Image filtering

$$g[m,n] = \sum_{k,l} I(m+k, n+l) * f(k,l)$$

Image I 8x8

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Kernel f 3x3

1	2	3
4	5	6
7	8	9

Output g

28	39	39	39	39	39	39	24
33	45	45	45	45	45	45	27
33	45	45	45	45	45	45	27
16	21	21	21	21	21	21	12
5	6	6	6	6	6	6	3
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Same position

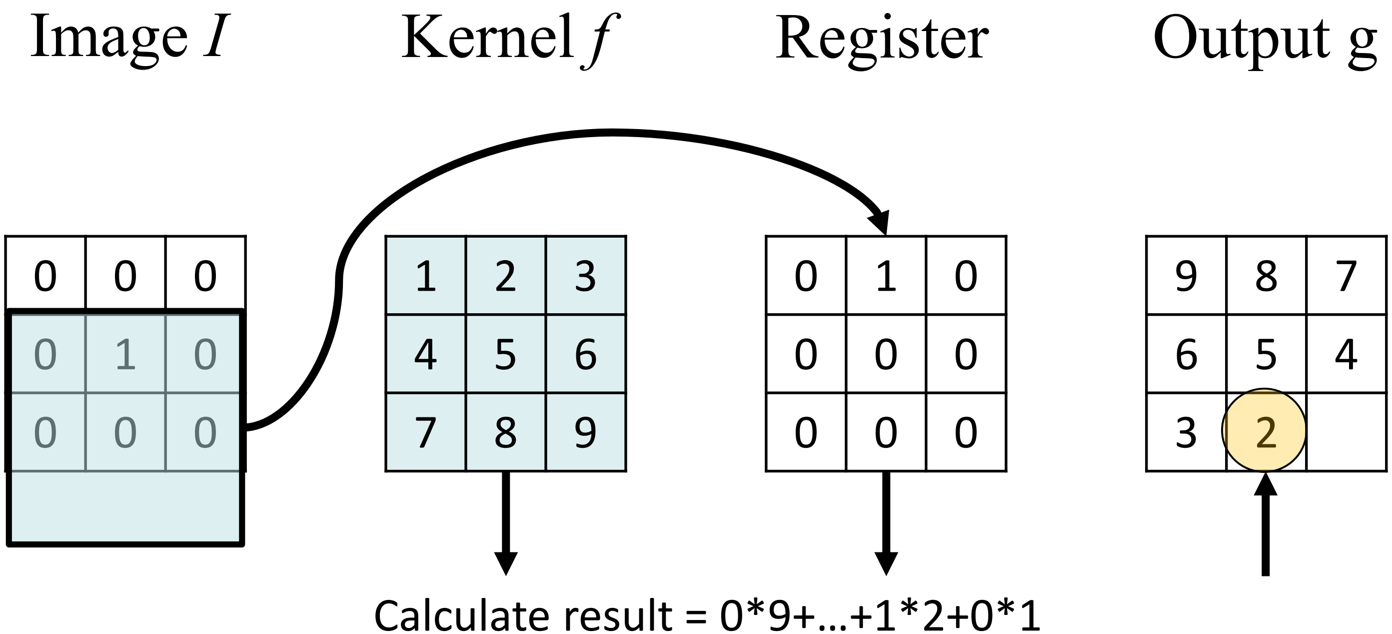
Register

a	b	c
d	e	f
g	h	i

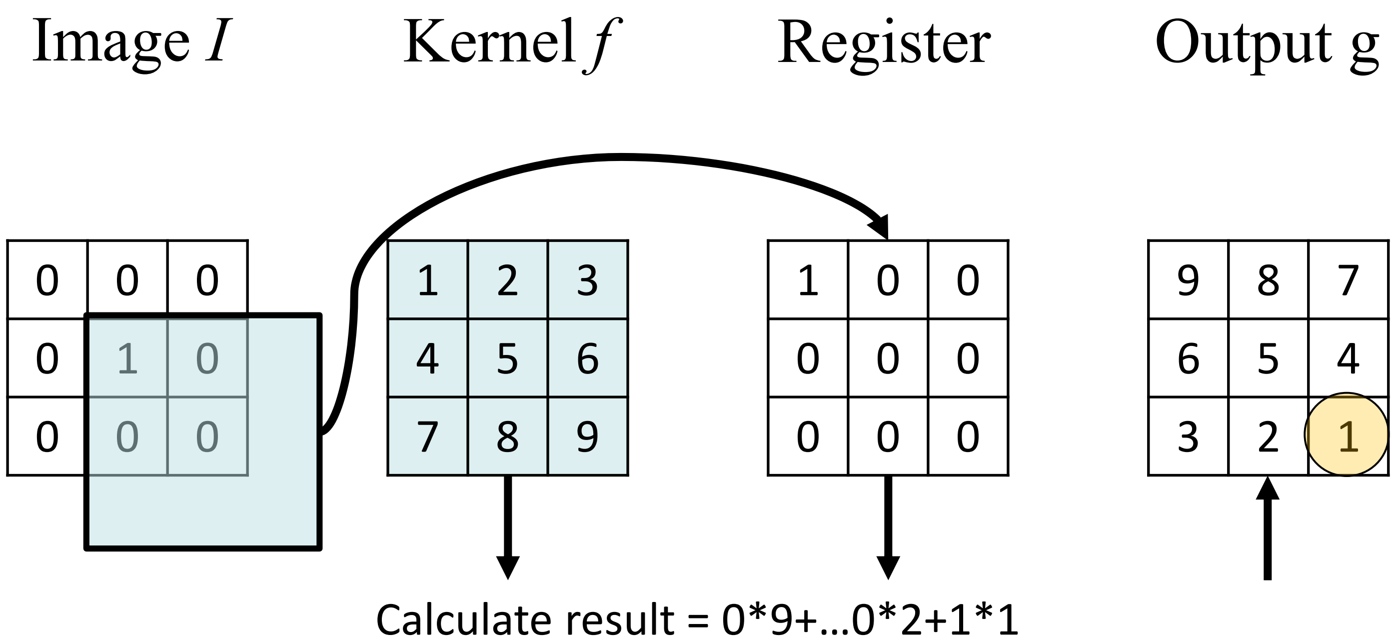
Loop over every pixel

Calculate result = a*1+b*2+...+i*9

Special case: impulse function



Special case: impulse function



<Note> The output is the kernel flipped left-right, up-down!

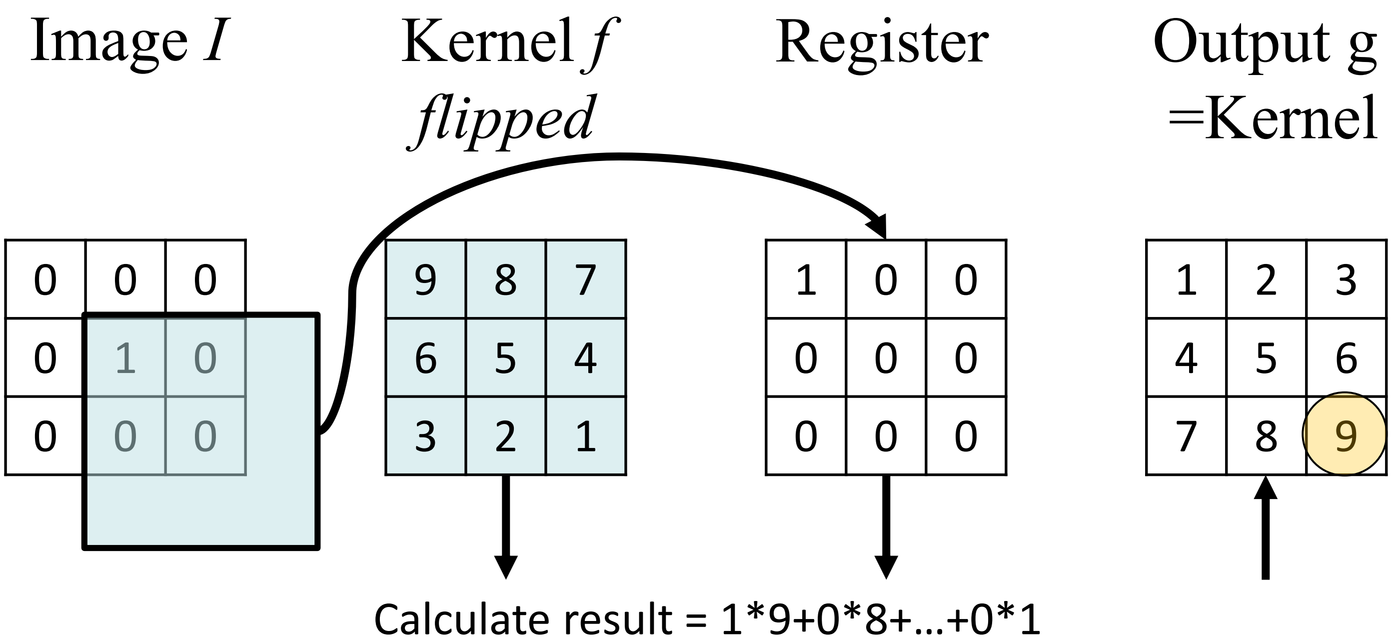
Convolution

- Let I be an Signal(image), Convolution kernel g,

$$f[m,n] = I \otimes g = \sum_{k,l} I[m-k, n-l]g[k,l]$$

Convolution

- $g[m,n] = I \otimes f = \sum_{k,l} I(m-k, n-l) * f(k,l)$
- Convolution is filtering with kernel flipped



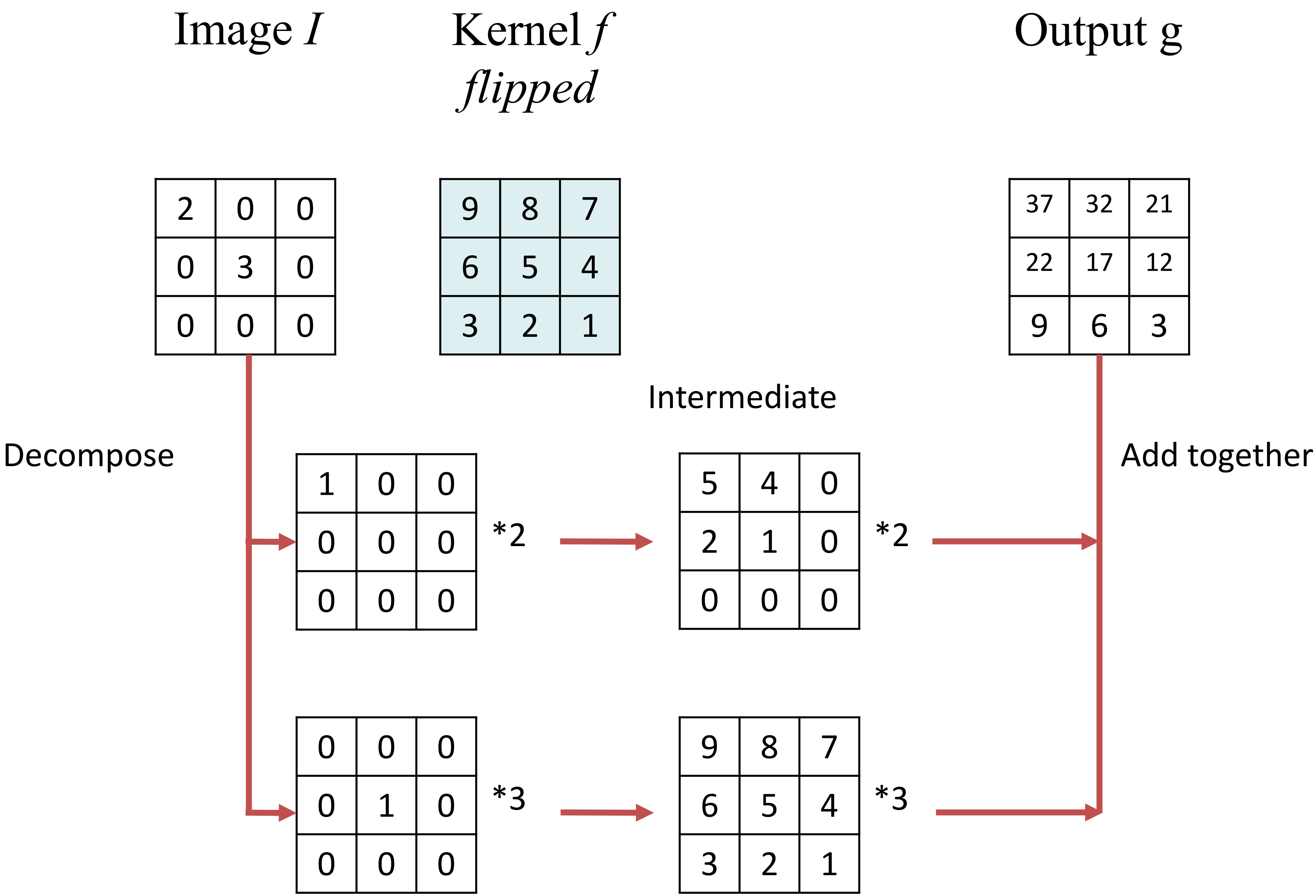


Image I
Kernel f flipped
Output g
2 0 0
0 3 0
0 0 0
9 8 7
6 5 4
3 2 1
37 32 21
22 17 12
9 6 3
Intermediate
Decompose
Add together
1 0 0
0 0 0
0 0 0
*2
5 4 0
2 1 0
0 0 0
*2
0 0 0
0 1 0
0 0 0
*3
9 8 7
6 5 4
3 2 1
*3

- Convolution has commutative property $I \otimes f$

Image I

a	b	c
d	e	f
g	h	i

Kernel f

1	0	0
1	0	0
1	1	0

- Convolution has commutative property $I \otimes f$

Image I

a	b	c
d	e	f
g	h	i

Kernel f

1	0	0
1	0	0
1	1	0

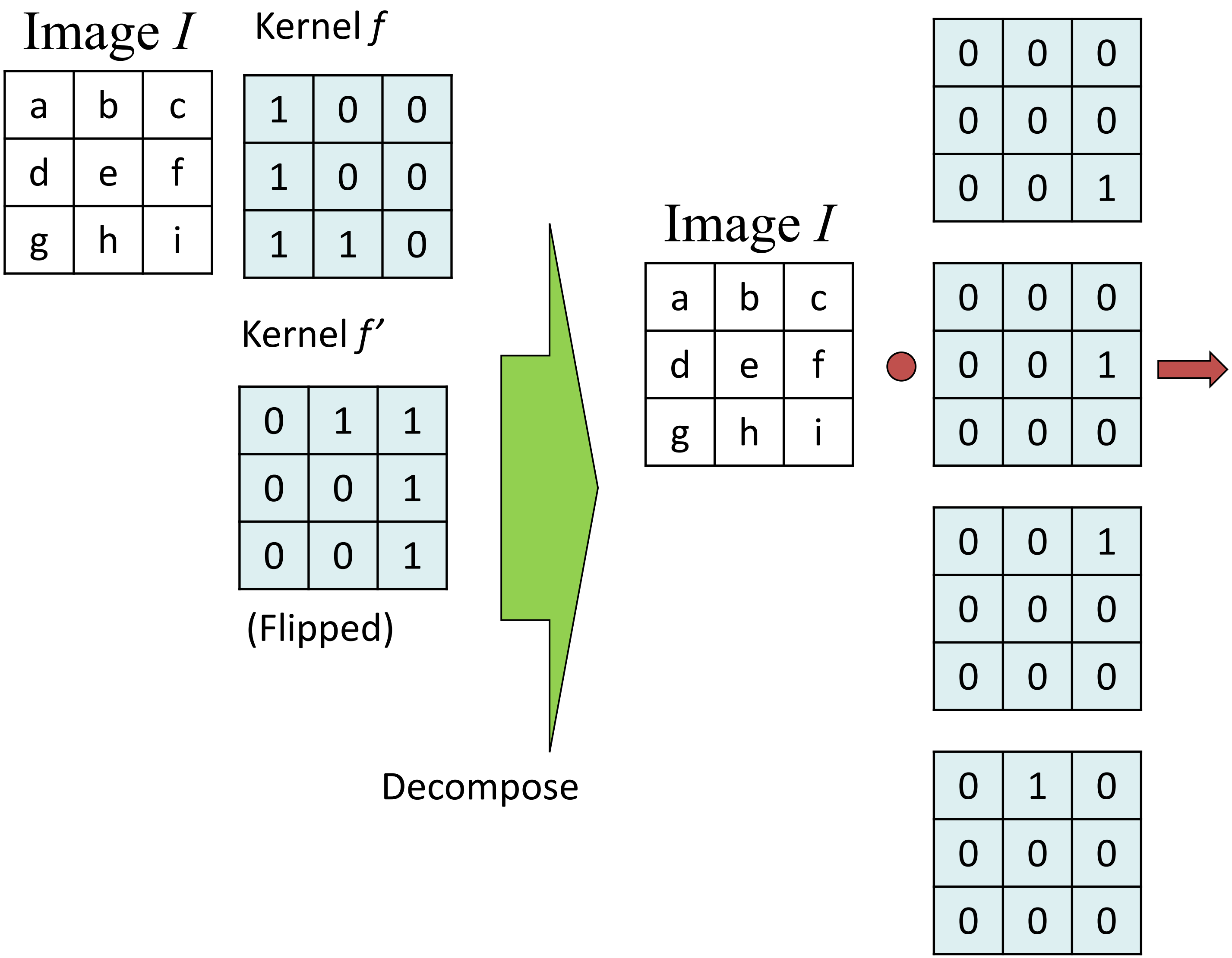
Kernel f'

0	1	1
0	0	1
0	0	1

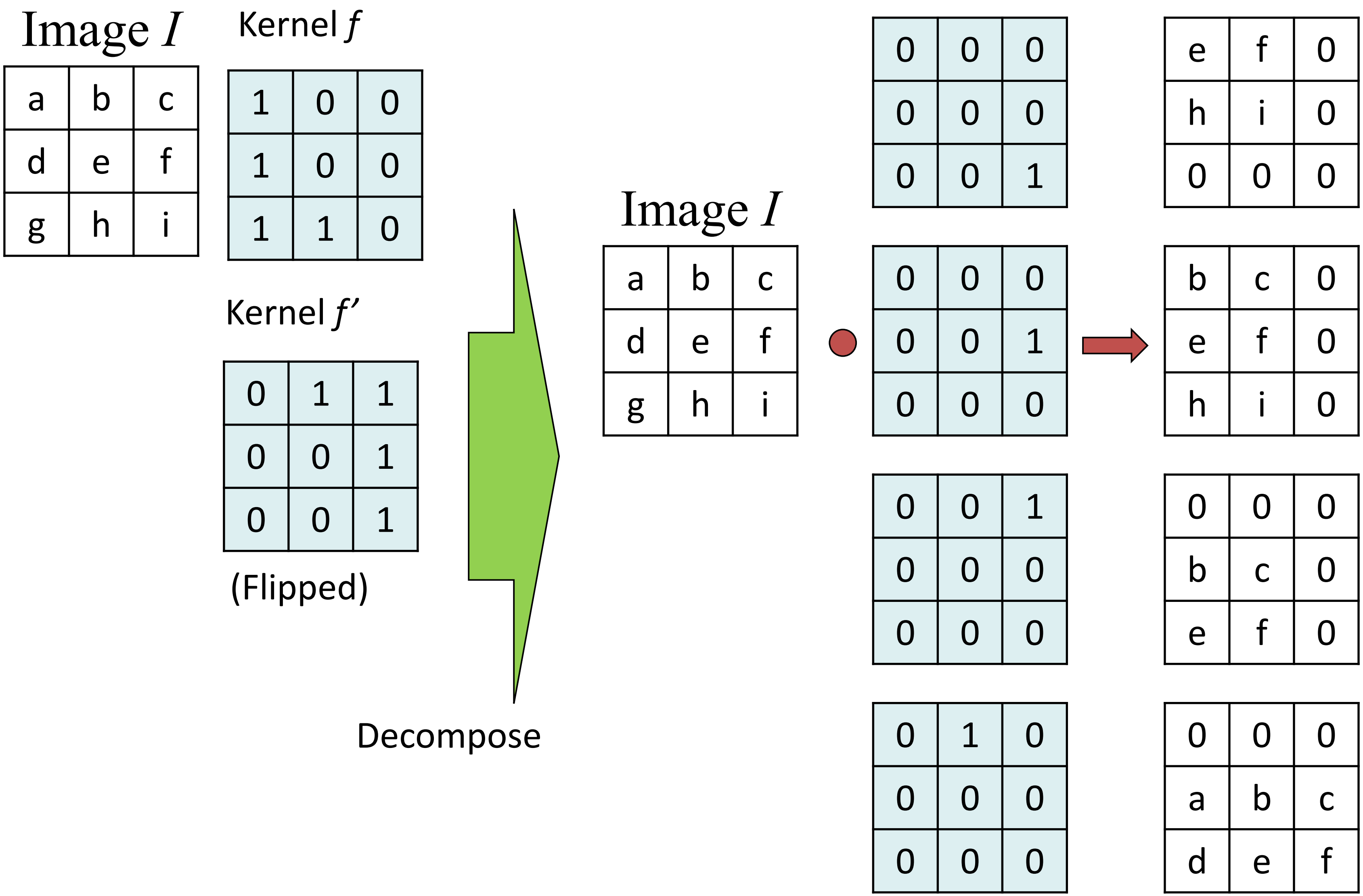
(Flipped)

Decompose

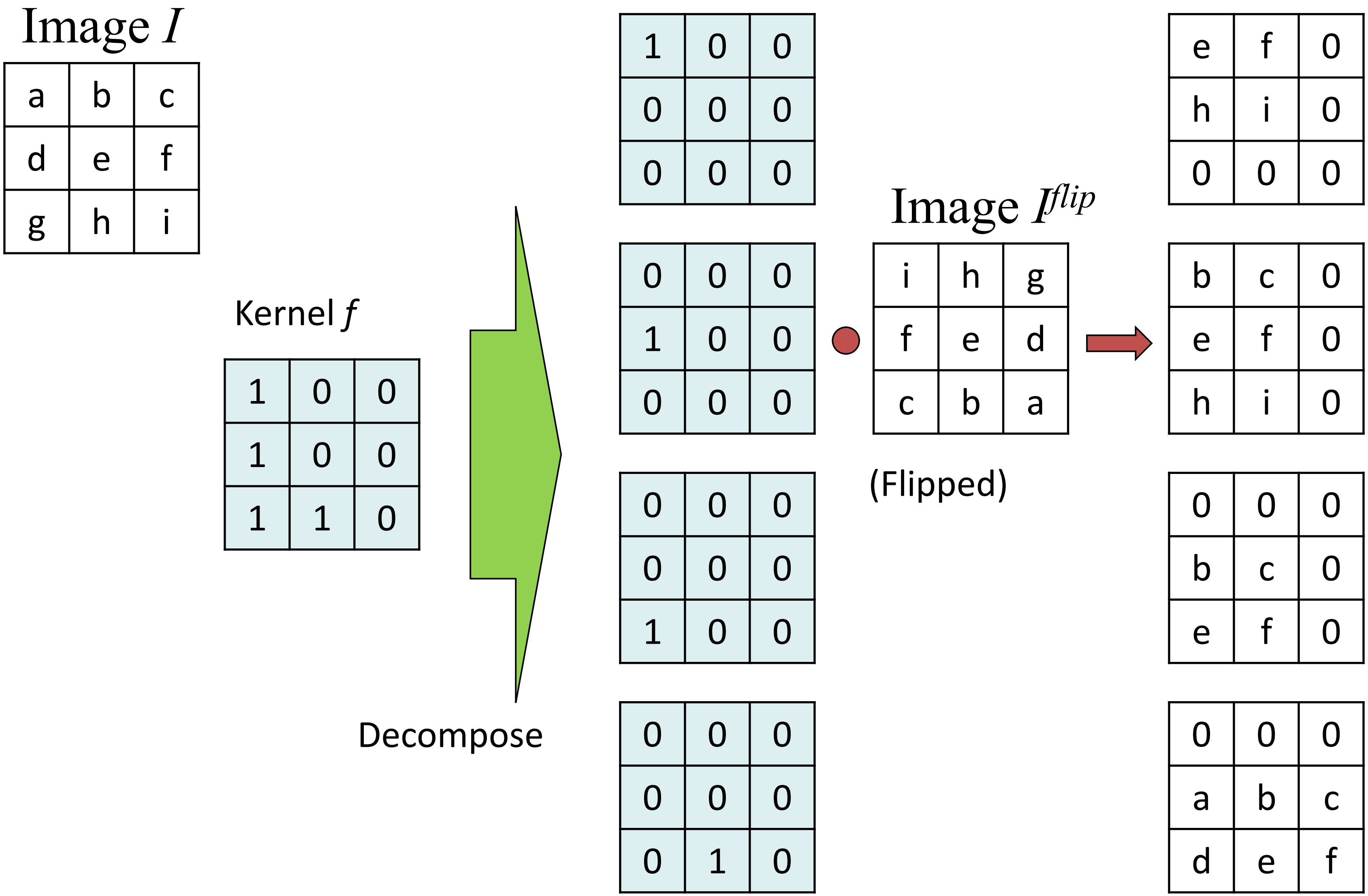
- Convolution has commutative property $I \otimes f$



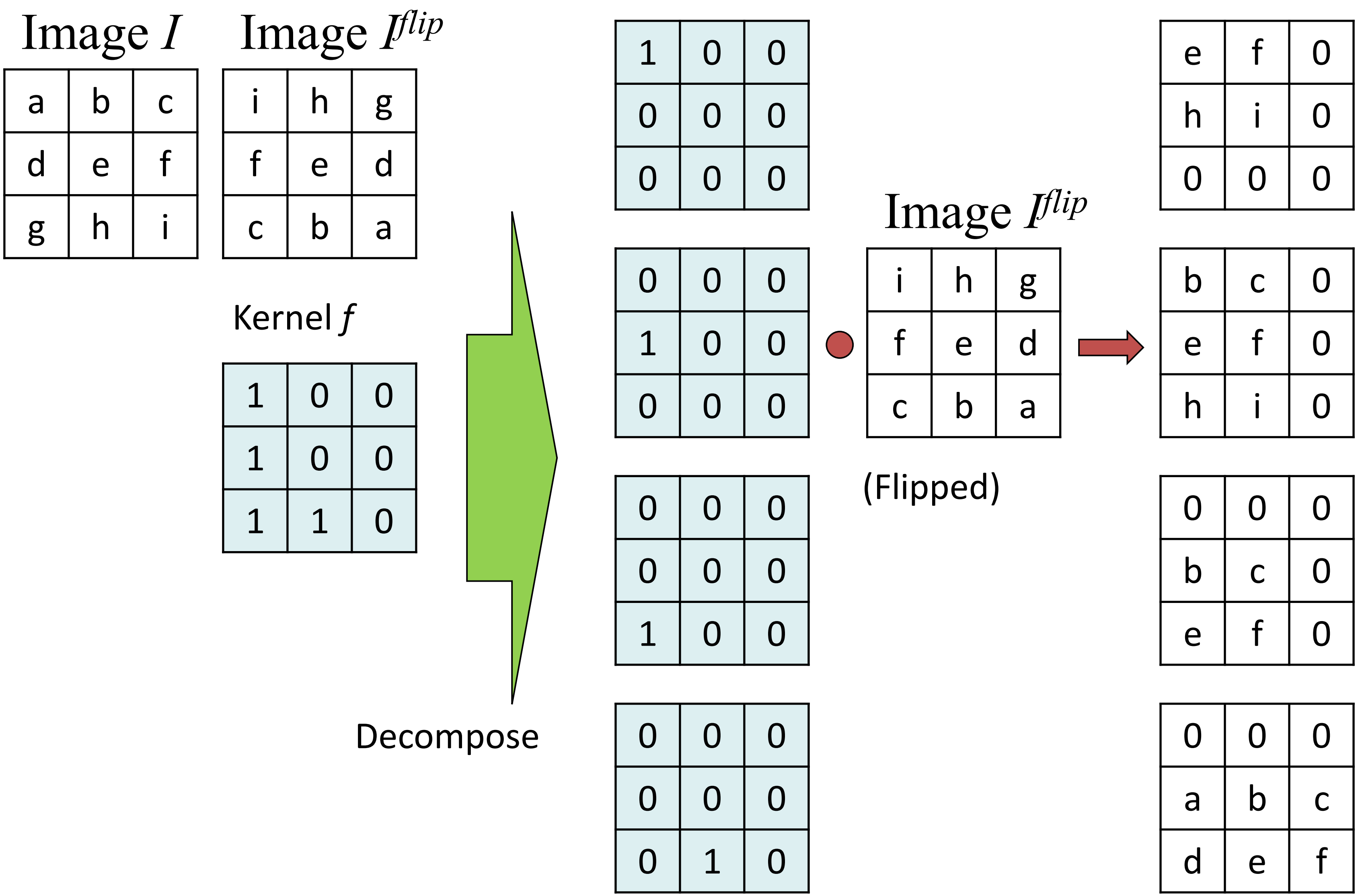
- Convolution has commutative property $I \otimes f$



- Convolution is commutative $I \otimes f = f \otimes I$



- Convolution is commutative $I \otimes f = f \otimes I$



Proof of Commutative property

- $g[m,n] = I \otimes f = f \otimes I$
- $g[m,n] = I \otimes f = \sum_{k,l} I(m-k, n-l) * f(k,l)$
- Let $k' = m-k, l' = n-l$,
  then $k = m-k', l = n-l'$
- $g[m,n] = \sum_{k',l'} I(k',l') * f(m-k', m-l') = f \otimes I$

Impulse functions shift images

Image I

a	b	c
d	e	f
g	h	i

Kernel f

1	0	0
0	0	0
0	0	0

Kernel f'

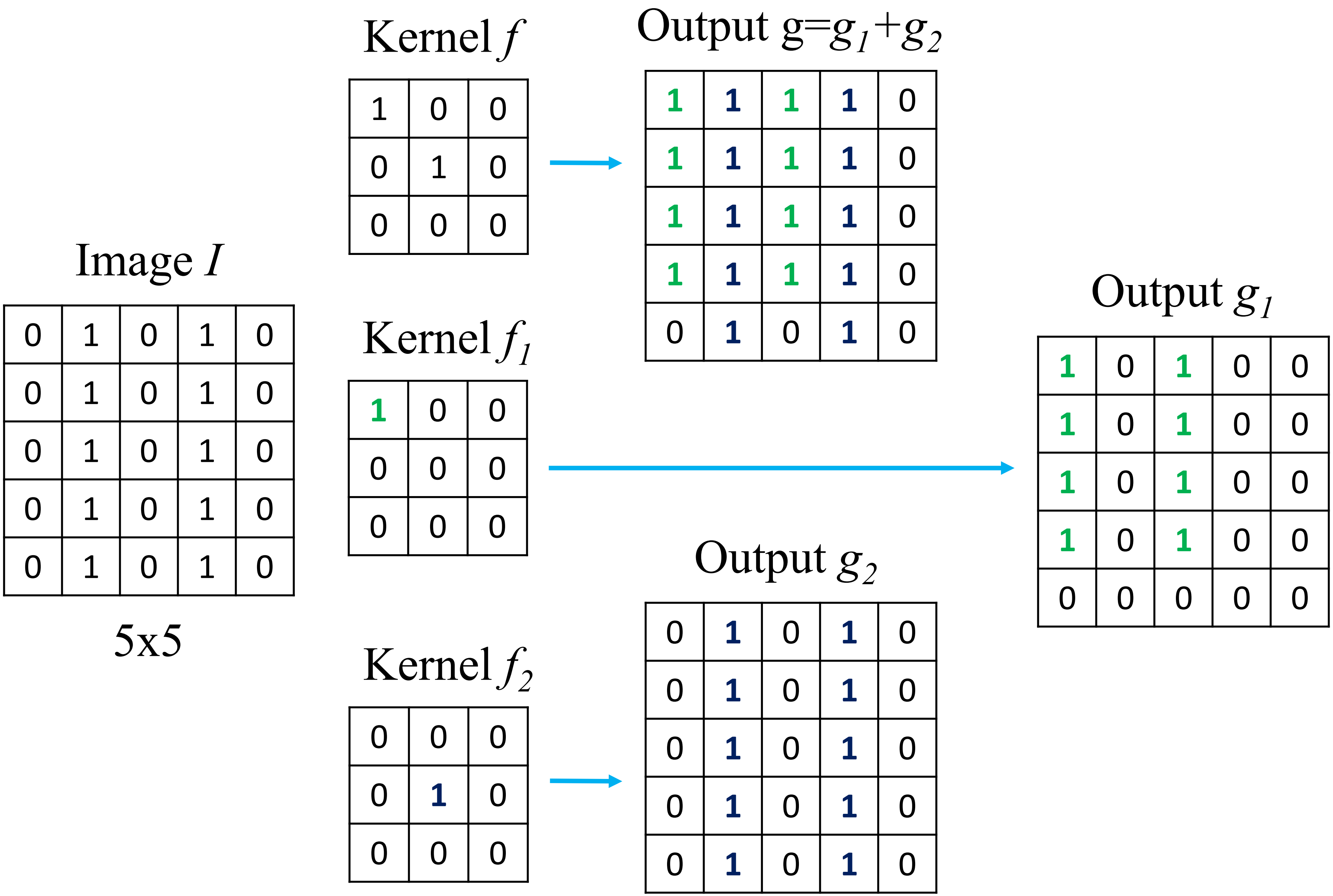
0	0	0
0	0	0
0	0	1

Result $I \otimes f$

e	f	0
h	i	0
0	0	0

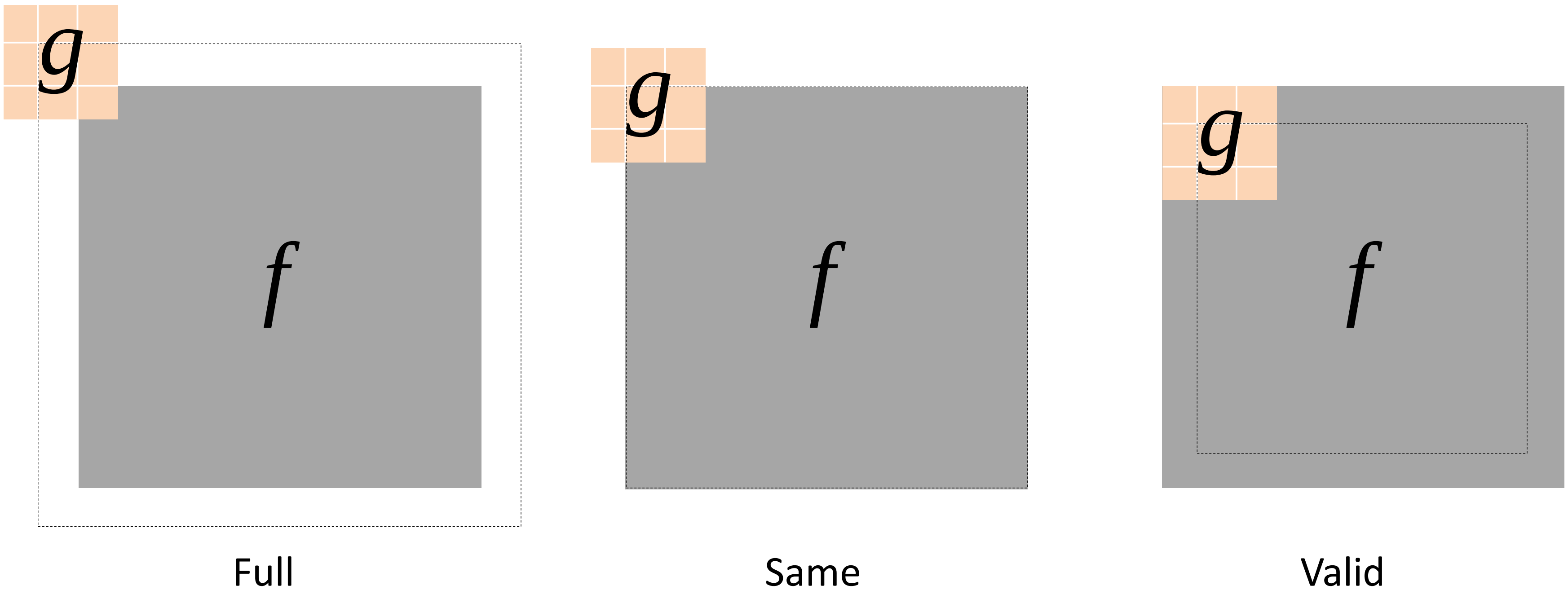
- In this case the resulting image shifted to the upper left

Linear independence



Output Size of Image Convolution

$f \otimes g$



filter2(g, f, shape) in MATLAB

Full: output_size = f_size + g_size – 1

Same: output_size = f_size

Valid: output_size = f_size – (g_size – 1)

2D visualization of convolution (full)

Image *I*

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

8x8

Kernel *f*

1	2	3
4	5	6
7	8	9

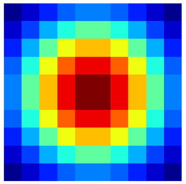
3x3

Output *g*

1	3	6	6	6	6	6	6	5	3
5	12	21	21	21	21	21	21	16	9
12	27	45	45	45	45	45	45	33	18
12	27	45	45	45	45	45	45	33	18
11	24	39	39	39	39	39	39	28	15
7	15	24	24	24	24	24	24	17	9
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

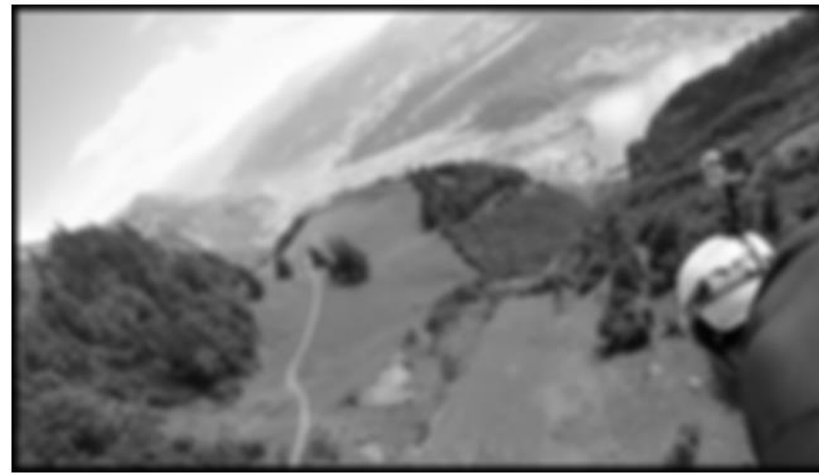
10x10

Output Size of Image Convolution



g : 10 x 10 Gaussian kernel

f : 640 x 360 resolution



Full

filter2(g, f, shape) in MATLAB

Full: output_size = f_size + g_size – 1

```
>> full = filter2(g, im, 'full');
>> size(full)

ans =

   369   649
```

2D visualization of convolution (same)

Image I

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

8x8

Kernel f

1	2	3
4	5	6
7	8	9

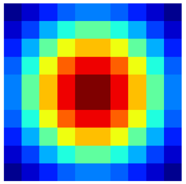
3x3

Output g

12	21	21	21	21	21	21	16
27	45	45	45	45	45	45	33
27	45	45	45	45	45	45	33
24	39	39	39	39	39	39	28
15	24	24	24	24	24	24	17
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

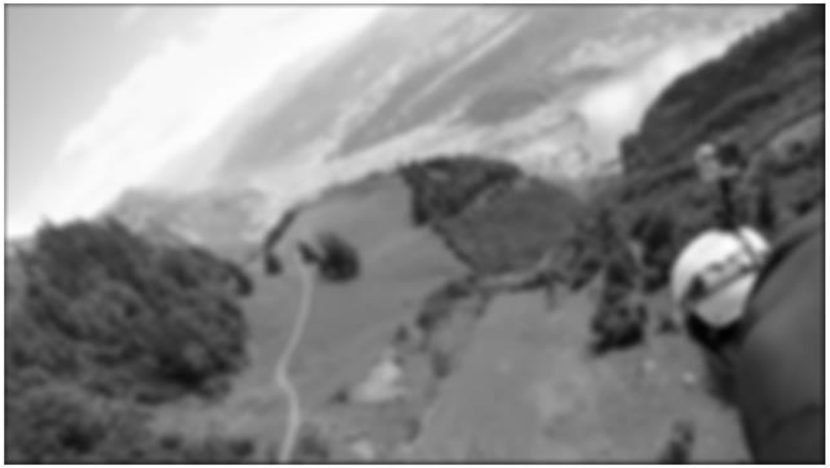
8x8

Output Size of Image Convolution



g : 10 x 10 Gaussian kernel

f : 640 x 360 resolution



Same

filter2(g, f, shape) in MATLAB

Full: output_size = f_size + g_size – 1

Same: output_size = f_size

```
>>same = filter2(g, im, 'same');
>> size(same)

ans =

   360   640
```

2D visualization of convolution (valid)

Image I

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

8x8

Kernel f

1	2	3
4	5	6
7	8	9

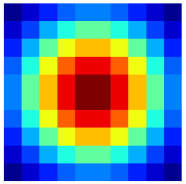
3x3

Output g

45	45	45	45	45	45
45	45	45	45	45	45
39	39	39	39	39	39
24	24	24	24	24	24
0	0	0	0	0	0
0	0	0	0	0	0

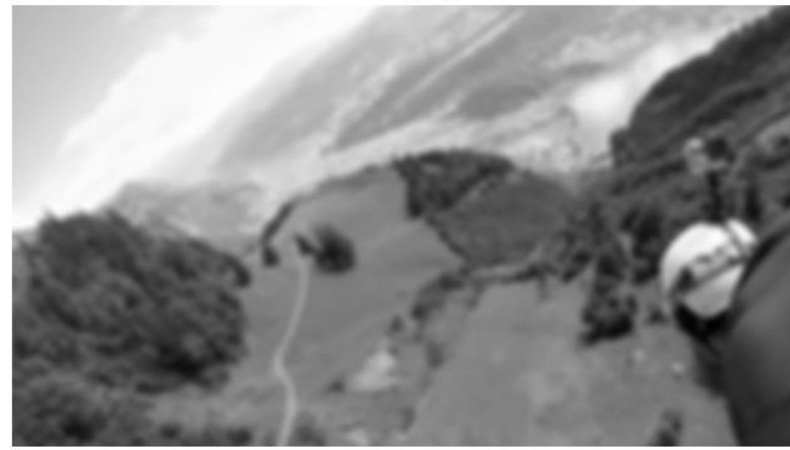
6x6

Output Size of Image Convolution



g : 10 x 10 Gaussian kernel

f : 640 x 360 resolution



Valid

filter2(g, f, shape) in MATLAB

Full: output_size = f_size + g_size – 1

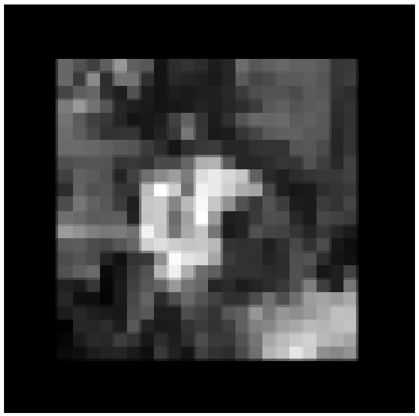
Same: output_size = f_size

Valid: output_size = f_size – (g_size – 1)

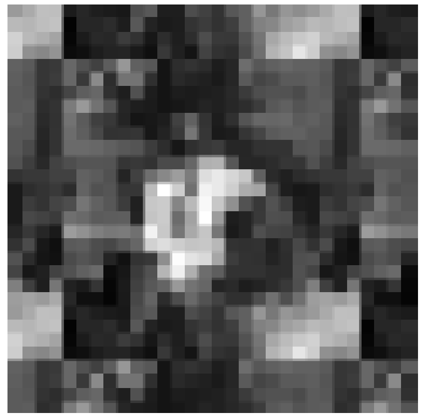
```
>> valid = filter2(g, im, 'valid');
>> size(valid)

ans =

  351   631
```

Image Boundary Effect

The filter window falls off at the edge of image.

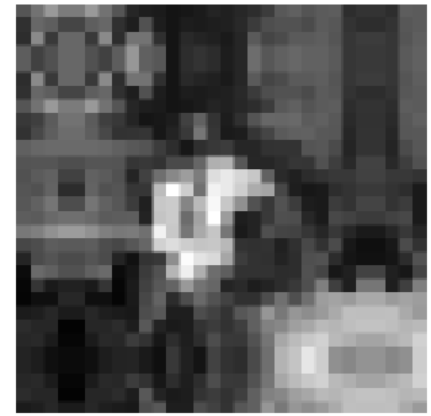


zero

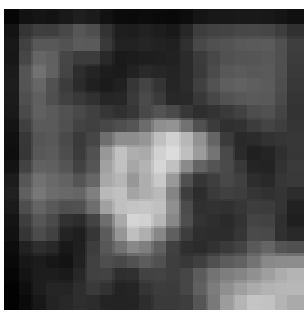


wrap

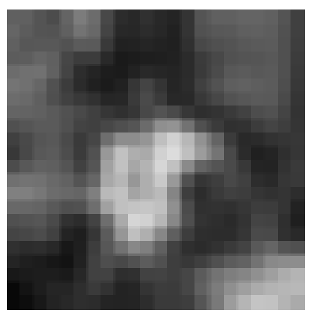
clamp



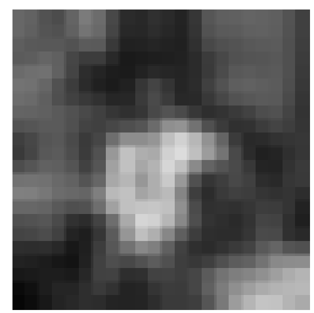
mirror



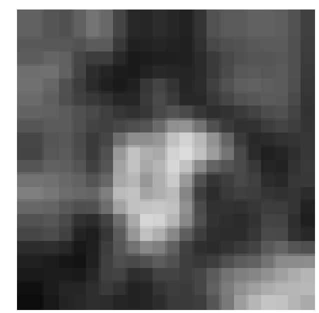
blurred zero



normalized zero



blurred clamp



blurred mirror

Image Extrapolation (Mirroring)

Code

```
J = imread('image.bmp');
figure; imshow(J);
```

J

Image Extrapolation (Mirroring)

Code

```
boarder = 40;
[nr,nc,nb] = size(J);
J_big = zeros(nr+2*boarder, nc + 2*boarder,nb);
J_big(boarder+1:boarder+nr,boarder+1:boarder+nc,:) = J;
```

J_big

Image Extrapolation (Mirroring)

Code

```
for i=1:border,
    for j=1:border,
        J_big(i,j,:) = J(border-i+1,border-j+1,:);
    end
end
```

Mirroring with respect to the board

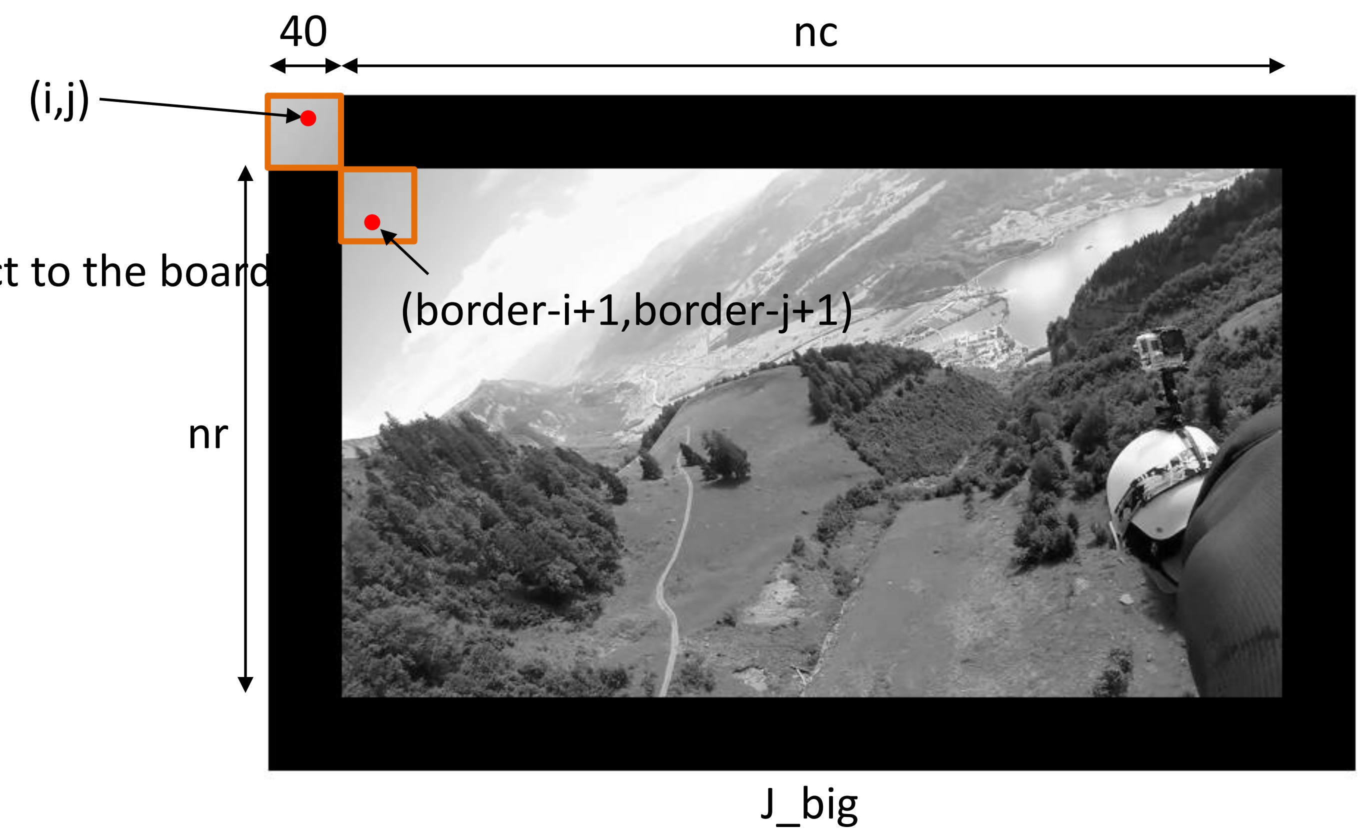


Image Extrapolation (Mirroring)

Code

```
for i=1:boarder,
   for j=border+1:border+nc,
        J_big(i,j,:) = J(border-i+1,j-border,:);
    end
end
```

Mirroring with respect to the board

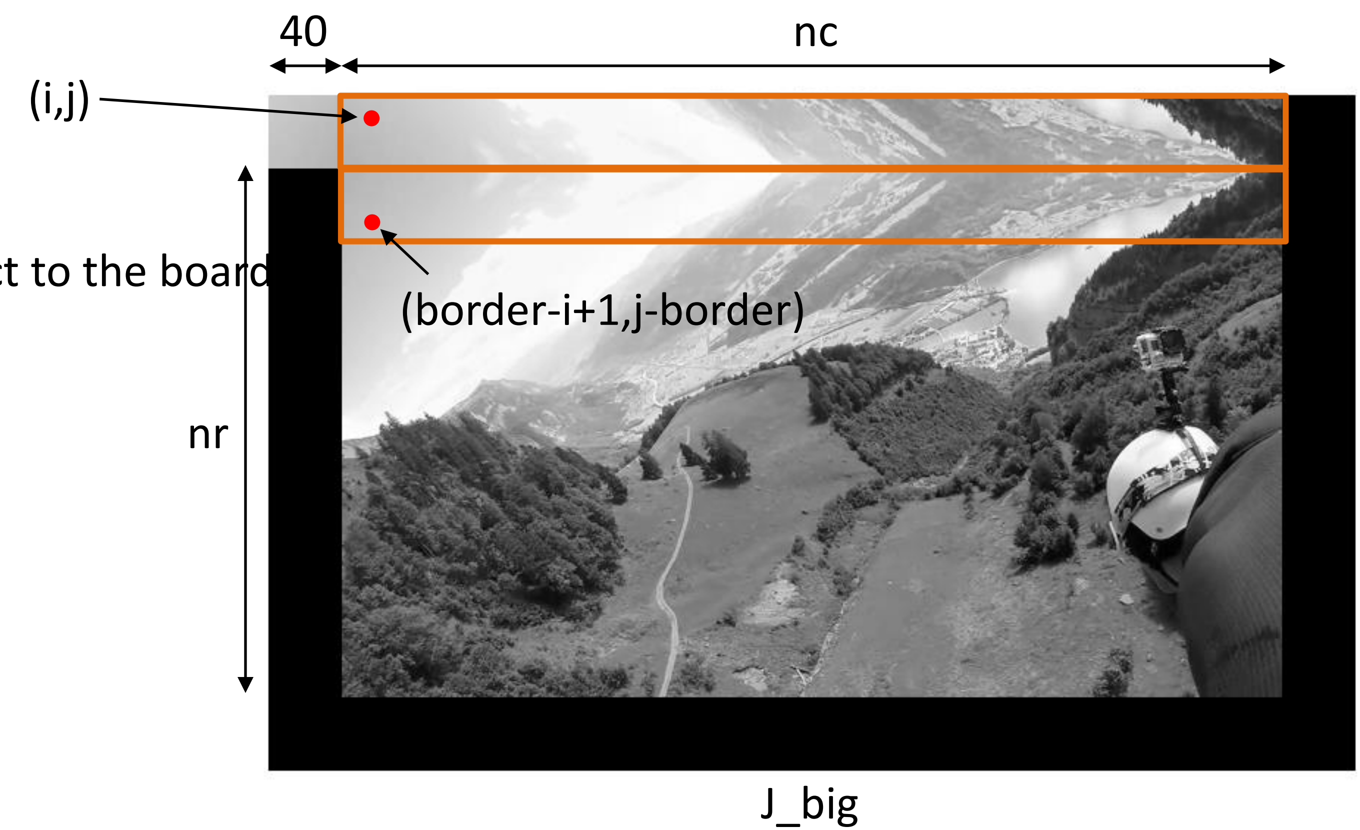


Image Extrapolation (Mirroring)

Code

```
for i=nr+border+1:border*2+nr,
    for j=border+1:border+nc,
        J_big(i,j,:) = J(2*nr-i+border+1,j-border,:);
    end
end
```

Mirroring with respect to the board

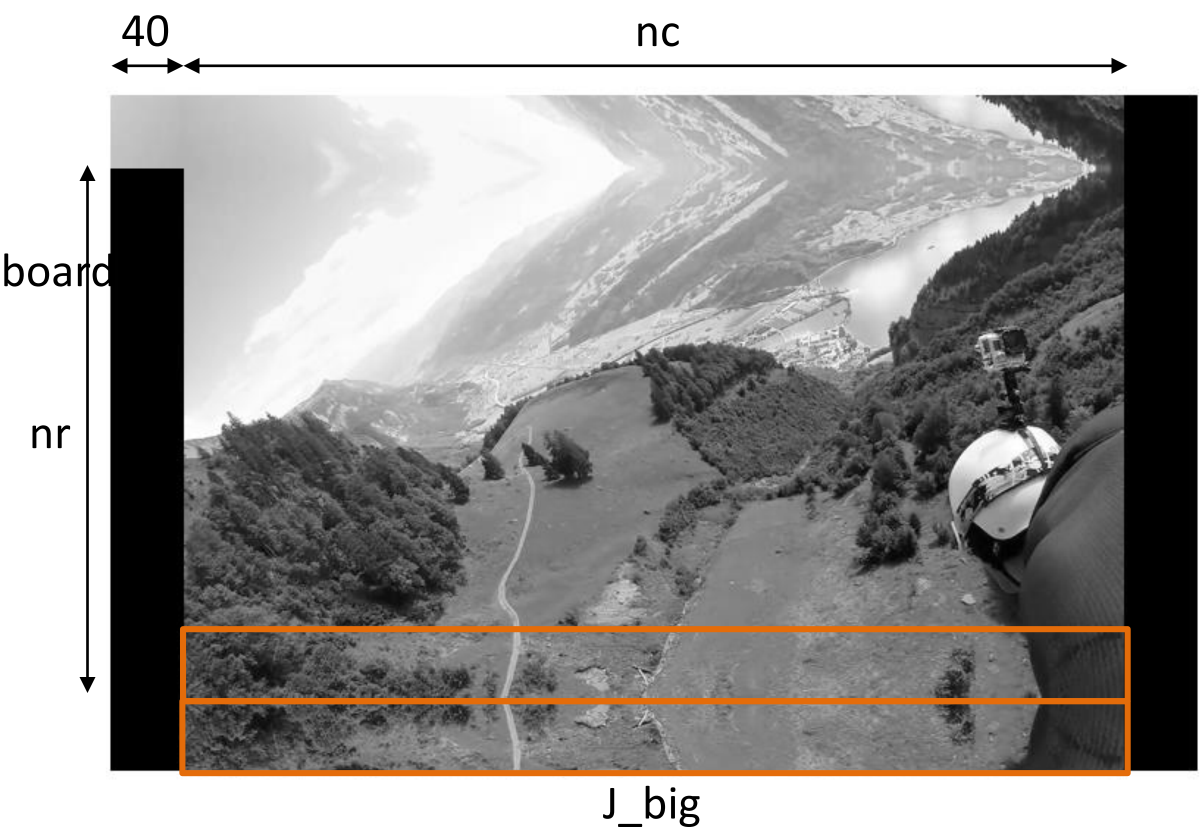
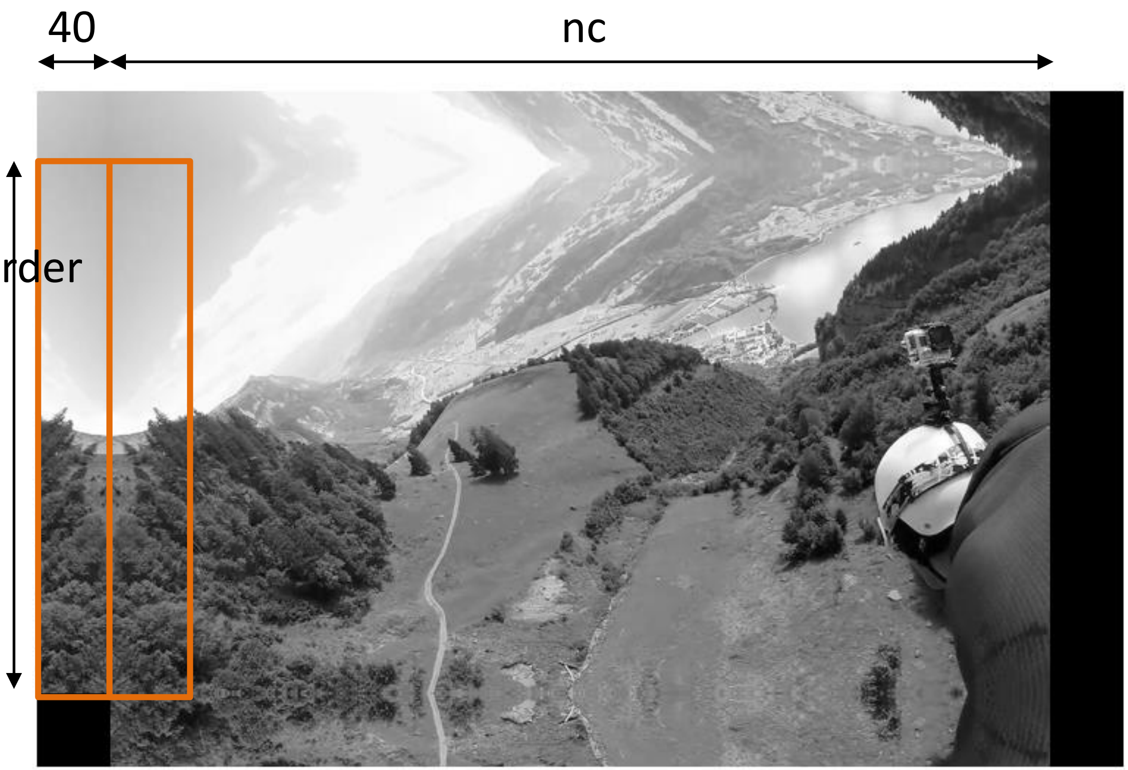


Image Extrapolation (Mirroring)

Code

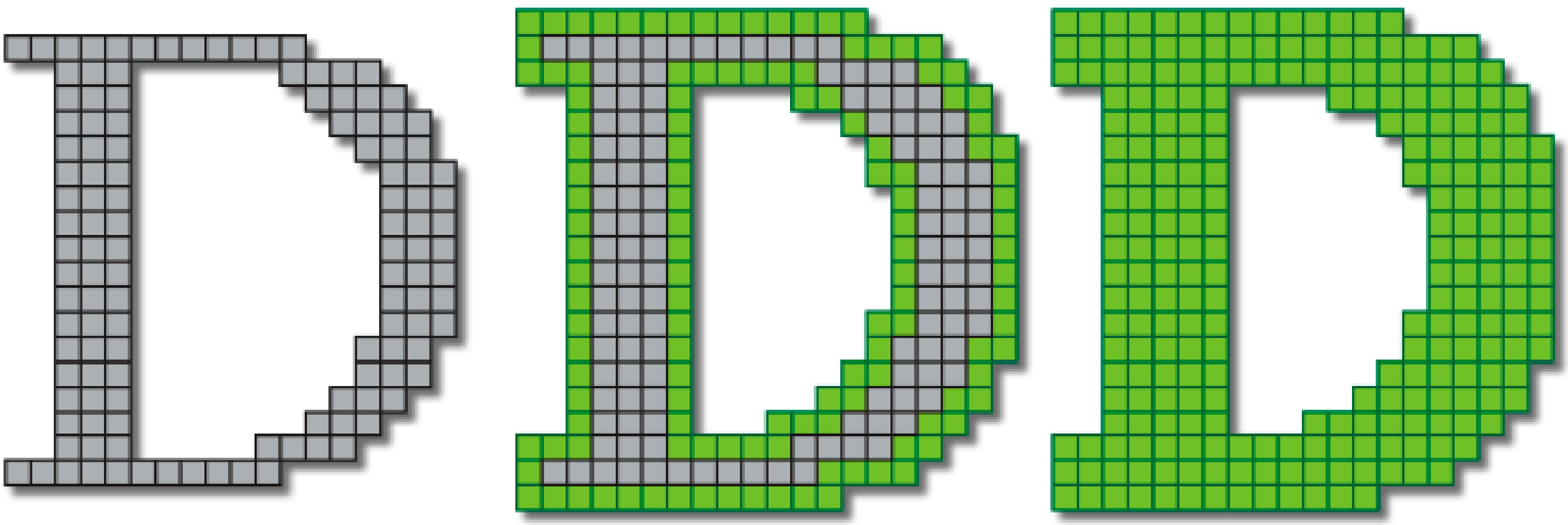
```
for i=border+1:border+nr;
    for j=1:border,
        J_big(i,j,:) = J(i-border,border-j+1,:);
    end
end
```

Mirroring with respect to the boarder



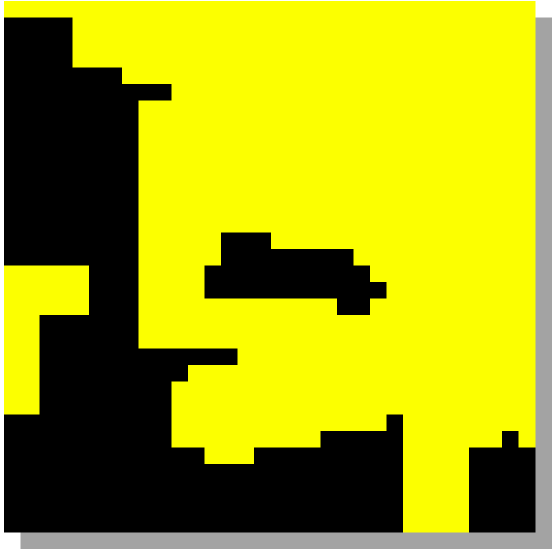
J_big

Dilation

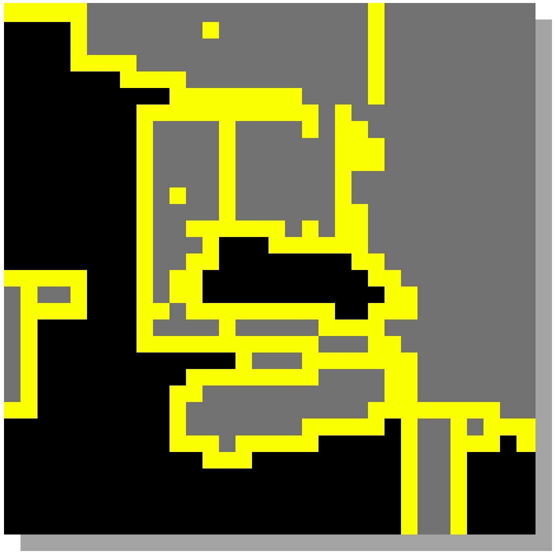


Dilation

The locus of pixels $\mathbf{p} \in S_{\mathrm{p}}$ such that $(\check{Z} + \mathbf{p}) \cap \mathrm{I} \neq \varnothing$.



dilated image

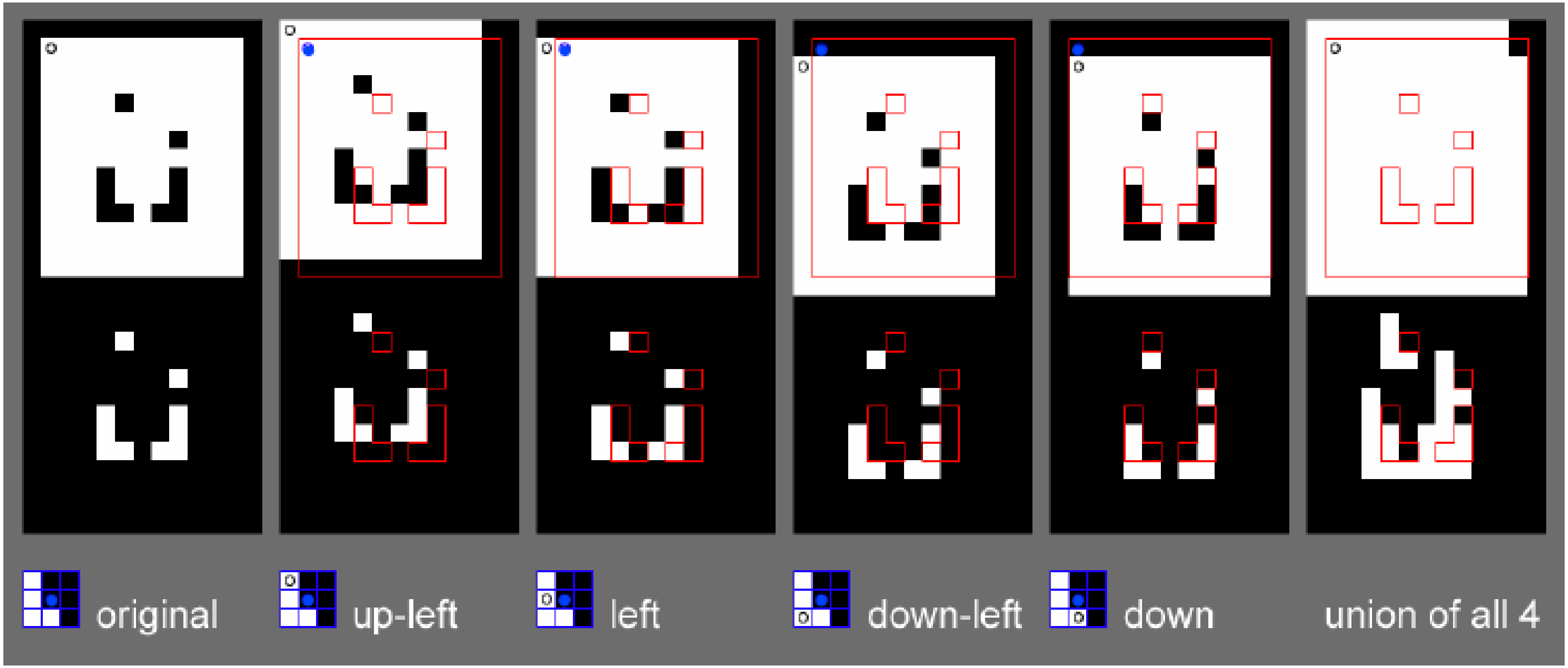


original / dilation

original image

SE = Z_8

Dilation through Image Shifting



Examples of image operation as convolution

Average Filter

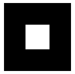
- Mask with positive entries, that sum 1.
- Replaces each pixel with an average of its neighborhood.
- If all weights are equal, it is called a BOX filter.

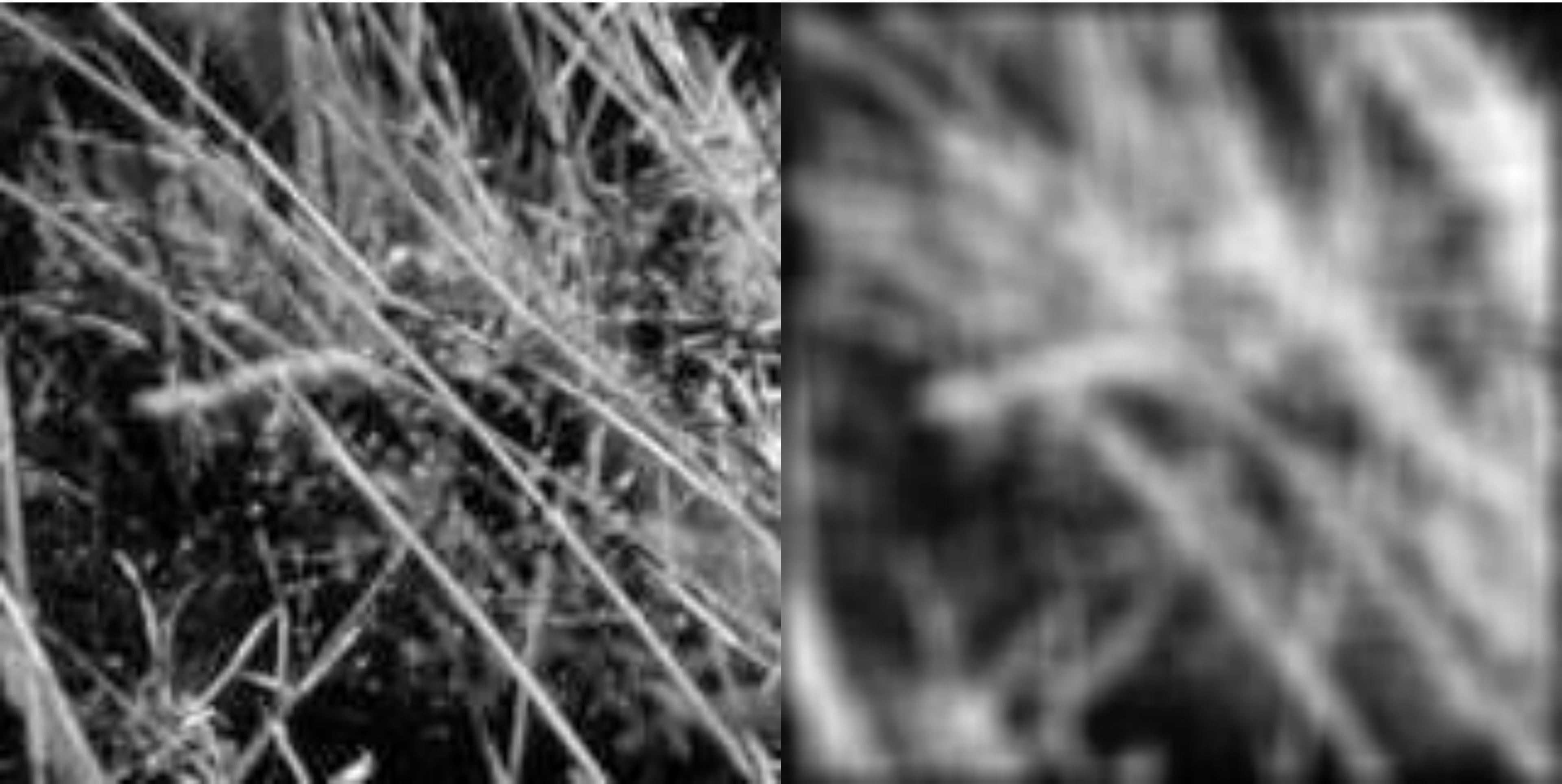
F

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

(Camps)

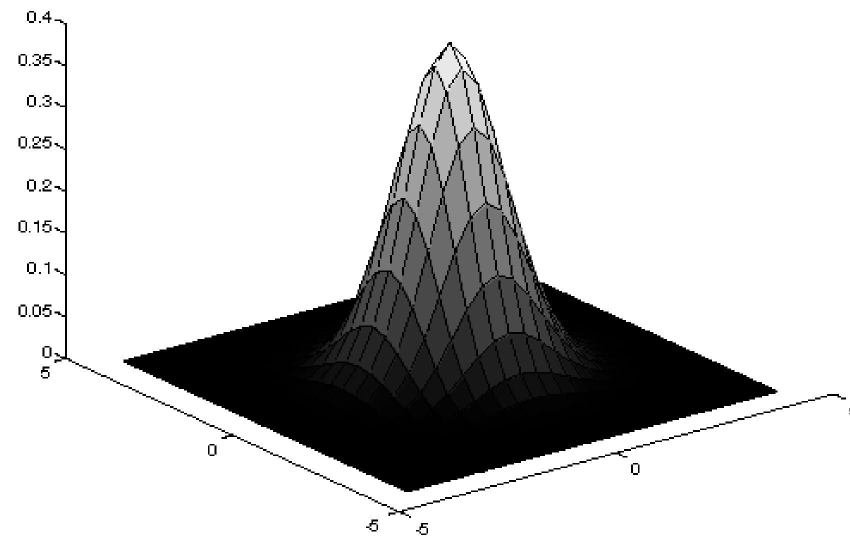
Example 1: Smoothing by Averaging





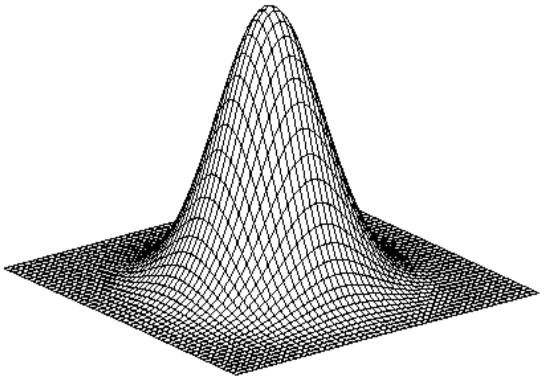
Gaussian Averaging

- Rotationally symmetric.
- Weights nearby pixels more than distant ones.
  - This makes sense as probabalistic inference.



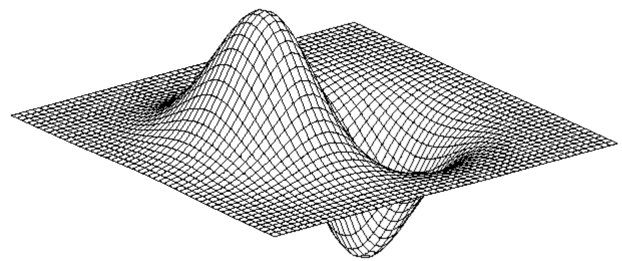
- A Gaussian gives a good model of a fuzzy blob

2D filters, more on this later…



Gaussian

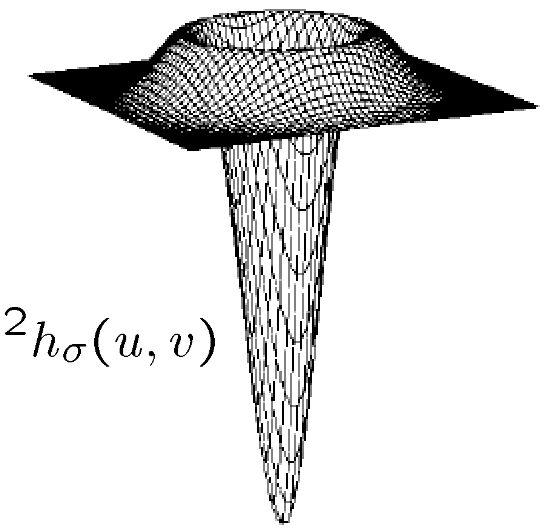
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian

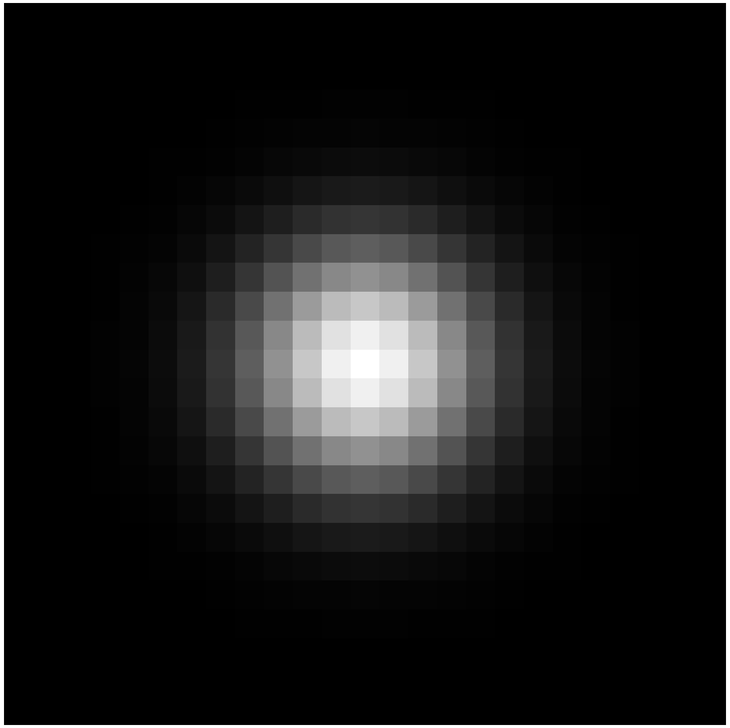


$$\nabla^2 h_\sigma(u,v)$$

- is the **Laplacian** operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

An Isotropic Gaussian

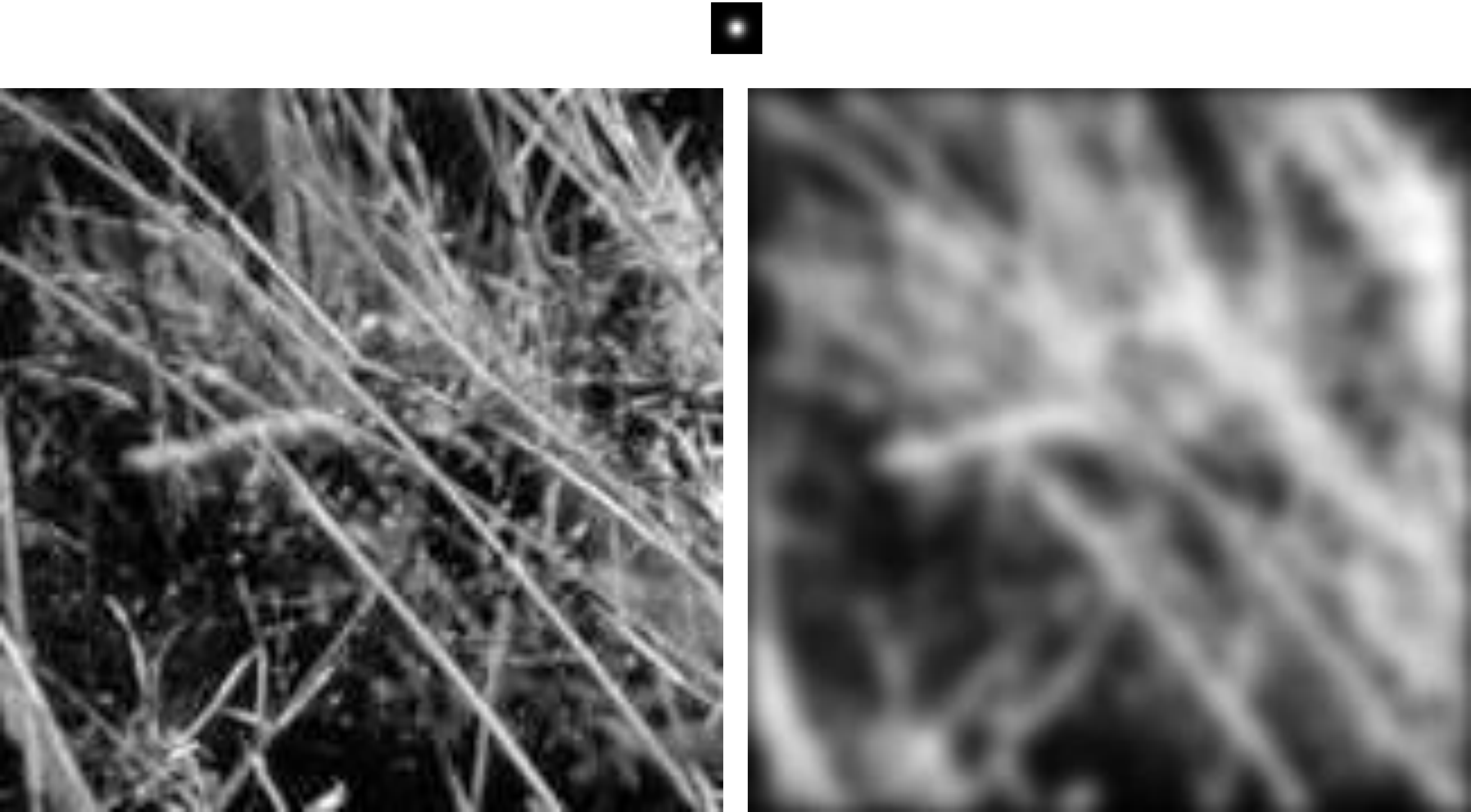


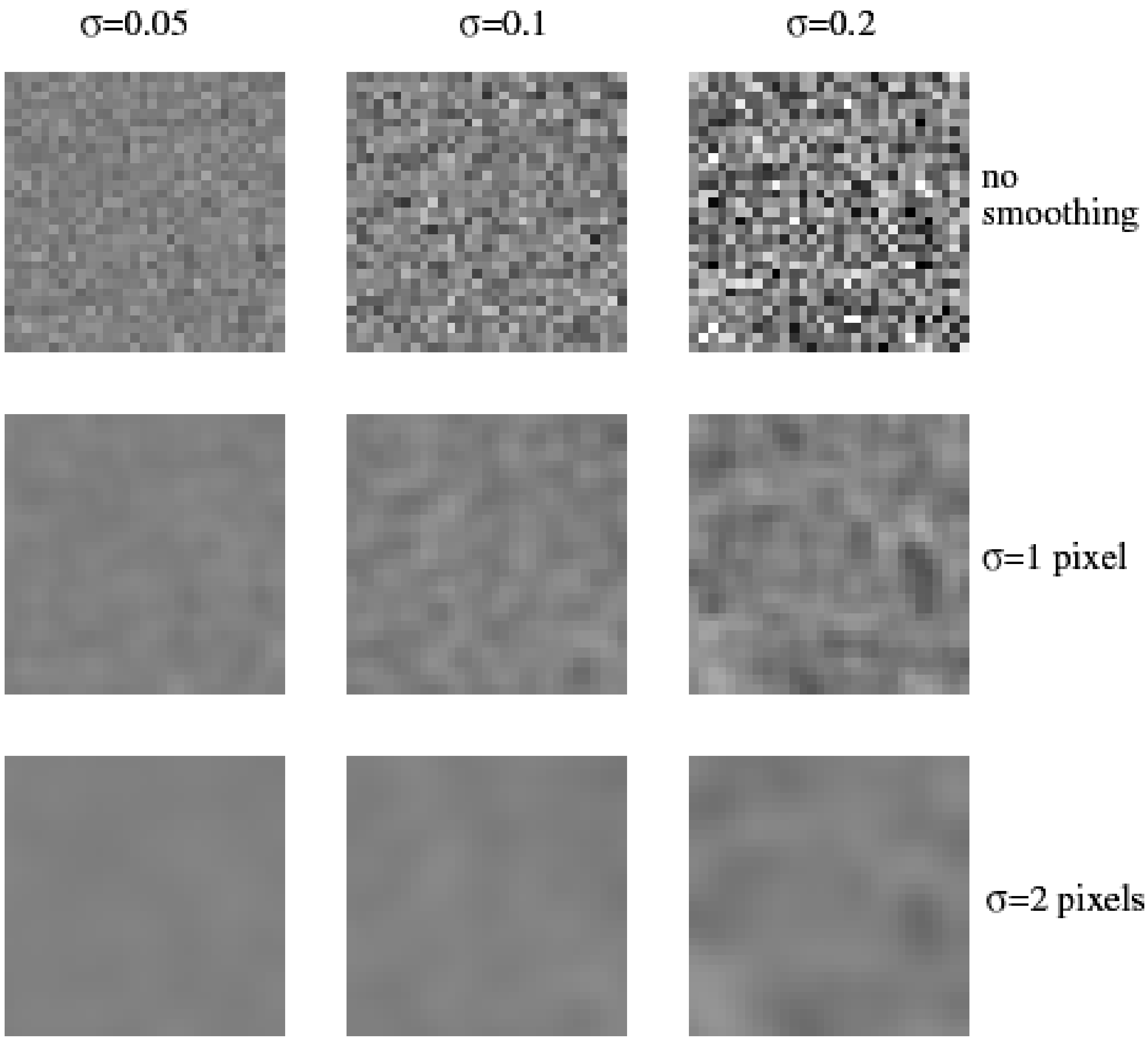
- The picture shows a smoothing kernel proportional to

$$e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- (which is a reasonable model of a circularly symmetric fuzzy blob)

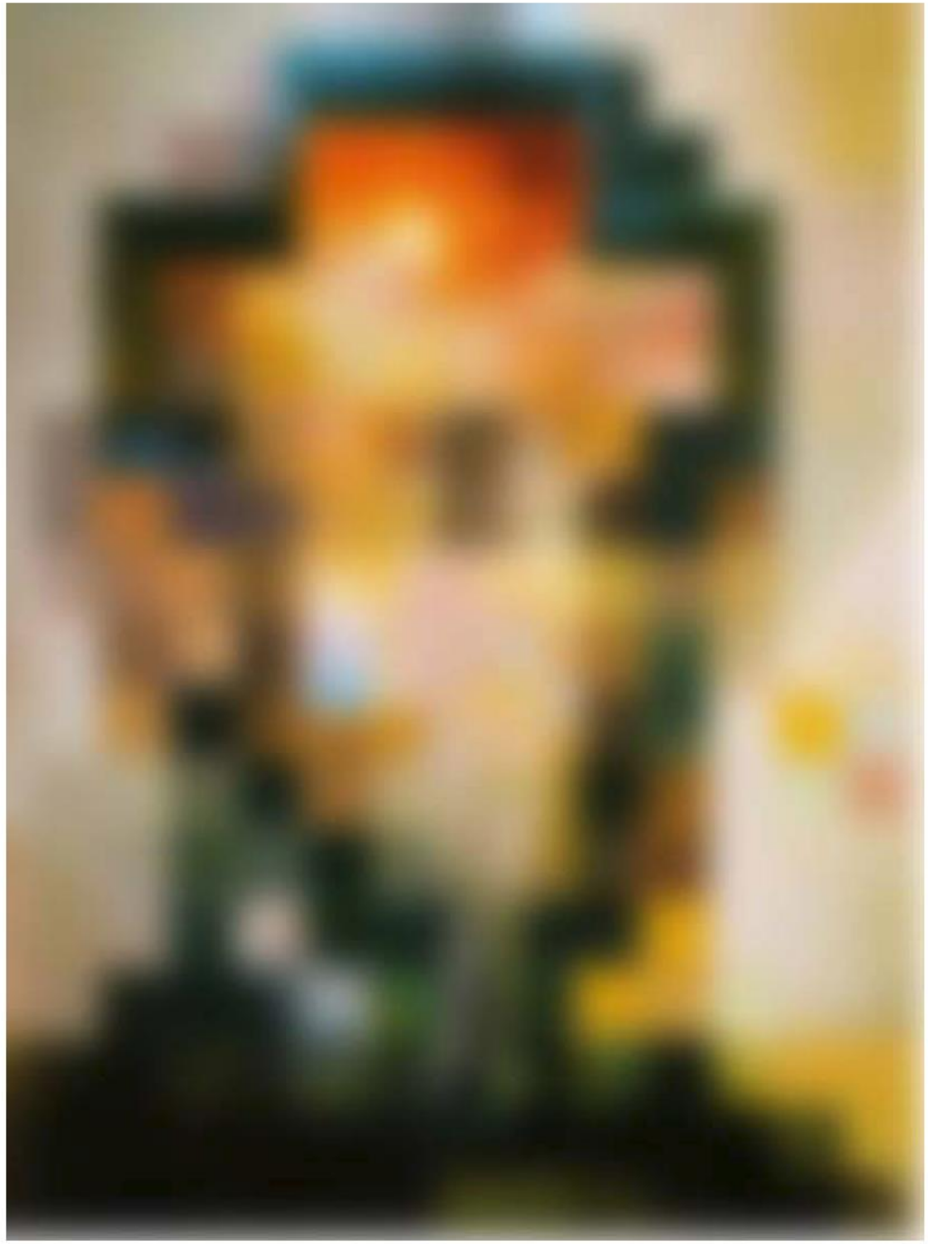
Smoothing with a Gaussian





The effects of smoothing
Each row shows smoothing with gaussians of different width; each column shows different realizations of an image of gaussian noise.

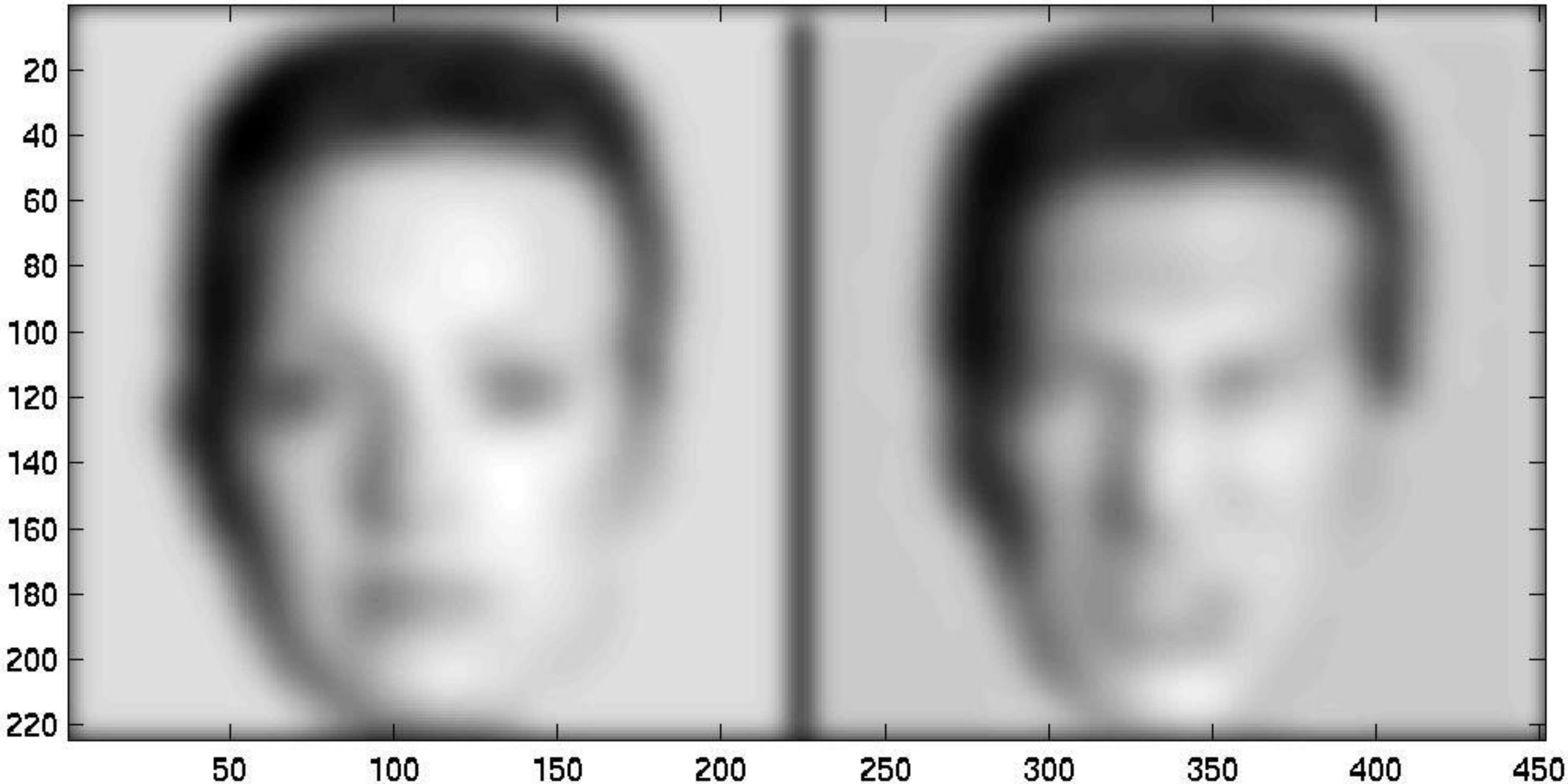
Salvador Dali, *"Gala Contemplating the Mediterranean Sea, which at 30 meters becomes the portrait of Abraham Lincoln*", 1976



Salvador Dali, *“Gala Contemplating the Mediterranean Sea, which at 30 meters becomes the portrait of Abraham Lincoln*”, 1976

Image smoothing can remove noise, and also …


20
40
60
80
100
120
140
160
180
200
220
50
100
150
200
250
300
350
400
450



20
40
60
80
100
120
140
160
180
200
220
50
100
150
200
250
300
350
400
450