image from Unsplash

Box Filter Examples

Original

Box Filter

Image from NASA (Margaret Hamilton)

Filtering: Mathematically, generally

If we have some kernel g, we can apply it to some input image f using the following operation:

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

This procedure is known as *the correlation*.

The Convolution: 1D Discrete

The *convolution* is defined as follows (note the negative sign):

$$(f * g)(x) = \sum_i f(i)g(x - i)$$

where f is the filter, g is the input signal, and f $*$ g is the filtered signal.

The 2D discrete convolution is defined similarly

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

The 2D discrete convolution is defined similarly

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

This is the same (or at least similar) formula we used to apply the discrete box filter earlier.

The Box Filter is a *separable* filter

Separable filters are the outer product of two 1-D filters:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix} \cdot \frac{1}{3}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}$$

Why would this matter?

The Box Filter is a *separable* filter

Separable filters are the outer product of two 1-D filters:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix} \cdot \frac{1}{3}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}$$

Why would this matter?

It is faster to evaluate two 1D filters than one 2D filter. If you have an M × M with an N × N filter.

Operations of a non-separable filter: N^2M^2

Operations of a separable filter: ??

The Box Filter is a *separable* filter

Separable filters are the outer product of two 1-D filters:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix} \cdot \frac{1}{3}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}$$

Why would this matter?

It is faster to evaluate two 1D filters than one 2D filter. If you have an M × M with an N × N filter.

Operations of a non-separable filter: N^2M^2

Operations of a separable filter: $2NM^2$

More Filters

What will this filter do?

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

More Filters

What will this filter do?

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

The Identity Filter: no change!

More Filters

What will this filter do?

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

More Filters

What will this filter do?

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

The Right Shift Filter: the image is moved over by a single pixel.

Why is the shift to the right? (Think about convolution versus correlation)

More Filters

What will this filter do?

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

More Filters

What will this filter do?

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

A Sharpening Filter: accentuates edges, yet leaves flat regions unchanged.

Sharpening Examples

Sharpening Examples (Oversharpening)

original sharpened oversharpened

More Filters

What will this filter do?

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

More Filters

What will this filter do?

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

The Gaussian Blur Filter: a Gaussian is used to smooth the function.

$$f(i,j) \propto \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)$$

Is the Gaussian Blur separable?

More Filters

What will this filter do?

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

The Gaussian Blur Filter: a Gaussian is used to smooth the function.

$$f(i,j) \propto \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)$$

Is the Gaussian Blur separable? Yes!

More Filters

What will this filter do?

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

The Gaussian Blur Filter: a Gaussian is used to smooth the function.

$$f(i,j) \propto \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)$$

Is the Gaussian Blur separable? Yes!

[How can you tell if a filter is separable?]

More Filters

What will this filter do?

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

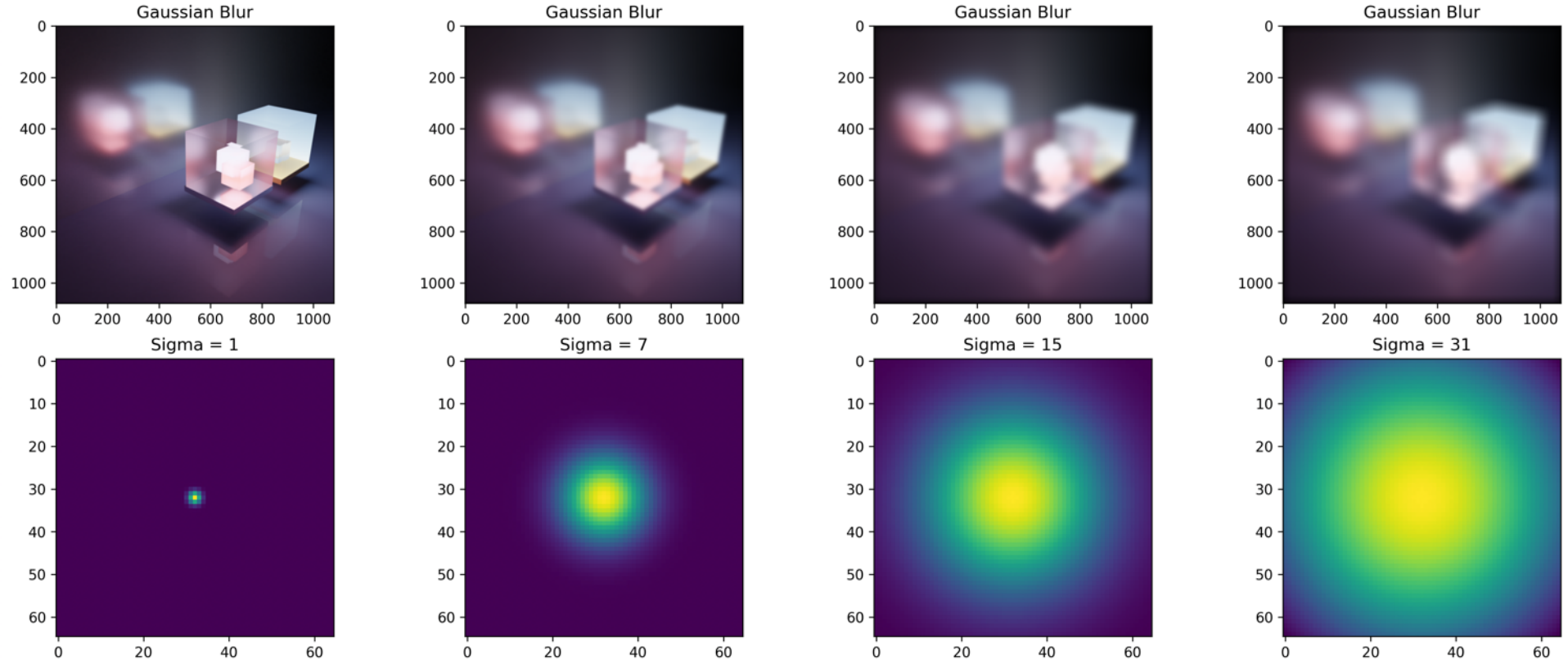
The Gaussian Blur Filter: a Gaussian is used to smooth the function.

$$f(i,j) \propto \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)$$

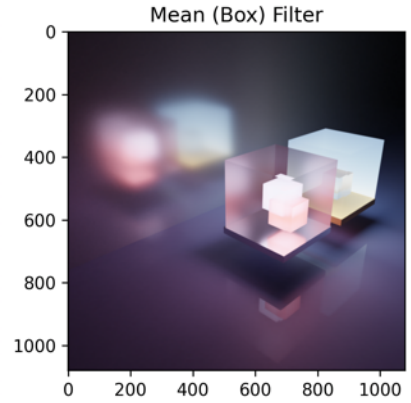
Is the Gaussian Blur separable? Yes!

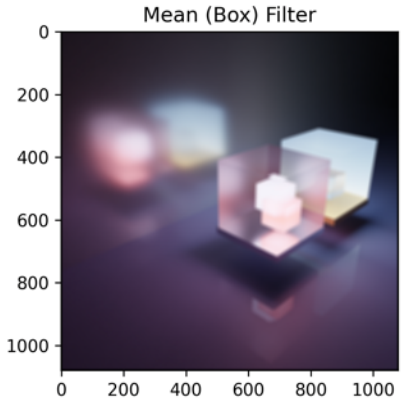
[How can you tell if a filter is separable? Matrix rank == 1]

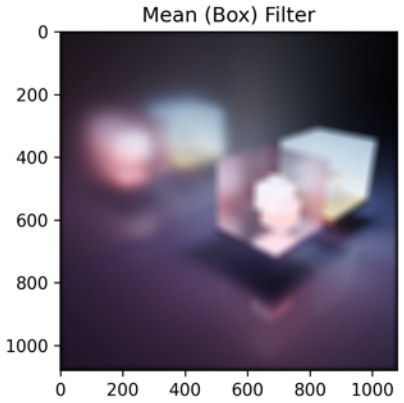
Varying the size of a Gaussian Blur

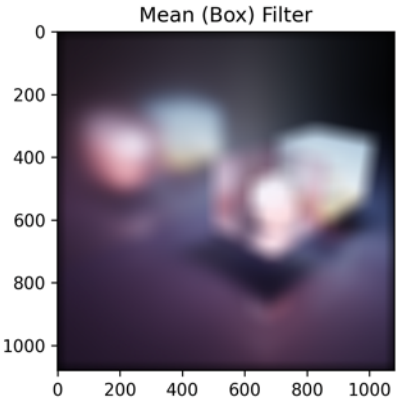


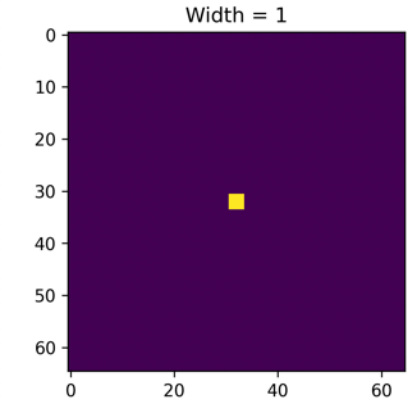
Varying the size of a Mean Filter

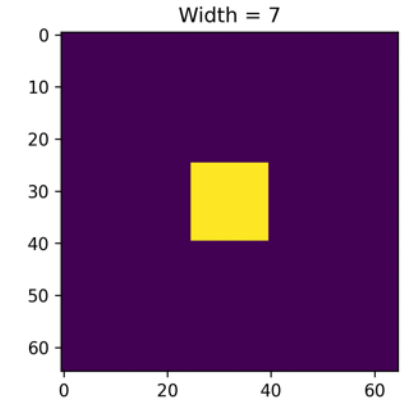


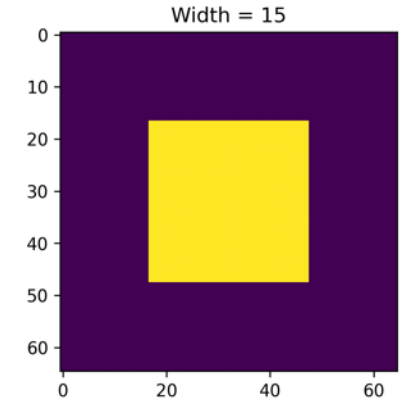


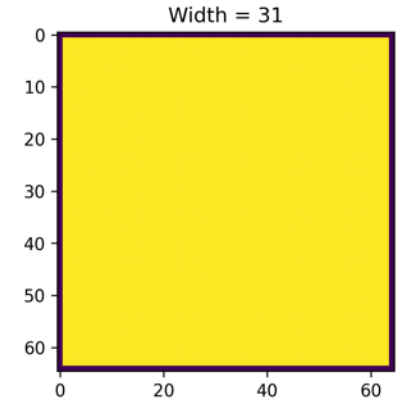








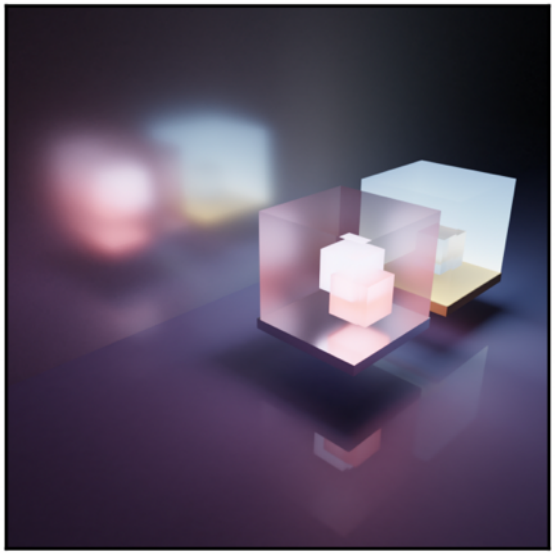




Mean Filter vs Gaussian Blur

Which of these looks “more appealing”:

Original

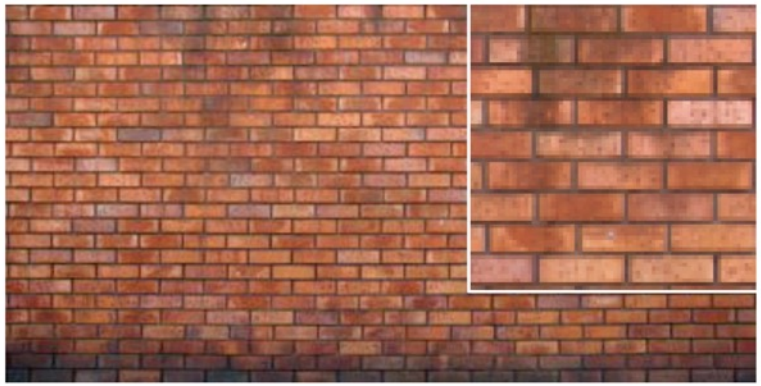


Gaussian

Box

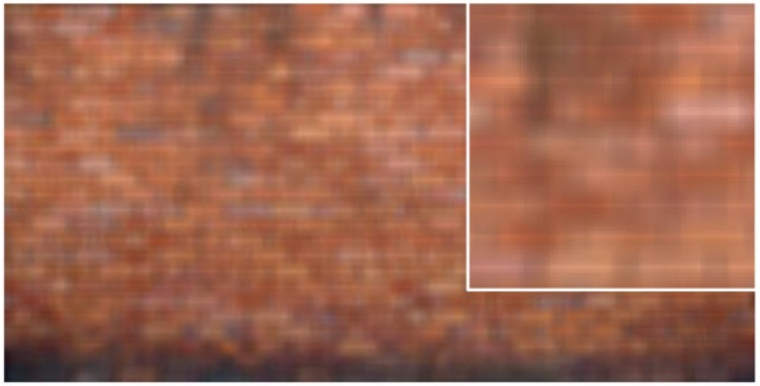
Mean Filter vs Gaussian Blur

Which of these looks “more appealing”:



original

7x7 Gaussian



7x7 box

What assumptions do we make when filtering an image?

What assumptions do we make when filtering an image?

We assume that the location of an object in an image does not determine how we process it.

What assumptions do we make when filtering an image?

We assume that the location of an object in an image does not determine how we process it.

Image from Wired

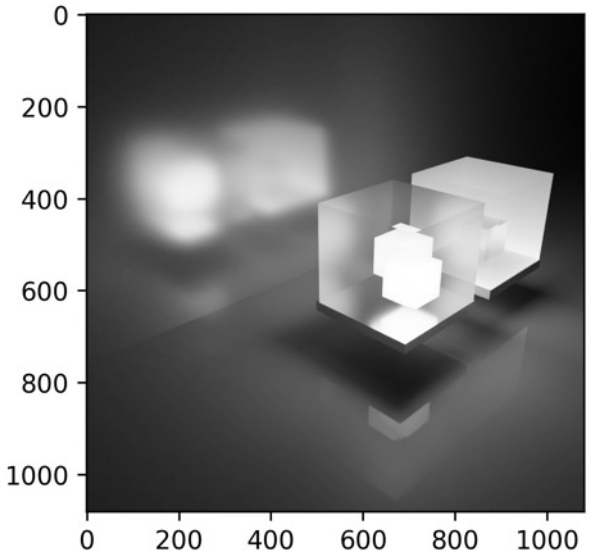
What assumptions do we make when filtering an image?

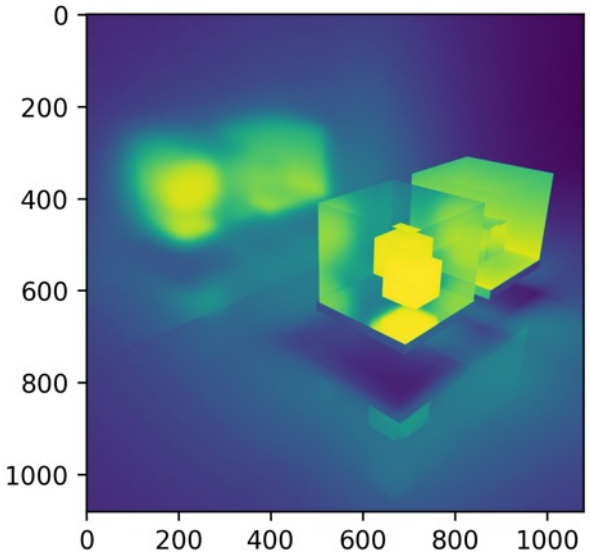
We assume that the location of an object in an image does not determine how we process it.

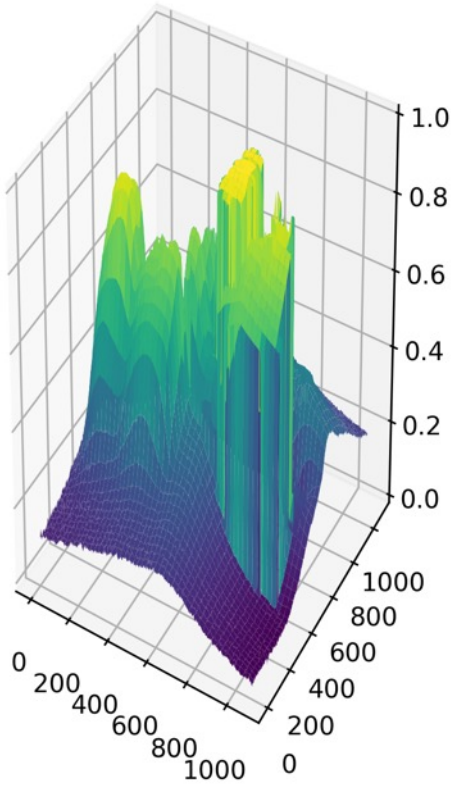
> *We can do more with filters than with point processing, but more parameters means more choices to make.*

Later on in the course, we will look at how learning can be used to make some of these choices for us from data.

How might we detect edges in an image?







The image is a function: let's take the derivative

How might we take the derivative of an image?

How would we take the derivative/gradient of any function?

How we normally take a derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

We can also use the central derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

How might we take the derivative of an image? Finite Differences.

For discrete signals, we approximate the derivative without the limit (finite difference approximation):

$$f'(x) \approx \frac{f(x+1) - f(x-1)}{2}$$

Can we represent this as a convolution filter?

How might we take the derivative of an image? Finite Differences.

For discrete signals, we approximate the derivative without the limit (finite difference approximation):

$$f'(x) \approx \frac{f(x+1) - f(x-1)}{2}$$

Can we represent this as a convolution filter?

Absolutely! Which of these is the 1D derivative? (Think of the flipped sign in the convolution.)

$$\frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \text{or} \quad \frac{1}{2}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

How might we take the derivative of an image? Finite Differences.

For discrete signals, we approximate the derivative without the limit (finite difference approximation):

$$f'(x) \approx \frac{f(x+1) - f(x-1)}{2}$$

Can we represent this as a convolution filter?

Absolutely! Which of these is the 1D derivative? (Think of the flipped sign in the convolution.)

$$\frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix}$$ The 1D derivative filter

The Sobel Filter is a common filter for computing image derivatives

$$\text{Horizontal Sobel Filter} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

$$\text{Vertical Sobel Filter} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

The Sobel Filter is a common filter for computing image derivatives

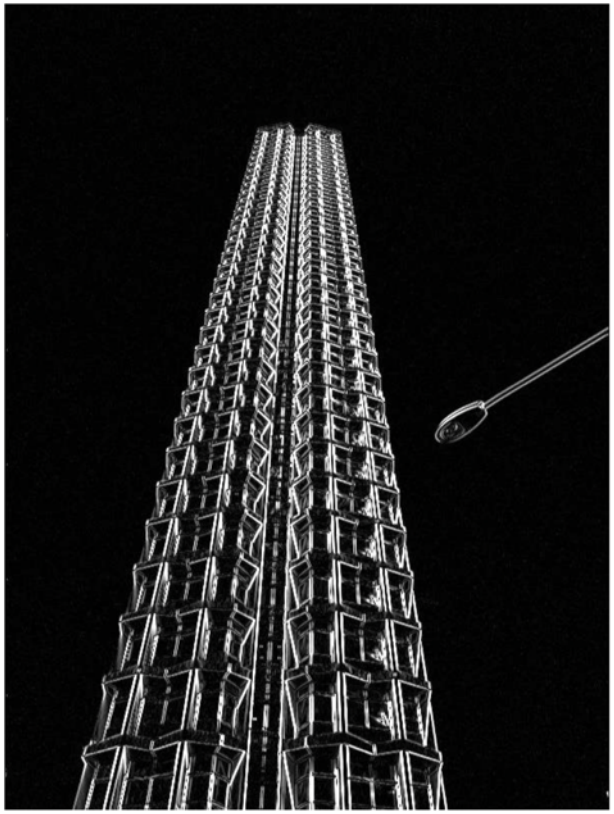
$$\text{Horizontal Sobel Filter} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

$$\text{Vertical Sobel Filter} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

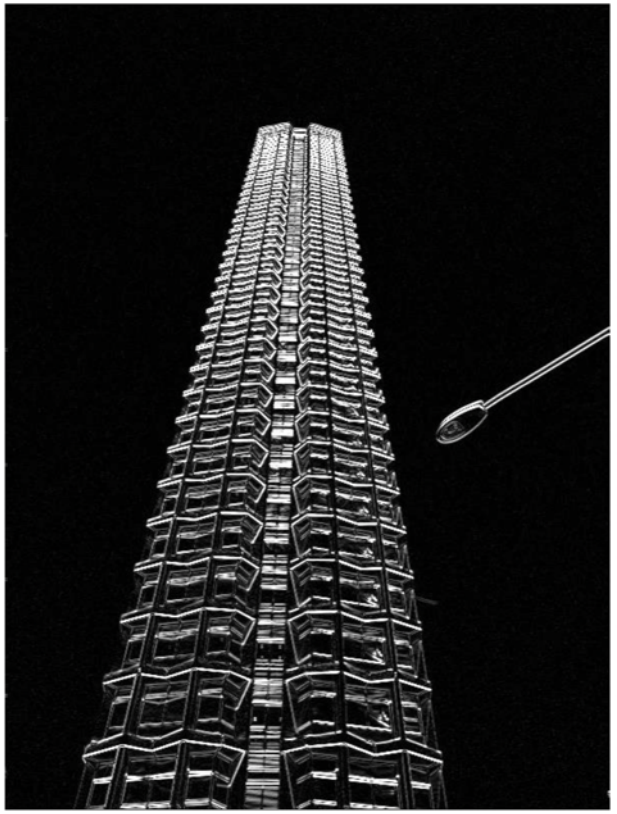
These filters combine blurring in one dimension and a derivative in the other.

The Sobel Filter is a common filter for computing image derivatives

original

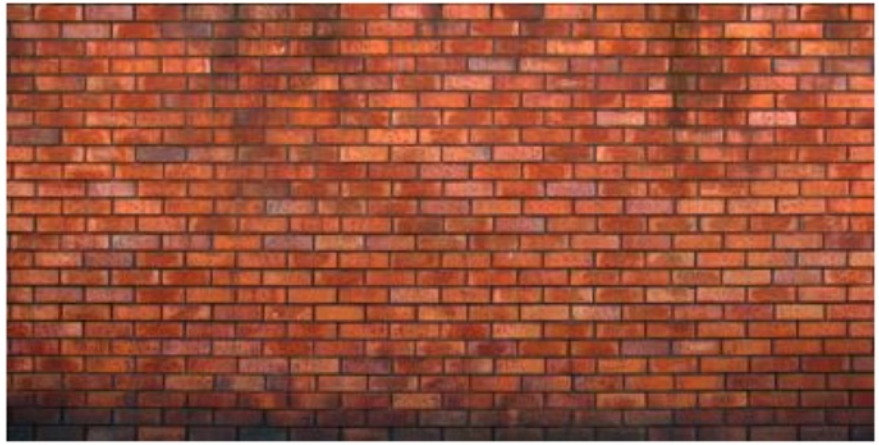


horizontal Sobel filter



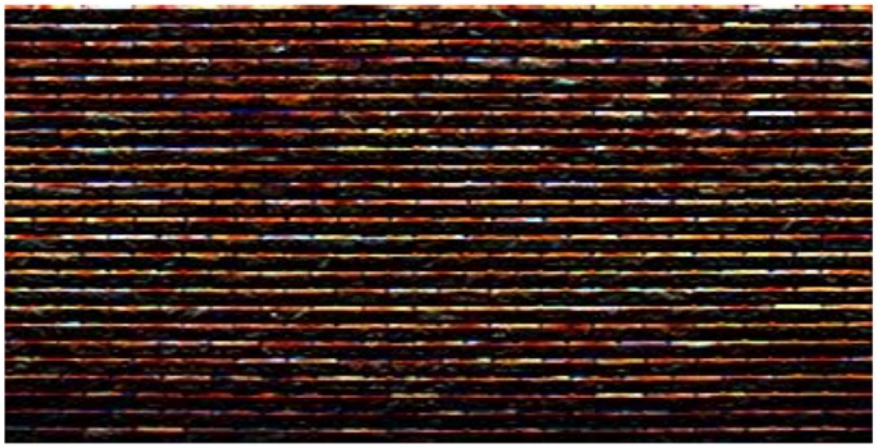
vertical Sobel filter

The Sobel Filter is a common filter for computing image derivatives



original

horizontal Sobel filter



vertical Sobel filter

Computing image gradients

1. Select your favorite derivative filters.

$$\boldsymbol{S}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \boldsymbol{S}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

2. Convolve with the image to compute derivatives.

$$\frac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S}_x \otimes \boldsymbol{f} \qquad \frac{\partial \boldsymbol{f}}{\partial y} = \boldsymbol{S}_y \otimes \boldsymbol{f}$$

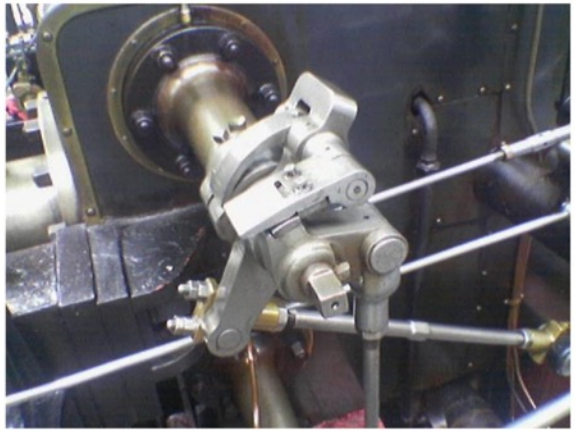
3. Form the image gradient, and compute its direction and amplitude.

$$\nabla \boldsymbol{f} = \left[\frac{\partial \boldsymbol{f}}{\partial x}, \frac{\partial \boldsymbol{f}}{\partial y}\right] \qquad \theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right) \qquad \|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

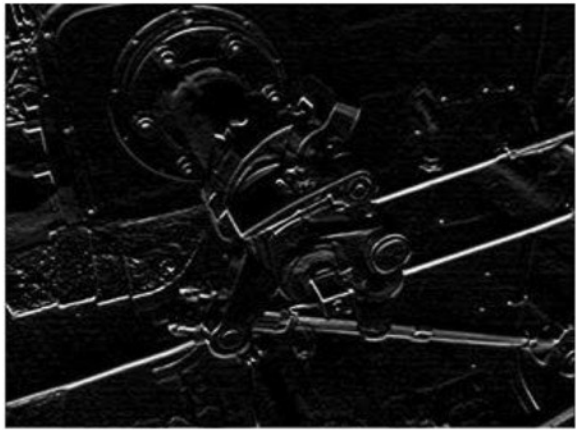
gradient direction amplitude

Computing image gradients

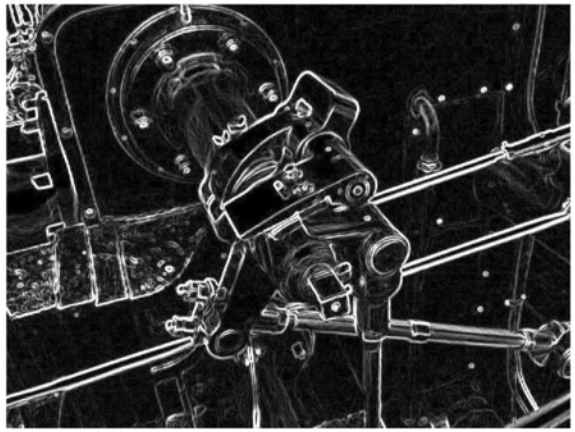
original



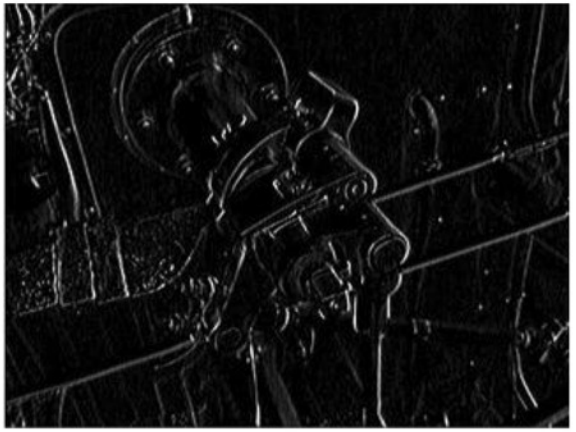
vertical derivative



gradient amplitude

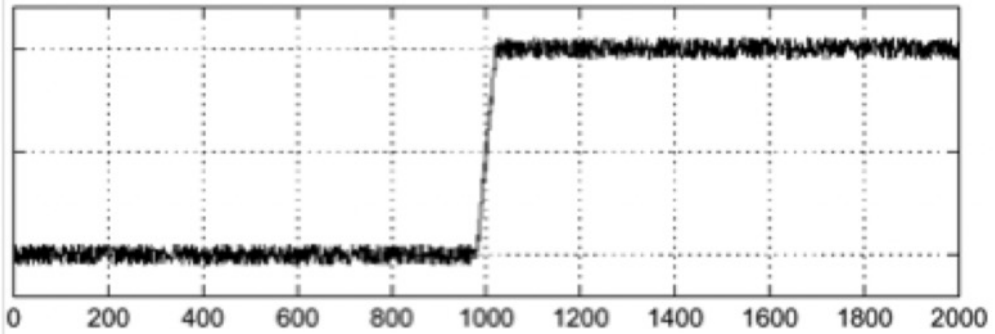


horizontal derivative



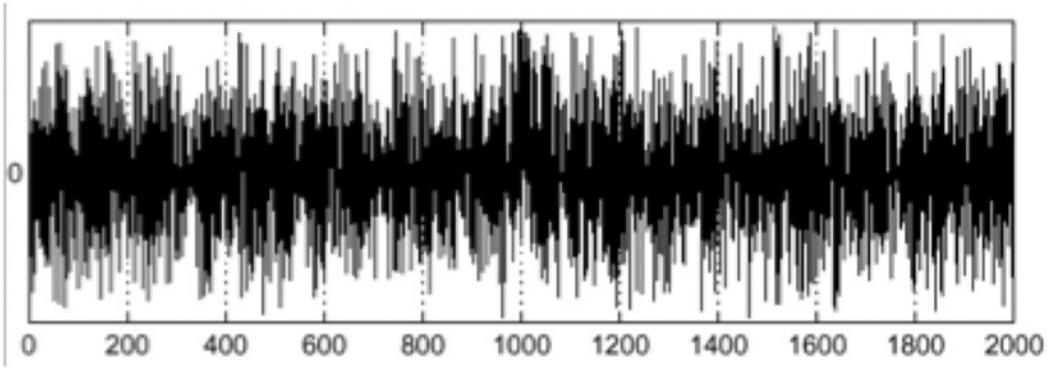
An Example: let's find the edge

intensity plot



Let's use a derivative filter:

derivative plot



The result is quite sensitive to noise.

An Example: let's find the edge

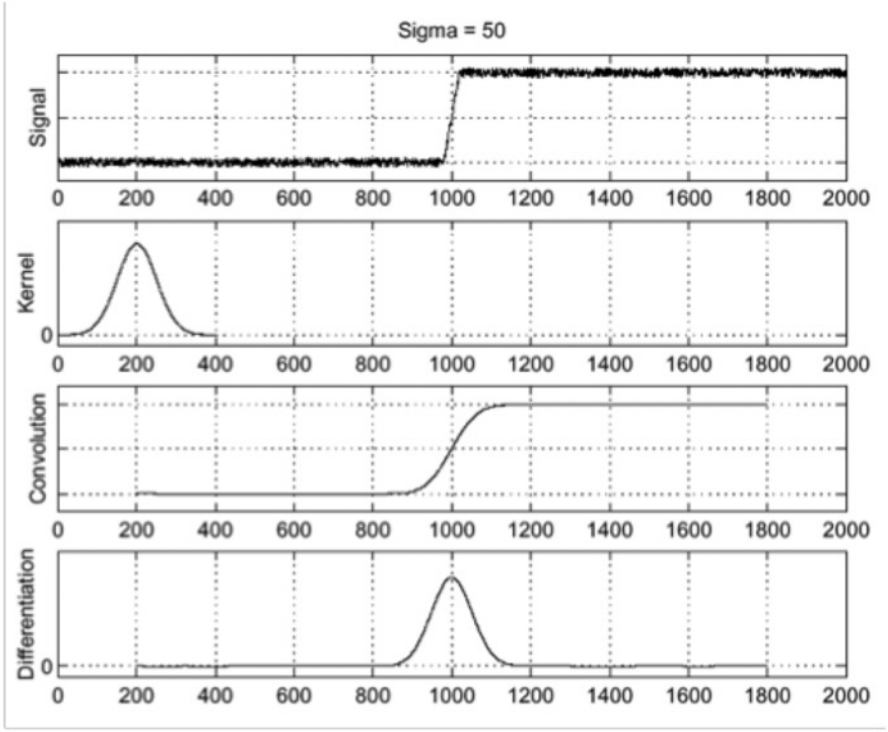
What if we blurred the signal first:

input

Gaussian

blurred

derivative of blurred



The output is much more stable.

An Example: let's find the edge

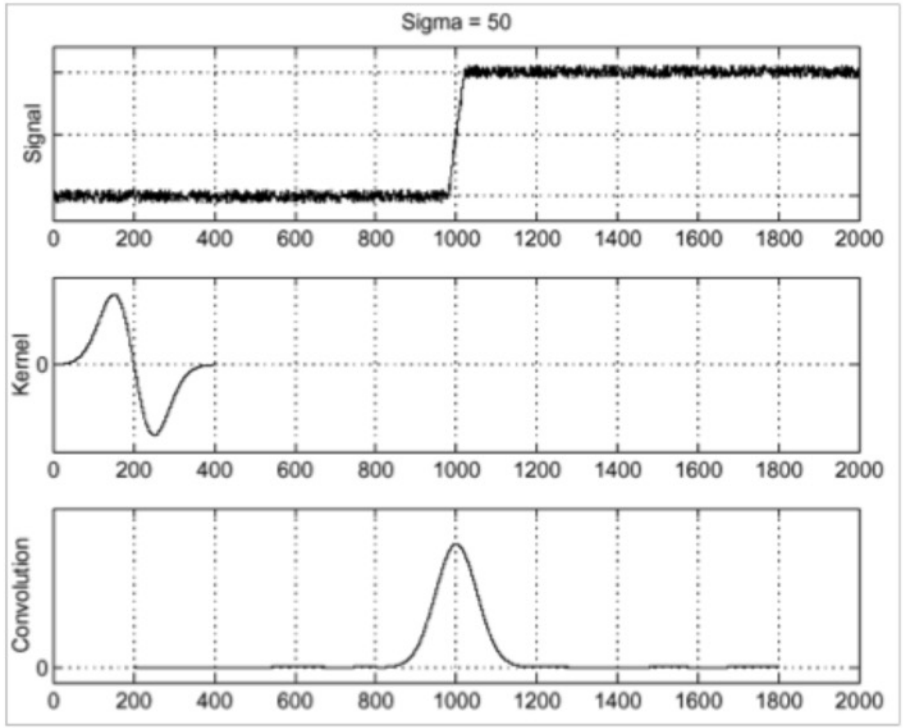
The derivative theorem of the convolution:

$$\frac{\partial}{\partial x}(f * h) = \frac{\partial f}{\partial x} * h$$

input

derivative of Gaussian

output (same as before)



We can also compute higher-order derivatives using finite differences

$$f''(x) = \lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \rightarrow \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

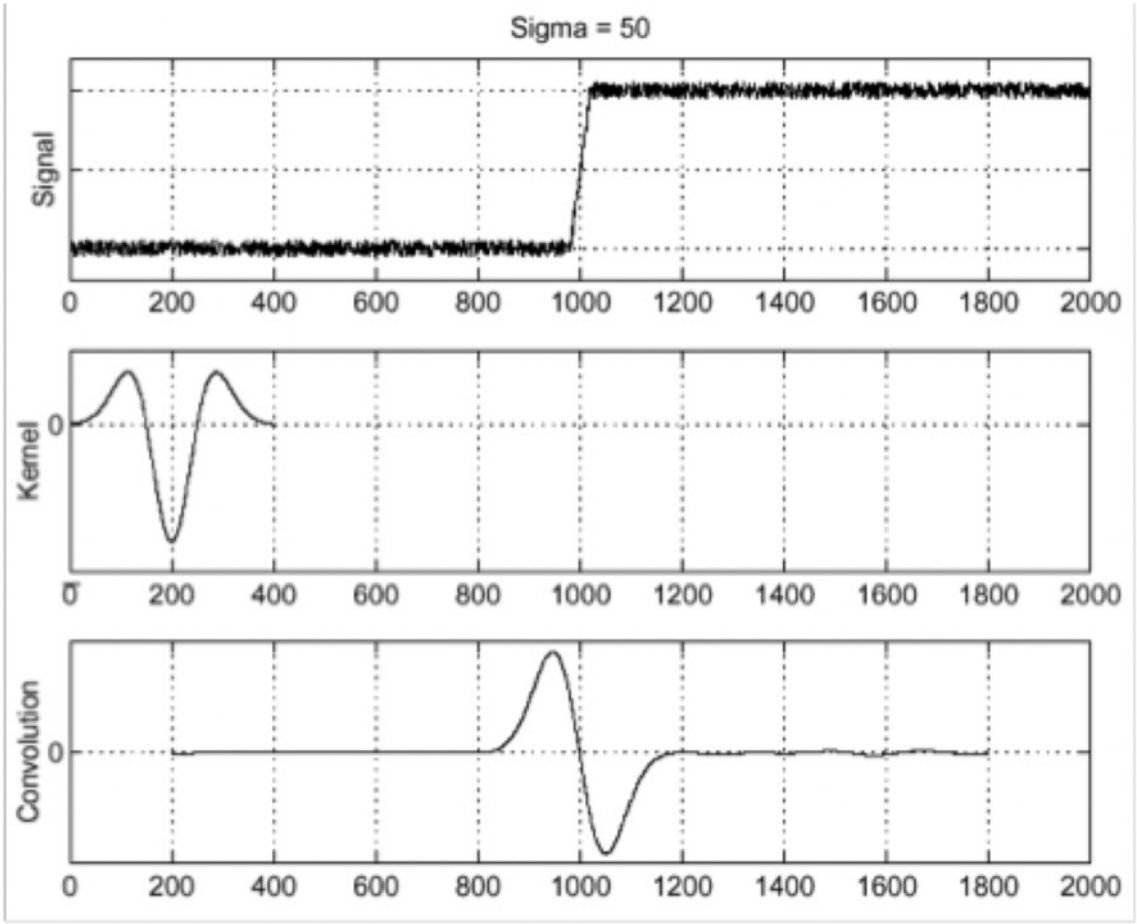
This is known as the Laplace filter.

The Laplace of Gaussian (LoG) Filter

input

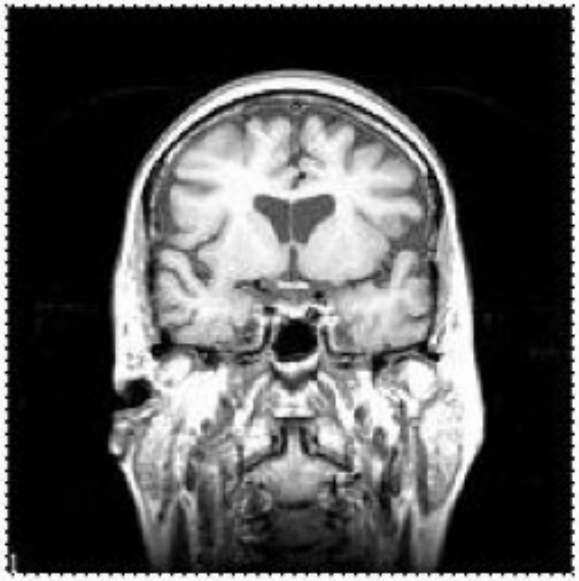
Laplacian of Gaussian

output



This filter has zero-crossings at edges

The Laplace of Gaussian (LoG) Filter



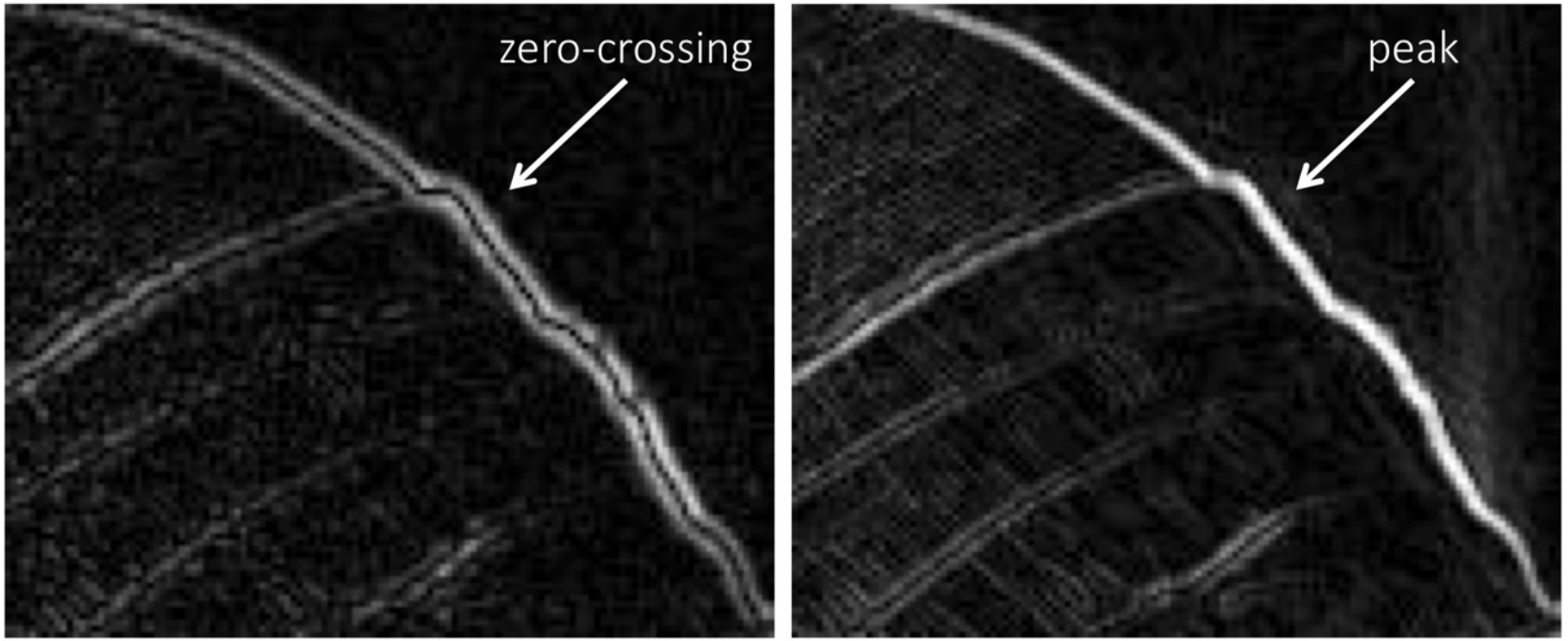
(A) Original MR image

(B) Laplacian results

(C) Extraction of the zero crossing of the Laplacian (object edges)

From: https://mipav.cit.nih.gov/pubwiki/index.php/Filters_(Spatial)_Laplacian

The Laplace of Gaussian (LoG) Filter



Laplacian of Gaussian filtering

Derivative of Gaussian filtering

Edges exist at zero-crossings of the LoG filter (but a peaks of the Derivative of Gaussian filter)

So what can't we do with filtering?

Point Processing

Warping

Filtering

So what can't we do with filtering?

Point Processing

Warping

Filtering

Breakout Session:
L02B: Convolutions & Filters

Download the L02 materials from Blackboard, unzip the file and open the Jupyter notebook I provided. After a few minutes of discussion, we'll regroup and work on this together!

You are not graded on these! They are meant to give you some quick experience with the tools we just learned about and prepare you for the homework.