

Tutorial A1: Introduction to Image Processing

Peter Corke

Images and pixels

For my part of the course we will be using a wrapped version of OpenCV - the Machine Vision Toolbox for Python - which builds on an earlier MATLAB toolbox

README.md

Machine Vision Toolbox for Python

collection PYTHONROBOTICS powered by spatial maths collection QUT Robotics

pypi package 0.9.4 python 3.7 | 3.8 | 3.9 | 3.10 powered by OpenCV powered by Open3D License MIT

Test-master passing codecov 28% downloads 19/week



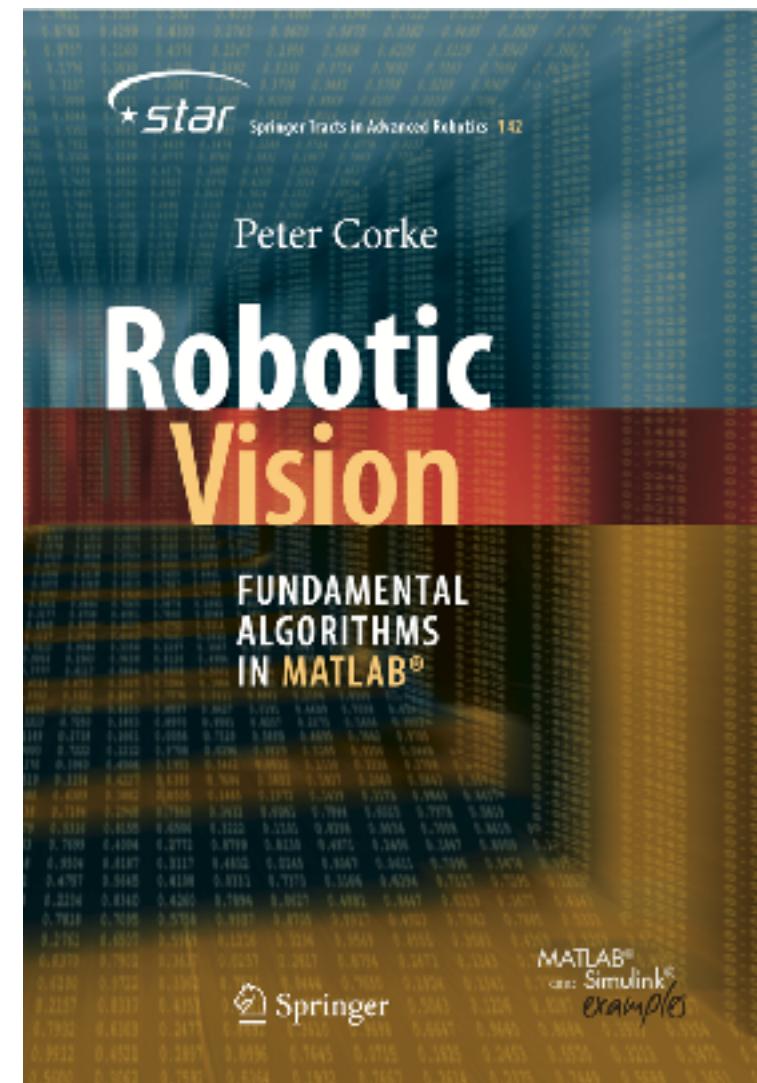
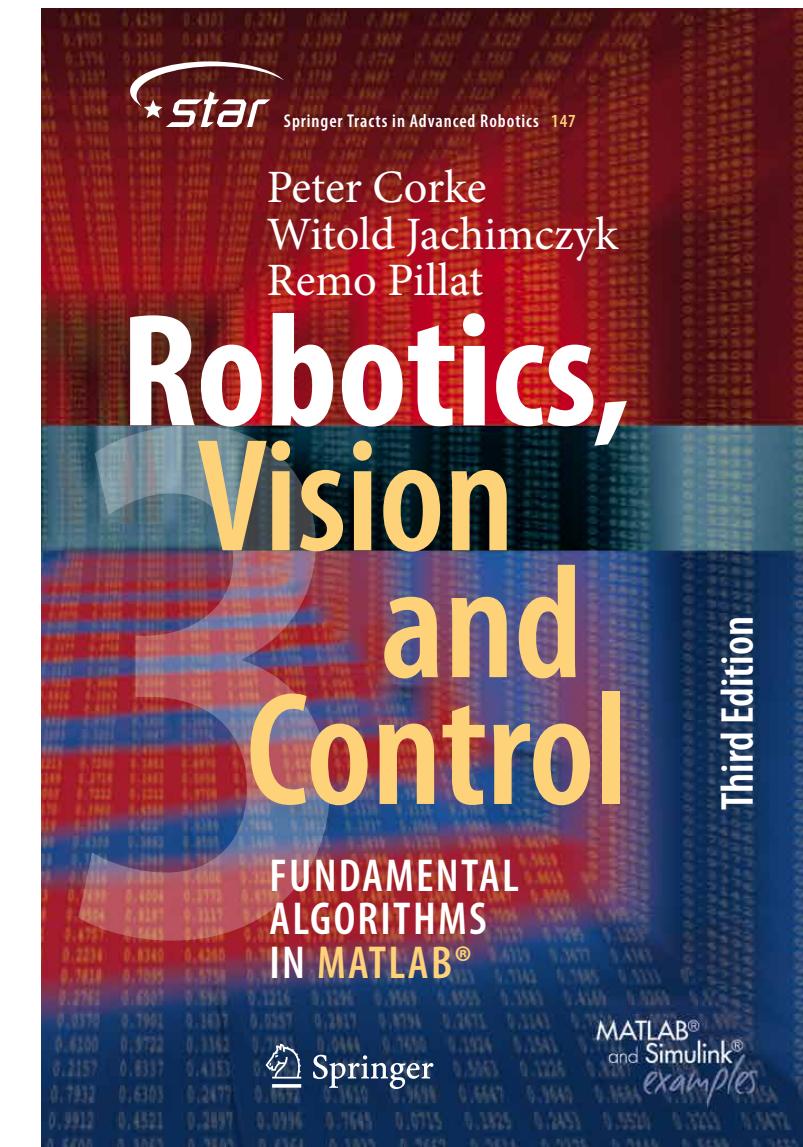
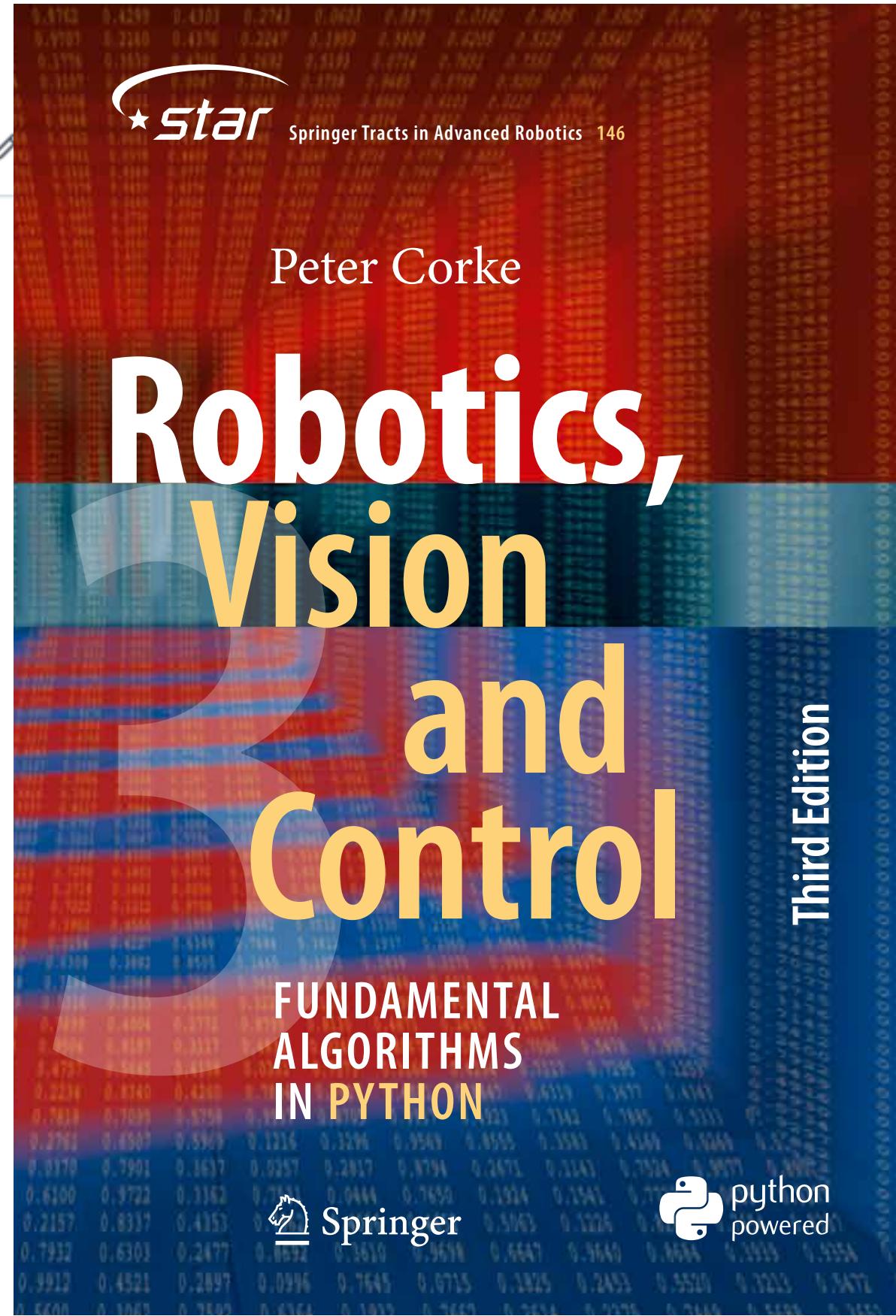
A Python implementation of the [Machine Vision Toolbox for MATLAB®](#)

- GitHub repository
- Documentation
- Examples and details
- Installation

Synopsis

The Machine Vision Toolbox for Python (MVTB-P) provides many functions that are useful in machine vision and vision-based control. The main components are:

- An `Image` object with nearly 200 methods and properties that wrap functions from OpenCV, NumPy and SciPy. Methods support monadic, dyadic, filtering, edge detection, mathematical morphology and feature extraction (blobs, lines and point/corner features), as well as operator overloading. Images are stored as encapsulated NumPy arrays along with image metadata.



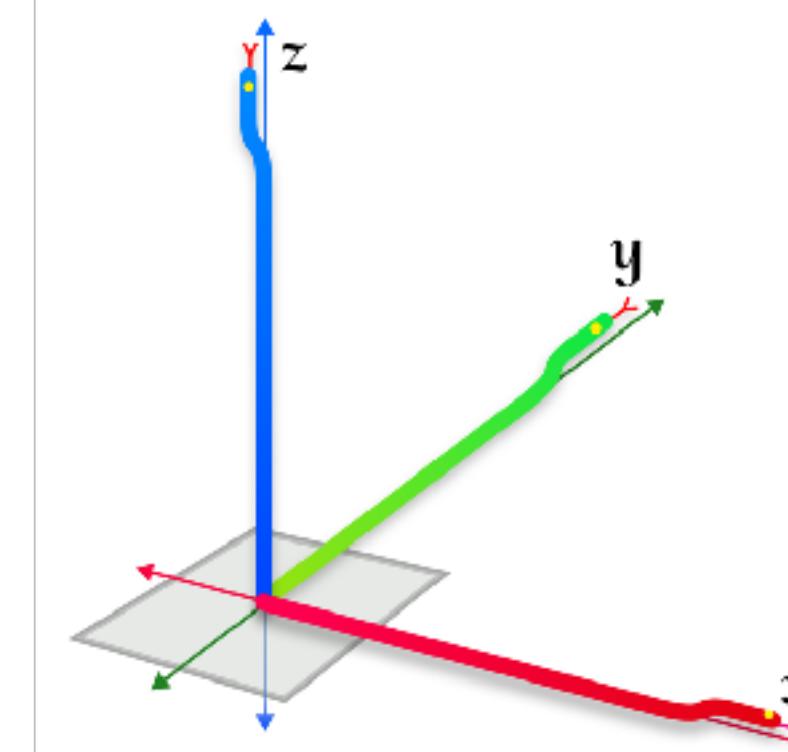
The install is easy



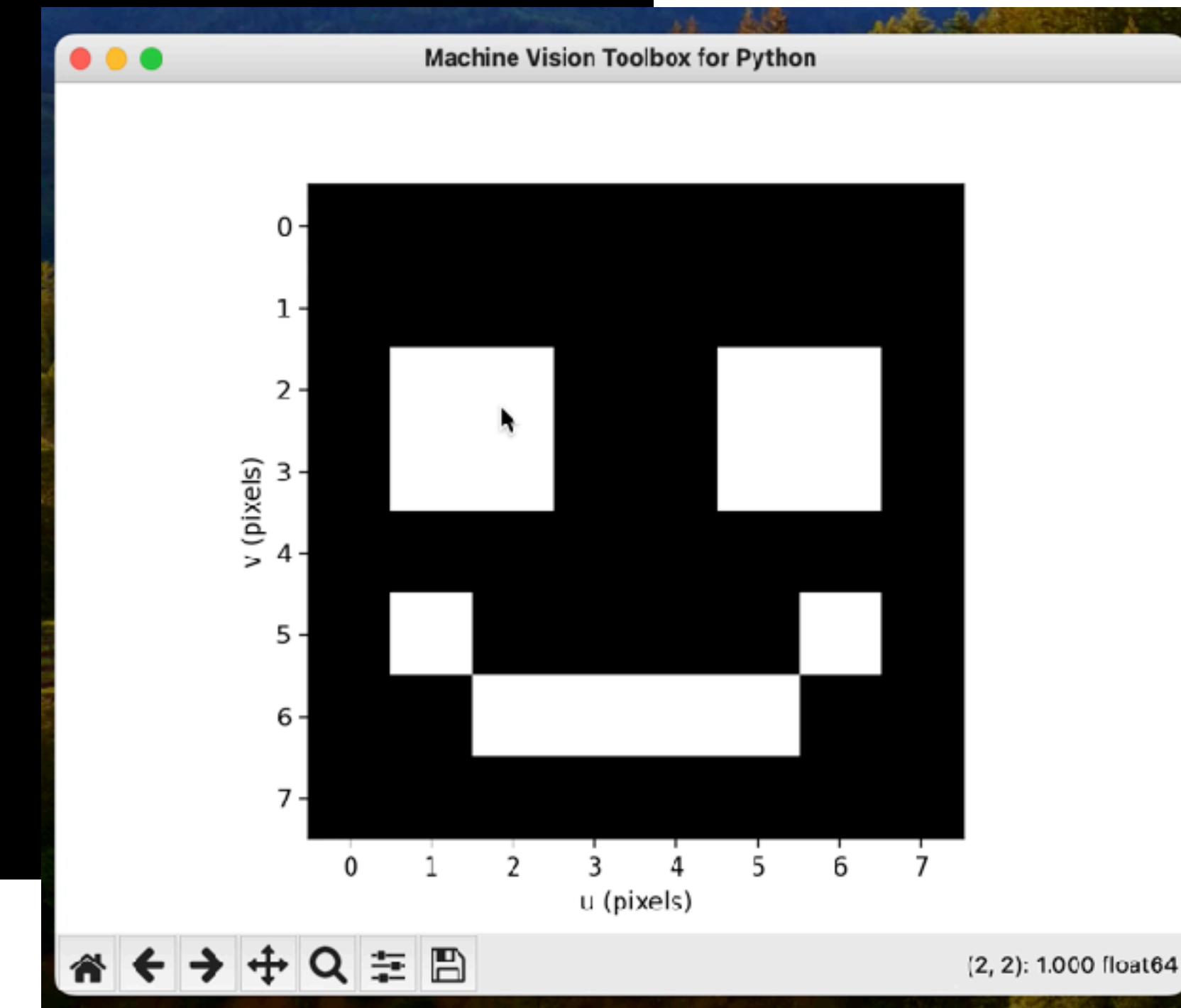
`pip install machinevision-toolbox-python`

and it also pulls in:

- `mvtb-data`
- `spatialmath-python`
- `OpenCV`
- `Open3D`



An image is simply a 2D array of numbers



An image is simply a 2D array of numbers

```
from machinevisiontoolbox import *

street, _ = iread("street.png")

print(street.shape)

(851, 1280)

print(street[125:140, 295:310])

[196 195 197 198 198 199 197 199 199 201 199 197 197 200 201]
[194 193 195 197 198 199 198 198 200 202 201 196 195 199 198]
[194 195 194 194 196 195 197 200 199 198 200 198 199 200 200]
[194 202 199 196 194 195 198 198 197 196 200 195 198 198 199]
[ 91 130 170 200 205 204 198 194 195 197 198 192 198 199 200]
[ 29  31  48  89 122 156 191 202 199 198 198 194 196 199 199]
[ 30  32  31  27  30  41  67 115 161 183 194 194 196 201 200]
[ 25  27  27  28  30  29  26  26  40  70 110 170 196 195 198]
[ 21  22  21  22  23  24  27  29  28  27  78 169 200 196 197]
[ 18  19  20  20  22  23  23  23  28  30 101 190 195 194 198]
[ 17  18  19  19  20  22  22  24  29  32  91 181 195 196 197]
[ 15  17  18  19  20  23  24  27  29  34 102 183 195 197 197]
[ 15  18  19  18  19  22  23  26  33  40 101 190 197 196 198]
[ 15  17  17  18  19  22  23  26  32  41 103 183 192 197 197]
[ 16  17  17  17  20  21  23  26  35  42 106 197 194 193 196]]
```

```
idisp(street, block=True)
```



There are advantages in making the images into Python objects

```
from machinevisiontoolbox import *
street, _ = iread("street.png")
print(street.shape)
(851, 1280)
print(street[125:140, 295:310])
[[196 195 197 ...
idisp(street, block=True)
```

```
from machinevisiontoolbox import *
street = Image.Read("street.png")
print(street)
Image: 1280 x 851 (uint8) [.../images/street.png]
print(street.shape)
(851, 1280)
print(street[125:140, 295:310].A)
[[196 195 197 ...
street.disp(block=True)
```

There are lots of ways to get an image into Python

```
from machinevisiontoolbox import Image
```

```
mona = Image.Read("monalisa.png")  
mona.disp()
```

read from a file

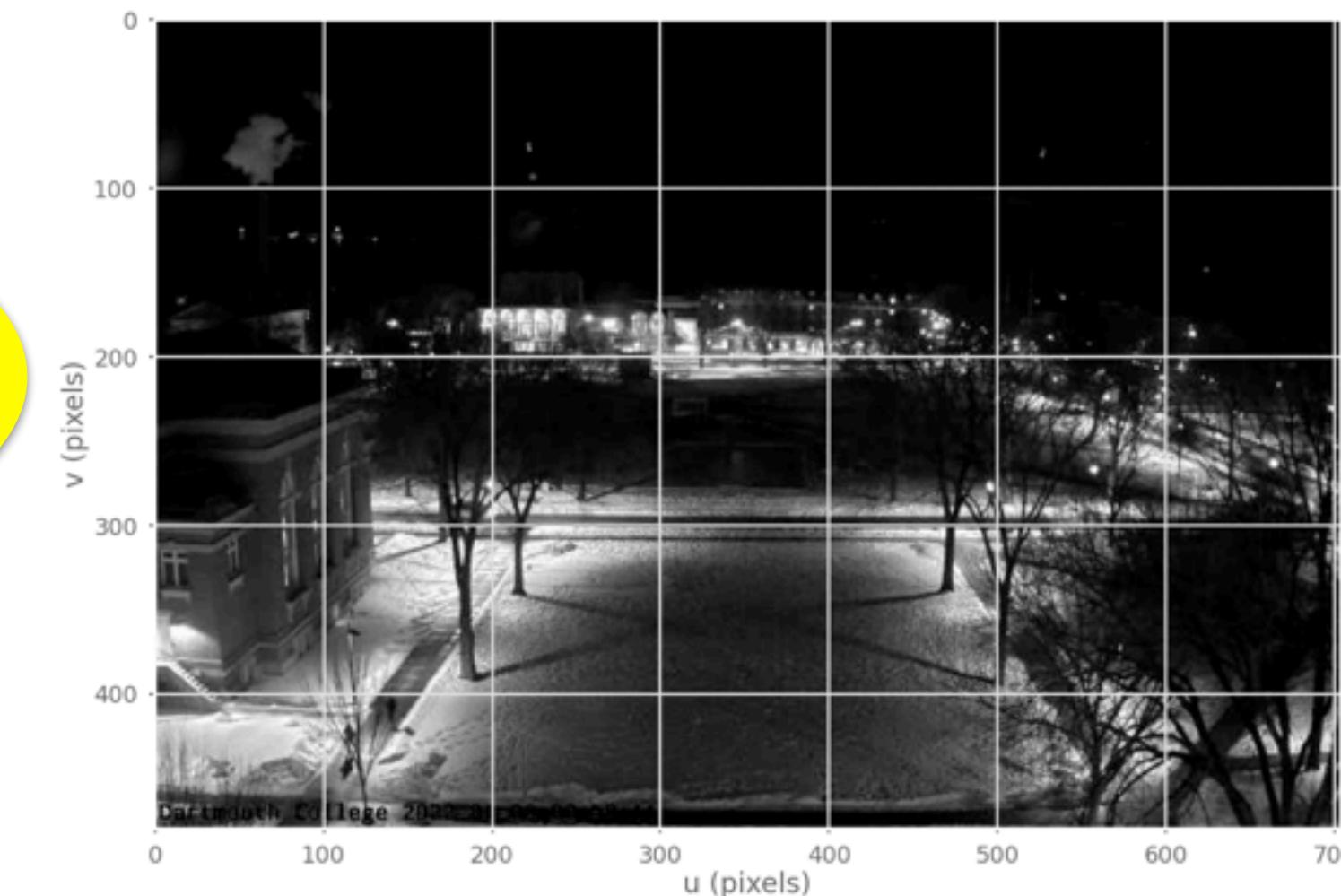


```
camera = VideoCamera()  
camera.grab().disp()
```

read from an attached camera

```
porjus = WebCam("https://uk.jokkmokk.jp/photo/nr4/latest.jpg")  
porjus.grab().disp()
```

read from somebody else's camera



```
im = Image(np.random.randint(low=0, high=255,  
size=(20,20)))
```

create it using code

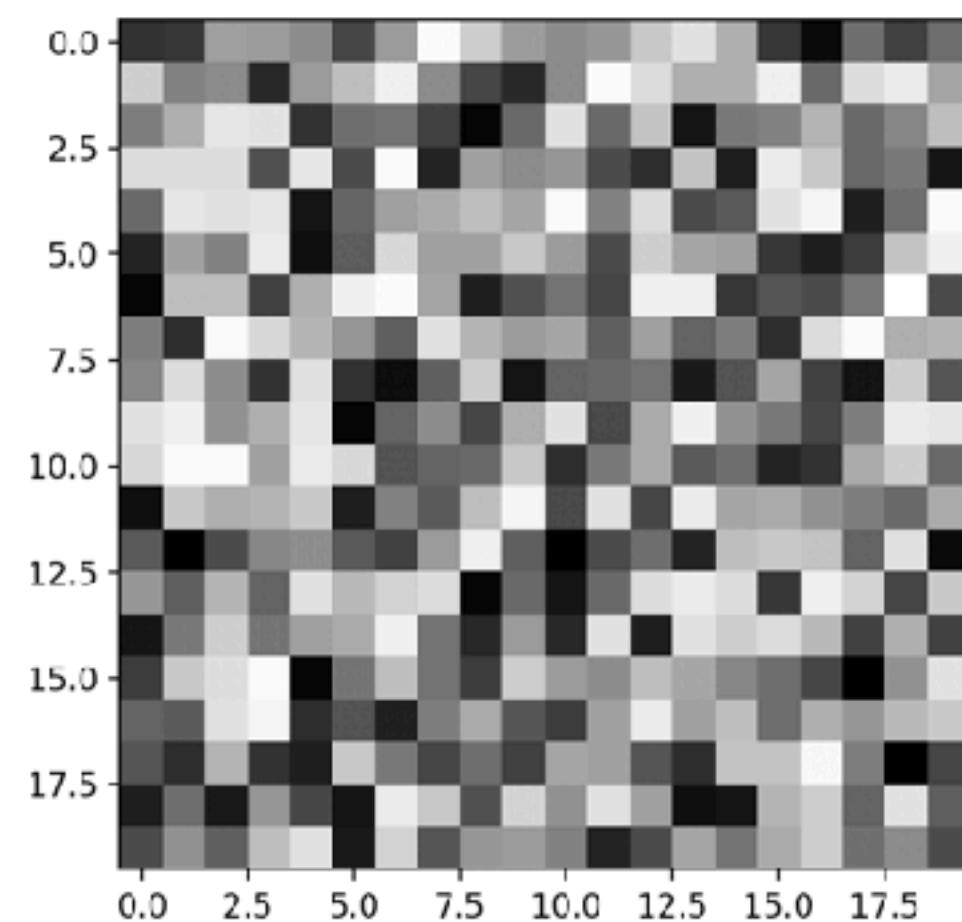
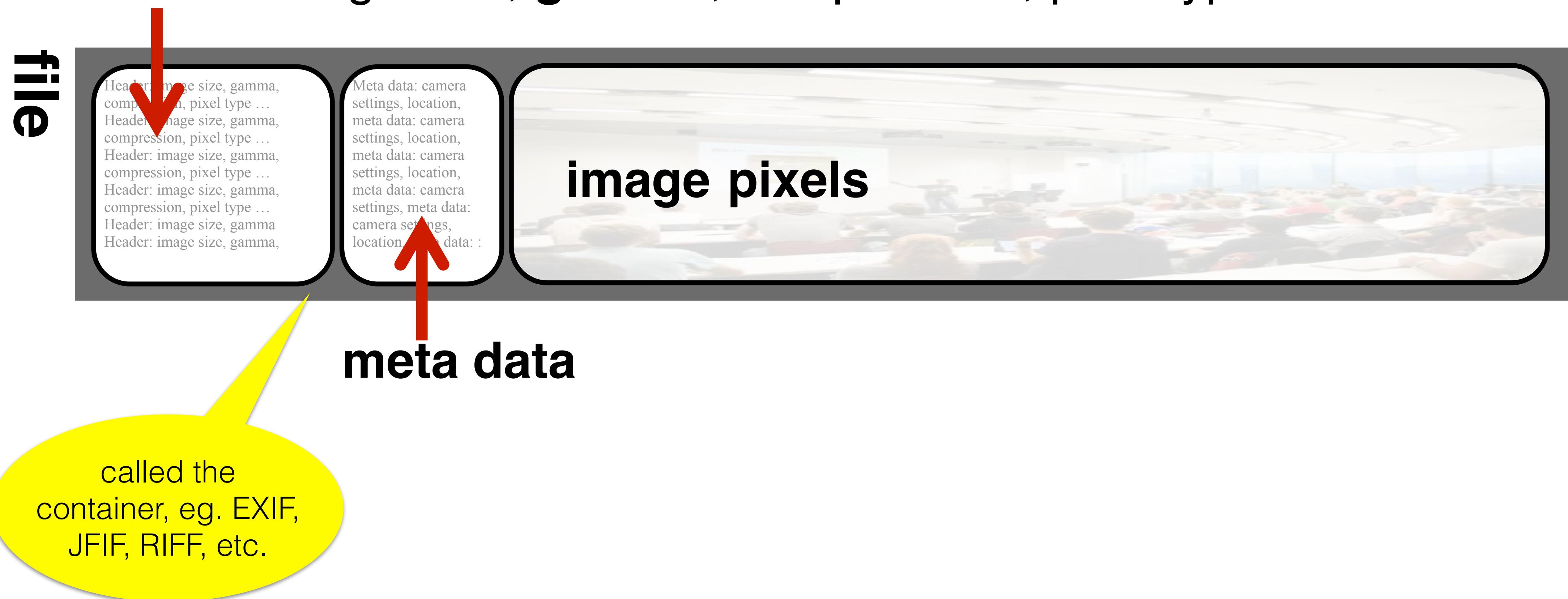
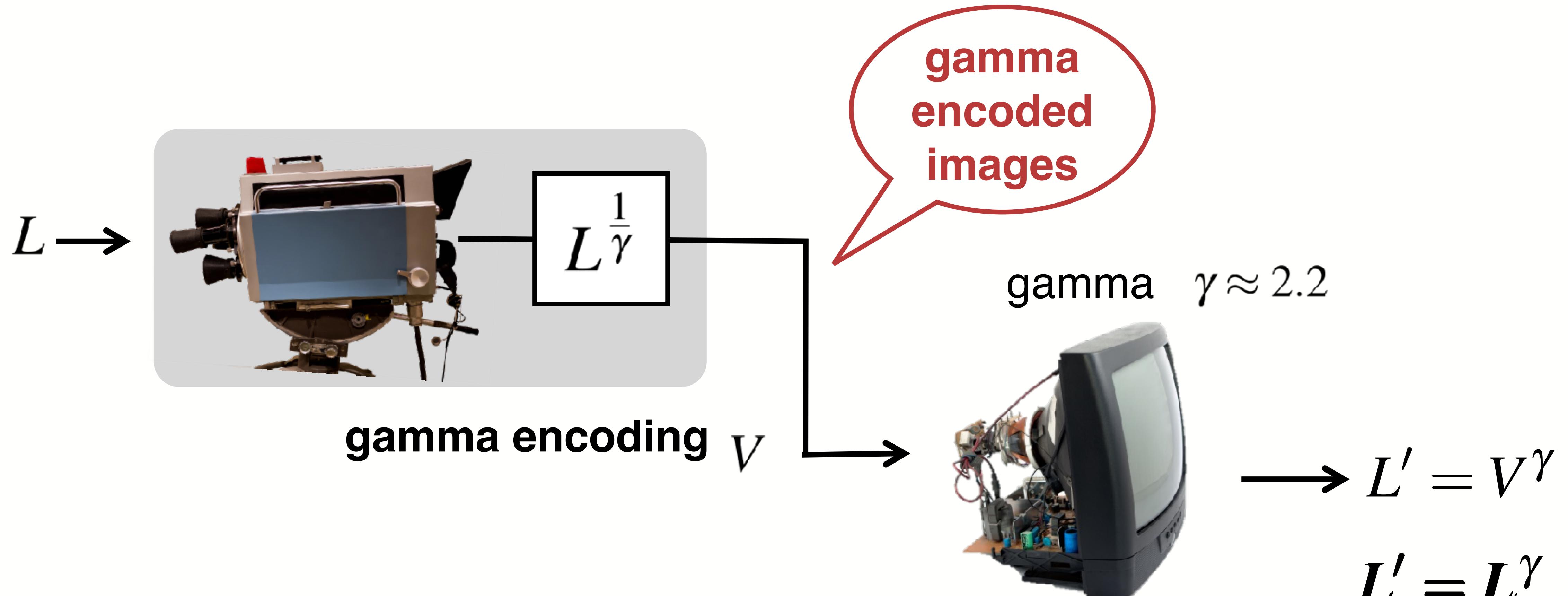


Image file formats

header: image size, gamma, compression, pixel type...



Display non-linearity

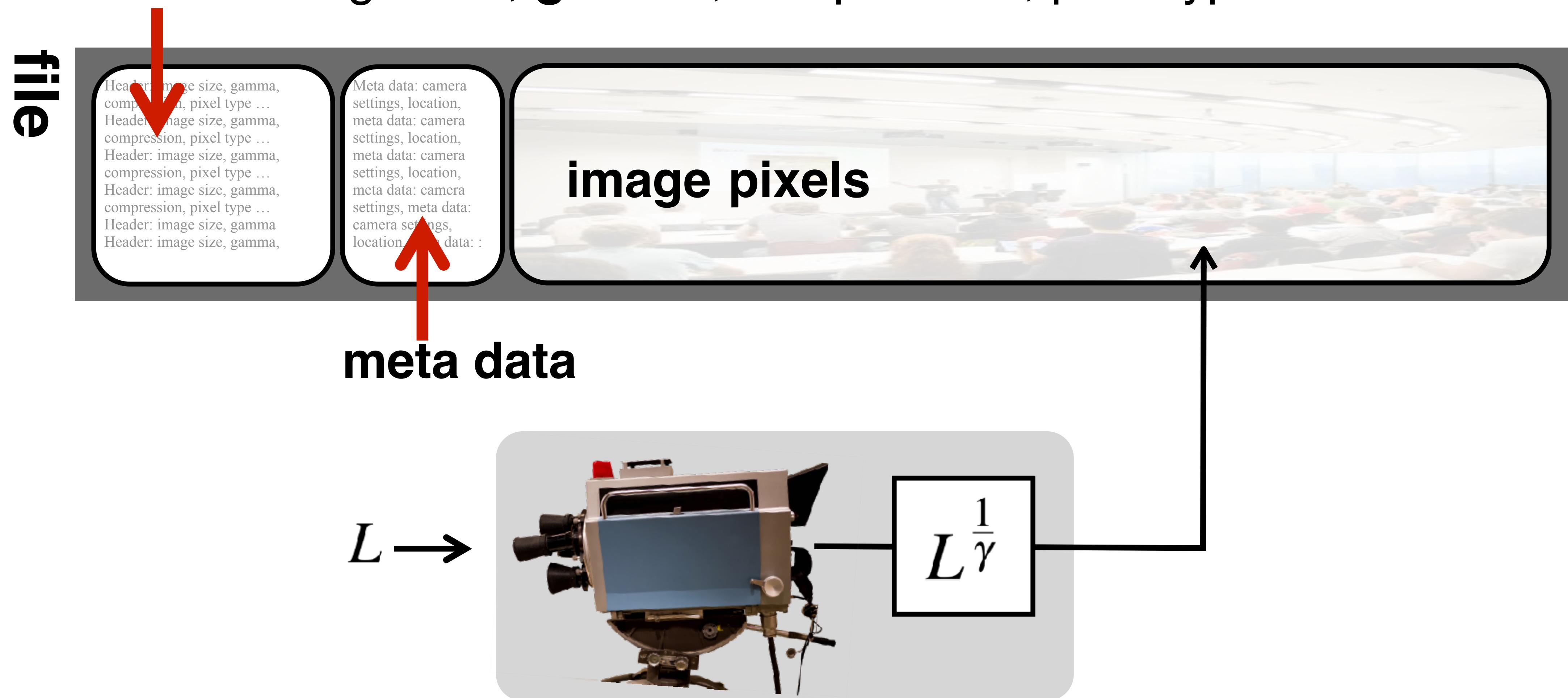


The system is linear end-to-end

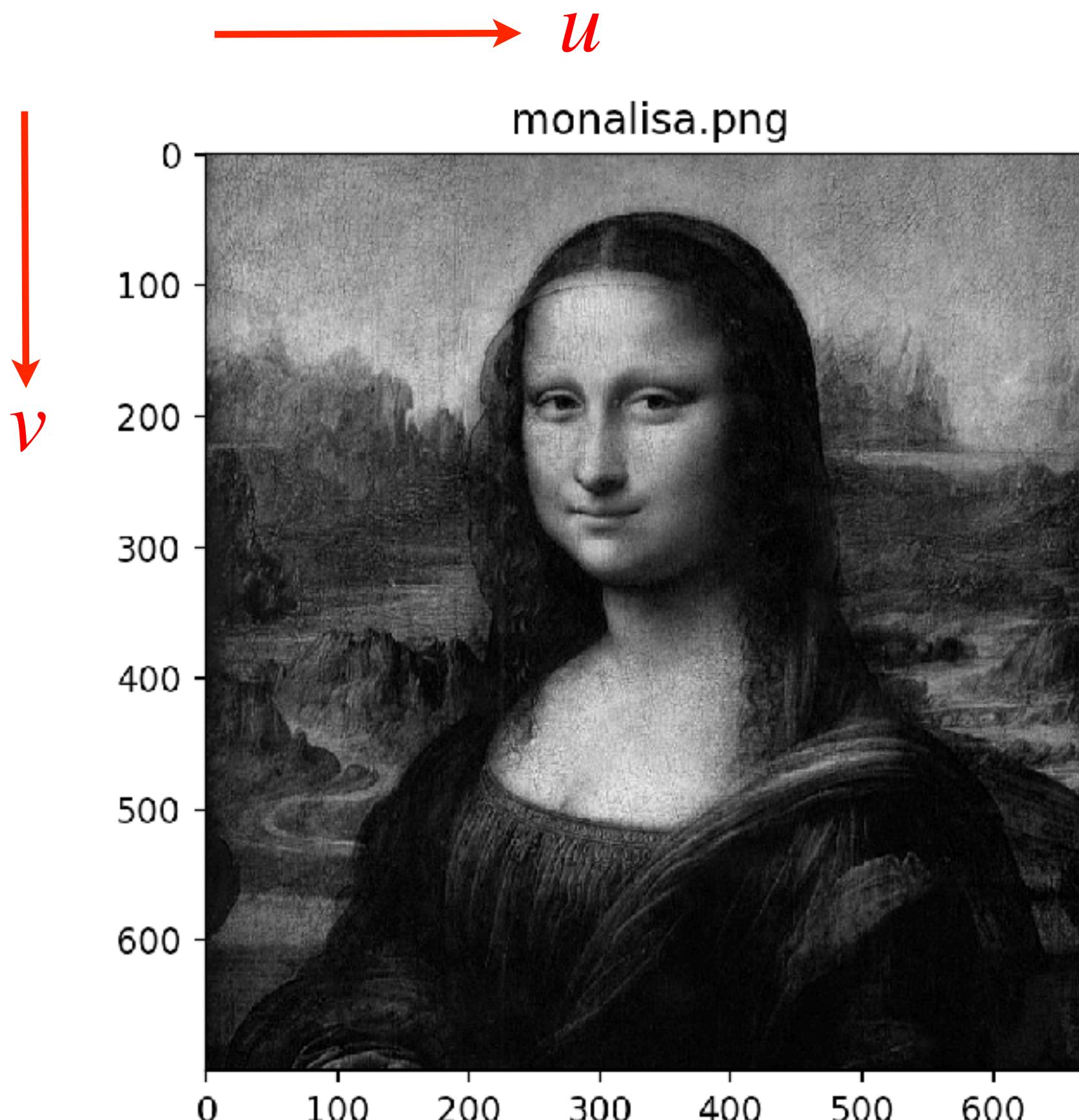
gamma decoding
gamma correction

Image file formats

header: image size, gamma, compression, pixel type...



Beware that indexing into an image pixel array is reversed compared to the way we normally think about coordinates



`image[v, u]`

row

column

DANGER: Reversed compared to graph coordinates!

Images can have tens of millions of pixels, so we want to use a compact representation of each pixel value. uint8 is common

single pixel
“images”

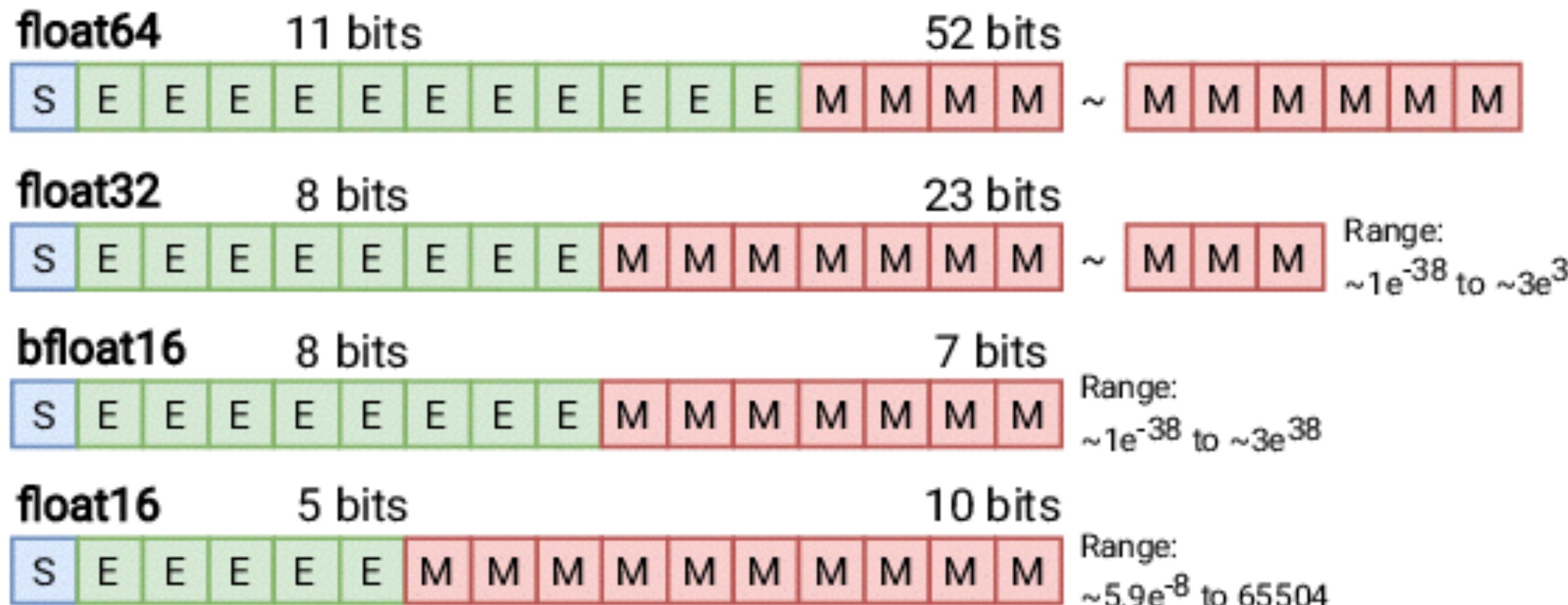
- uint8 = unsigned integer 8 bits
 - range 0 to 255
- each pixel requires 1 byte
- limited dynamic range
- saturation effects

```
>>> a = np.array(5, "uint8")
>>> b = np.array(10, "uint8")
>>> a + b
15
import cv2
>>> cv2.add(a,b)
array([[15]], dtype=uint8)

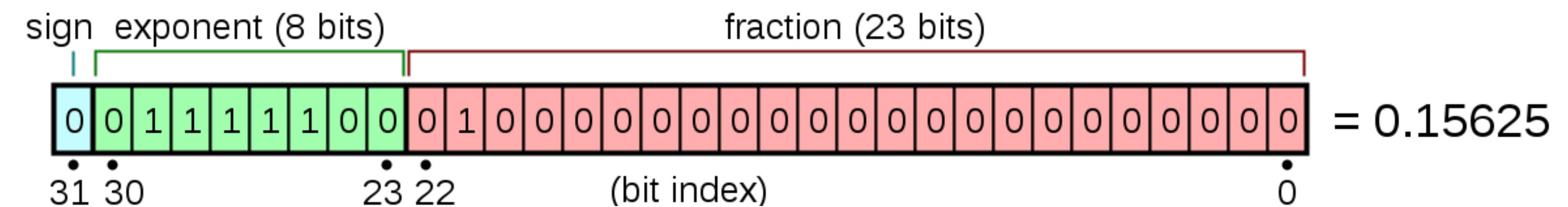
>>> a - b
251
>>> cv2.subtract(a,b)
array([[0]], dtype=uint8)

>>> c = np.array(250, "uint8")
>>> b + c
4
>>> cv2.add(b,c)
array([[255]], dtype=uint8)
```

Sometimes we need fractional values so we choose an IEEE 754 standard floating point number. They come in 3 sizes: 16, 32 or 64 bits.



$$p = M \cdot 2^{(E-E_0)}, M \in [0.5, 1)$$



- greyscale values typically in the range $[0, 1]$

Image processing

Histograms

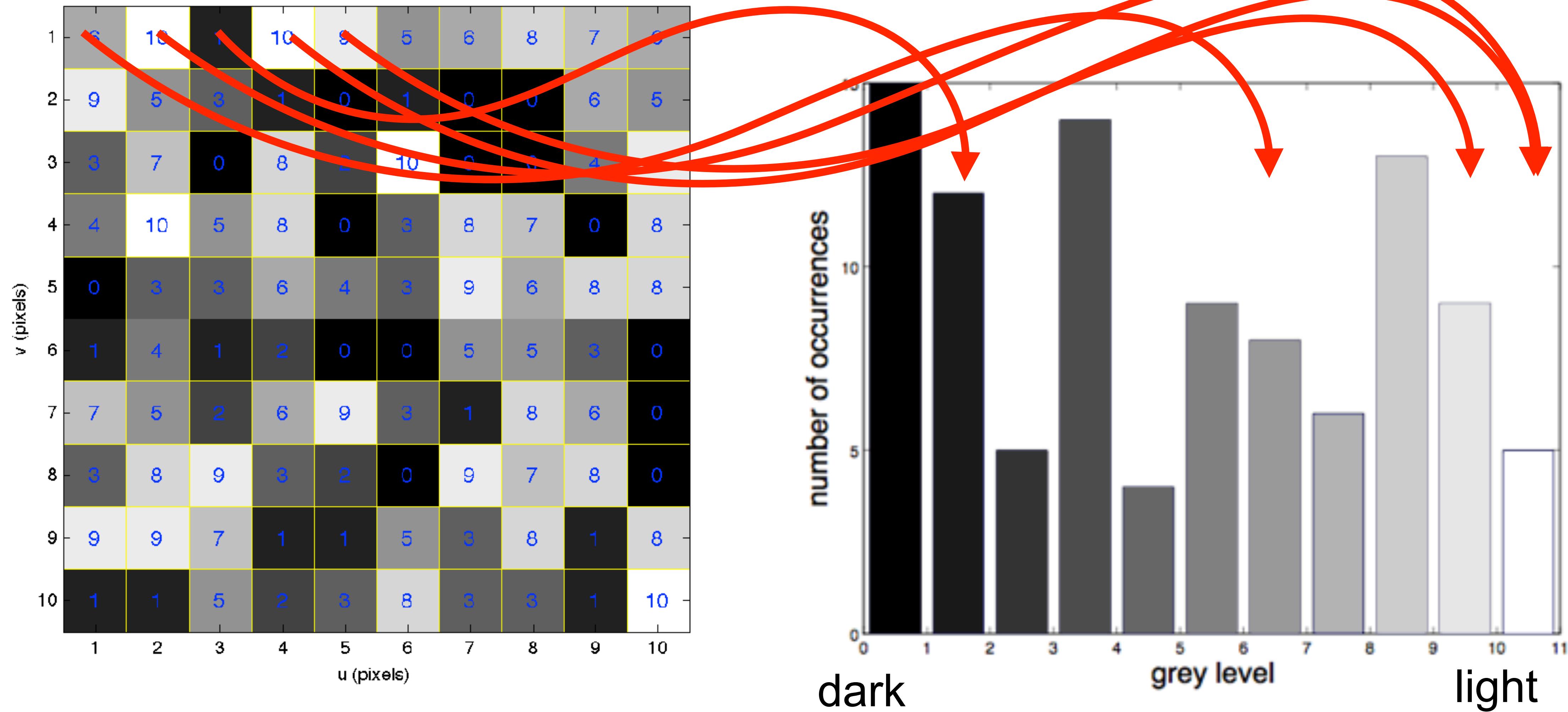
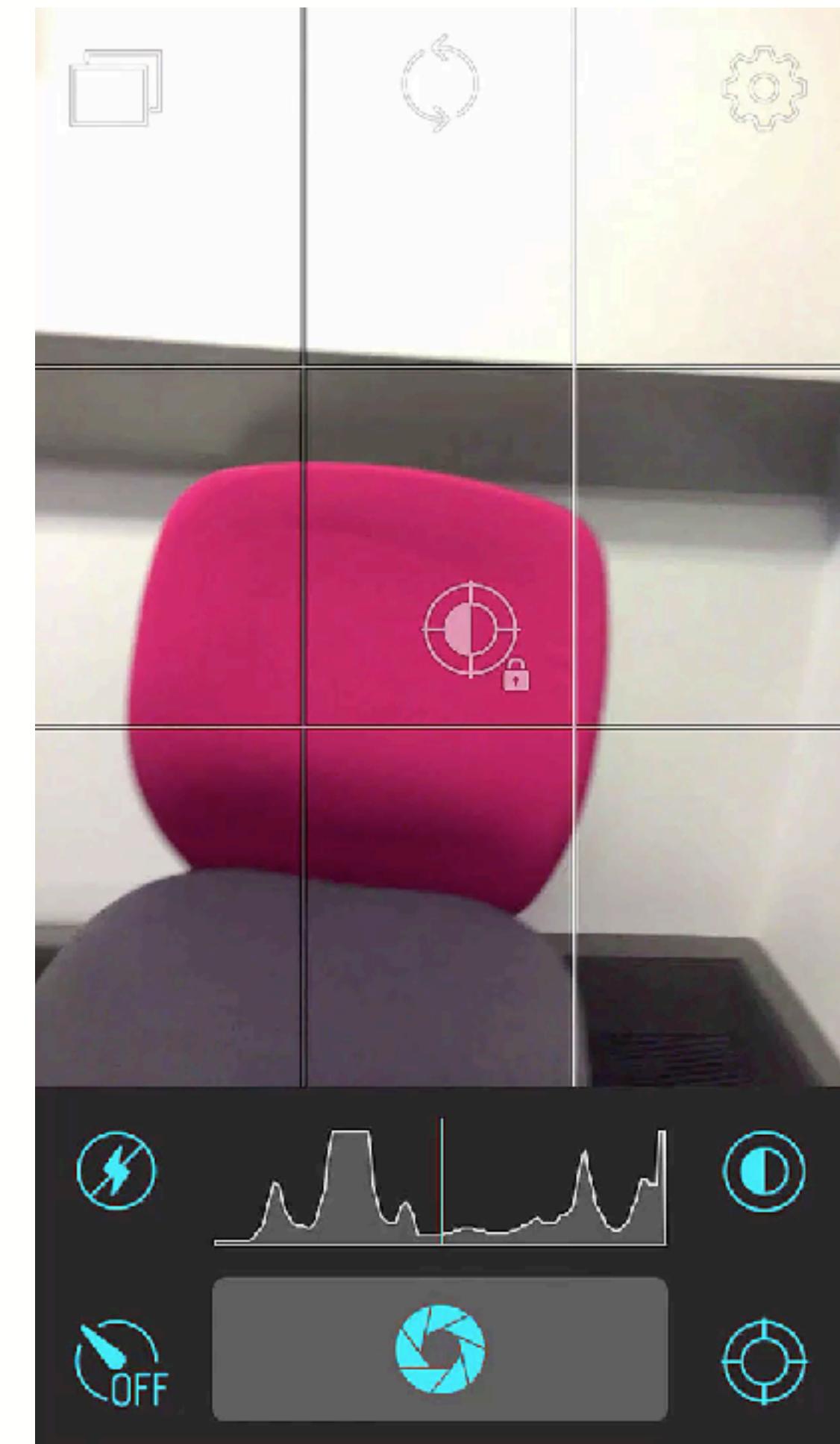
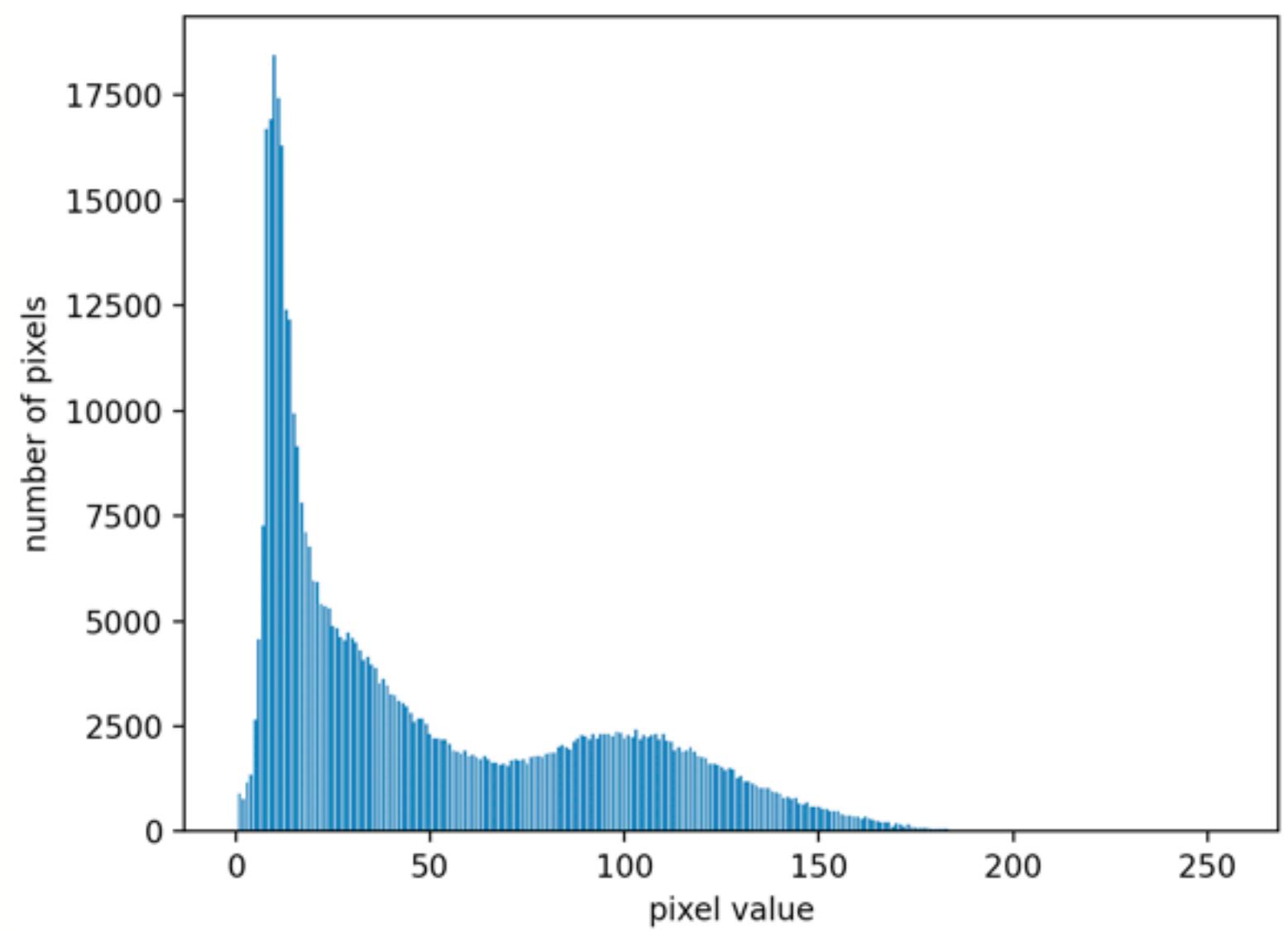
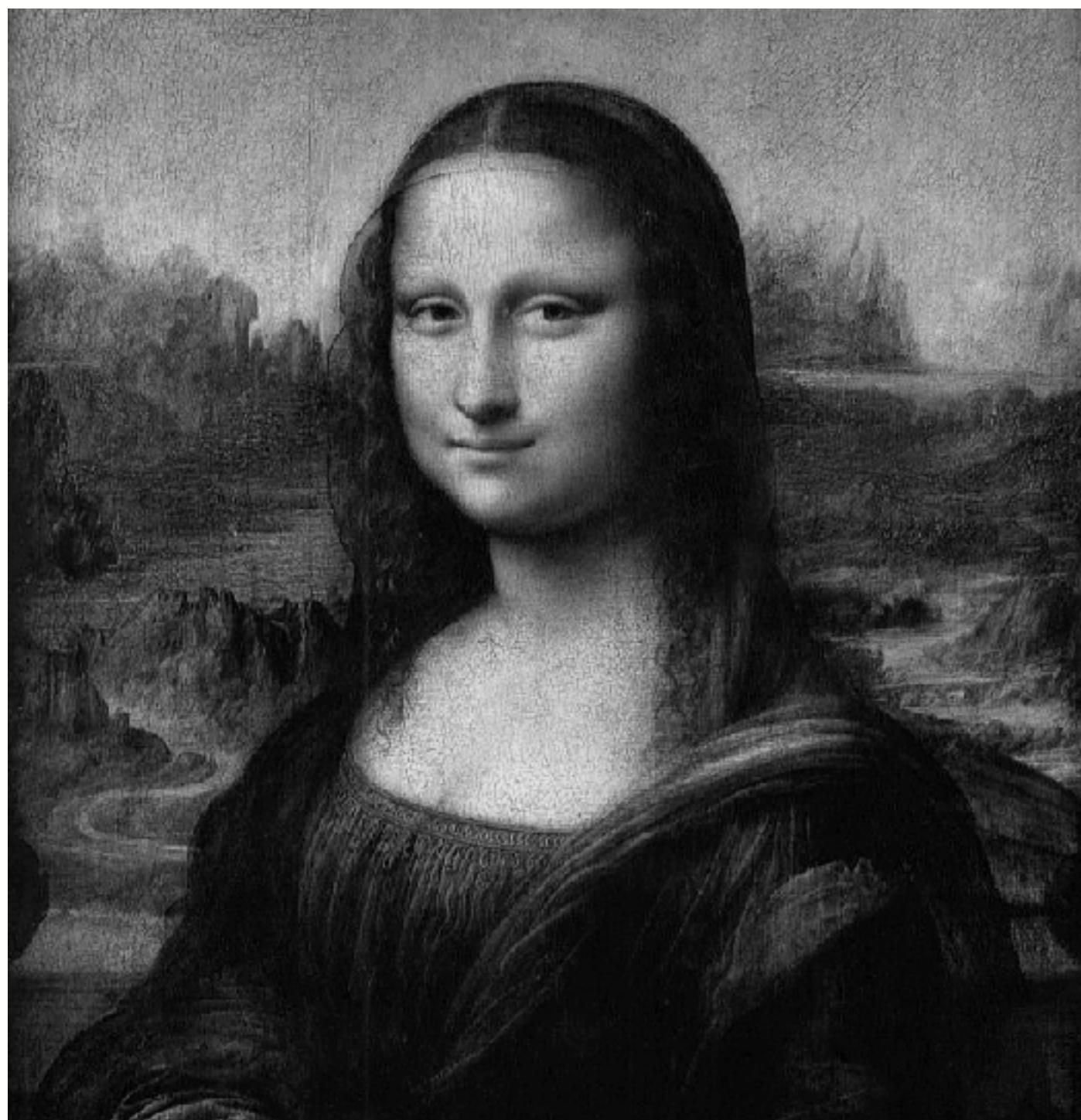
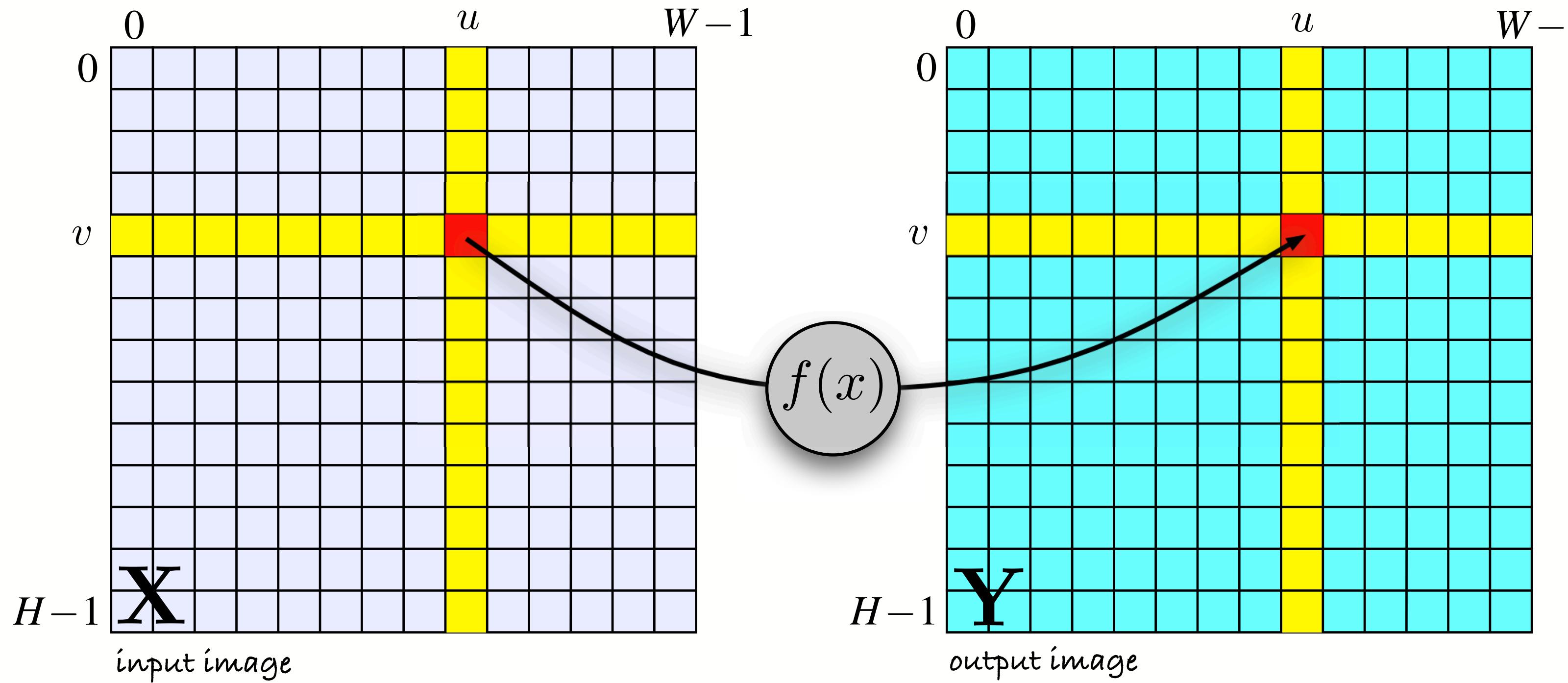


Image histograms

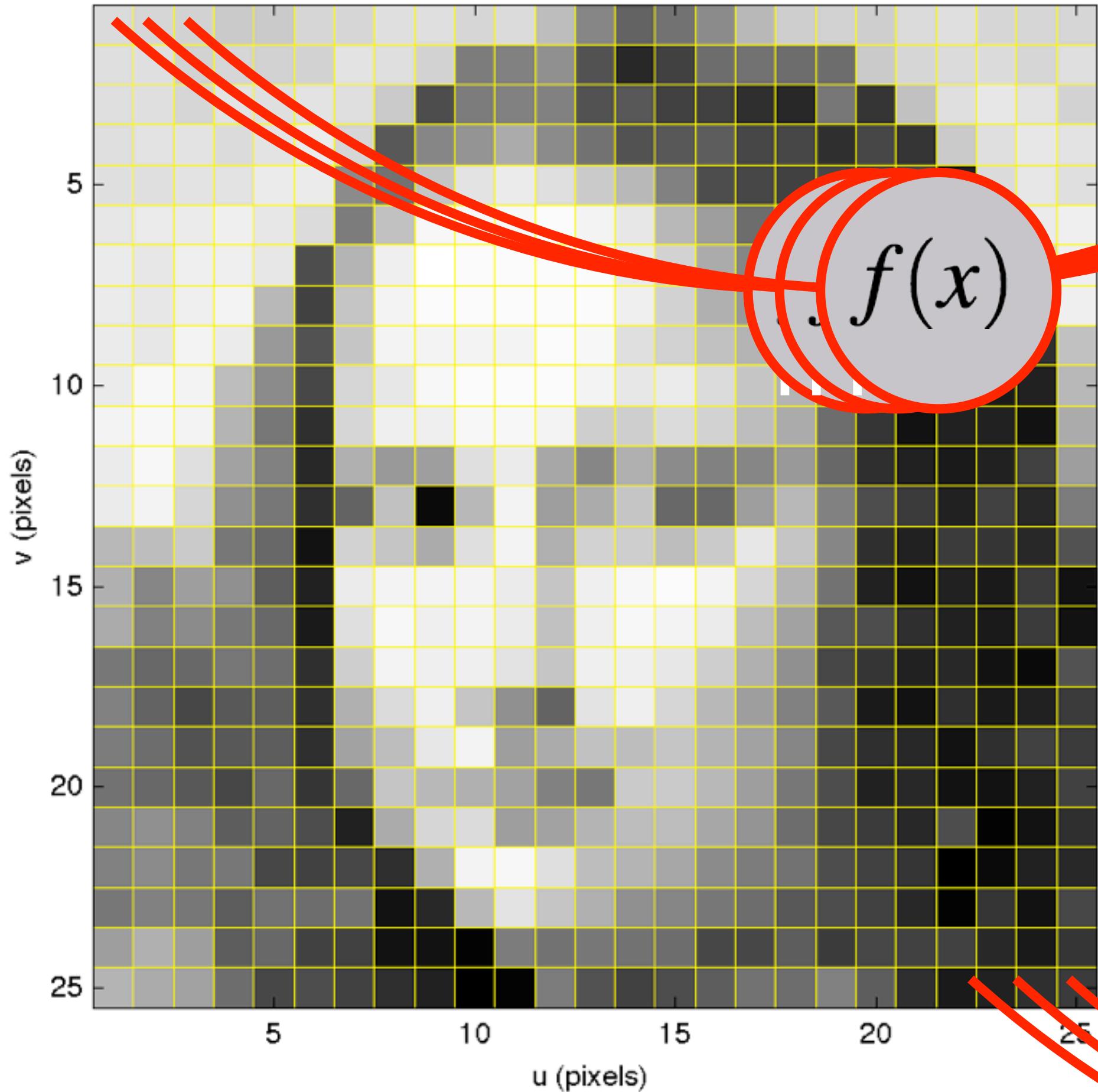


Monadic processing

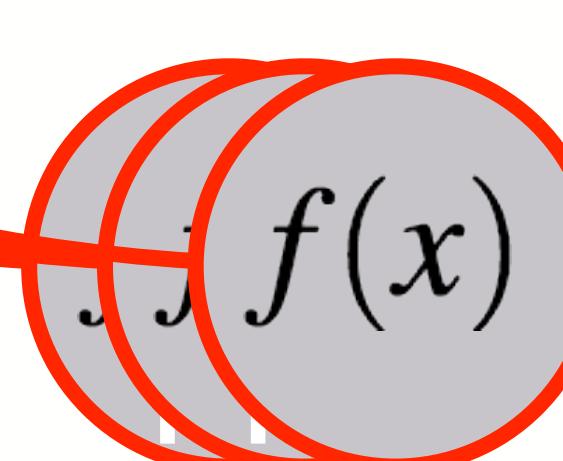
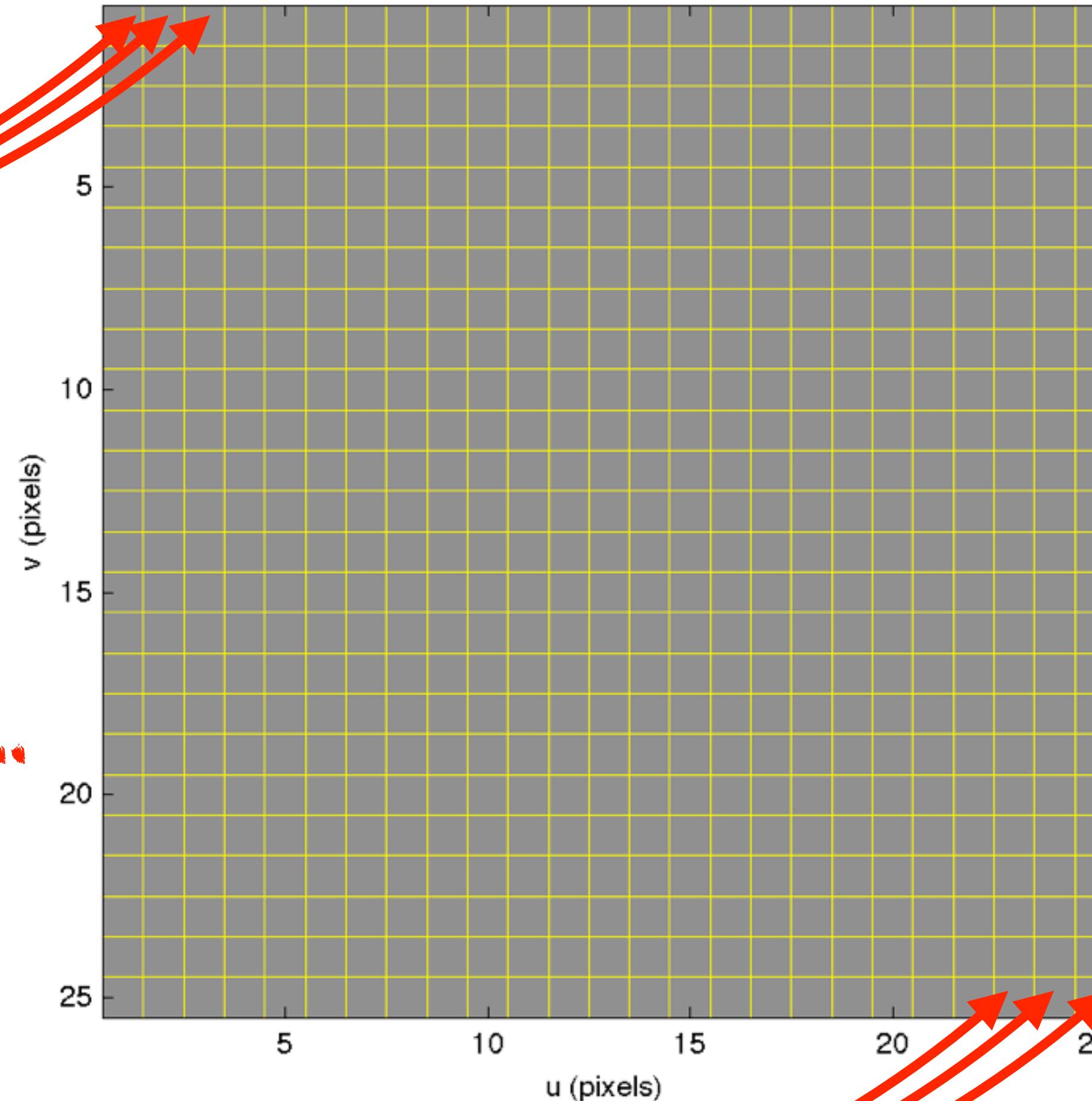


- Each output pixel is a function of corresponding input pixel
- The function is **the same** for all pixels

Monadic image processing

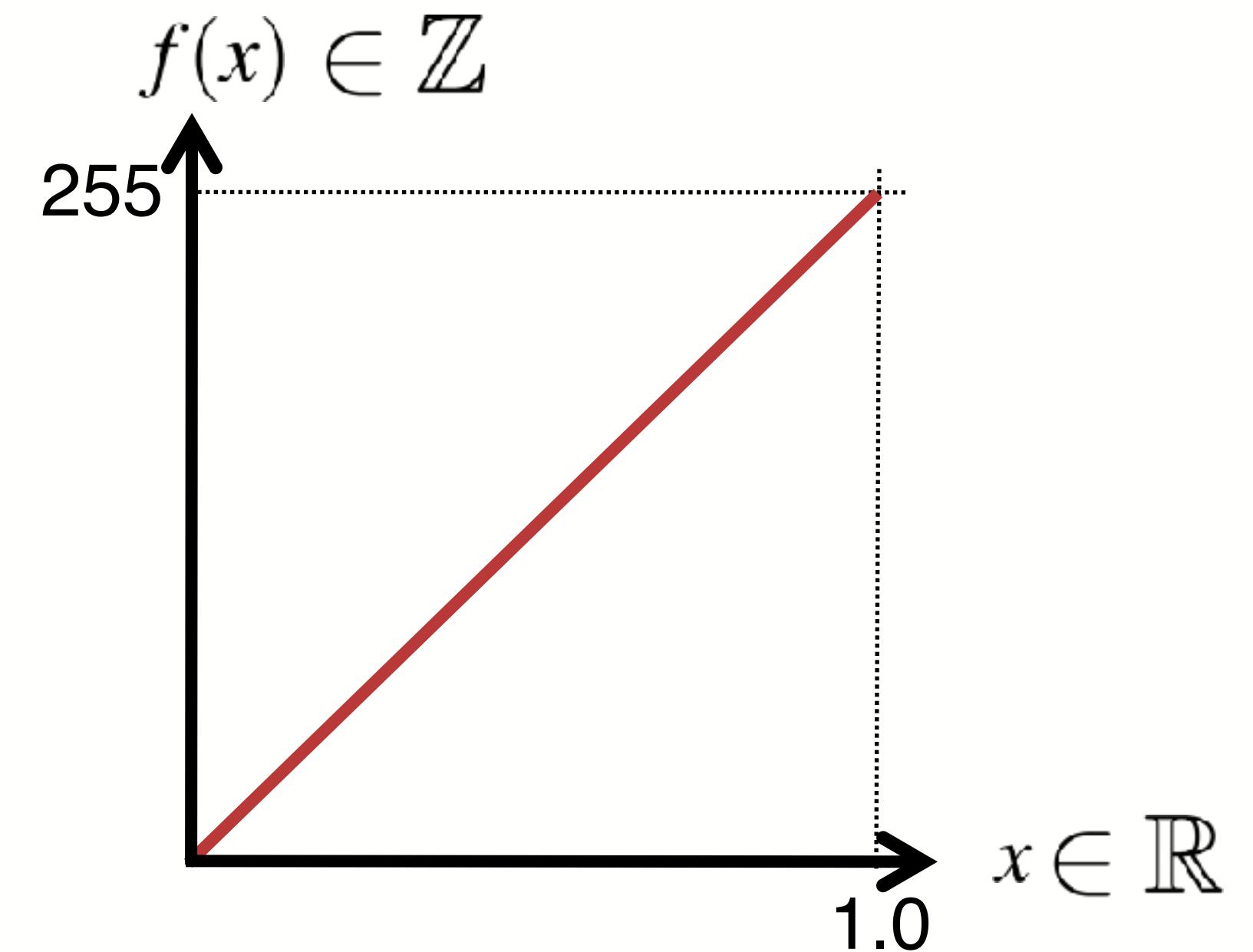
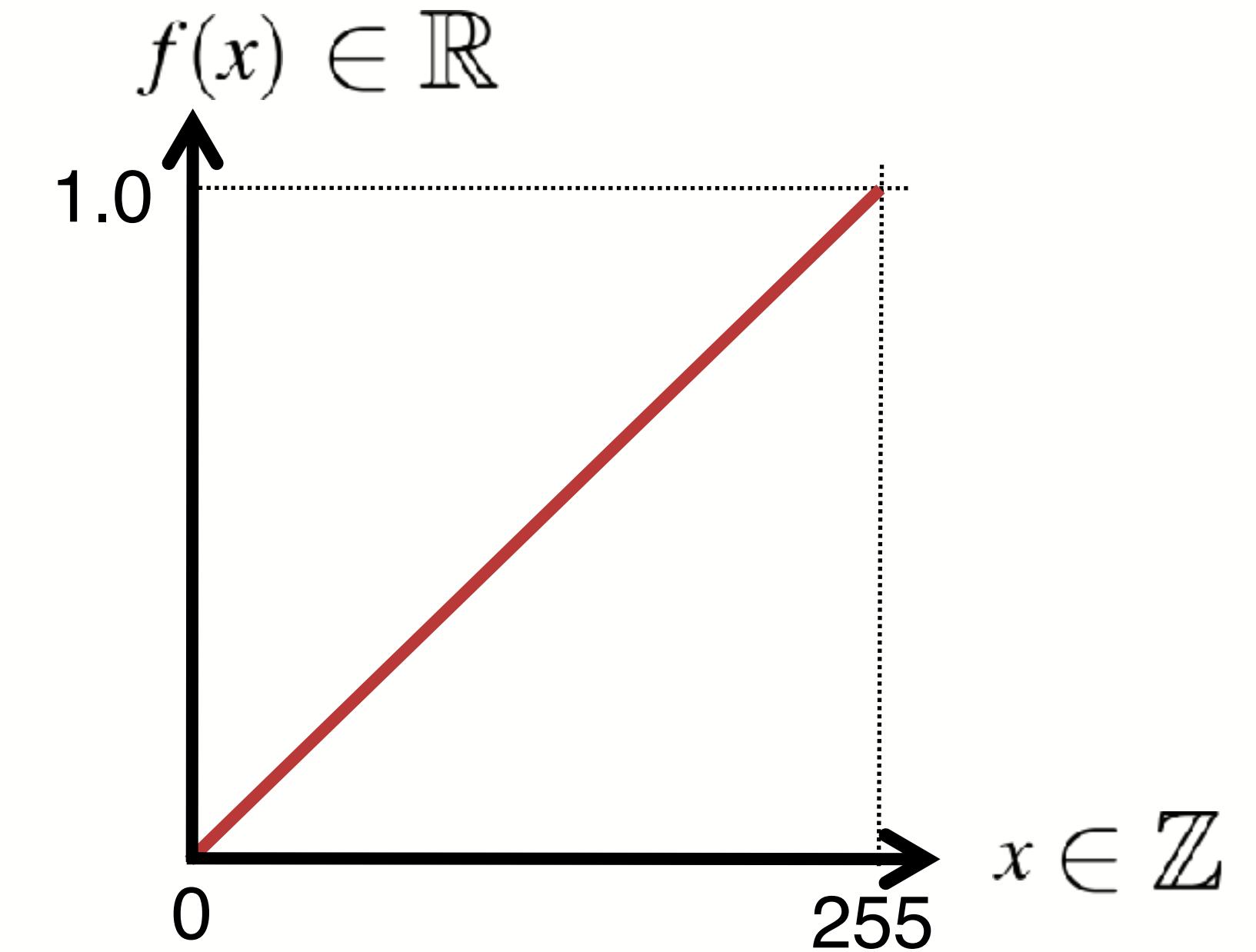


and
so
on ...



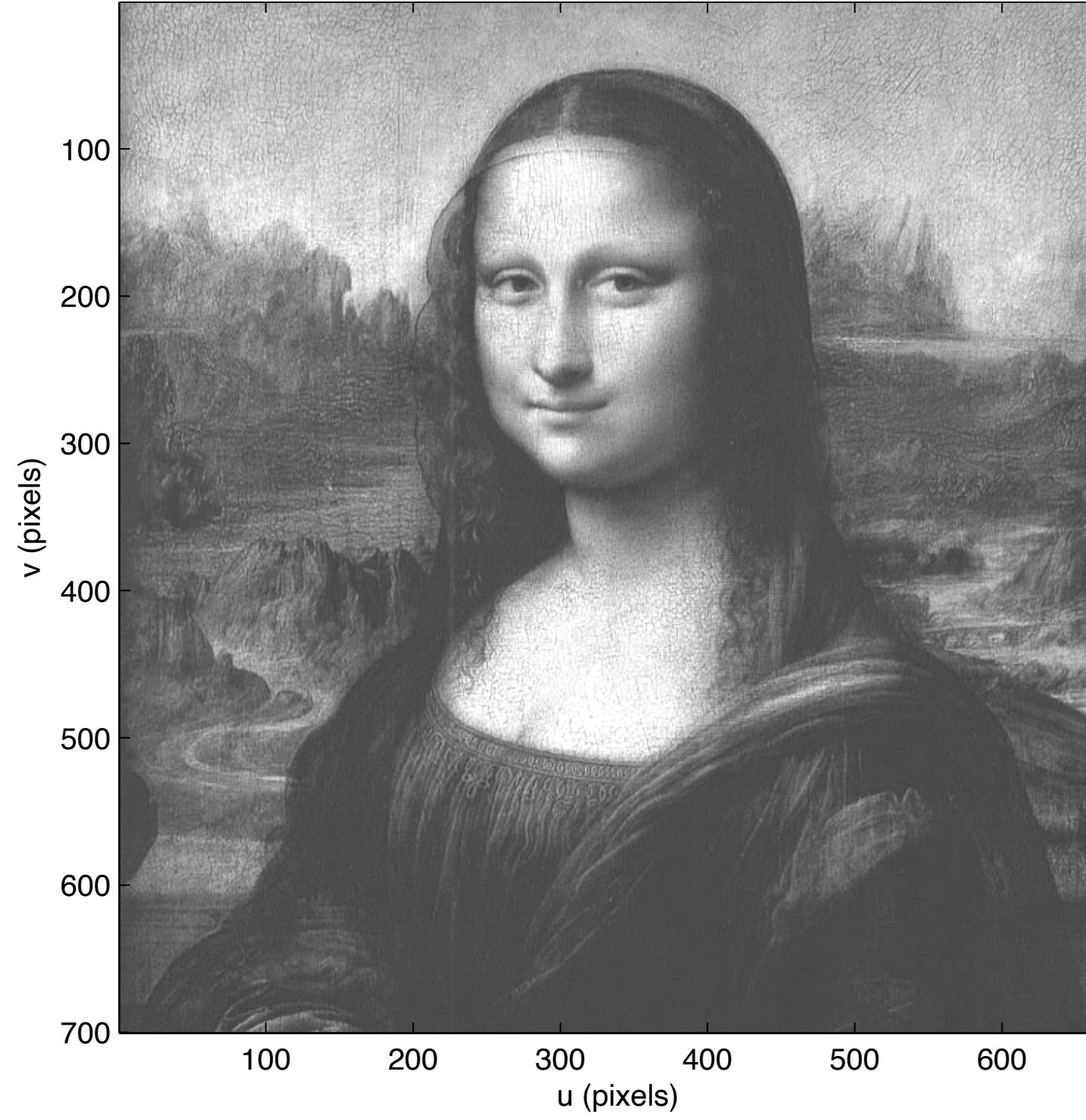
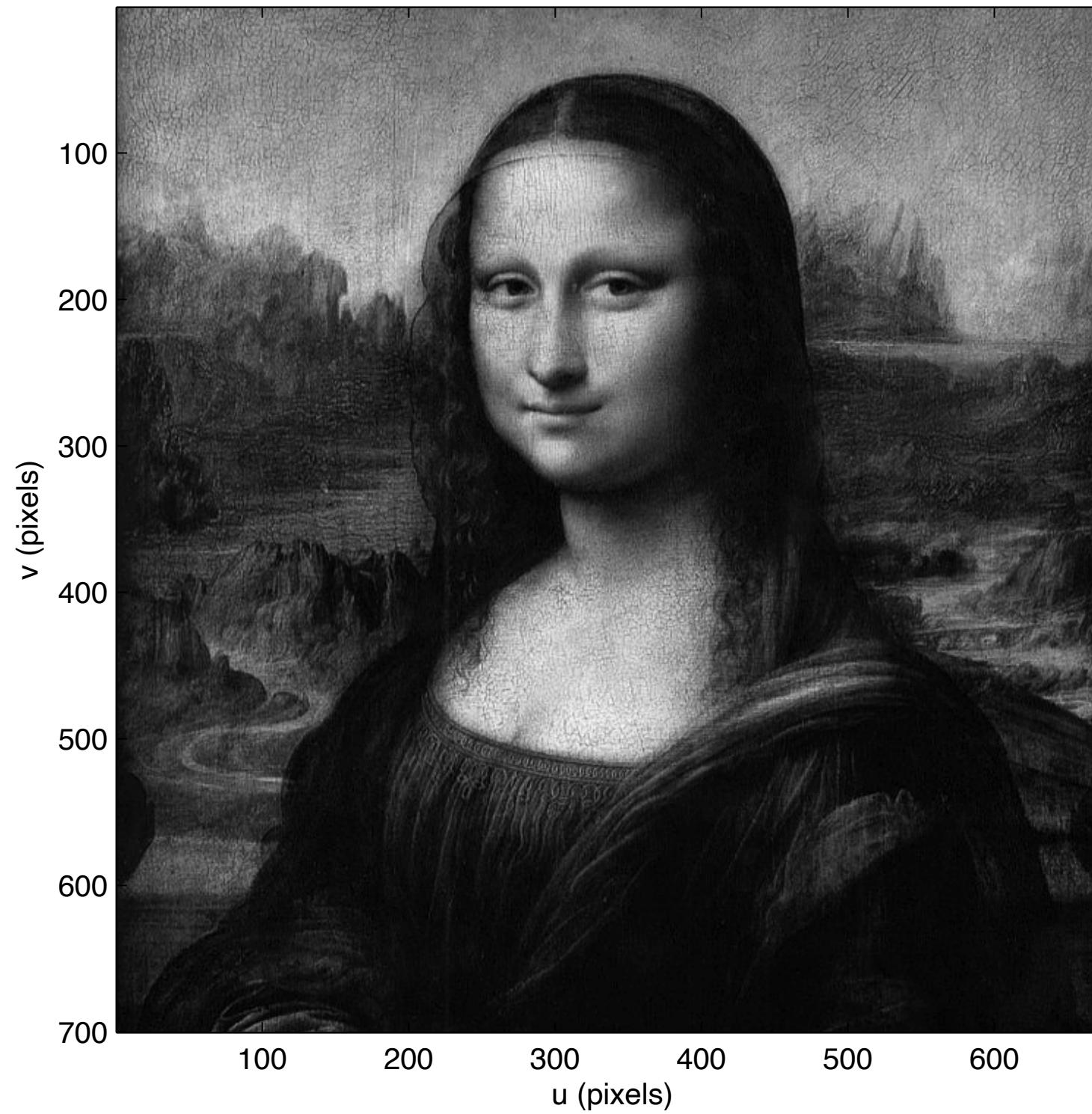
Changing pixel data type

- $\text{uint8} \rightarrow \text{float32}$
 - pixels in the range $[0, 1 \dots 255]$ mapped to $[0.0 \dots 1.0]$
- $\text{float32} \rightarrow \text{uint8}$
 - pixels in the range $[0.0 \dots 1.0]$ mapped to $[0, 1 \dots 255]$

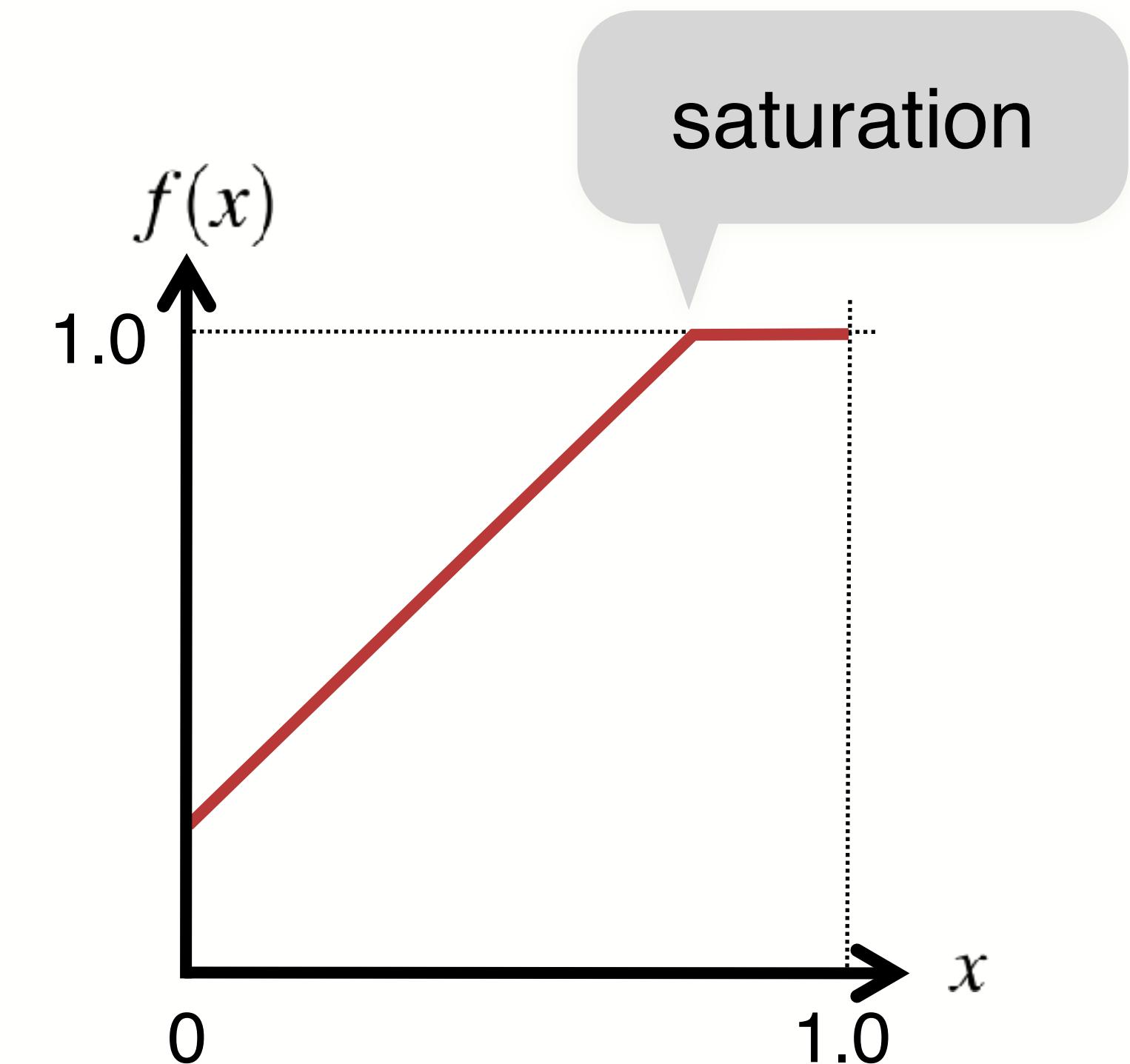


Changing brightness

- by offsetting the grey values
- $f(x) = x + 0.25$

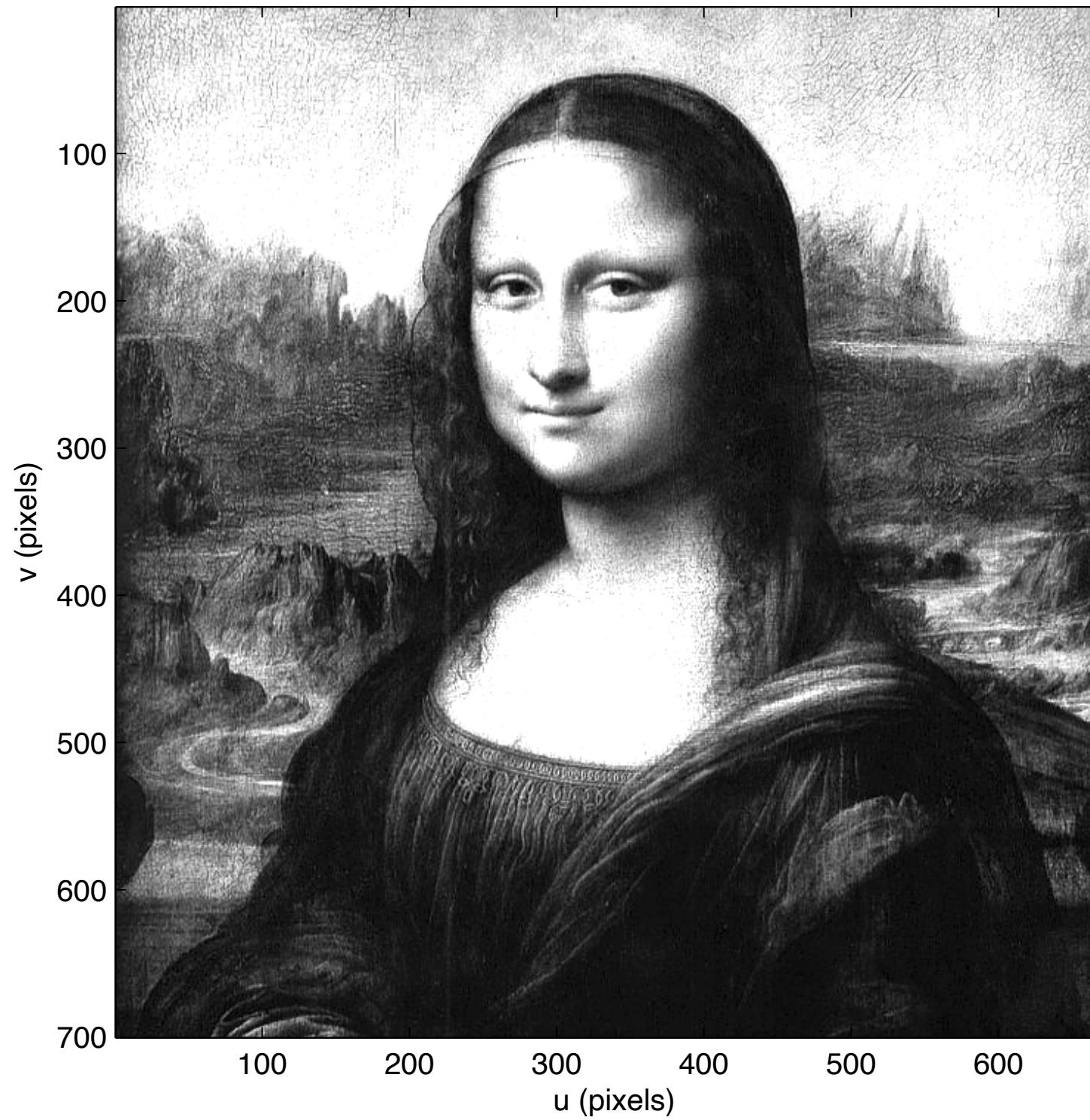
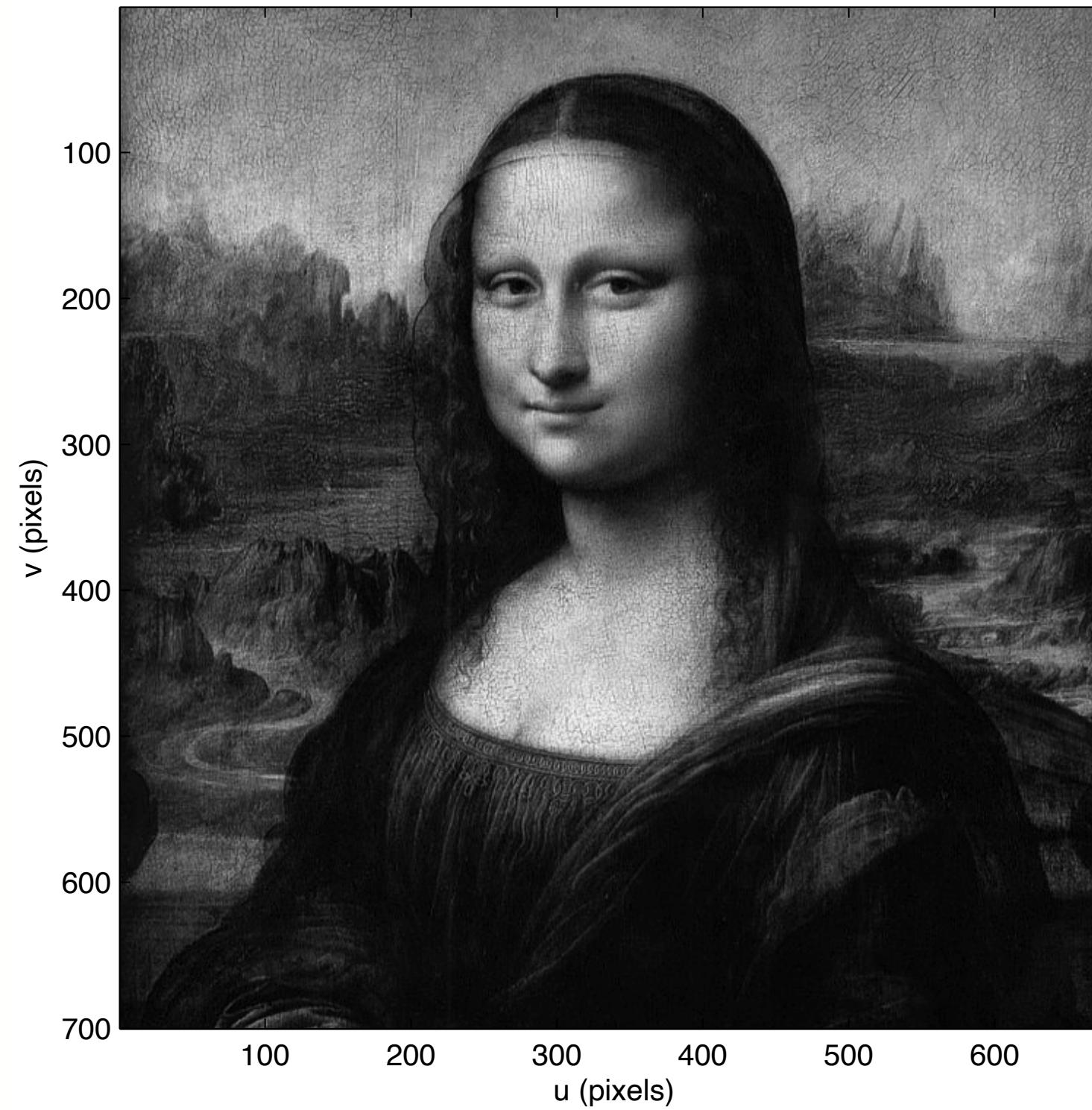


Mona Lisa c.1503–1505
Da Vinci, L.

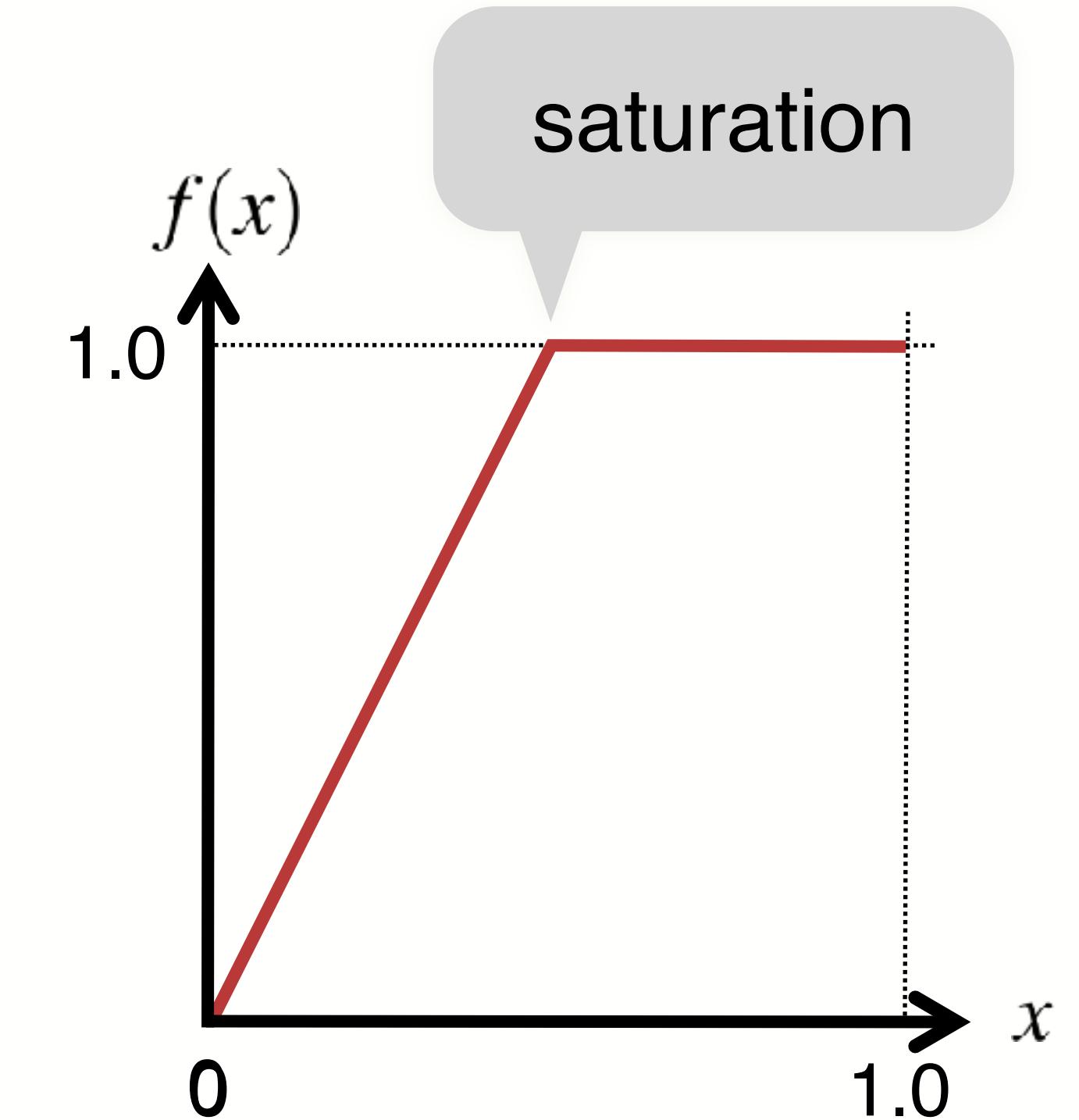


Changing contrast

- by scaling the grey values
→ $f(x) = 2x$

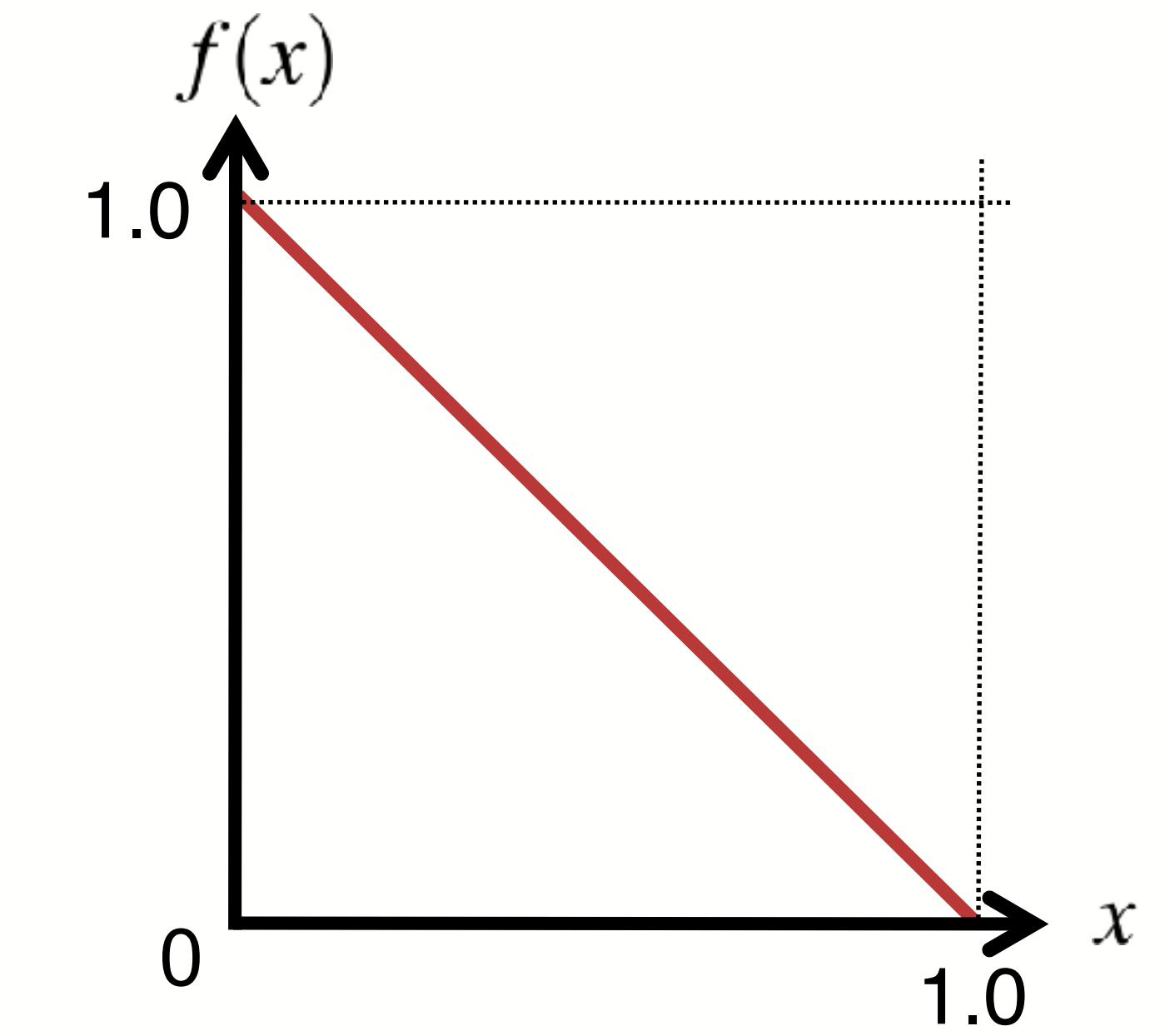
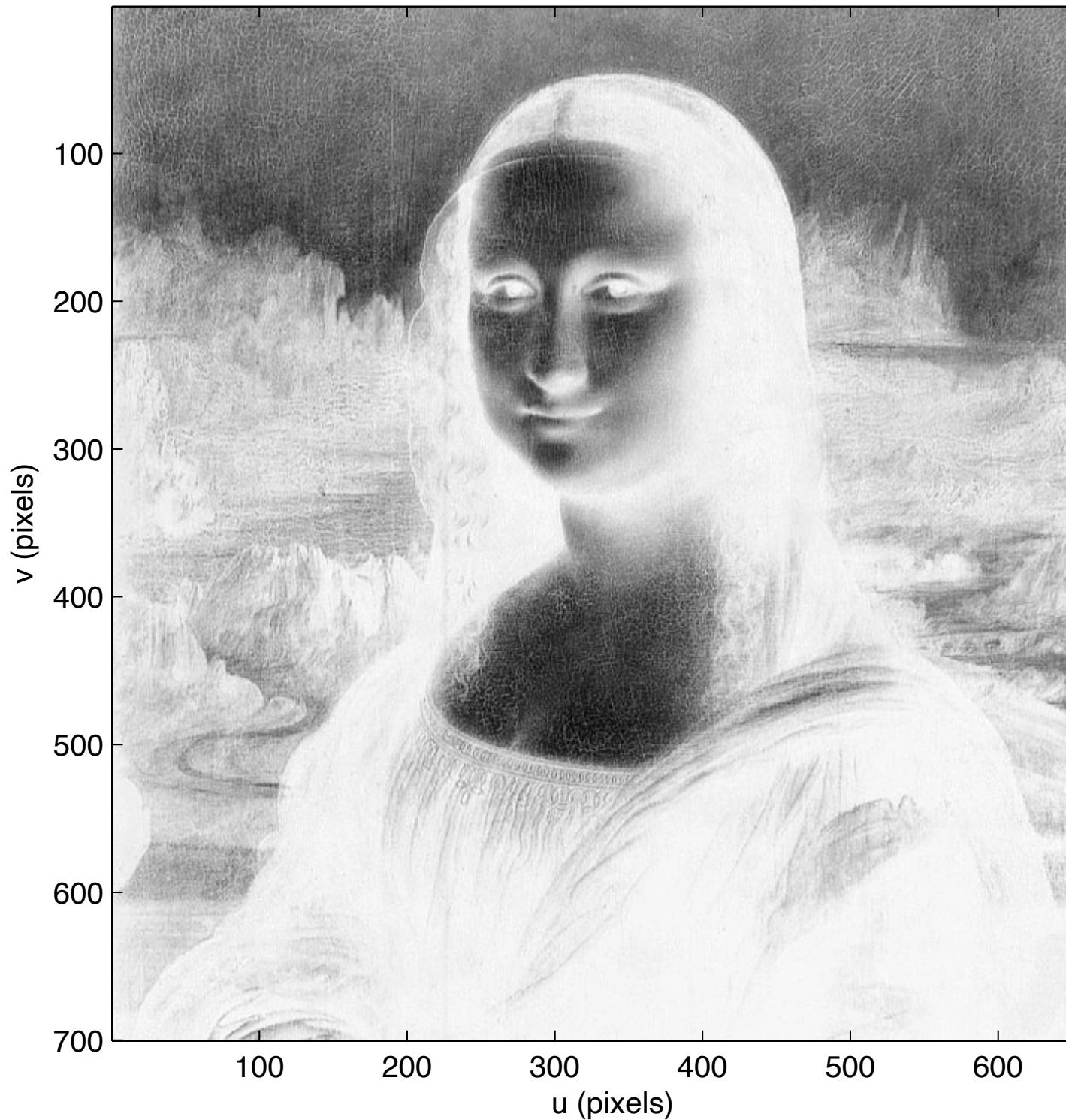
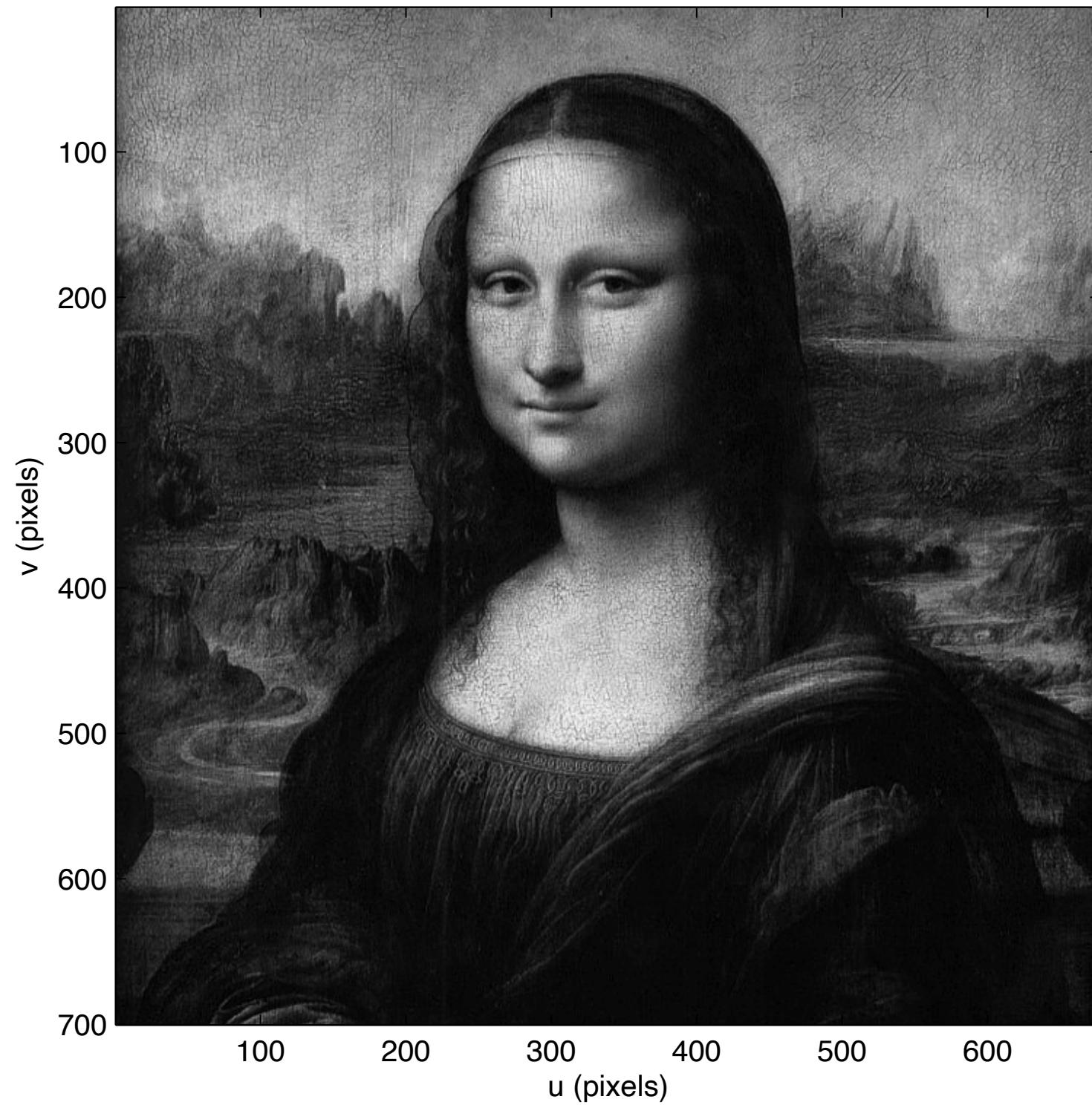


Mona Lisa c.1503–1505
Da Vinci, L.



Negative image

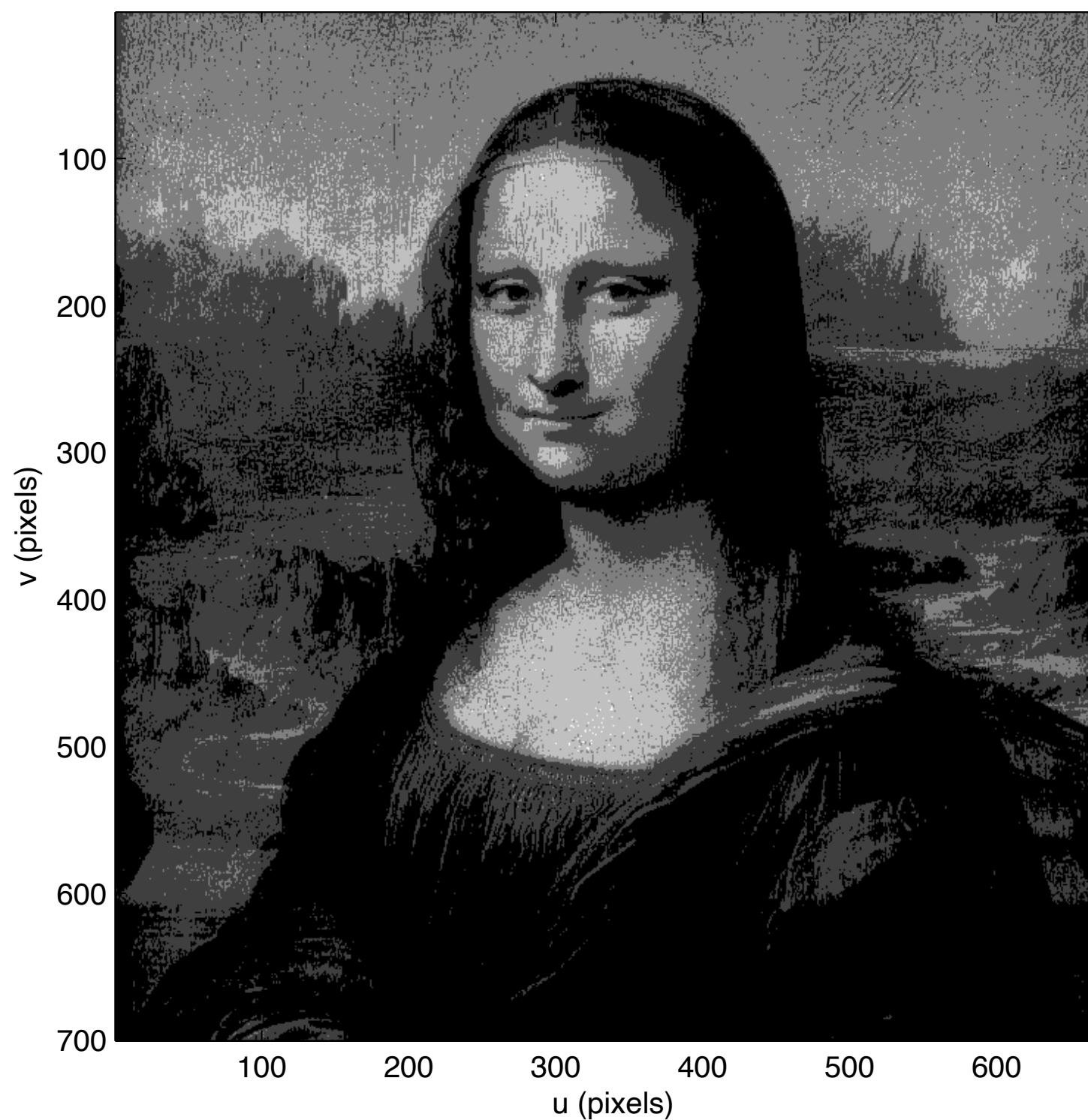
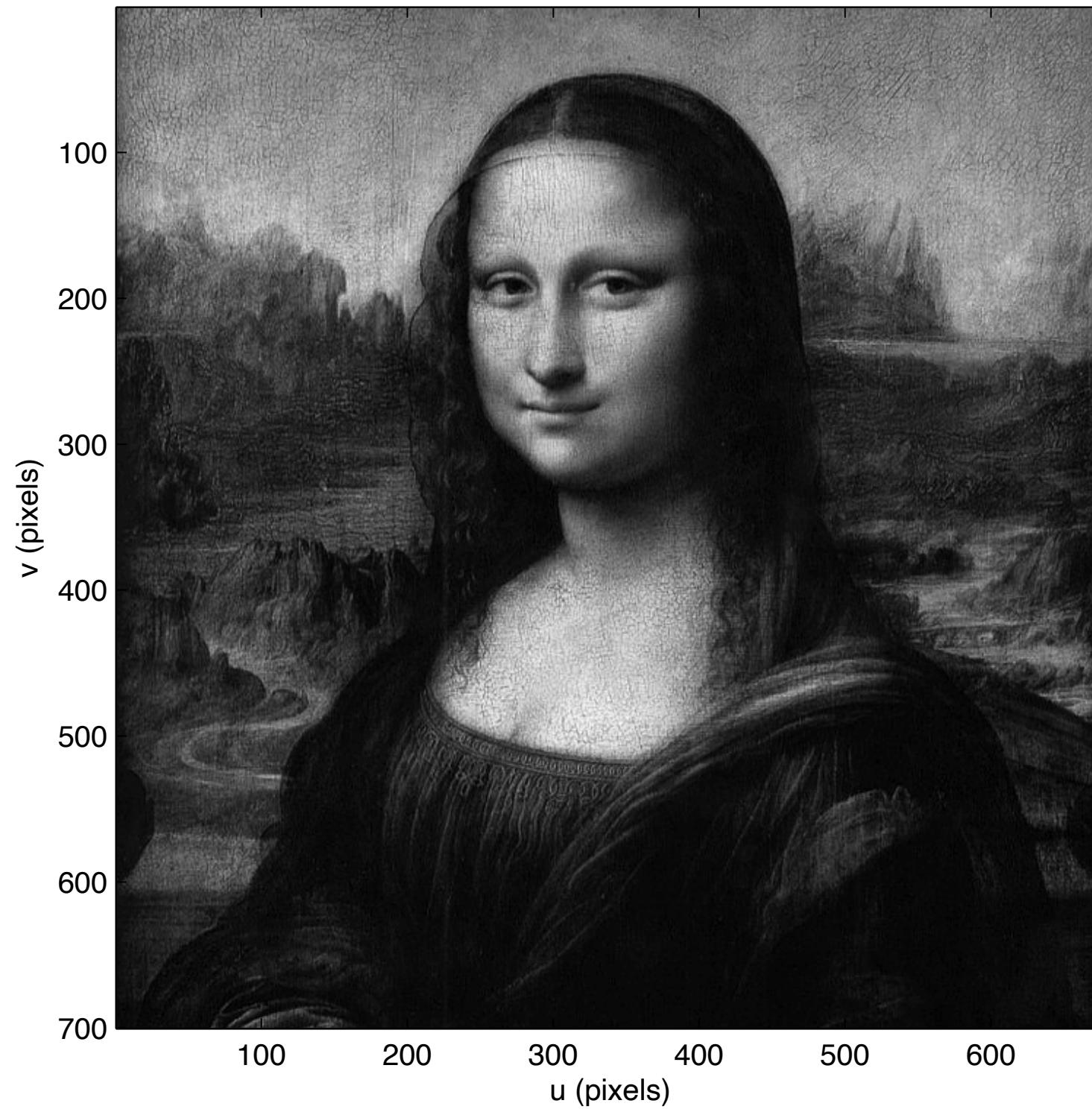
- $f(x) = 1-x$



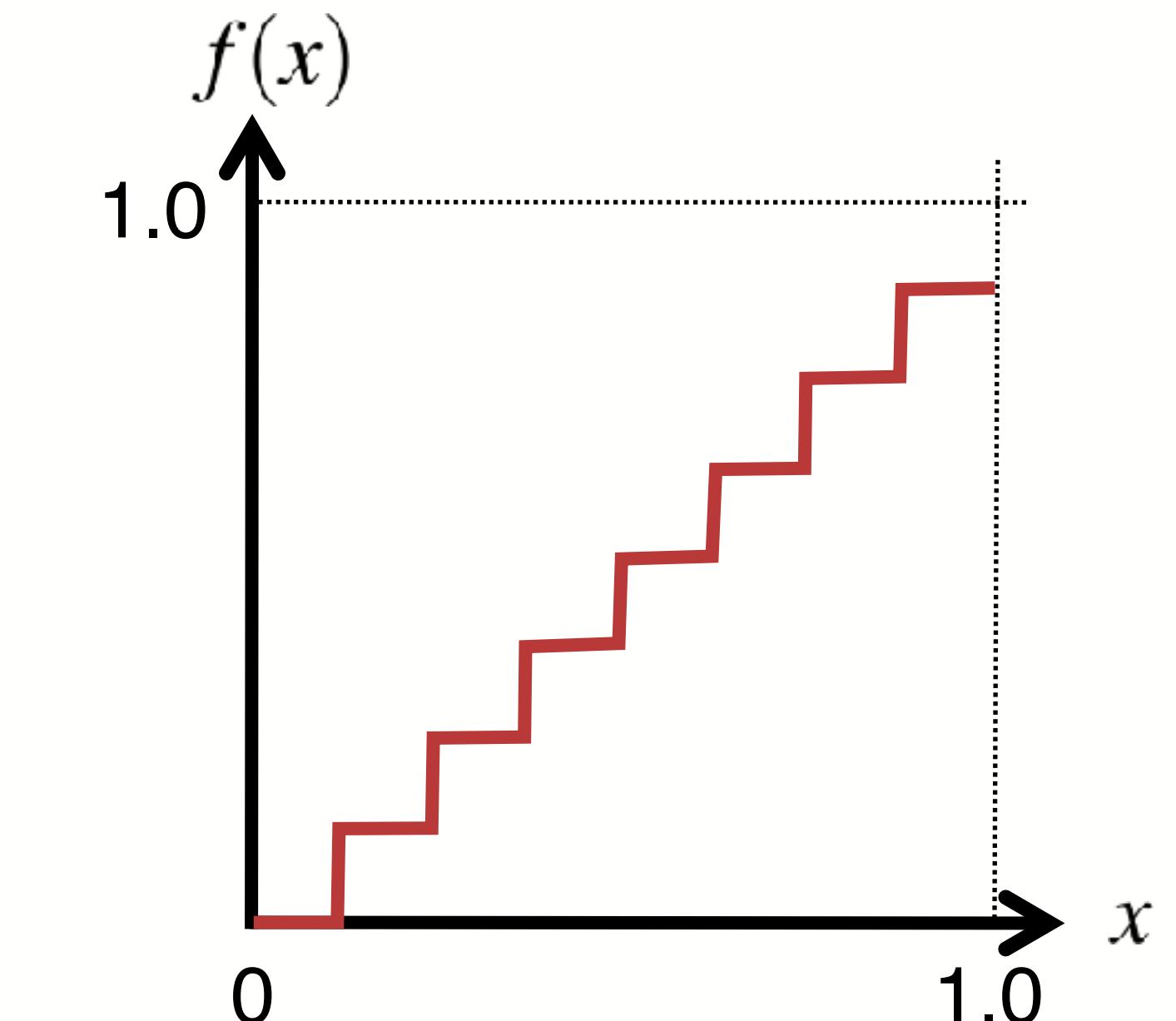
Mona Lisa c.1503–1505
Da Vinci, L.

Posterisation

- Only N unique grey levels in the output image
→ $f(x) = N \text{ floor}(x/N)$



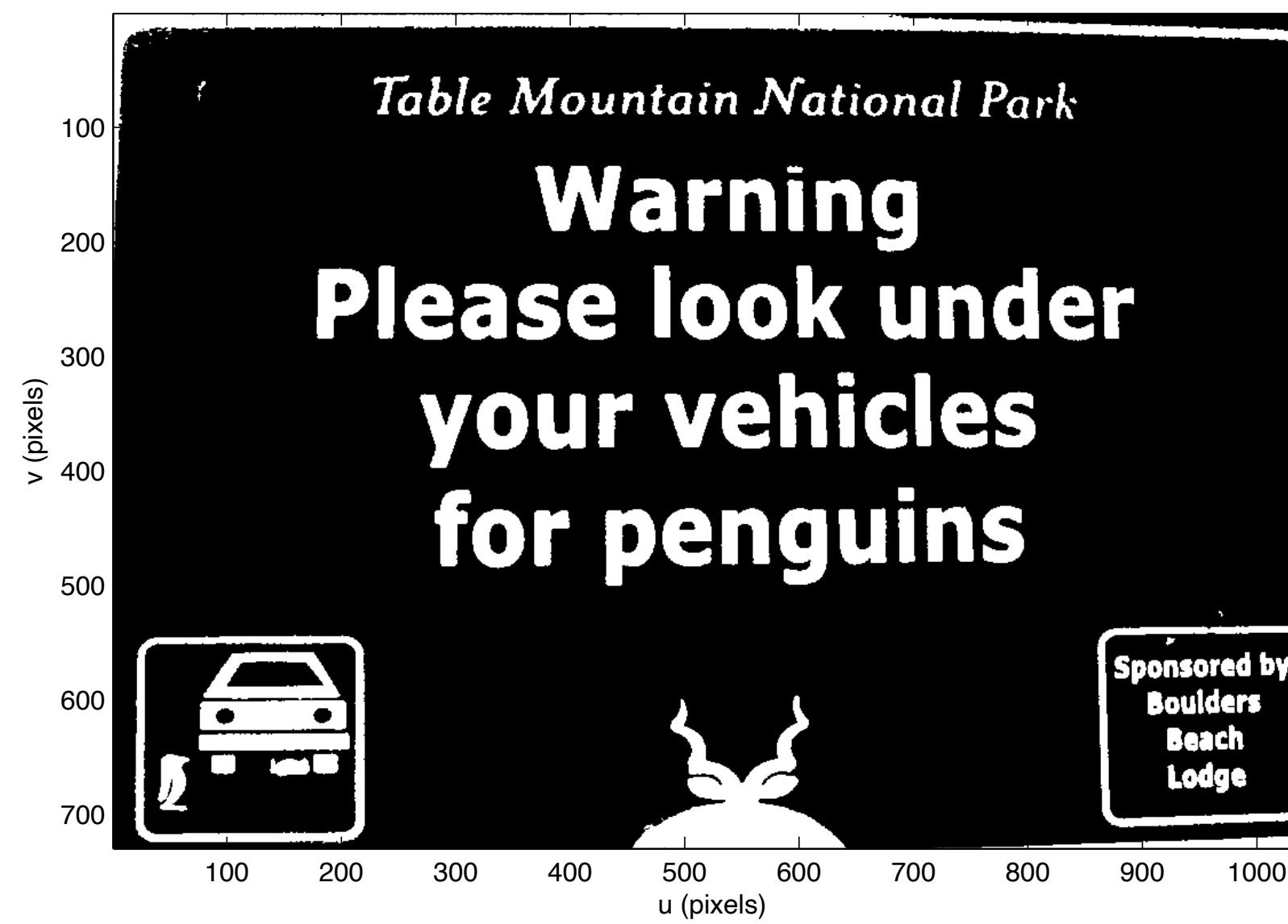
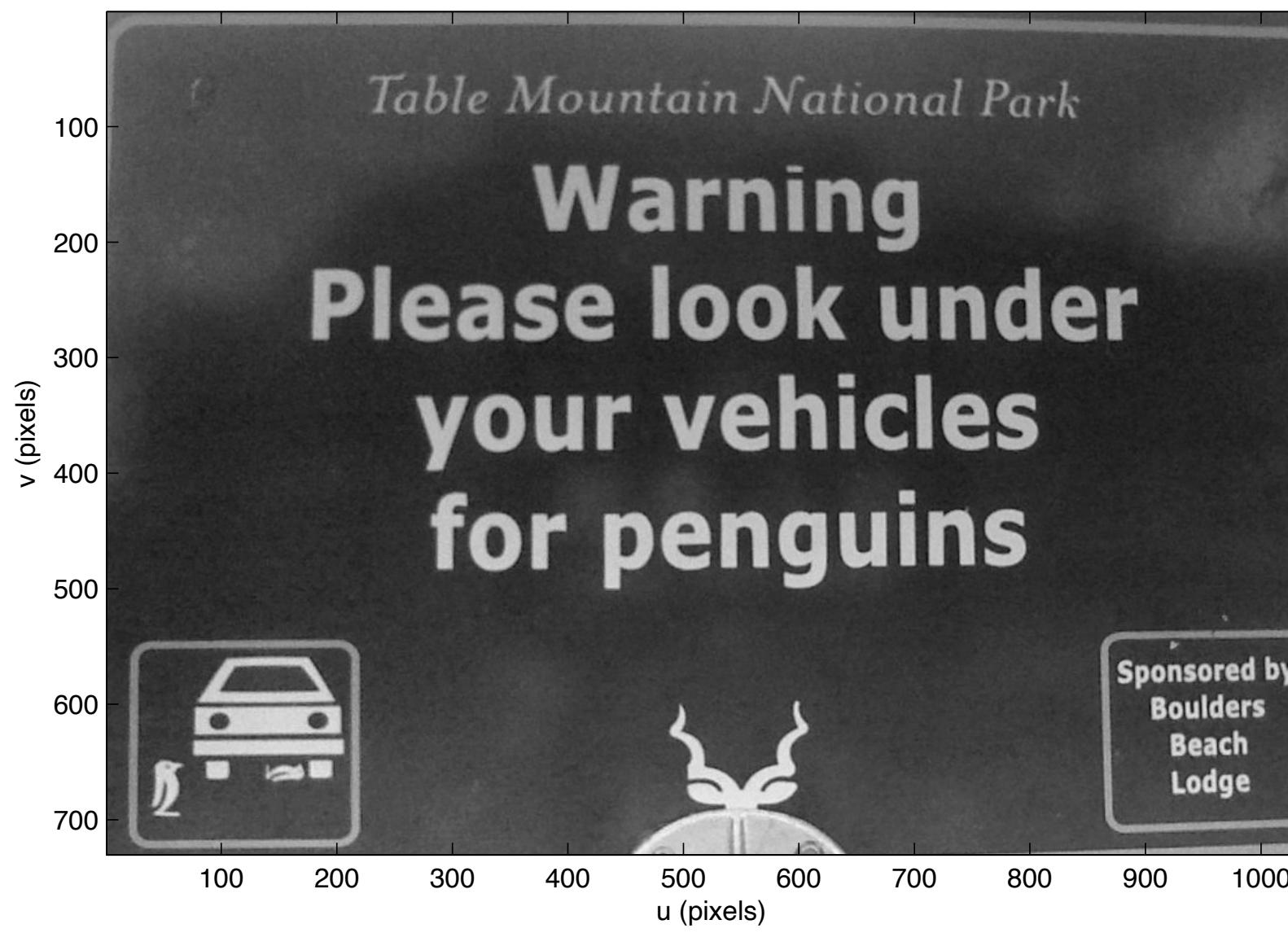
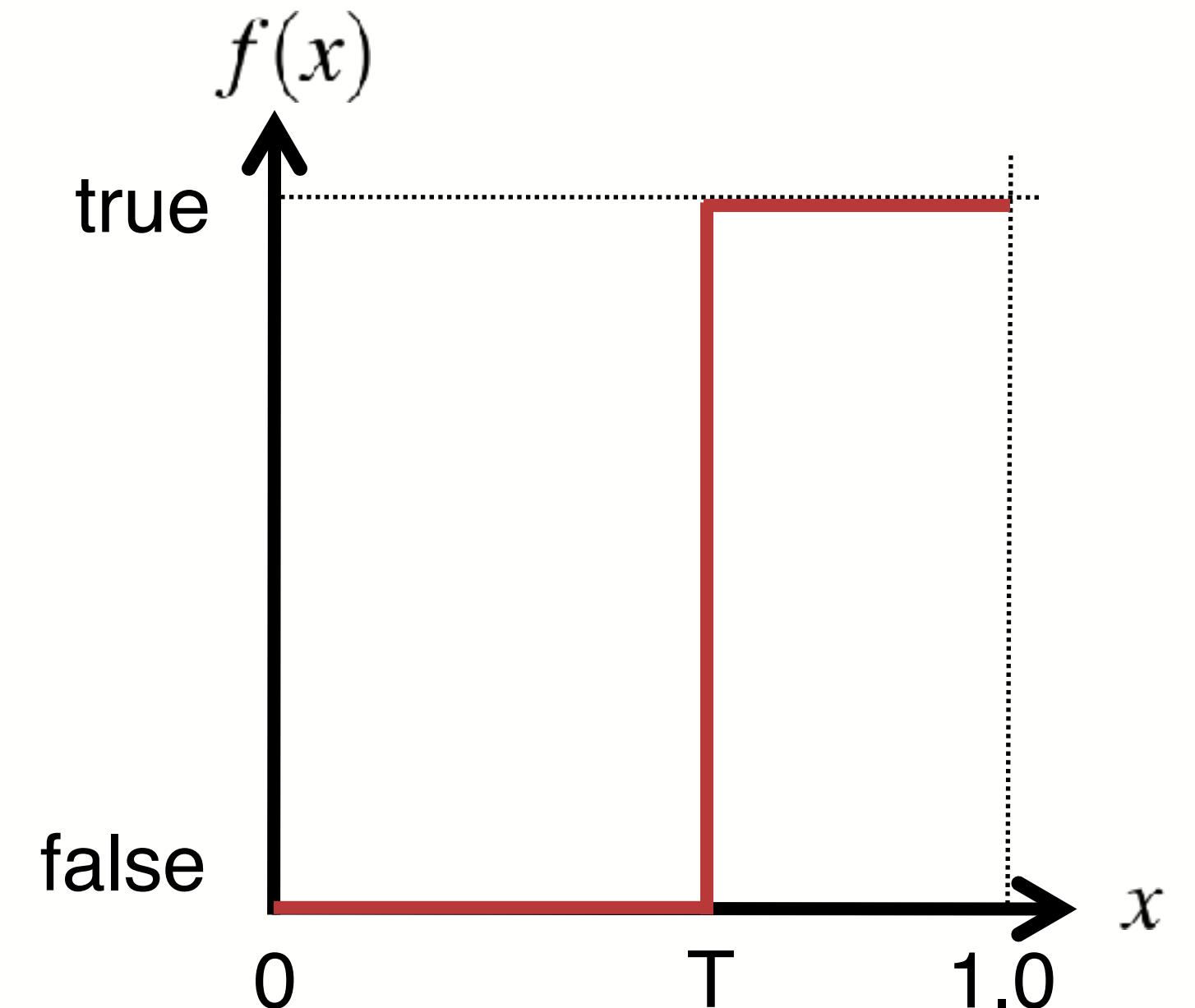
N=4



Mona Lisa c.1503–1505
Da Vinci, L.

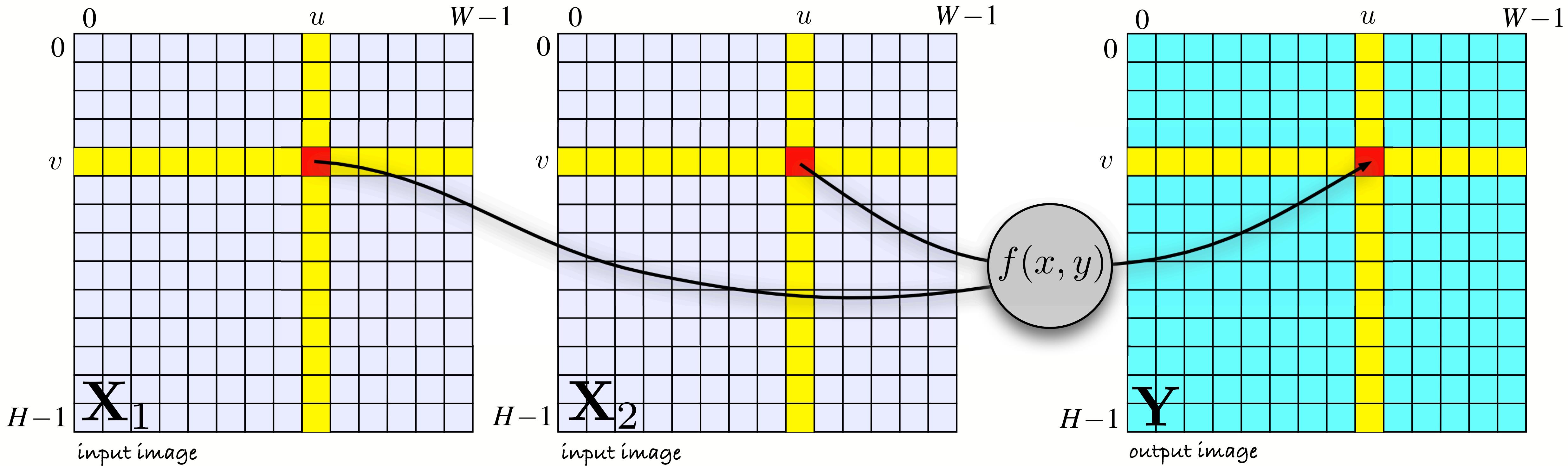
Thresholding

- The resulting image has only two values:
 - false (black) if $x < T$
 - true (white) if $x \geq T$



$T = 0.4$

Diadic operations



- Each output pixel is a function of two corresponding input pixels
- The function is **the same** for all pixels

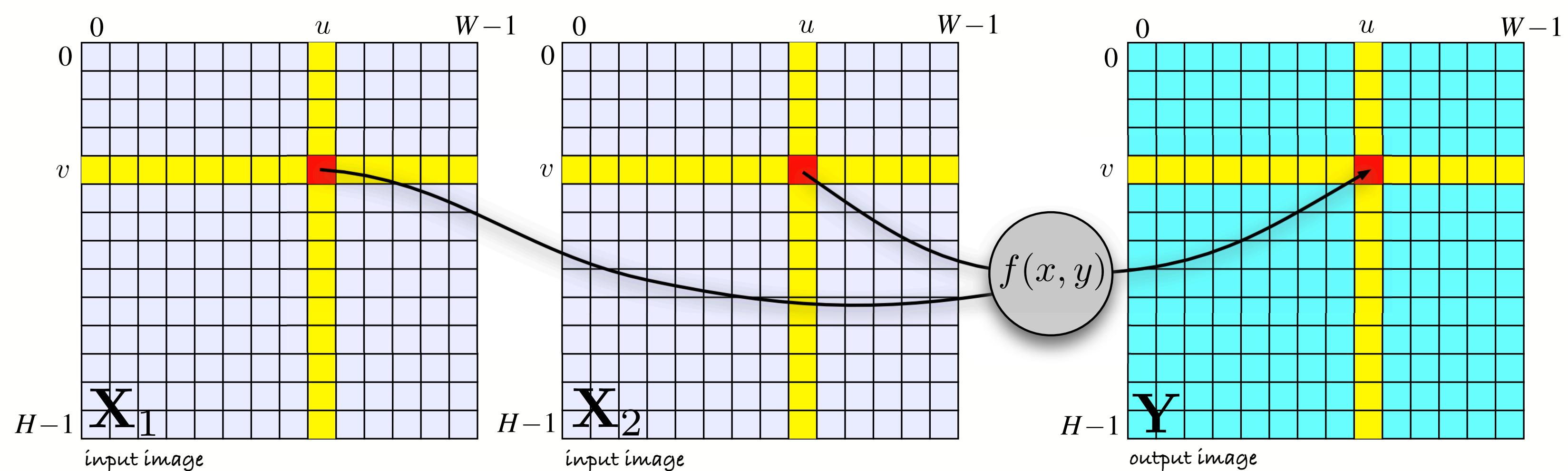
Diadic image processing

- What is $f(x,y)$?

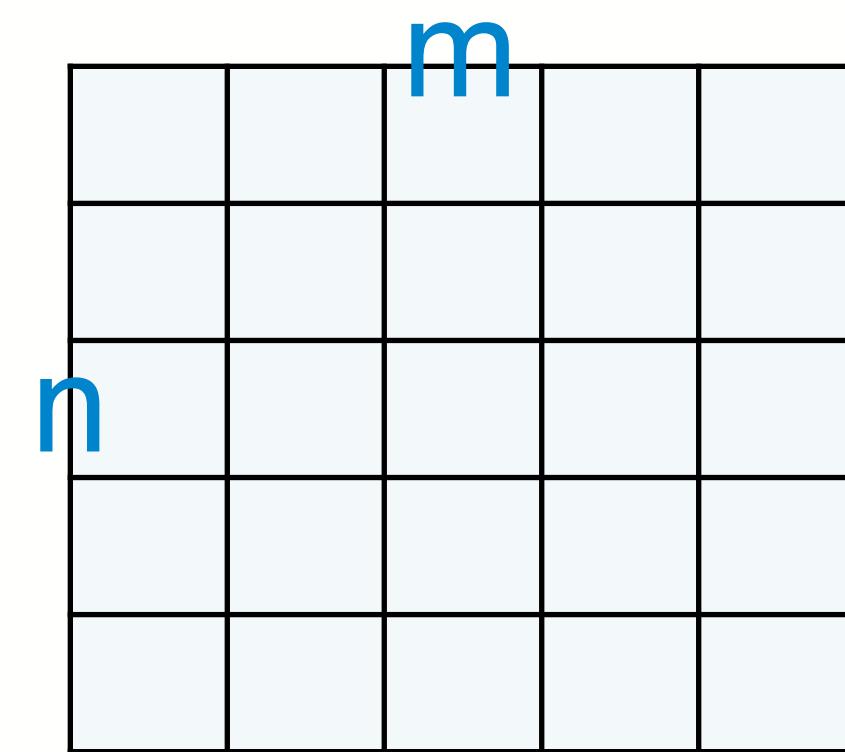
→ arithmetic

- $x+y$
- $x-y$
- x^*y
- etc.

→ $\sqrt{x^2 + y^2}$

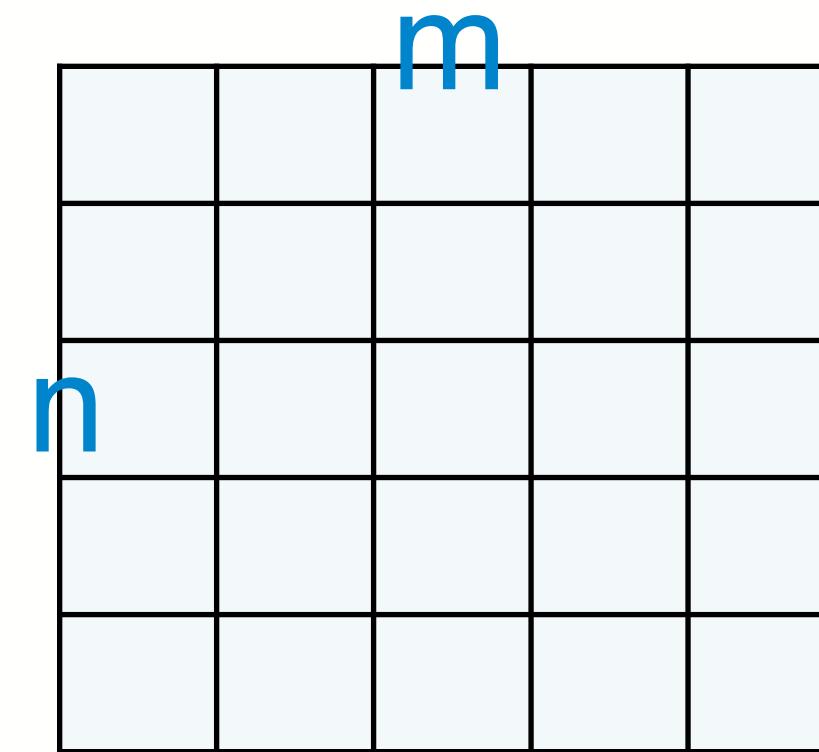


NumPy array operations

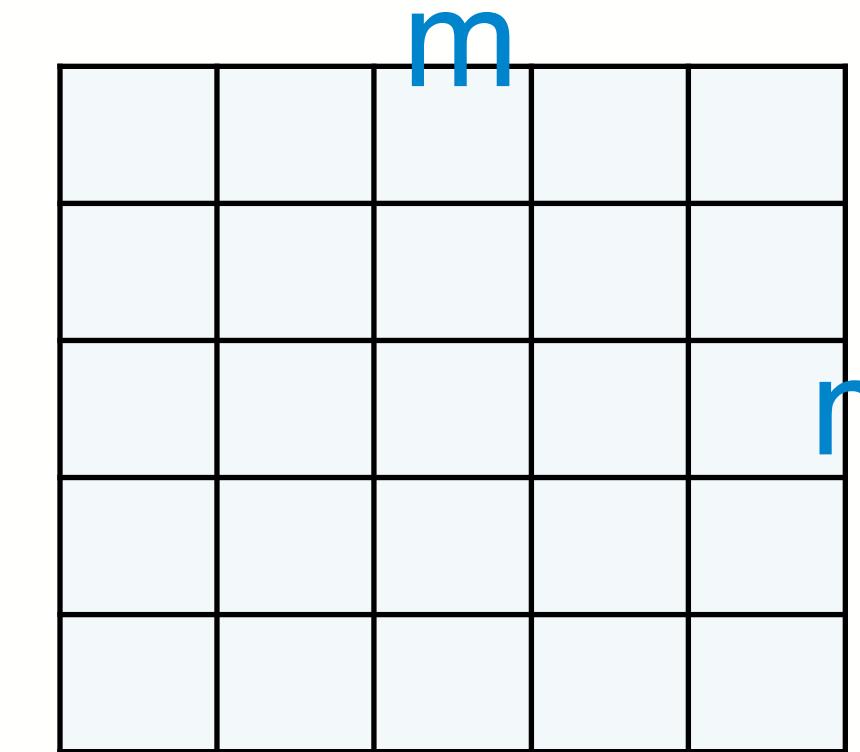


A vertical list of operators in blue:

- +
-
- *
- /
- >
- \geq
- <
- \leq
- $=$
- \neq



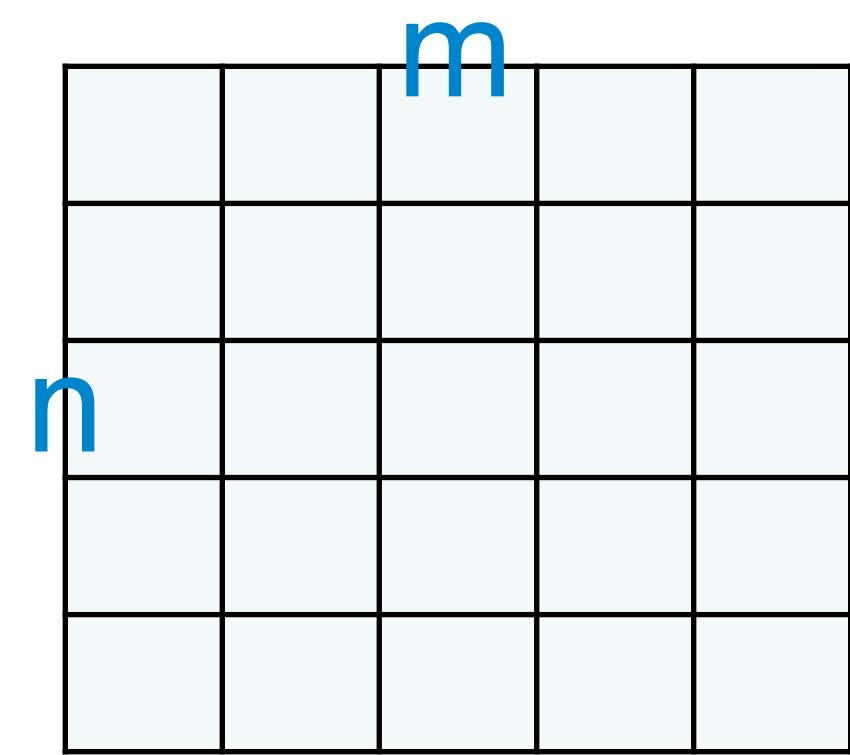
$=$



Both operands must have the same size

The output is the same size as well

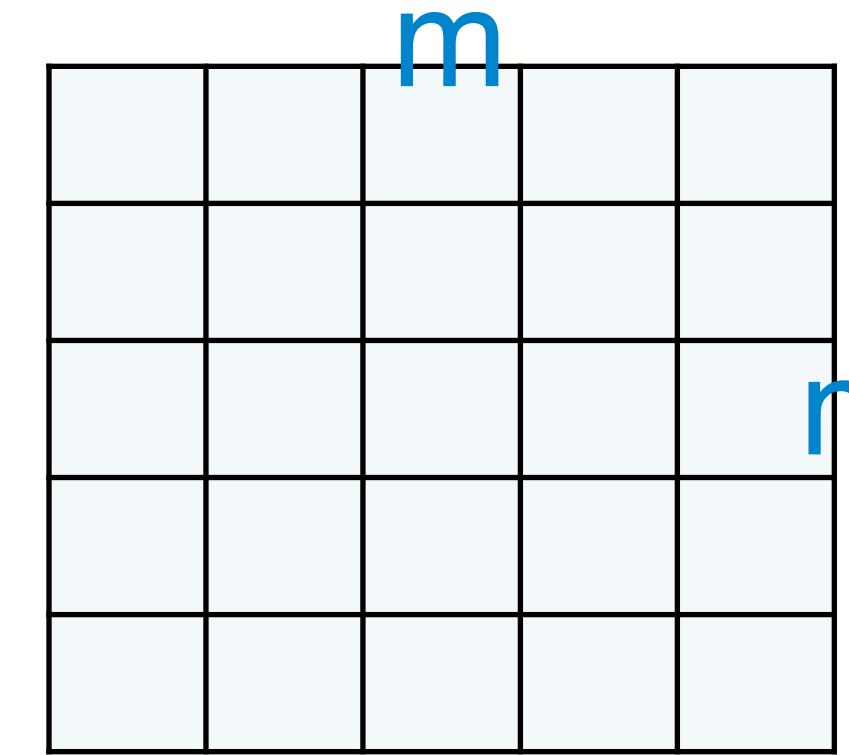
NumPy array operations



+ >
- >=
* <
/ <=
==
!=

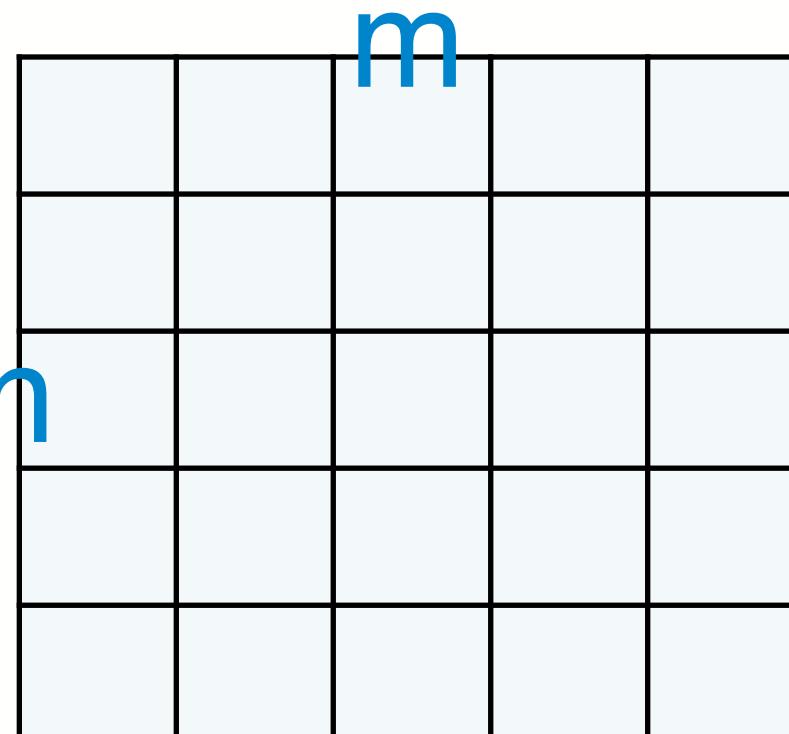
scalar

=



scalar

+ >
- >=
* <
/ <=
==
!=



=

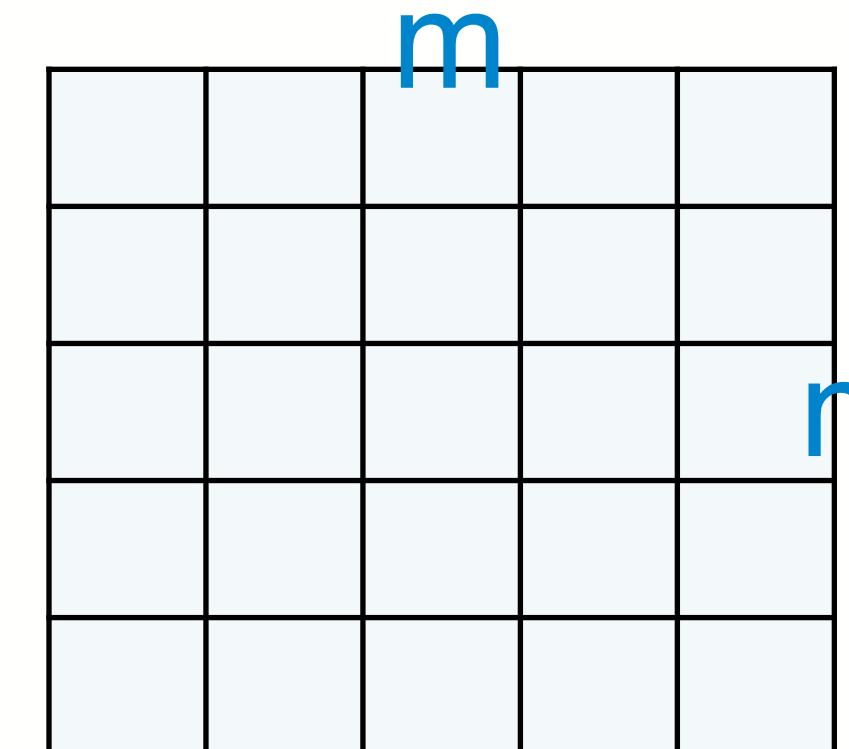
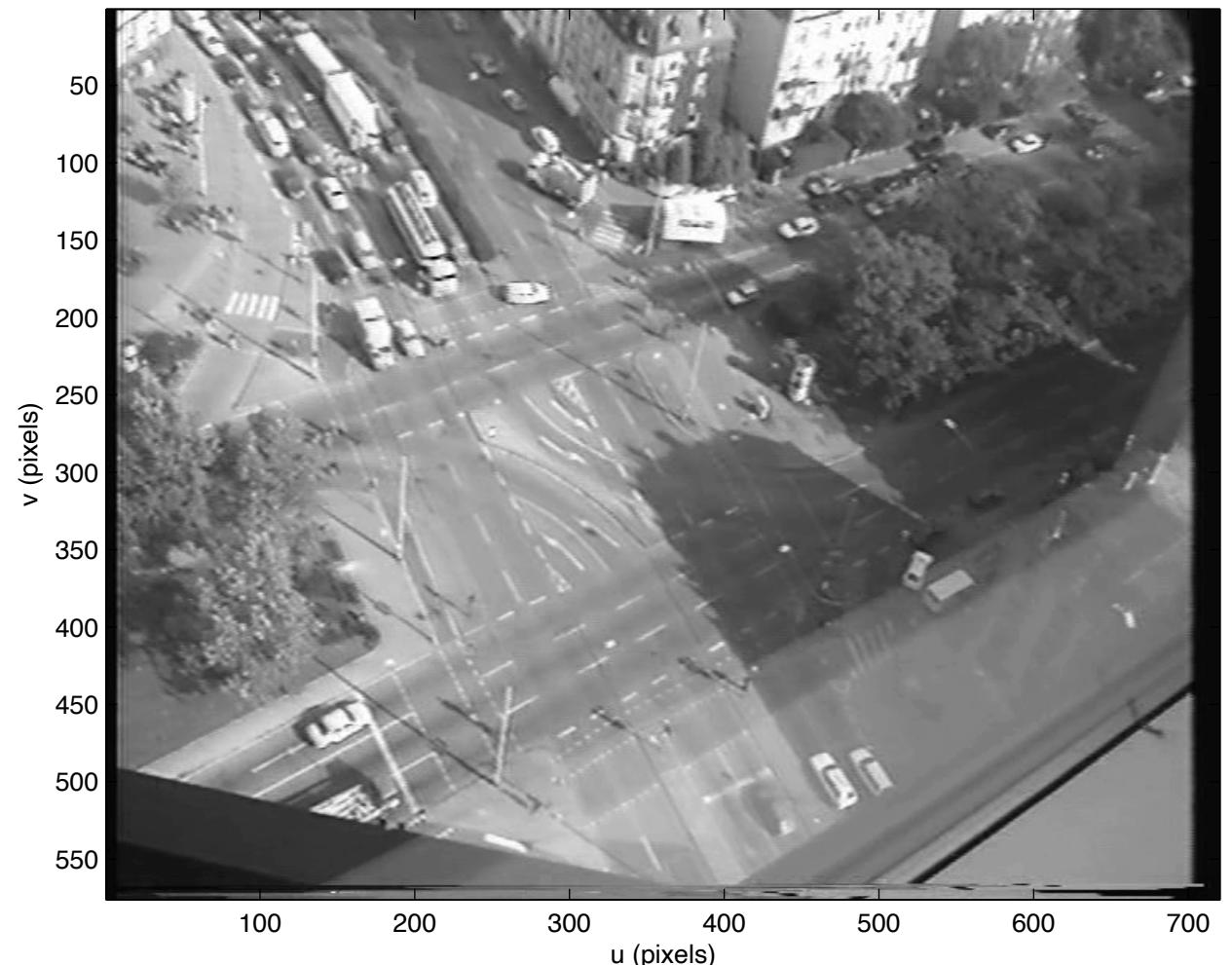
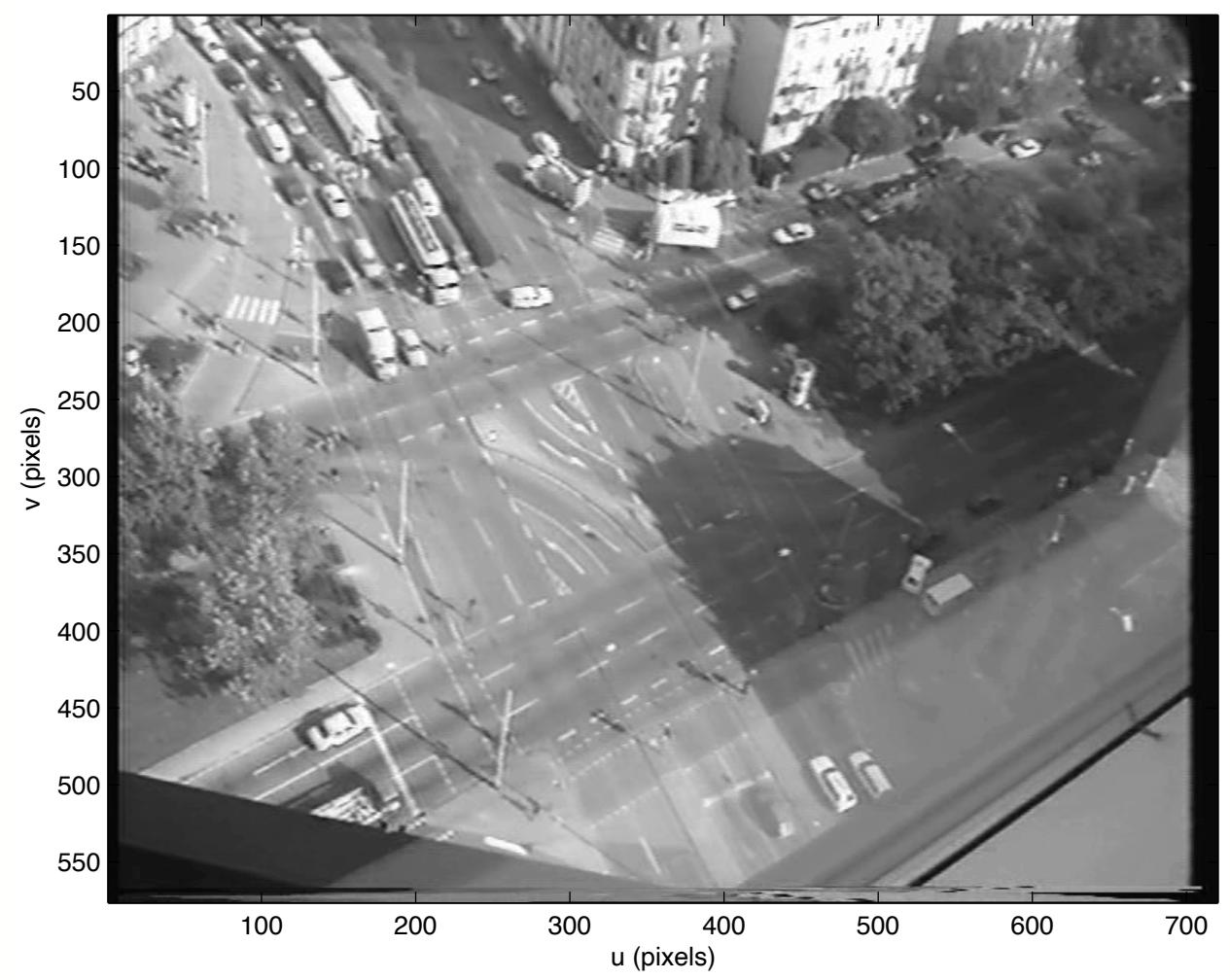


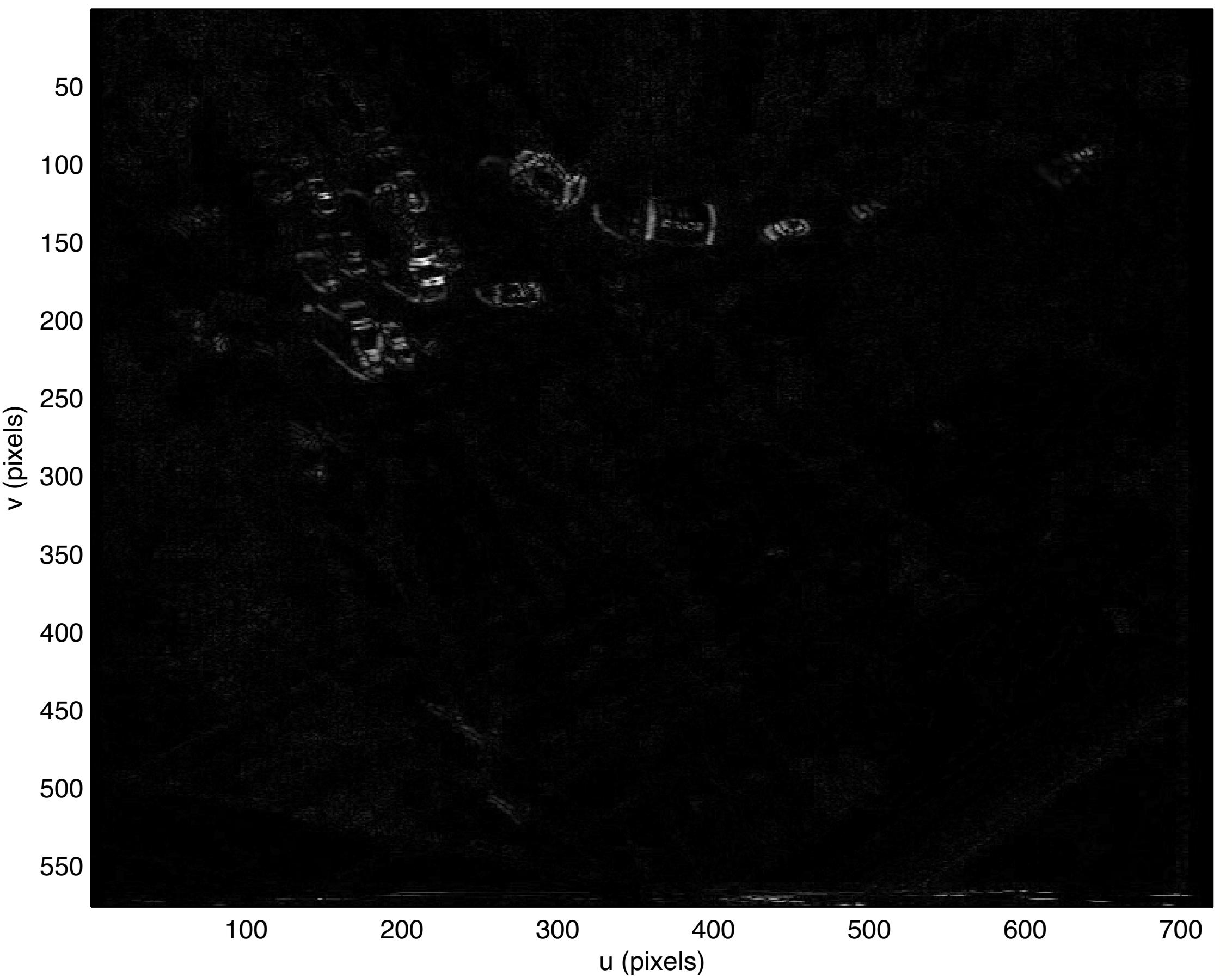
Image subtraction



frame 1



frame 4



- Pixel by pixel difference highlights image change
- Be careful when subtracting uint8 types



1 AC 60000 m
1 AC 2F
EV
LFEN + go

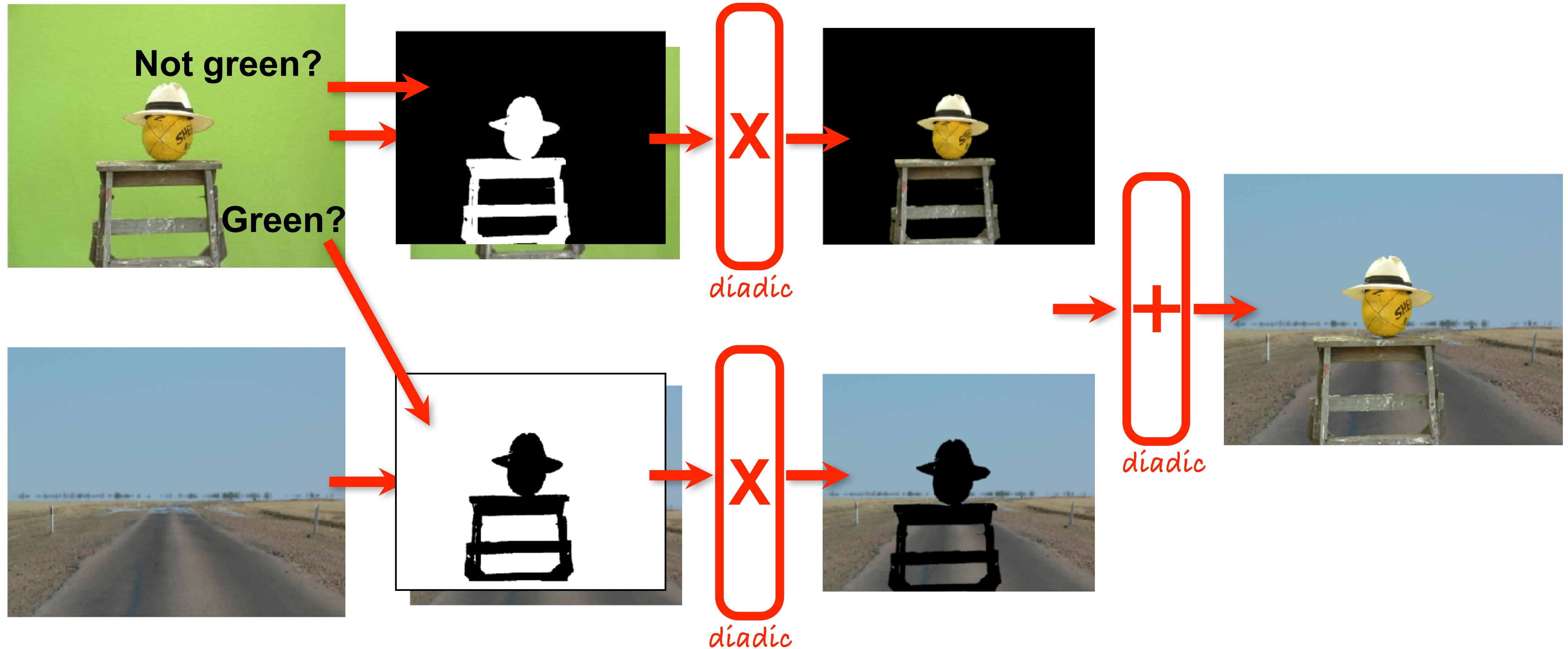
Green screen effect



- Merge two images to create a composite image
- Green pixels are substituted for background pixels



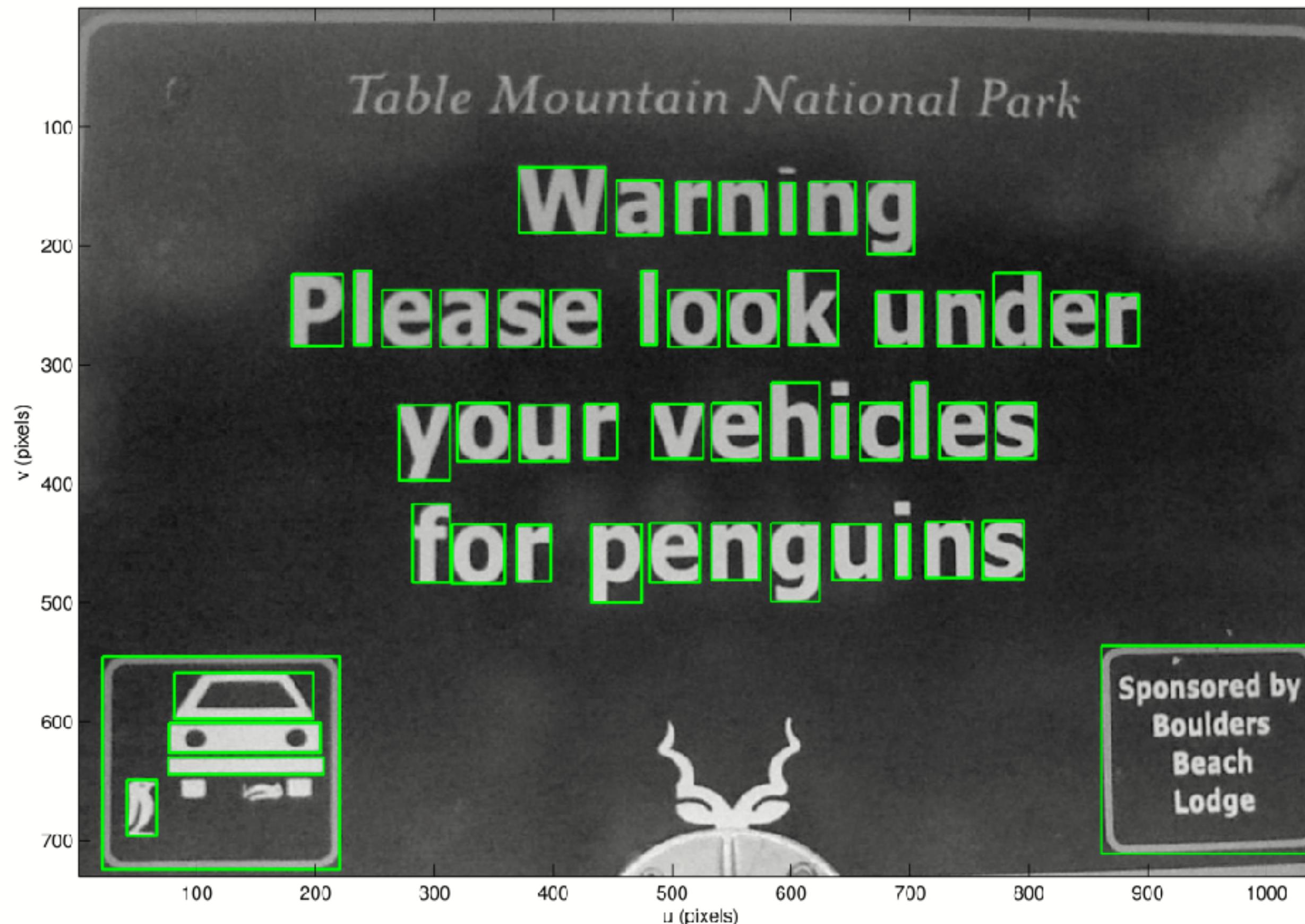
Green screen effect



Section 2

Binary blobs

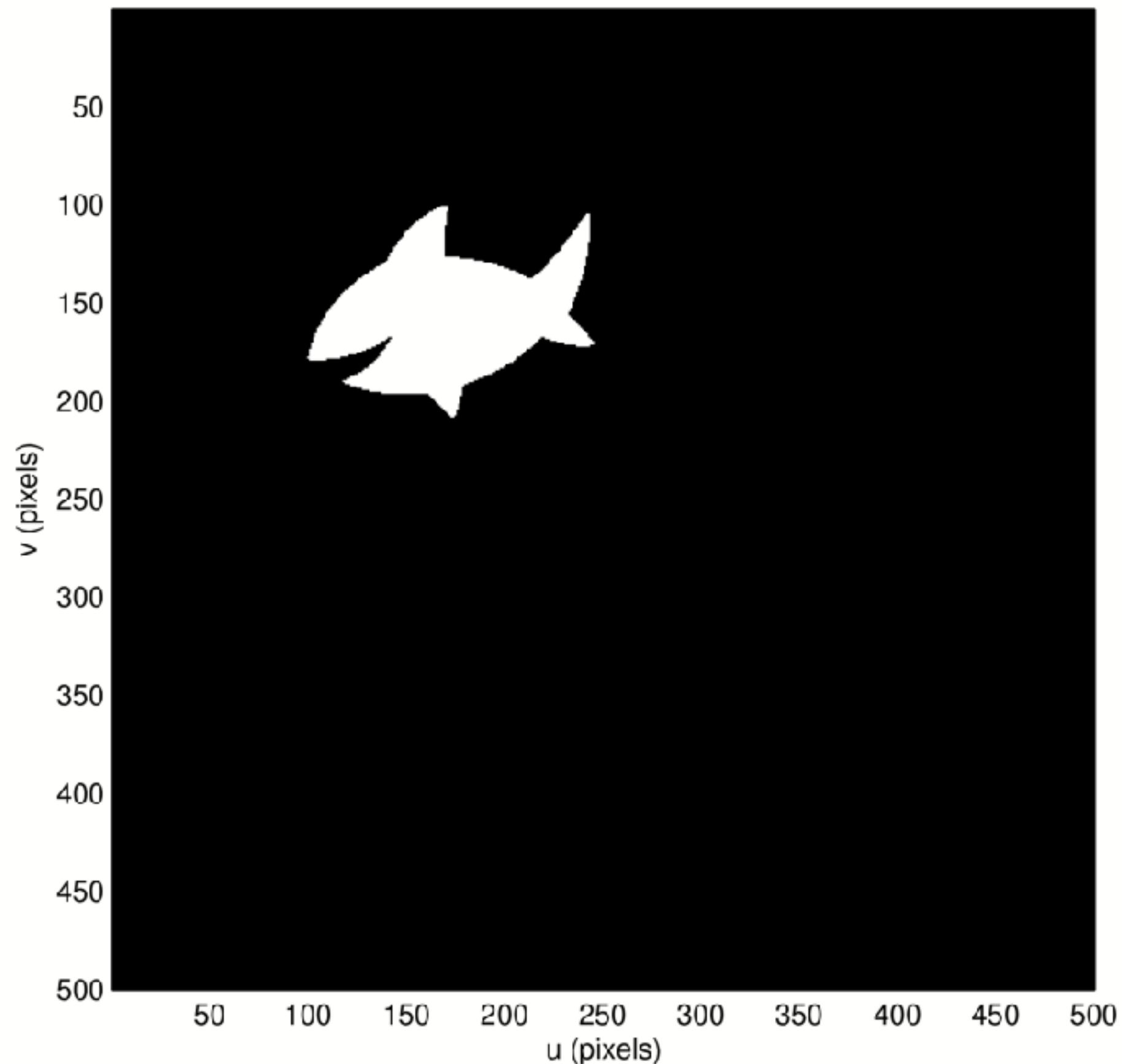
Region features



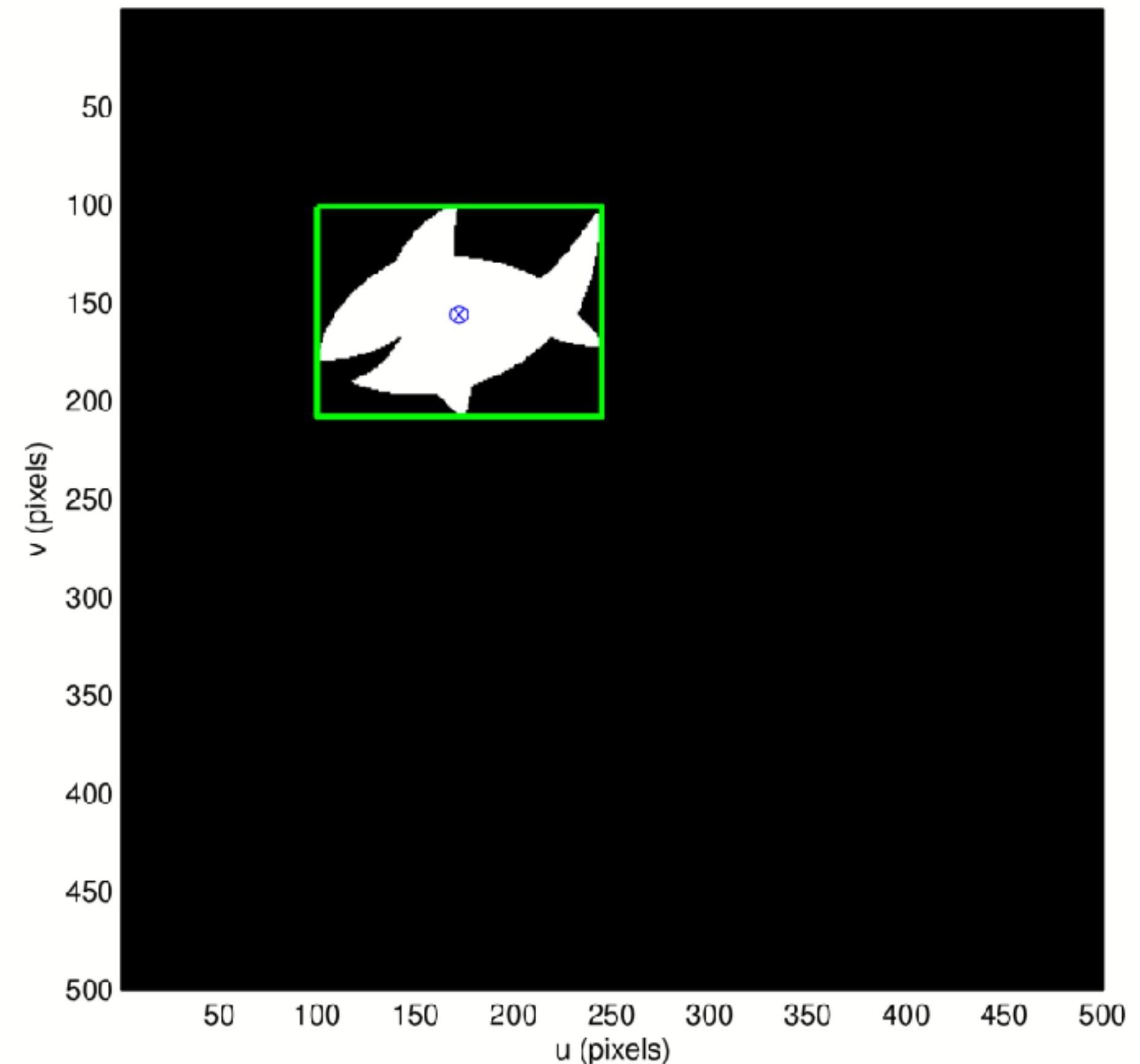
Where are the symbols on this sign?

Finding regions

<http://sweetclipart.com>



a bunch of pixels



a region

Moments

The p - q^{th} moment of an image is given by

- where $(p+q)$ is the **order** of the moment

$$m_{pq} = \sum_{(u,v) \in I} u^p v^q I[u, v]$$

The zeroth ($p=q=0$) moment of an image is simply

- which is the number of one pixels in the region
- the **area** of the region

$$m_{00} = \sum_{(u,v) \in I} I[u, v]$$

First moments

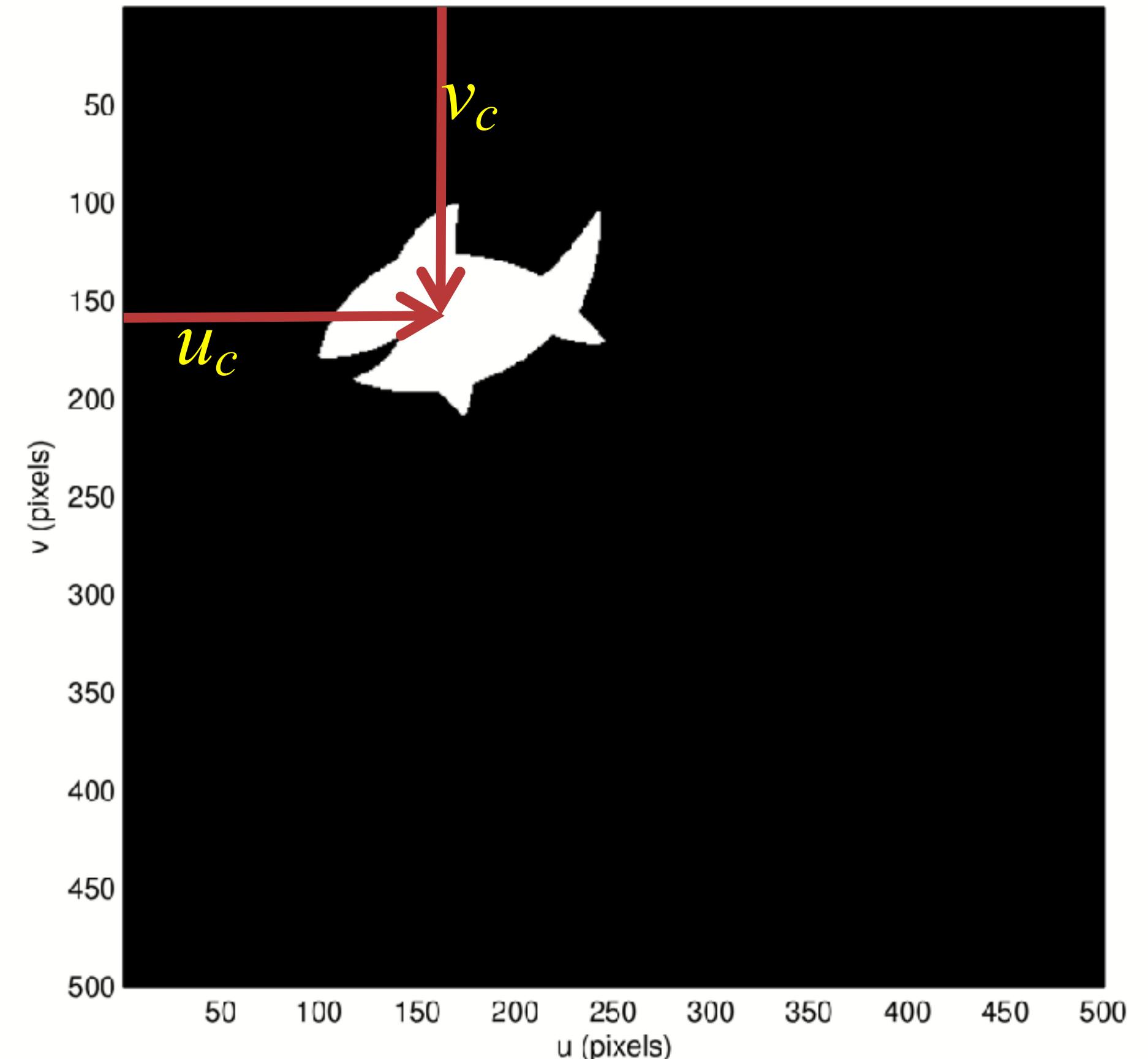
$$m_{10} = \sum_{(u,v) \in I} u I[u, v]$$

$$m_{01} = \sum_{(u,v) \in I} v I[u, v]$$

The **centroid**, or **centre** of mass, is given by

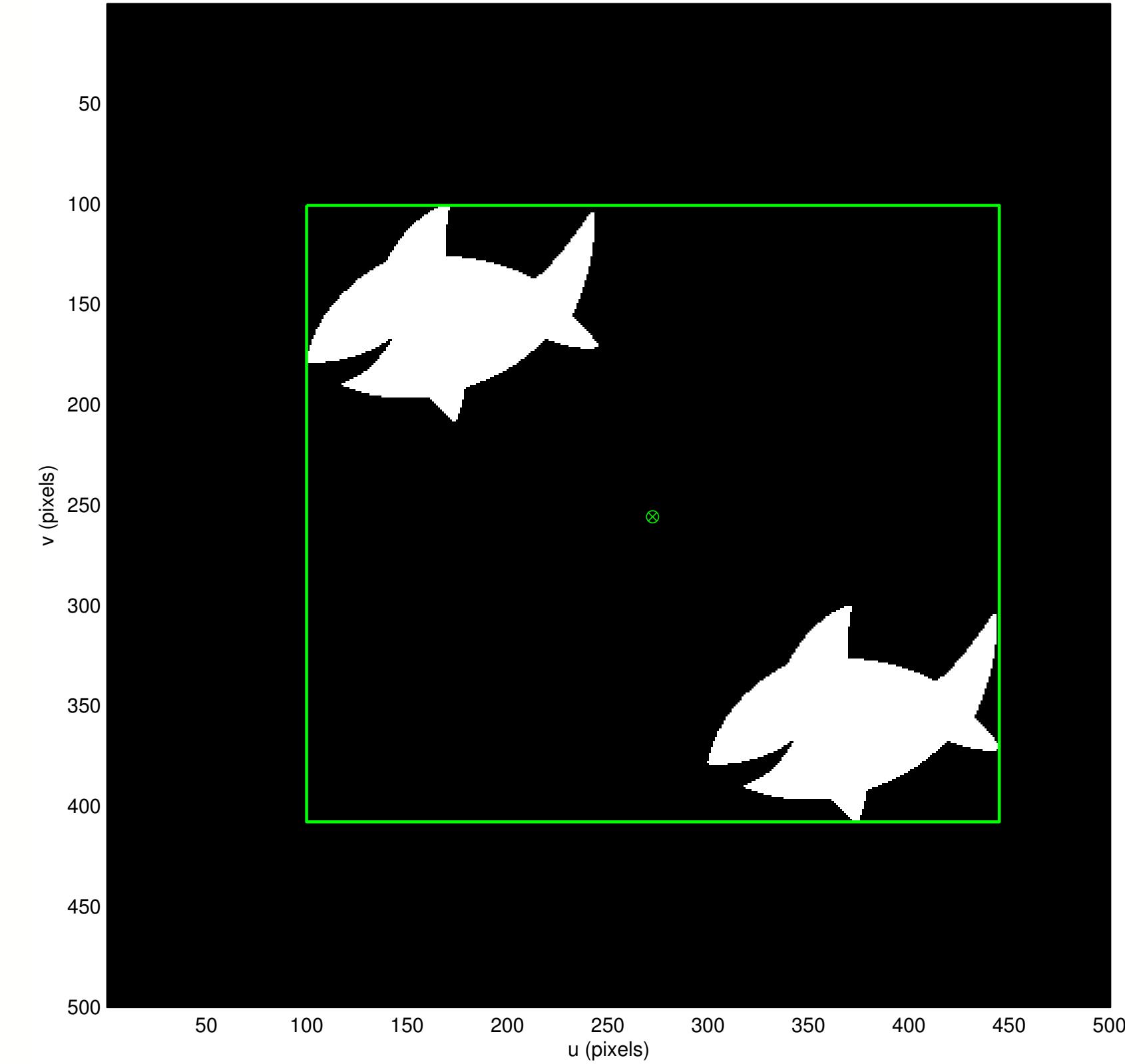
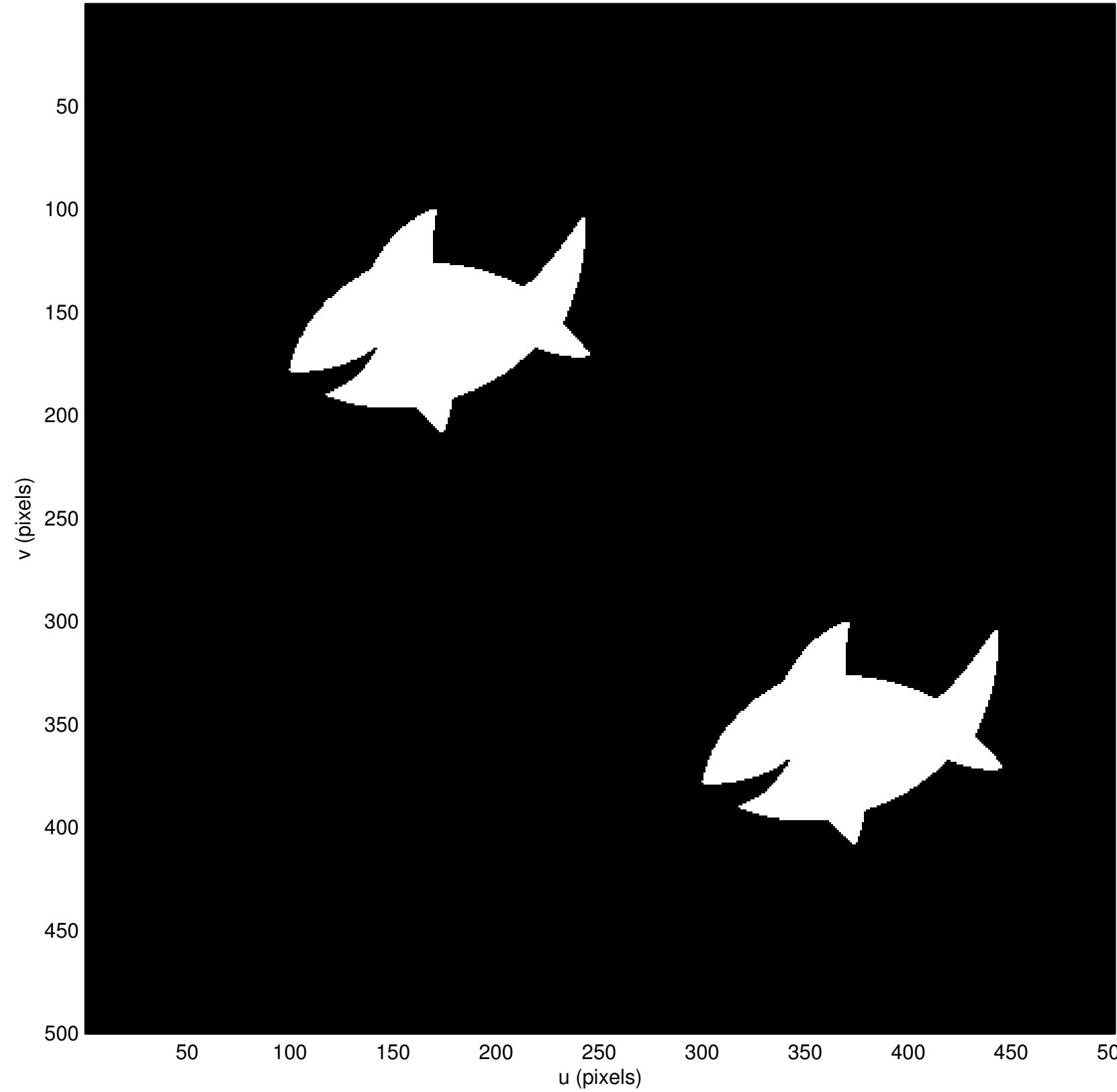
$$u_c = \frac{m_{10}}{m_{00}}, \quad v_c = \frac{m_{01}}{m_{00}}$$

<http://sweetclipart.com>



Multiple blobs

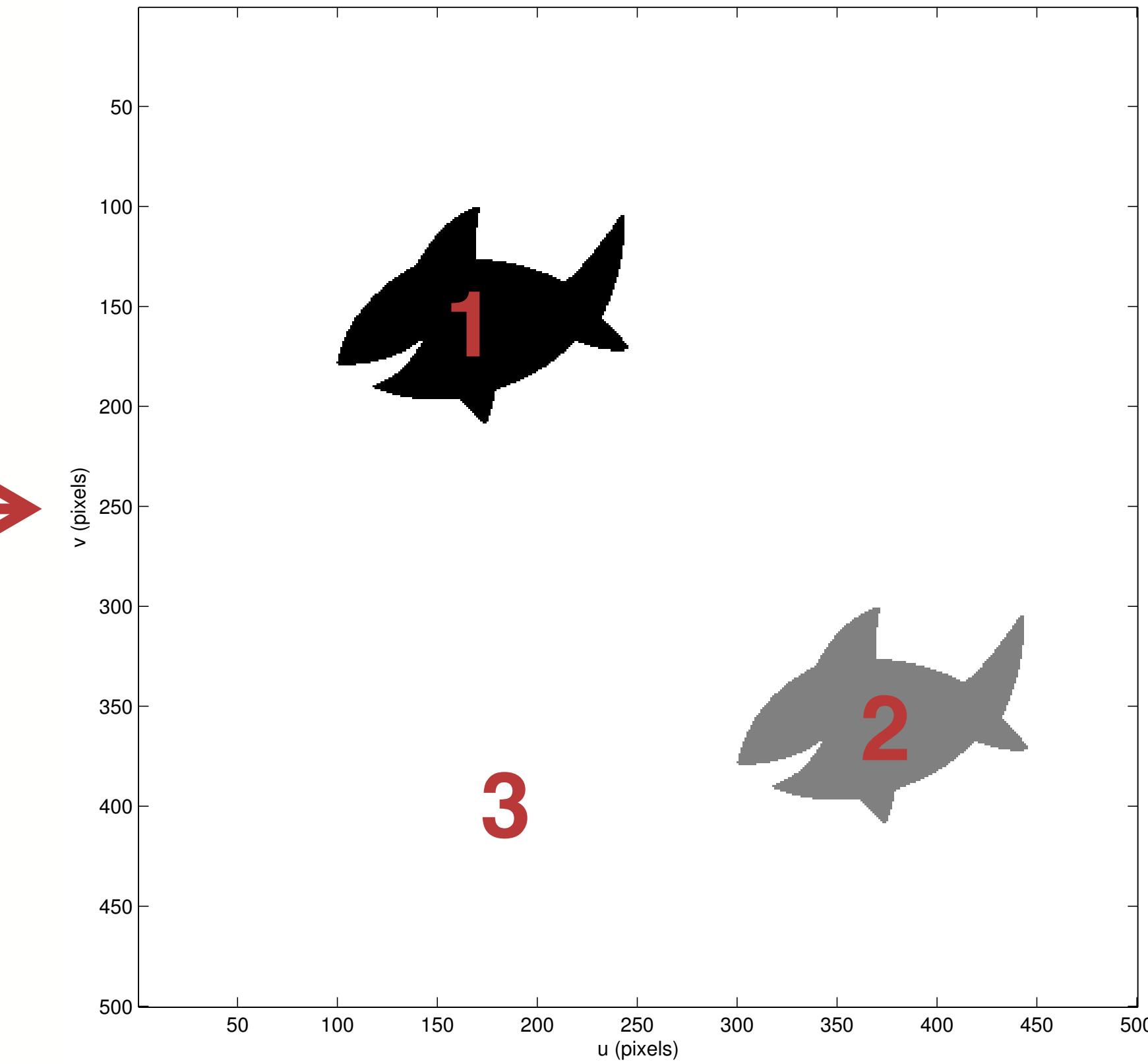
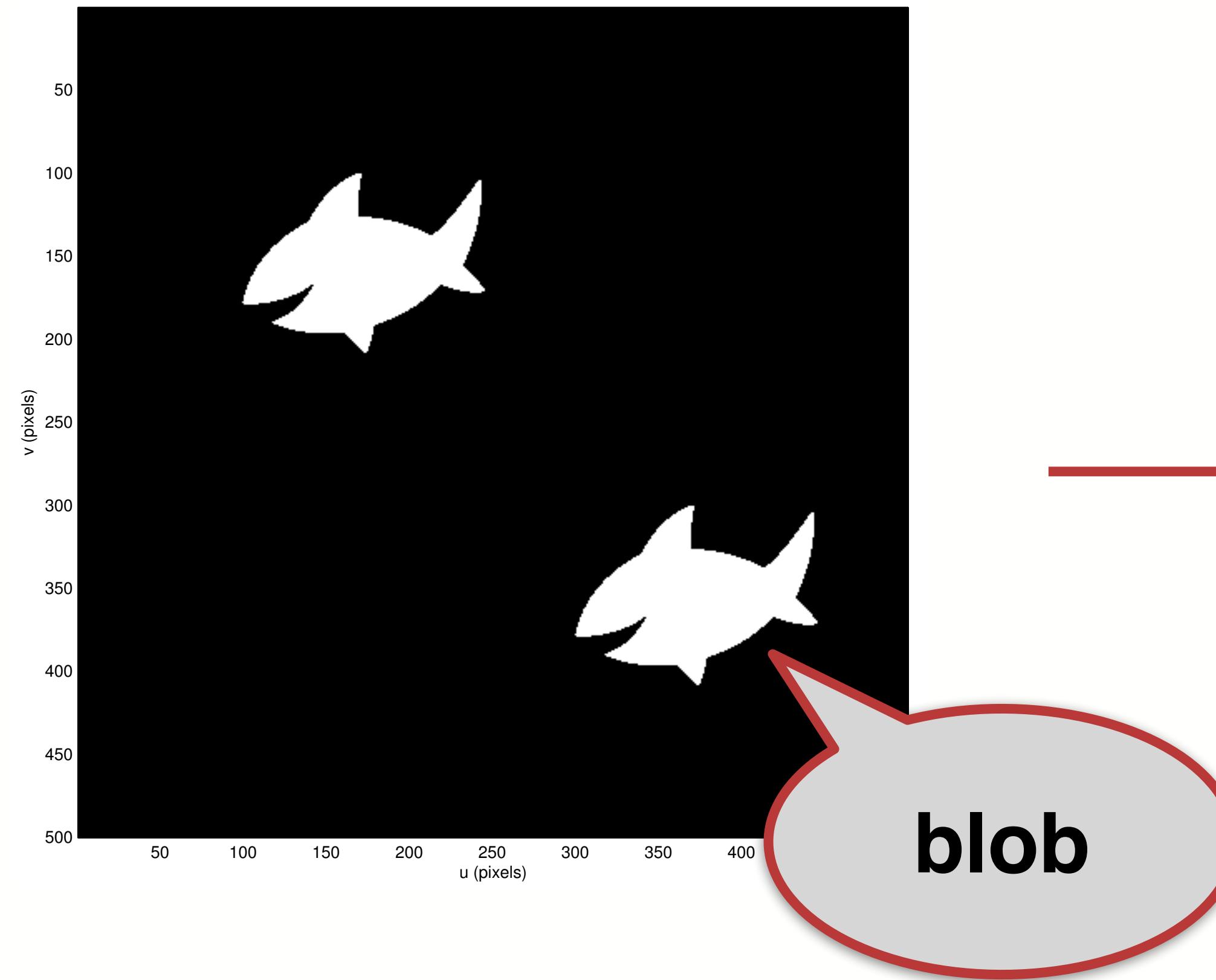
<http://sweetclipart.com>



- We can see two distinct regions
- but the moment computation has no sense of this

Transform the image

<http://sweetclipart.com>



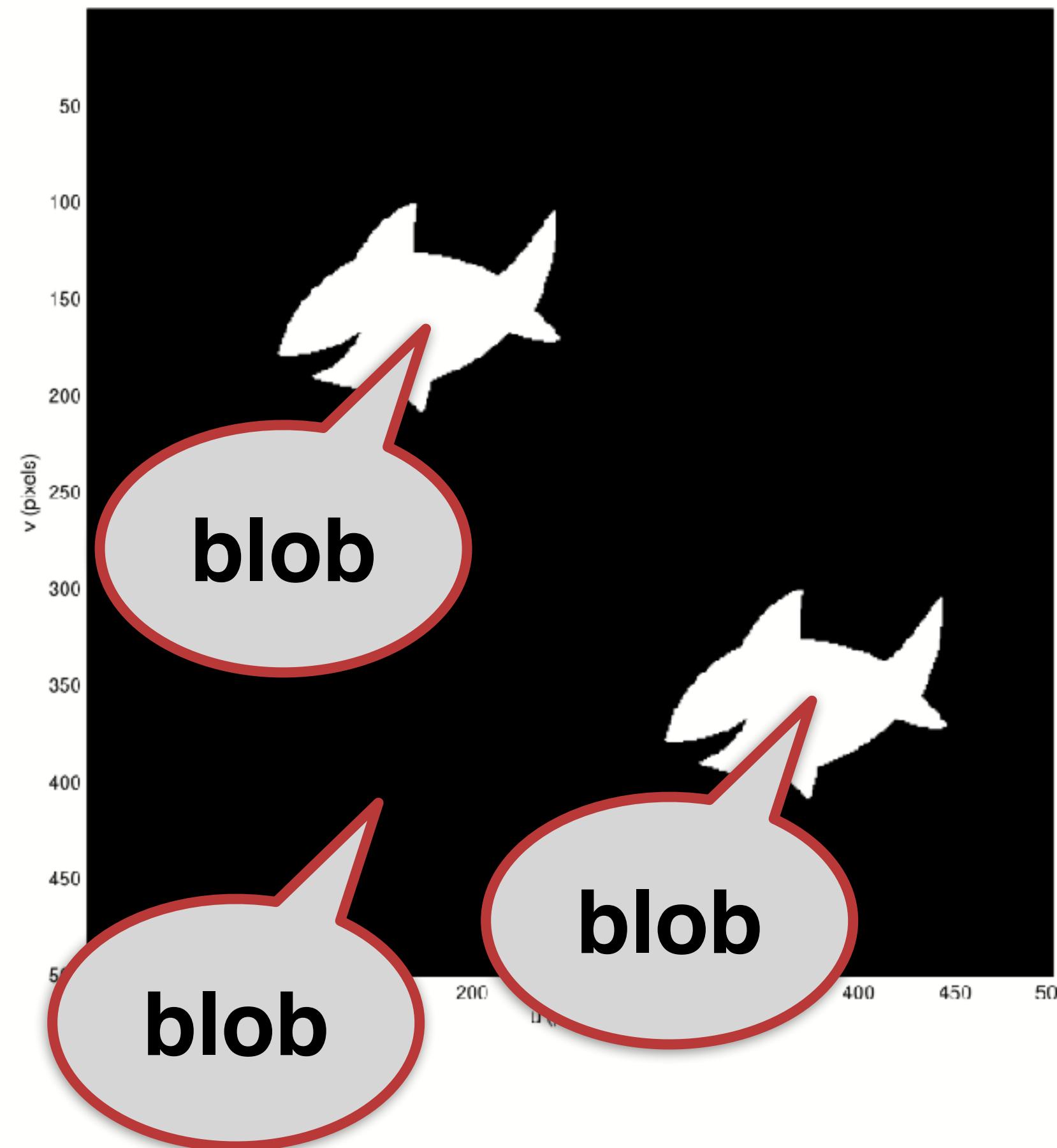
- We apply **connectivity analysis**
- Each pixel in the output image (value 1 to 3) indicates the **blob number** of the corresponding input pixel

Label image

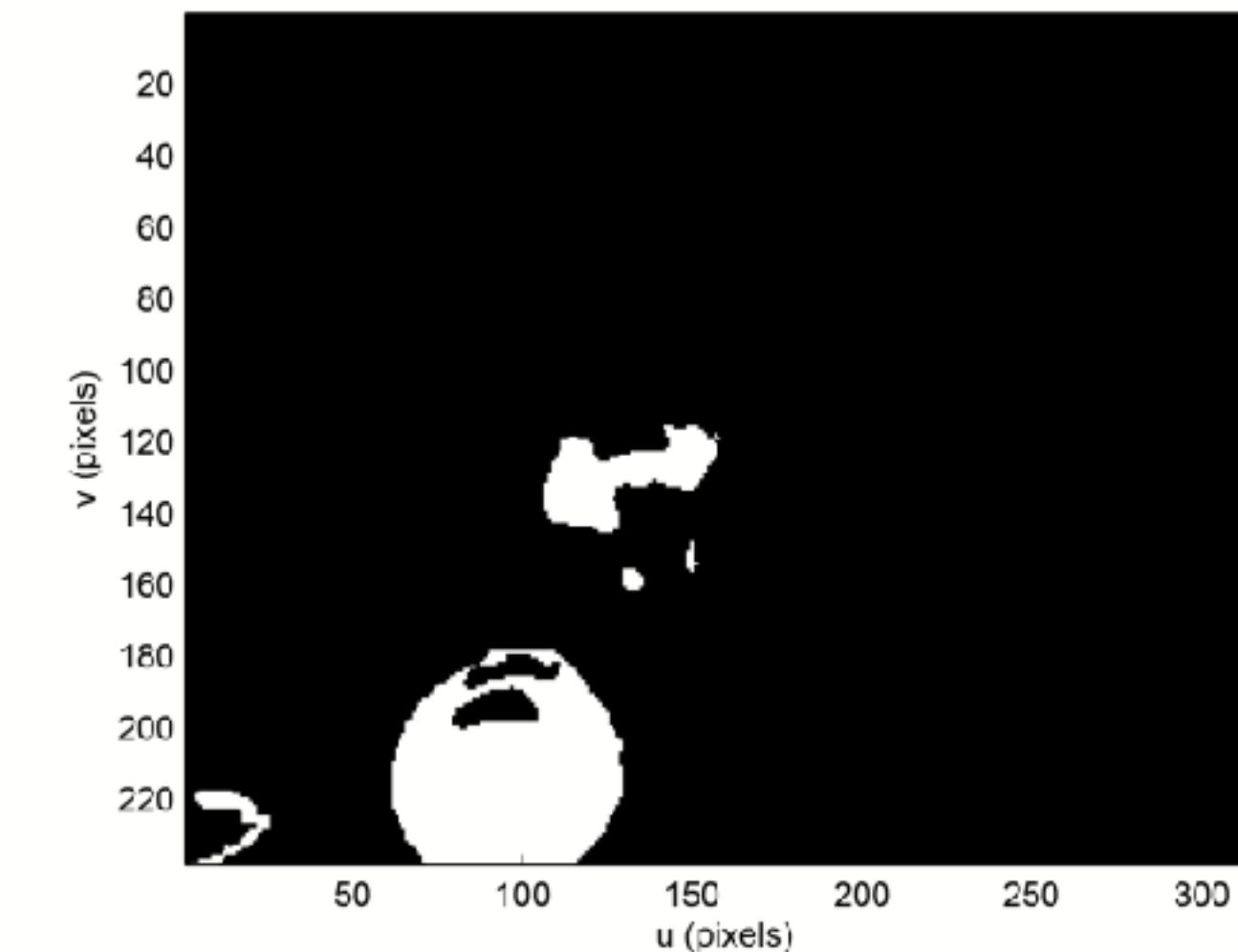
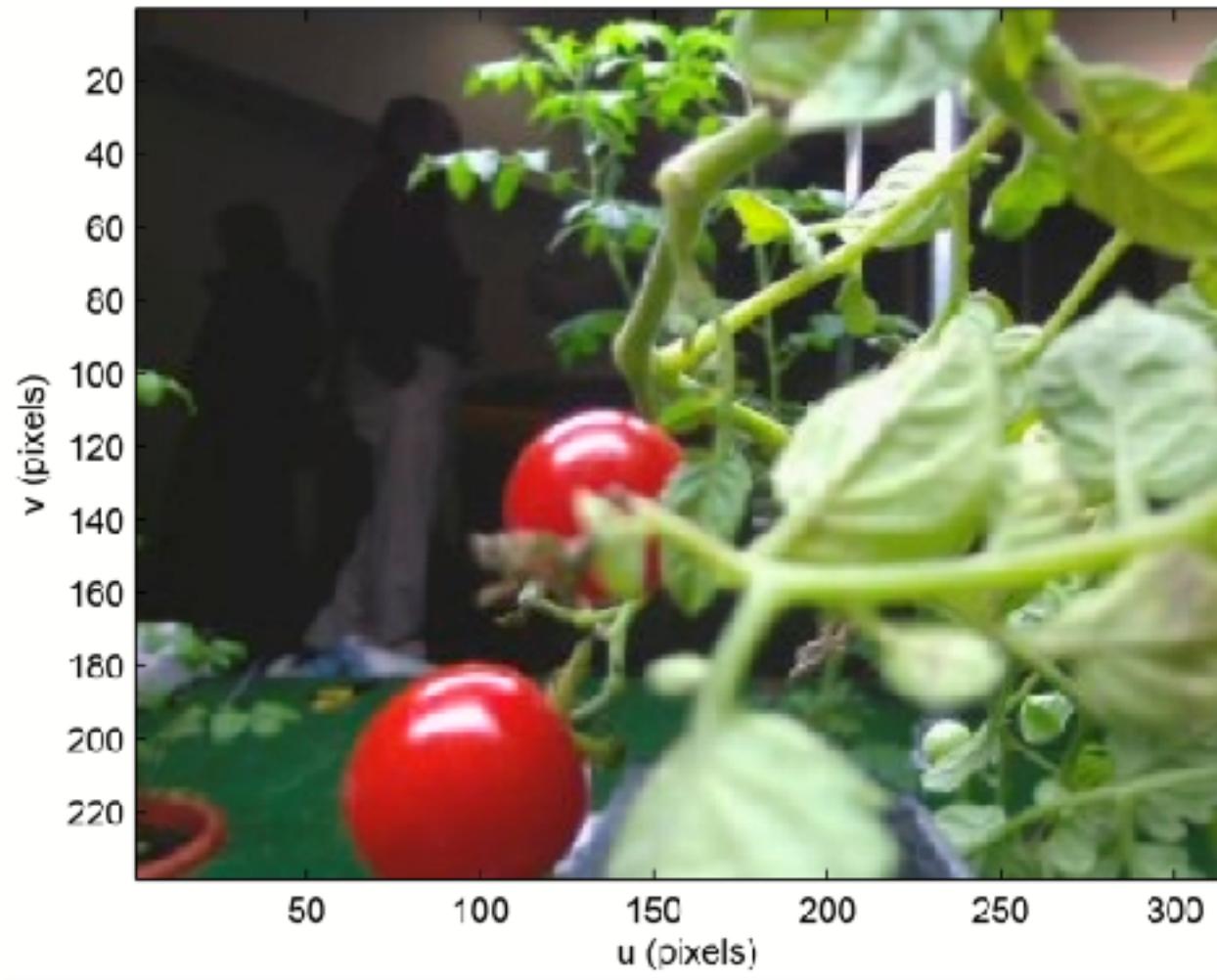
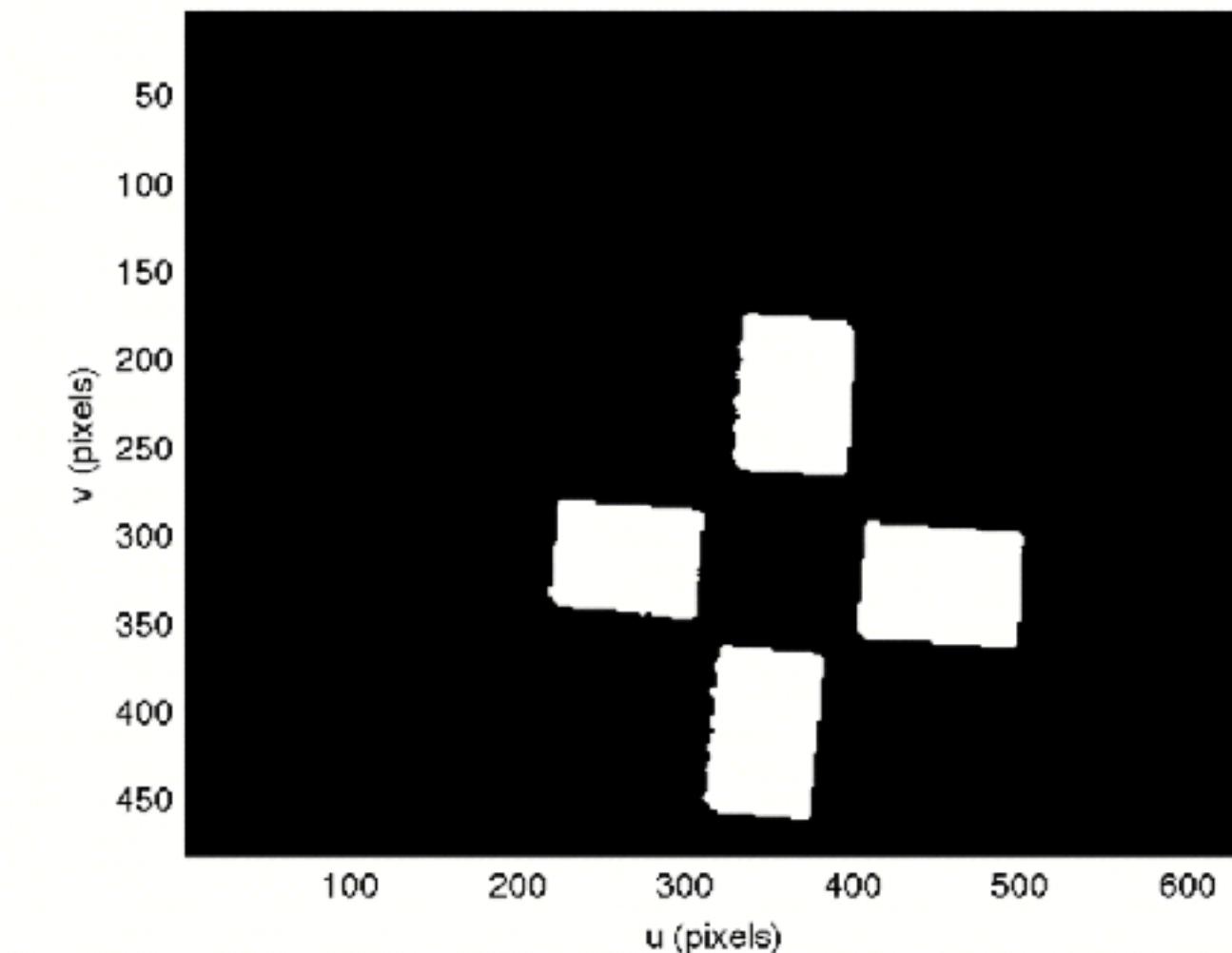
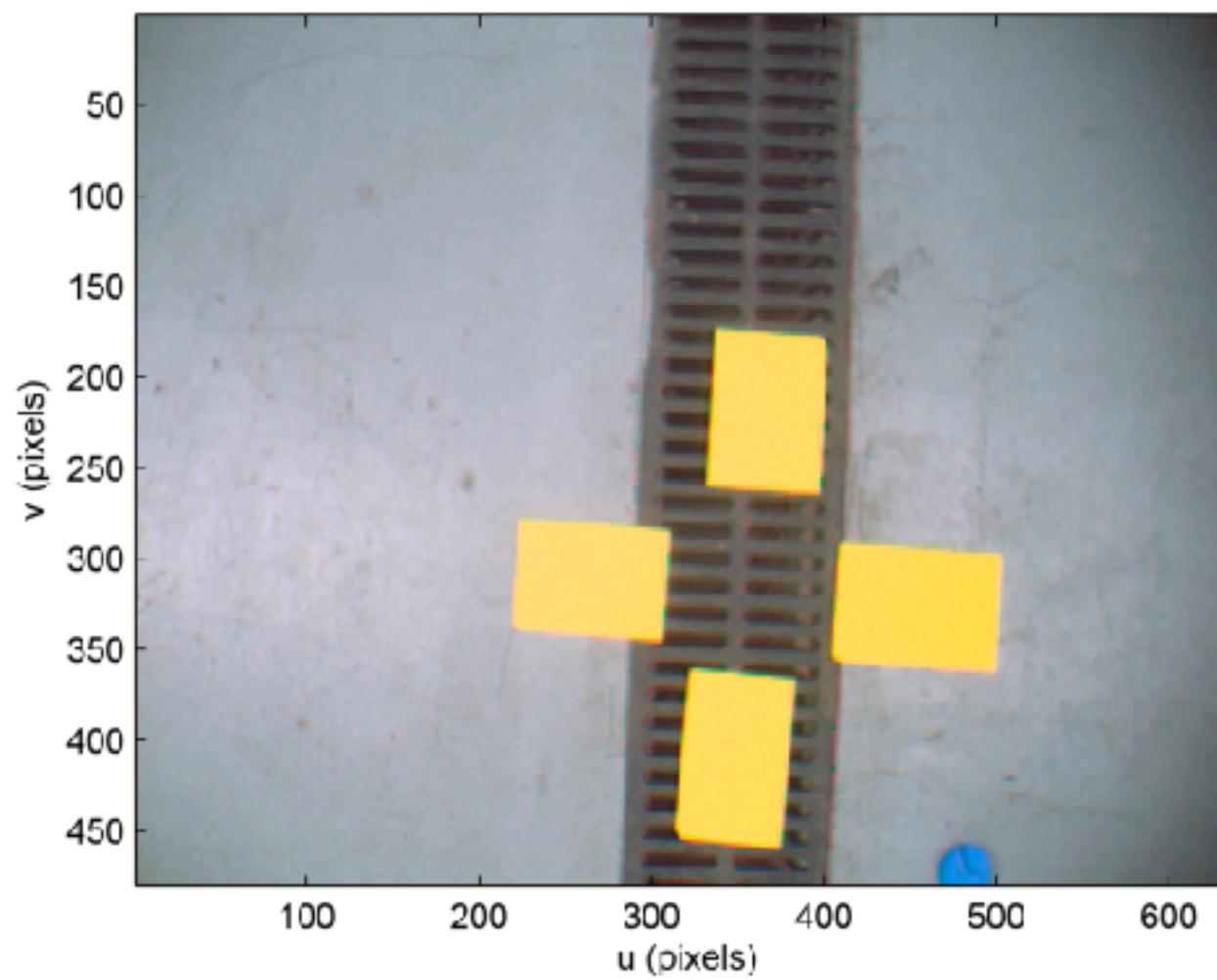
What is a blob?

- aka region, connected component
- a set of **contiguous** pixels of the same color (value)

<http://sweetclipart.com>

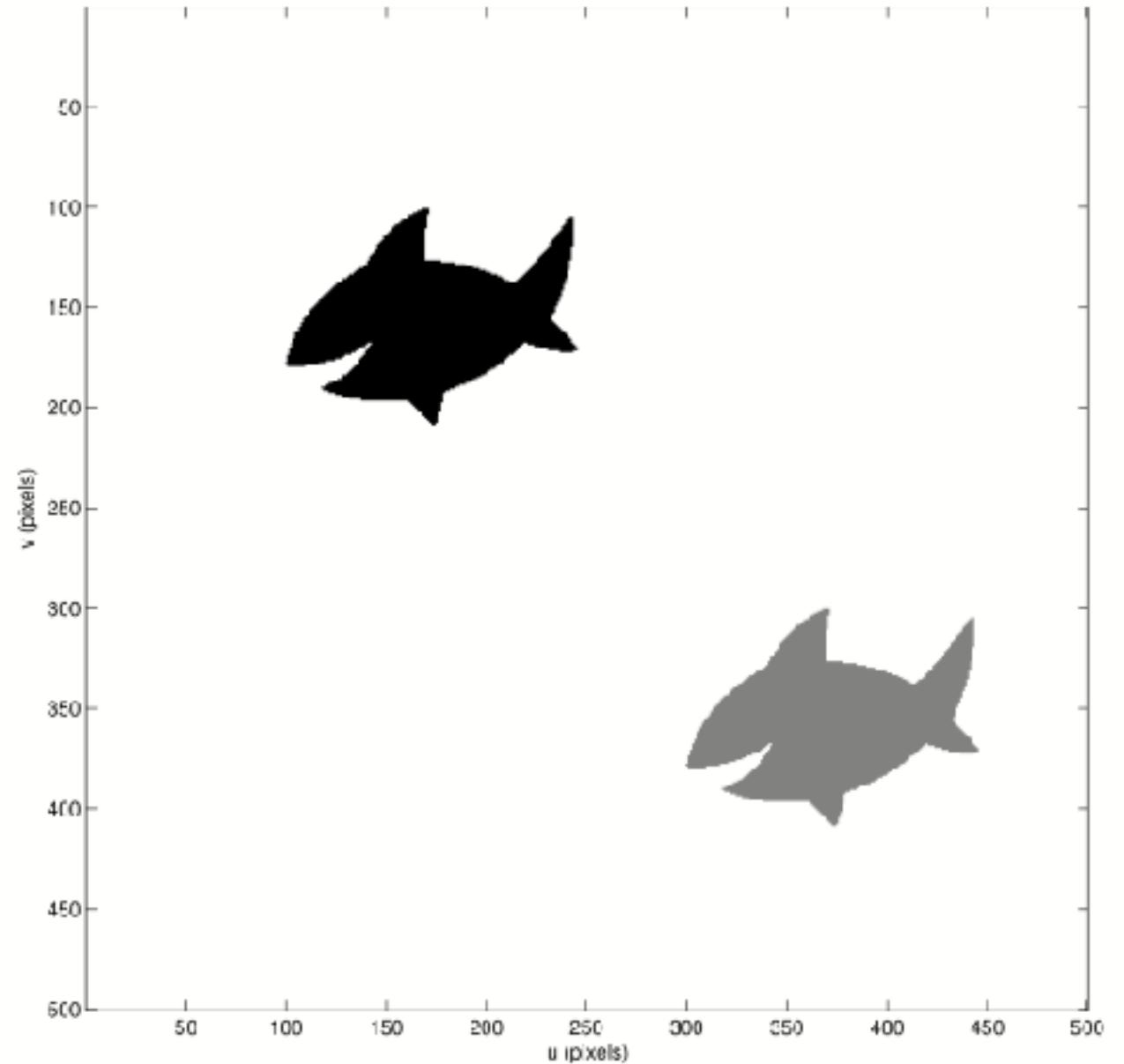
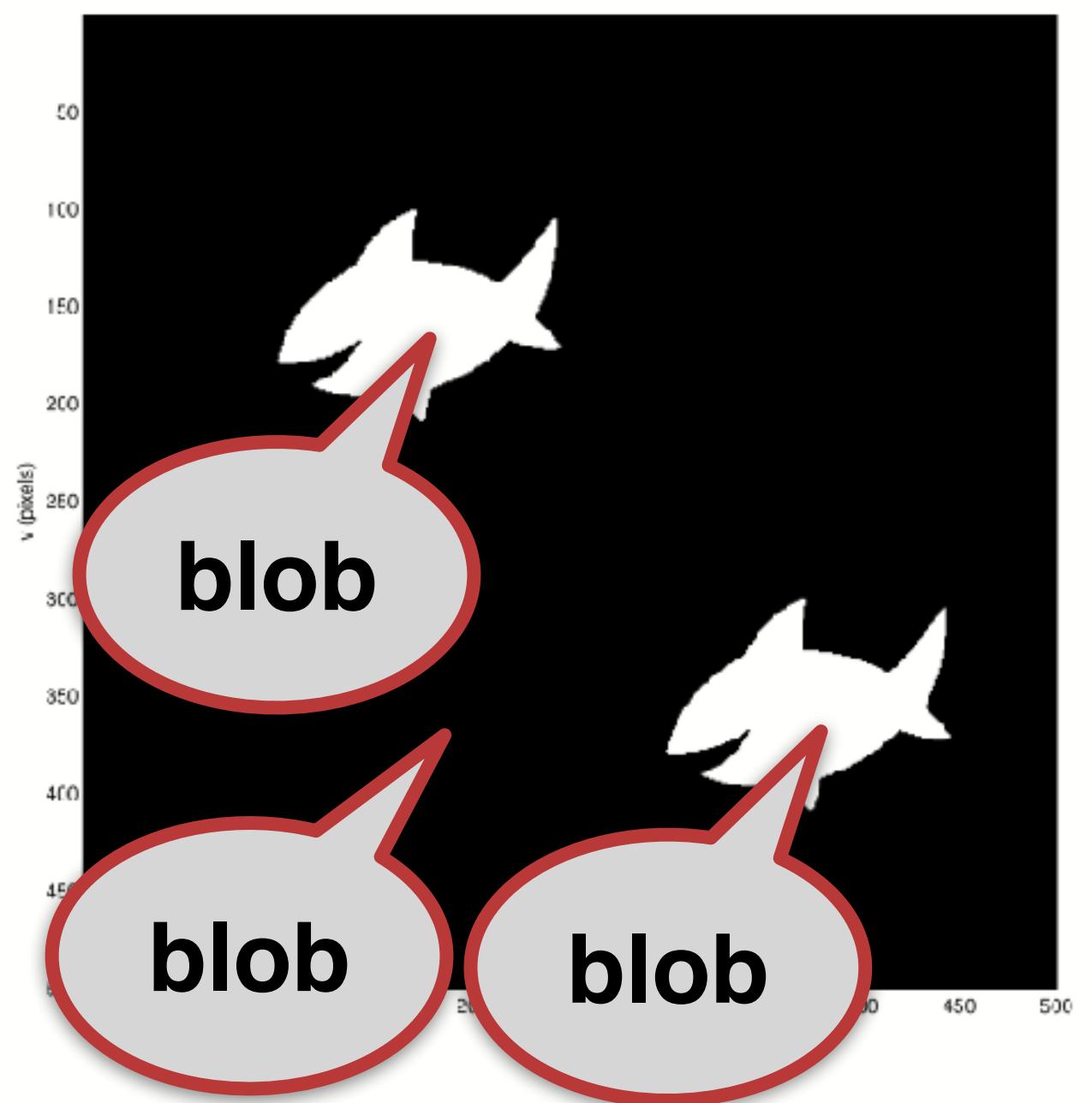


Pixel classification



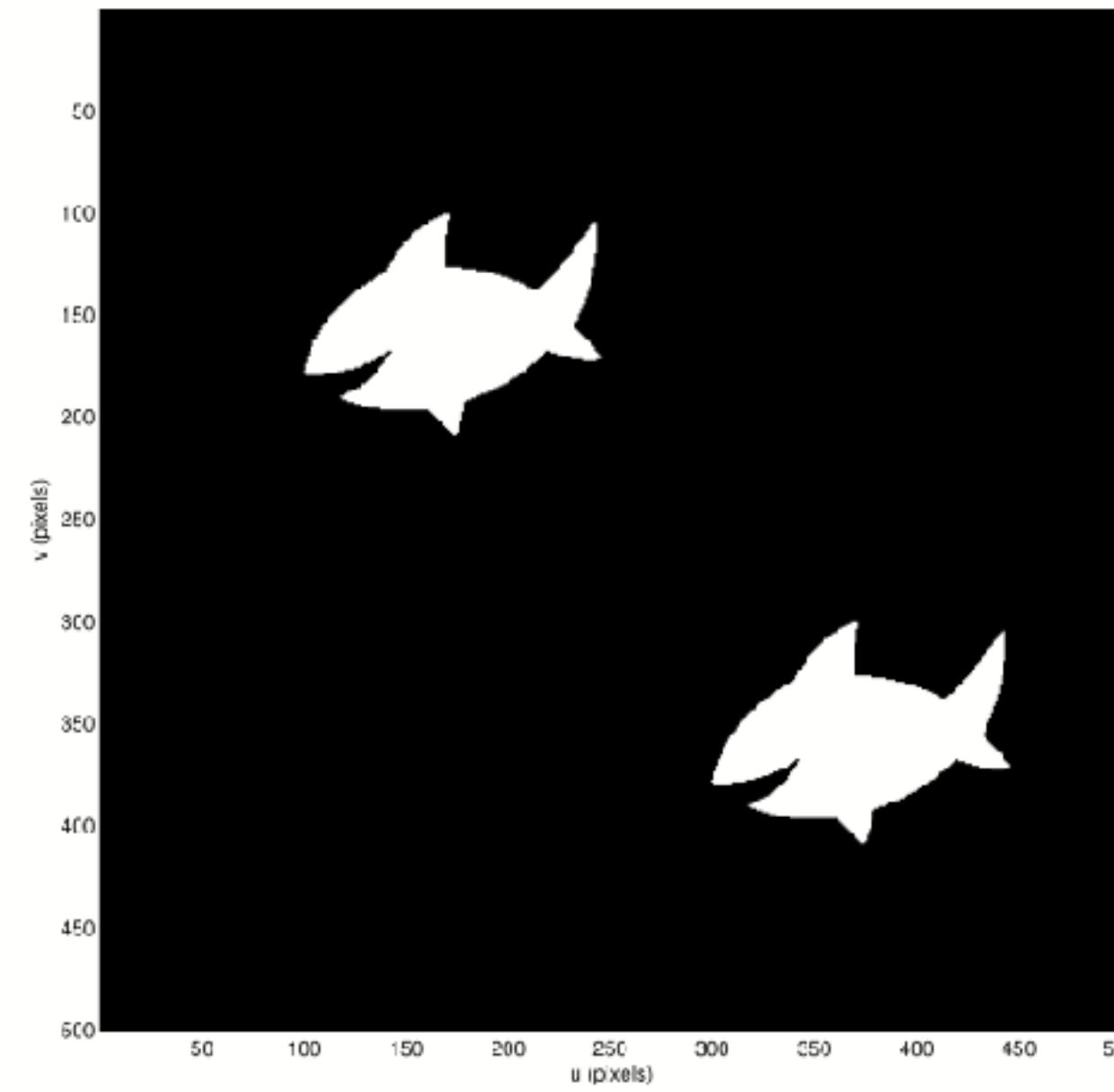
What is a blob?

- aka region, connected component
- a set of **contiguous** pixels of the **same color** (value)
- the label assigned to a pixel indicates which set (1...N) it belongs to
- every pixel has the same label as its N, S, E or W neighbour of the same color
- the process has many names:
 - connectivity/blob analysis,
 - connected component analysis,
 - blob/region/image labelling,
 - blob/region coloring

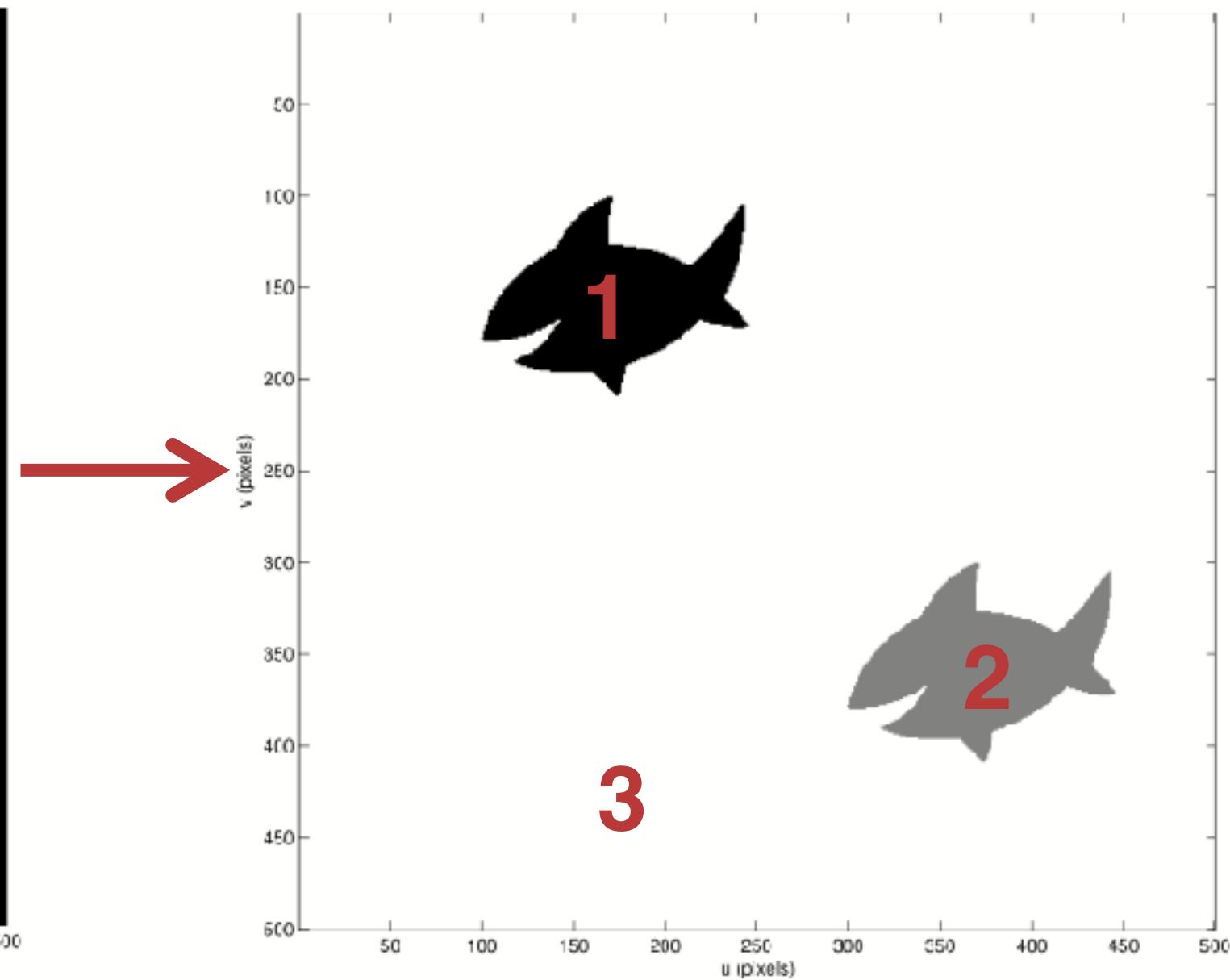


Select individual blobs

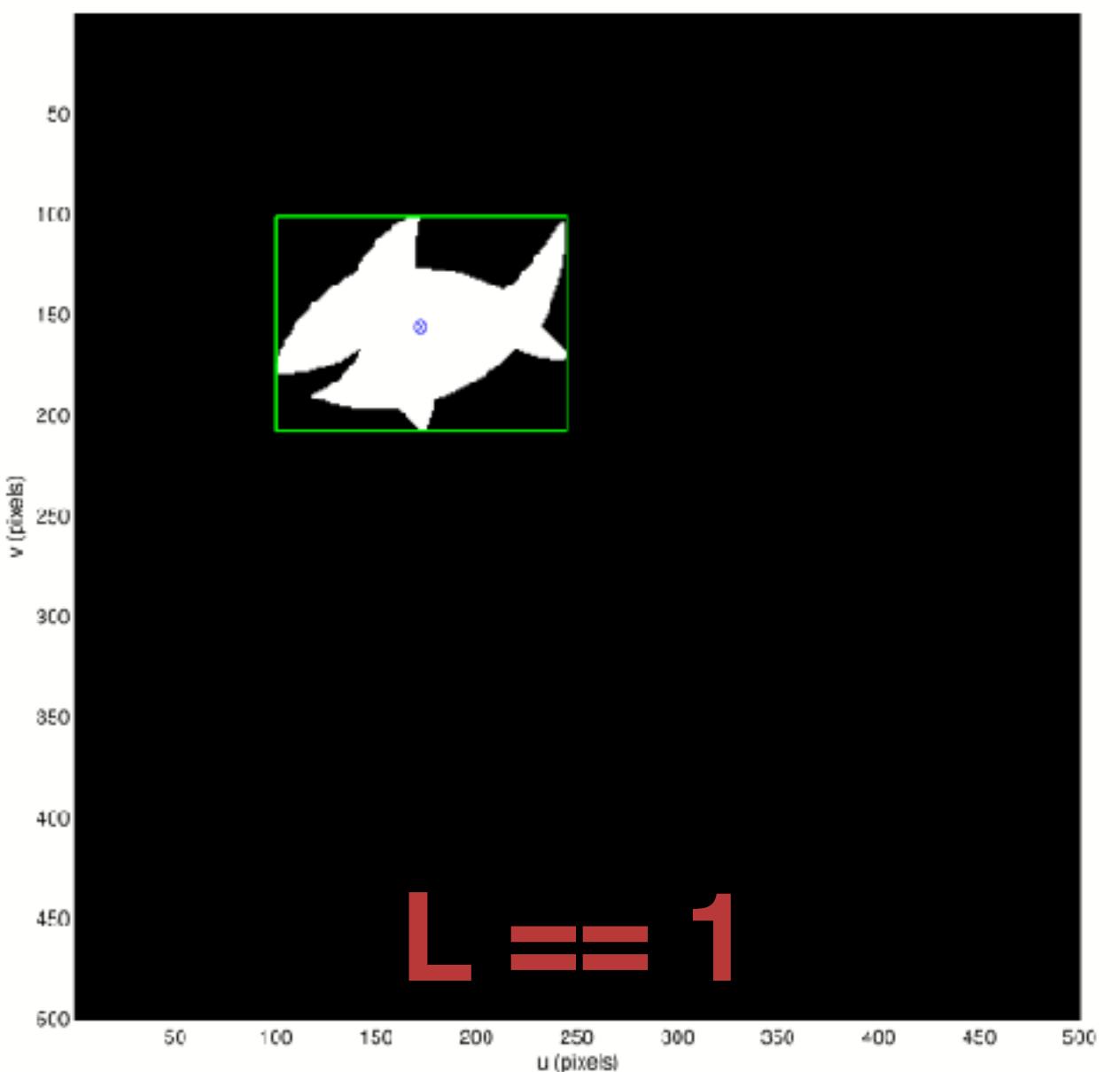
<http://sweetclipart.com>



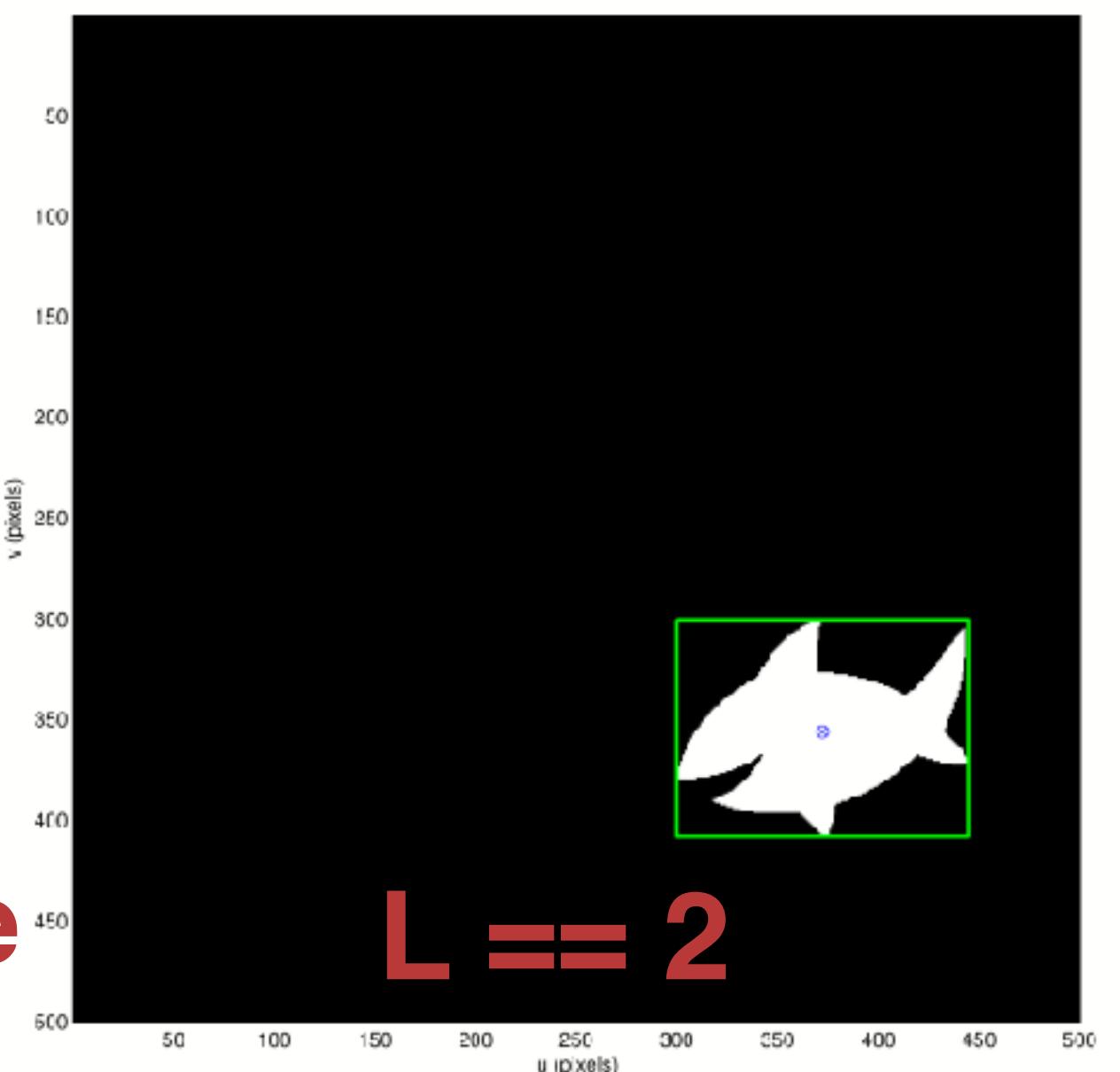
Original binary
image



$L = \text{label image}$



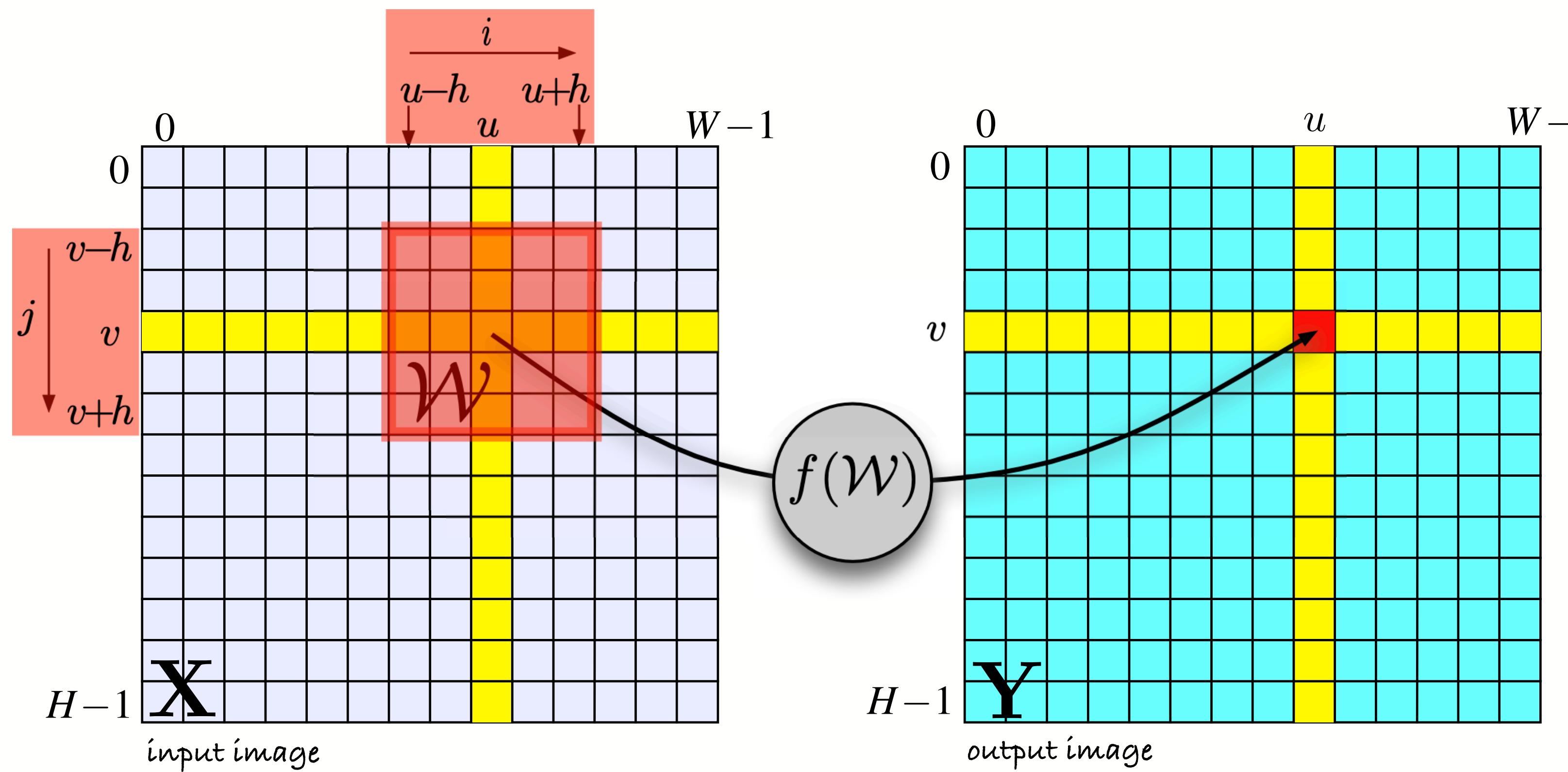
$L == 1$



$L == 2$

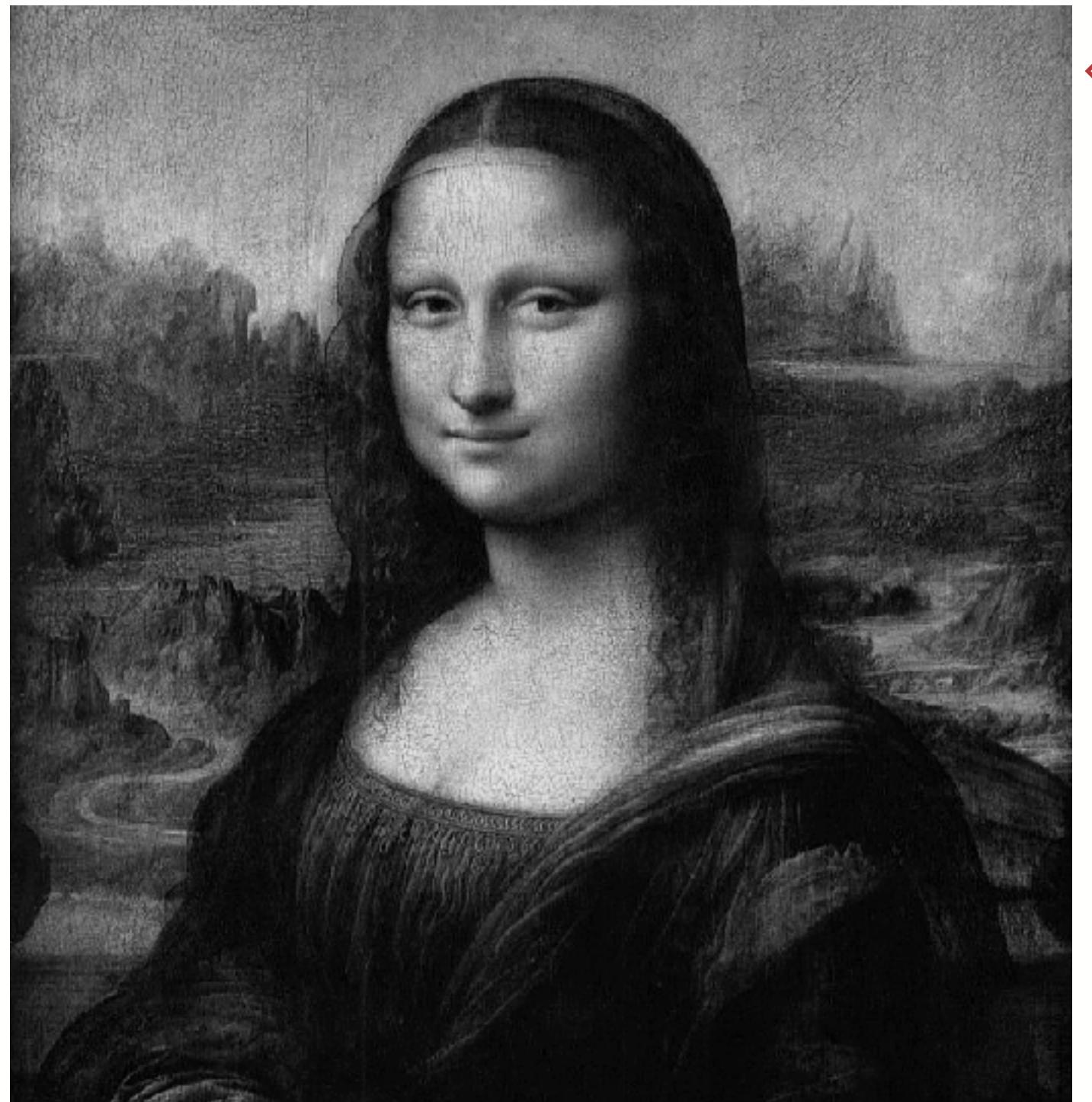
Individual blob binary image

Spatial operators



- The function can capture something about the **uniformity** or **variation** over the local pixel **window \mathcal{W}**

$f(\cdot)$ could be an average



↙Original image

Average each
pixel over a
7x7 window⇒



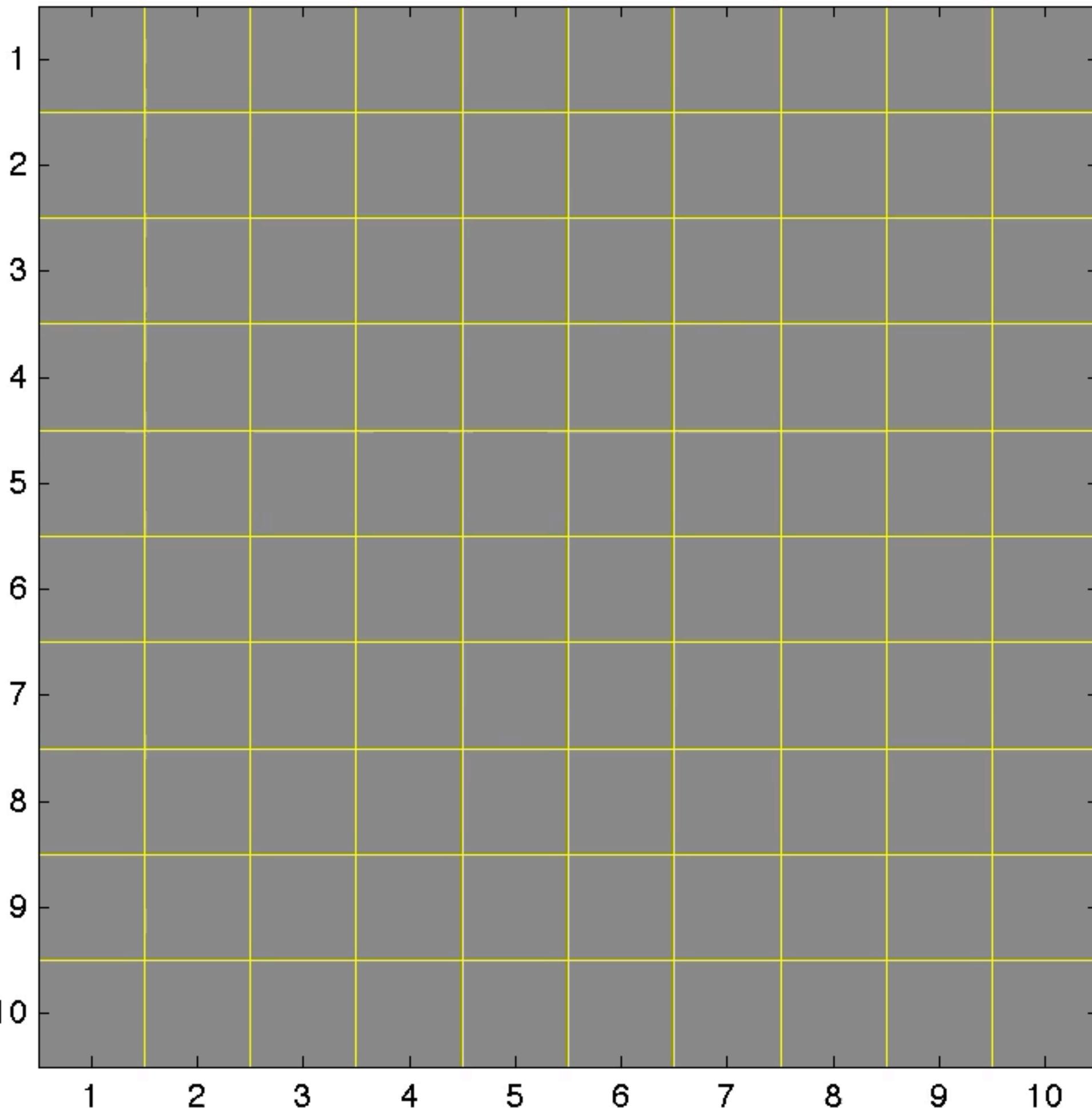
- The average over the window
 - can reduce noise
 - reduces the resolution

Moving window process

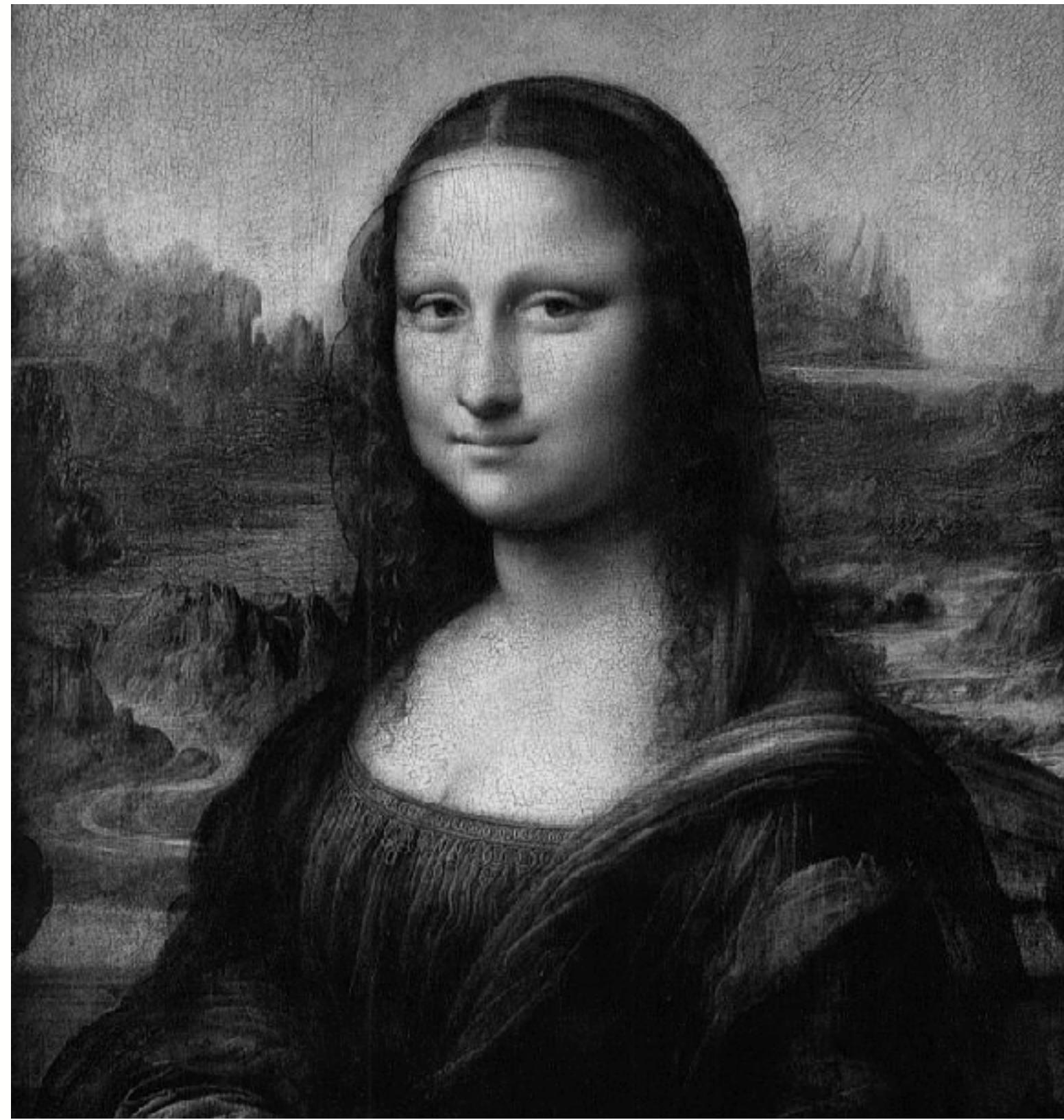
Input image

1	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30
2	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30
3	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30
4	0.30	0.30	0.30	0.80	0.80	0.80	0.30	0.30	0.30	0.30
5	0.30	0.30	0.30	0.80	0.80	0.80	0.30	0.30	0.30	0.30
6	0.30	0.30	0.30	0.80	0.80	0.80	0.30	0.30	0.30	0.30
7	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30
8	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30
9	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30
10	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30

Output image



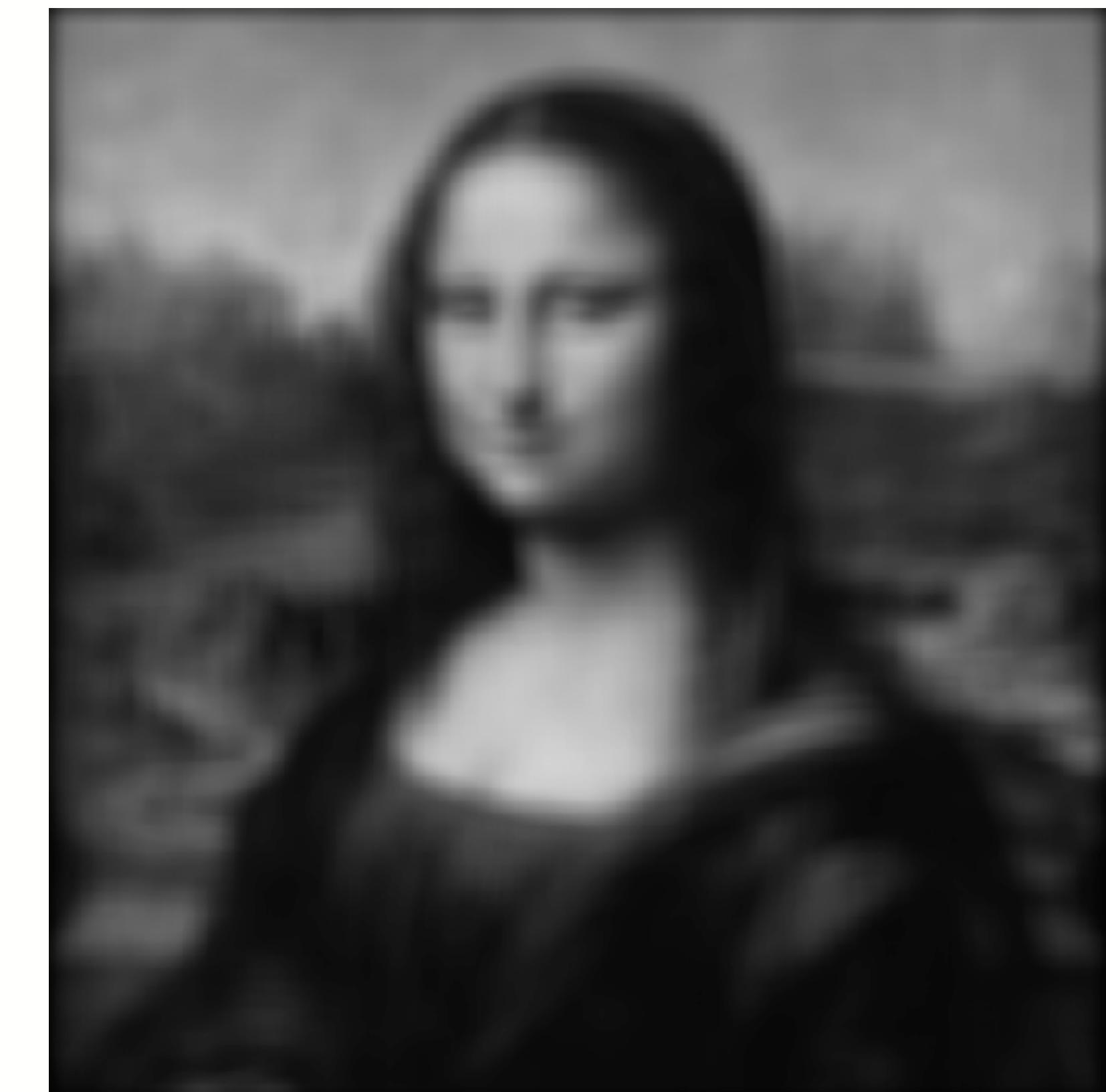
Effect of window size



Original image



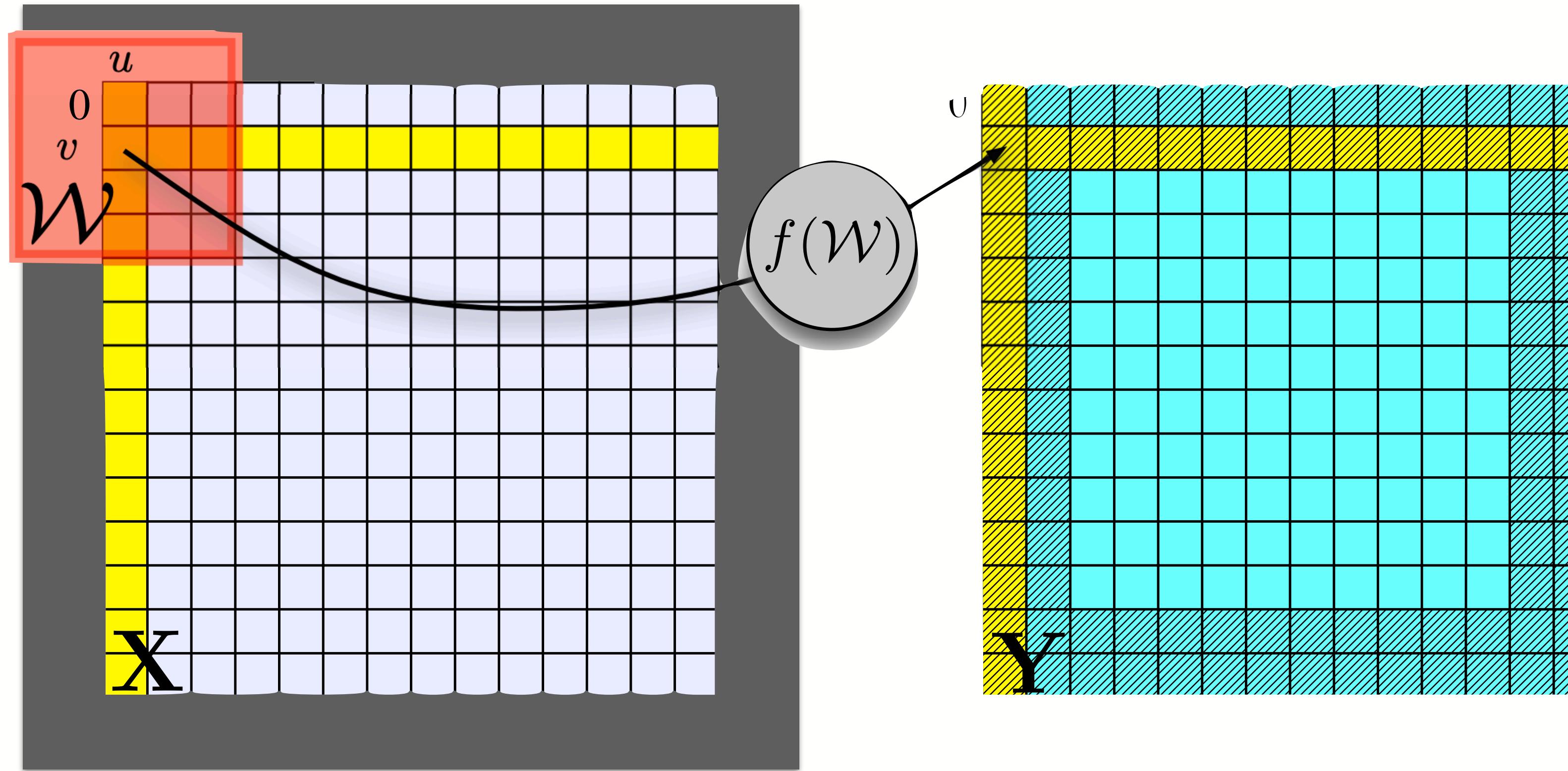
Average each pixel over a
7x7 window



Average each pixel over a
21x 21 window

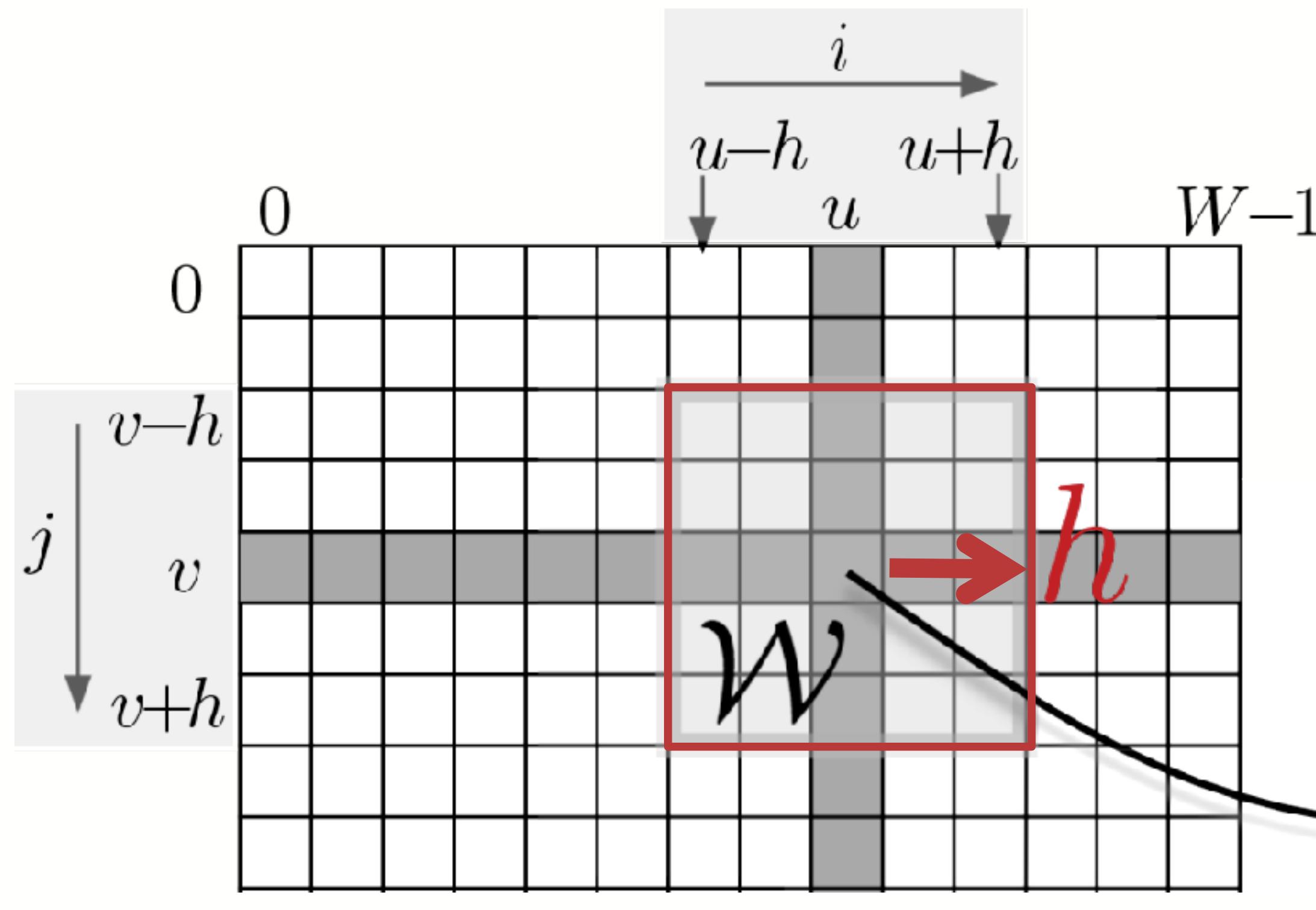
- Features smaller than the window size will be strongly attenuated
 - we say the image is blurry, fuzzy, lower resolution etc.

The edge problem



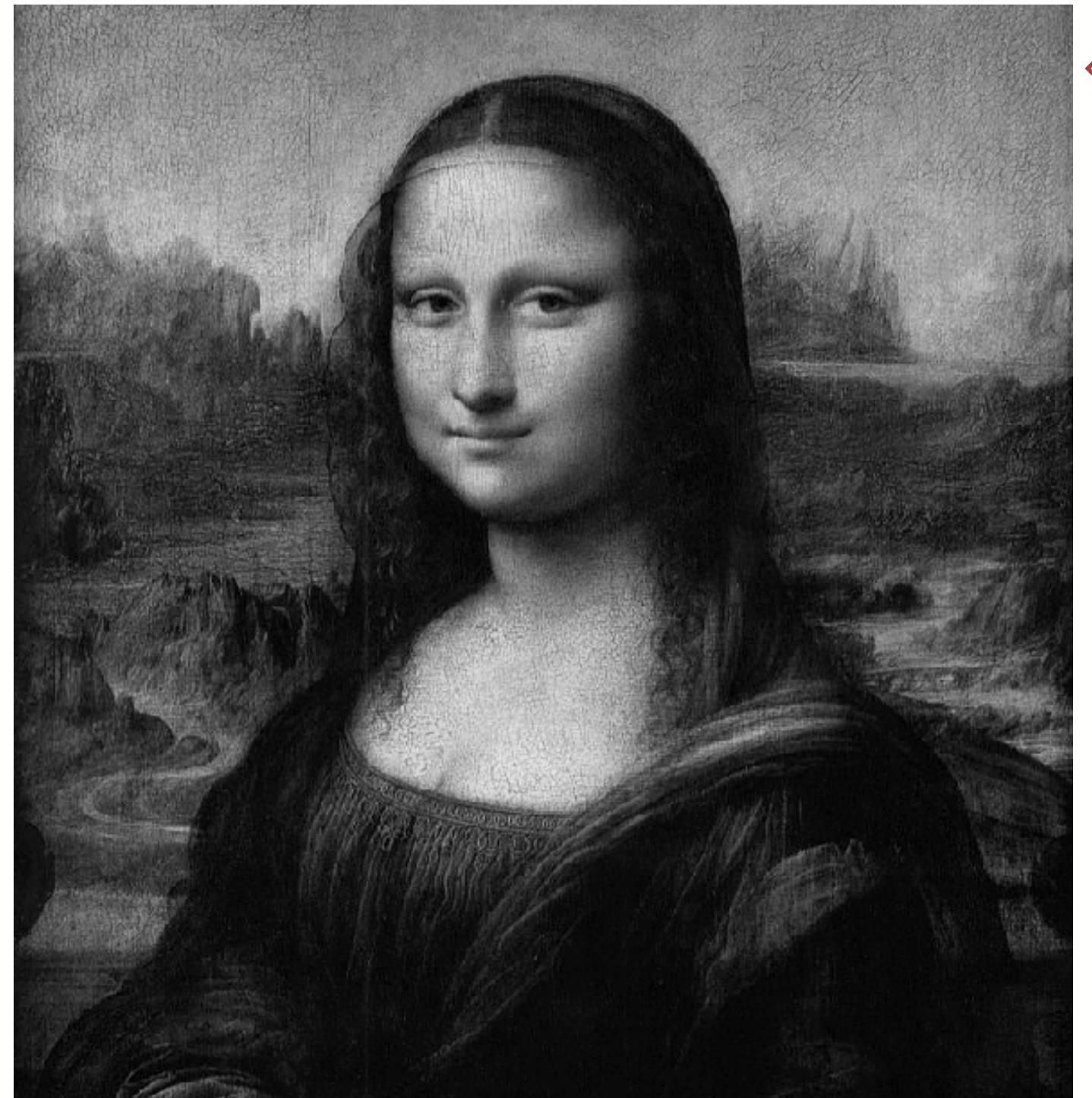
- Some solutions:
 - don't compute the output value when the window "falls off the edge"
 - assume the image is surrounded by zero pixels
 - assume the edge pixels are replicated outward

Window size is always odd



- The window is:
 - square
 - always centred on the input pixel
 - edge is integer h pixels from the centre
- The window width is $2h + 1$
 - always odd

Artefacts of averaging



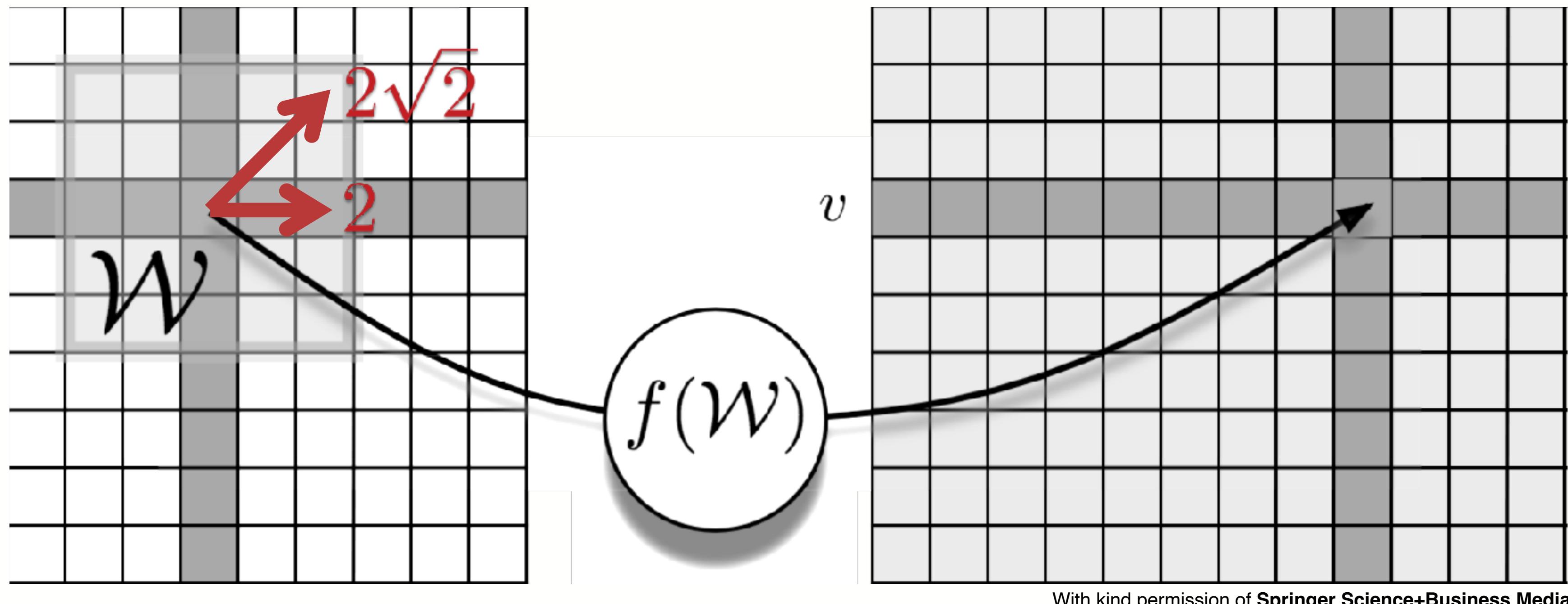
⬅Original image

Average each
pixel over a
21x 21 window⇒



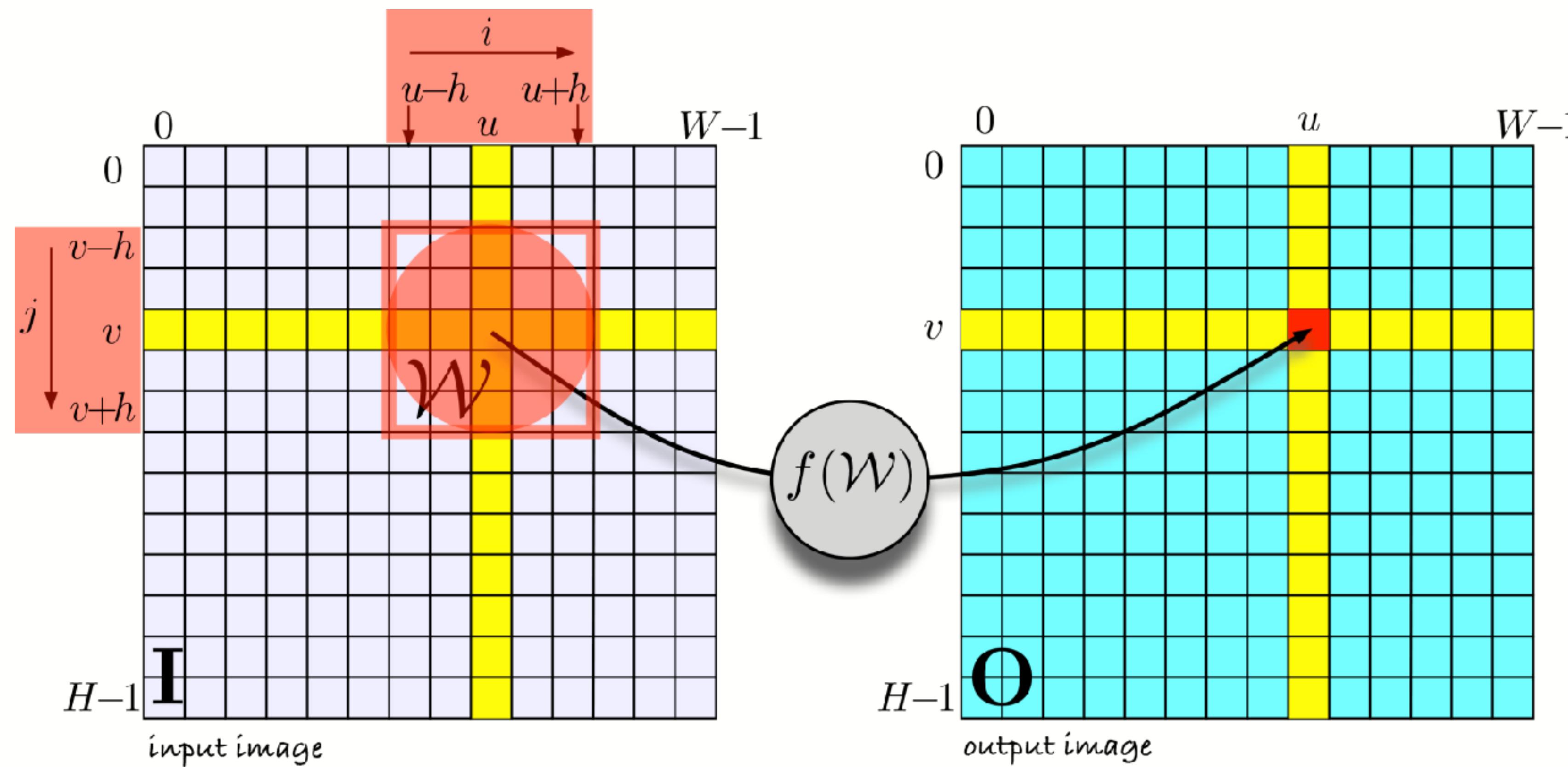
- Can lead to *ringing*
 - faint vertical & horizontal lines are introduced

Averaging over a square is not isotropic



- Not all values used in the average are the same distance away
 - Undue influence by distant values

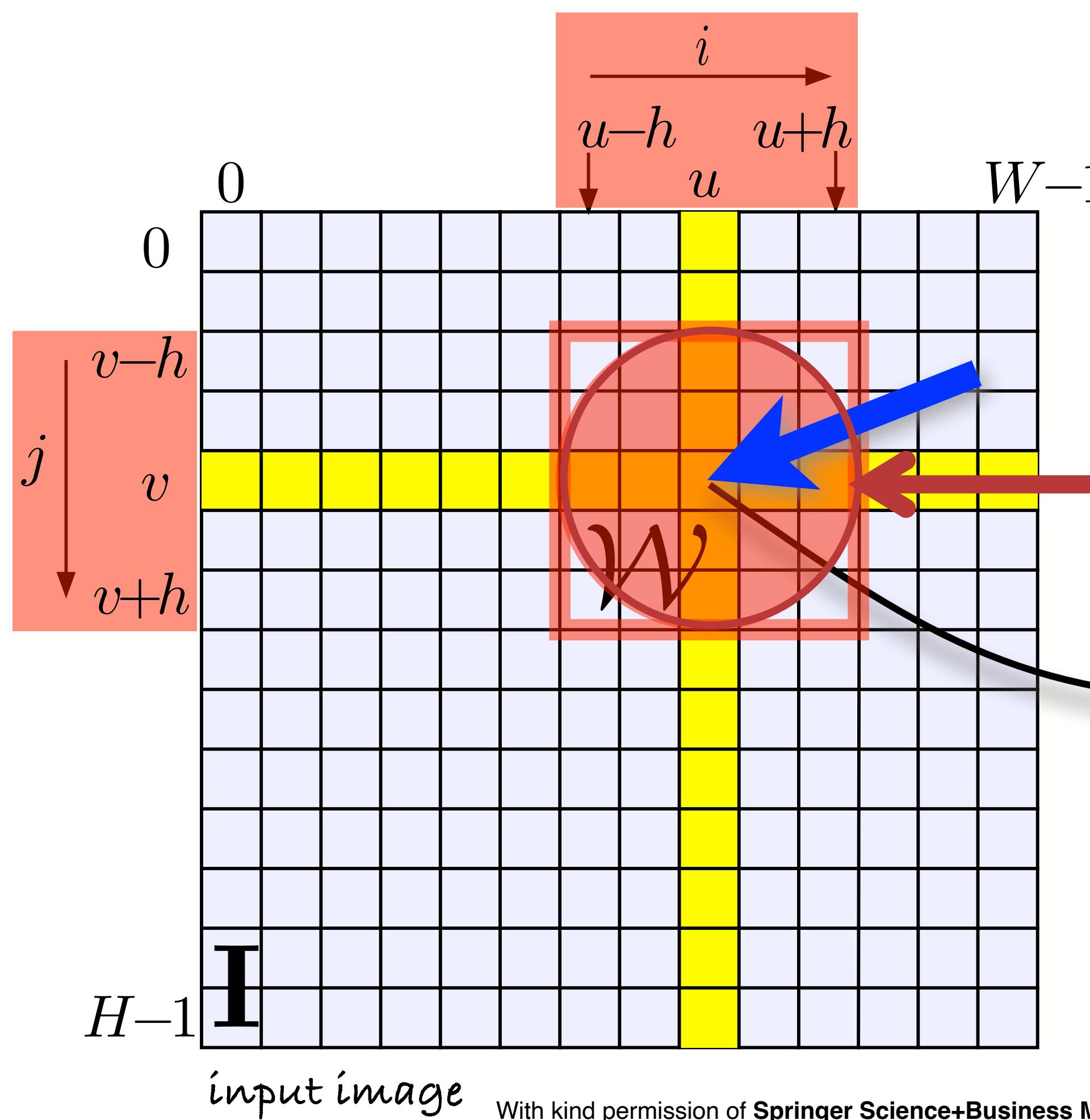
Going isotropic



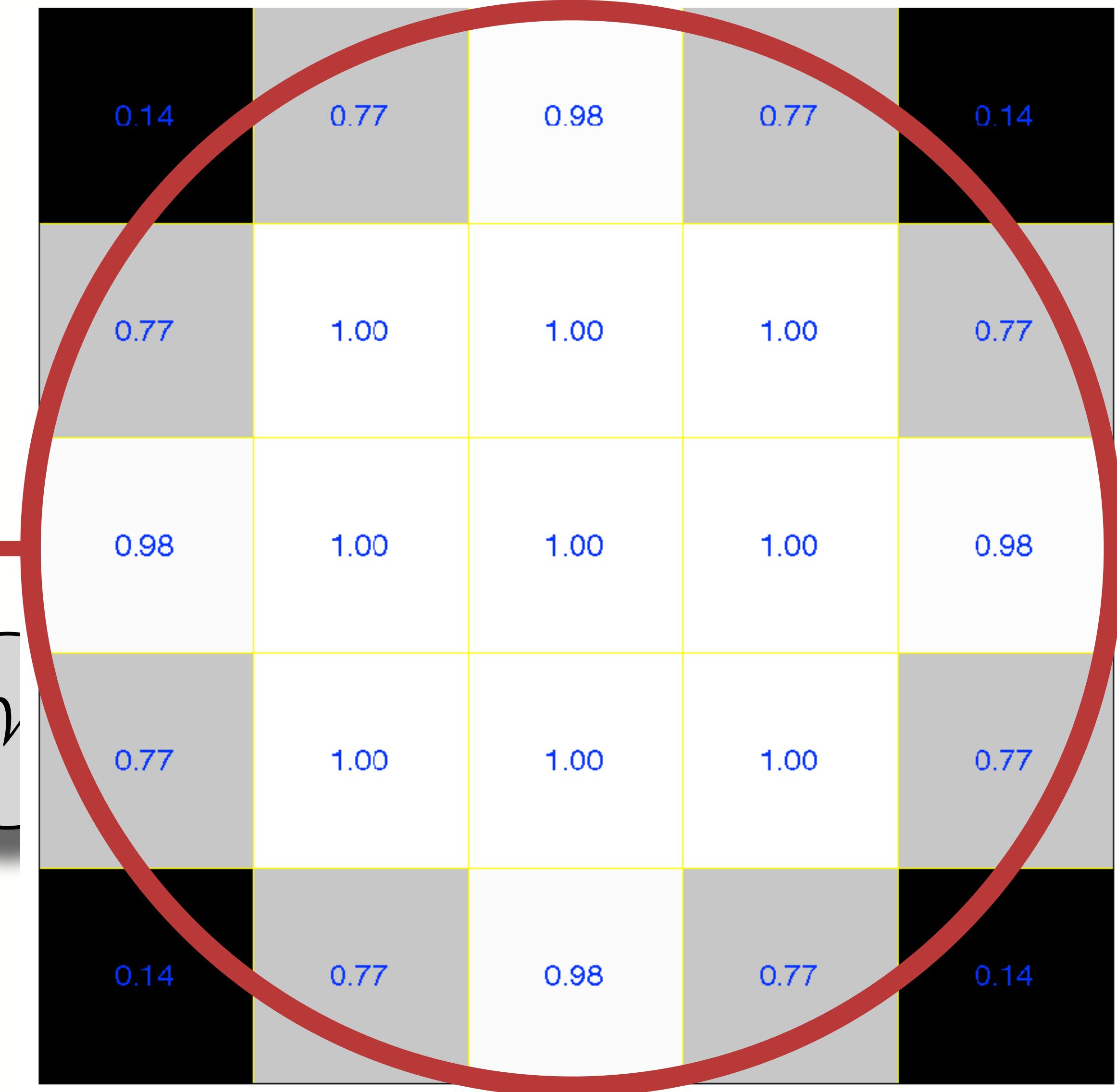
With kind permission of Springer Science+Business Media

- Ideally we'd like to extract a circular region
 - but that would involve taking fractions of pixels

Apply a weighting

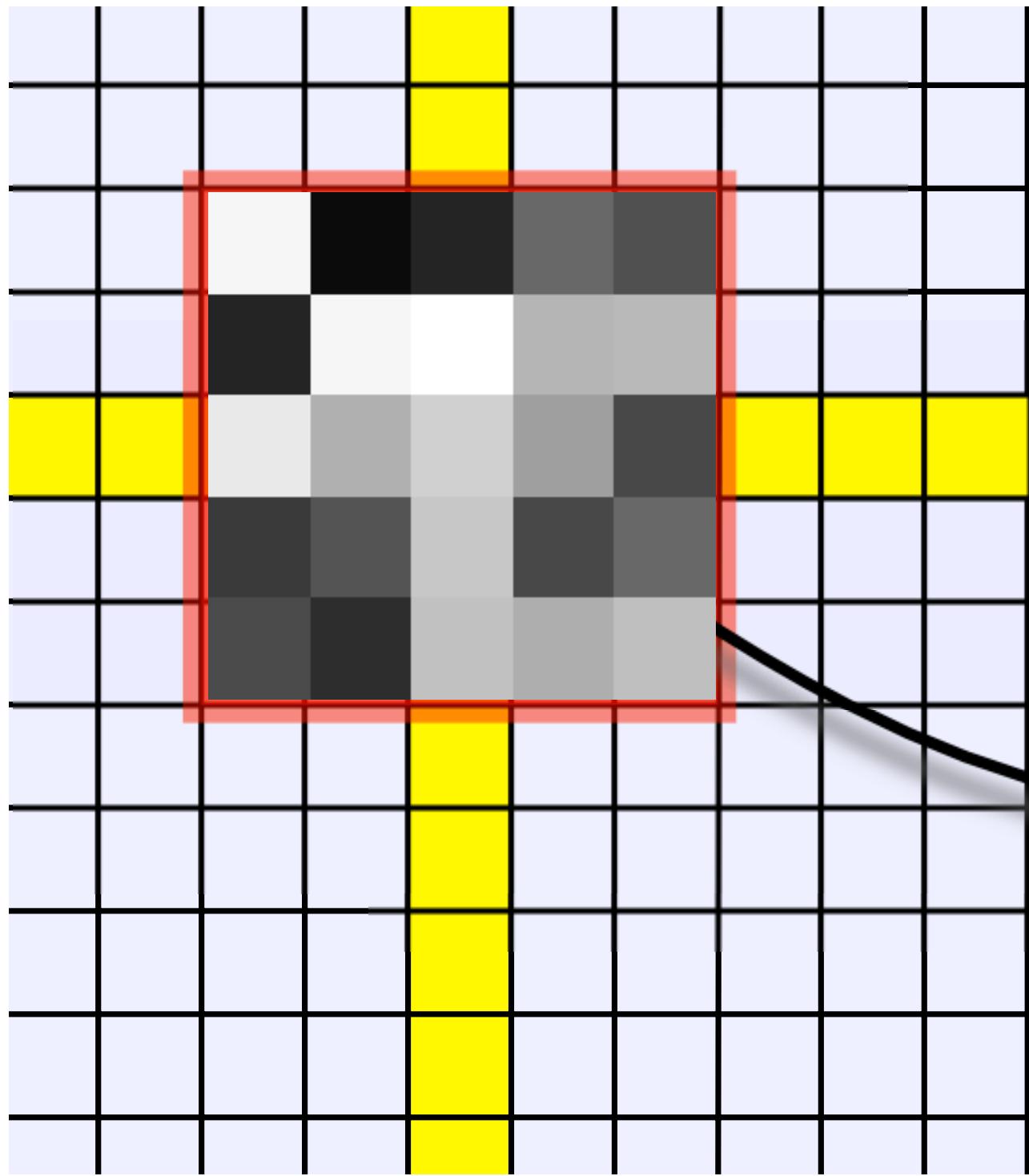


With kind permission of Springer Science+Business Media



- Circle of radius 2.5 pixels

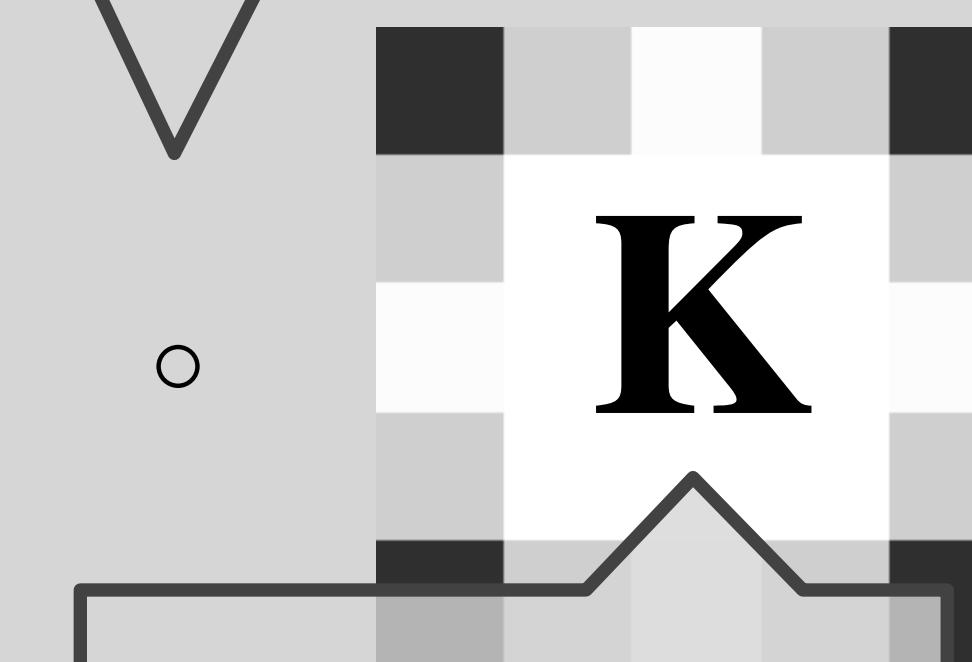
Weighted kernel



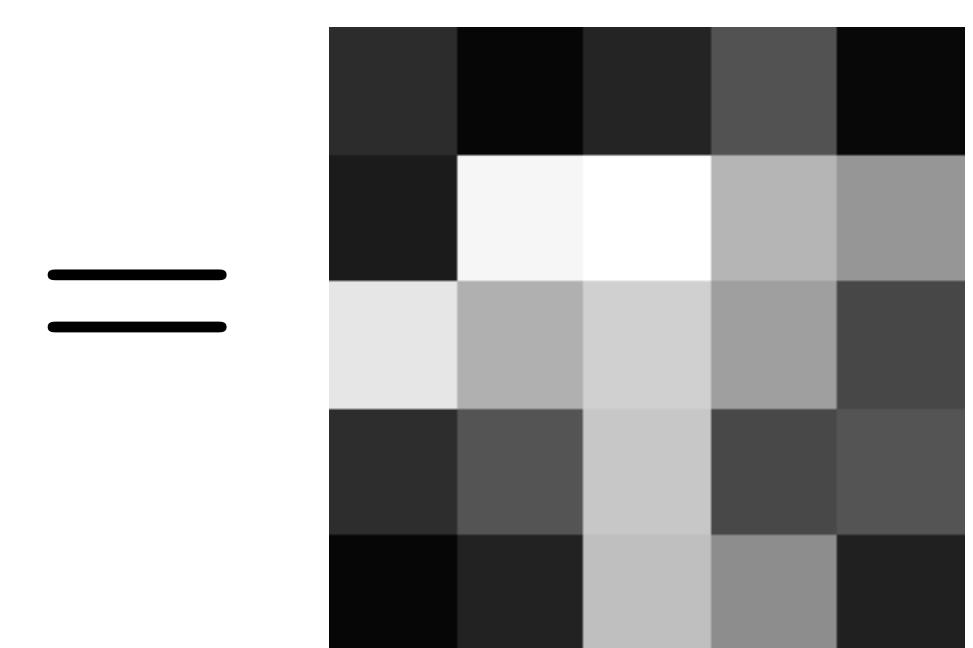
$f(\mathcal{W})$

$$f(\mathcal{W}) = \sum (\mathcal{W} \circ K)$$

Hadamard
product



Kernel



Scaling the kernel

$$\mathbf{K} = \begin{pmatrix} 0.14 & 0.77 & 0.98 & 0.77 & 0.14 \\ 0.77 & 1.0 & 1.0 & 1.0 & 0.77 \\ 0.98 & 1.0 & 1.0 & 1.0 & 0.98 \\ 0.77 & 1.0 & 1.0 & 1.0 & 0.77 \\ 0.14 & 0.77 & 0.98 & 0.77 & 0.14 \end{pmatrix}$$

Total value is 19.6

- The scale factor is $S = \sum |\mathbf{K}_{i,j}|$

Scaling the kernel

$$\mathbf{K} = \frac{1}{19.6} \begin{pmatrix} 0.14 & 0.77 & 0.98 & 0.77 & 0.14 \\ 0.77 & 1.0 & 1.0 & 1.0 & 0.77 \\ 0.98 & 1.0 & 1.0 & 1.0 & 0.98 \\ 0.77 & 1.0 & 1.0 & 1.0 & 0.77 \\ 0.14 & 0.77 & 0.98 & 0.77 & 0.14 \end{pmatrix}$$

- The scale factor is $S = \sum |\mathbf{K}_{i,j}|$
- Typically make $S = 1$ to keep grey levels the same as the input image

Simple averaging is also a kernel



Original image

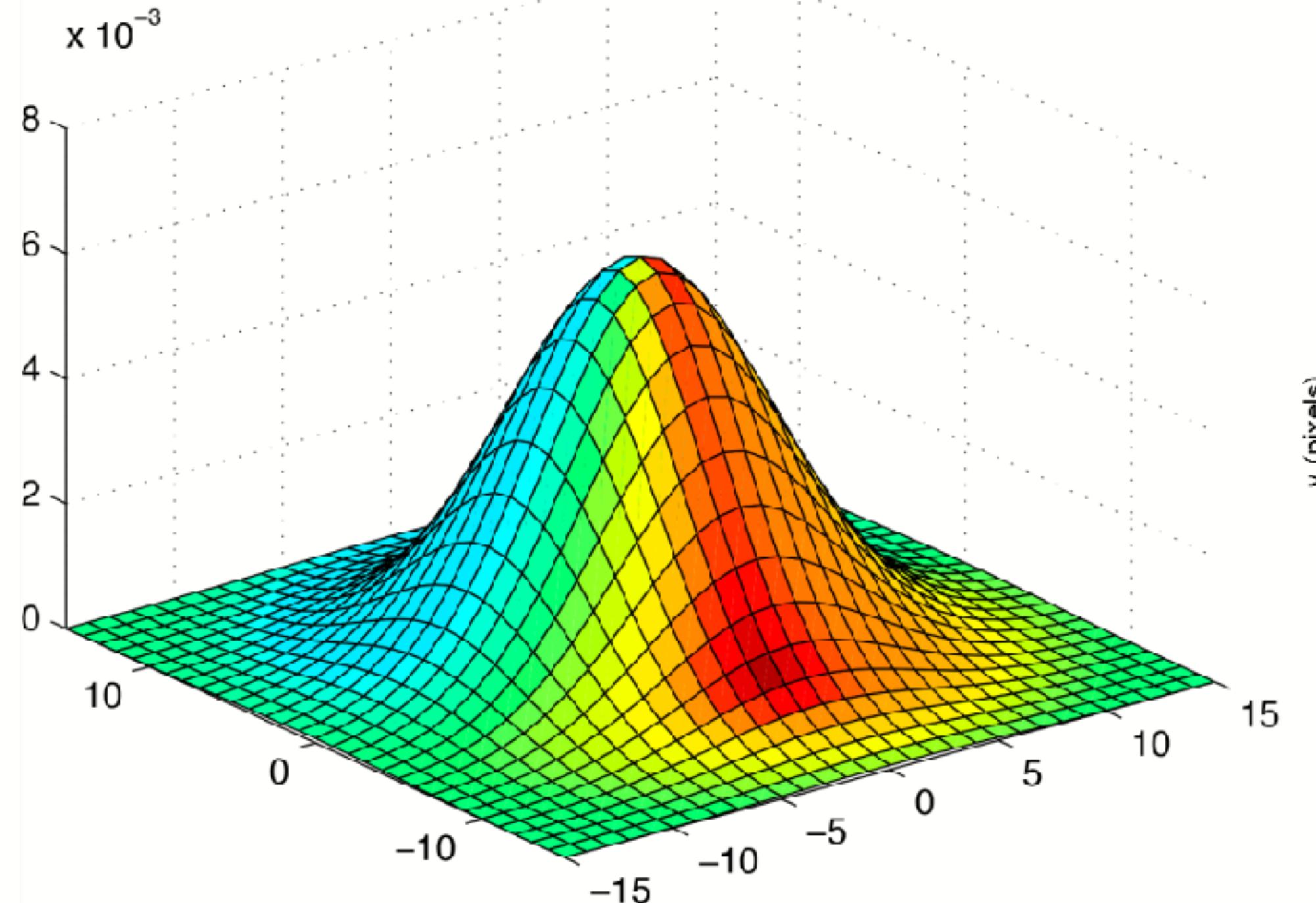


Average each pixel over a 7x7 window

$$\mathbf{K} = \frac{1}{49} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

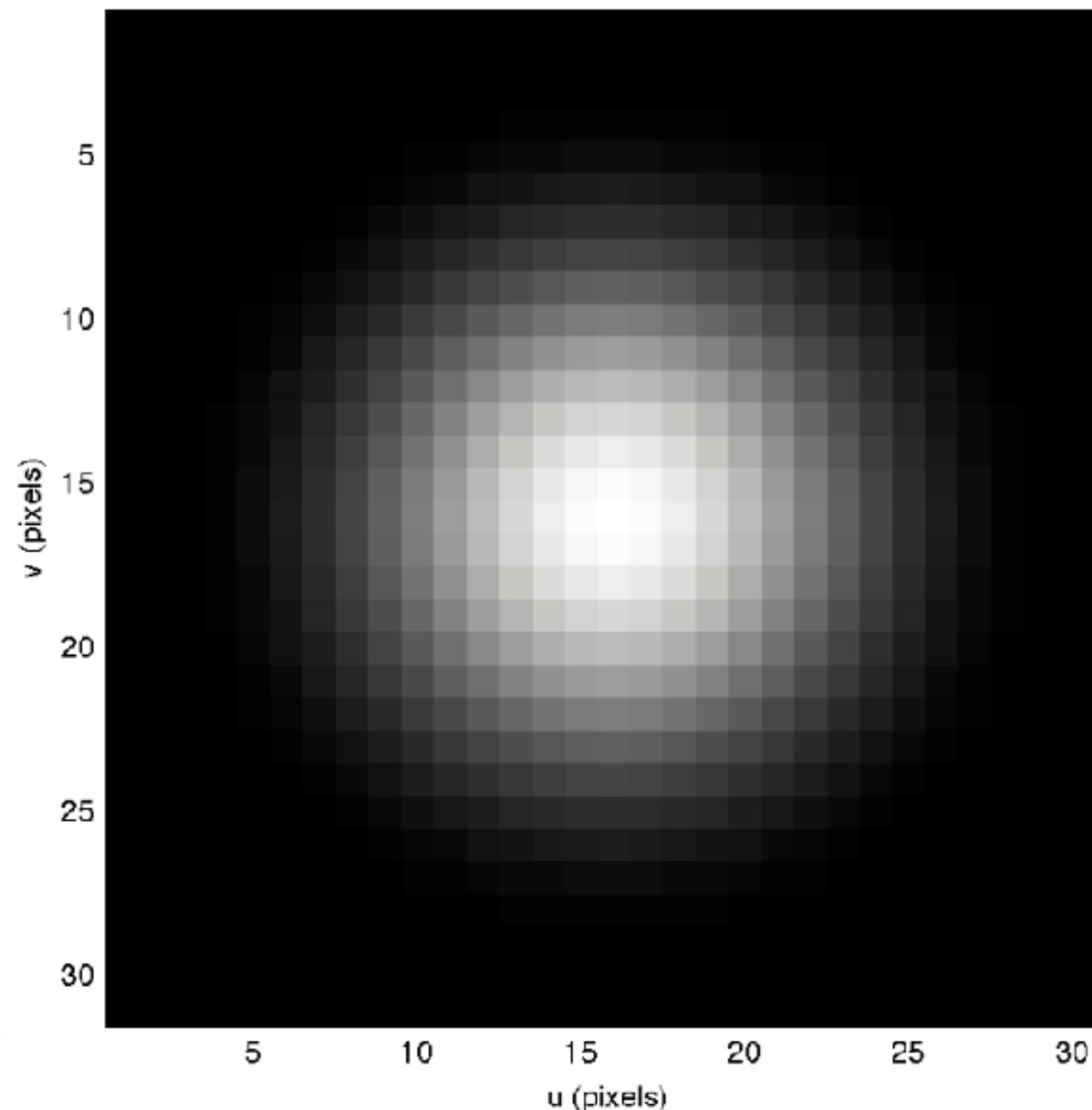
```
>>> out = mona.convolve(np.ones((7,7)) / 49)
```

Gaussian kernel



With kind permission of Springer Science+Business Media

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



- Isotropic
- Decreasing weight away from centre

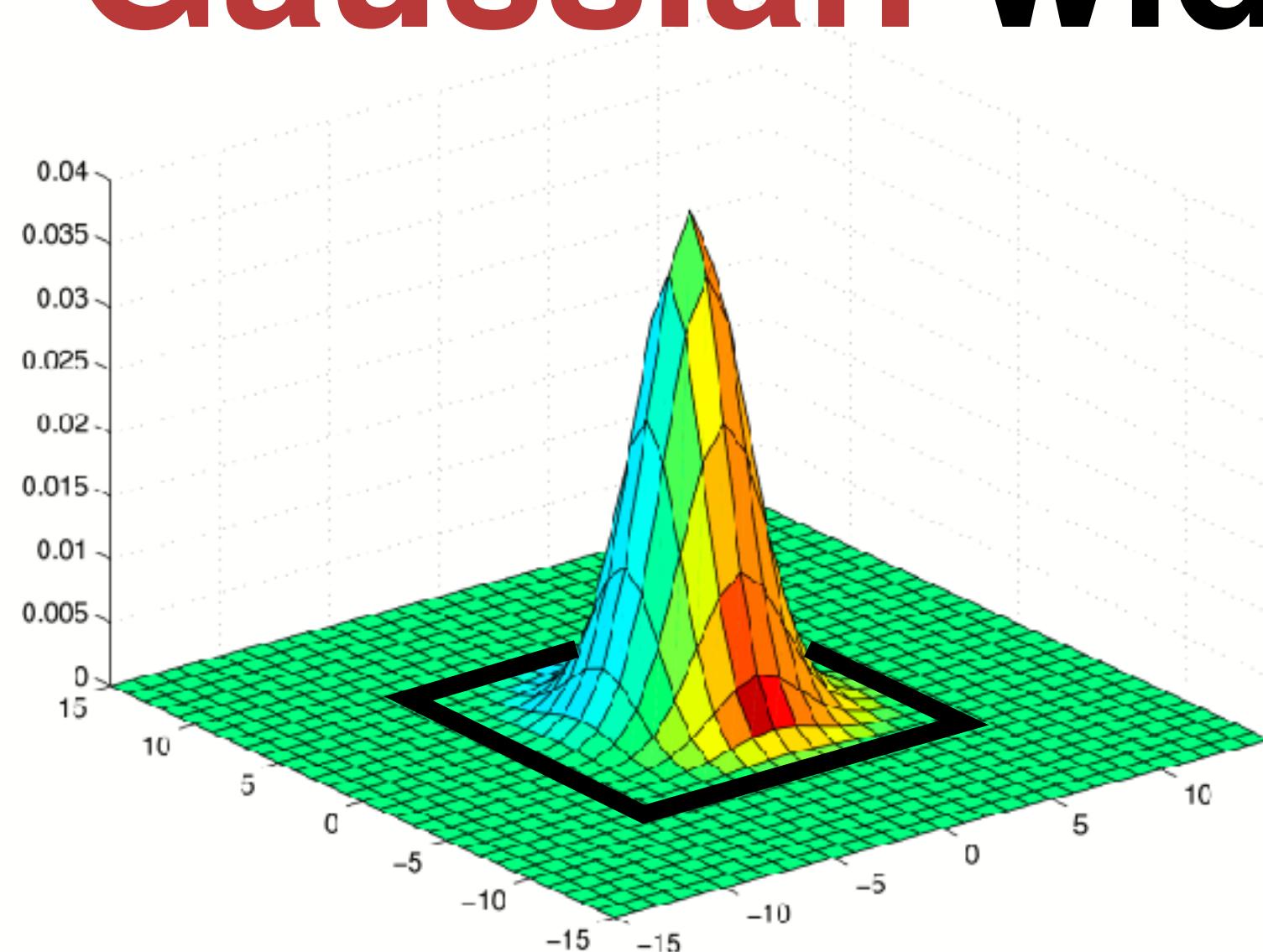


Carl-Friedrich Gauss
1777–1855

By Gottlieb Biermann A. Wittmann (photo)
[Public domain], via Wikimedia Commons

Gaussian width

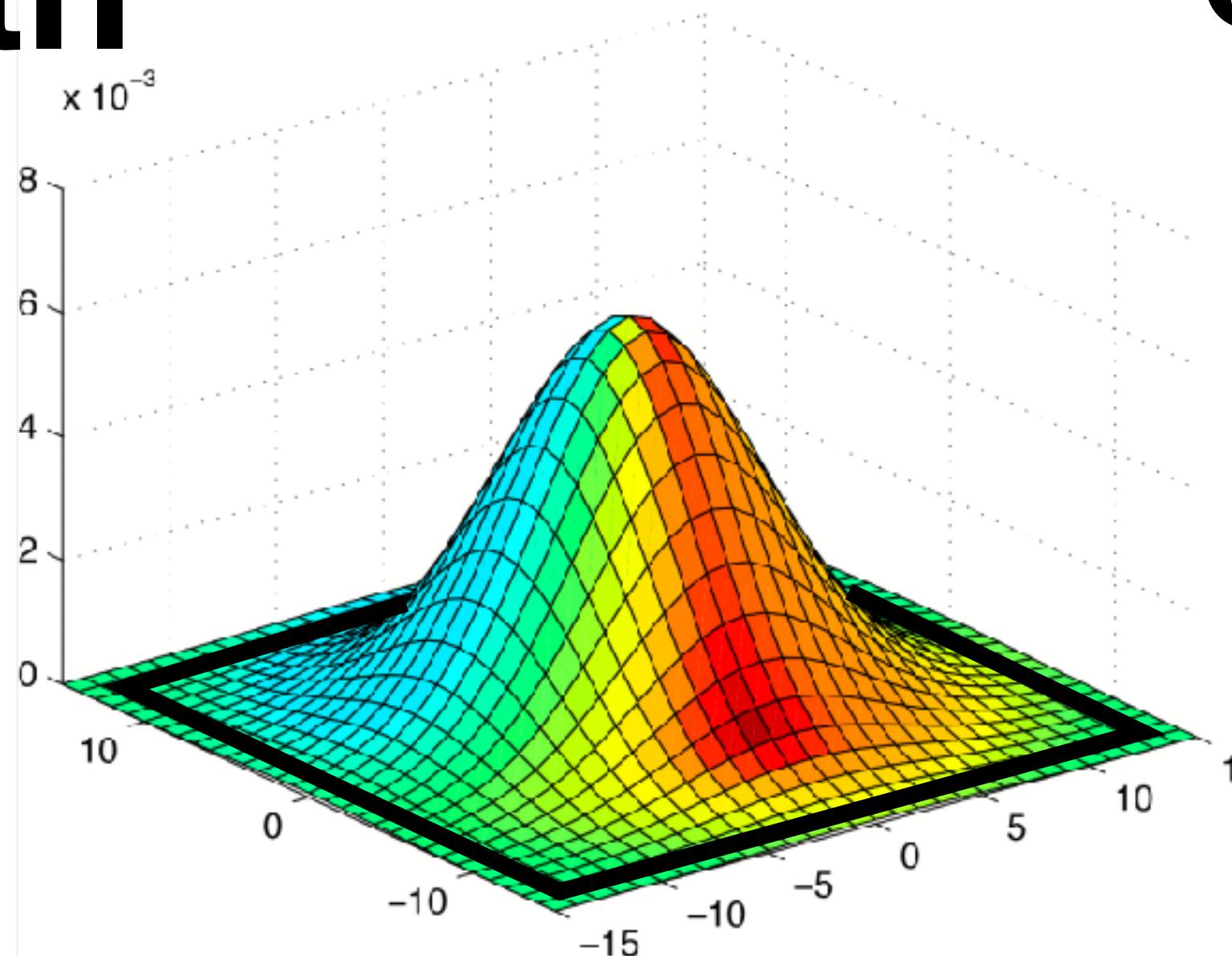
$$G(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



With kind permission of Springer Science+Business Media

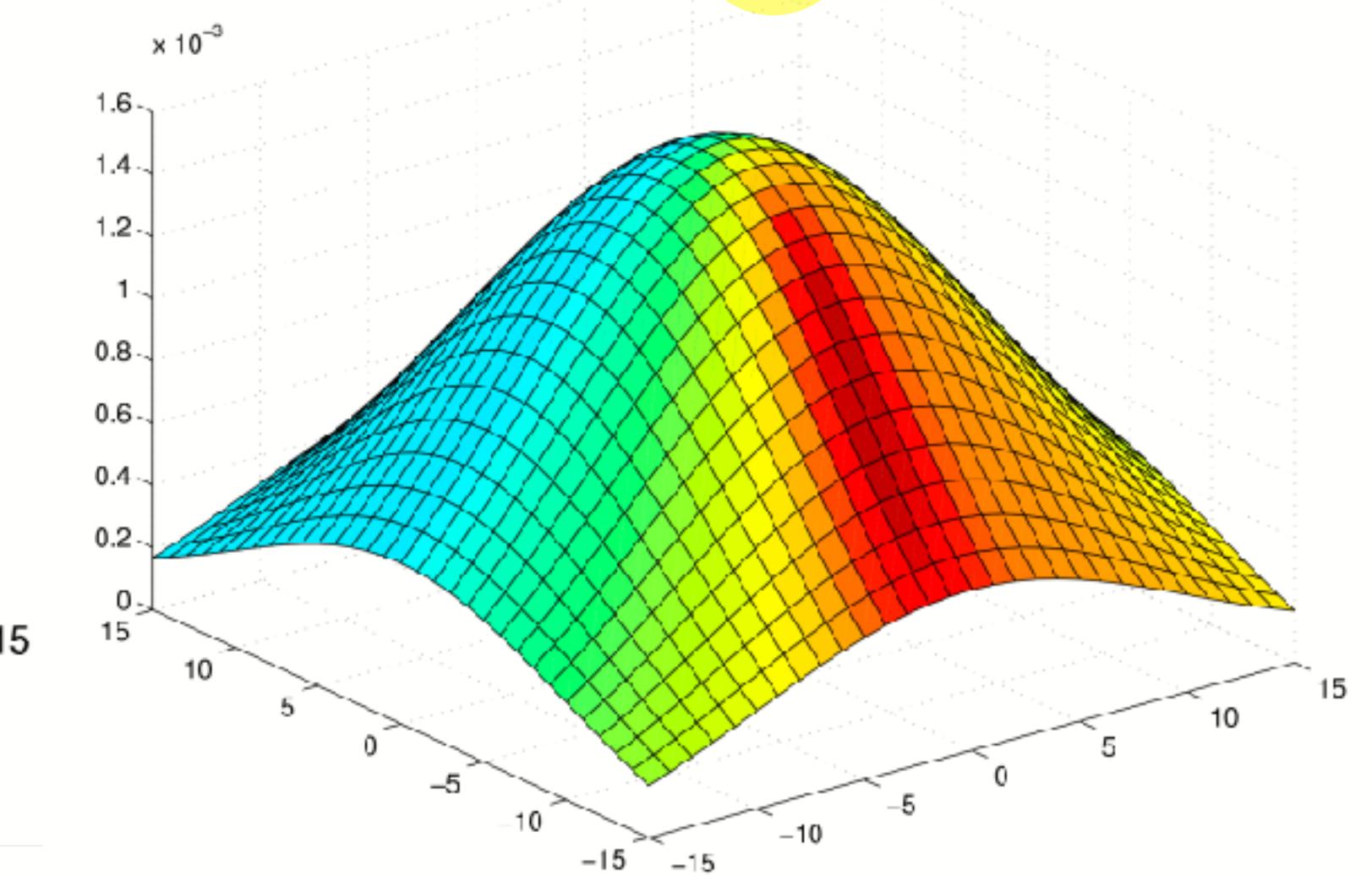
$$\sigma = 2$$

`Kernel.Gauss(2, 15)`



$$\sigma = 5$$

`Kernel.Gauss(5, 15)`

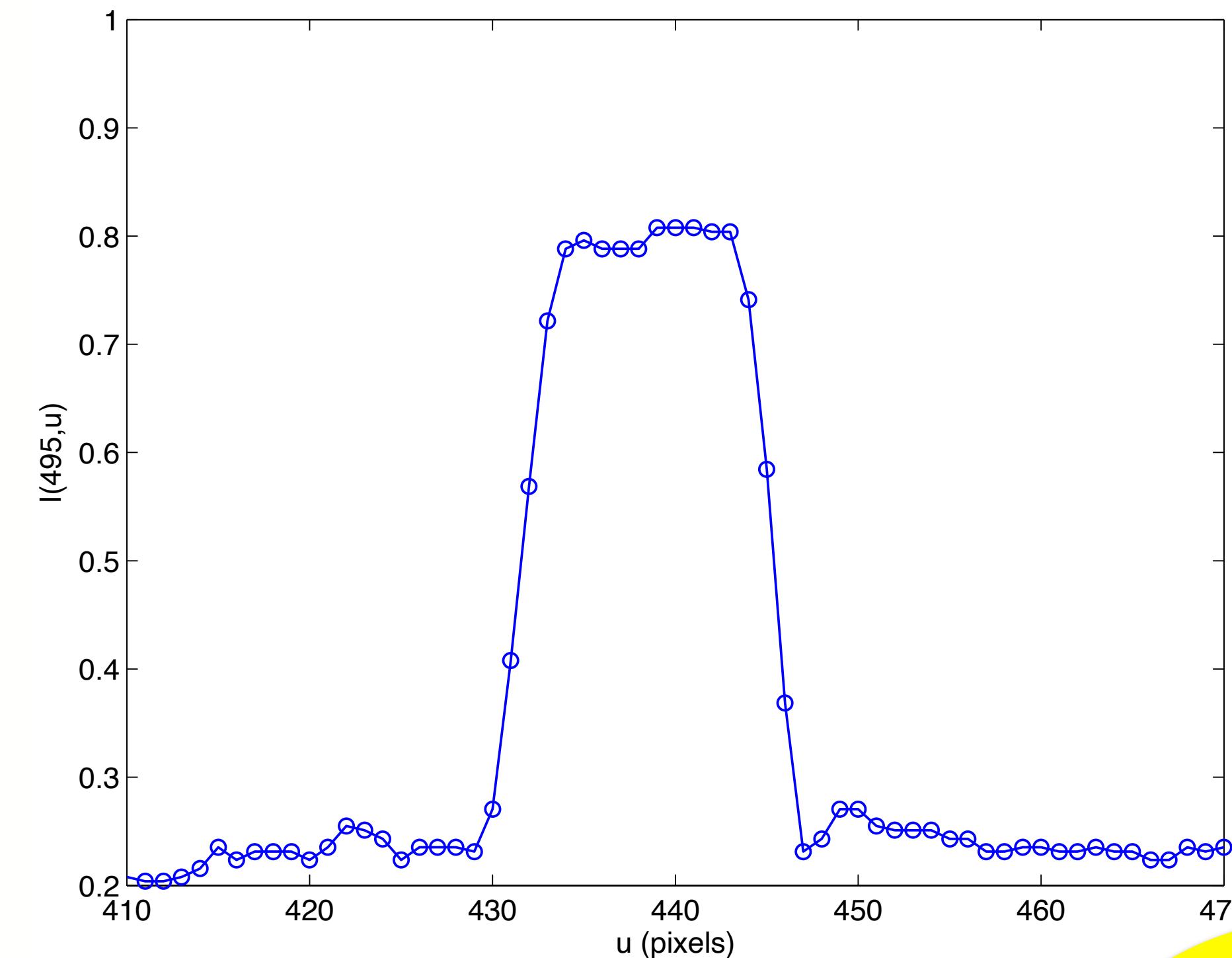
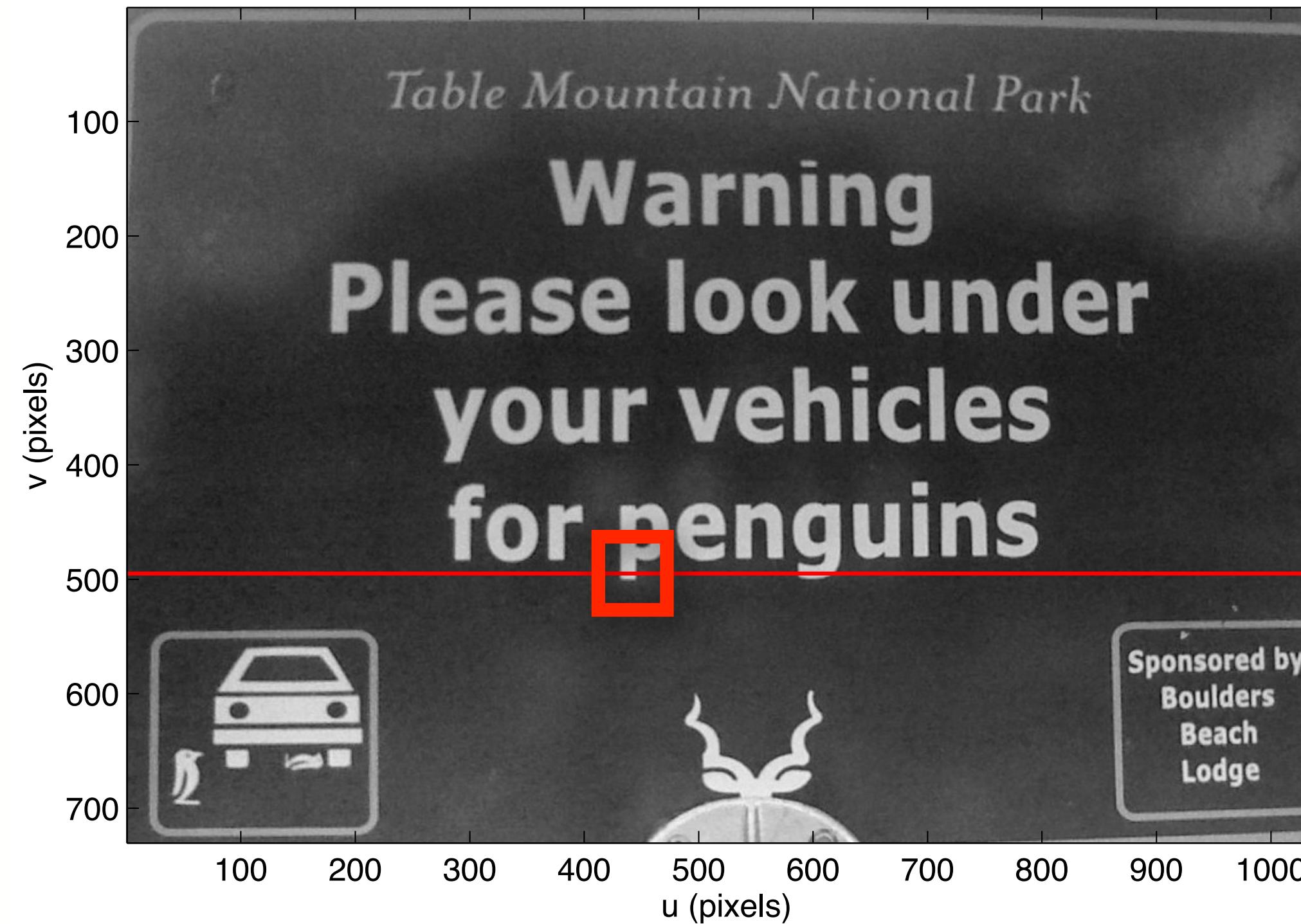


$$\sigma = 10$$

`Kernel.Gauss(10, 15)`

- Choose the size of the square kernel to fit the Gaussian
- Rule of thumb $h=3\sigma$

The intensity function



- Horizontal gradient can be approximated:

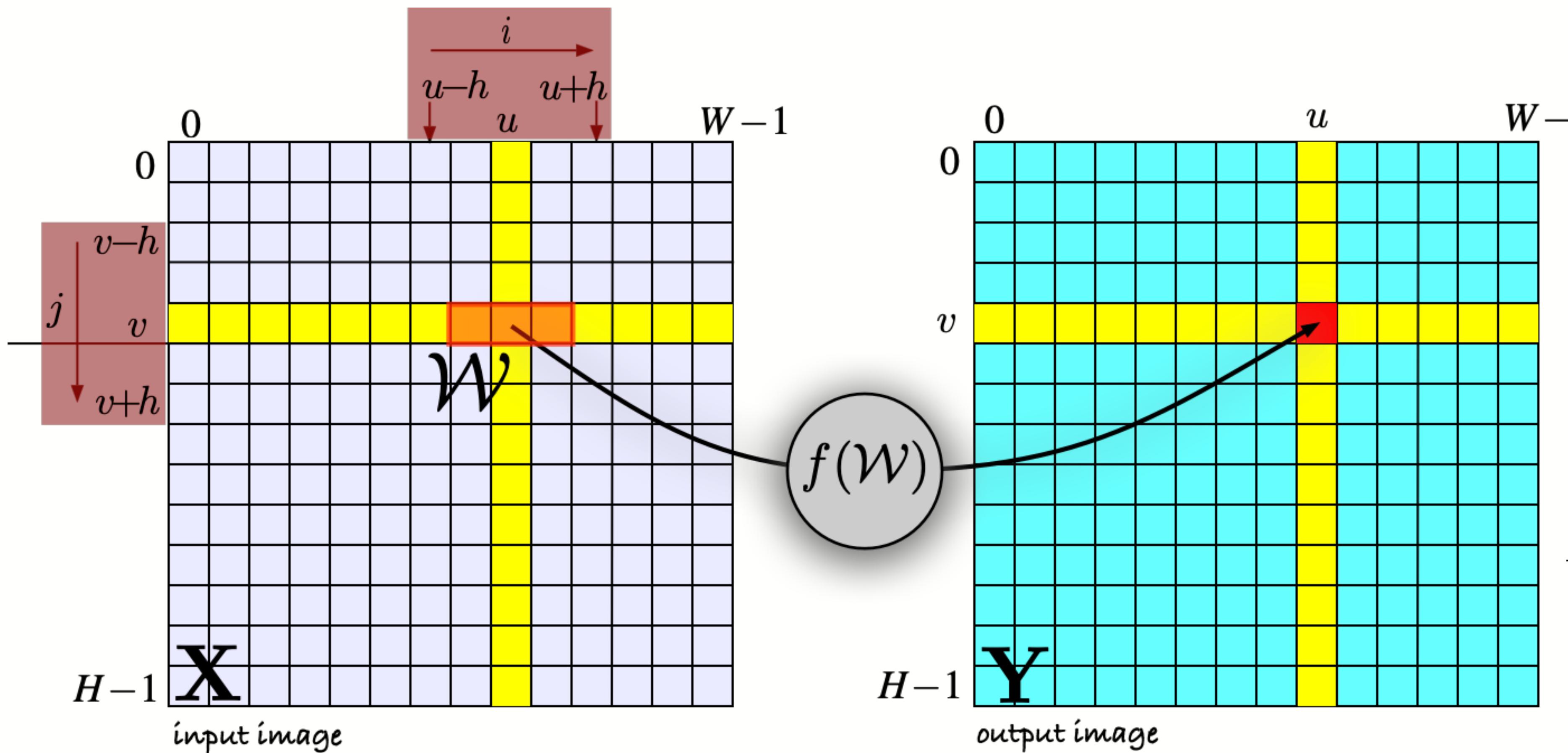
$$\frac{dI}{du} \approx I(u+1) - I(u)$$

regular “first-order difference”

$$\left. \frac{dI}{du} \right|_u \approx \frac{I(u+1) - I(u-1)}{2}$$

symmetrical difference about u

Expressed as correlation with a kernel



With kind permission of Springer Science+Business Media

$$\frac{d\mathbf{X}}{du} \Big|_u \approx \frac{-\mathbf{X}(u-1) + \mathbf{X}(u+1)}{2}$$

$$\mathbf{K} = \frac{1}{2} \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$$

Common confusion. Correlation and convolution are similar but different.

correlation

$$y_{u,v} = \sum_{(i,j) \in \mathcal{W}} x_{u+i, v+j} k_{i,j}, \quad \forall (u, v) \in \mathbf{X}$$

convolution

$$y_{u,v} = \sum_{(i,j) \in \mathcal{W}} x_{u-i, v-j} k_{i,j}, \quad \forall (u, v) \in \mathbf{X}$$

If the kernel is symmetric then they are the same

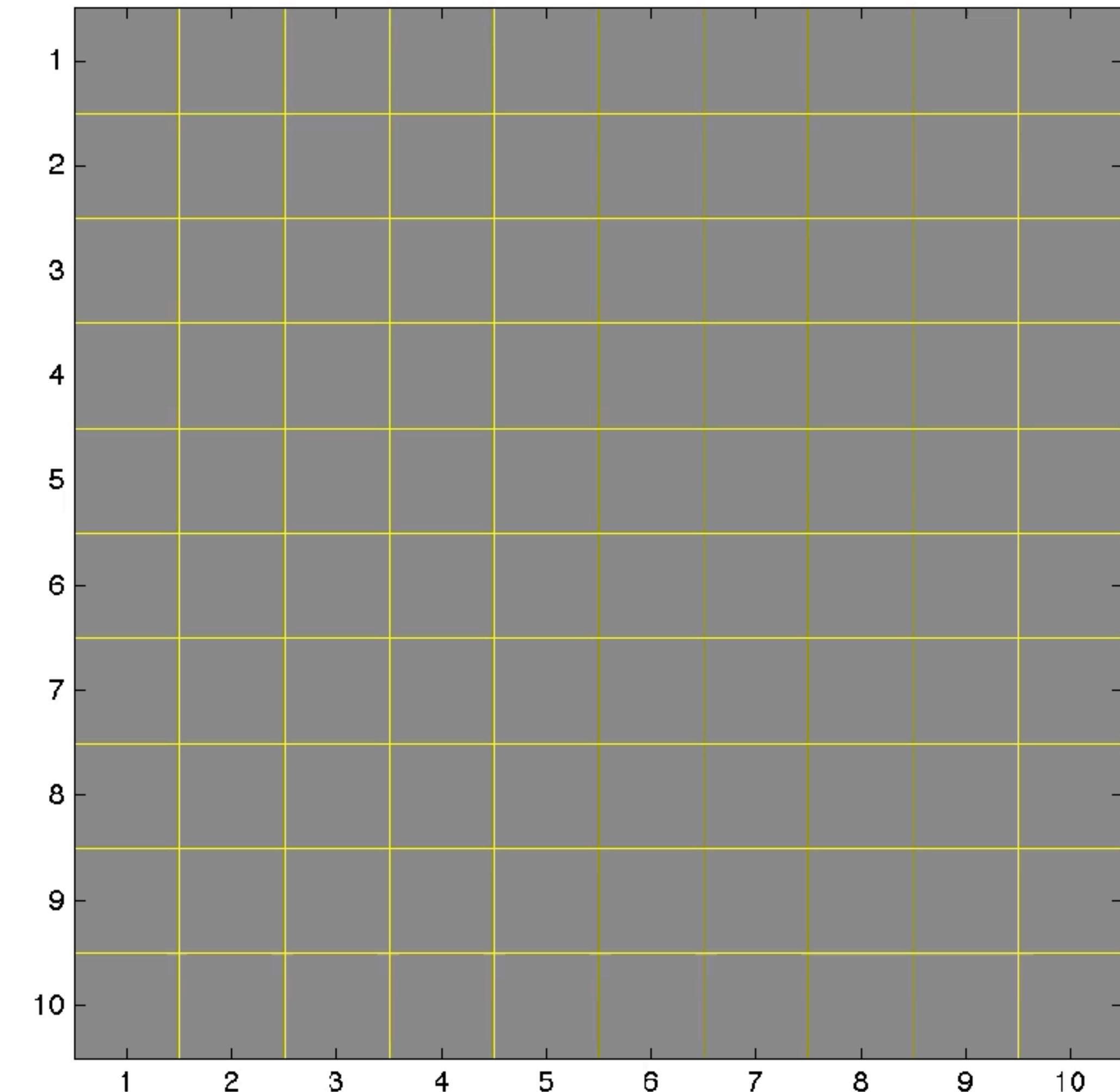
- smoothing kernels are always symmetric
- derivative kernels are never symmetric

Convolution for edge detection

Input image

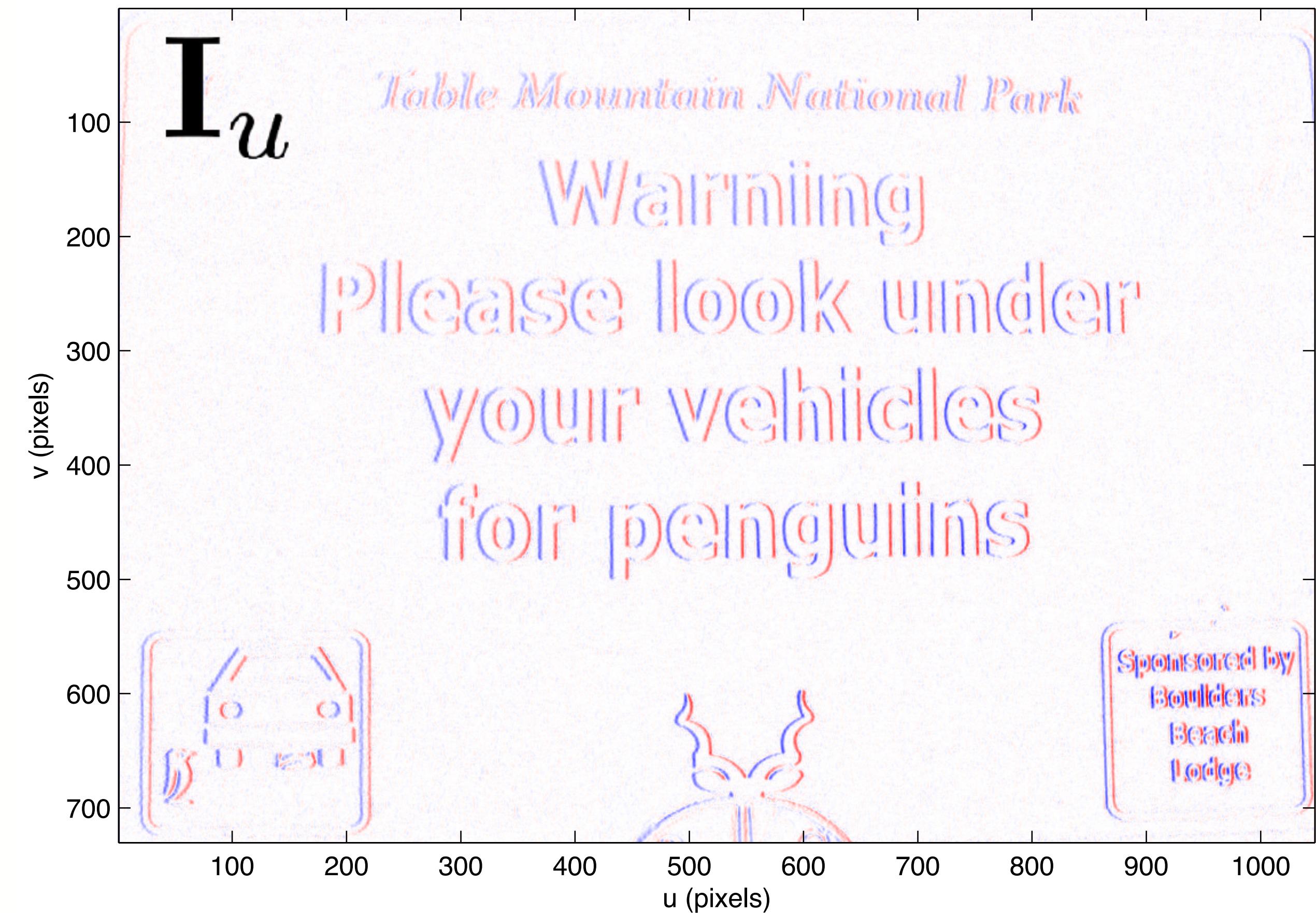
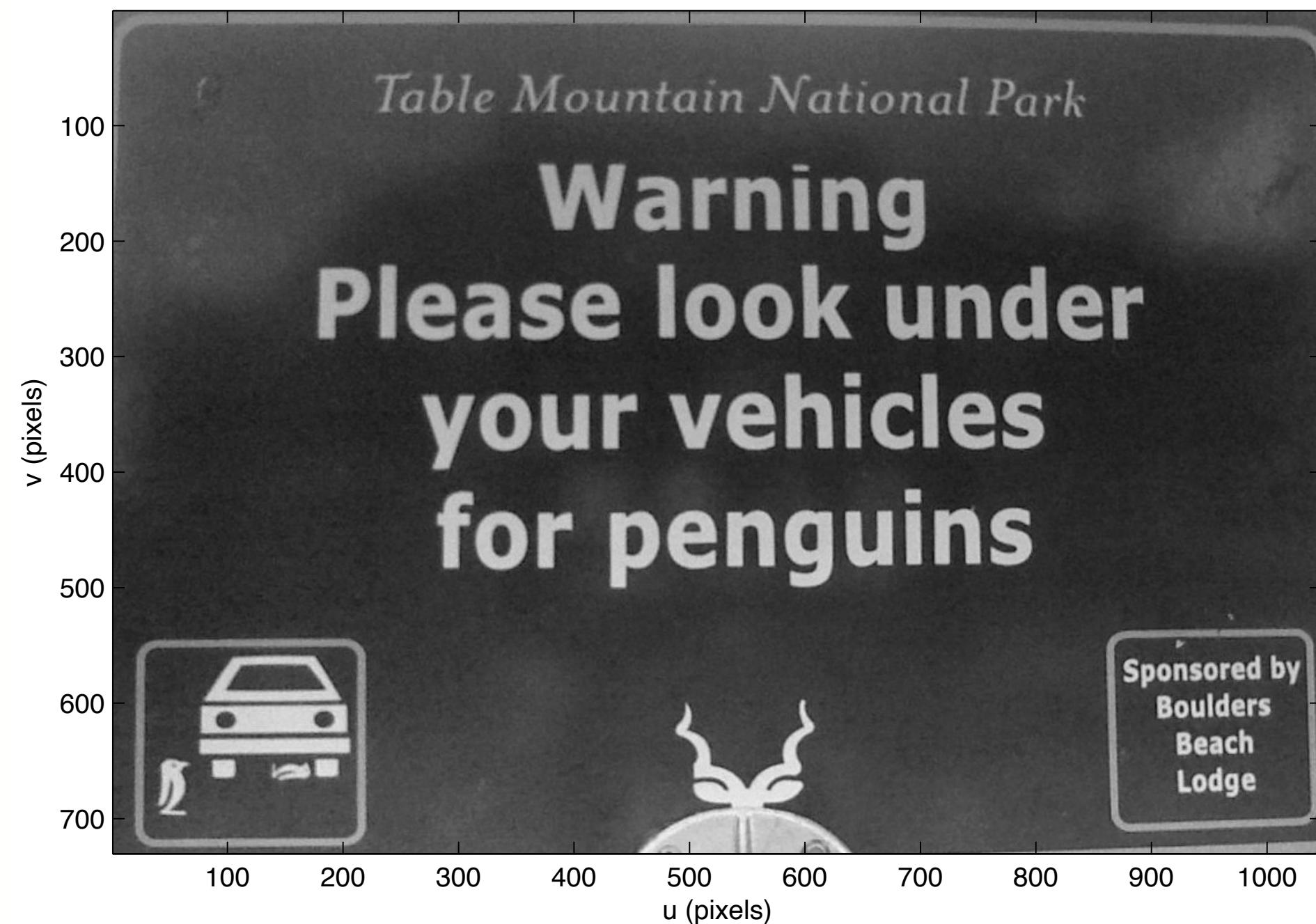
1	0.15	0.21	0.28	0.35	0.41	0.65	0.72	0.79	0.85	0.92
2	0.16	0.23	0.30	0.36	0.43	0.67	0.74	0.80	0.87	0.93
3	0.18	0.25	0.31	0.38	0.44	0.69	0.75	0.82	0.88	0.95
4	0.20	0.26	0.33	0.39	0.46	0.70	0.77	0.84	0.90	0.97
5	0.21	0.28	0.35	0.41	0.48	0.72	0.79	0.85	0.92	0.98
6	0.23	0.30	0.36	0.43	0.49	0.74	0.80	0.87	0.93	1.00
7	0.25	0.31	0.38	0.44	0.51	0.28	0.25	0.21	0.18	0.15
8	0.26	0.33	0.39	0.46	0.53	0.28	0.25	0.21	0.18	0.15
9	0.28	0.35	0.41	0.48	0.54	0.28	0.25	0.21	0.18	0.15
10	0.30	0.36	0.43	0.49	0.56	0.28	0.25	0.21	0.18	0.15

Output image



$$K = \frac{1}{2} \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$$

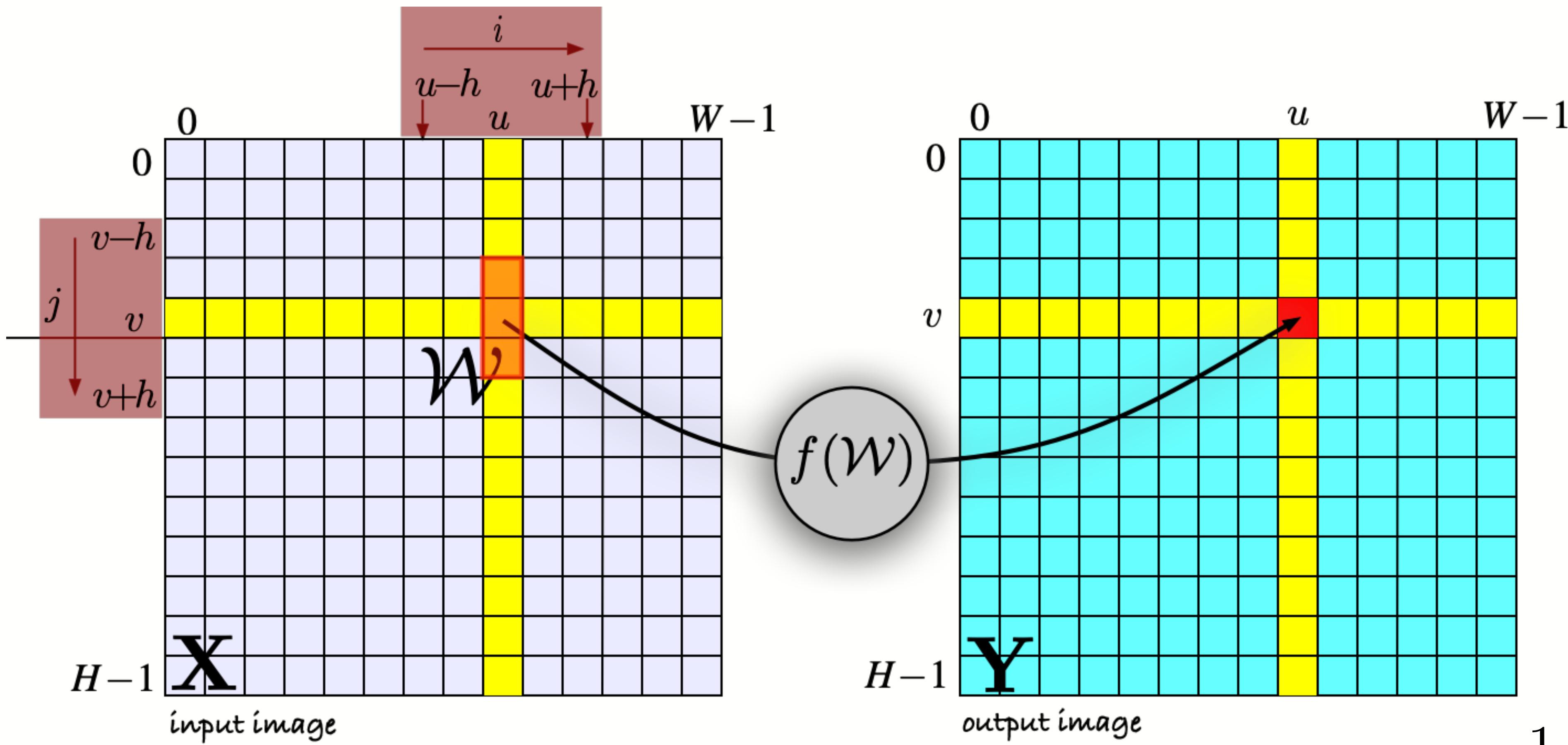
Horizontal gradient image



$$K = \frac{1}{2} \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$$

```
>>> K = np.c_[1, 0, -1] / 2
>>> im.convolve(K)
```

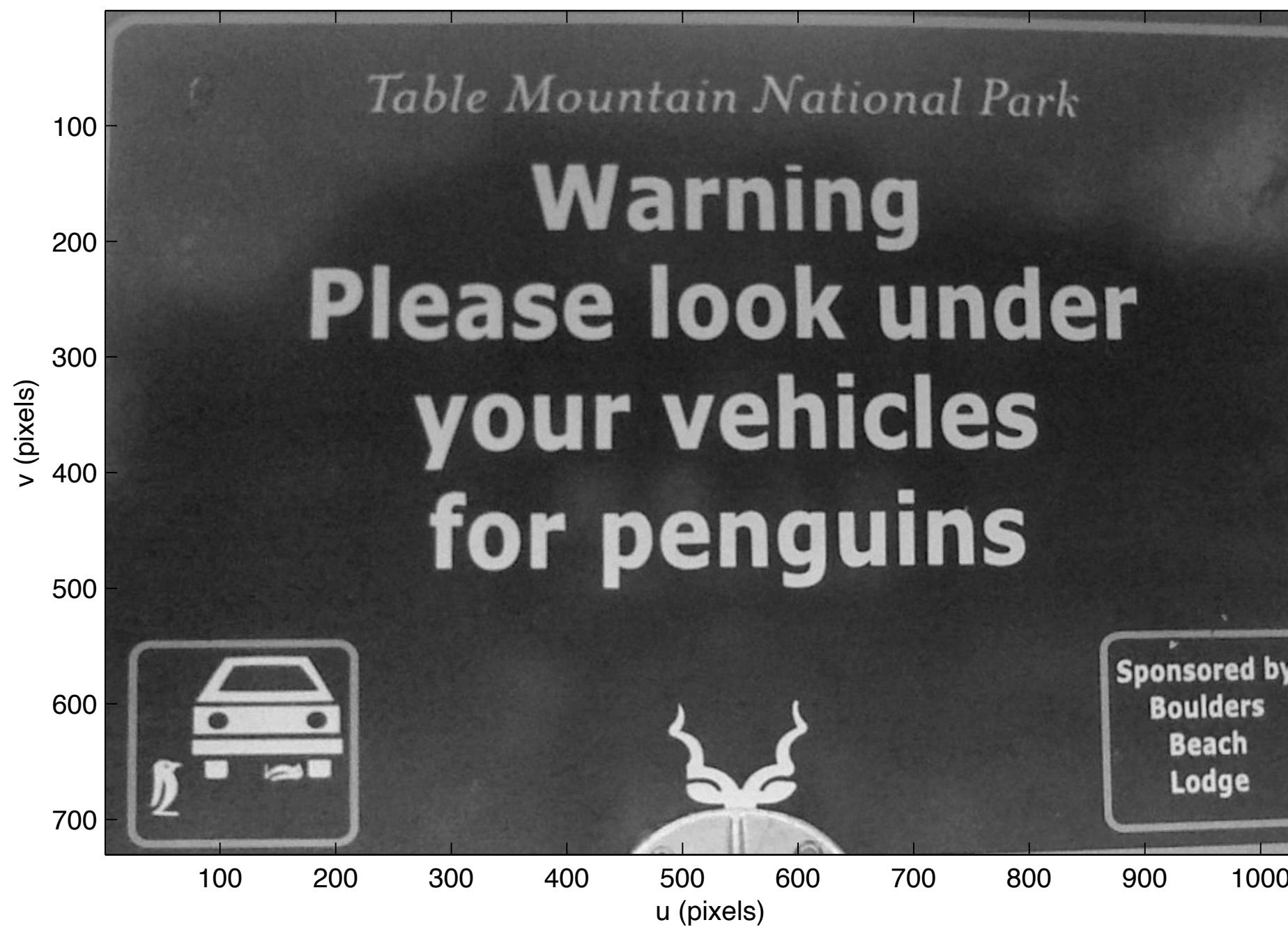
Vertical gradient as correlation with a kernel



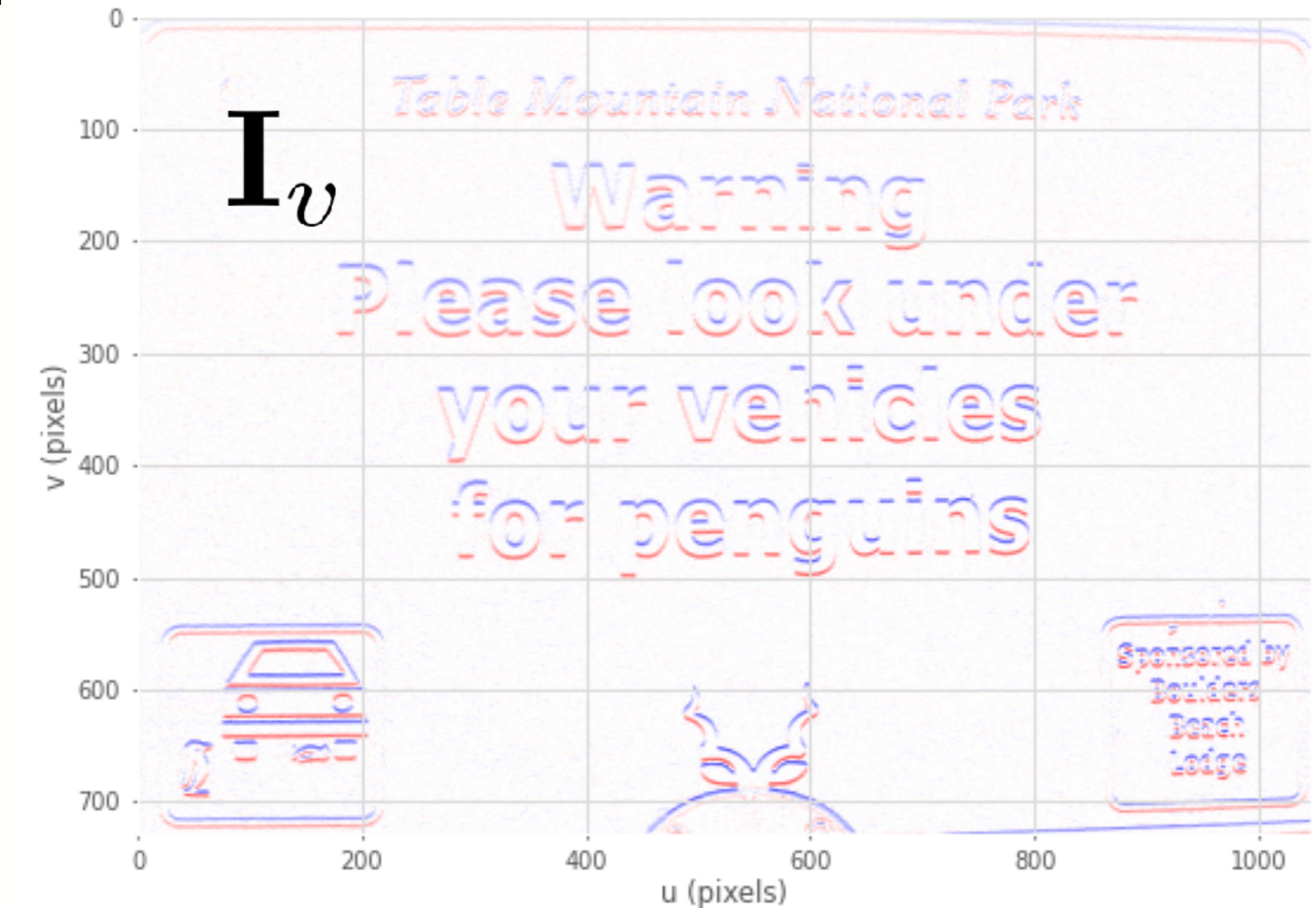
With kind permission of Springer Science+Business Media

$$\mathbf{K} = \frac{1}{2} \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

Vertical gradient image, use the transposed kernel

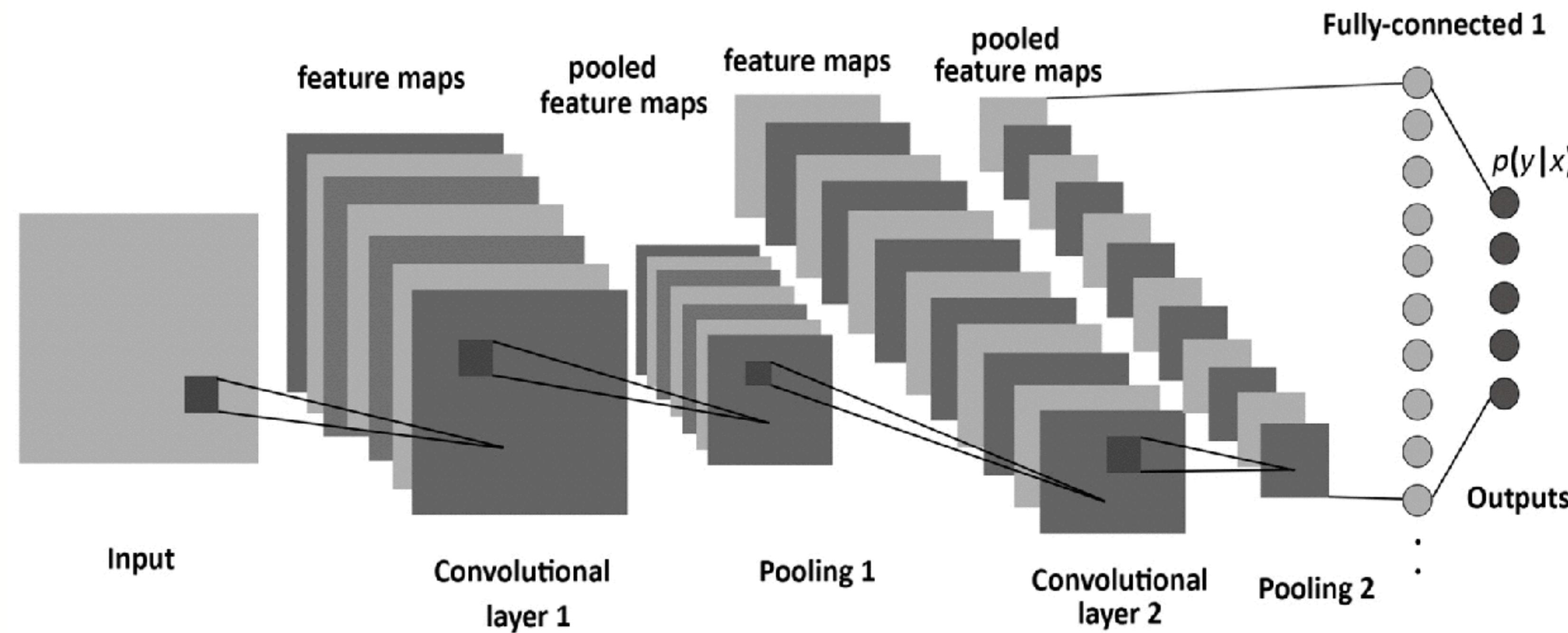


$$K = \frac{1}{2} \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$



```
>>> im.convolve(K.T)
```

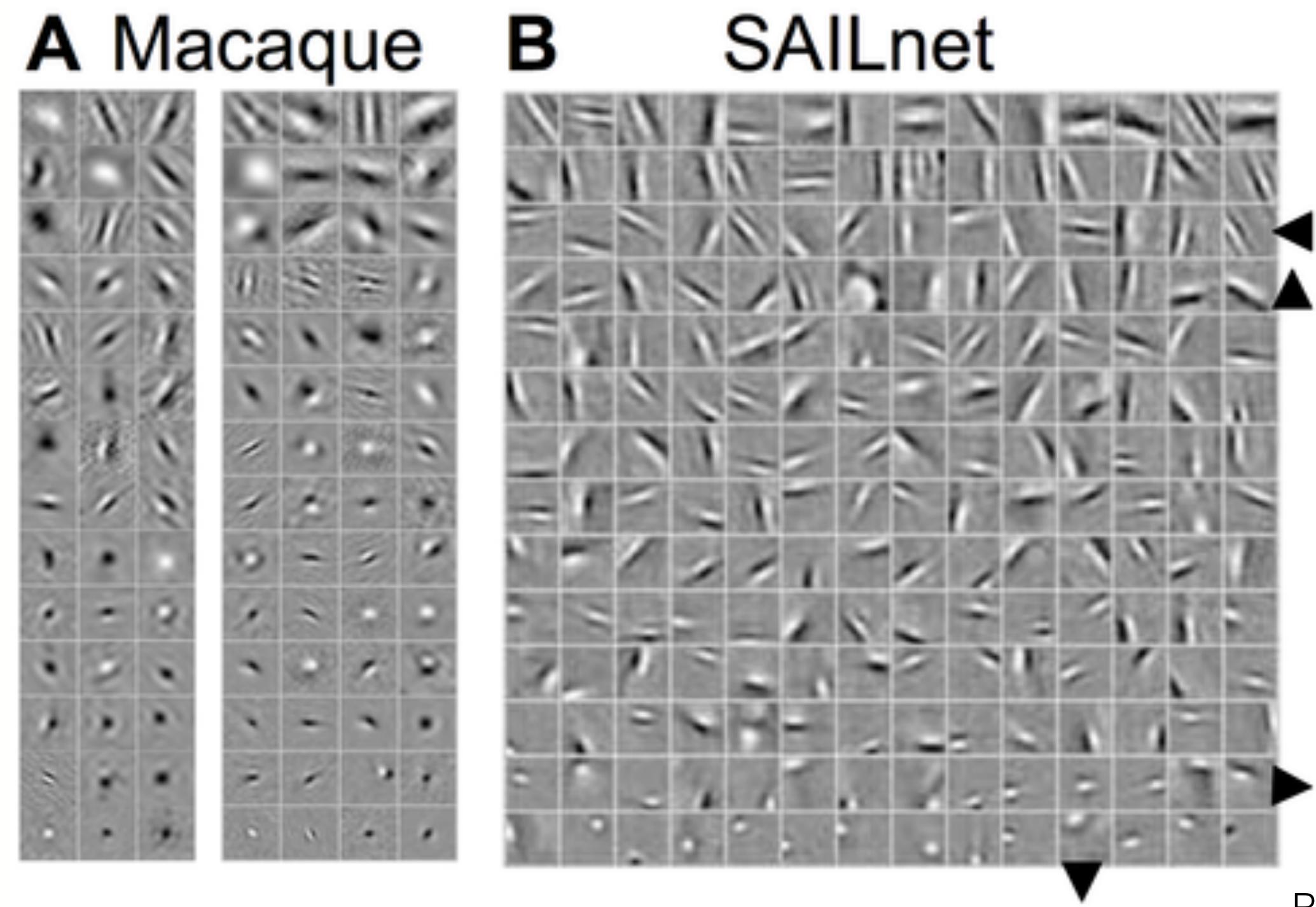
Convolutional neural networks



Entropy 2017, 19(6), 242; <https://doi.org/10.3390/e19060242>

- The input image is convolved with various learnt kernels to form each feature map:
 - a non-linear activation function (often ReLU which is $\max(0, x)$) is applied
- Pooling is average or max over the window
 - the window moves in steps greater than 1 to reduce resolution

Convolutional neural networks



PLoS computational biology. 7. e1002250. 10.1371/journal.pcbi.1002250

- The kernels are tuned to respond to edges (at various angles) and simple light/dark patterns
- The first-level feature maps have filters remarkably similar to those of primate visual cortex (V1)

Image point features

We often want to find the same points in another image



between different views captured
at the same time
across consecutive views

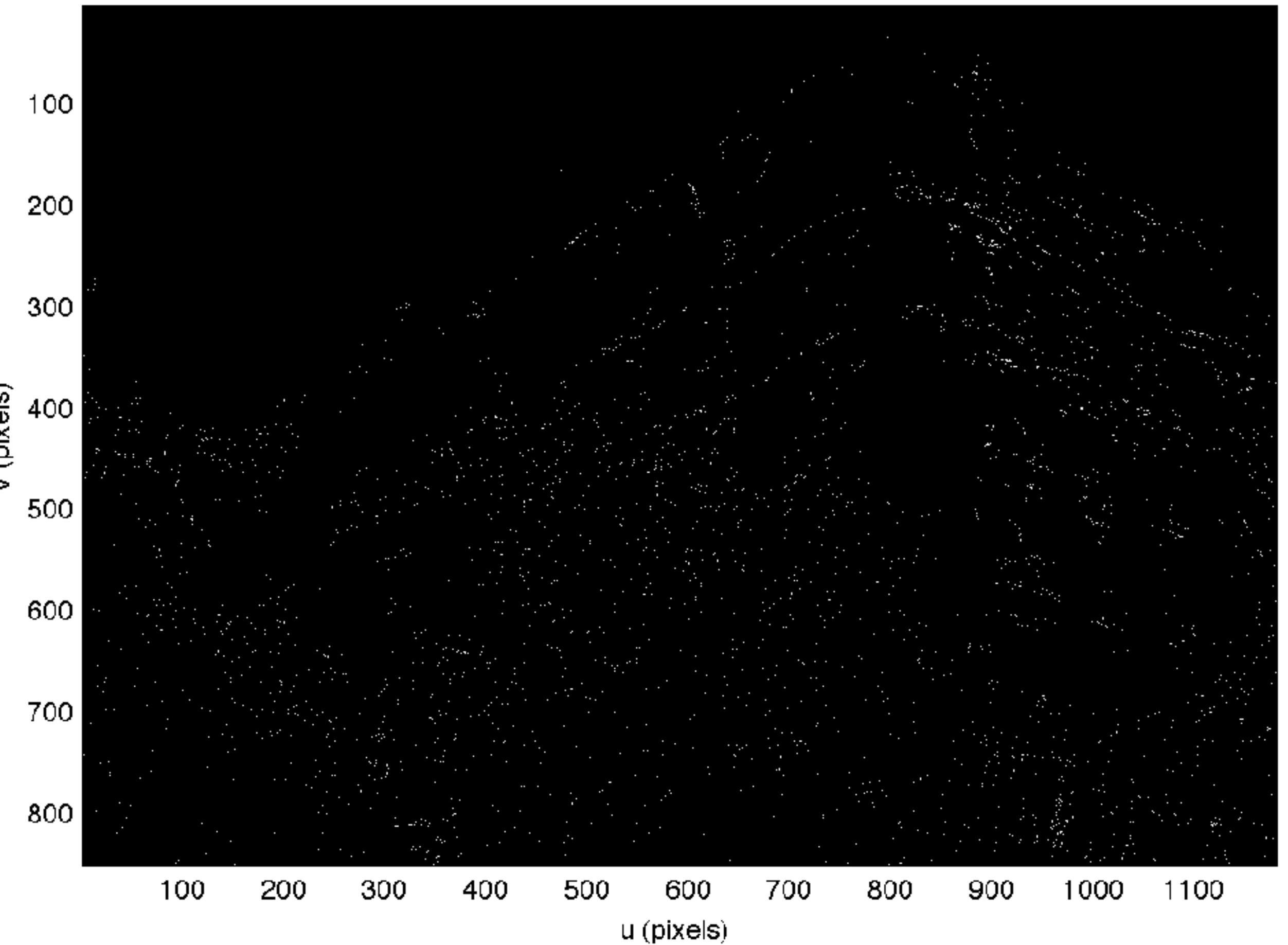
Finding corresponding points



(818,164) = 51



(619,184)



- All the pixels with a value of 51 in second view
 - 6017 points (0.6% of the image)

Finding corresponding points



$(818, 164) = 51$

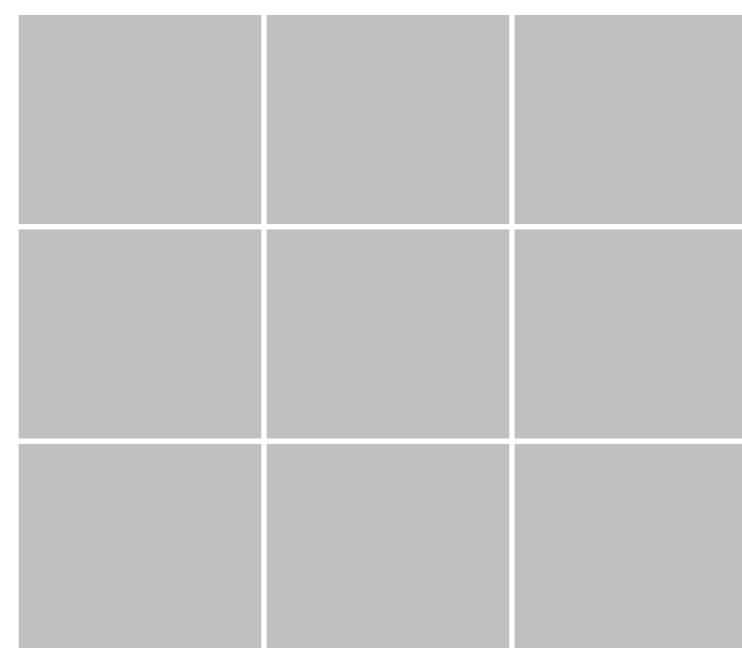
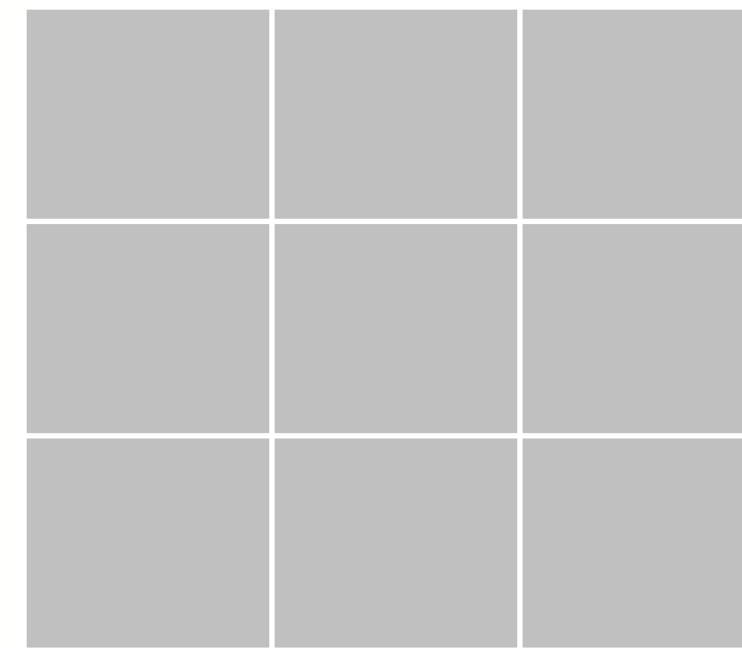
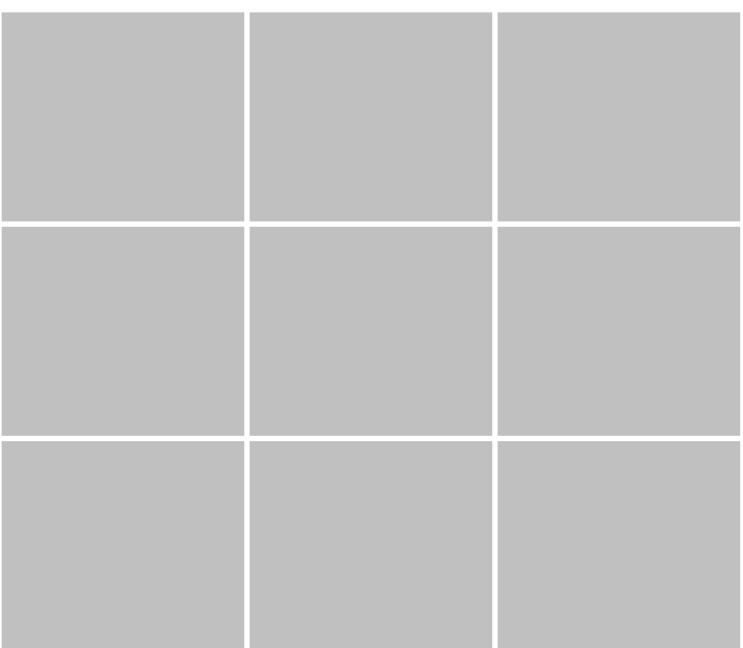
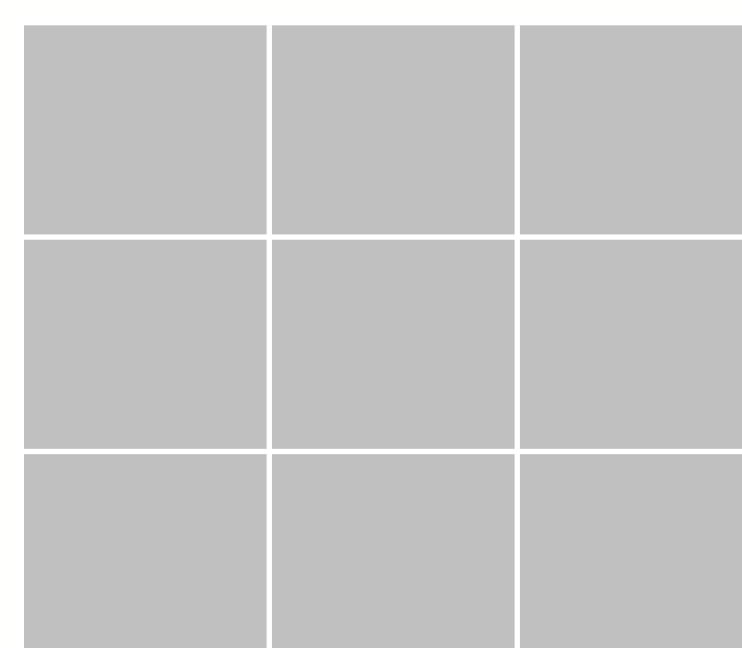
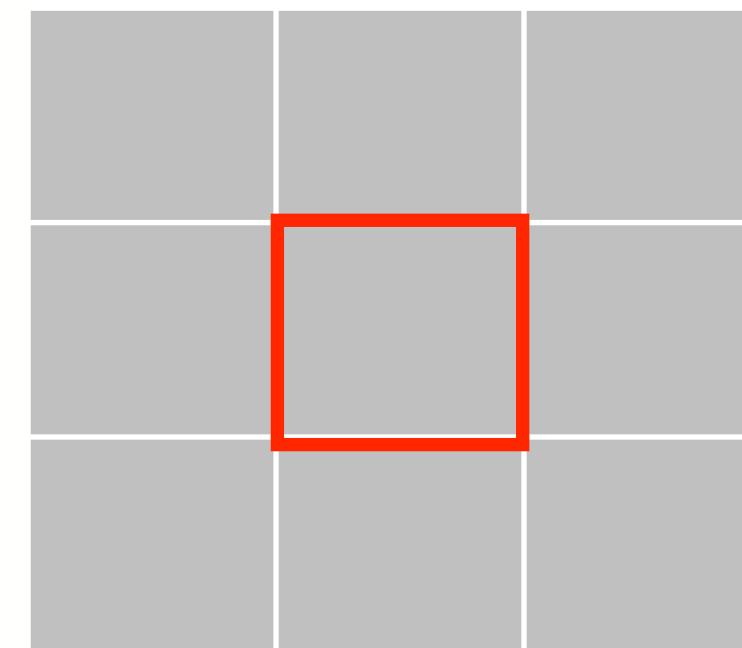
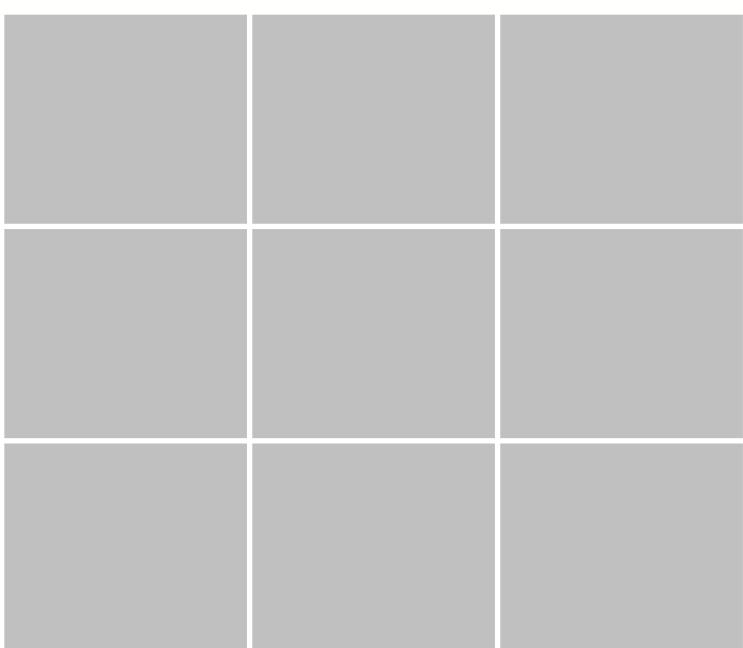
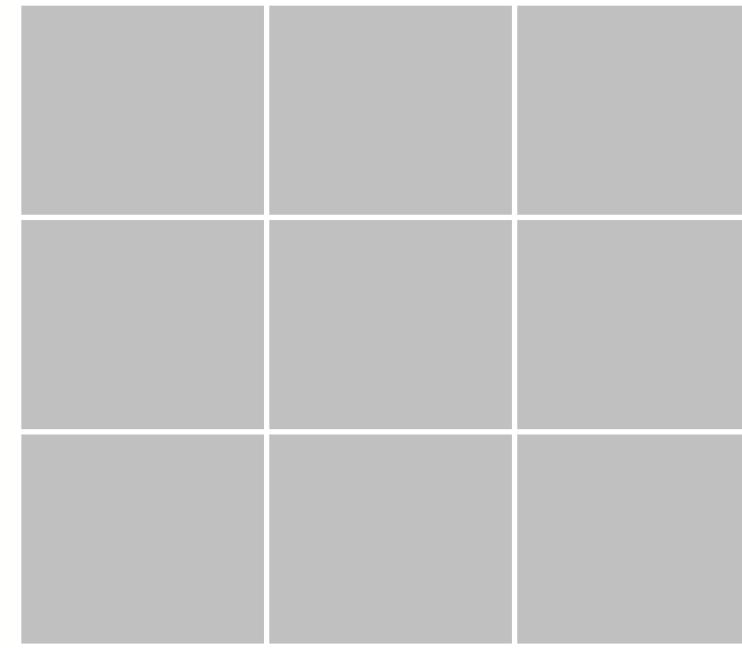
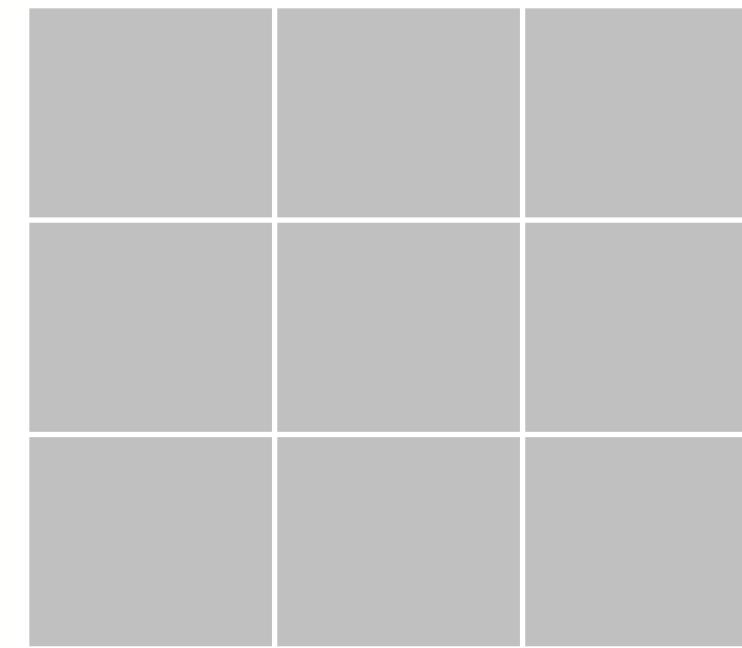
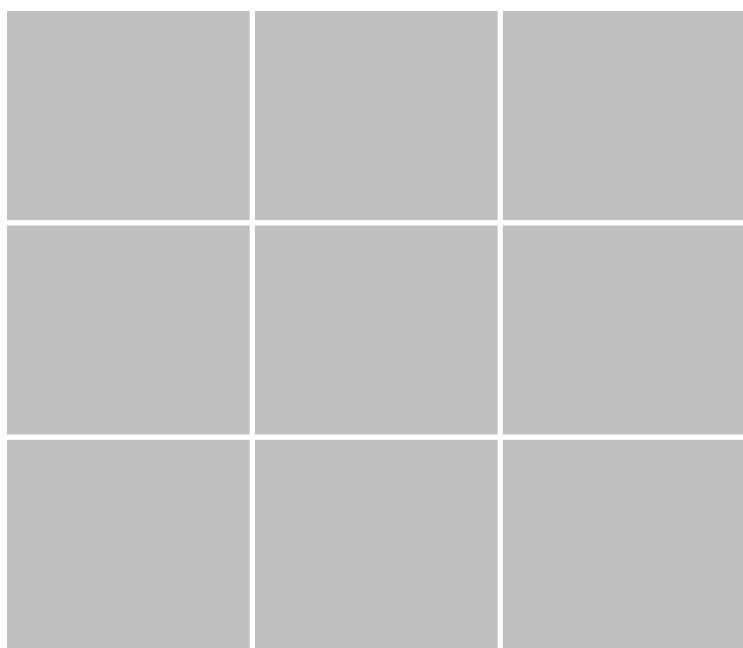
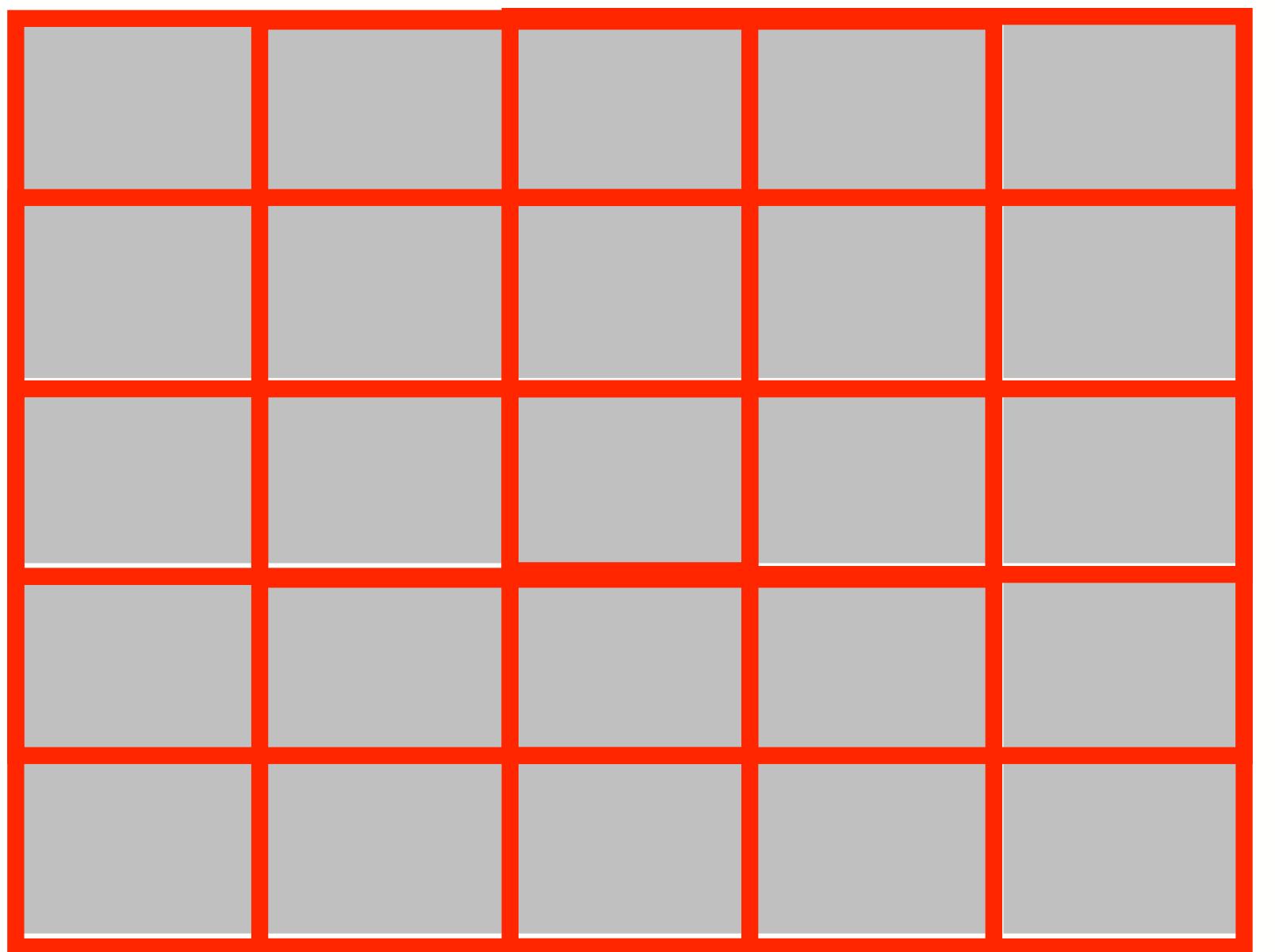


$(619, 184) = 47$

Finding corresponding points



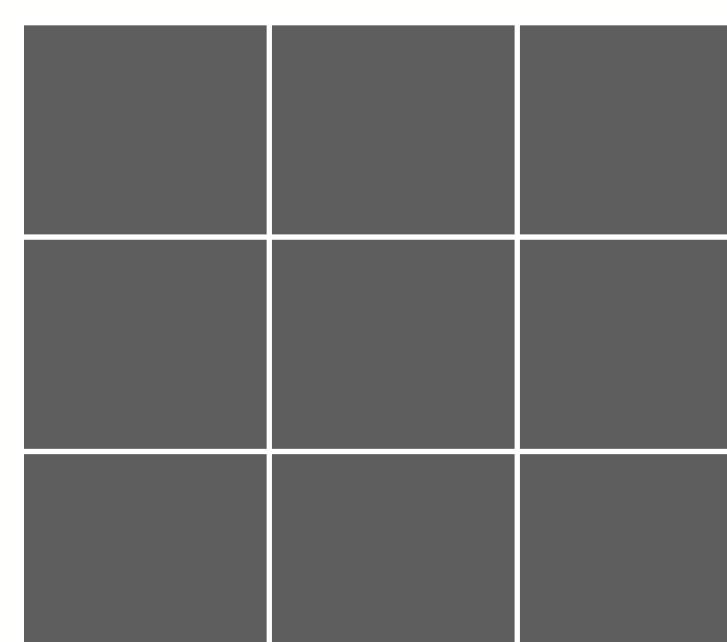
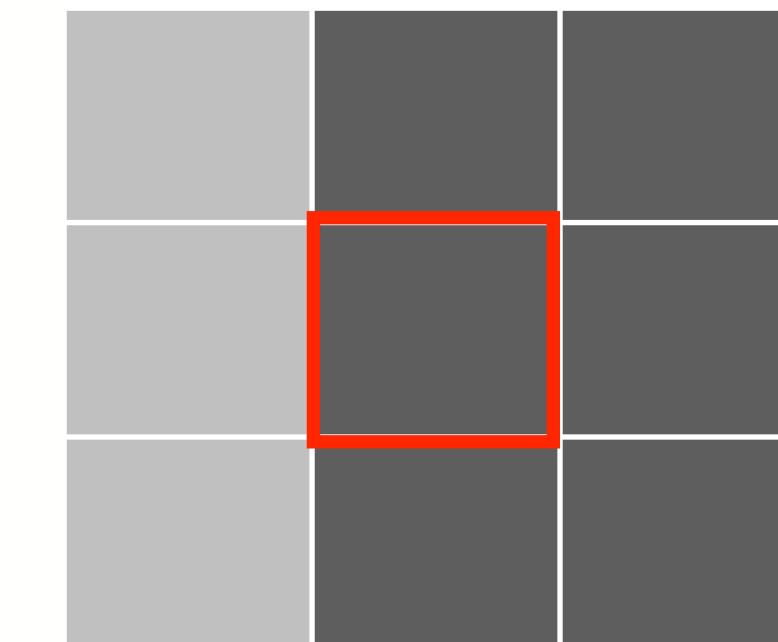
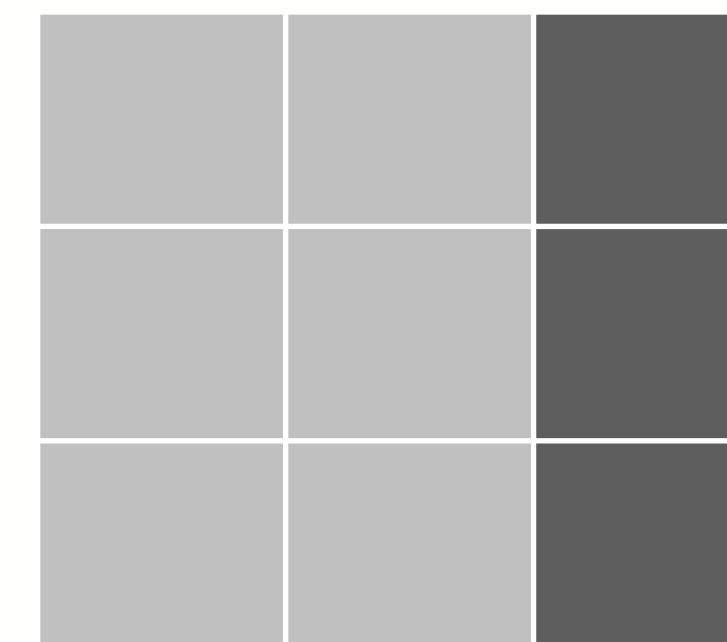
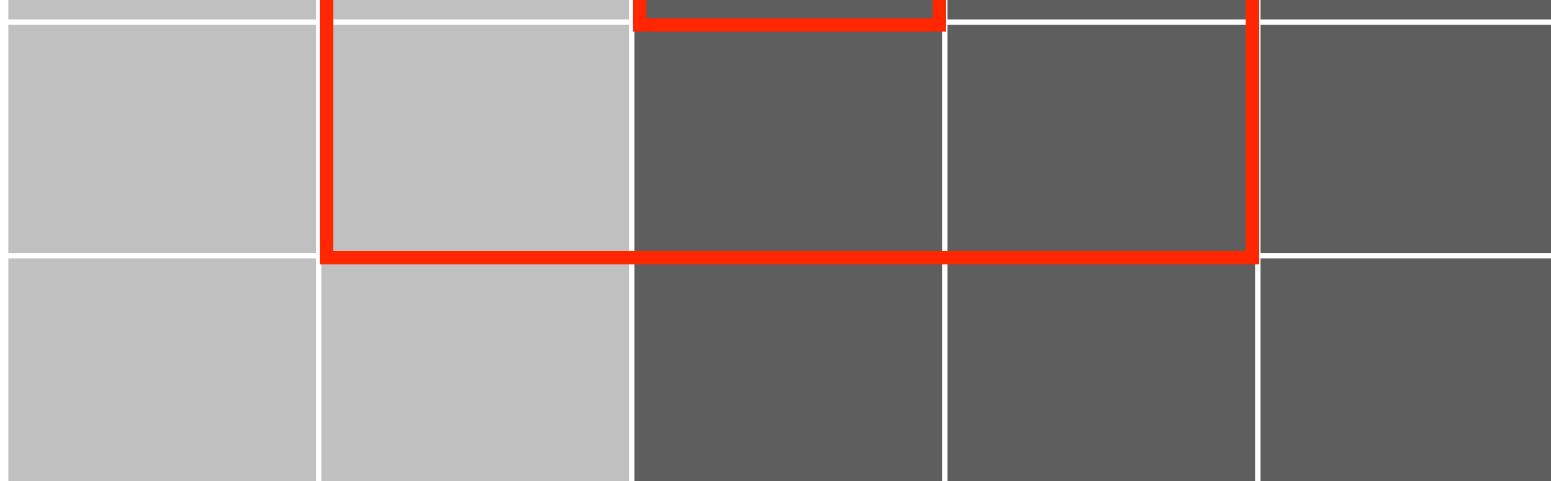
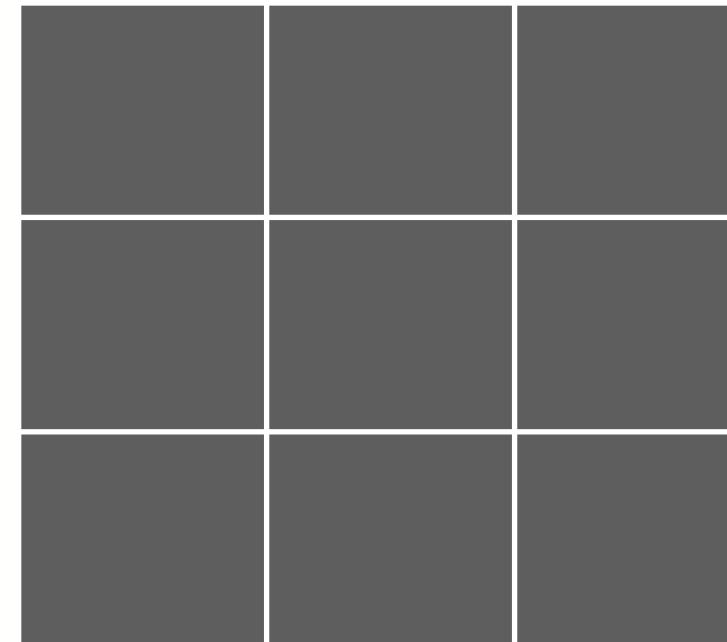
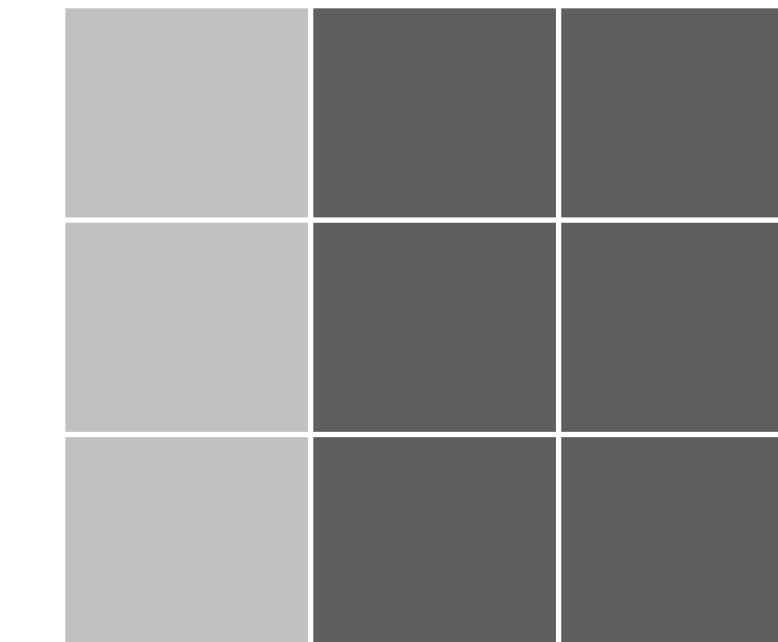
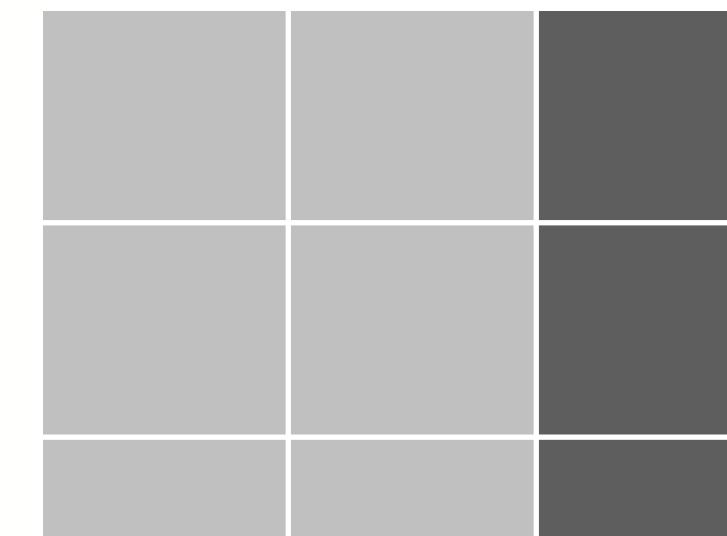
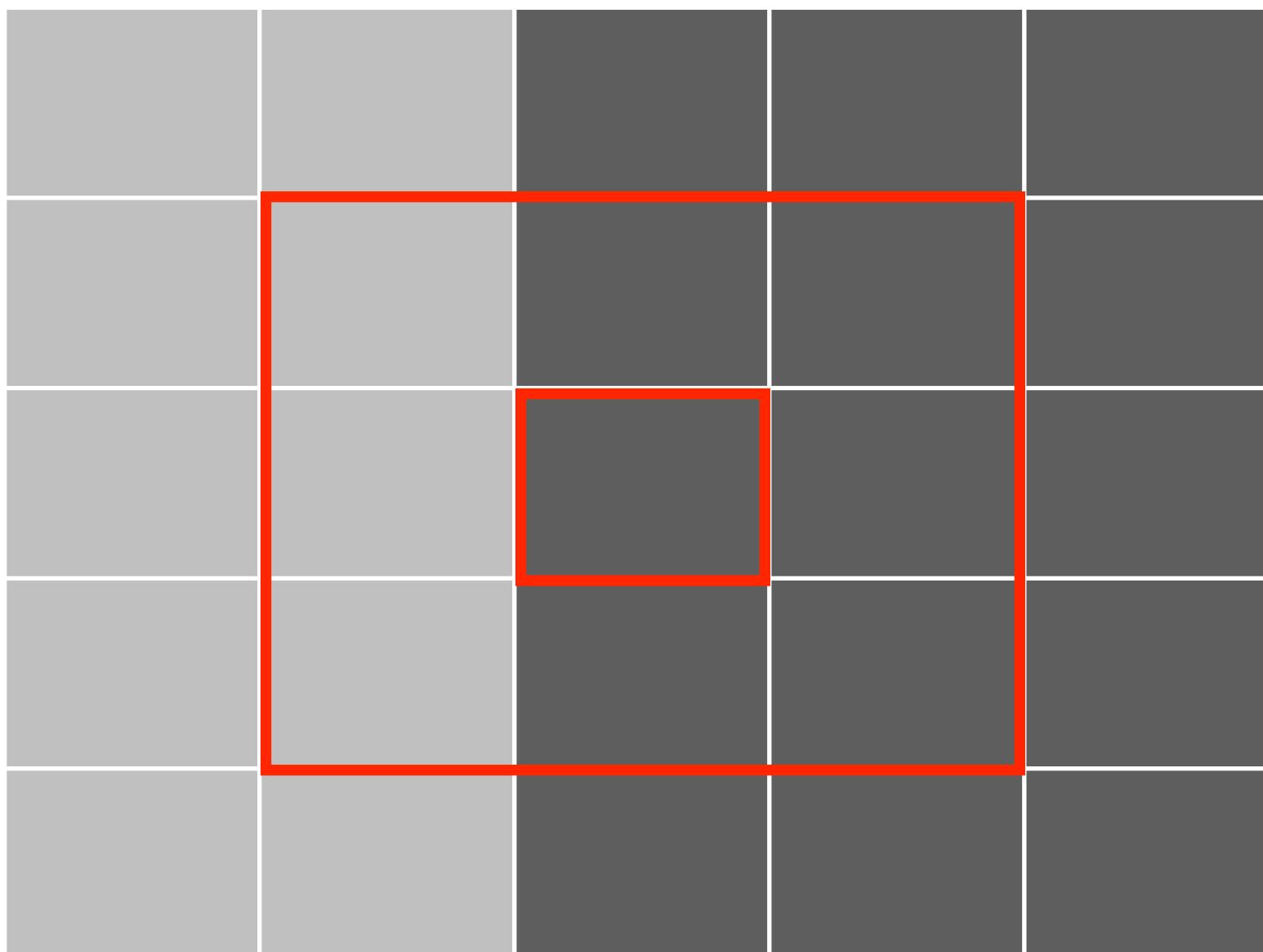
Corner detector intuition



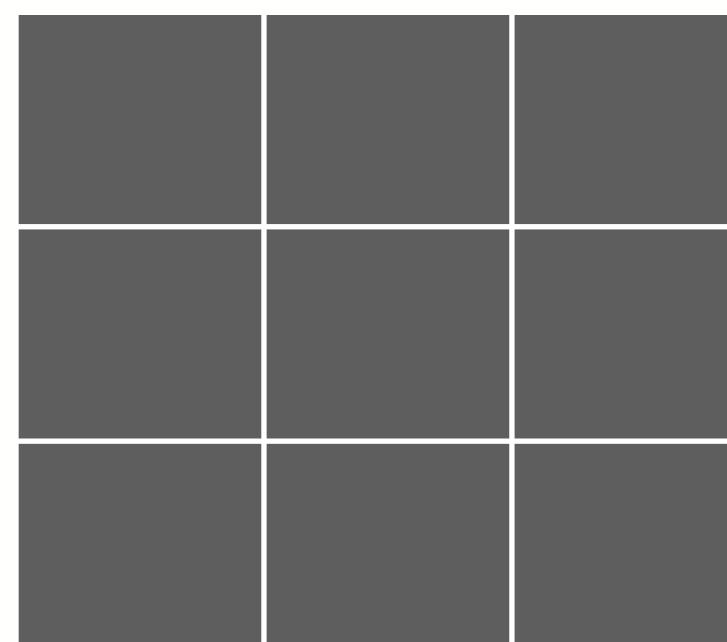
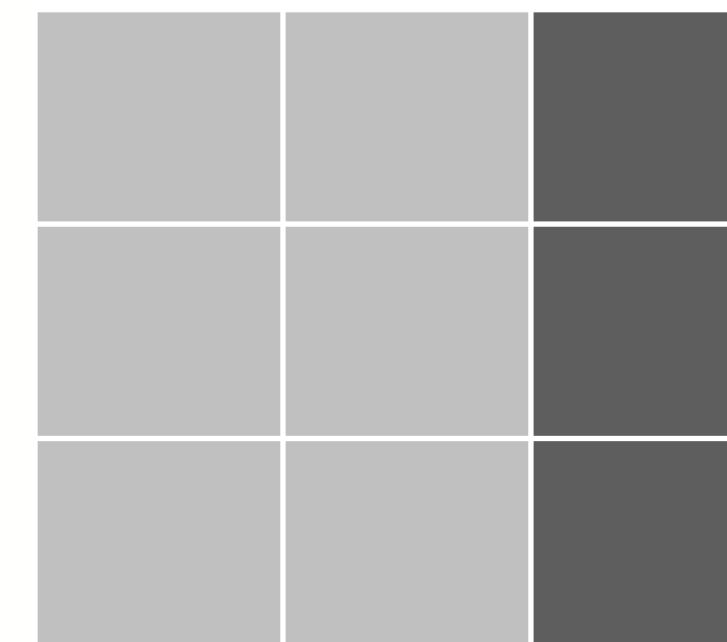
- Similar at 8 other locations
- ▶ not very unique

■ Similar at 8 other locations

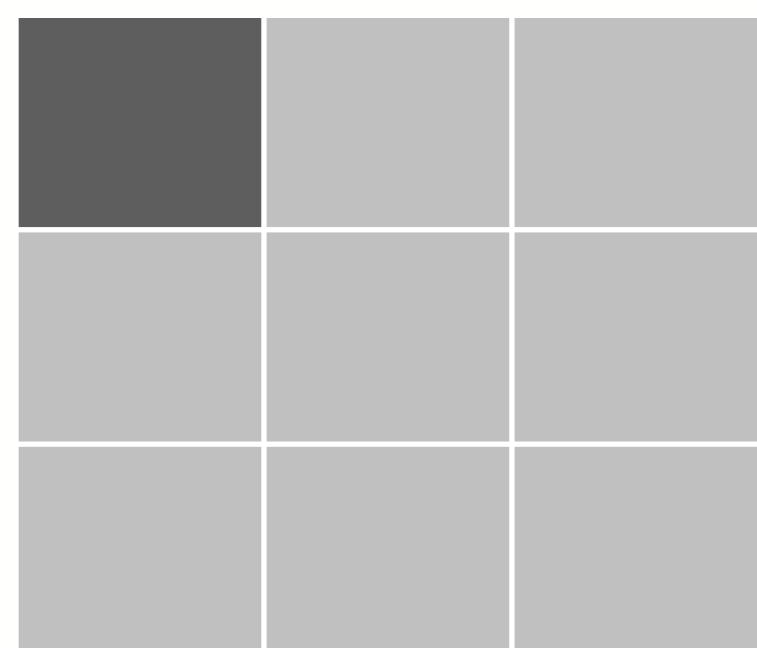
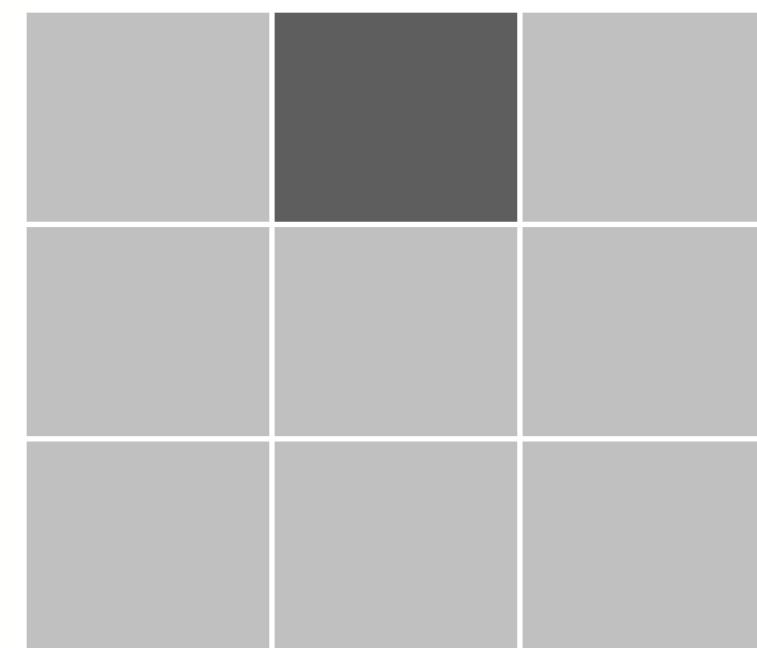
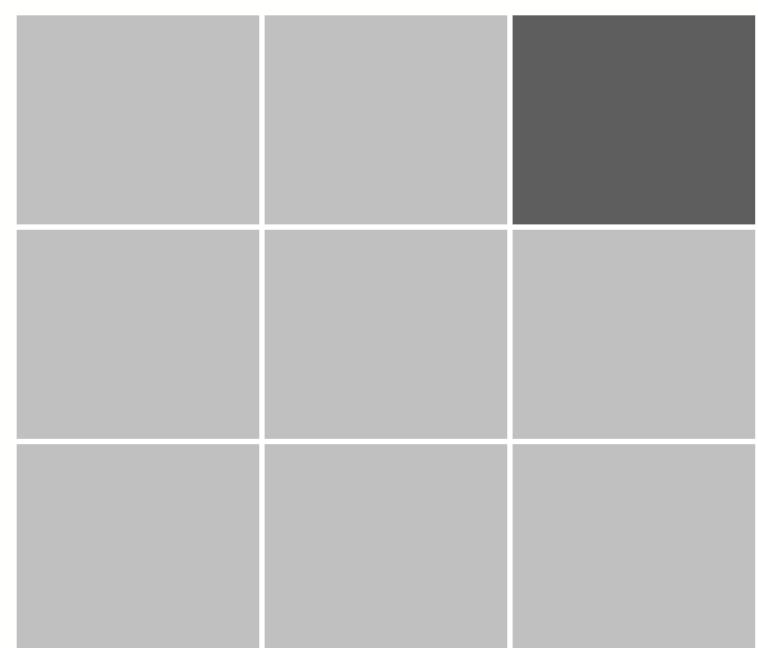
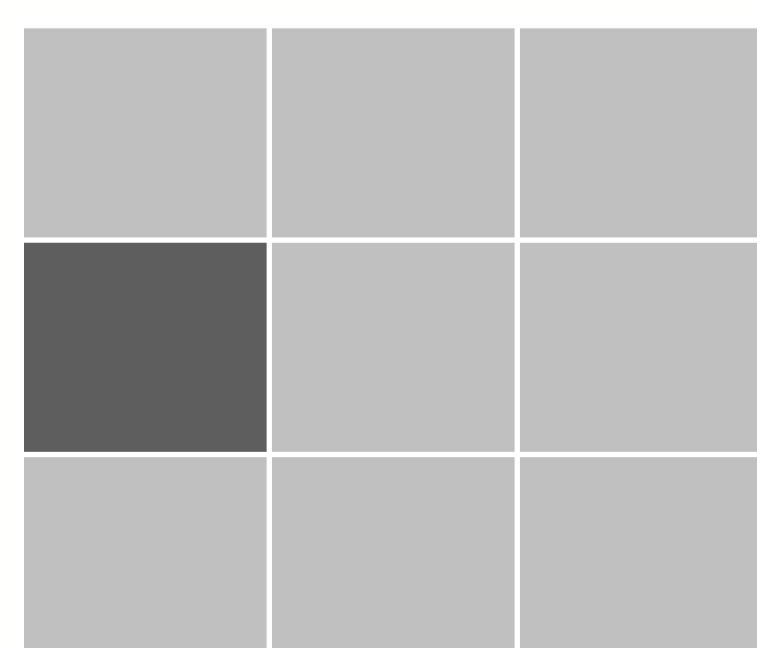
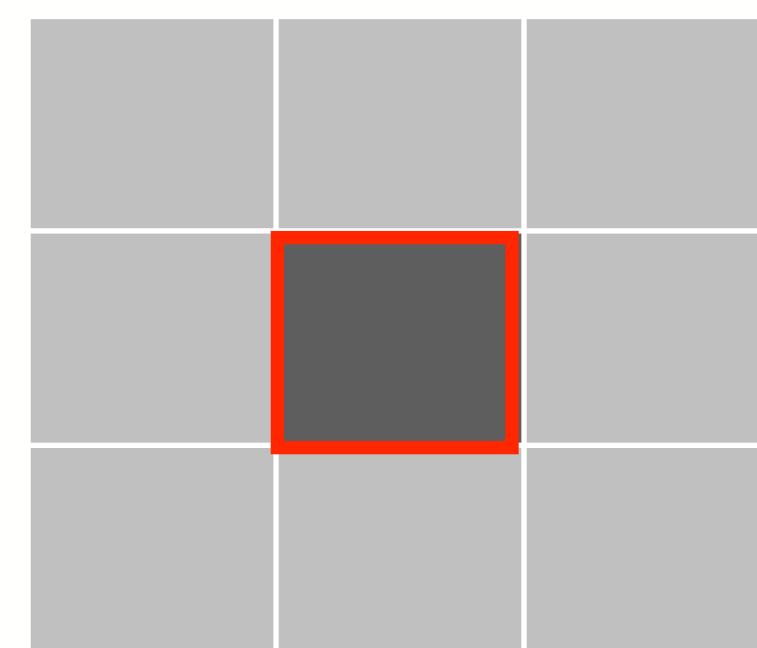
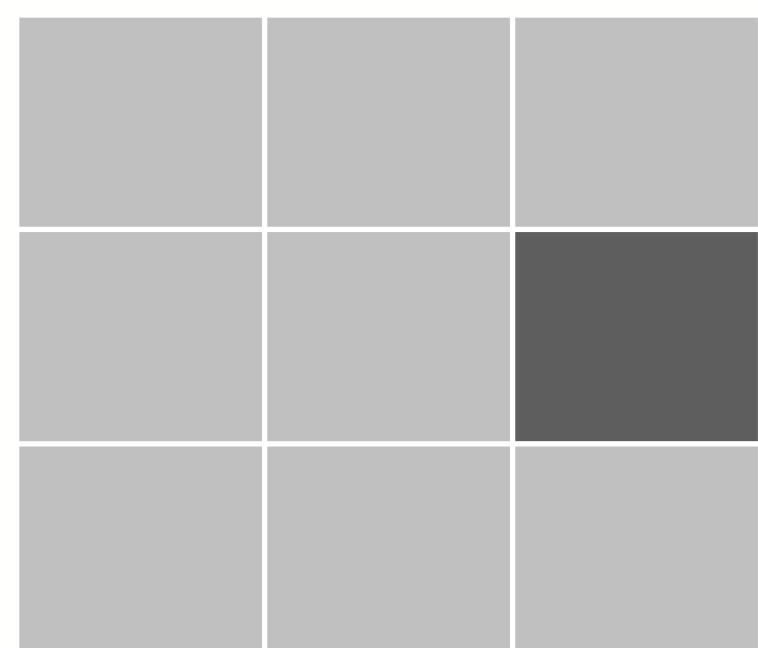
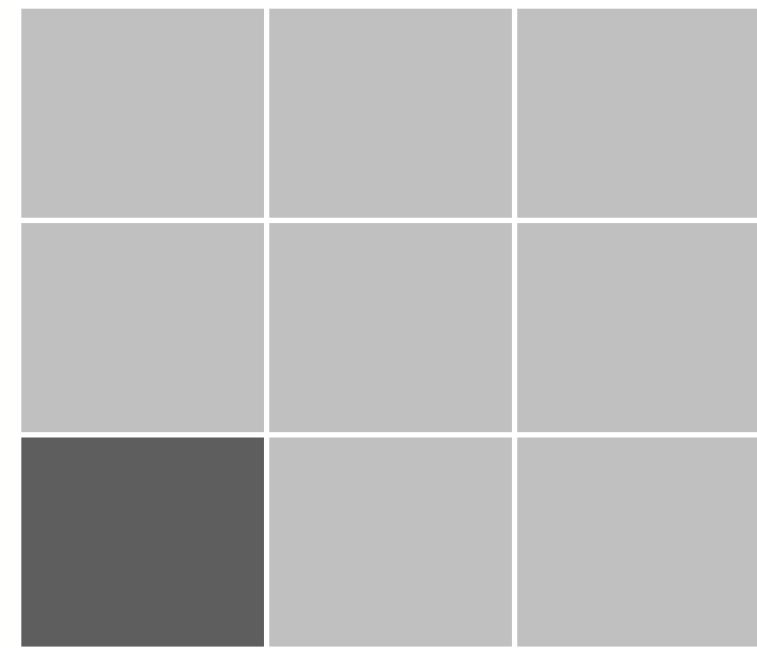
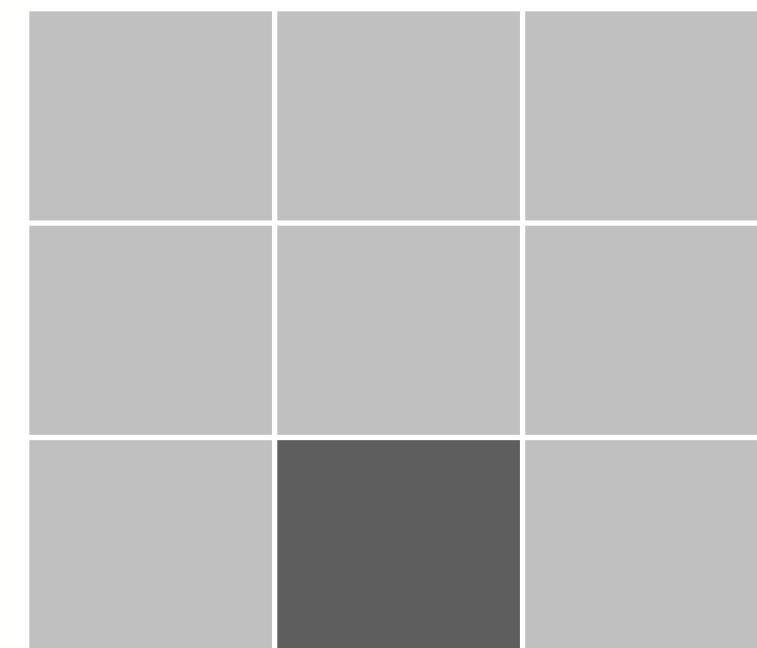
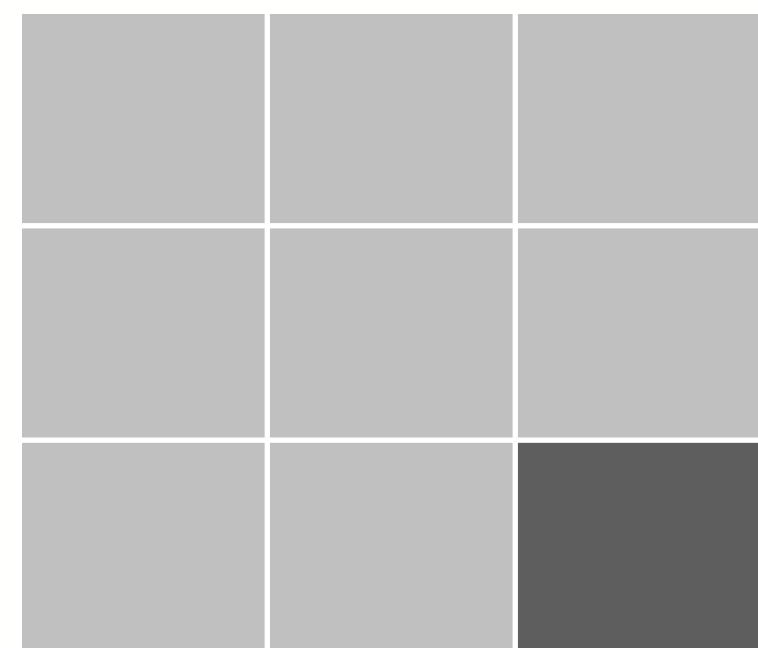
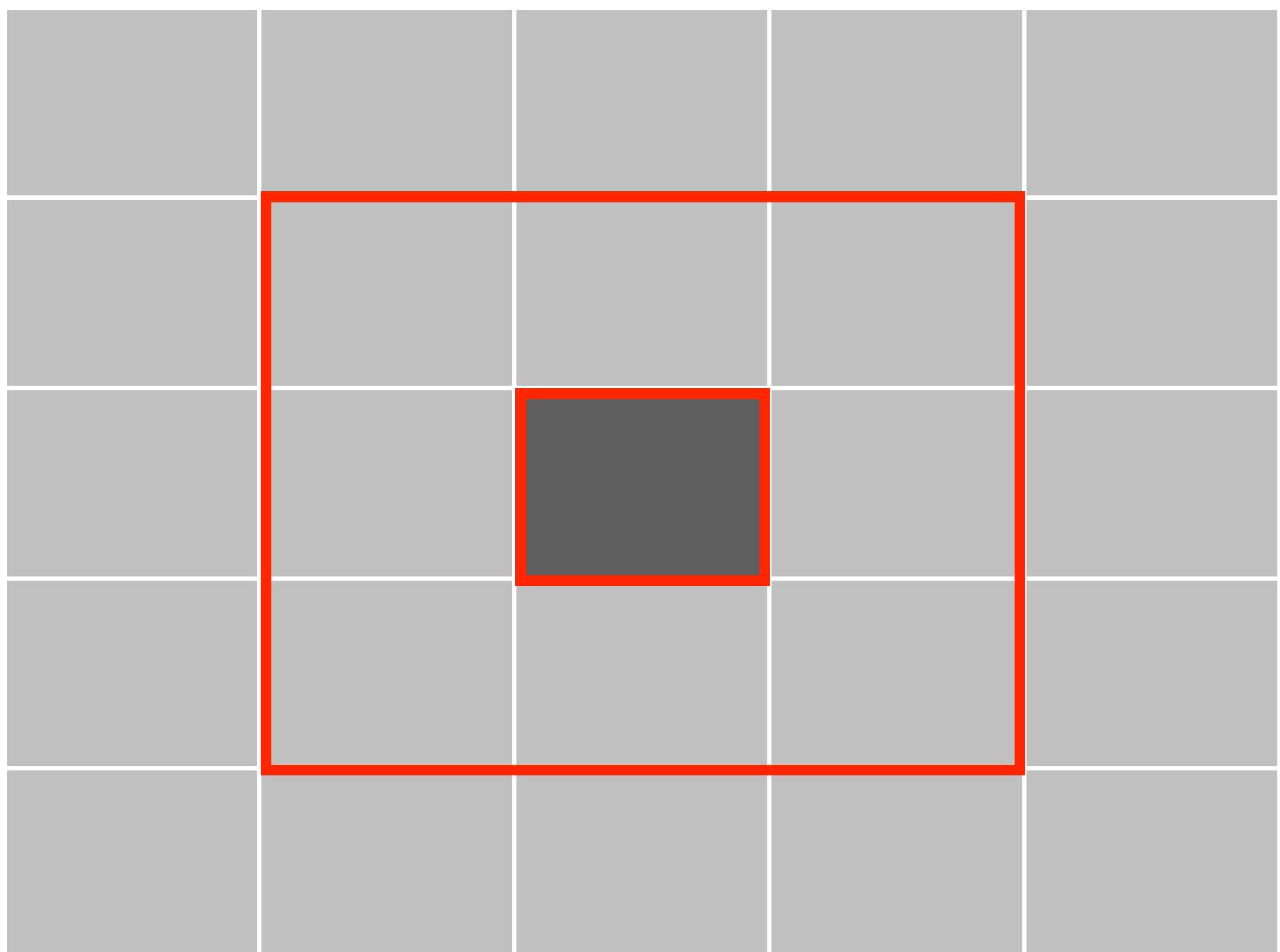
Corner detector intuition



- Similar at 2 other locations
- vertical motion causes no change
- parallel to the line

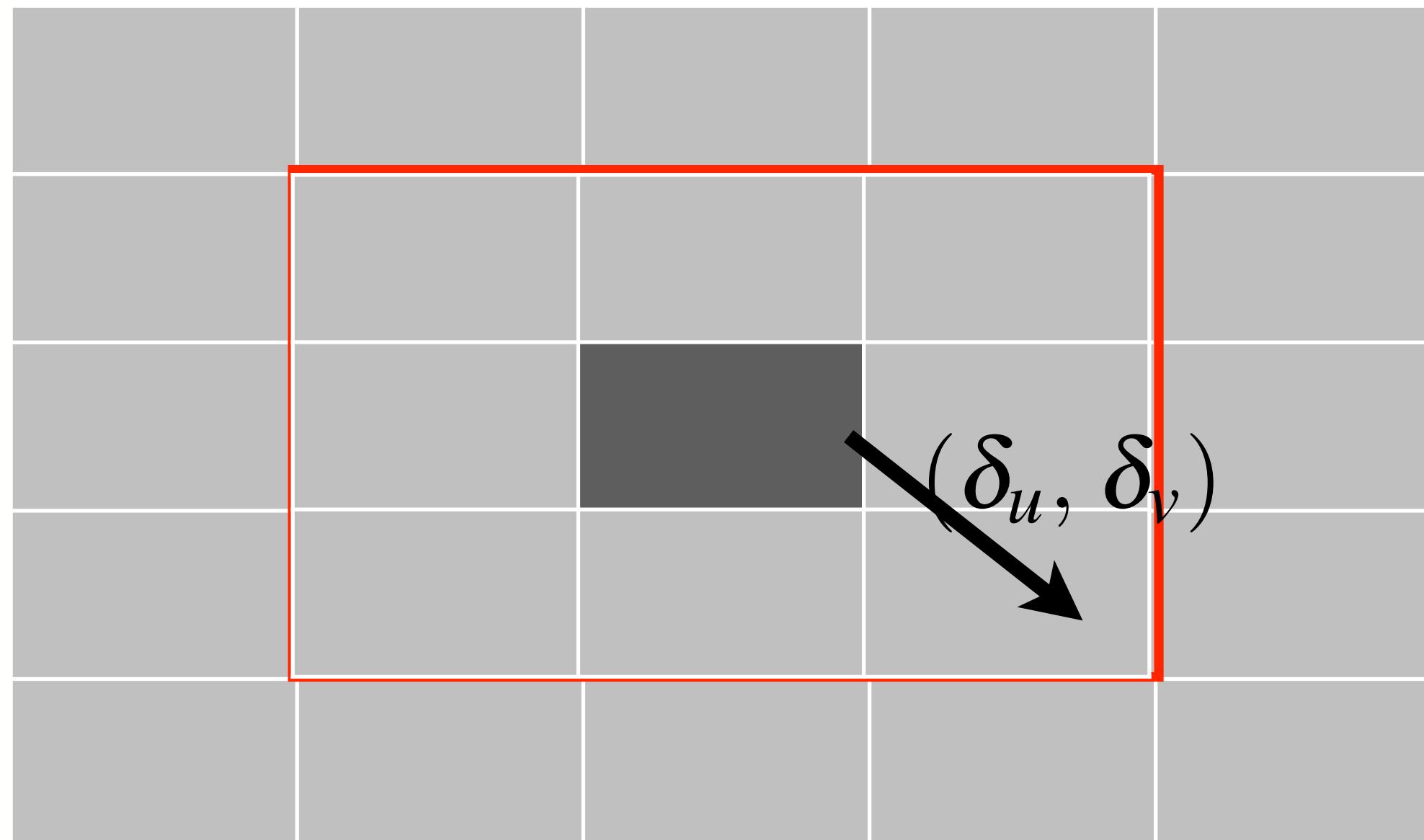


Corner detector intuition



- Similar at no other locations
→ locally unique

...Corner detector intuition



- Similar at 0 other locations (locally unique)

$$s(u, v, \delta_u, \delta_v) = \sum_{(i,j) \in \mathcal{W}} (I[u + \delta_u + i, v + \delta_v + j] - I[u + i, v + j])^2$$

$$C_M(u, v) = \min_{(\delta_u, \delta_v) \in \mathcal{D}} s(u, v, \delta_u, \delta_v)$$

Corner detector

- More general approach
 - ➡ add a Gaussian weighting matrix

$$s(u, v, \delta_u, \delta_v) = \sum_{(i,j) \in \mathcal{W}} W[i, j] \left(\underbrace{I[u + \delta_u + i, v + \delta_v + j]} - I[u + i, v + j] \right)^2$$

$$s(u, v, \delta_u, \delta_v) = (\delta_u \quad \delta_v) A \begin{pmatrix} \delta_u \\ \delta_v \end{pmatrix}$$

structure tensor $A = \begin{pmatrix} G(\sigma_I) \otimes I_u^2 & G(\sigma_I) \otimes I_u I_v \\ G(\sigma_I) \otimes I_u I_v & G(\sigma_I) \otimes I_v^2 \end{pmatrix}$

based on gradients
so intensity difference
is eliminated

- The eigenvalues of \mathbf{A} contain important information

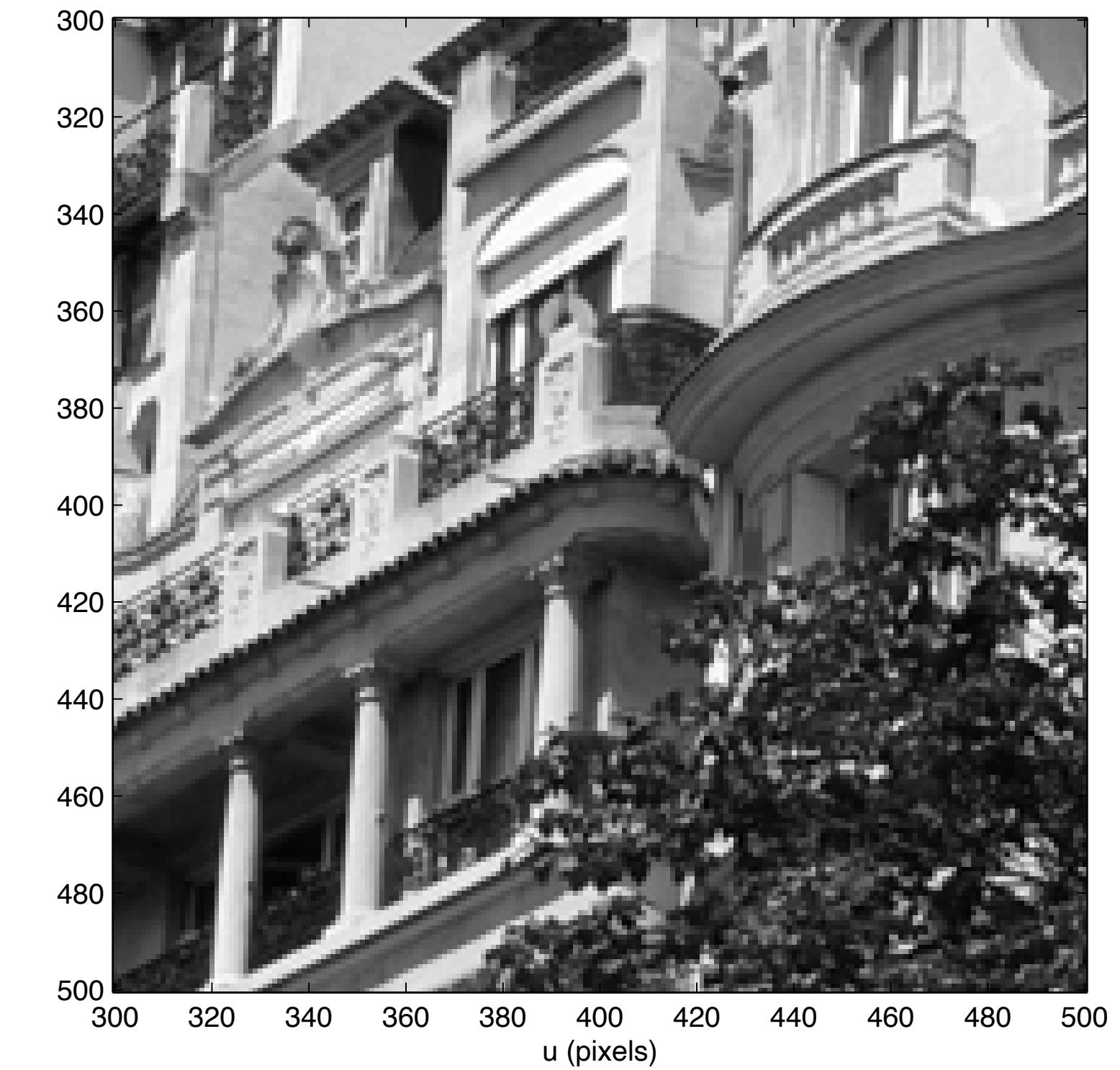
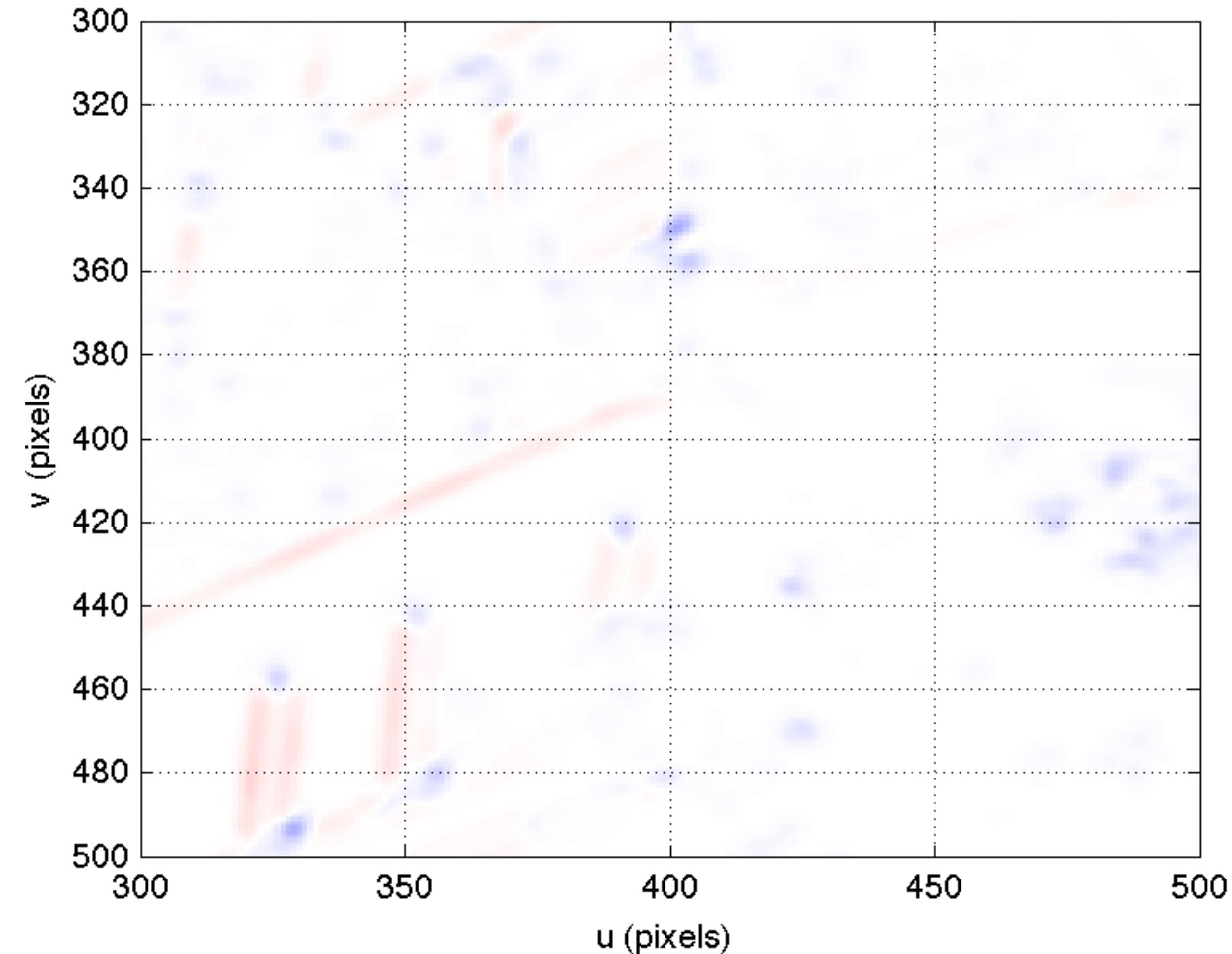
	λ_2 small	λ_2 large
λ_1 small	constant	edge
λ_1 large	edge	peak

Shi-Tomasi detector $C_{ST}(u, v) = \min(\lambda_1, \lambda_2)$

Harris detector

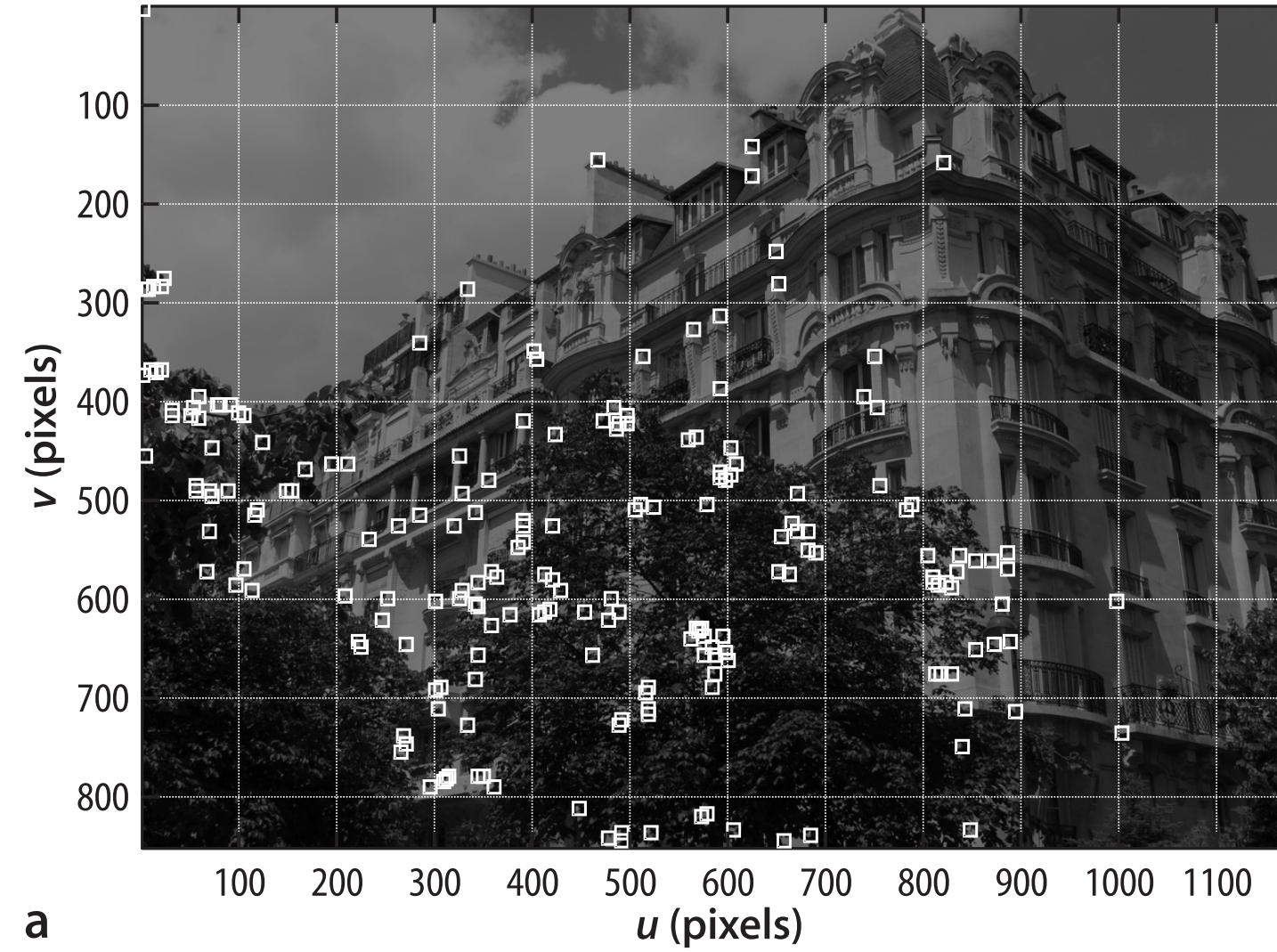
$$C_H(u, v) = \det(A) - k \text{tr}(A)$$

Harris corner value

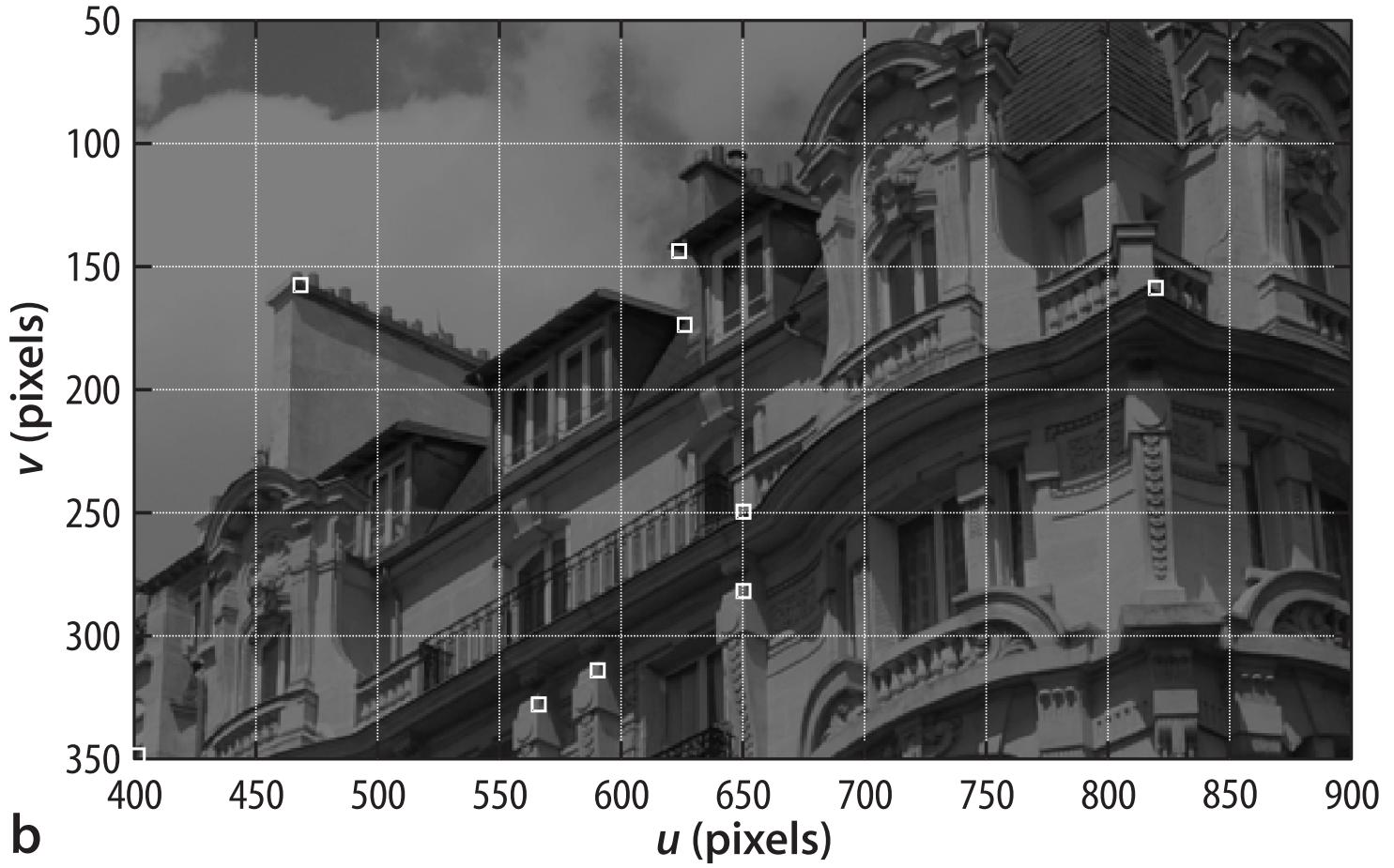


blue is positive,
red is negative

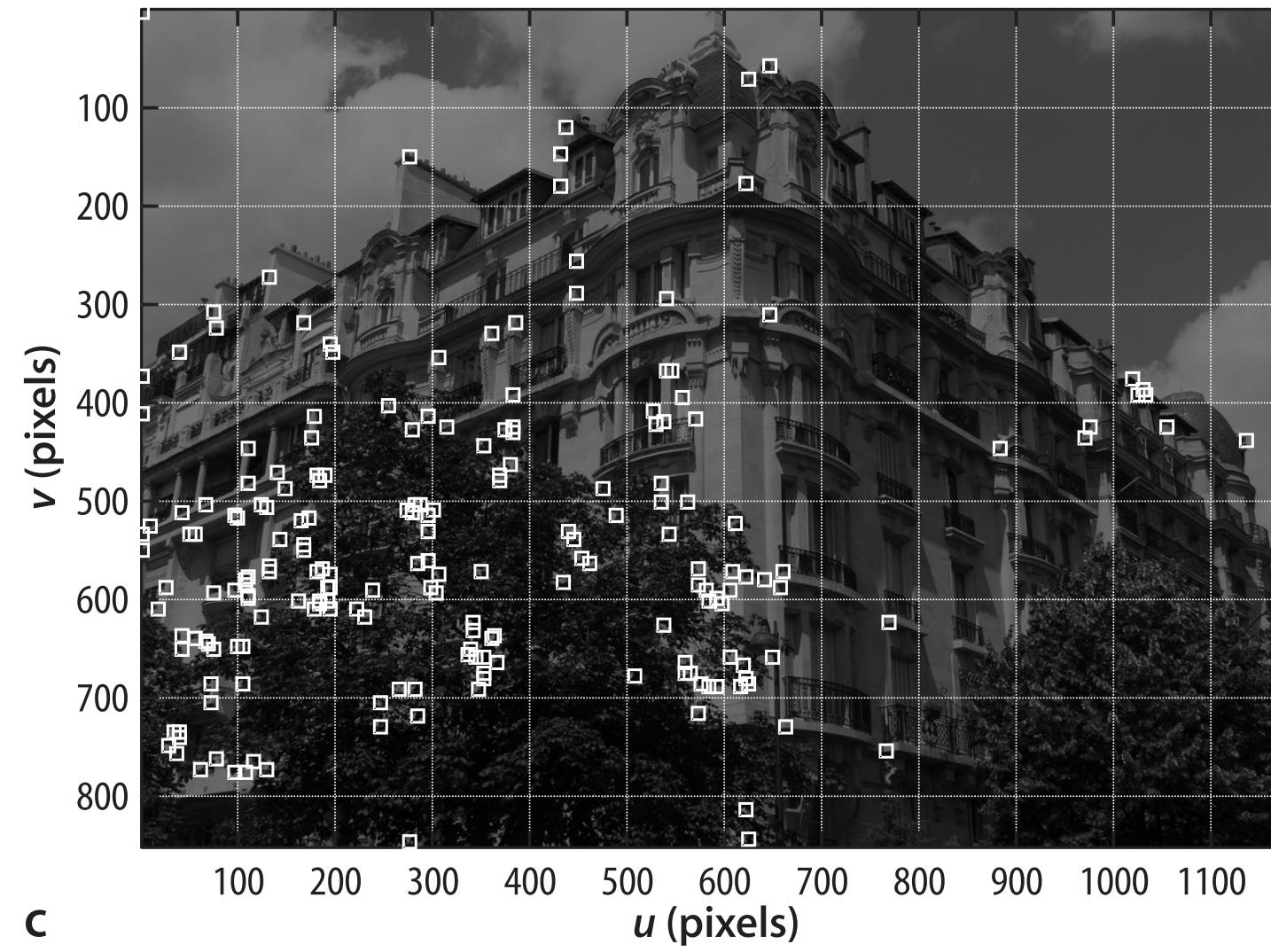
Harris features for 2 views



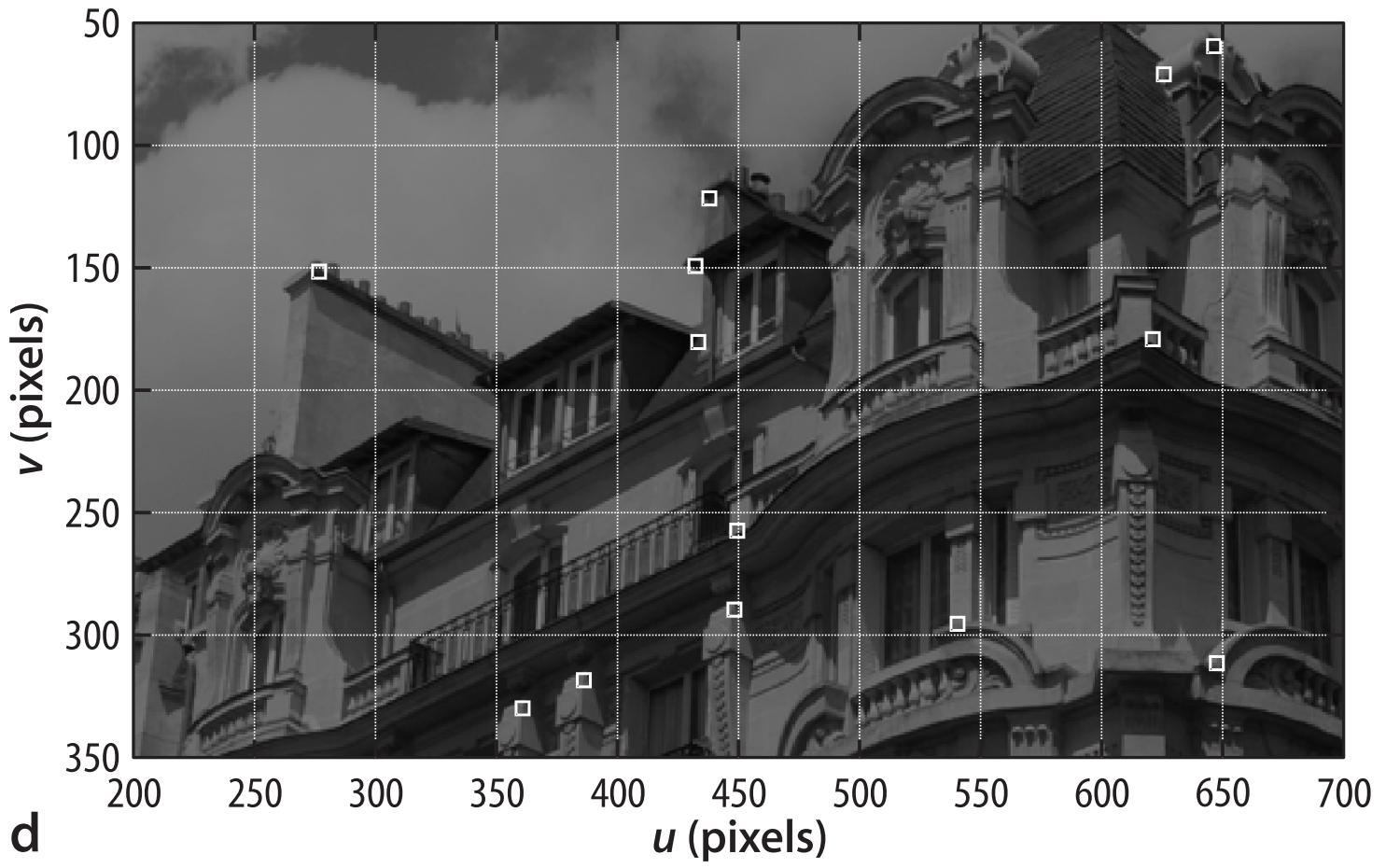
a



b



c

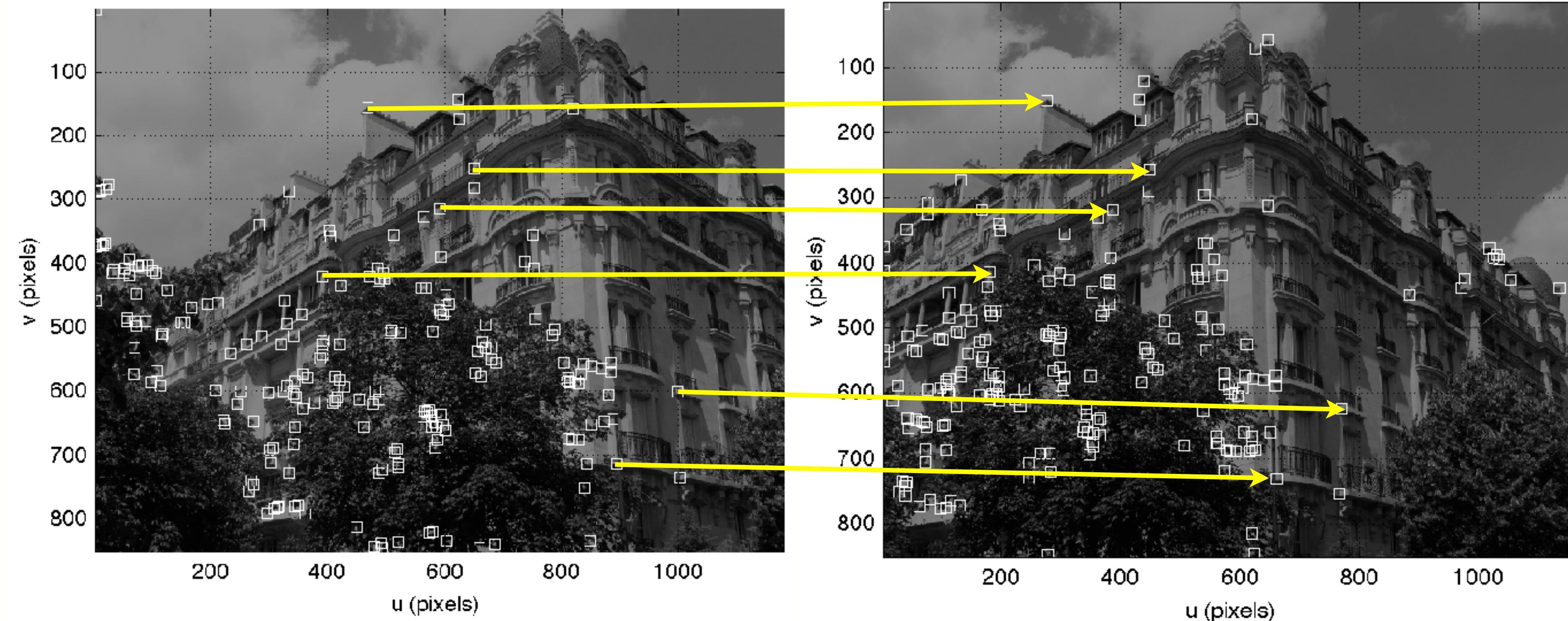


d

- Some features are clearly the same point in the scene (good)
- Unduly excited about the trees at the lower left (not good)

Comparing features

- We've found the coordinates of some interesting points in each image:
 $\{^1\mathbf{p}_i, i \in 1 \cdots N_1\}$ $\{^2\mathbf{p}_j, j \in 1 \cdots N_2\}$
- Now we need to determine the correspondence, which $^1\mathbf{p}_i \leftrightarrow ^2\mathbf{p}_j$



Feature matching

- We use a $W \times W$ window of pixels centred on each corner point (W is odd)
- We use a similarity metric to compare the windows
- For each point ${}^1\mathbf{p}_i$ we test the similarity against all the points $\{{}^2\mathbf{p}_j, j \in 1 \cdots N_2\}$ in the other image
- This is an $N_1 \times N_2$ search problem

Name	Measure	Toolbox function
SAD	$s = \sum_{(u,v) \in \mathbf{I}} \mathbf{I}_1[u, v] - \mathbf{I}_2[u, v] $	<code>sad()</code>
SSD	$s = \sum_{(u,v) \in \mathbf{I}} (\mathbf{I}_1[u, v] - \mathbf{I}_2[u, v])^2$	<code>ssd()</code>
ZNCC	$s = \frac{\sum_{(u,v) \in \mathbf{I}} \mathbf{I}_1[u, v] \cdot \mathbf{I}_2[u, v]}{\sqrt{\sum_{(u,v) \in \mathbf{I}} \mathbf{I}_1^2[u, v] \cdot \sum_{(u,v) \in \mathbf{I}} \mathbf{I}_2^2[u, v]}}$	<code>zncc()</code>

Tracking across frames



```
>> t = Tracker(im, c)
200 continuing tracks, 41 new tracks, 0 retired
241 continuing tracks, 46 new tracks, 0 retired
287 continuing tracks, 34 new tracks, 0 retired
.
.
```

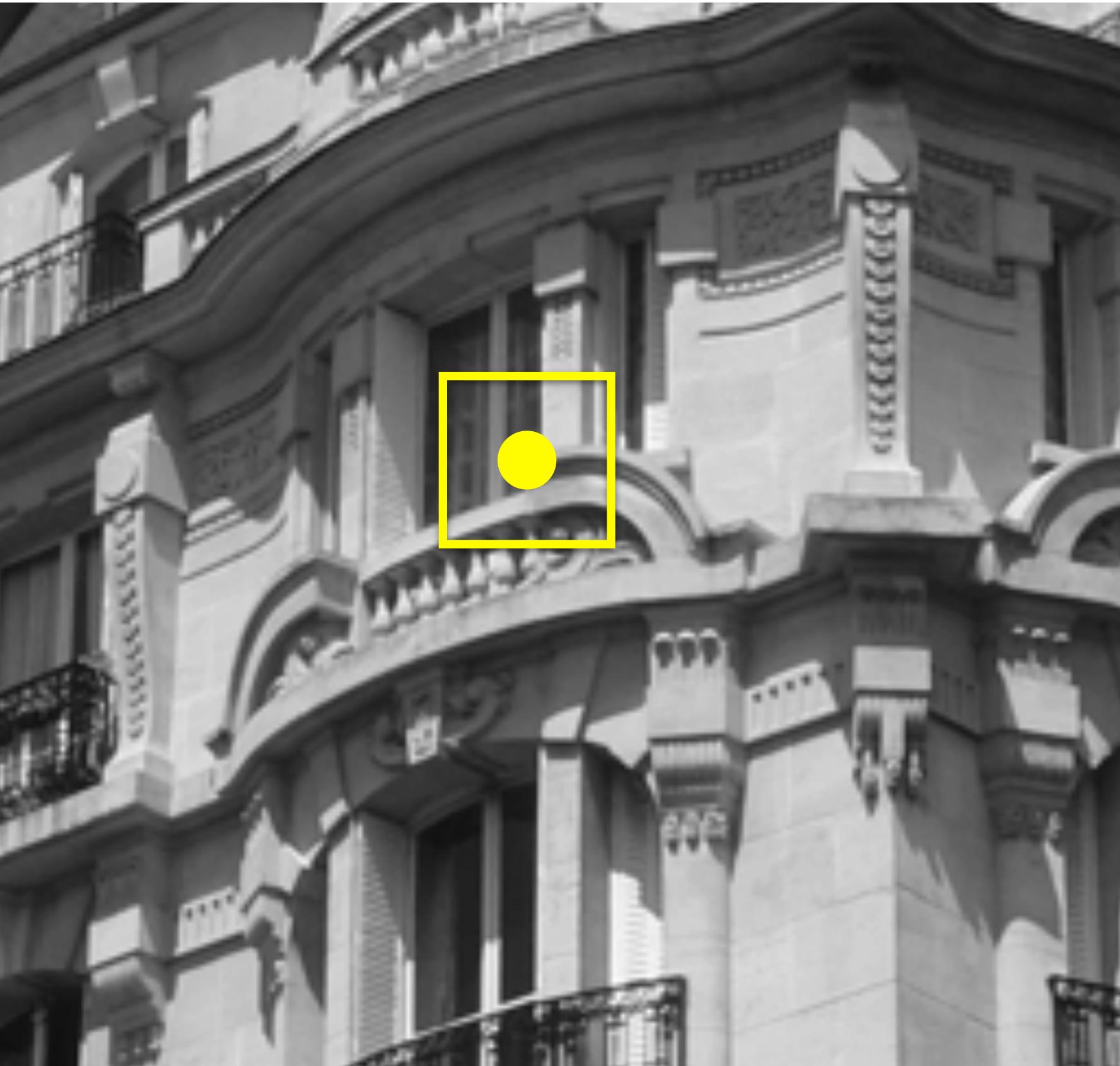
Harris feature recap

- Concise:
 - ➡ hundreds of features instead of millions of pixels
 - ➡ “*a description that is useful for the viewer and not cluttered with irrelevant information*” (Marr)
- Finds points that are distinct and easily located in a different view of the same scene
- Computationally efficient (good for real-time tracking)

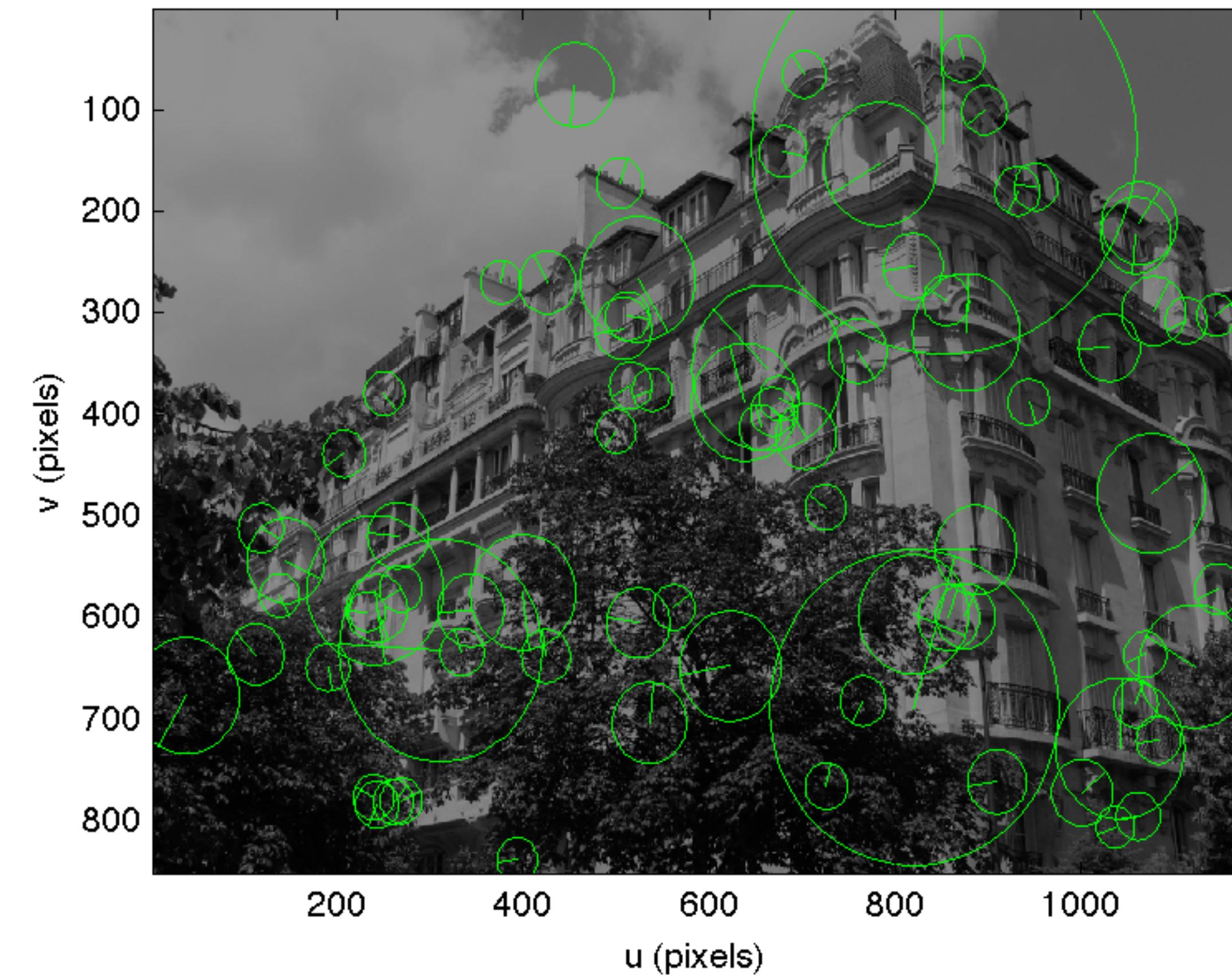
The problem of scale



The problem of orientation



Scale Invariant Feature Transform



SIFT detector (Lowe, 2004)

Harris vs SIFT

