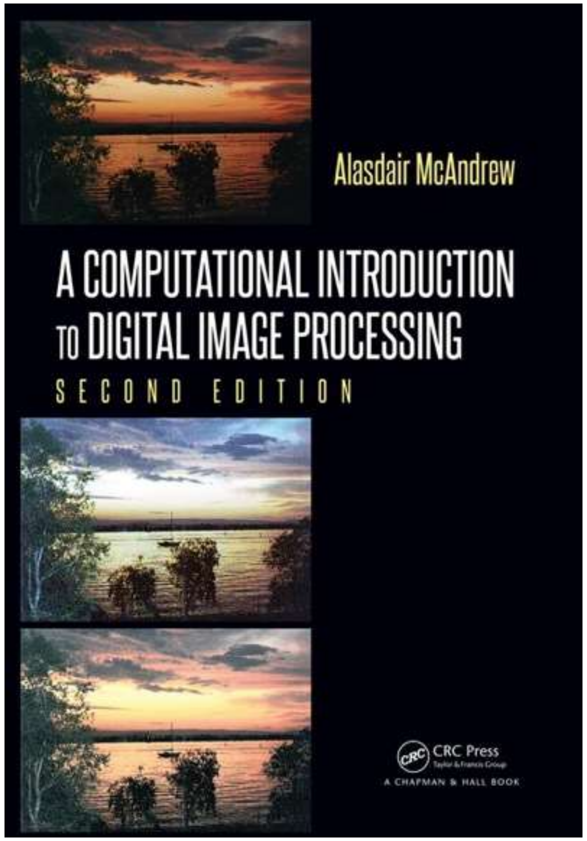
Chapter 5

Neighborhood Processing



Neighborhood Processing

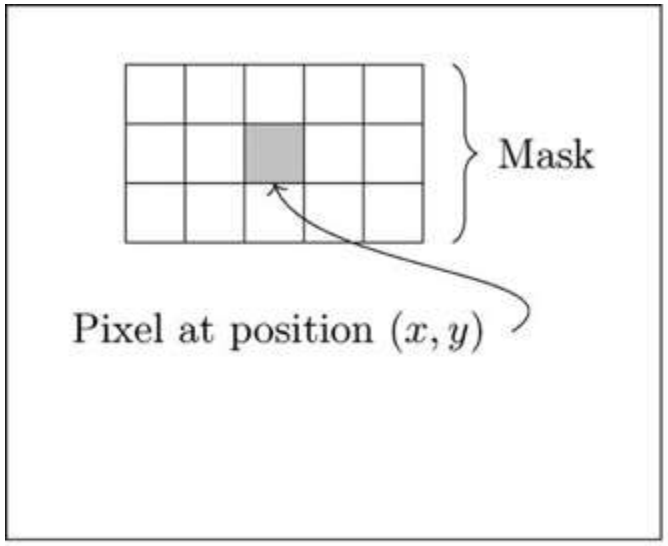
Neighborhood processing: An extension of point processing, where a function is applied to a neighborhood of each pixel.

Why??

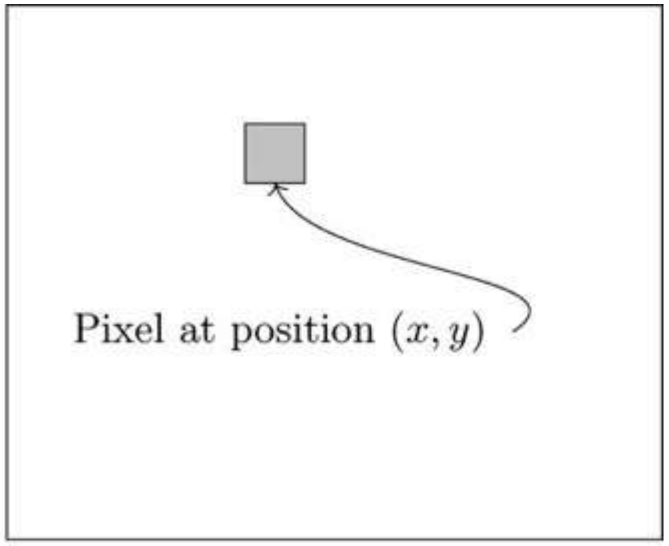
Mask: A rectangle (usually with sides of odd length) or other shape over the given image.

Filter: A combination of a mask and a function used to calculate new pixel values under the mask.

Neighborhood Processing



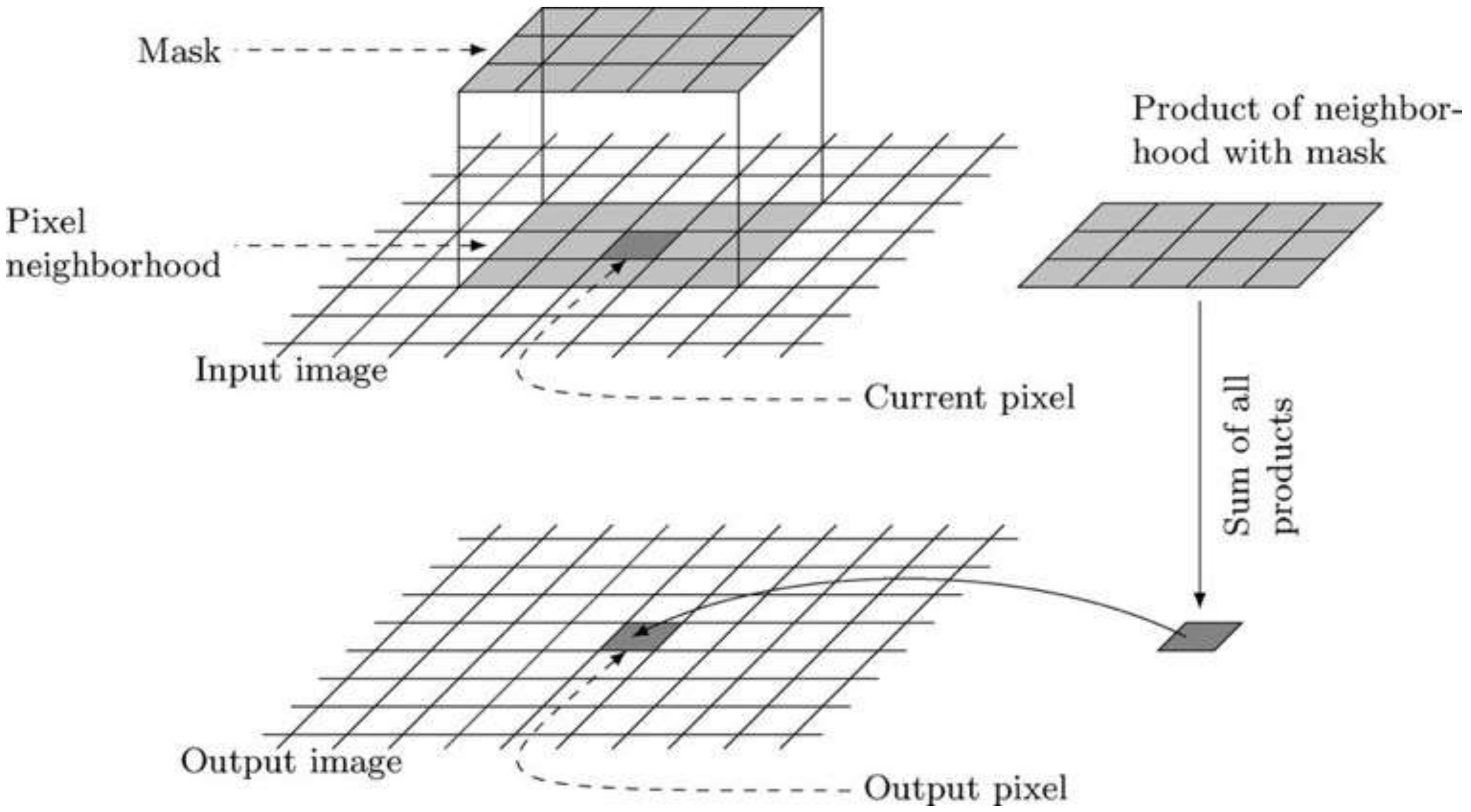
(a) Original image



(b) Image after filtering

Linear Spatial Filtering

Linear Filter: A filter that uses a linear function to calculate new pixel values in the mask.



Linear Spatial Filtering

New pixel value at p(i,j) = $$\sum_{s=-1}^{1} \sum_{t=-2}^{2} m(s,t)p(i+s, j+t).$$

Mask Values:

$m(-1,-2)$	$m(-1,-1)$	$m(-1,0)$	$m(-1,1)$	$m(-1,2)$
$m(0,-2)$	$m(0,-1)$	$m(0,0)$	$m(0,1)$	$m(0,2)$
$m(1,-2)$	$m(1,-1)$	$m(1,0)$	$m(1,1)$	$m(1,2)$

Pixel Values:

$p(i-1,j-2)$	$p(i-1,j-1)$	$p(i-1,j)$	$p(i-1,j+1)$	$p(i-1,j+2)$
$p(i,j-2)$	$p(i,j-1)$	$p(i,j)$	$p(i,j+1)$	$p(i,j+2)$
$p(i+1,j-2)$	$p(i+1,j-1)$	$p(i+1,j)$	$p(i+1,j+1)$	$p(i+1,j+2)$

Linear Spatial Filtering

New pixel value at p(i,j) = $\sum_{s=-1}^{1} \sum_{t=-2}^{2} m(s,t)p(i+s, j+t).$

In programming terms:

```
newp = 0;
for (s = -1; s <= 1; s ++)
   for (t = -2; t <= 2; t++)
         newp += m[s, t] * p[i + s][j + t];
newpic[i, j] = newp;
```

Linear Spatial Filtering

New pixel value at p(i,j) = $\sum_{s=-1}^{1} \sum_{t=-2}^{2} m(s,t)p(i+s,j+t).$

m

0	0	0
1	1	1
2	2	2

p

1	5	4	3	1	2
5	2	1	4	8	3
3	5	2	1	2	3
3	2	1	5	3	6
6	3	2	4	8	2

New pixel value for p(3,2)

Linear Spatial Filtering

New pixel value at p(i,j) = $\sum_{s=-1}^{1} \sum_{t=-2}^{2} m(s,t)p(i+s, j+t).$

m

0	0	0
1	1	1
2	2	2

p

1	5	4	3	1	2
5	2	1	4	8	3
3	5	2	1	2	3
3	2	1	5	3	6
6	3	2	4	8	2

New pixel value =
0 * 5 + 0 * 2 + 0 * 1 +
1 * 3 + 1 * 5 + 1 * 2 +
2 * 3 + 2 * 2 + 2 * 1 =

22

Linear Spatial Filtering

Linear Function:

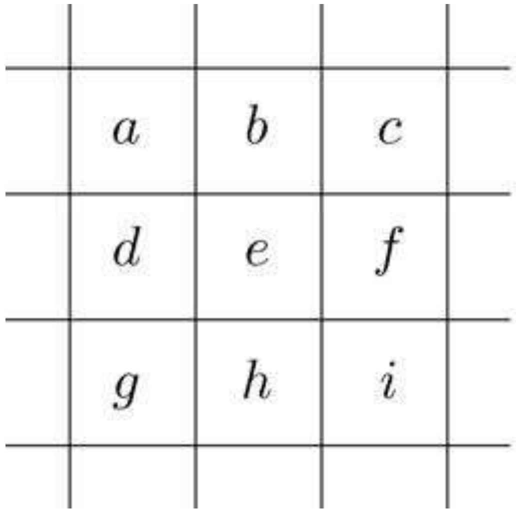
$$\sum_{s=-1}^{1} \sum_{t=-2}^{2} m(s,t)p(i+s, j+t).$$

Process (repeated for every pixel in the image):

1. Position the mask over the current pixel
2. Form all products of filter elements with the corresponding elements of the neighborhood
3. Add up all the products

Linear Filtering Example

One important linear filter is to use a 3 × 3 mask and take the average of all nine values within the mask. This value becomes the gray value of the corresponding pixel in the new image. This operation may be described as follows:



$$\longrightarrow \frac{1}{9}(a+b+c+d+e+f+g+h+i)$$

e is the gray value of the current pixel in the original image, and the average is the gray value of the corresponding pixel in the new image.

Linear Filtering Example

It is convenient to describe a linear filter simply in terms of the coefficients of all the gray values of pixels within the mask. This can be written as a matrix.

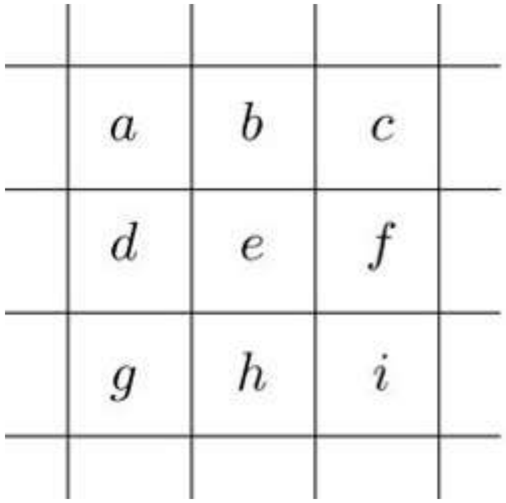
Averaging filter:

$$\frac{1}{9}a + \frac{1}{9}b + \frac{1}{9}c + \frac{1}{9}d + \frac{1}{9}e + \frac{1}{9}f + \frac{1}{9}g + \frac{1}{9}h + \frac{1}{9}i$$

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

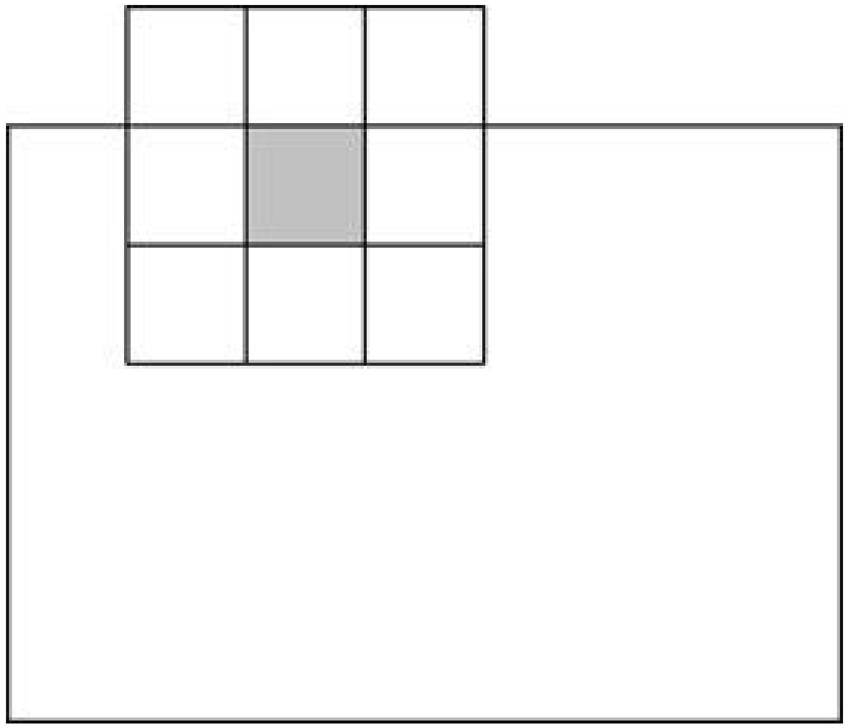
Linear Filtering Example 2

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$



$$\longrightarrow a - 2b + c - 2d + 4e - 2f + g - 2h + i$$

What about borders?



What about borders?

Ignore the edges

- Mask is only applied to pixels for which the mask will lie fully within the image.
- Output image that is smaller than the original.
- May lose a significant amount of information.

“Pad” with zeros

- Assume that all necessary values outside the image are zero.
- Returns an output image of the same size as the original.
- May have the effect of introducing unwanted artifacts (for example, dark borders) around the image.

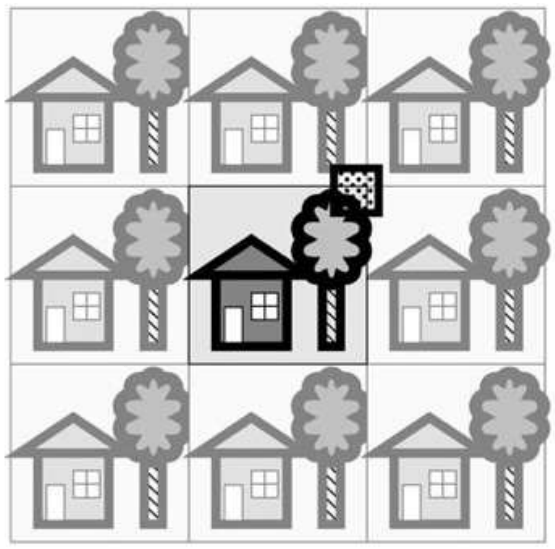
What about borders?

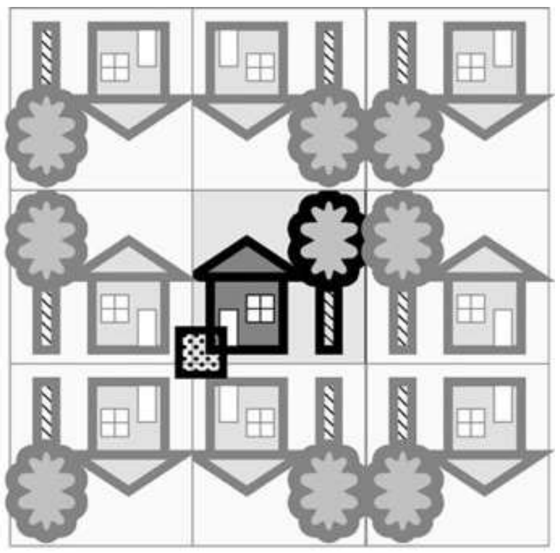
Repeat the image

- Tile the image in all directions, so that the mask always lies over image values.
- May introduce unwanted artifacts in the result.

Reflect the image

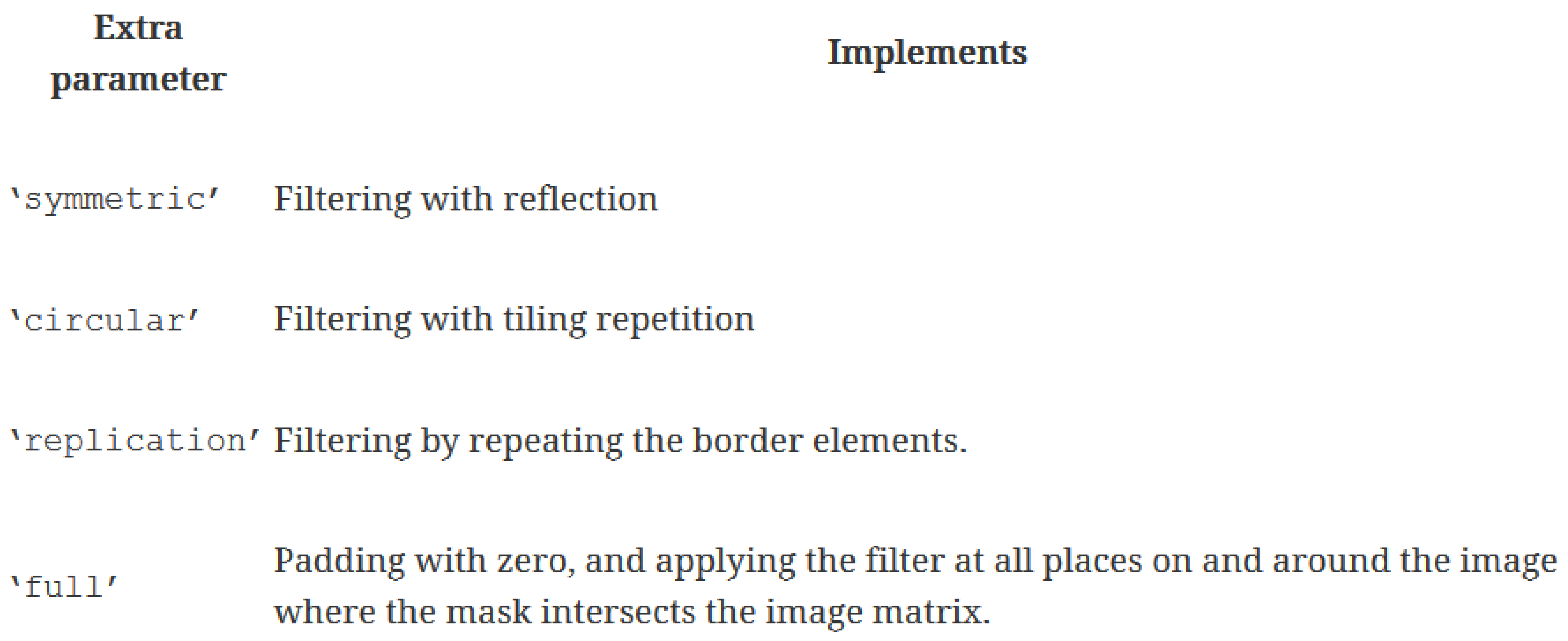
- Similar to repeating, except that the image is reflected across all of its borders.
- Ensures that there are no extra sudden changes are introduced at the borders.


Repetition


Reflection

Linear Filtering MatLab/Octave

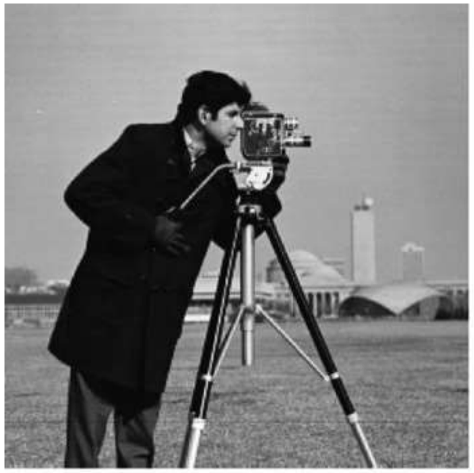
```
imfilter(image,filter,...)
```

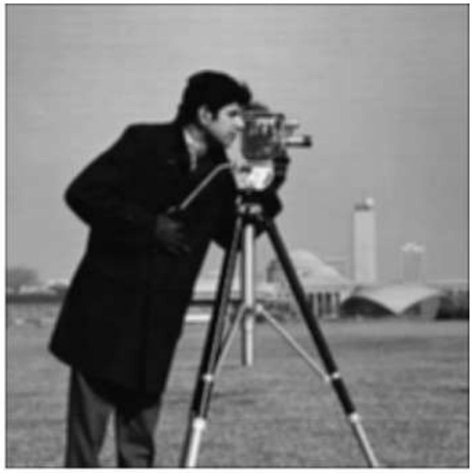


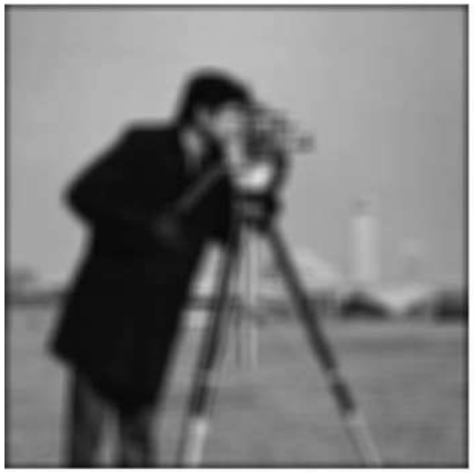
Extra parameter	Implements
'symmetric'	Filtering with reflection
'circular'	Filtering with tiling repetition
'replication'	Filtering by repeating the border elements.
'full'	Padding with zero, and applying the filter at all places on and around the image where the mask intersects the image matrix.

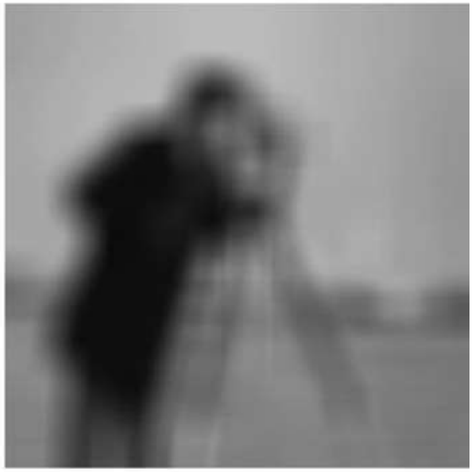
The last option returns a result that is larger than the original image.

Average Filter


(a) Original image


(b) Average filtering 3 x 3 filter

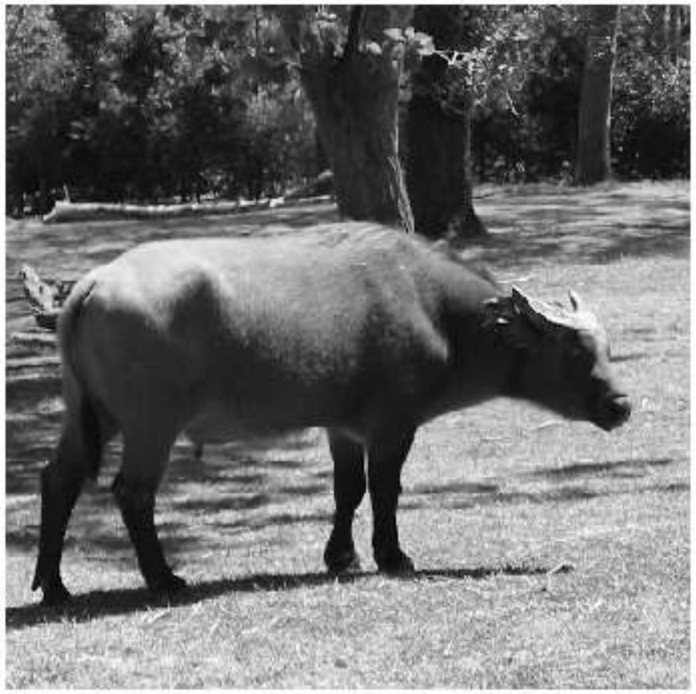

(c) Using a 9 × 9 filter


(d) Using a 25 × 25 filter

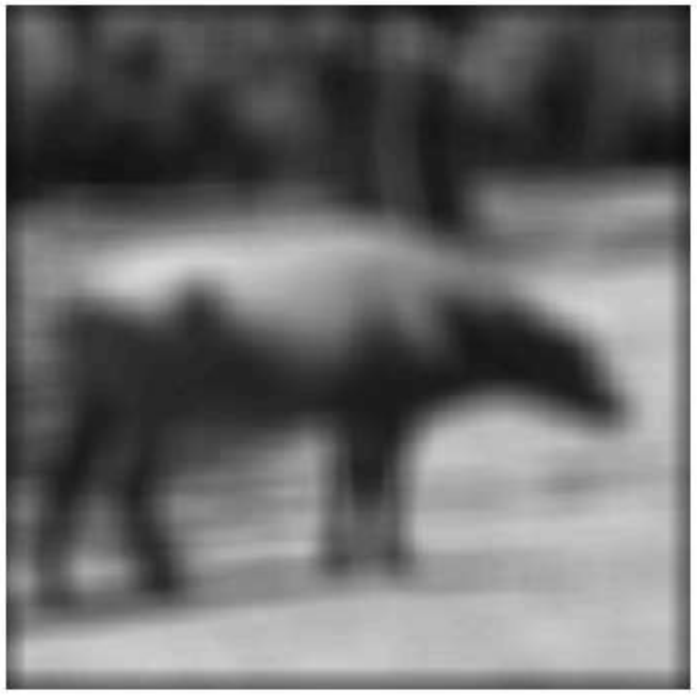
Octave Example

```
pic = imread('buffalo.png');
f1 = fspecial('average', 25);
newpic = imfilter(pic, f1);
imshow(newpic)
```

Pixels at the boundary of the image padded with zeros by default



pic

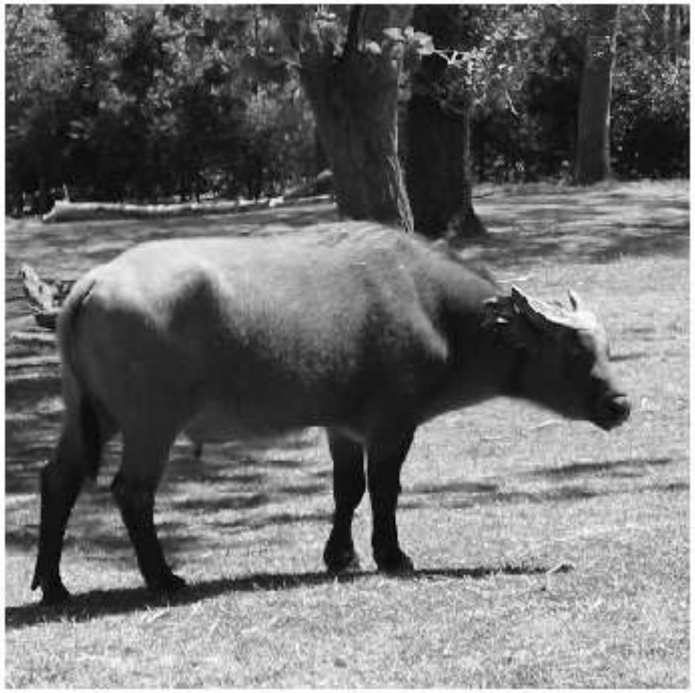


newpic

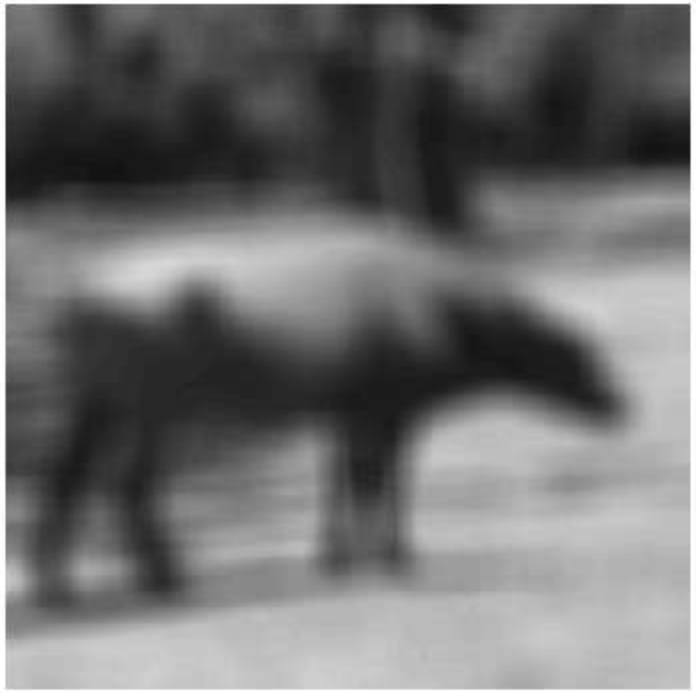
Octave Example

```
pic = imread('buffalo.png');
f1 = fspecial('average', 25);
newpic = imfilter(pic, f1, 'symmetric');
imshow(newpic)
```

Symmetric filters with reflection.



pic

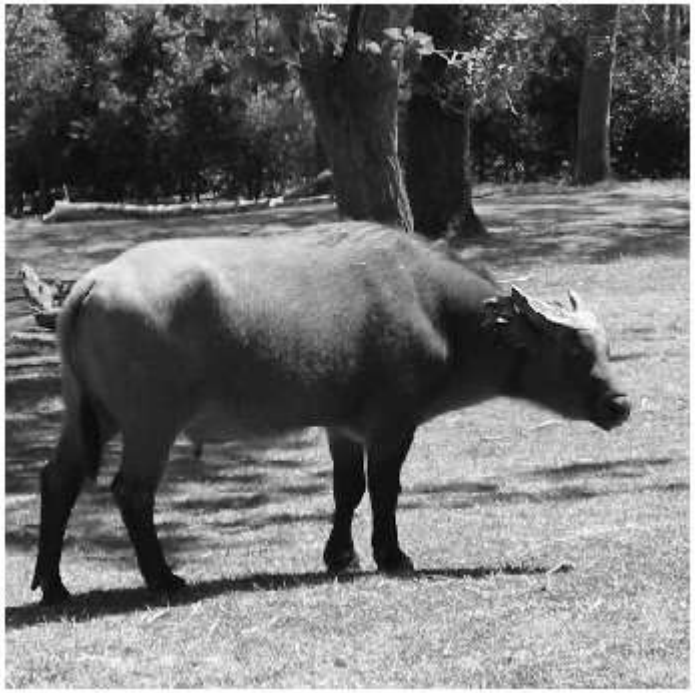


newpic

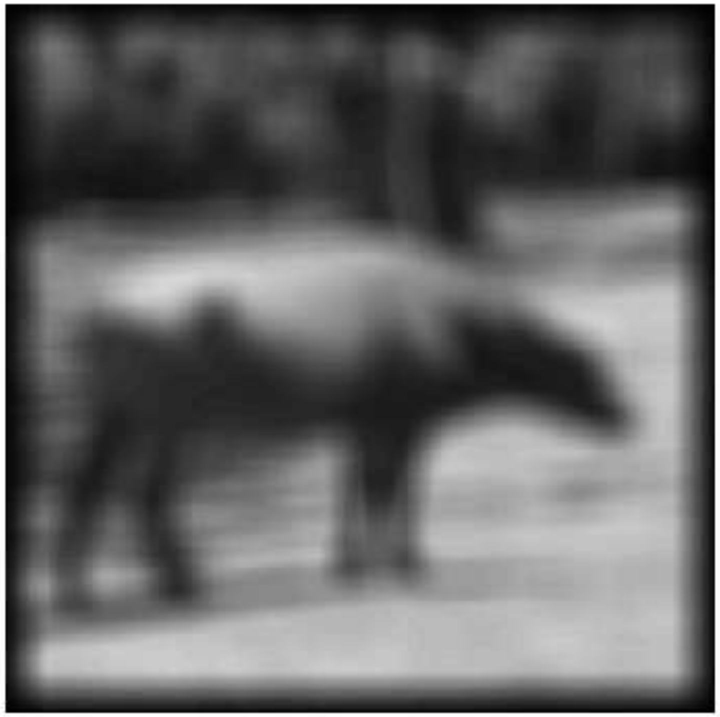
Octave Example

```
pic = imread('buffalo.png');
f1 = fspecial('average', 25);
newpic = imfilter(pic, f1, 'full');
imshow(newpic)
```

Full applies the filter at all places on and around the image where the mask intersects the image matrix - returns a result that is larger than the original.



pic



newpic

What is best?????

??????????

Ignore edges…

Pad with zeros…

??????????

??????????

Reflect…

??????????

Repeat…

??????????

What is best?????

Trick Question…

No single “best” approach; the method must be dictated by the problem at hand, by the filter being used, and by the result required.

Separable Filters

Some filters can be implemented by the successive application of two simpler filters.

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

The 3 × 3 averaging filter can be implemented by first applying a 3 × 1 averaging filter, and then applying a 1 × 3 averaging filter to the result.

The application of an n x n filter requires n^2 multiplications, and $n^2 - 1$ additions for each pixel in the image. But the application of an n x 1 filter only requires n multiplications and n − 1 additions. Since this must be done twice, the total number of multiplications and additions are 2n and 2n − 2, respectively.

Separable Filters

All averaging filters are separable; another separable filter is the Laplacian

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}.$$

We will also consider other examples.

Laplacian Filtering 3x3 mask

New pixel value at p(i,j) = $\sum_{s=-1}^{1} \sum_{t=-2}^{2} m(s,t)p(i+s,j+t).$

m

1	-2	1
-2	4	-2
1	-2	1

p

1	5	4	3	1	2
5	2	1	4	8	3
3	5	2	1	2	3
3	2	1	5	3	6
6	3	2	4	8	2

New pixel value =
1 * 5 + -2 * 2 + 1 * 1 +
-2 * 3 + 4 * 5 + -2 * 2 +
1 * 3 + -2 * 2 + 1 * 1 =

12

Note: 3^2 = 9 multiplications
3^2 -1 = 8 additions

Laplacian Filtering Separated

m1

1
-2
1

p

1	5	4	3	1	2
5	2	1	4	8	3
3	5	2	1	2	3
3	2	1	5	3	6
6	3	2	4	8	2

After m1 applied to p

1 * 2 + -2 * 5 + 1 * 2 = -6

m2

1	-2	1

3	-8	-7	-2	6	-1
-6	6	4	-4	-13	-1
2	-6	-2	7	7	3
3	4	2	-5	4	-7
-9	-4	-3	-3	-13	2

After m2 applied to p' =

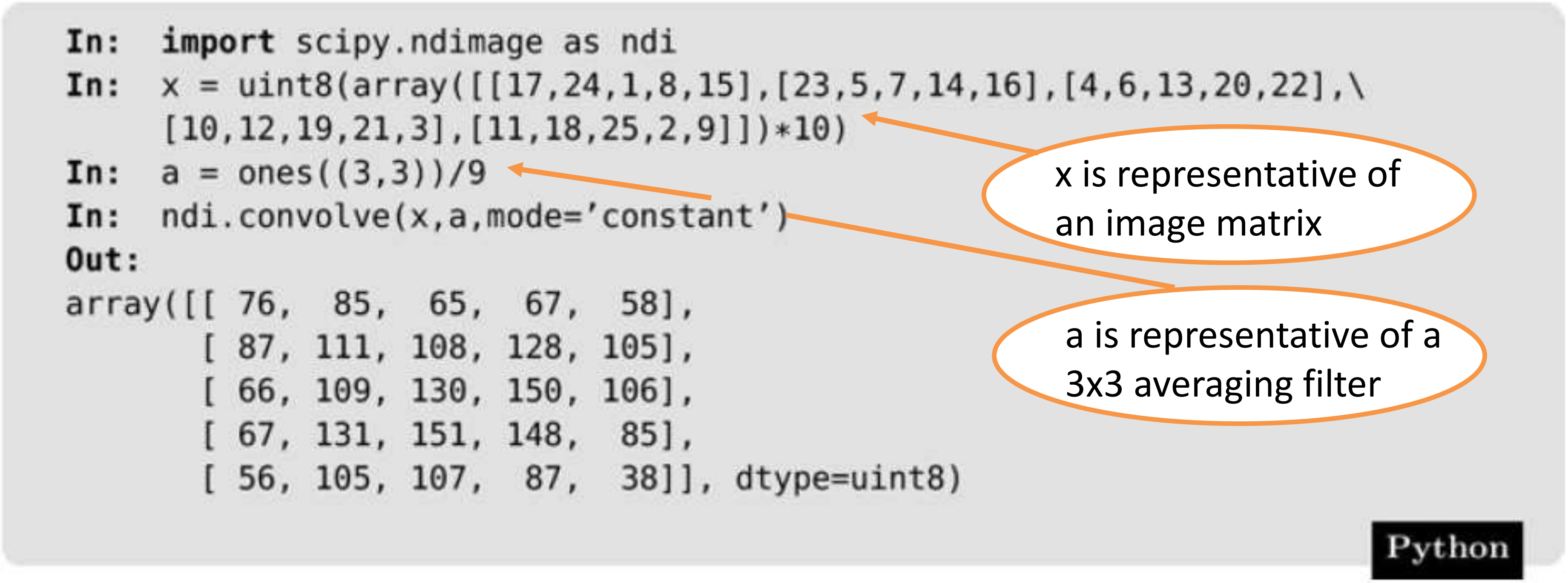
1 * 2 + -2 * -6 + 1 * -2 = 12

Note: 2 * 3 = 6 multiplications
2 * 3 – 2 = 4 additions

Filtering in Python

Python has no equivalent to *fspecial* and *imfilter*

Linear filtering can be performed by using *convolve* or *correlate* from the ndimage module of the scipy library, or generic_filter



```
In:  import scipy.ndimage as ndi
In:  x = uint8(array([[17,24,1,8,15],[23,5,7,14,16],[4,6,13,20,22],\
     [10,12,19,21,3],[11,18,25,2,9]])*10)
In:  a = ones((3,3))/9
In:  ndi.convolve(x,a,mode='constant')
Out:
array([[ 76,  85,  65,  67,  58],
       [ 87, 111, 108, 128, 105],
       [ 66, 109, 130, 150, 106],
       [ 67, 131, 151, 148,  85],
       [ 56, 105, 107,  87,  38]], dtype=uint8)
```

x is representative of an image matrix

a is representative of a 3x3 averaging filter

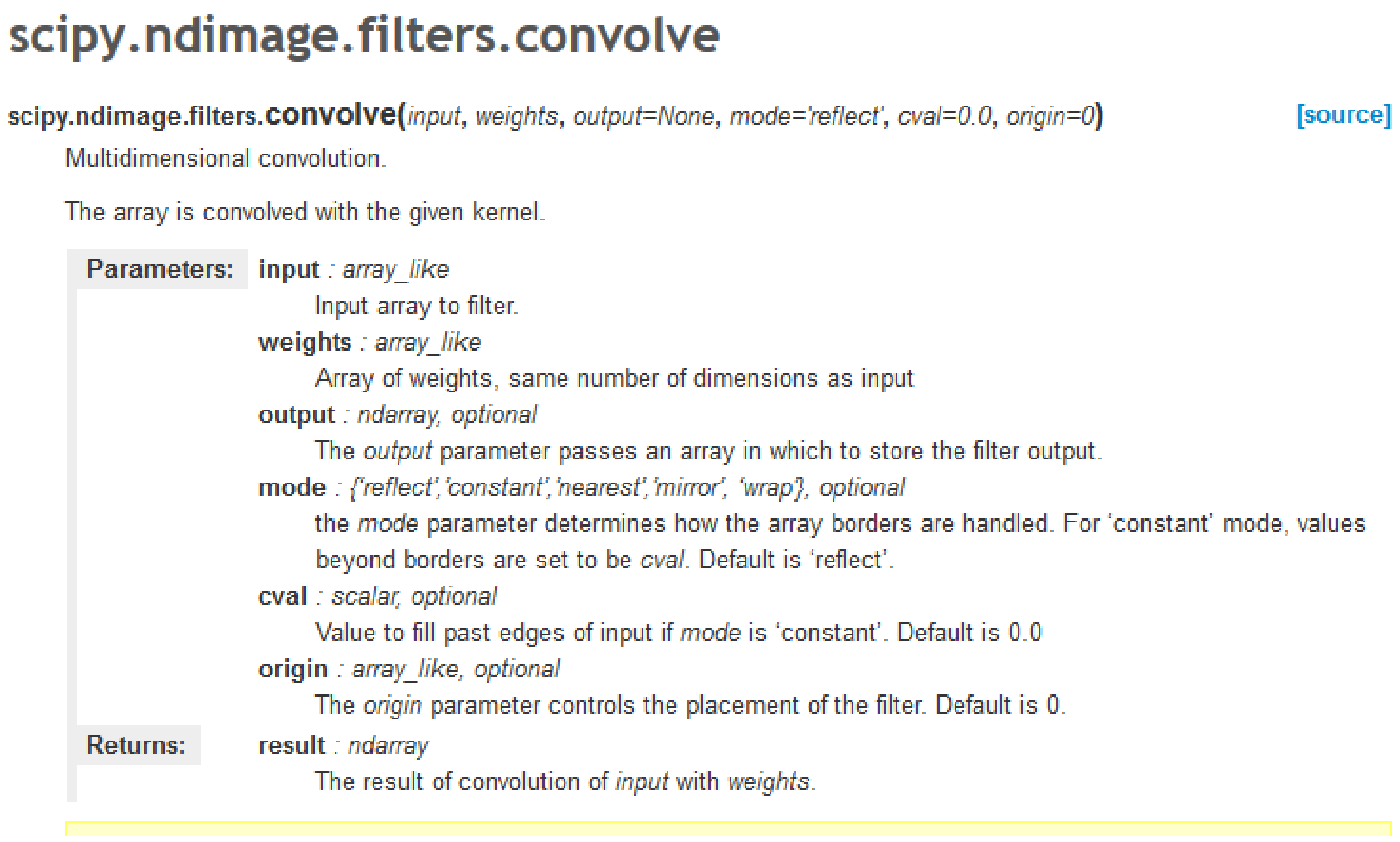
Filtering in Python

MIDWESTERN STATE UNIVERSITY

Try it:

```
import skimage.io as io
import scipy.ndimage as ndi
import numpy as np
import matplotlib.pyplot as plt

c = io.imread('cameraman.png')
cf = ndi.convolve(c, np.ones((9, 9))/81)
fig, (ax1, ax2) = plt.subplots(ncols=2)

ax1.imshow(c, cmap='gray')
ax2.imshow(cf, cmap='gray')
```

Filtering in Python



scipy.ndimage.filters.convolve

scipy.ndimage.filters.**convolve**(*input*, *weights*, *output=None*, *mode='reflect'*, *cval=0.0*, *origin=0*) [source]

Multidimensional convolution.

The array is convolved with the given kernel.

Parameters:

input : *array_like*
Input array to filter.

weights : *array_like*
Array of weights, same number of dimensions as input

output : *ndarray, optional*
The *output* parameter passes an array in which to store the filter output.

mode : *{'reflect','constant','nearest','mirror', 'wrap'}, optional*
the *mode* parameter determines how the array borders are handled. For 'constant' mode, values beyond borders are set to be *cval*. Default is 'reflect'.

cval : *scalar, optional*
Value to fill past edges of input if *mode* is 'constant'. Default is 0.0

origin : *array_like, optional*
The *origin* parameter controls the placement of the filter. Default is 0.

Returns:

result : *ndarray*
The result of convolution of *input* with *weights*.

Frequencies

Frequency: A measure of the amount by which gray (color) values change with distance.

High Frequency: Large changes in gray values over small distances.

- Edges
- Noise

Low Frequency: Little change in the gray values.

- Backgrounds
- Skin textures

High Pass vs. Low Pass Filter

High Pass Filter: Passes over high frequency components, and reduces or eliminates low frequency components

- Edge detection/enhancement
- Example: Laplacian

Low Pass Filter: Passes over low frequency components, and reduces or eliminates high frequency components

- Blurs edges, areas of high contrast
- Example: Averaging filters, Gaussian

High Pass Filter

High pass filter:

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Note: Sum of the coefficients in this high pass filter is zero. In a low frequency part of an image, where the gray values are similar, the result of using this filter is that the corresponding gray values in the new image will be close to zero.

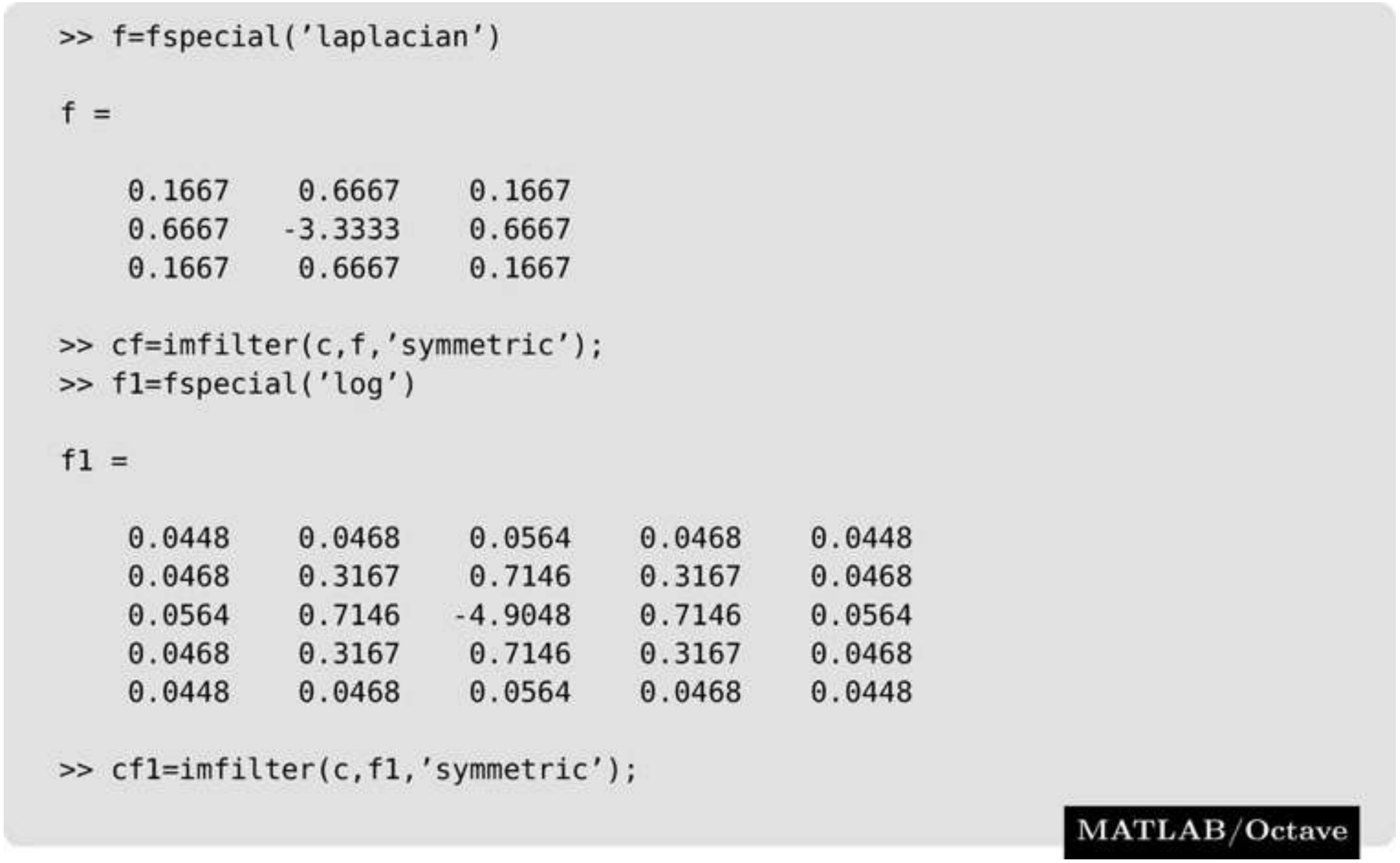
Consider a 4 × 4 block of similar values pixels, and apply the above high pass filter to the central four:

150	152	148	149
147	152	151	150
152	148	149	151
151	149	150	148

$\longrightarrow$

11	6
−13	−5

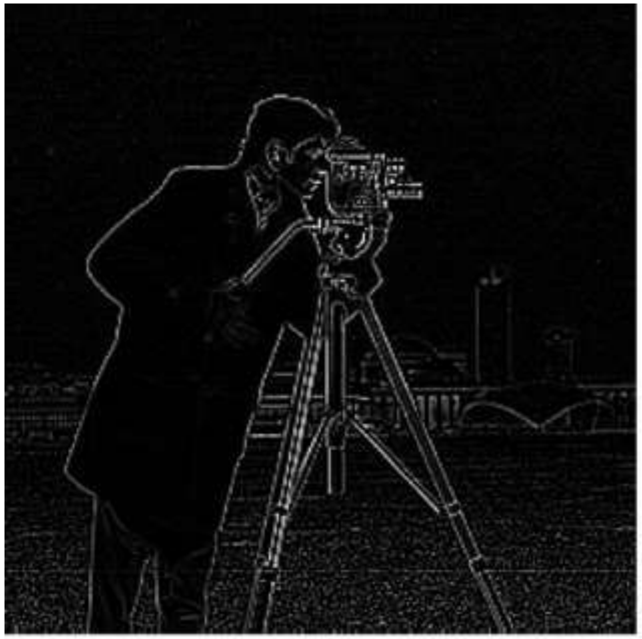
High Pass Filter

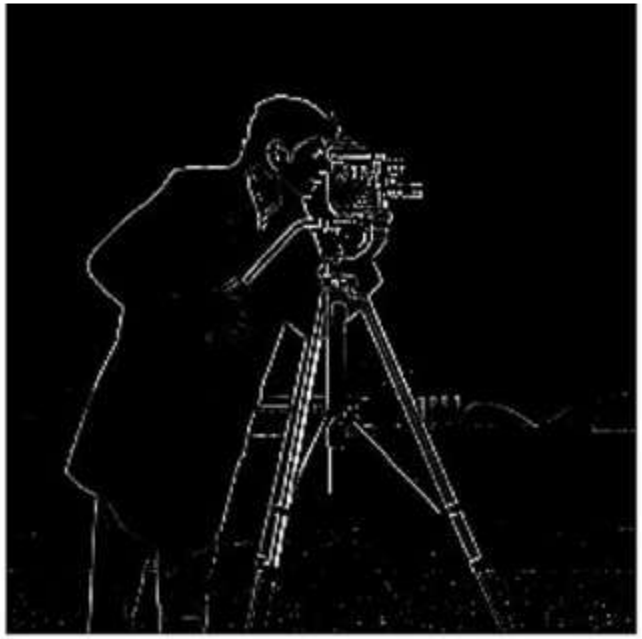


```
>> f=fspecial('laplacian')

f =

    0.1667    0.6667    0.1667
    0.6667   -3.3333    0.6667
    0.1667    0.6667    0.1667

>> cf=imfilter(c,f,'symmetric');
>> f1=fspecial('log')

f1 =

    0.0448    0.0468    0.0564    0.0468    0.0448
    0.0468    0.3167    0.7146    0.3167    0.0468
    0.0564    0.7146   -4.9048    0.7146    0.0564
    0.0468    0.3167    0.7146    0.3167    0.0468
    0.0448    0.0468    0.0564    0.0468    0.0448

>> cf1=imfilter(c,f1,'symmetric');
```

MATLAB/Octave

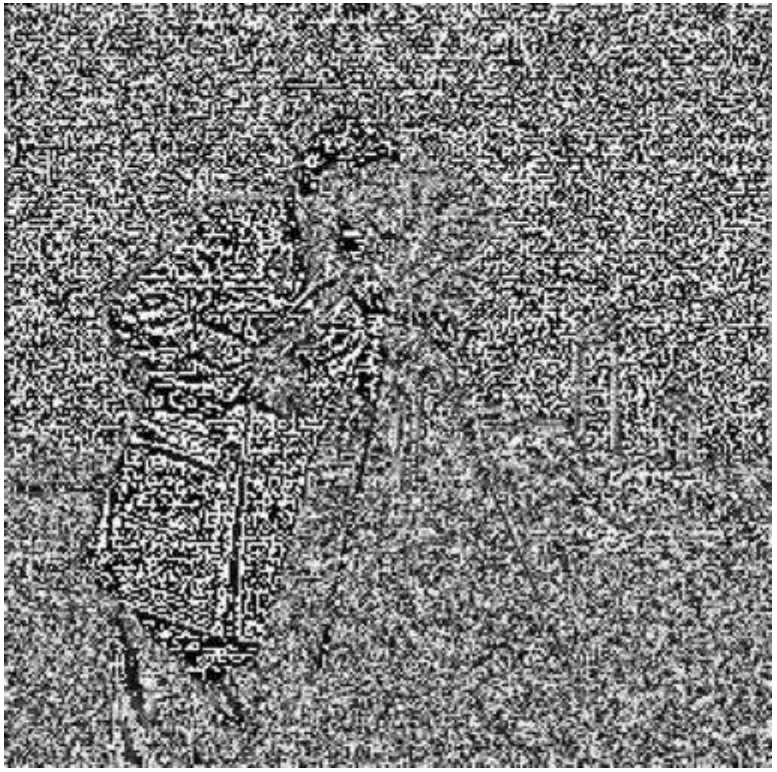
High Pass Filter


(a) Laplacian filter


(b) Laplacian of Gaussian ("log") filtering

Note: MATLAB and Octave "clean up" the raw filtering.

High Pass Filter



Raw filtering without any extra processing

Values Outside Range 0-255

Applying a linear filter may produce values that lie outside 0-255. Solutions include:

Make negative values positive

- Only fixes negative values
- Only used in specific circumstances – only a few negative values and they are close to zero

Clip values

$$y = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x \leq 255 \\ 255 & \text{if } x > 255 \end{cases}$$

- Not suitable if there are many gray values outside the 0–255 range

Scaling transformation

- Lowest gray value gL, highest value is gH
- Transform all values in the range gL – gH to the range 0–255 by the linear transformation:

$$y = 255 \frac{x - g_L}{g_H - g_L}$$

Values Outside Range 0-255

Scaling transformation example

Assume lowest gray value gL = -56
Assume highest gray value gH = 528

Transform the following gray values:

-10, 6, 32, -56, 45, 350, 450, 528, 200

$$y = 255\frac{x - g_L}{g_H - g_L}$$

Values Outside Range 0-255

Scaling transformation example

$$y = 255\frac{x - g_L}{g_H - g_L}$$

Assume lowest gray value gL = -56
Assume highest gray value gH = 528

Transform the following gray values:

-10, 6, 32, -56, 45, 350, 450, 528, 200

Transformed Values:

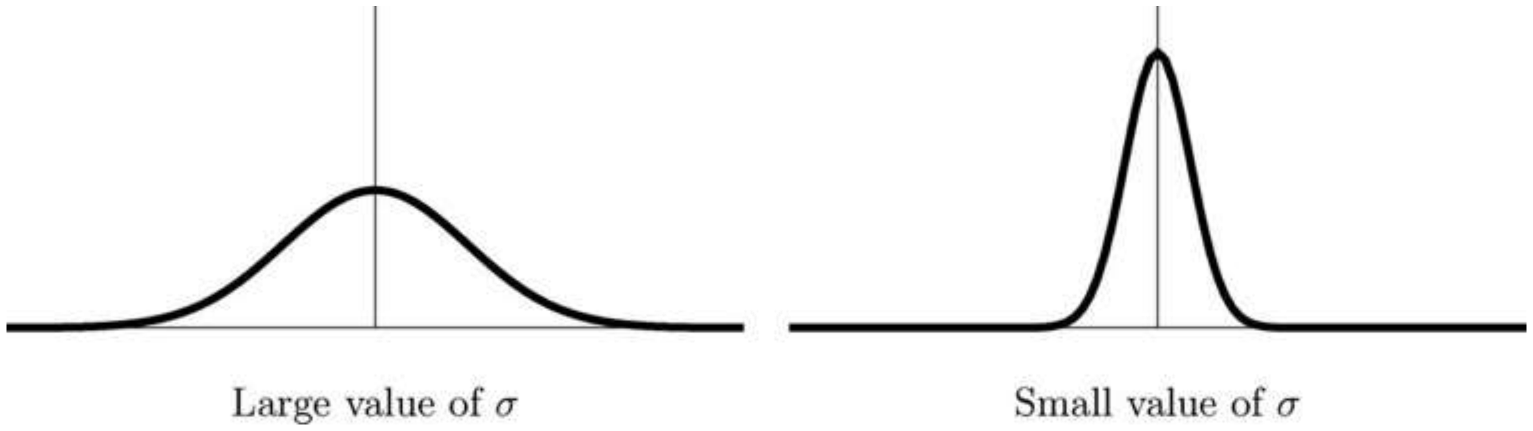
20, 27, 38, 0, 44, 177, 221, 255, 112

Gaussian Filters

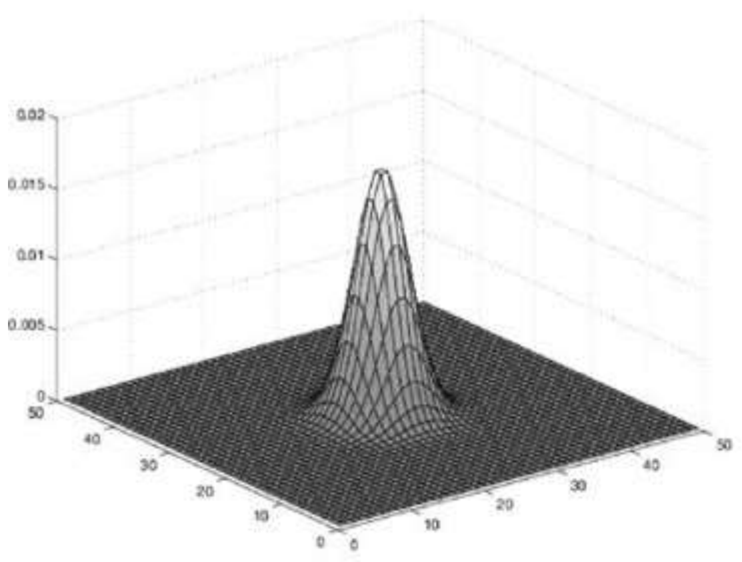
Class of low pass filters based on the Gaussian Probability distribution function

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

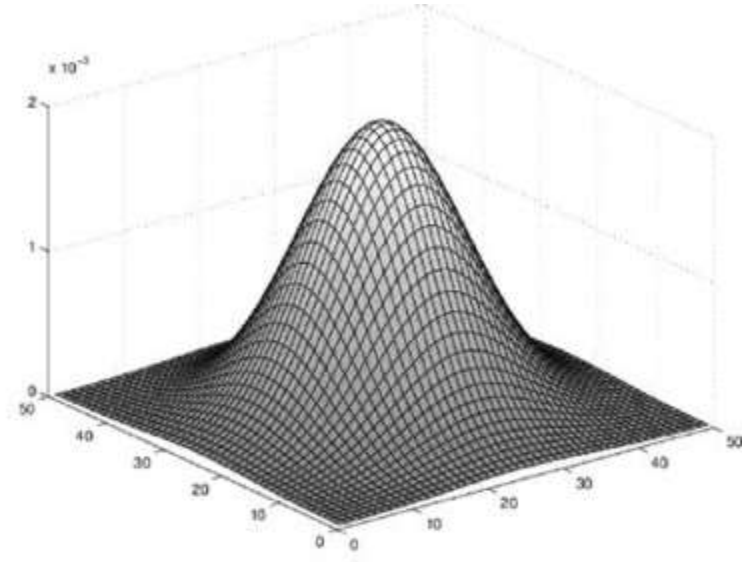
where σ is the standard deviation: a large value of σ produces to a flatter curve, and a small value leads to a “pointier” curve.



Gaussian Filters



$\sigma = 3$



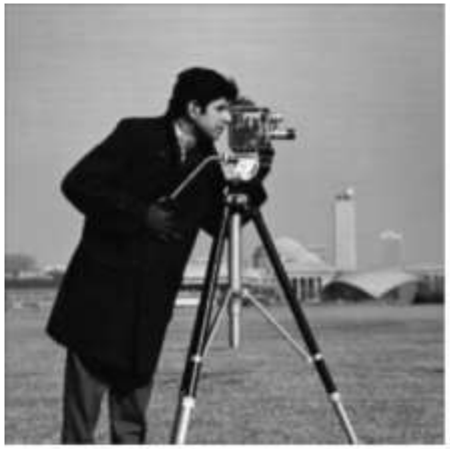
$\sigma = 9$

Gaussian Filters

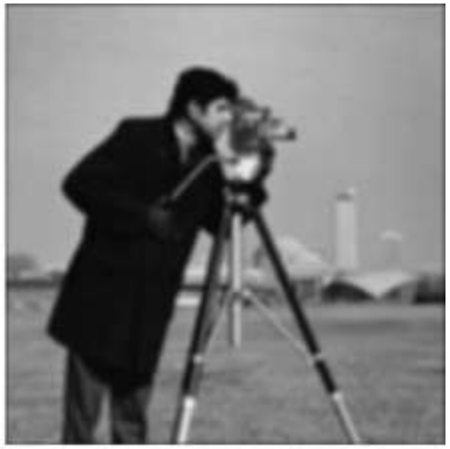
Important because:

1. Mathematically very “well behaved”; in particular the Fourier transform (see Chapter 7) of a Gaussian filter is another Gaussian.
2. Rotationally symmetric - very good starting points for some edge detection algorithms (see Chapter 9),
3. They are separable; in that a Gaussian filter may be applied by first applying a one-dimension Gaussian in the x direction, followed by another in the y direction. This can lead to very fast implementations.
4. The convolution (filter rotated by 180°) of two Gaussians is another Gaussian.

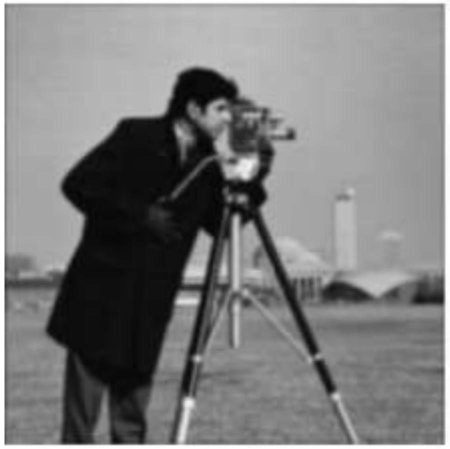
Gaussian Filters



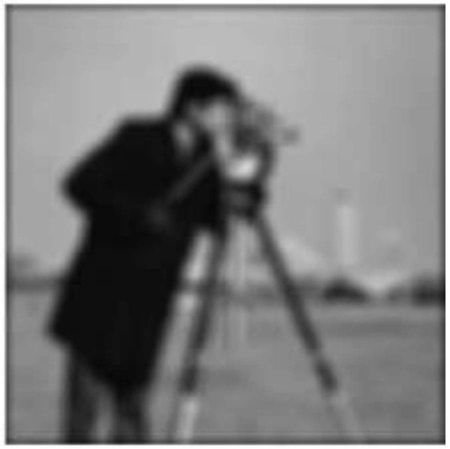
$5 \times 5, \sigma = 0.5$



$5 \times 5, \sigma = 2$



$11 \times 11, \sigma = 1$

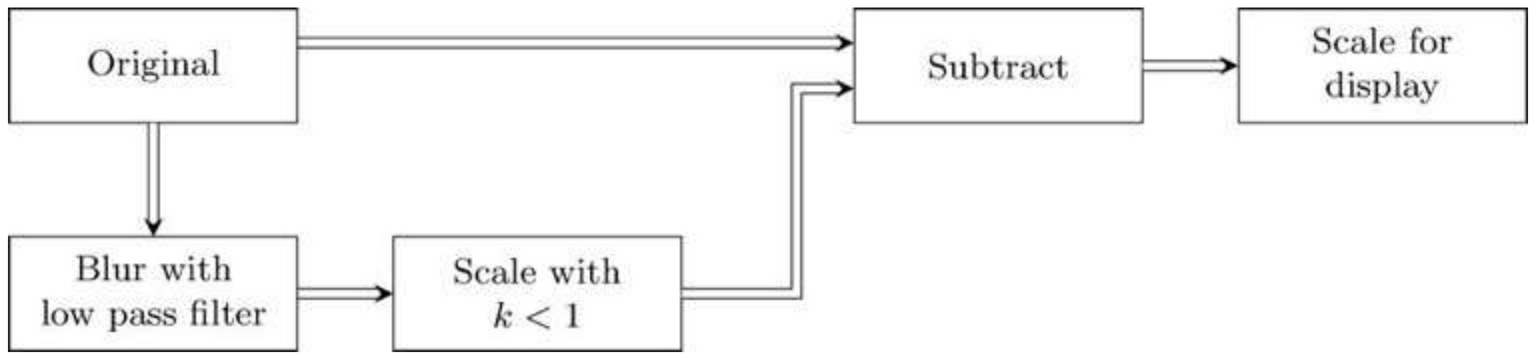


$11 \times 11, \sigma = 5$

Edge Sharpening

Spatial filtering can be used to make edges in an image slightly sharper and crisper

- Generally results in an image more pleasing to the human eye
- “edge enhancement,” “edge crispening,” or “unsharp masking”

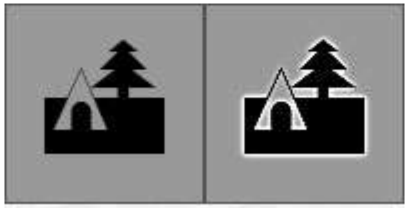


Unsharp??? To sharpen???

The name has its origins, as do many things photographic, in the conventional darkroom.

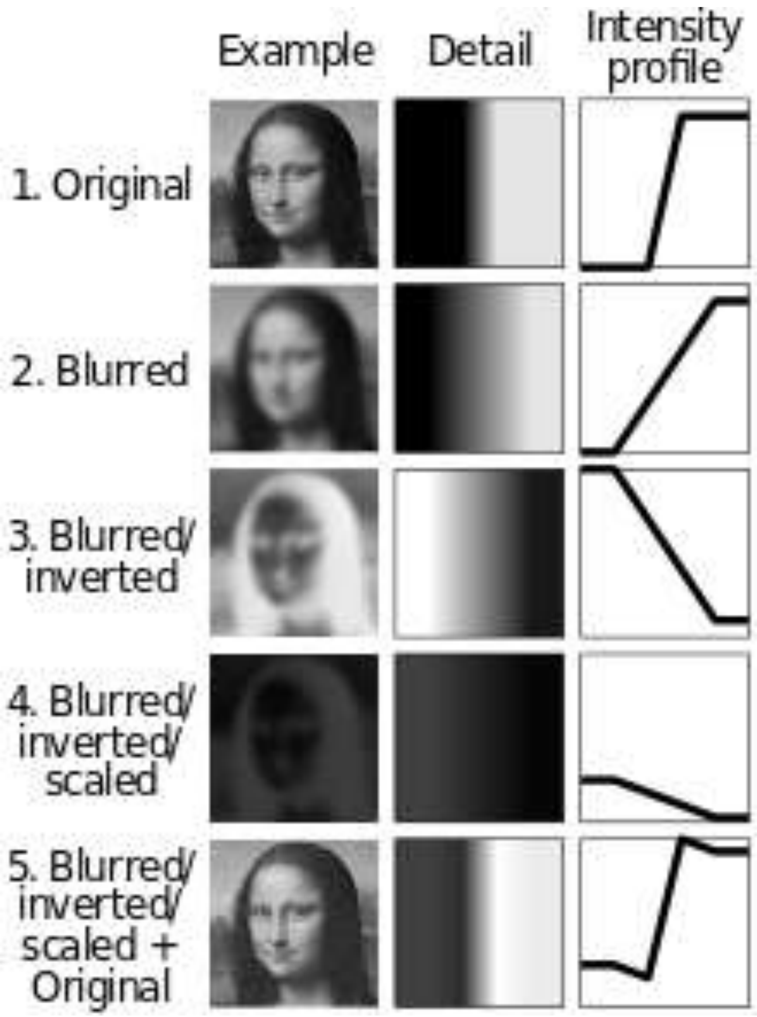
Creating a print the old-fashioned way involves exposing photosensitive paper by projecting onto it a negative of the desired image. In order to accentuate edges in a print and thus make it look sharper, the darkroom operator made something called an unsharp mask.

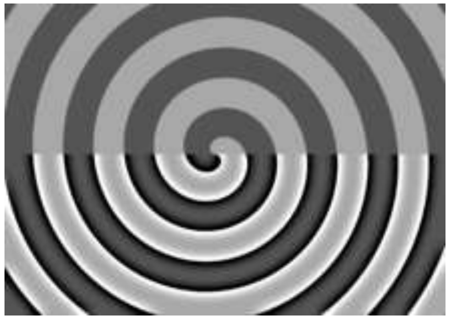
The technique consists of creating a weak, slightly out of focus positive transparency of the original negative. Rather than exposing the negative on photographic paper, it is instead exposed on another sheet of film. Thus, this new version will be a positive image. Since we recorded it slightly out of focus, if we sandwich it with the original piece of film, all the edges in the composite image will have slight halos in the exact opposite color of whatever is on the other side of each edge.

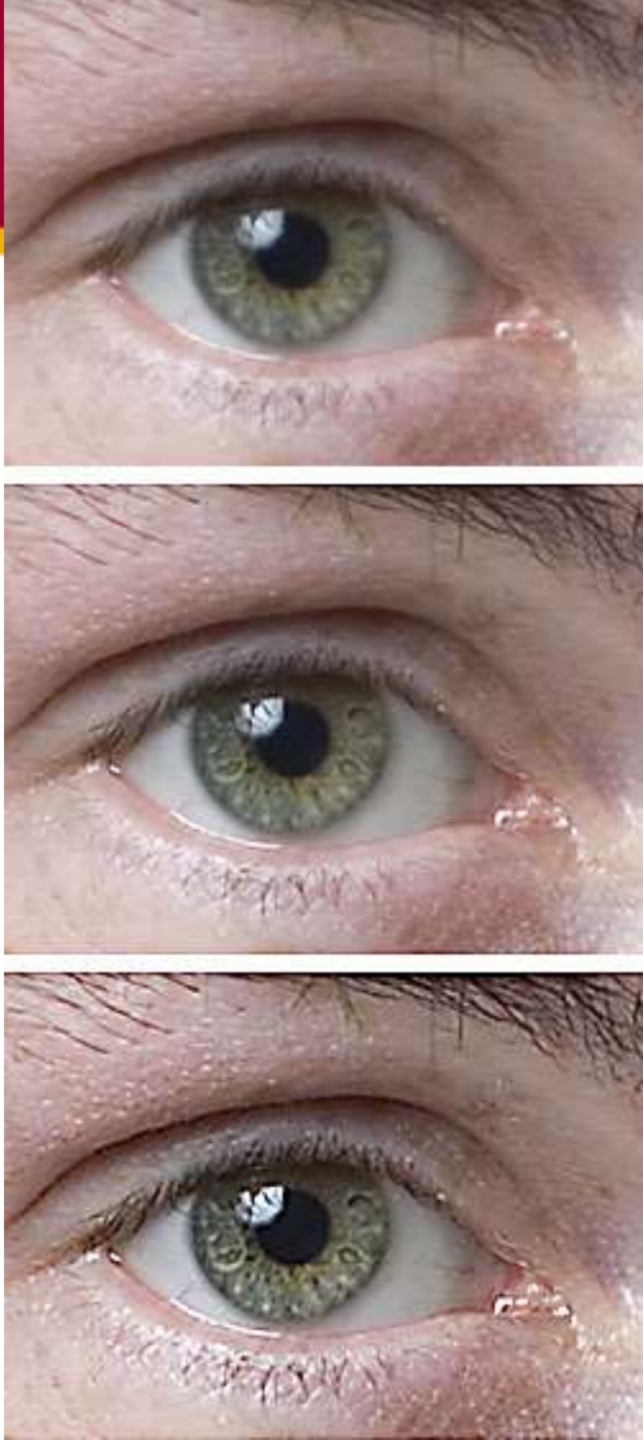

The effect of excessive sharpening

http://www.earthboundlight.com/phototips/unsharp-mask.html

Simplified principle







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

We are in effect subtracting a blur from a scaled version of the original; the scaling factor may also be split between the identity and blurring filters.

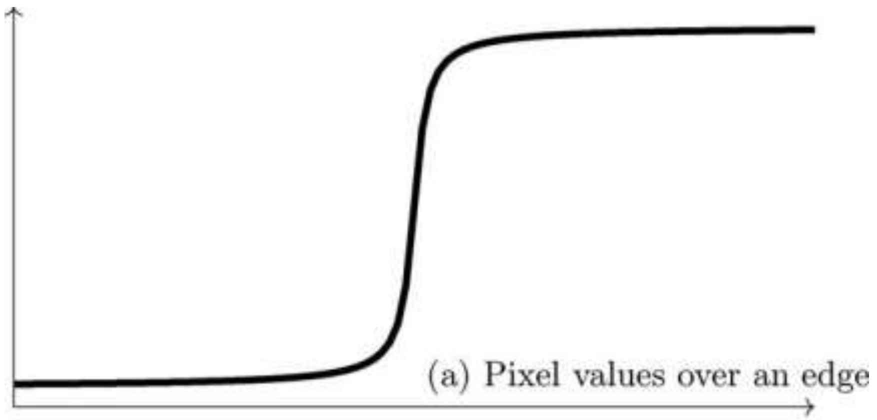
Examples using averaging filter to blur:

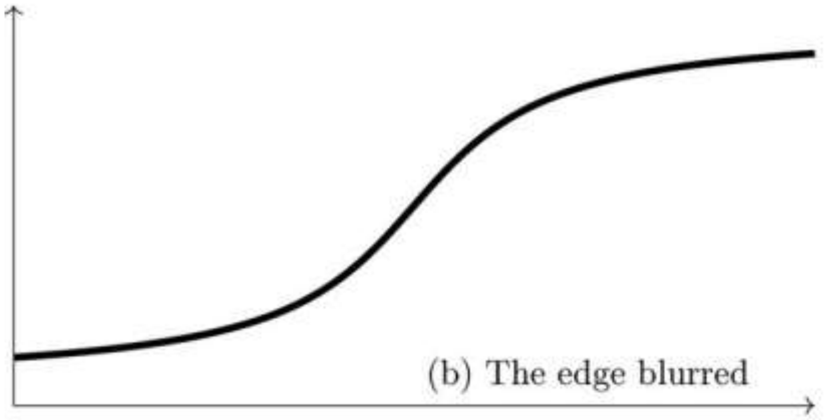
$$f = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{k} \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

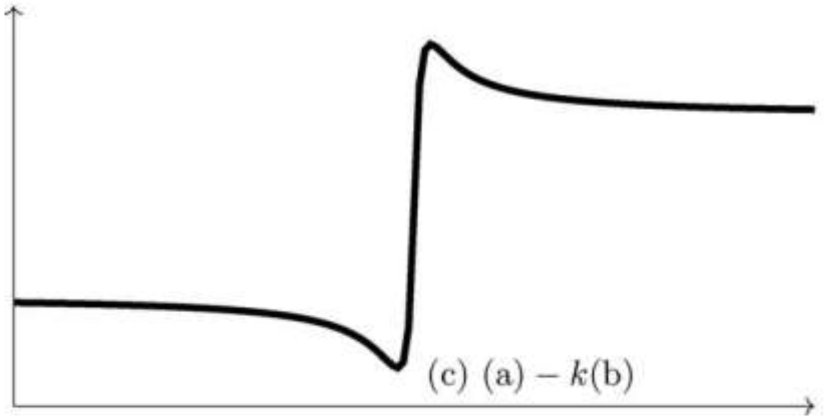
or

$$f = k \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

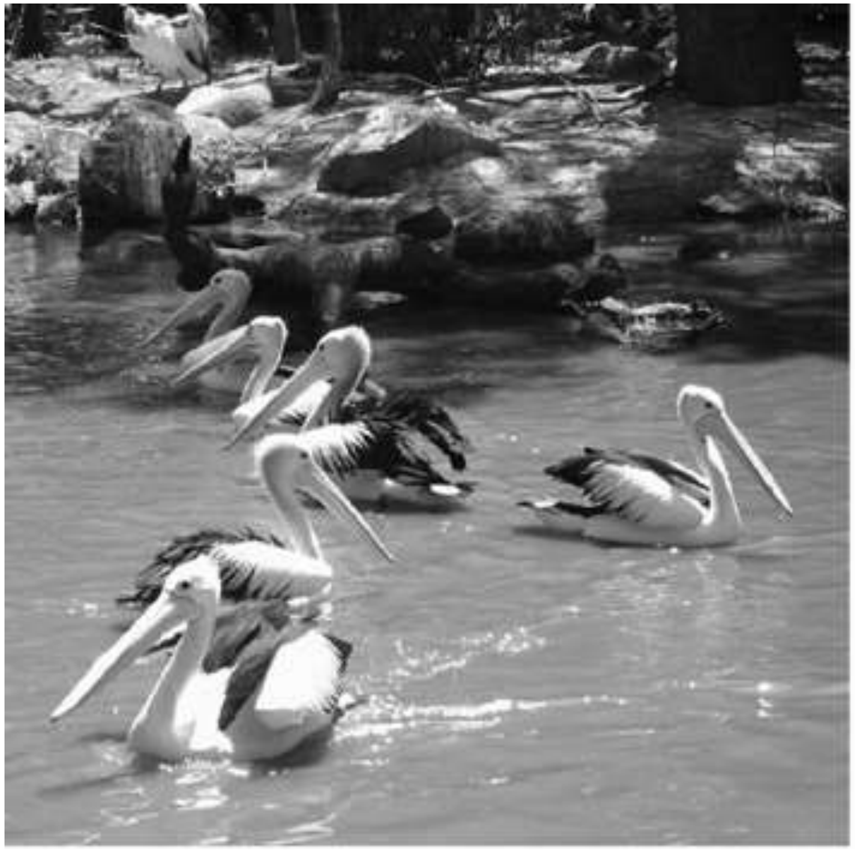
where k is a constant chosen to provide the best result.

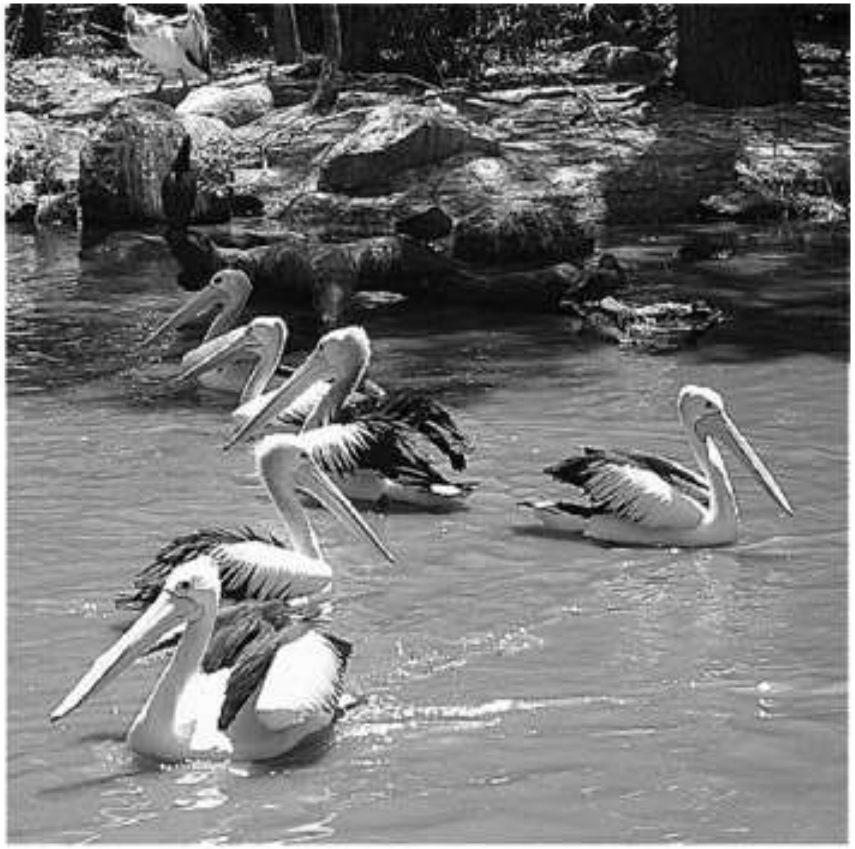

(a) Pixel values over an edge


(b) The edge blurred


(c) (a) − k(b)

Unsharp Mask

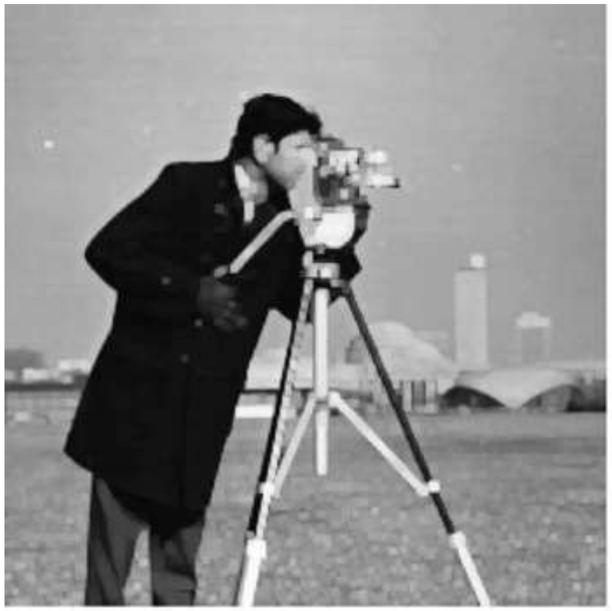

(a) The original

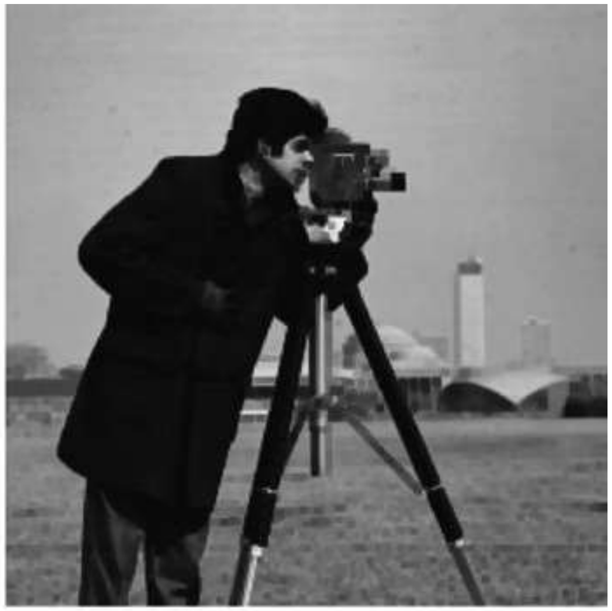

(b) After unsharp masking

Non-linear filters

Non-linear filter: A filter that applies a non-linear function to the grayscale values in the mask.

Example: Max or min filter, which has as its output the max or min value under the mask


(a) Using a maximum filter


(b) Using a minimum filter

Regions of Interest (ROI)

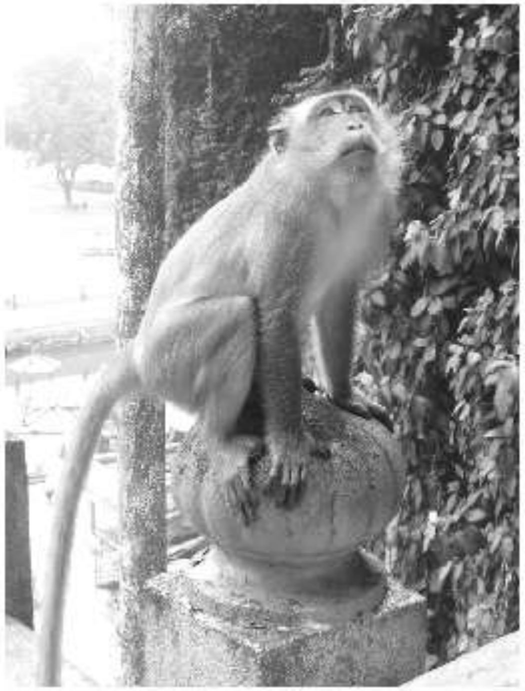
Two ways to define an ROI:

- List the coordinates of a polygonal region
- Define interactively with the mouse

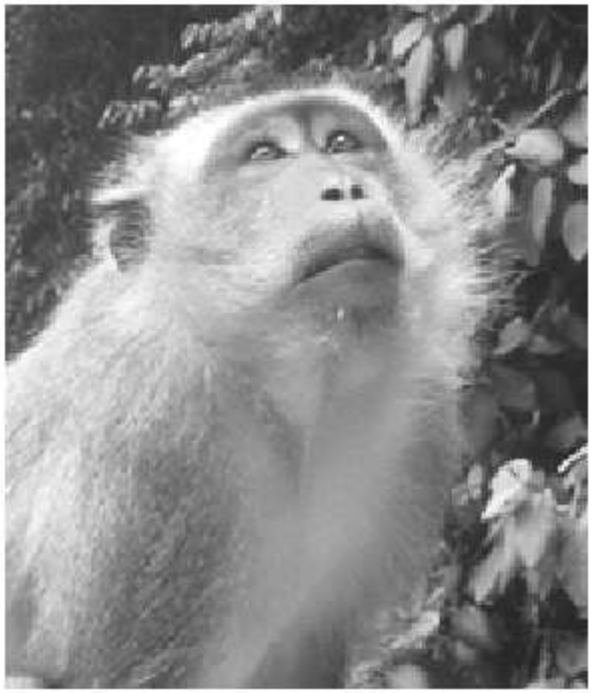
Regions of Interest (ROI)

```
>> m2 = imread('monkey.png');
>> m = m2(56:281,221:412);
```

MATLAB/Octave



imshow(m2)



imshow(m)

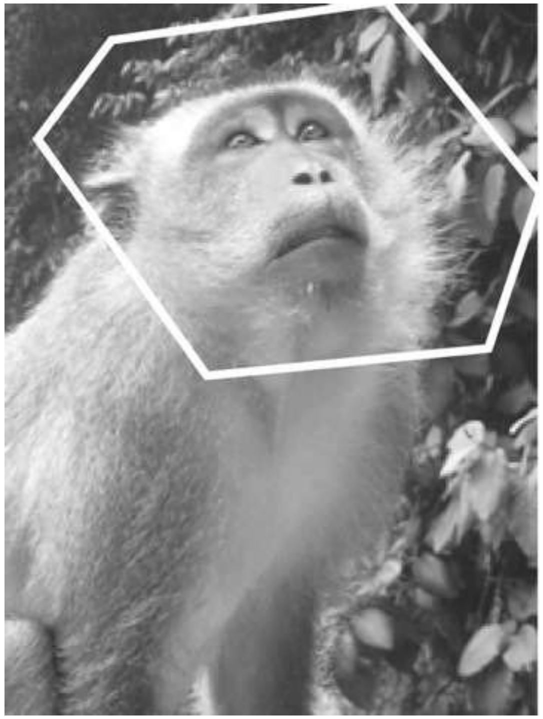
Regions of Interest (ROI)

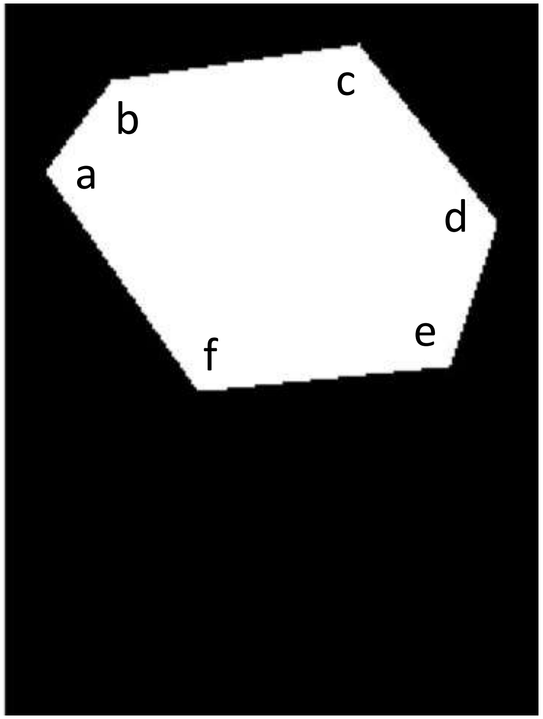
```
>> m2 = imread('monkey.png');
>> m = m2(56:281,221:412);
```

MATLAB/Octave

```
>> xi = [60 27 14 78 130 139]
>> yi = [14 38 127 177 160 69]
>> roi=roipoly(m,yi,xi);
```

$\cancel{\text{MATLAB/Octave}}$





warning: the 'roipoly' function belongs to the image package from Octave Forge
but has not yet been implemented.

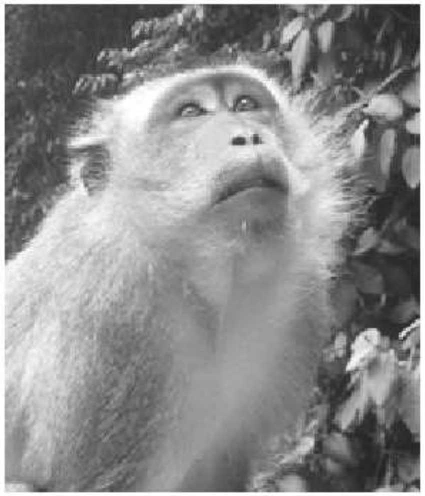
Please read `http://www.octave.org/missing.html' to learn how you can
contribute missing functionality.

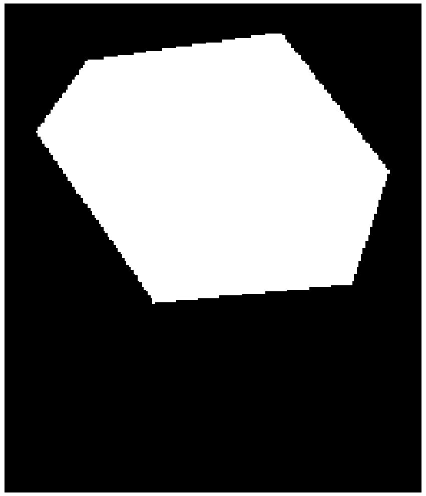
Hexagon Points:

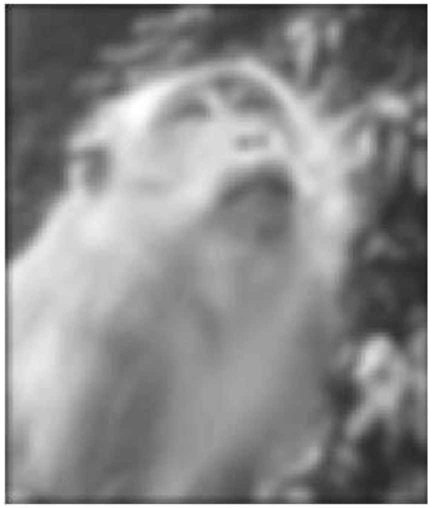
a(60, 14)	b(27, 38),	c(14, 127)
d(78, 177)	e(130, 160)	f(139, 69)

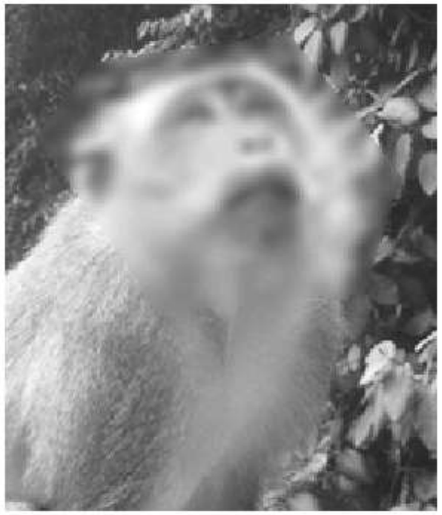
Regions of Interest (ROI)

```
m = imread('monkey_part.png');
[r, c] = size(m);
xi = [60 27 14 78 130 139];
yi = [14 38 127 177 160 69];
roi = poly2mask(yi, xi, r, c);
f = fspecial('gaussian', 9, 3);
mg = imfilter(m, f);
mr = imadd(mg.*roi, m.*~roi);
```


imshow(m)


imshow(roi)


imshow(mg)


imshow(mr)