CSC589 Introduction to Computer Vision
Lecture 4

More on Histogram Equalization, Border Effect, Image Derivatives

Bei Xiao

Last lecture

- Unsharp masking
- Gaussian Filter
- Image histograms
- Basic image tutorial with Python

Separability example

2D convolution (center location only)

$$\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array}$$

The filter factors into a product of 1D filters:

$$\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array}$$

Perform convolution along rows:

$$\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline & 11 & \\ \hline & 18 & \\ \hline & 18 & \\ \hline \end{array}$$

Followed by convolution along the remaining column:

Source: K. Grauman

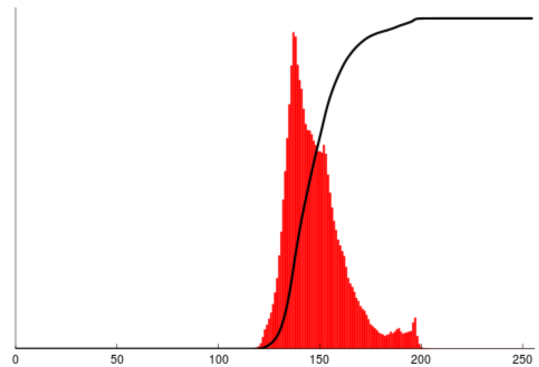
Matrix Multiplication

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^{\mathrm{T}} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & u_1 v_3 \\ u_2 v_1 & u_2 v_2 & u_2 v_3 \\ u_3 v_1 & u_3 v_2 & u_3 v_3 \\ u_4 v_1 & u_4 v_2 & u_4 v_3 \end{bmatrix}.$$

Today's Lecture

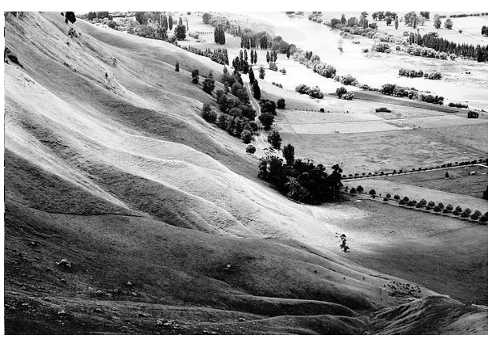
- More on image histograms
- Border effect and padding
- Image Derivatives/gradients

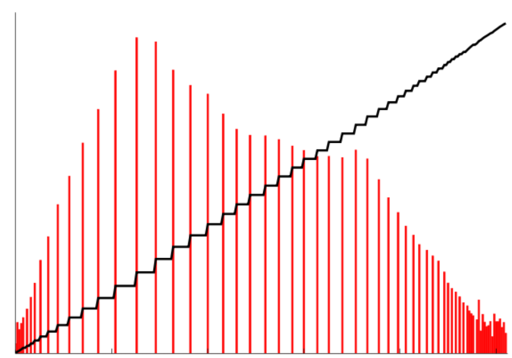
Image Histogram Equalization



Notice the “shape” of the histogram is preserved!

Same number of pixels
In each bin!





Questions

1. Why don’t we do the equalization directly on the histograms? What will happen to the image if it has a perfectly flat histograms?
2. What is CDF? Why are there steps (zig-zags) on the CDF?
3. How to make sure the “shape” of the PDF is preserved in the transformation?

Let’s look at in more details

We start with a gray-scale jpeg image of 8 by 8

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

https://en.wikipedia.org/wiki/Histogram_equalization

Let’s look at in more details

The histograms for this image (8 by 8 pixels) is shown in the following table (intensity that has zero pixels are skipped:

Value	Count	Value	Count	Value	Count	Value	Count	Value	Count
52	1	64	2	72	1	85	2	113	1
55	3	65	3	73	2	87	1	122	1
58	2	66	2	75	1	88	1	126	1
59	3	67	1	76	1	90	1	144	1
60	1	68	5	77	1	94	1	154	1
61	4	69	3	78	1	104	2		
62	1	70	4	79	2	106	1		
63	2	71	2	83	1	109	1		

Let's look at in more details

The cumulative distribution function (CDF) is shown below

Value	cdf	scaled cdf
52	1	0
55	4	
58	6	
59	9	
60	10	
61	14	
62	15	
…	…	
154	64	255

Number of pixels

This cdf shows that the min value in the subimage is 52 and max is 154. The cdf of value 154 corresponding to the total number of pixels (64)

Let's look at in more details

How do we compute the normalized CDF?

$$h(v) = \mathrm{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

Cdf(v): original cdf of pixel v
Cdfmin: minimum non-zero vale of the cdf
M×N: number of pixels, e.g. 64 (8x8)
L: 256

Quiz:

How do we compute normalized CDF of pixel 62?

Value	cdf	scaled cdf
52	1	0
55	4	
58	6	
59	9	
60	10	
61	14	
62	15	?
…	…	
154	64	255

$$h(v) = \mathrm{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

Quiz:

How do we compute normalized CDF of pixel 62?

Value	cdf	scaled cdf
52	1	0
55	4	
58	6	
59	9	
60	10	
61	14	
62	15	?
...	...	
154	64	255

H(62) = round((15-1)/
63*255)
= 57

Let's look at in more details

The cumulative distribution function (CDF) is shown below

Value	cdf	scaled cdf
52	1	0
55	4	12
58	6	22
59	9	32
60	10	36
61	14	53
62	15	57
…	…	…
154	64	255

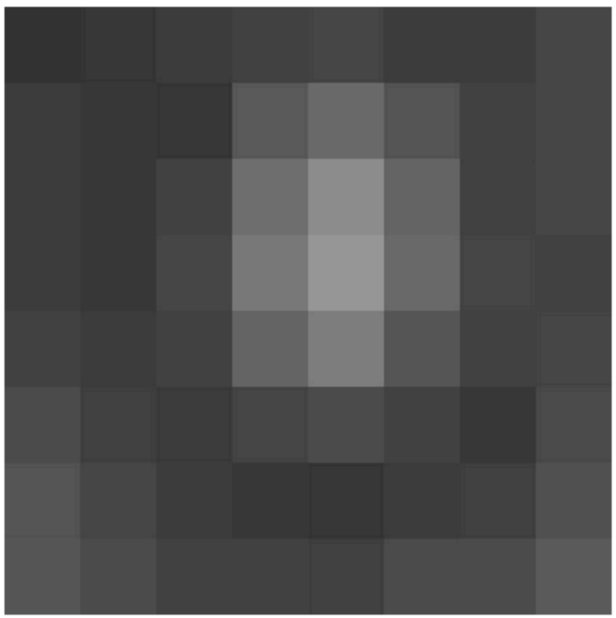
Let’s look at in more details

Now we can directly map the scaled cdf back to pixel values using the look up table we derived above

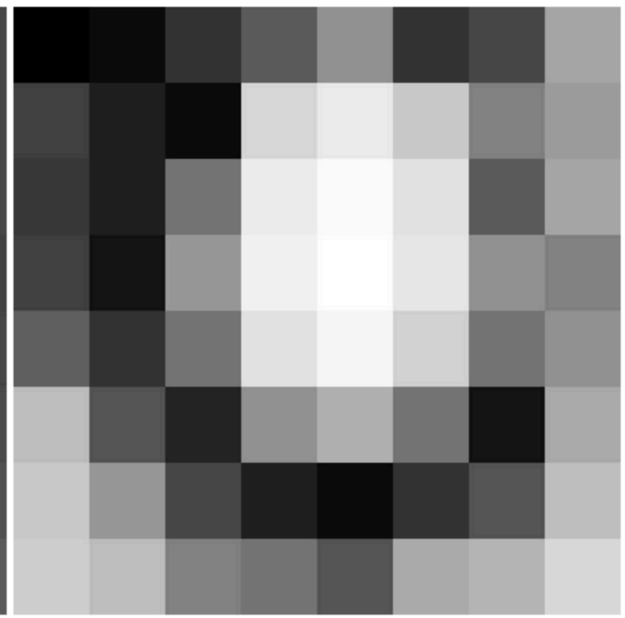
$$\begin{array}{c}
\begin{array}{cccccccc} 52 & 55 & 61 & 66 & & & & \end{array} \\
\begin{bmatrix}
0 & 12 & 53 & 93 & 146 & 53 & 73 & 166 \\
65 & 32 & 12 & 215 & 235 & 202 & 130 & 158 \\
57 & 32 & 117 & 239 & 251 & 227 & 93 & 166 \\
65 & 20 & 154 & 243 & 255 & 231 & 146 & 130 \\
97 & 53 & 117 & 227 & 247 & 210 & 117 & 146 \\
190 & 85 & 36 & 146 & 178 & 117 & 20 & 170 \\
202 & 154 & 73 & 32 & 12 & 53 & 85 & 194 \\
206 & 190 & 130 & 117 & 85 & 174 & 182 & 219
\end{bmatrix}
\end{array}$$

Let’s look at in more details

Notice that the minimum value (52) is now 0 and the maximum value (154) is now 255.



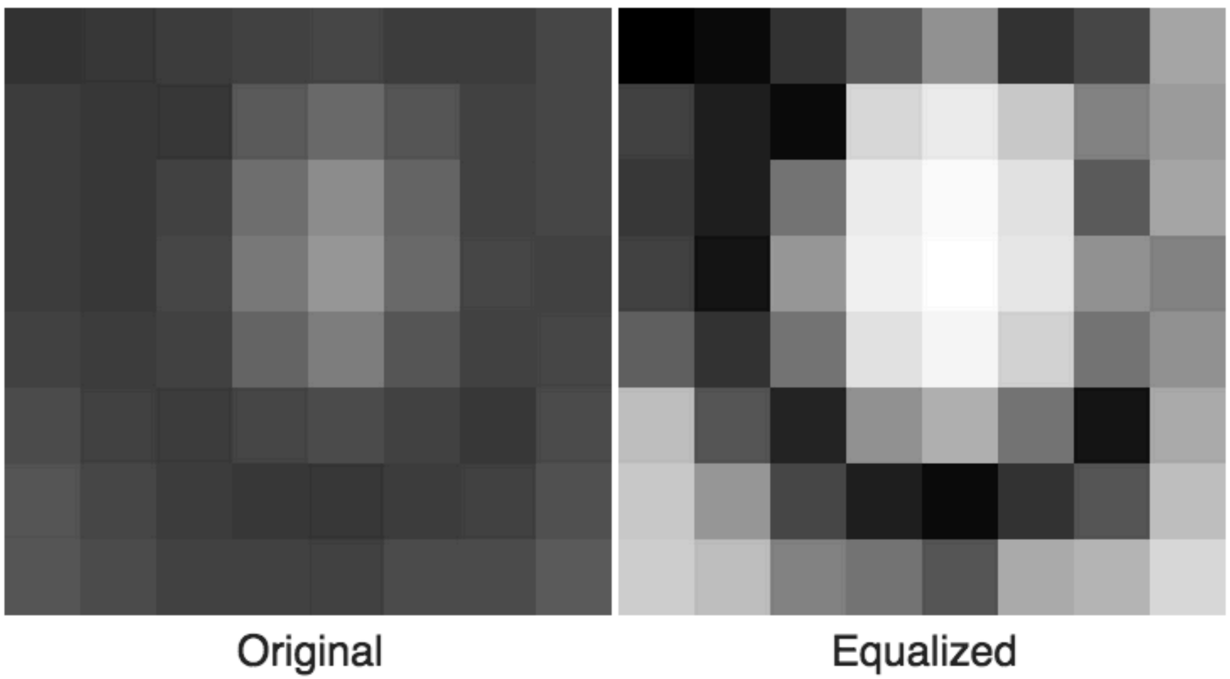
Original



Equalized

Let’s look at in more details

Notice that the minimum value (52) is now 0 and the maximum value (154) is now 255.



Homework 2: histogram equalization

Hint:

#Compute cdf in Python:

```
np.histogram(im.flatten(),nbr_bins,normed=True)

cdf = imhist.cumsum()

# normalize
cdf = 255 * cdf / cdf[-1]
# Using linear interpretation of cdf to find new pixels.

im2 = np.interp(im.flatten(),bins[:-1],cdf)
```

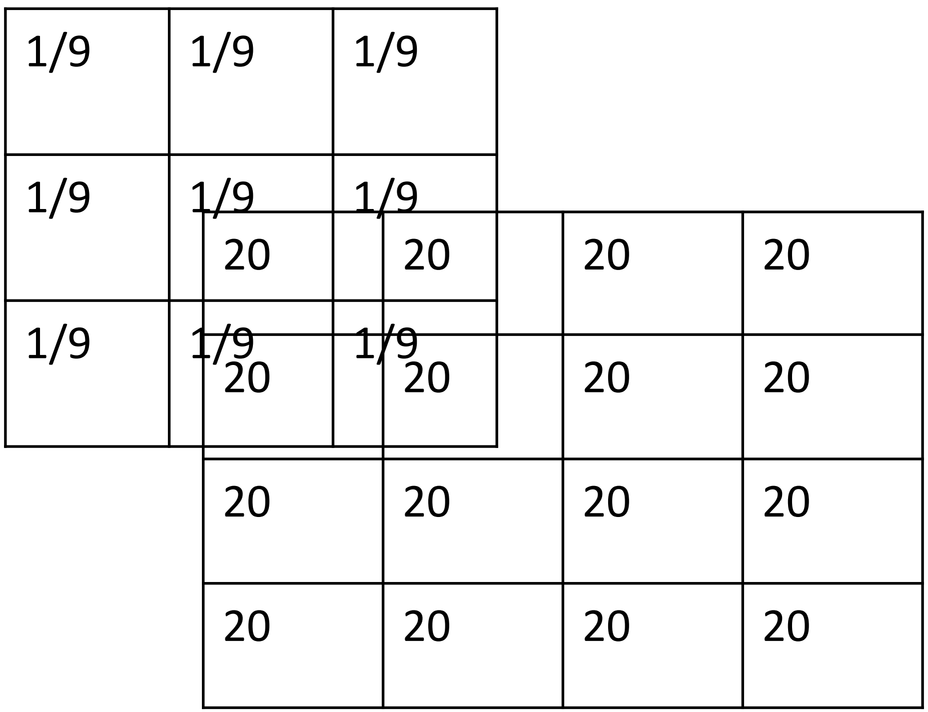
Back to Convolution

- Let’s blur a flat gray image:

How should it look like?

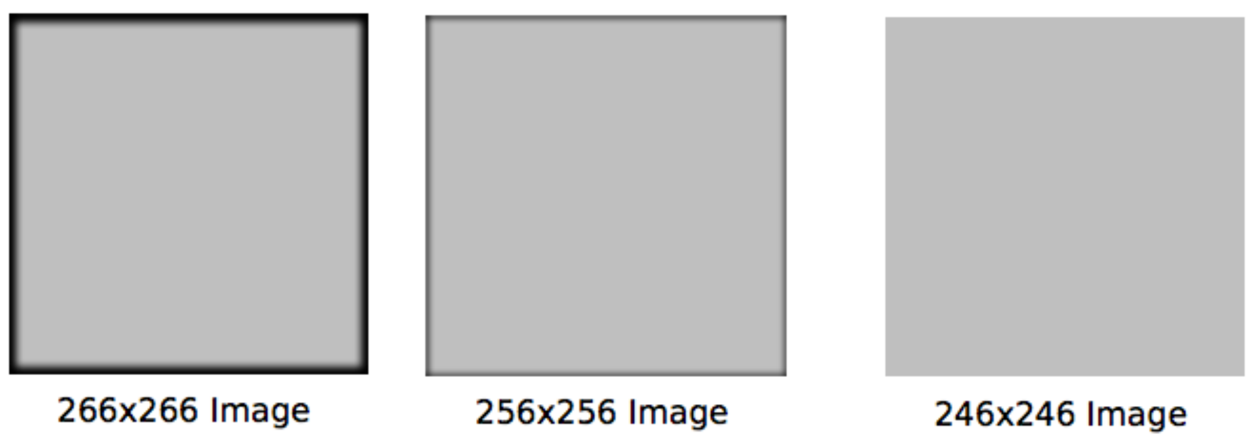
Back to Convolution

- Convolve with a box kernel



Border Handling

Depending on how you do the convolution, you could end up with 3 different images.



full same valid

Back to Convolution

- Zero padding:

Filled in borders with zeros, computed everywhere the kernel touches.

Numpy.convolve(a,v,mode ='full' or 'same')

This returns the convolution at each point of overlap, with an output shape of (N+M-1,). At the end-points of the convolution, the signals do not overlap completely, and boundary effects may be seen.

Back to Convolution

- Convolve with a box kernel, suppose there are zeros outside the image matrix.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

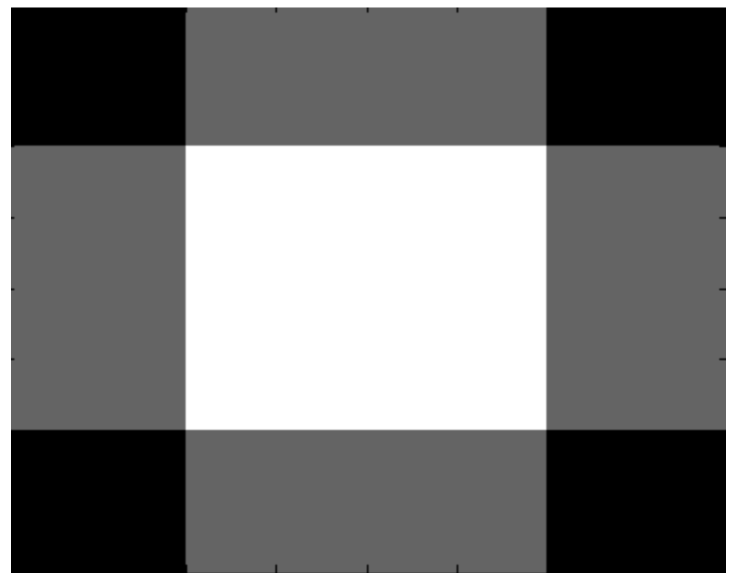
20	20	20	20
20	20	20	20
20	20	20	20
20	20	20	20

8.9	13.3	13.3	8.9
13.3	20.0	20.0	13.3
13.3	20.0	20.0	13.3
8.9	13.3	13.3	8.9

Back to Convolution

- This is called “same” in MATLAB/Numpy

4 ×4



4 ×4

Back to Convolution

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

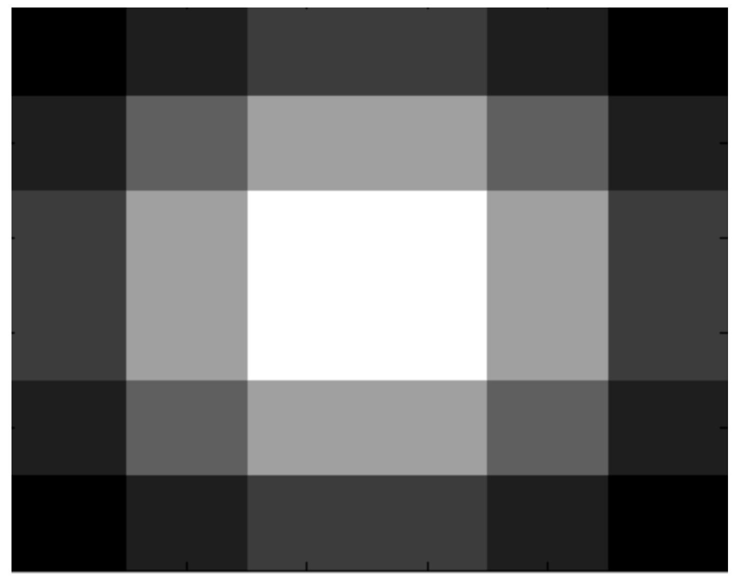
20	20	20	20
20	20	20	20
20	20	20	20
20	20	20	20

2.2	4.4	6.6	6.6	4.4	2.2
4.4	8.9	13.3	13.3	8.9	4.4
6.6	13.3	20.0	20.0	13.3	6.7
6.6	13.3	20.0	20.0	13.3	6.7
4.4	8.9	13.3	13.3	8.9	4.4
2.2	4.4	6.6	6.6	4.4	2.2

Back to Convolution

- This is called "full" in MATLAB/Numpy.

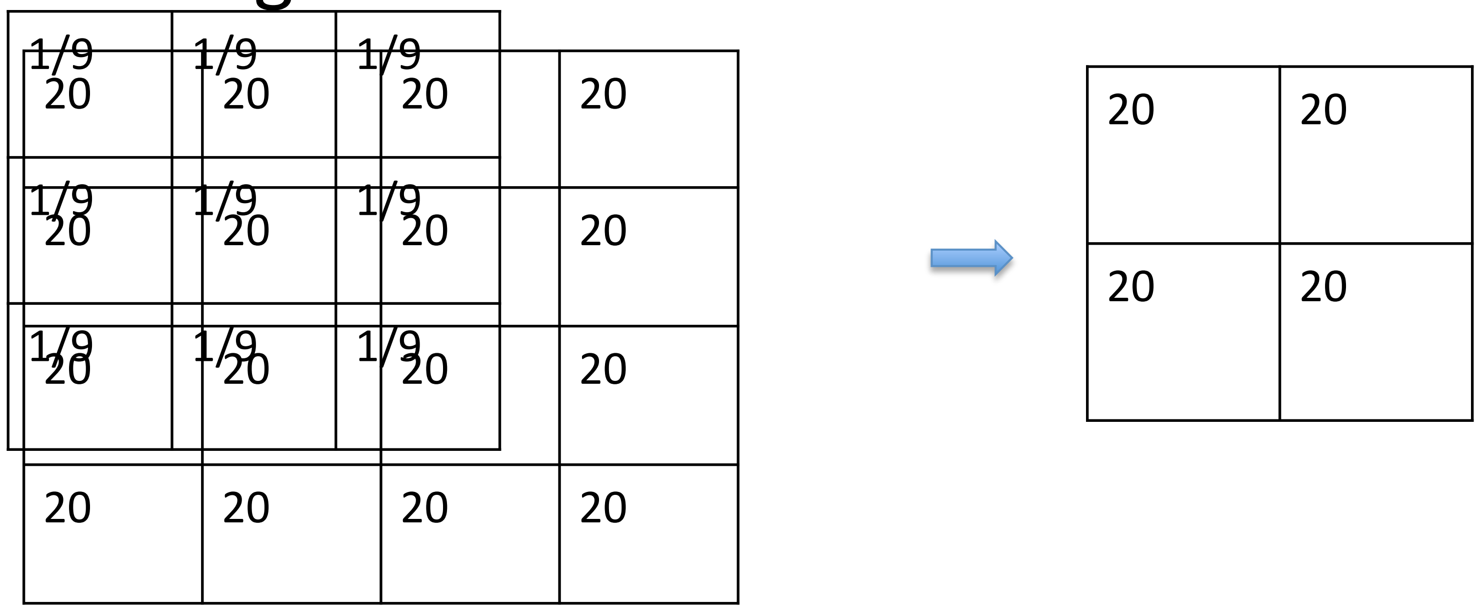
4×4



6×6

Back to Convolution

- Only compute at places where the kernel fits the image



Back to Convolution

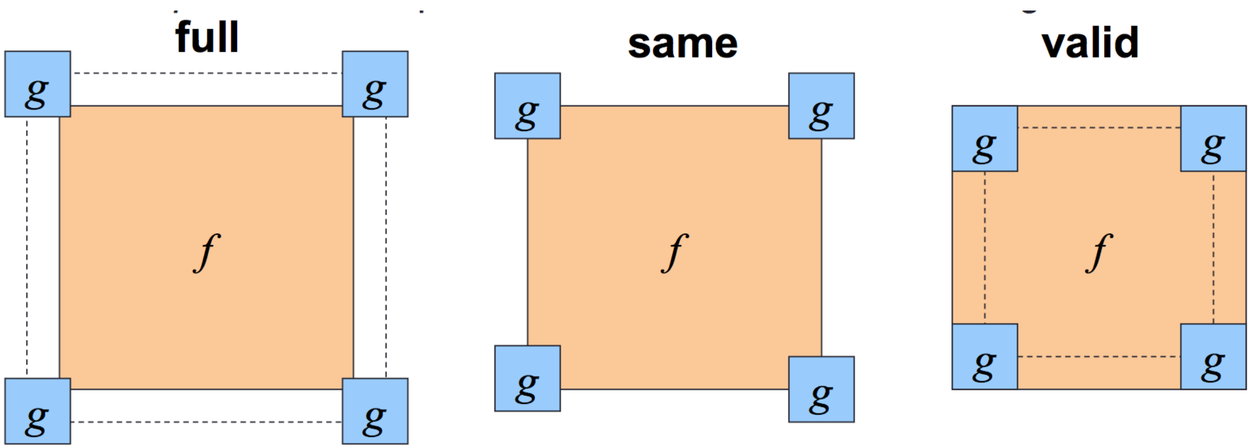
- This is called "valid" in MATLAB/Numpy.

4 ×4

2 ×2

To summarize

- Python:
  - Numpy.convolve(a,v,mode)
  - Scipy.signal.convole2d(a,v,mode, boundary, value)



There are other methods

- The first two methods that I described fill missing values in by substituting zero
- Can fill in values with different methods
  - Reflect image along border
  - Pull values from other side
- Read Chapter 3.2 “padding”.

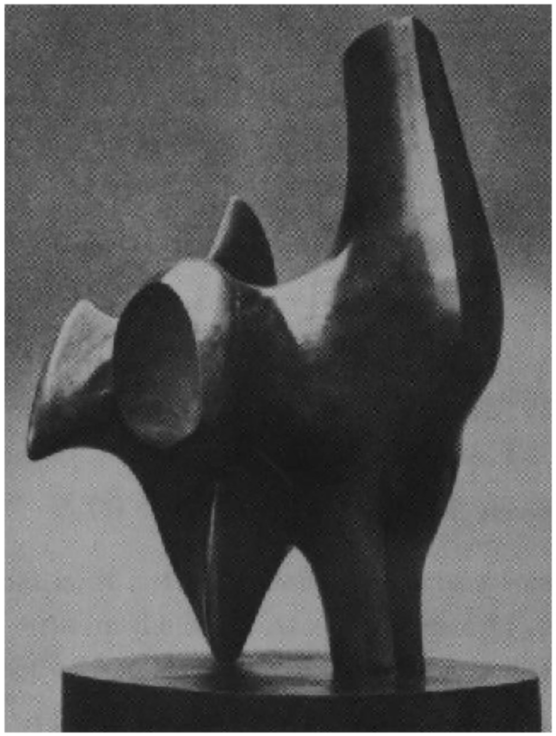
There are other methods

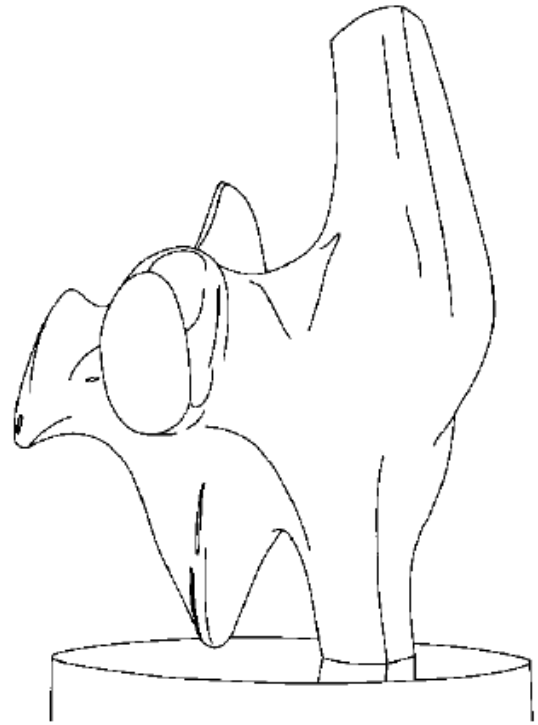
zero

valid

Nearest neighbor

Edge detection

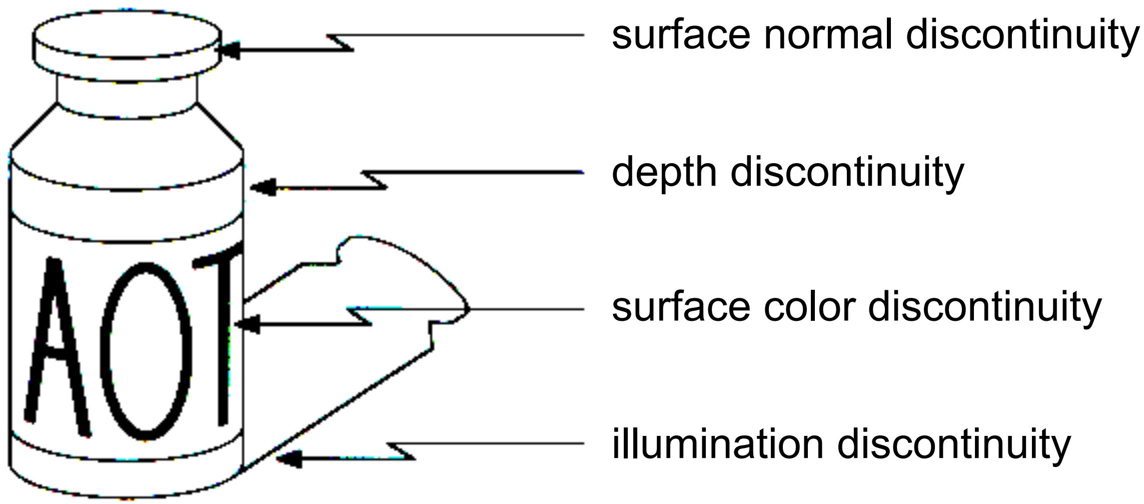




- Convert a 2D image into a set of curves
  - Extracts salient features of the scene
  - More compact than pixels

Source: N. Snavely

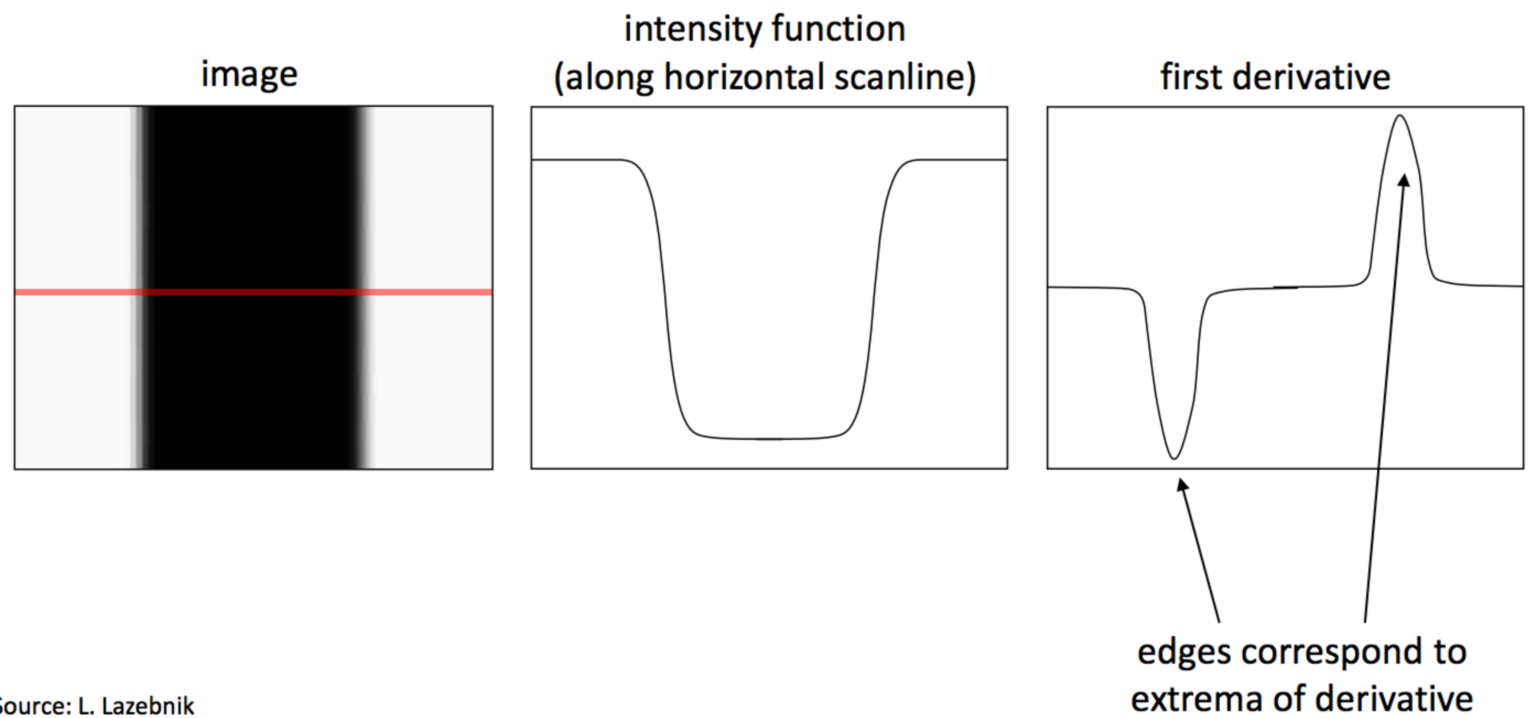
Origin of Edges



- Edges are caused by a variety of factors
- It is still a very active research topic! We will learn more about it!

Image derivate and edges

- An edge is a place of rapid change in image intensity function



Source: L. Lazebnik

Image derivatives

- How can we differentiate a *digital* image F[x,y]?
  - Option 1: reconstruct a continuous image, *f,* then compute the derivative
  - Option 2: take discrete derivative (finite difference) $$\frac{\partial f}{\partial x}[x,y] \approx F[x+1,y] - F[x,y]$$

How would you implement this as a linear filter?

$\frac{\partial f}{\partial x}$:

H_x

$\frac{\partial f}{\partial y}$:

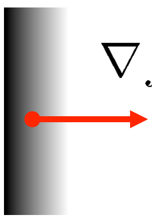
H_y

Source: S. Seitz

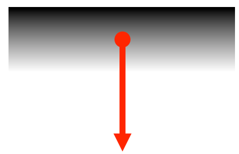
Image gradient

- The *gradient* of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

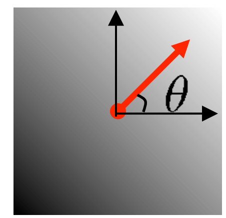
The gradient points in the direction of most rapid increase in intensity



$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$



$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

The *edge strength* is given by the gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

The gradient direction is given by:

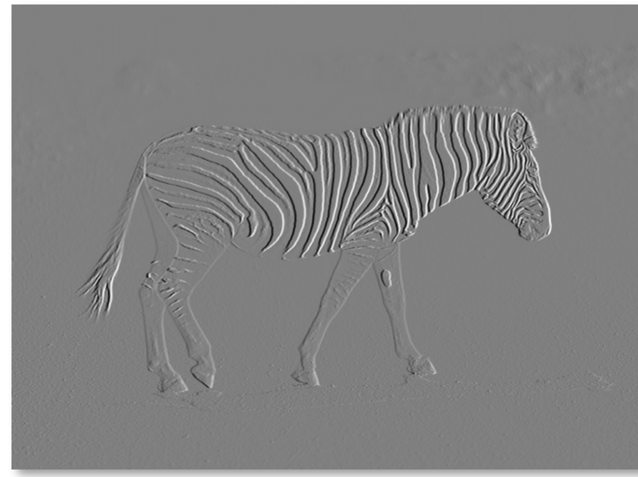
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- how does this relate to the direction of the edge?

Source: Steve Seitz

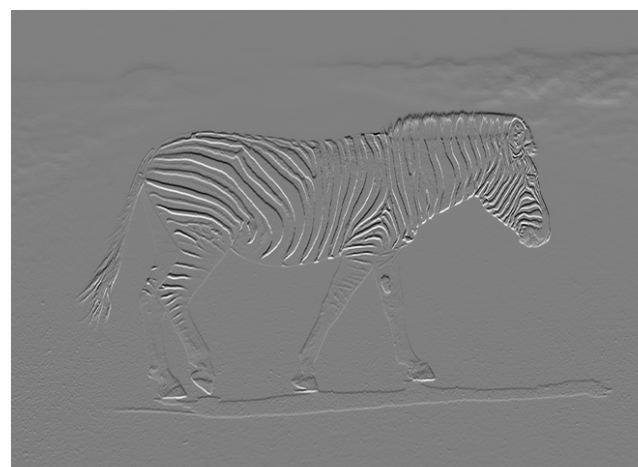
Image gradient

f



$\frac{\partial f}{\partial x}$

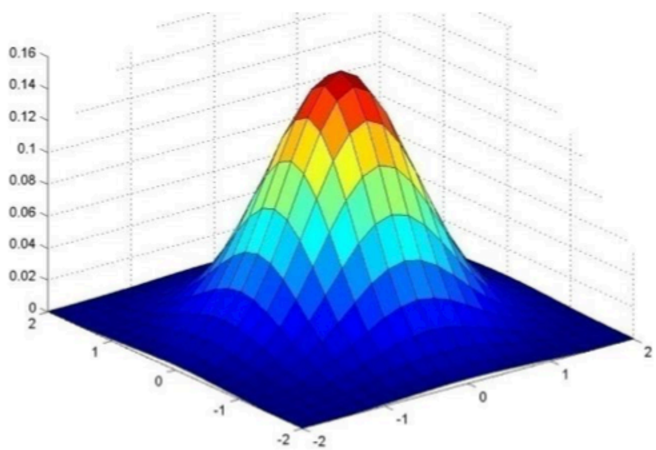
$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

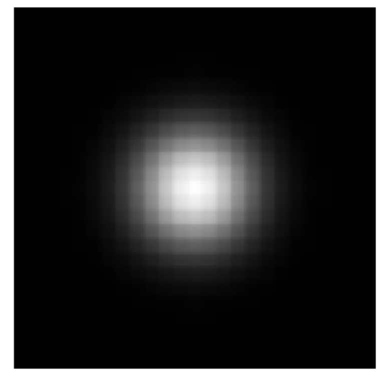


$\frac{\partial f}{\partial y}$

Gaussian Filter

- Gaussian = normal distribution function





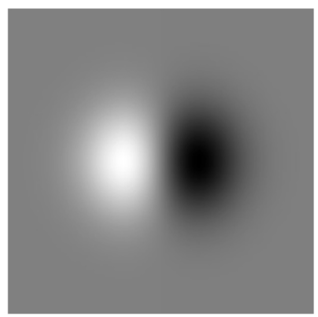
$$K(i,j) = \frac{1}{Z} \exp\left(-\frac{i^2 + j^2}{2\sigma^2}\right)$$

Derivative of Gaussian

- Take the derivative of the filter with respect to i:

$$\frac{\partial K(i,j)}{\partial i} = \frac{-i}{\sigma^2 Z} \exp\left(-\frac{i^2 + j^2}{2\sigma^2}\right)$$

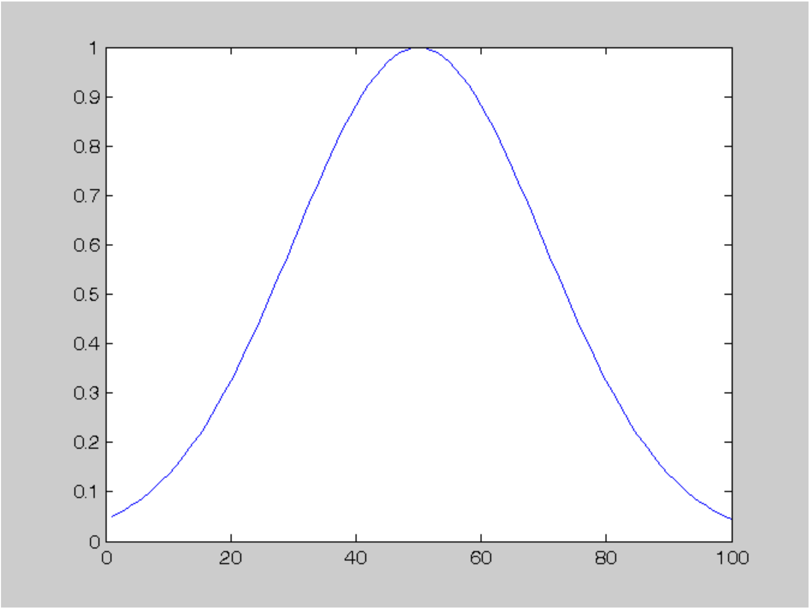
- Filter looks like:

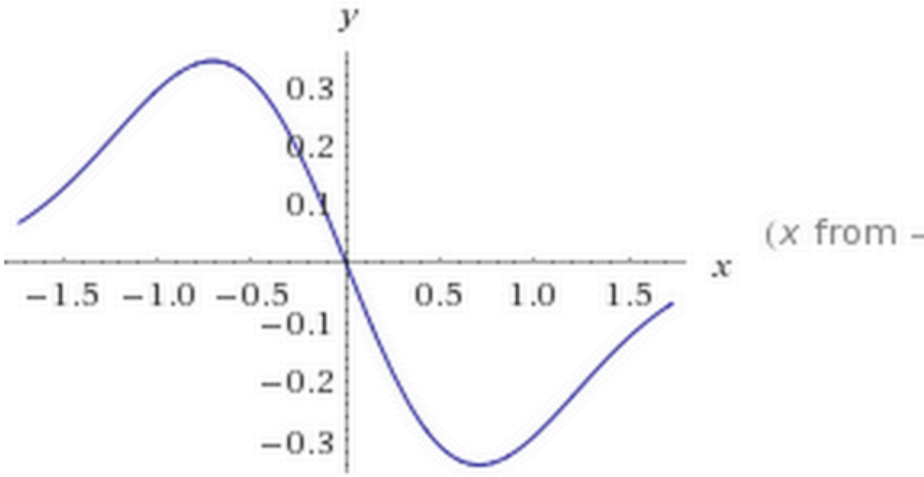


- Basically blur then take the derivative

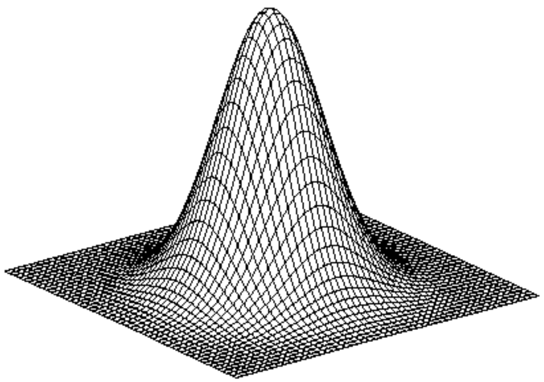
Source: M. Tappen

Derivative of Gaussian



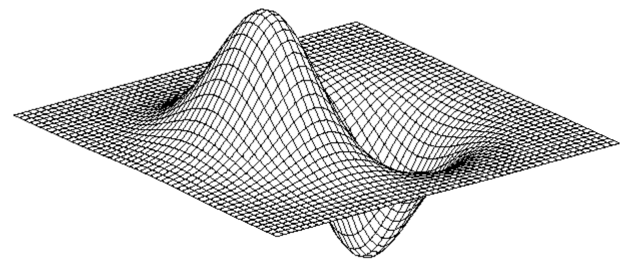


2D edge detection filters



Gaussian

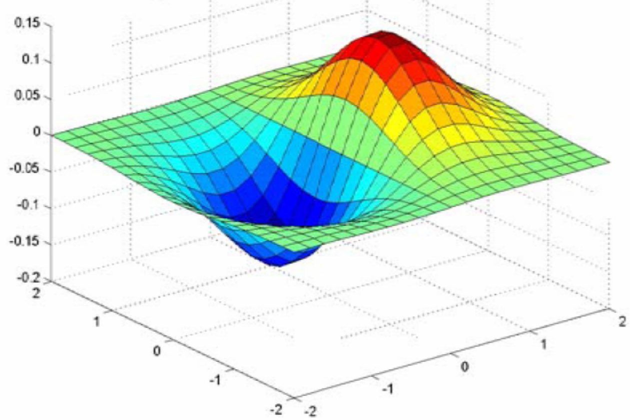
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



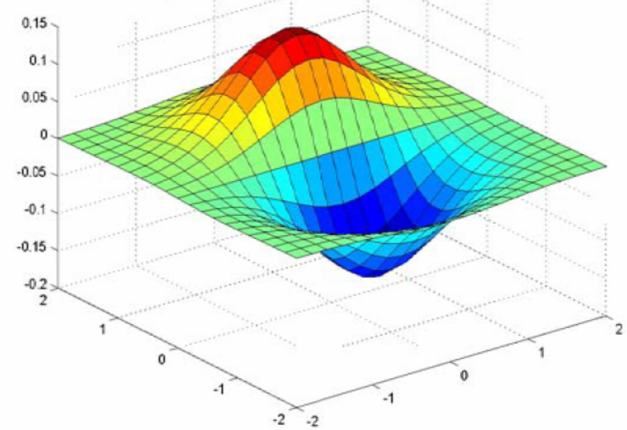
derivative of Gaussian (x)

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

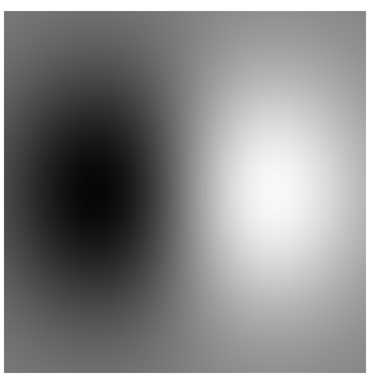
Derivative of Gaussian filter

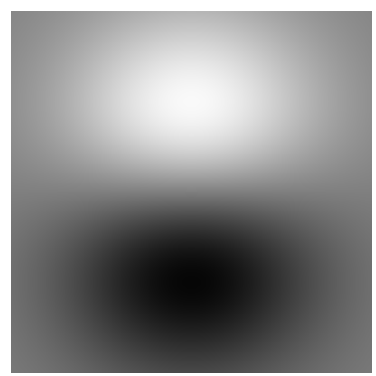


x-direction



y-direction





The Sobel operator

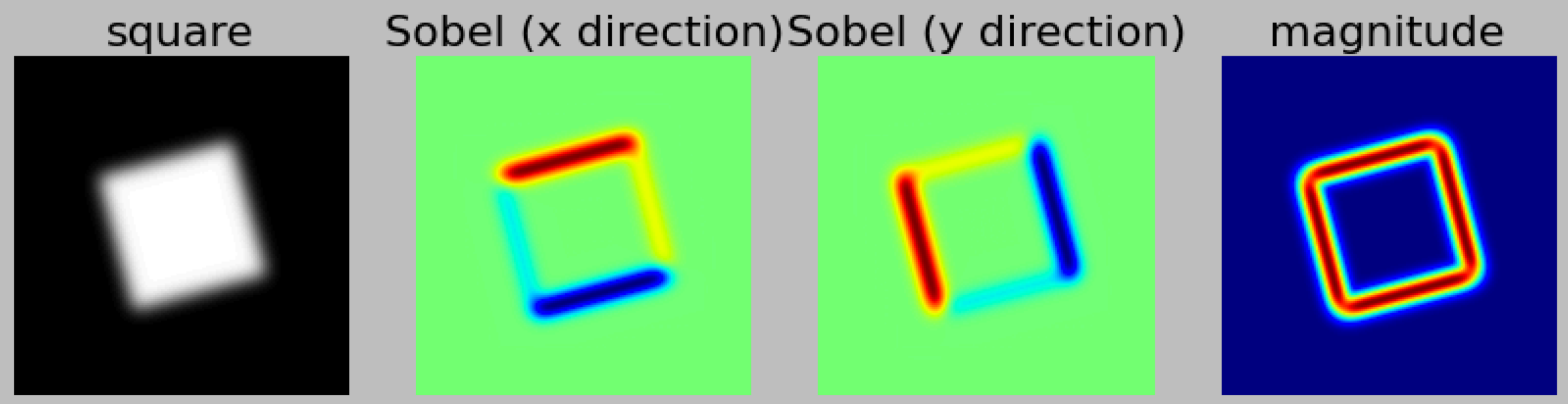
- Common approximation of derivative of Gaussian

$$\frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad \frac{1}{8}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

$$s_x \qquad\qquad s_y$$

- The standard defn. of the Sobel operator omits the 1/8 term
  - doesn't make a difference for edge detection
  - the 1/8 term **is** needed to get the right gradient value

Example of Sobel filtered image



Take-home exercise: Create image derivatives using Sobel filter

1. Generate an image of a rotated rectangle

```
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt

im = np.zeros((256, 256))
im[64:-64, 64:-64] = 1
im = ndimage.rotate(im, 15, mode='constant’)
```

2. Blur the image using a Gaussian filter

```
im = ndimage.gaussian_filter(im, 8)
```

3. Apply Sobel filter to both x and y direction.

```
sx =  ndimage.sobel(im, axis=0, mode='constant’)
```

4. Display the original image, x-derivatives, y-derivatives, and the gradient magnitude. You can use np.hypot to compute magnitude.
See here: http://docs.scipy.org/doc/numpy/reference/generated/numpy.hypot.html)

Next class

- Laplacian of Gaussian
- Steerable filter
- Fourier transform

- Reading: Chapter 3.1-3.3! Very important to keep up the reading.

- You should have read Chapter 1. Skip Chapter 2 since we haven’t covered it.