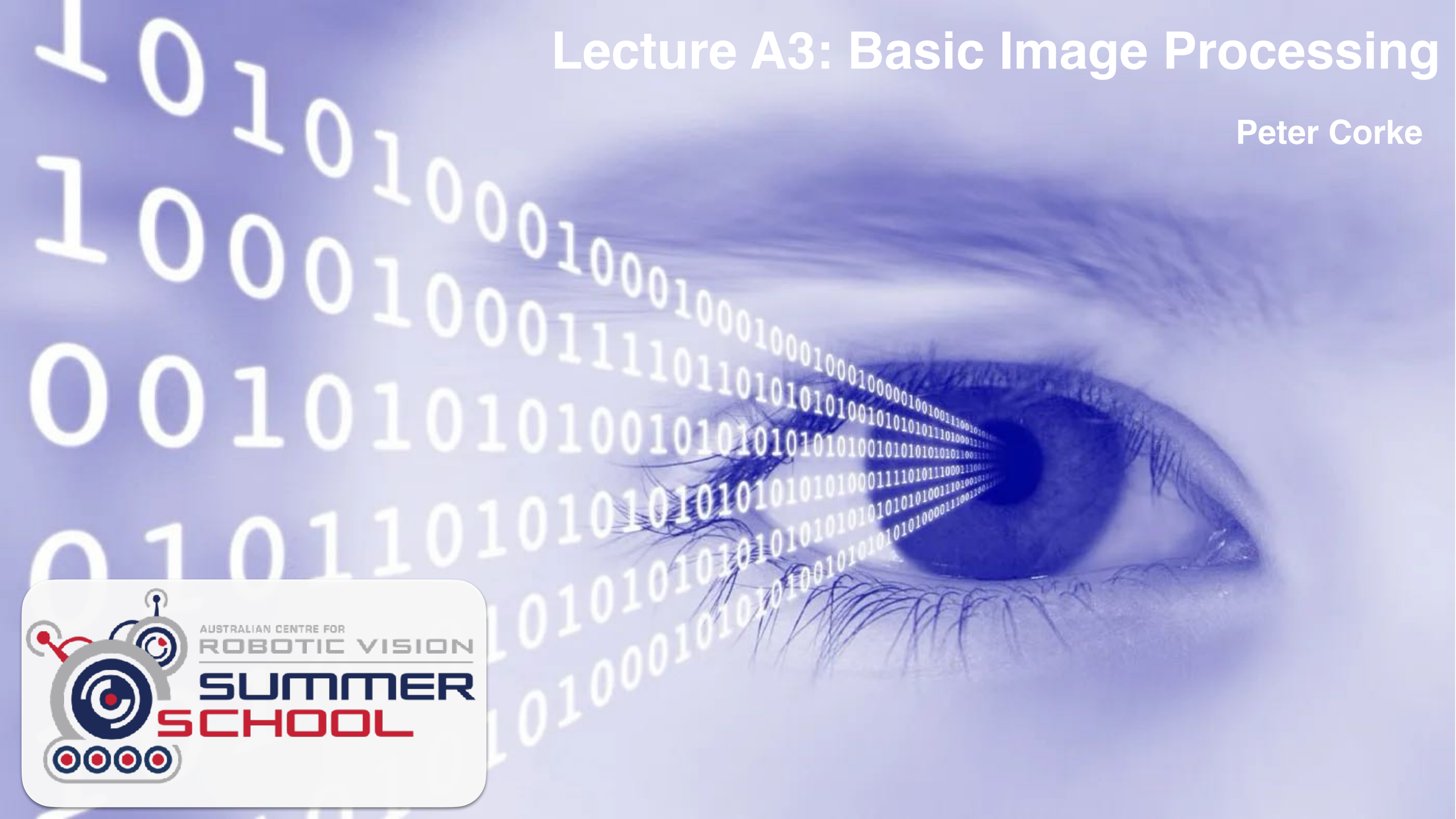
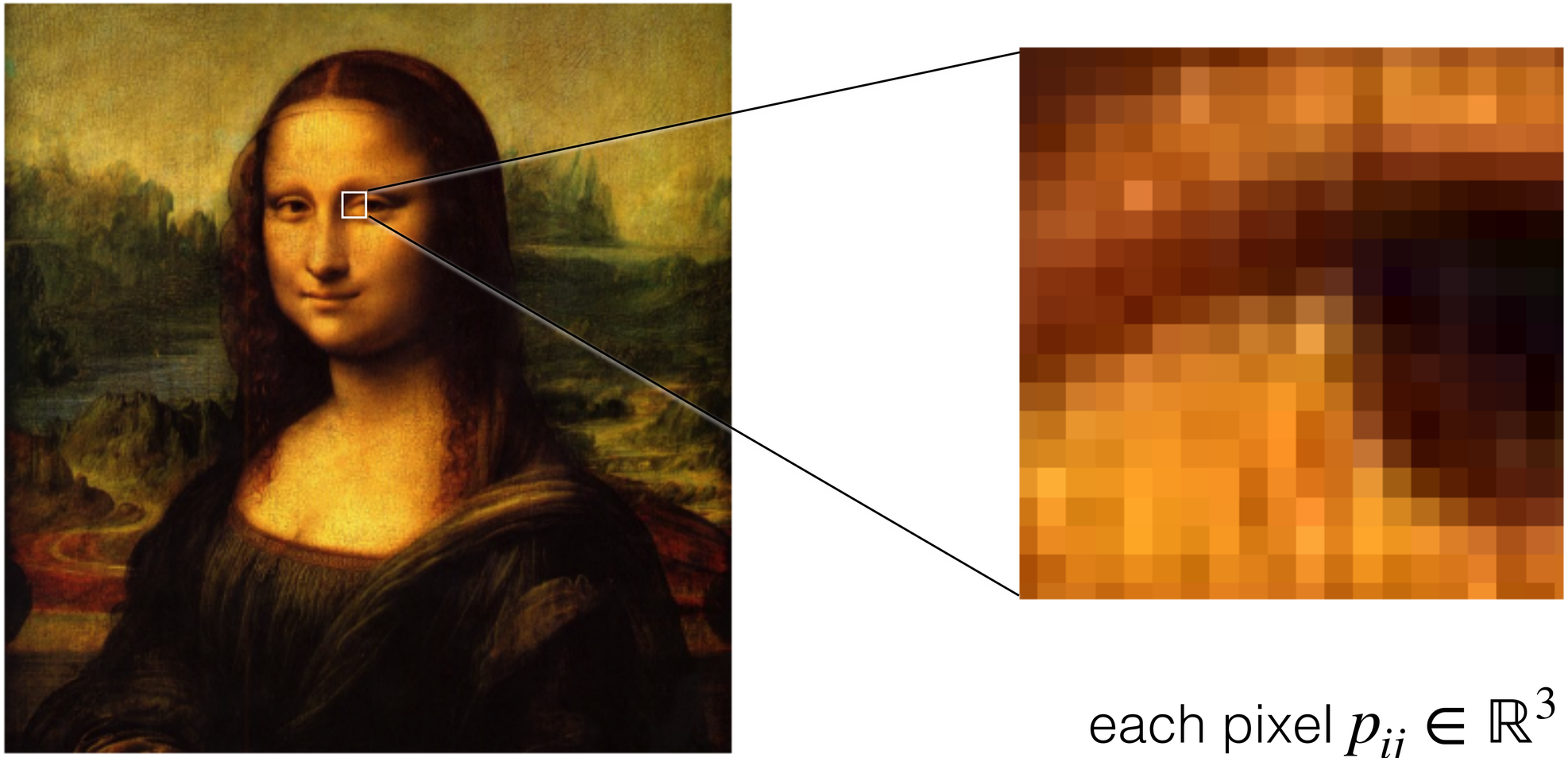
Lecture A3: Basic Image Processing

Peter Corke



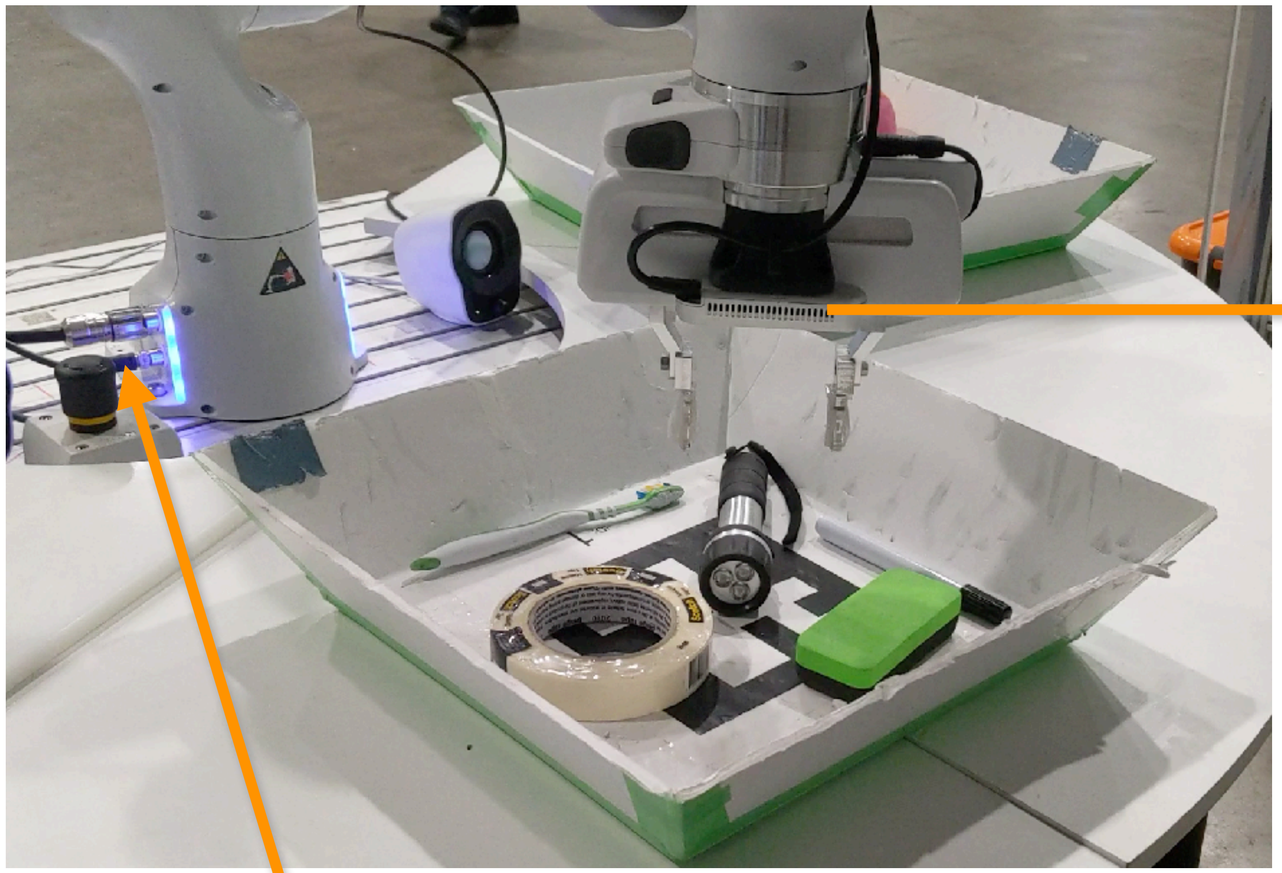
AUSTRALIAN CENTRE FOR ROBOTIC VISION SUMMER SCHOOL

A digital image comprises an array of square picture elements (pixels)



each pixel $p_{ij} \in \mathbb{R}^3$

The data output rate of a camera and the requirements of a robot are fundamentally mismatched

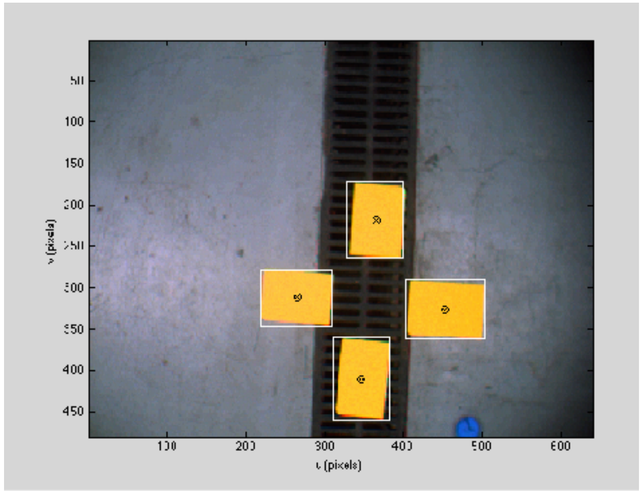


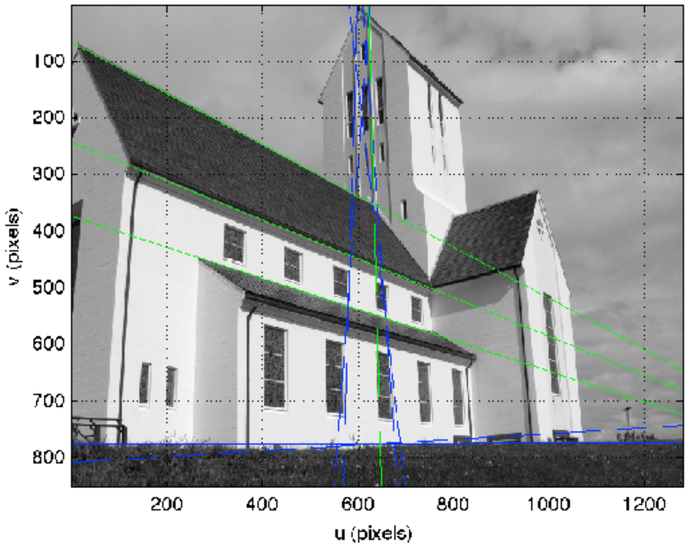
Camera output: 10 Mpix color image at 30Hz → 900 Mbyte/sec

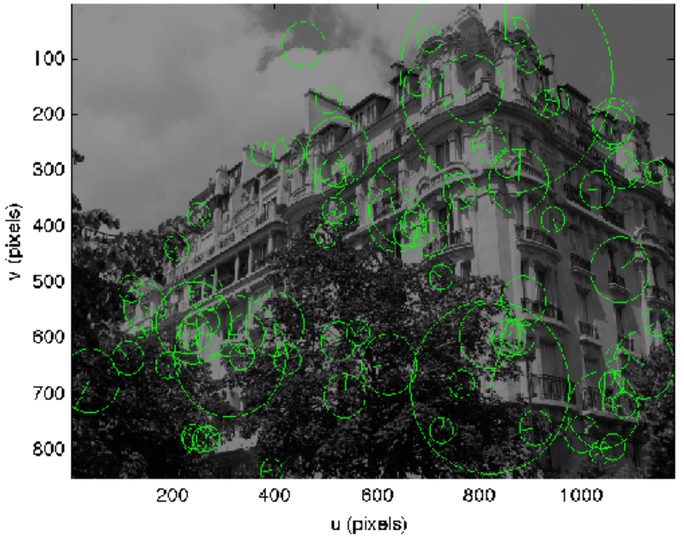
Robot input: ~10 velocities at 10Hz → 400 byte/sec

~ x10^6 mismatch

A classic approach to dealing with the data rate mismatch is to extract features from the image







Regions

- centroid coordinate
- size
- shape
- orientation
- etc.

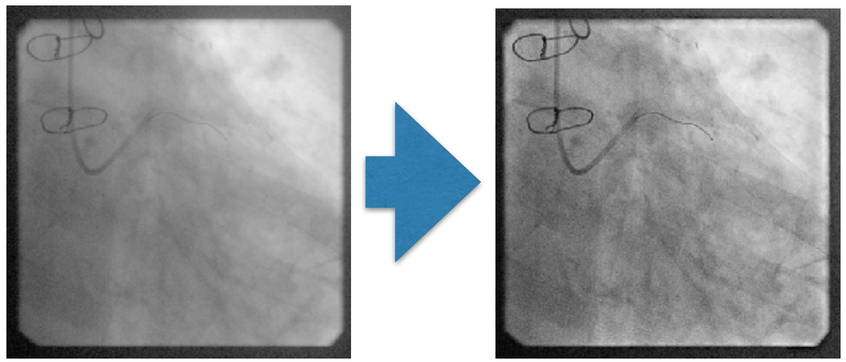
Lines

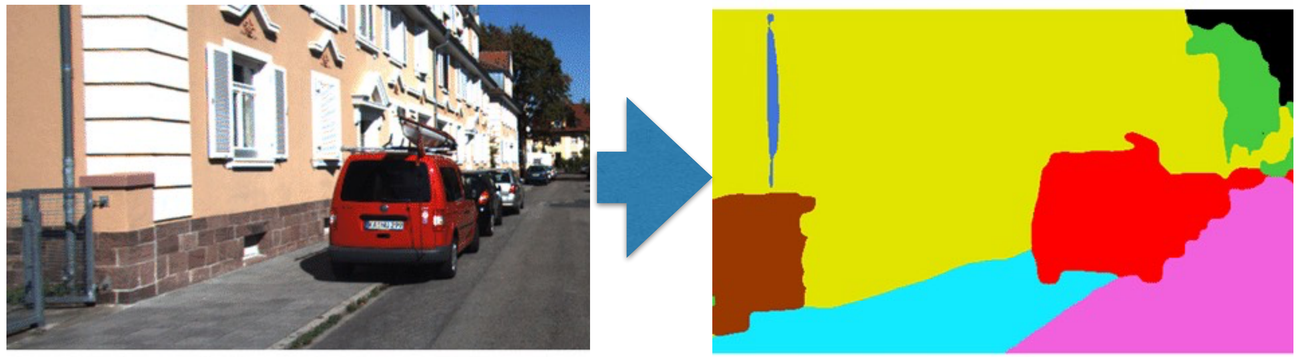
- equations of lines
- start/end of segments

Points

- coordinate
- scale

Image processing and computer vision are different things





- Image processing
  - image → image
    - enhancement
    - semantic segmentation

id	parent	centroid	area	touch	perim	circul	orient	aspect
0	-1	371.2, 355.2	7.59e+03	False	556.6	0.343	-11.7°	0.584
1	-1	171.2, 155.2	7.59e+03	False	556.6	0.343	-11.7°	0.584

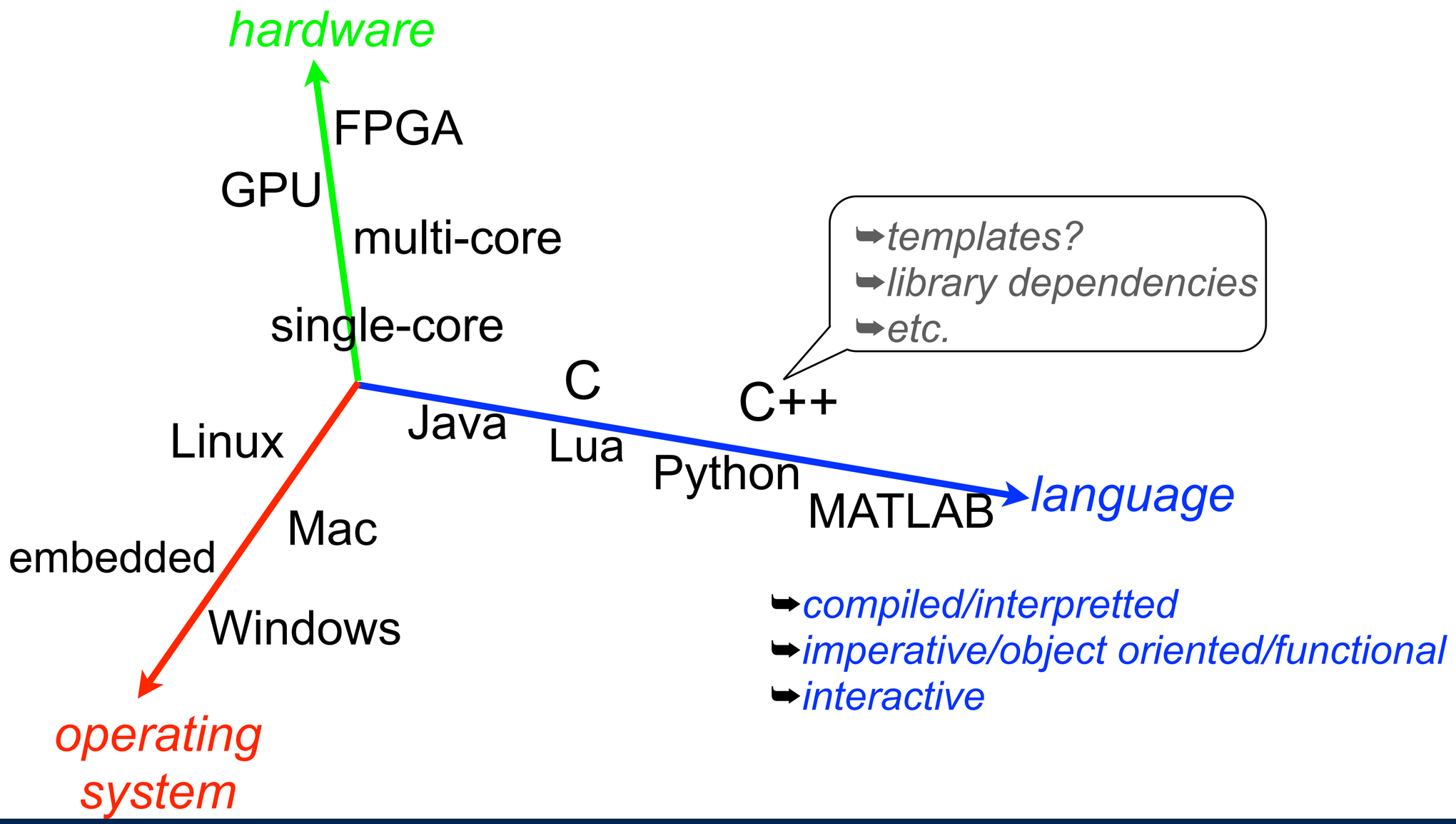
- Computer vision
  - image → features
- aka machine vision

Technology for processing images

Over recent decades there have been hundreds of software implementations for computer vision (and image processing)

IMLAB
IM
CImg
VLFeat
NeatVision
XVision
VASARI
pythonvision
ViSP
Vista
PIL
VXL
vgg_
OpenCV
pixelvision
CVIPtools
Gandalf
CMVision
Khoros
DIPlib
HIPS
ImageLib
ImageJ
PBMplus
TargetJr
ImageMagick
TINA
CppIma

The design space has a number of axes



We have chosen to use Python and OpenCV

- Python
  - ✓ Common in teaching, research and industry
  - ✓ Runs on many platforms
  - ✓ With NumPy it eats matrices and vectors for breakfast
  - ✓ Interactive mode or programming mode
  - ✓ Great graphics
  - ✓ Object oriented
  - x Slow
- OpenCV
  - ✓ Common in research and industry
  - ✓ Runs on many platforms
  - ✓ Has good Python wrappers
  - ✓ Highly optimised code
  - x Not object oriented

For my part of the course we will be using a wrapped version of OpenCV - the Machine Vision Toolbox for Python

Machine Vision Toolbox for Python

collection PYTHONROBOTICS | powered by spatial maths | collection QUT Robotics

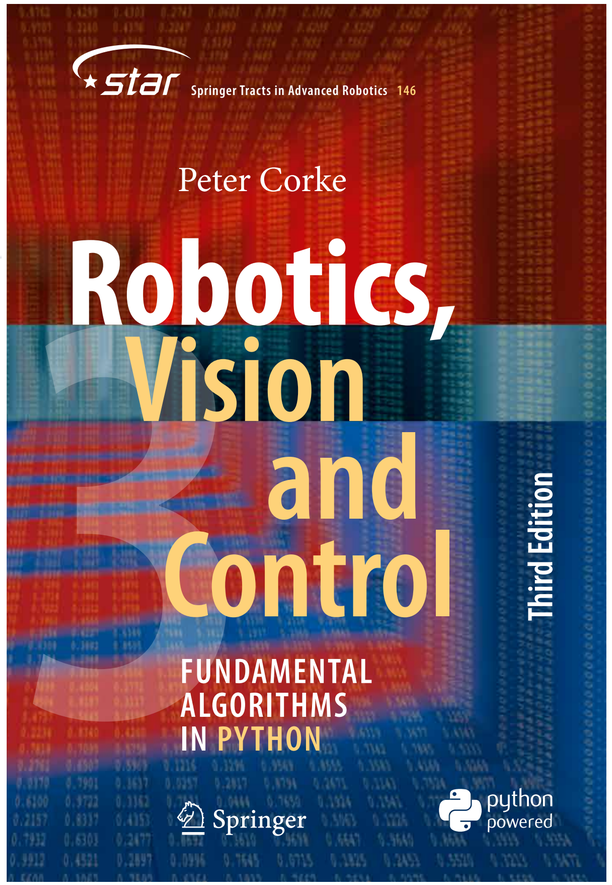
pypi package 1.0.0 | python 3.9 | 3.10 | 3.11 | 3.12 | powered by OpenCV | powered by Open3D

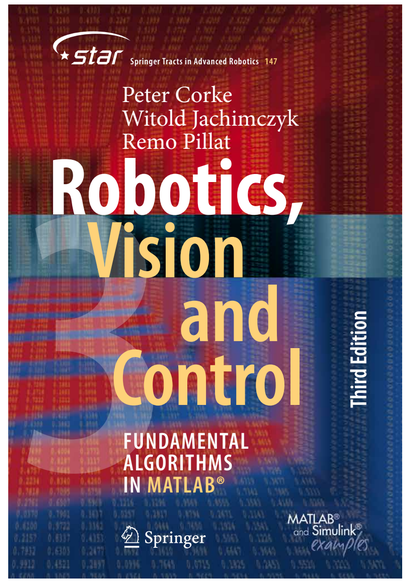
License MIT

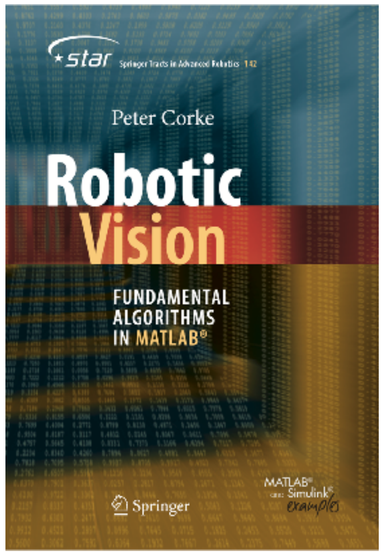
mvtb-main passing | codecov 32% | downloads 122/week

- GitHub repository
- Documentation
- Examples and details
- Installation
- Examples and details
- Changelog

A Python implementation of the Machine Vision Toolbox for MATLAB®





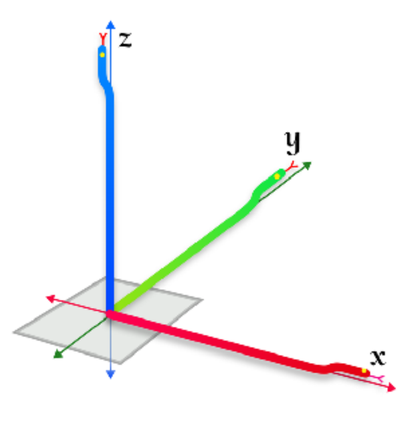


The install is easy

```
pip install machinevision-toolbox-python
```

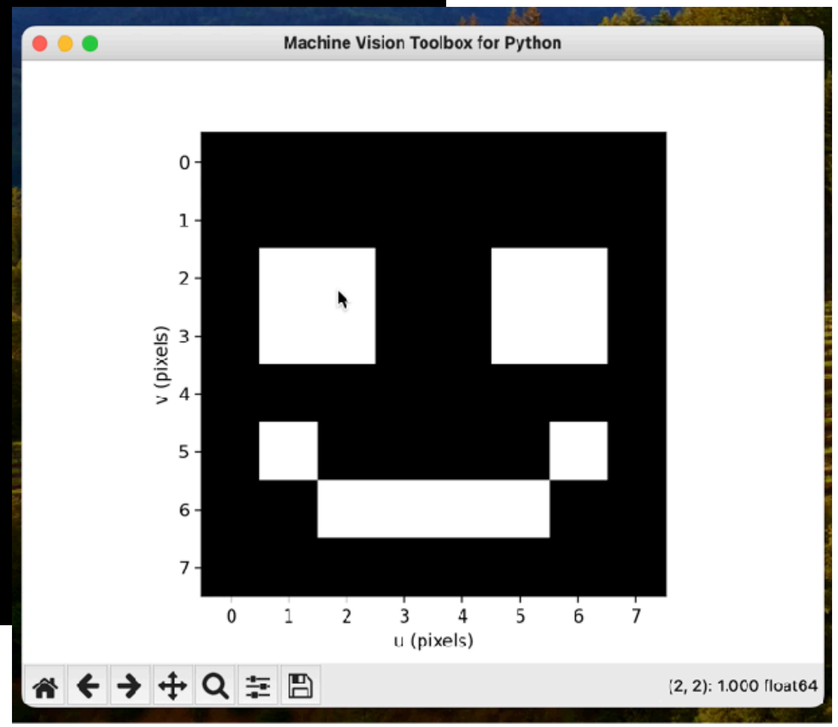
and it also pulls in:

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



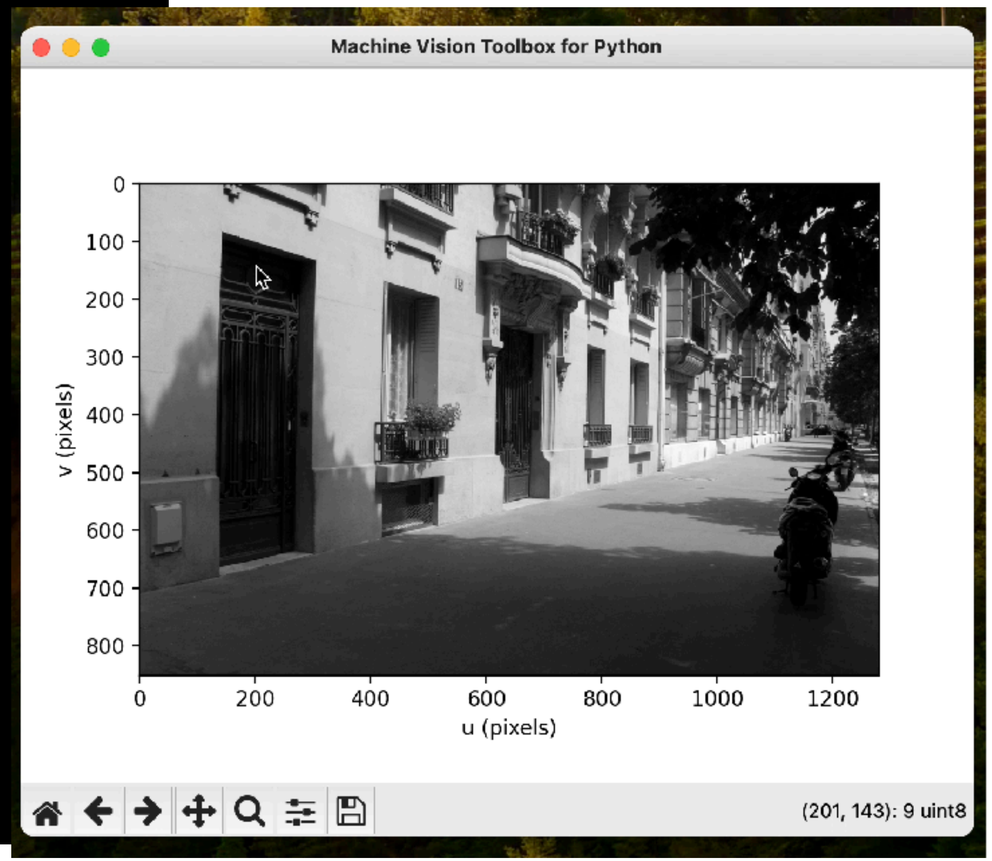
An image is simply a 2D array of numbers

```
from machinevisiontoolbox import *
import numpy as np

a = np.zeros((64,))
a[[17, 18, 21, 22, 25, 26, 29, 30, 41, 46, 50, 51, 52, 53]] = 1
print(a)

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 1. 0.
 0. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.
 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

a = a.reshape((8, 8))

[[0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 1. 0. 0. 1. 1. 0.]
 [0. 1. 1. 0. 0. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 1. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]


idisp(a, block=True)
```



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)
```

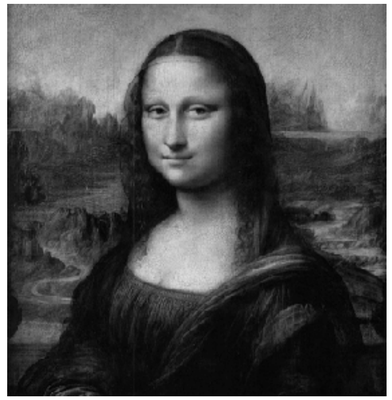
```
from machinevisiontoolbox import *

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)
```

Python awesomeness (language, tools, support) plus the speed of OpenCV

There are lots of ways to acquire an image to work with

```
from machinevisiontoolbox import Image

mona = Image.Read(“monalisa.png”)
mona.disp()
```

read from a file

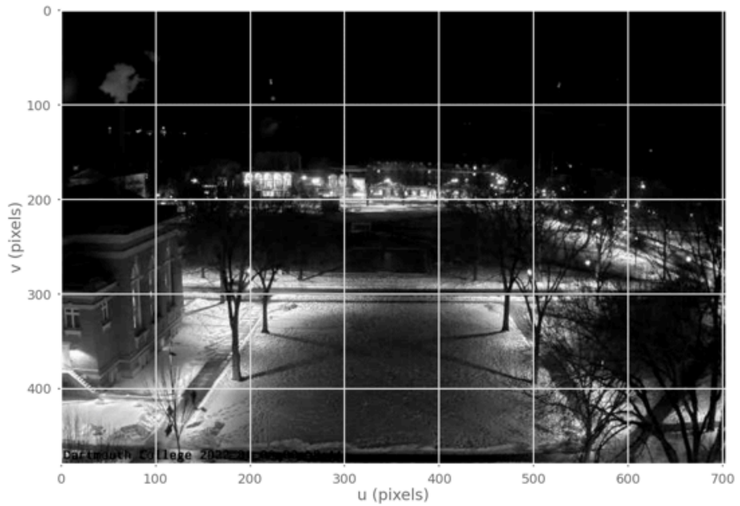


```
camera = VideoCamera()
camera.grab().disp()
```

read from an attached camera via OpenCV

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

read from somebody else’s camera




```
im = Image.Random(20).disp()
```

create it using code

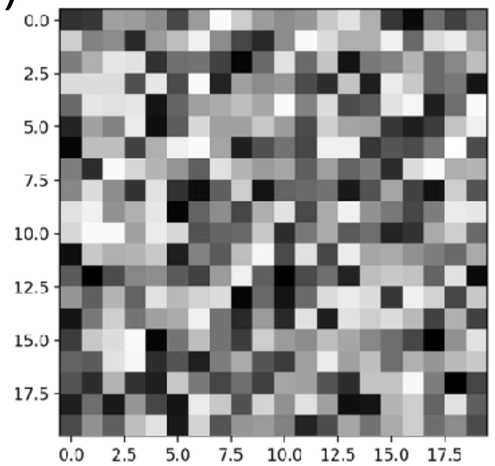
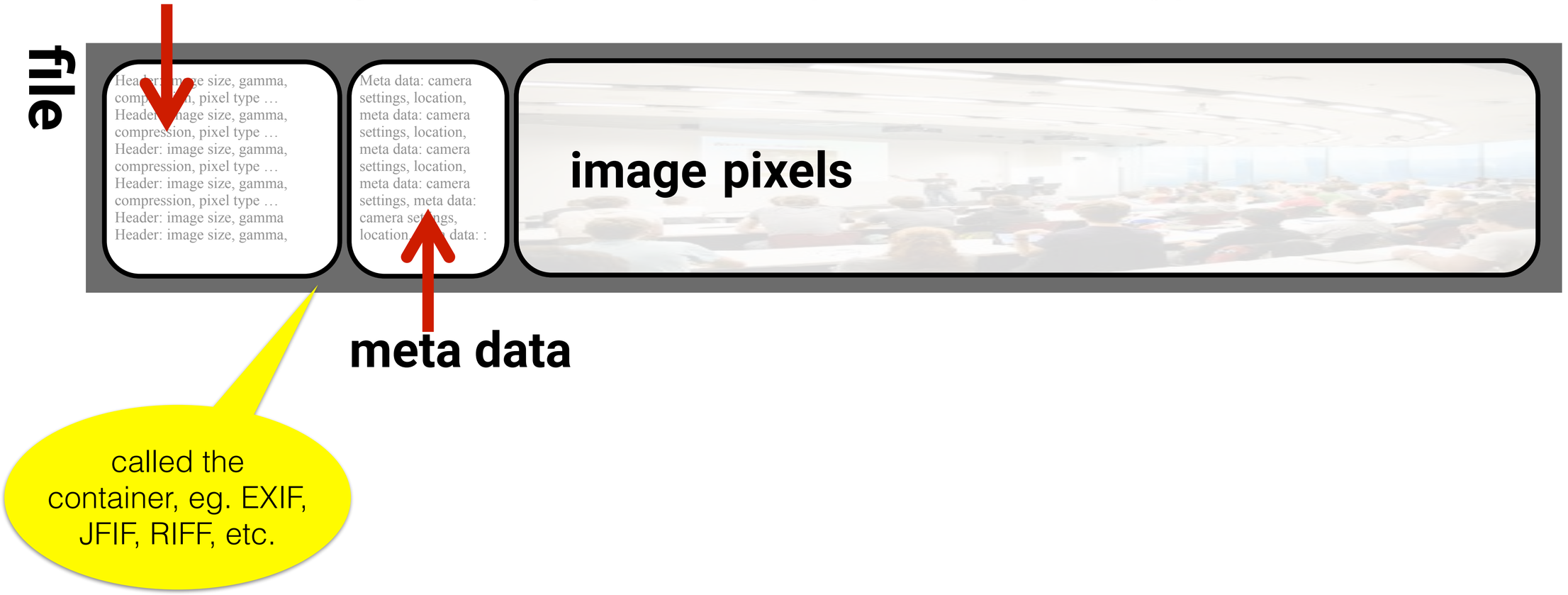
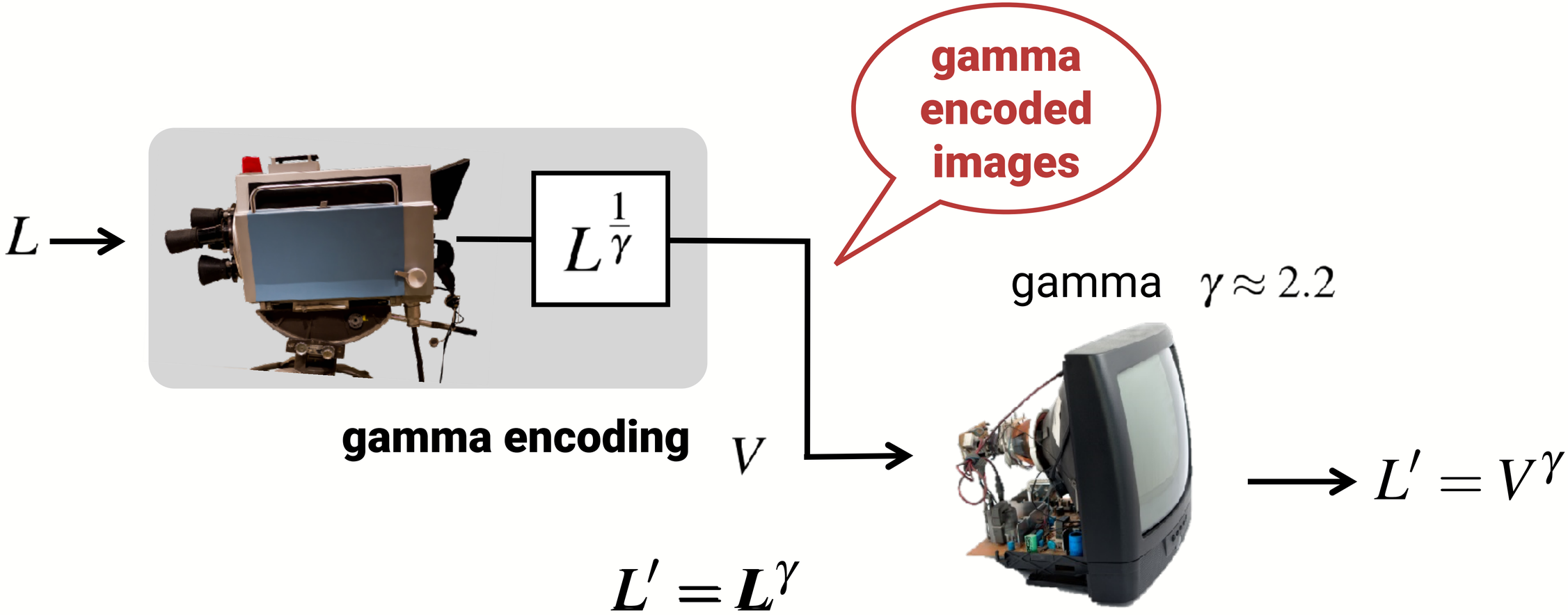


Image file formats

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



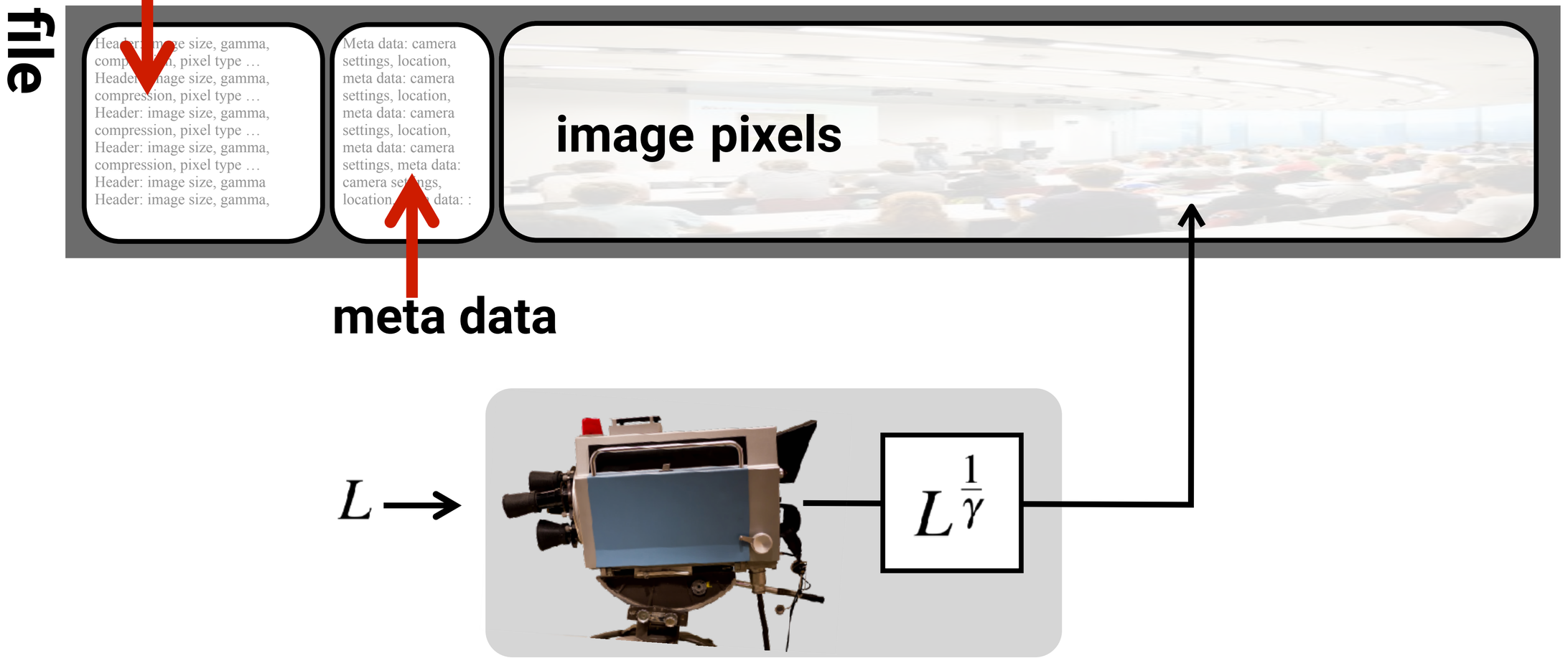
Display non-linearity



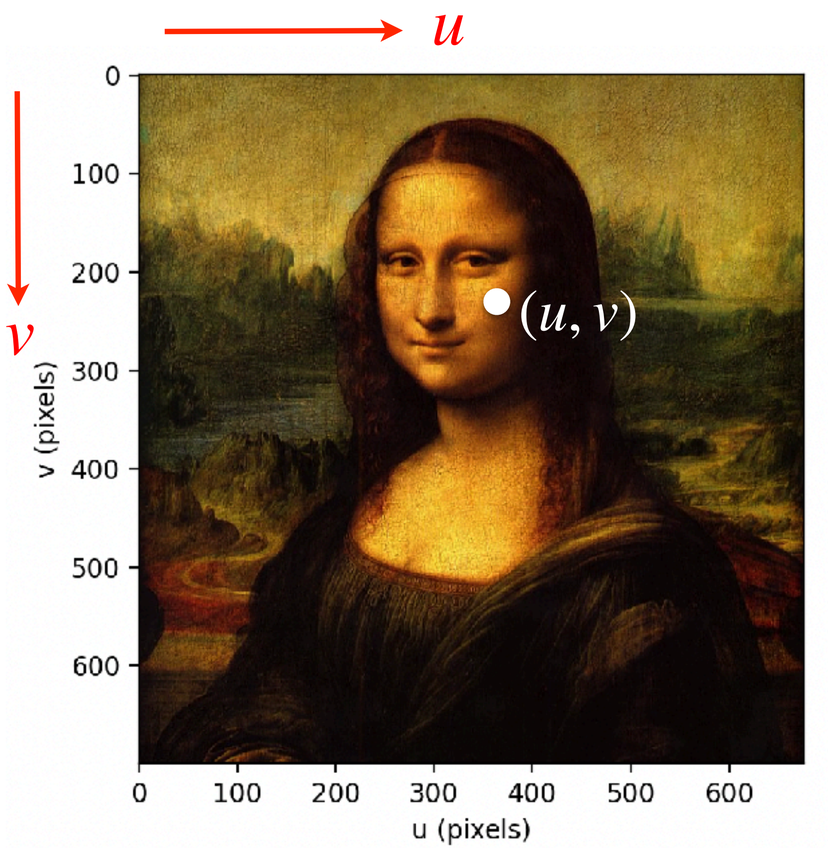
The system is linear end-to-end

Image file formats

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



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



row=v column=u

```
>>> mona, _ = iread("monalisa.png")
>>> mona[200,100]
array([111,  98,  39], dtype=uint8)
```

DANGER: Reversed compared to graph coordinates!

u v

```
>>> mona = Image.Read("monalisa.png")
>>> mona[100,200]
array([111,  98,  39], dtype=uint8)
```

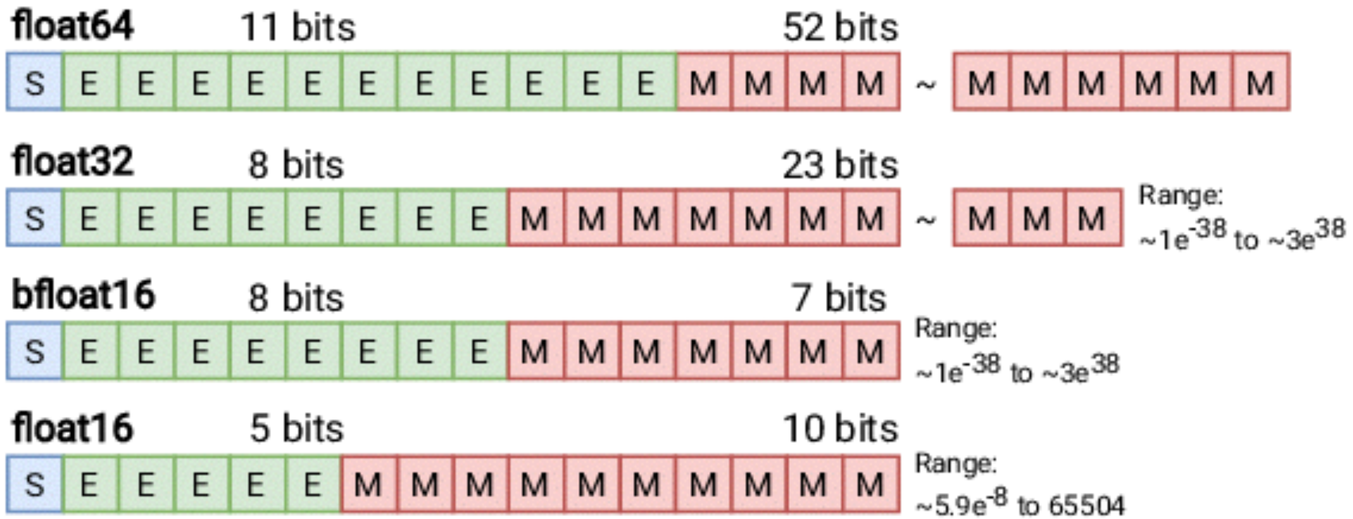
Images can have tens of millions of pixels, so we want to use a compact representation of each pixel value. uint8 is common

single pixel "images"

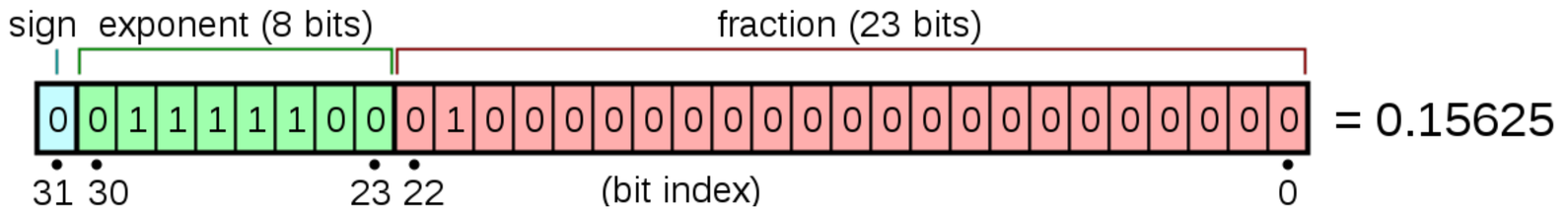
```
>>> a = np.array(5, “uint8”)
>>> b = np.array(10, “uint8”)
>>> a + b
15
>>> import cv2
>>> cv2.add(a,b)
array([[15]], dtype=uint8)

>>> 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)
```

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

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

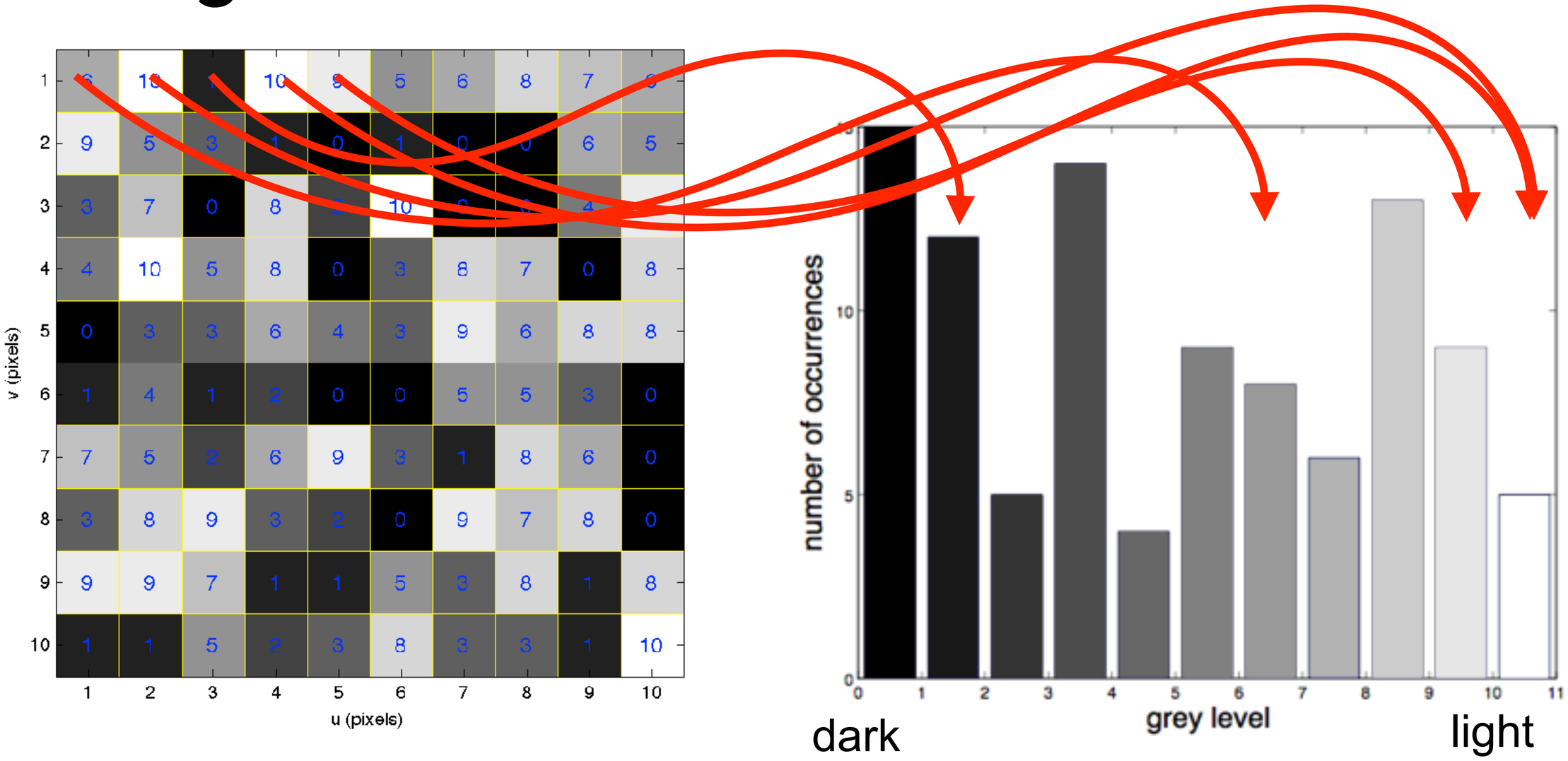
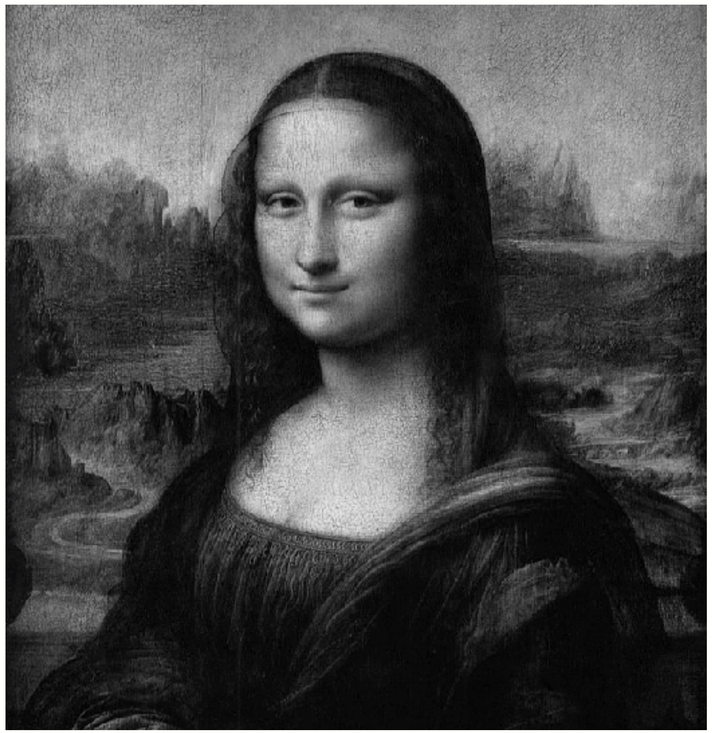
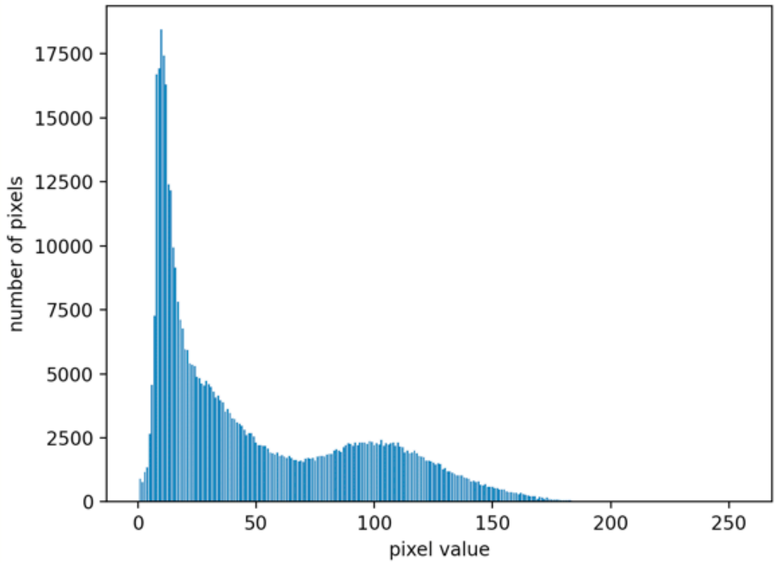
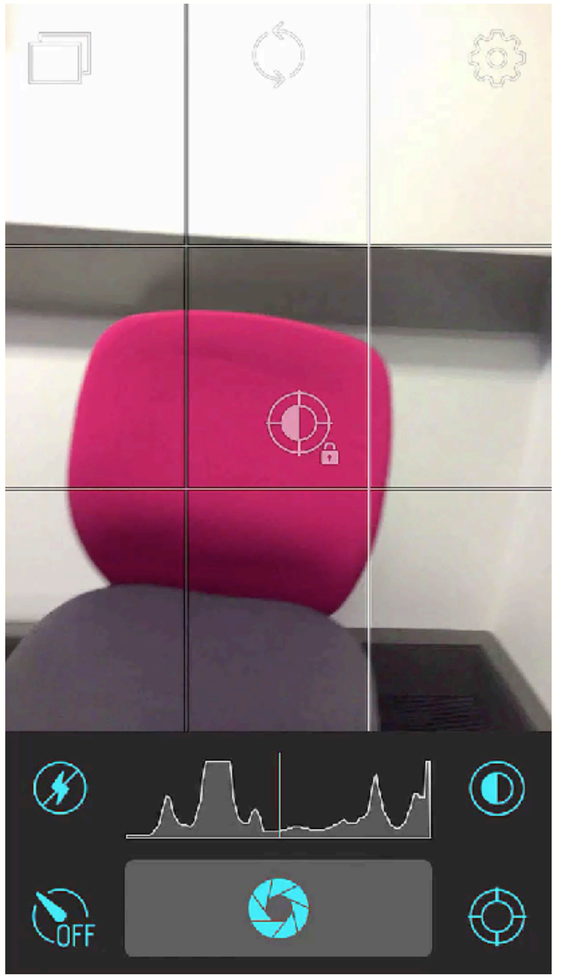

v (pixels)
u (pixels)
number of occurrences
grey level
dark
light

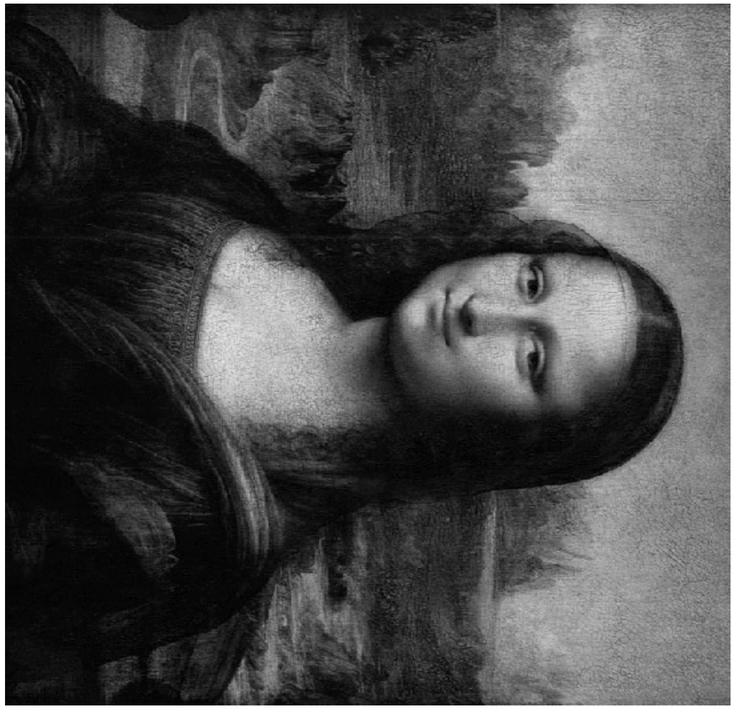
Image histograms

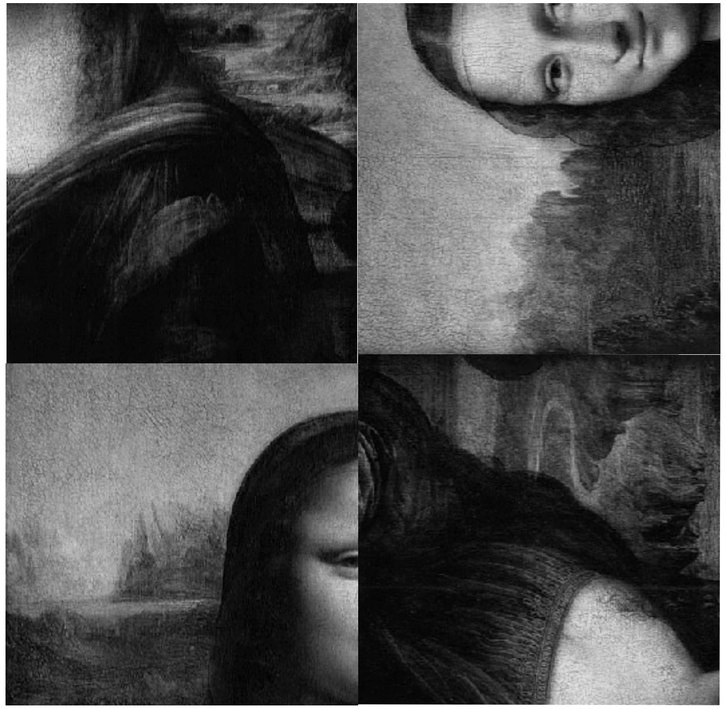


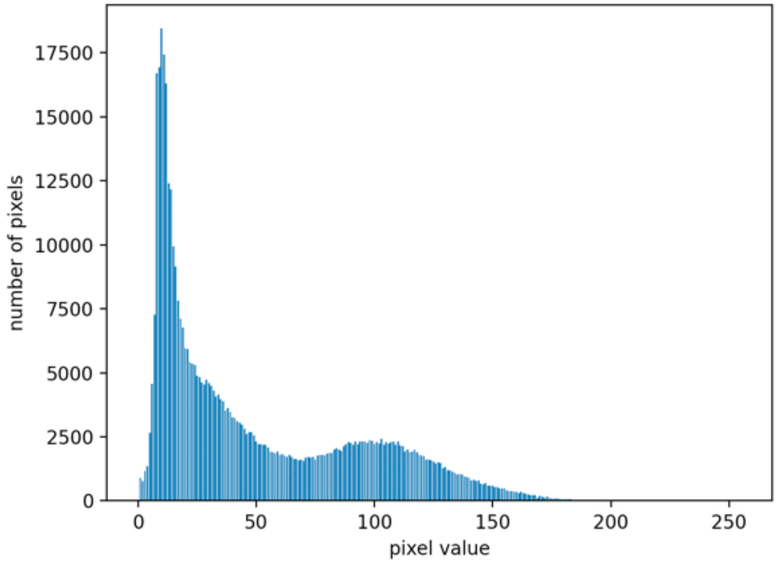




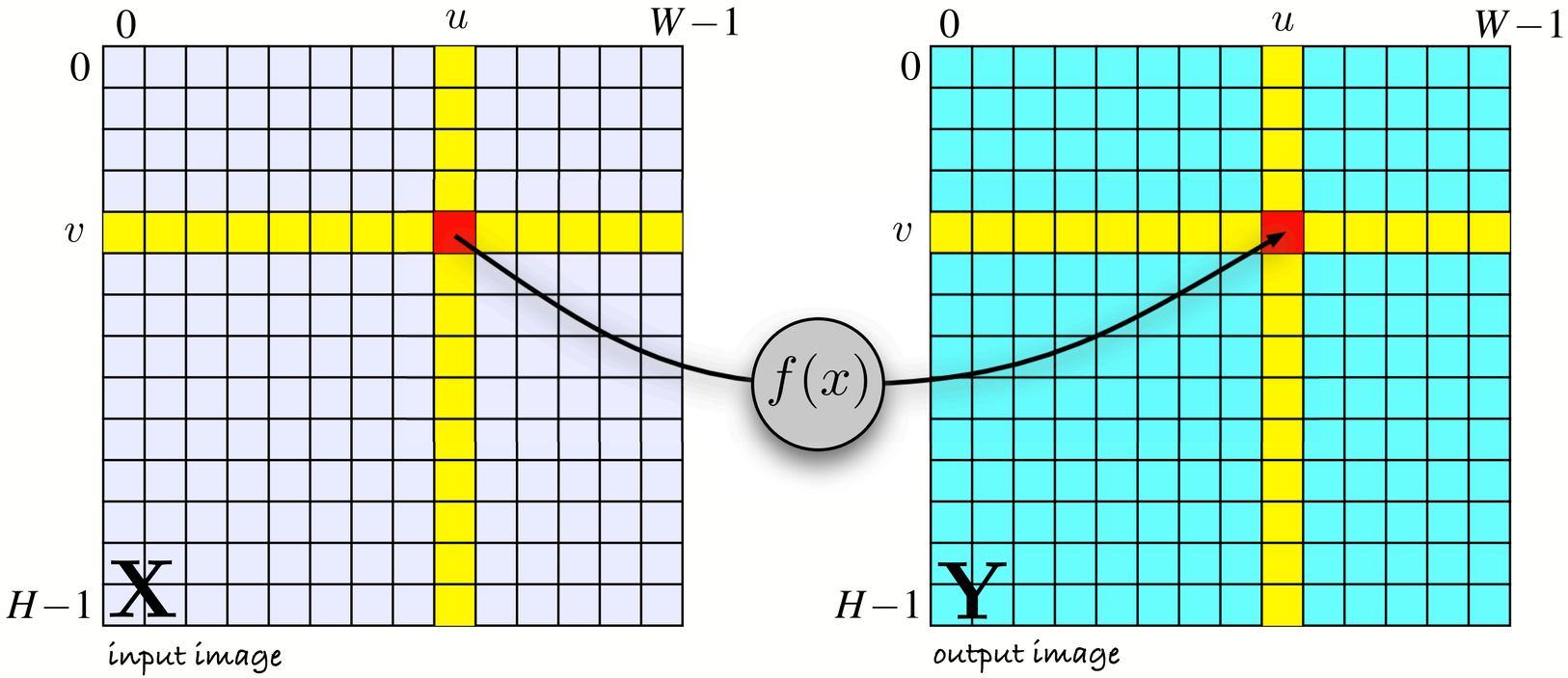
The histogram is invariant to rotation or even complete spatial rearrangement of the pixels





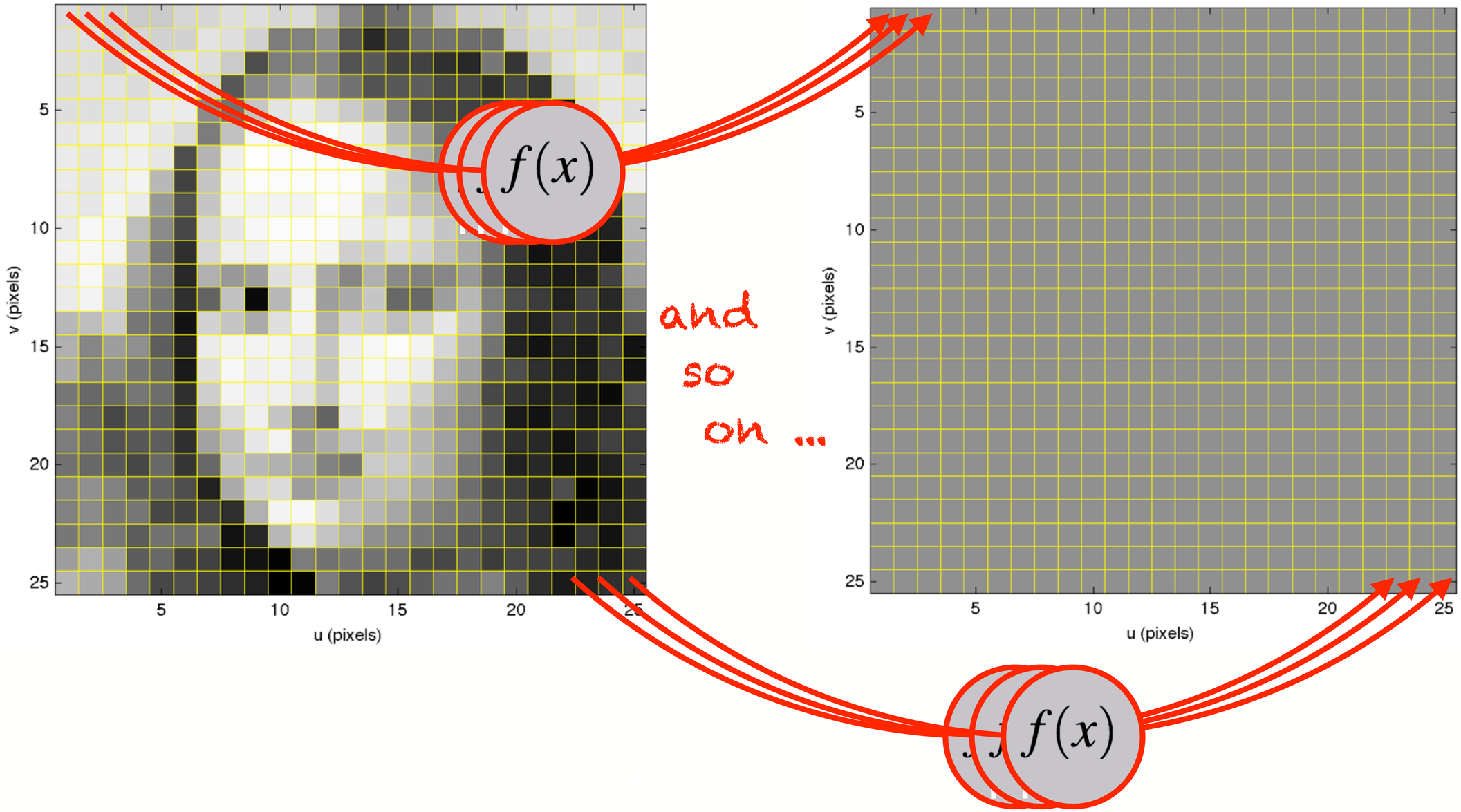


Monadic processing



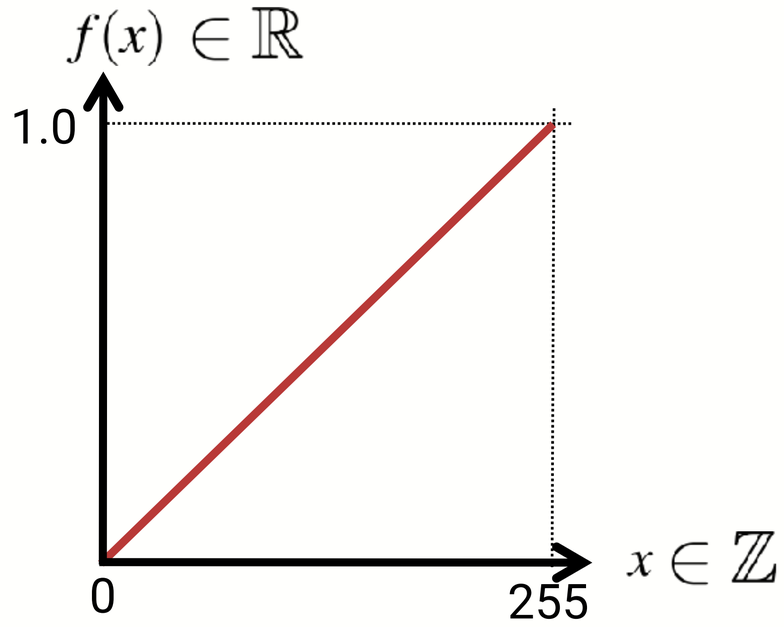
- Each output pixel is a function of corresponding input pixel
- The function is **the same** for all pixels

Monadic image processing

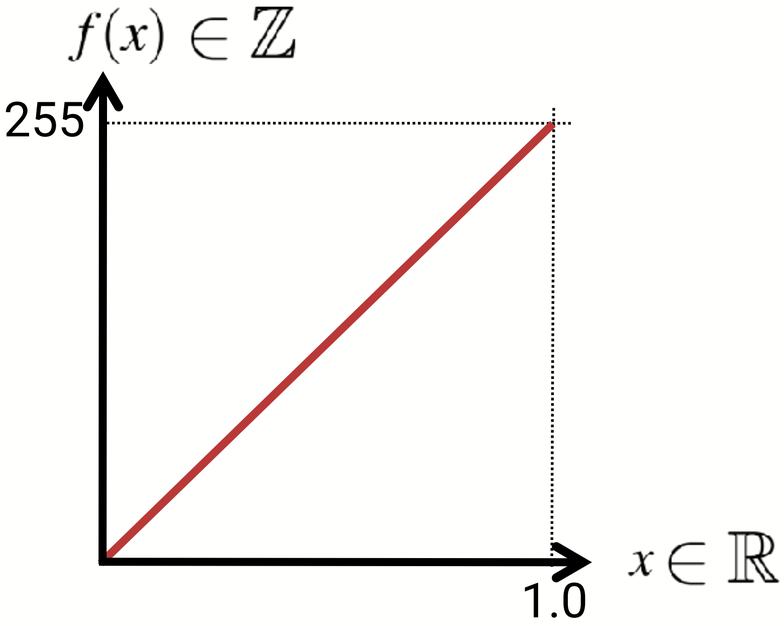


Changing pixel data type

- uint8 ➞ float32
  - ⮑ pixels in the range [0, 1 ⋯ 255] mapped to [0.0 ⋯ 1.0]

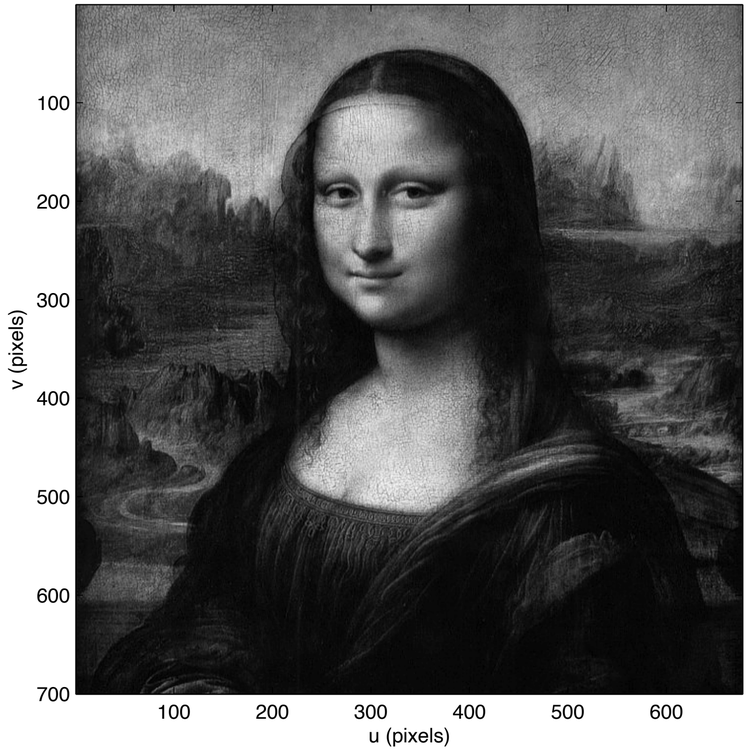


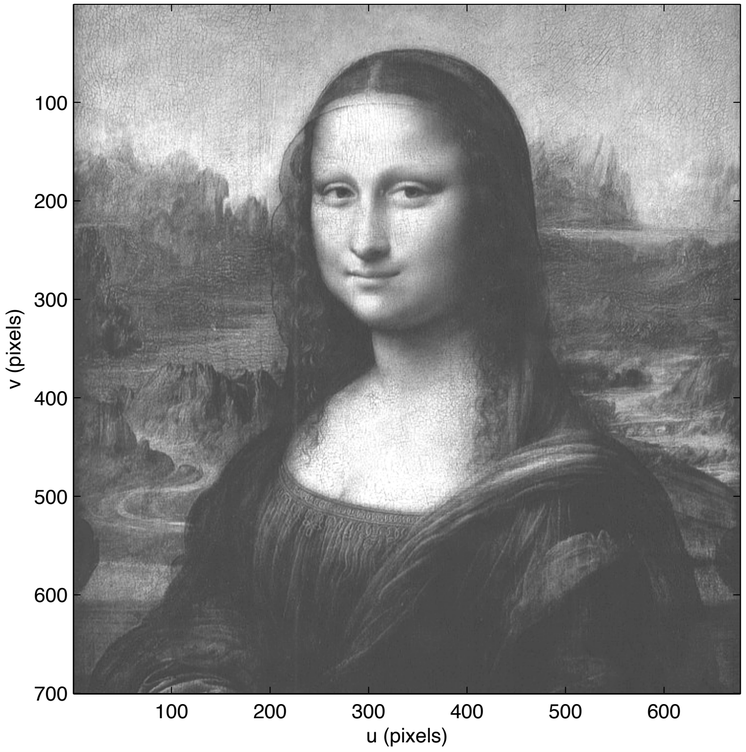
- float32 ➞ uint8
  - ⮑ pixels in the range [0.0 ⋯ 1.0] mapped to [0, 1 ⋯ 255]

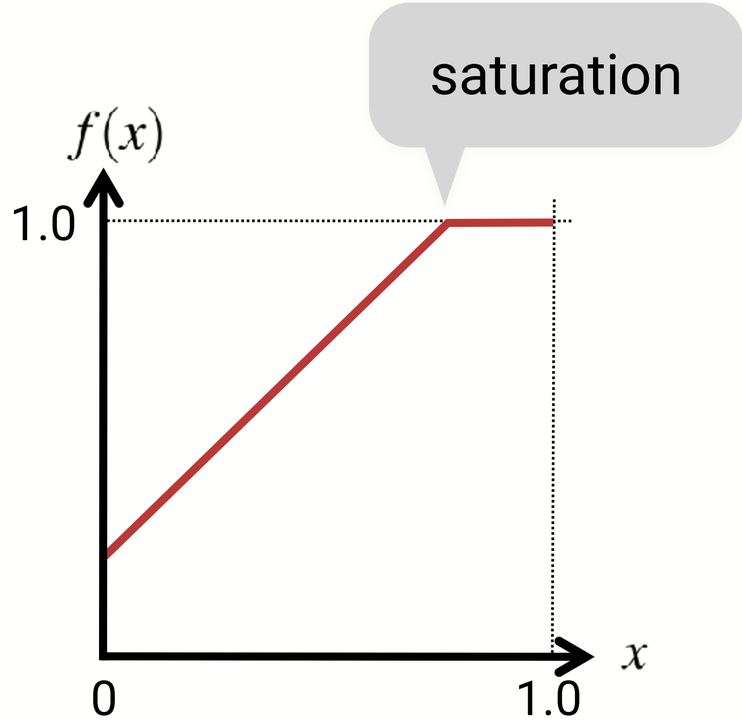


Changing brightness

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





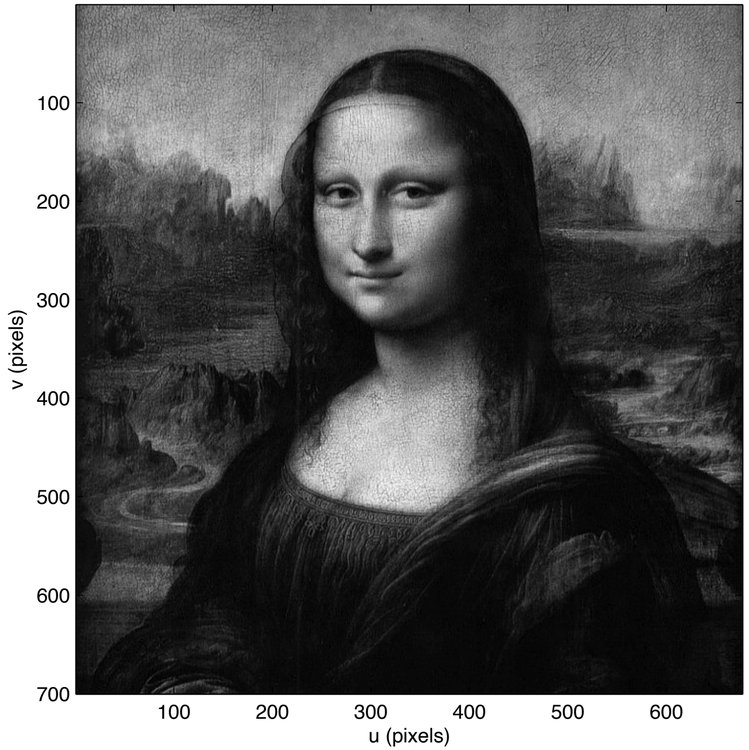


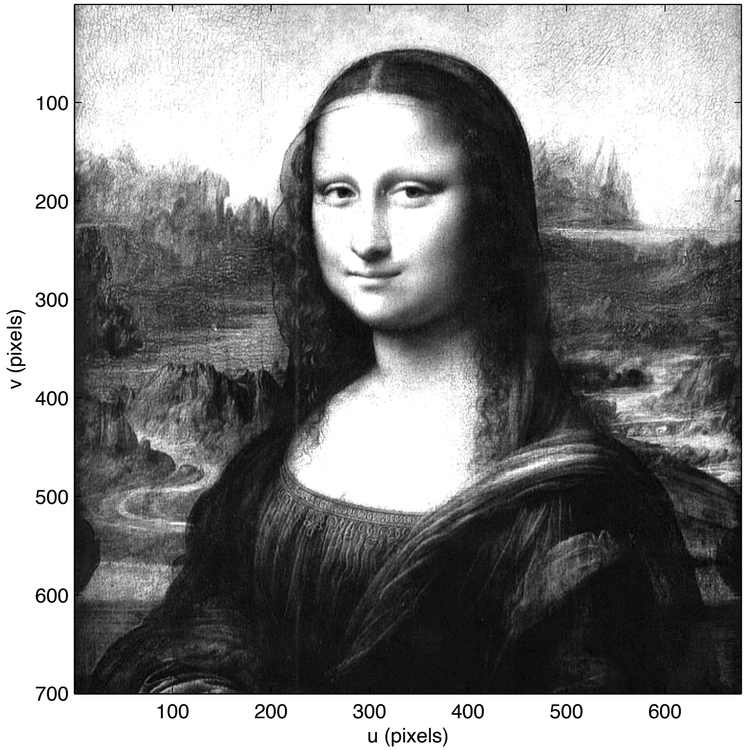
Mona Lisa c.1503–1505

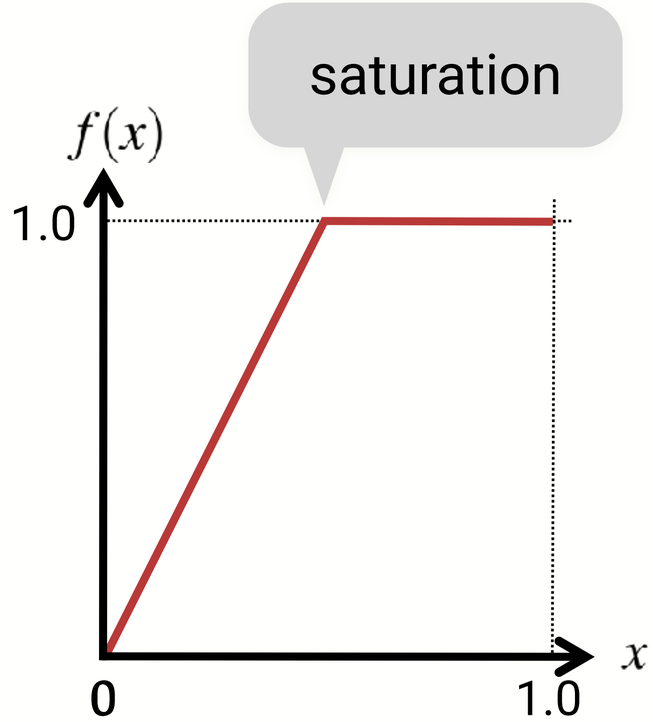
Da Vinci, L.

Changing contrast

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





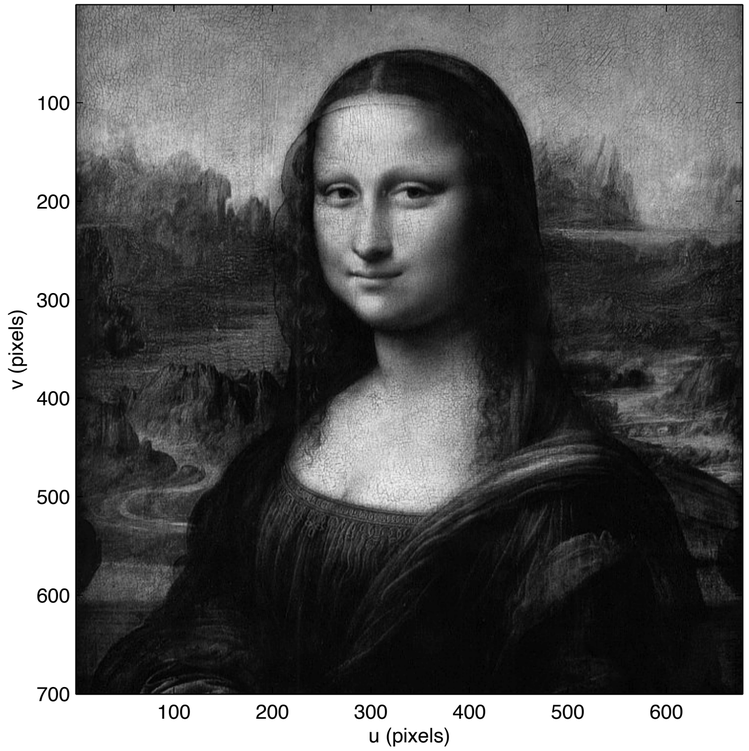


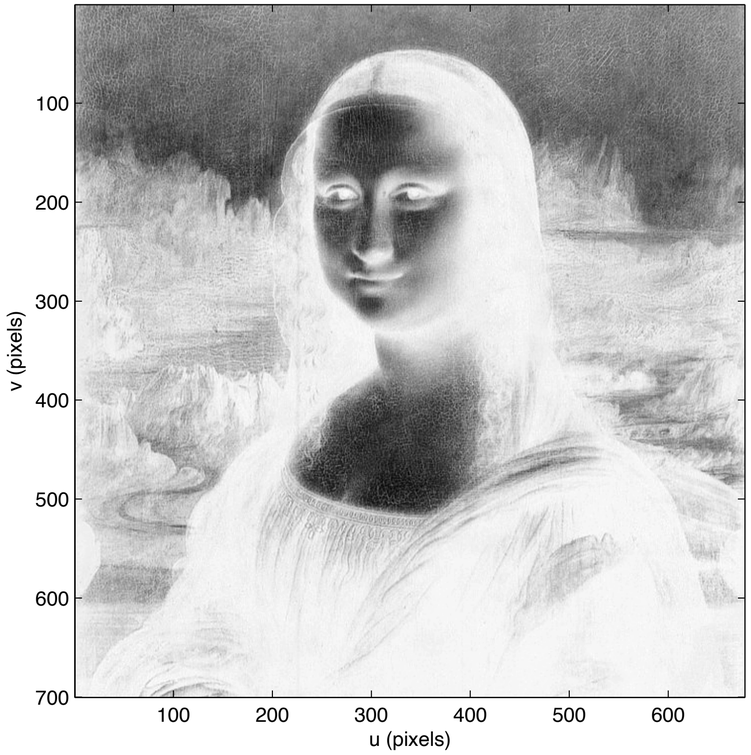
Mona Lisa c.1503–1505

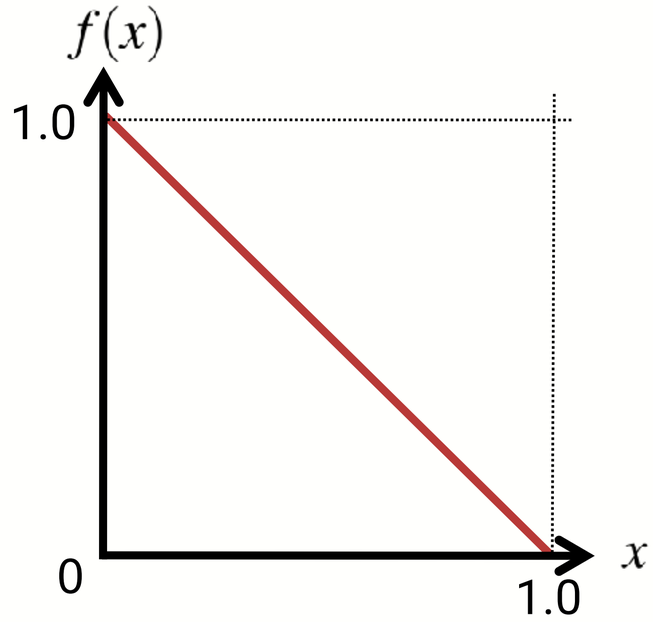
Da Vinci, L.

Negative image

- f(x) = 1-x





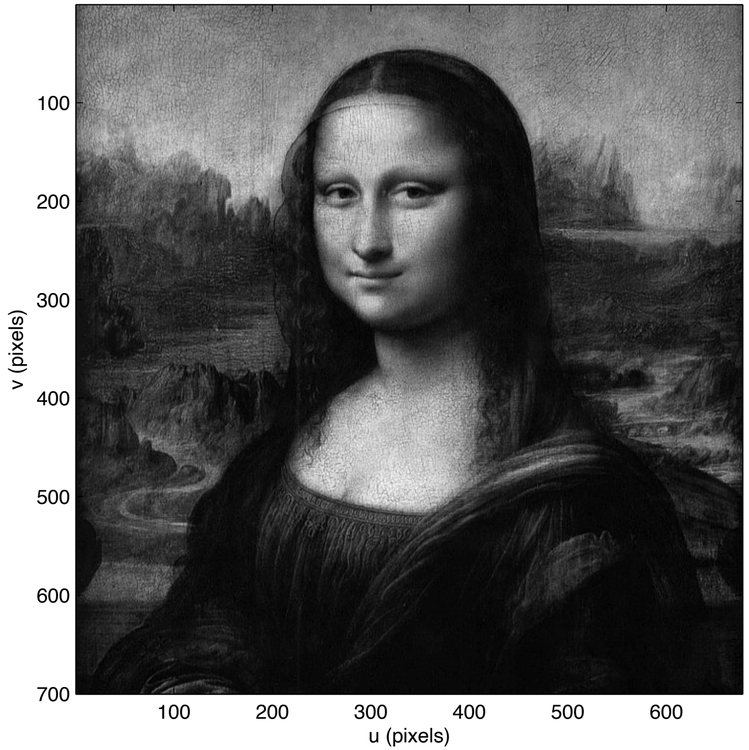


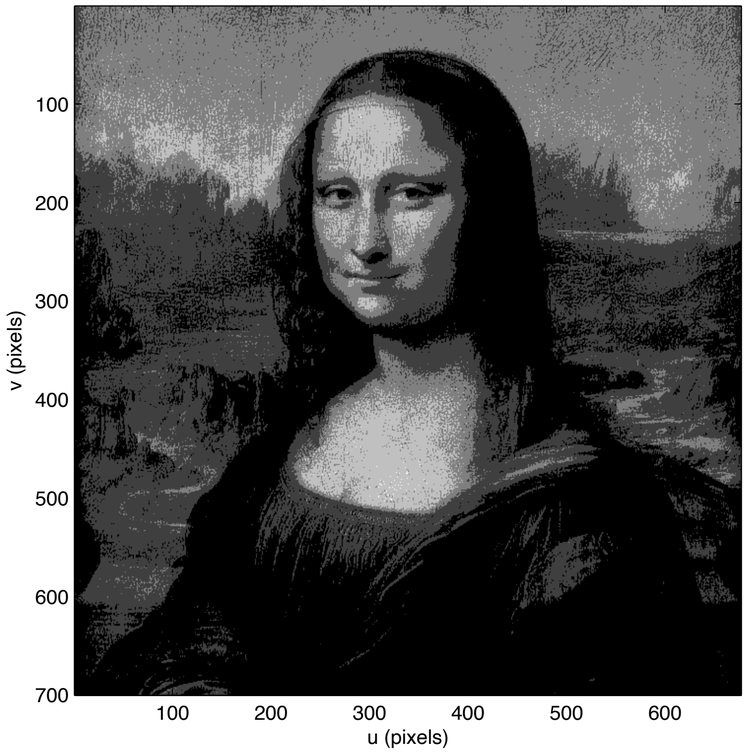
Mona Lisa c.1503–1505

Da Vinci, L.

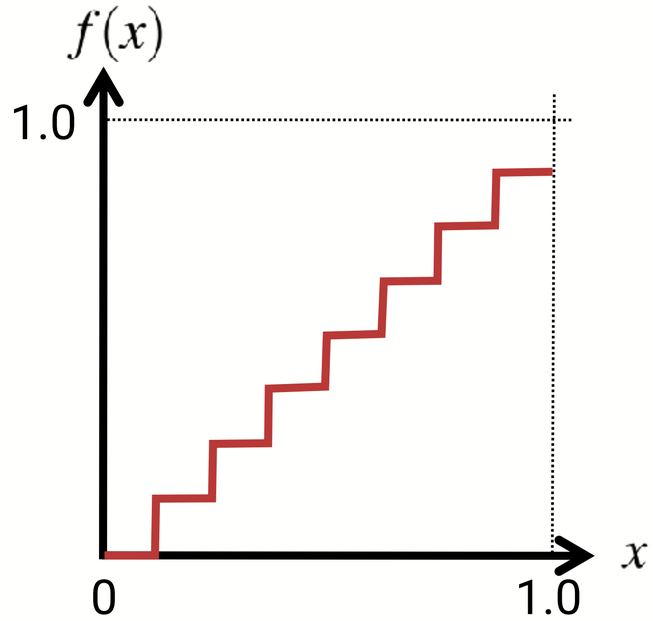
Posterisation

- Only N unique grey levels in the output image
  - ↪ f(x) = N floor(x/N)





N=4

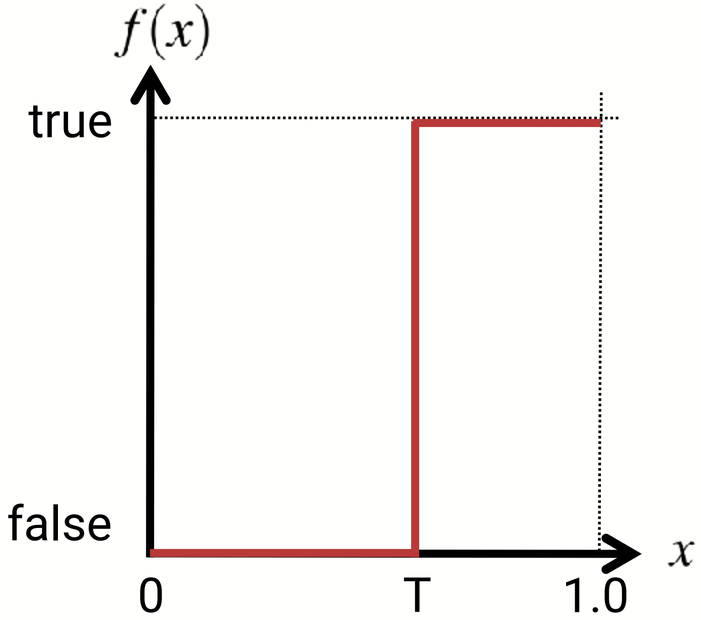


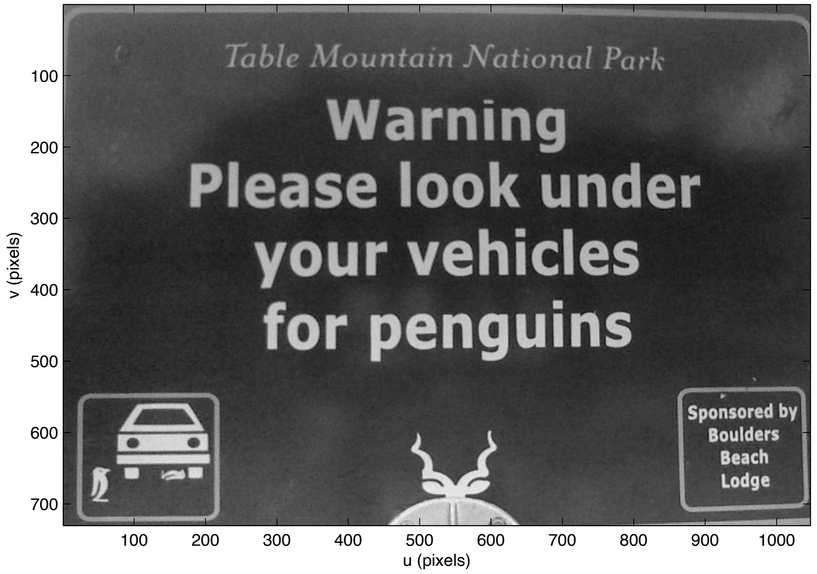
Mona Lisa c.1503–1505

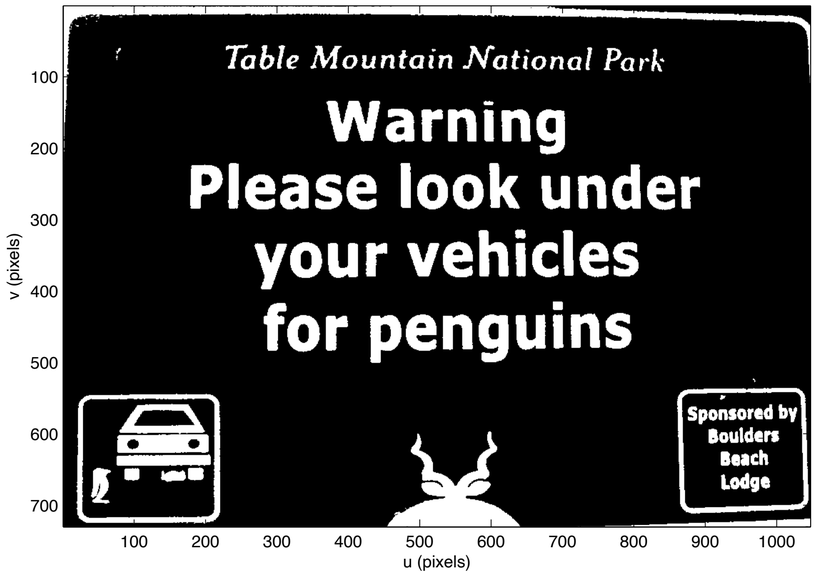
Da Vinci, L.

Thresholding

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

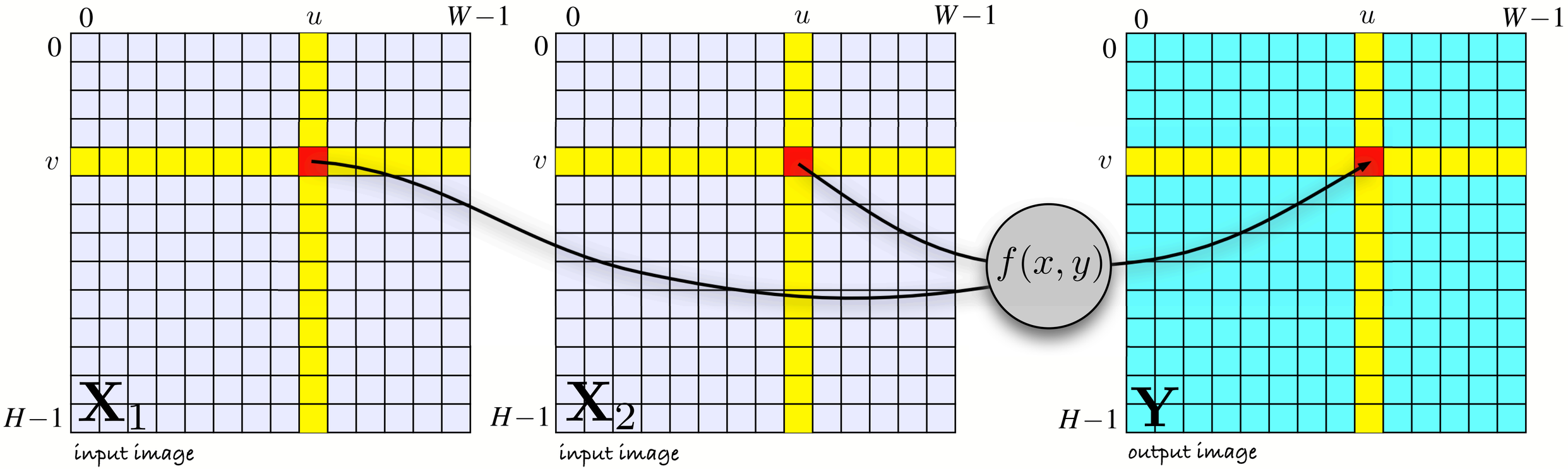






T = 0.4

Diadic operations

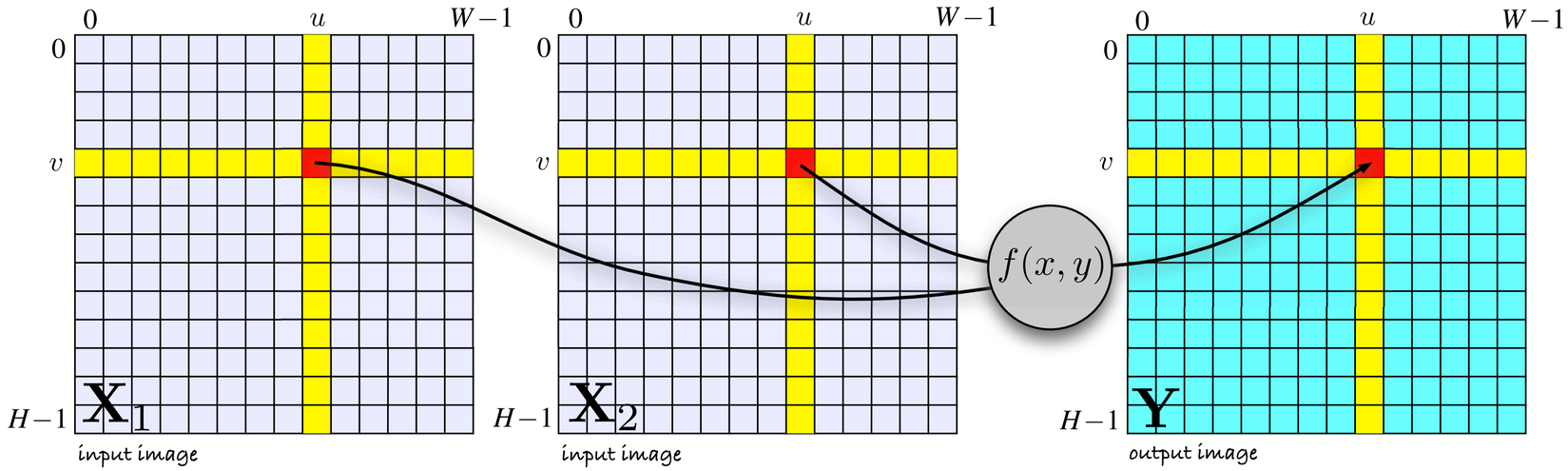


- Each output pixel is a function of two corresponding input pixels
- The function is **the same** for all pixels

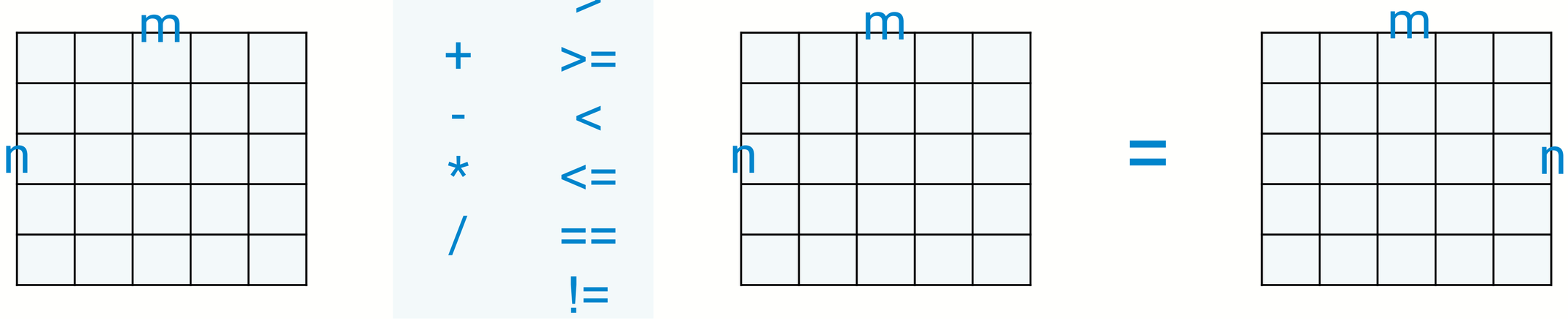
Robotics, vision and control: Fundamental algorithms in PYTHON 2023

Diadic image processing

- What is f(x,y)?
  - arithmetic
    - x+y
    - x-y
    - x*y
    - etc.
  - $\sqrt{x^2+y^2}$
  - relational
    - x > y
    - x < y
    - etc.



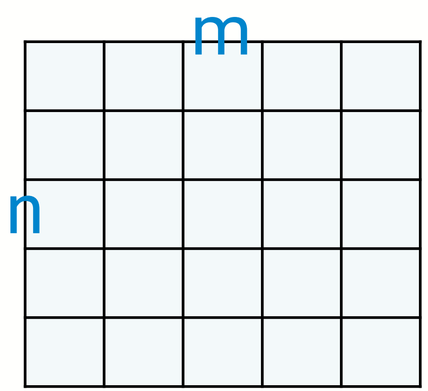
NumPy array operations

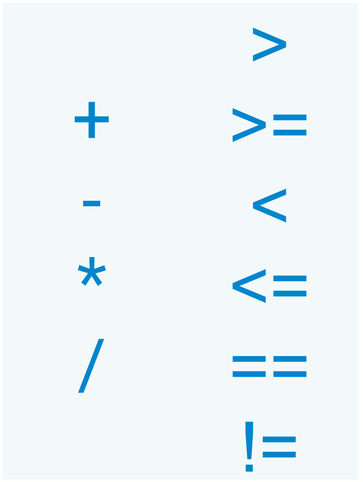


Both operands must have the same size

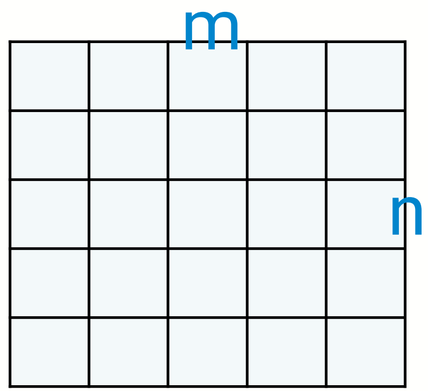
The output is the same size as well

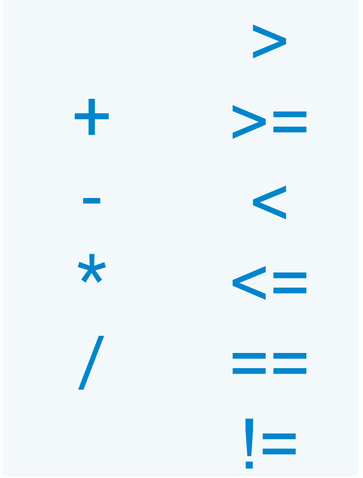
NumPy array operations with broadcasting

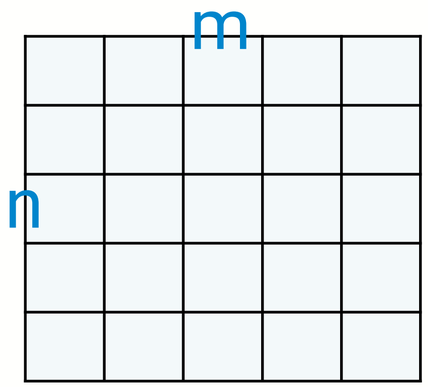




=







=

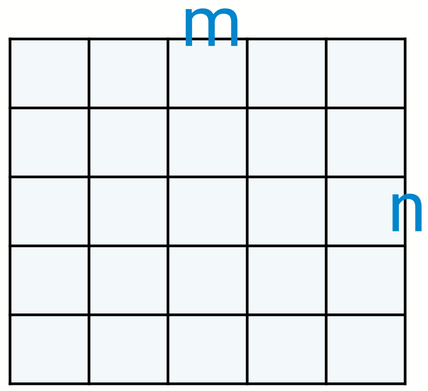
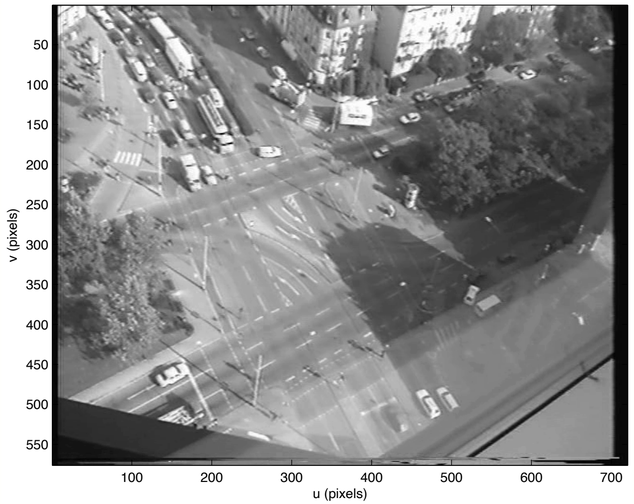
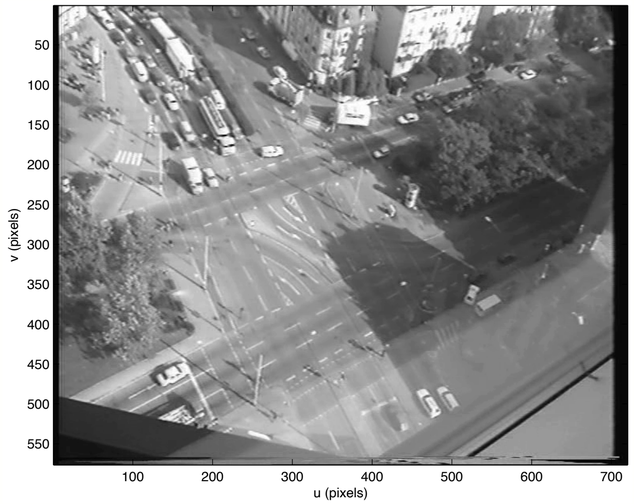


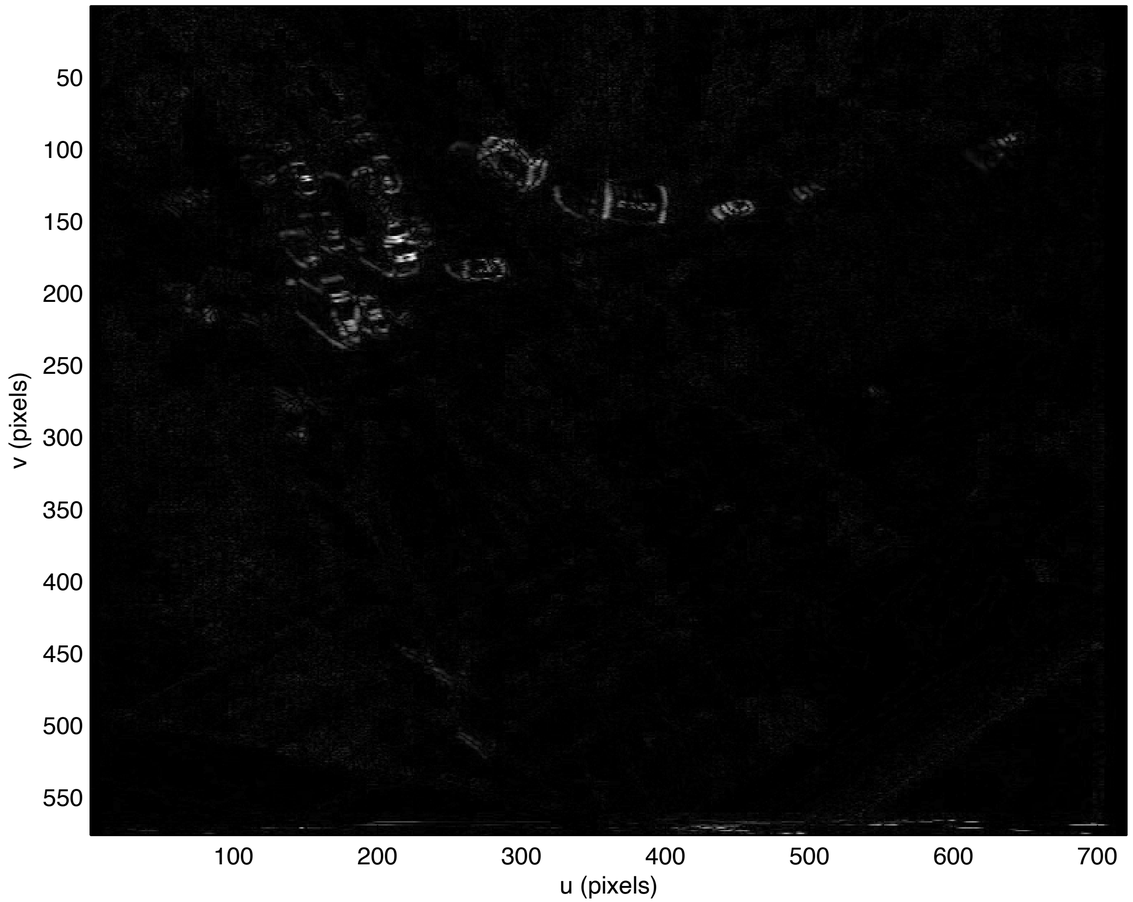
Image subtraction



frame 1



frame 4



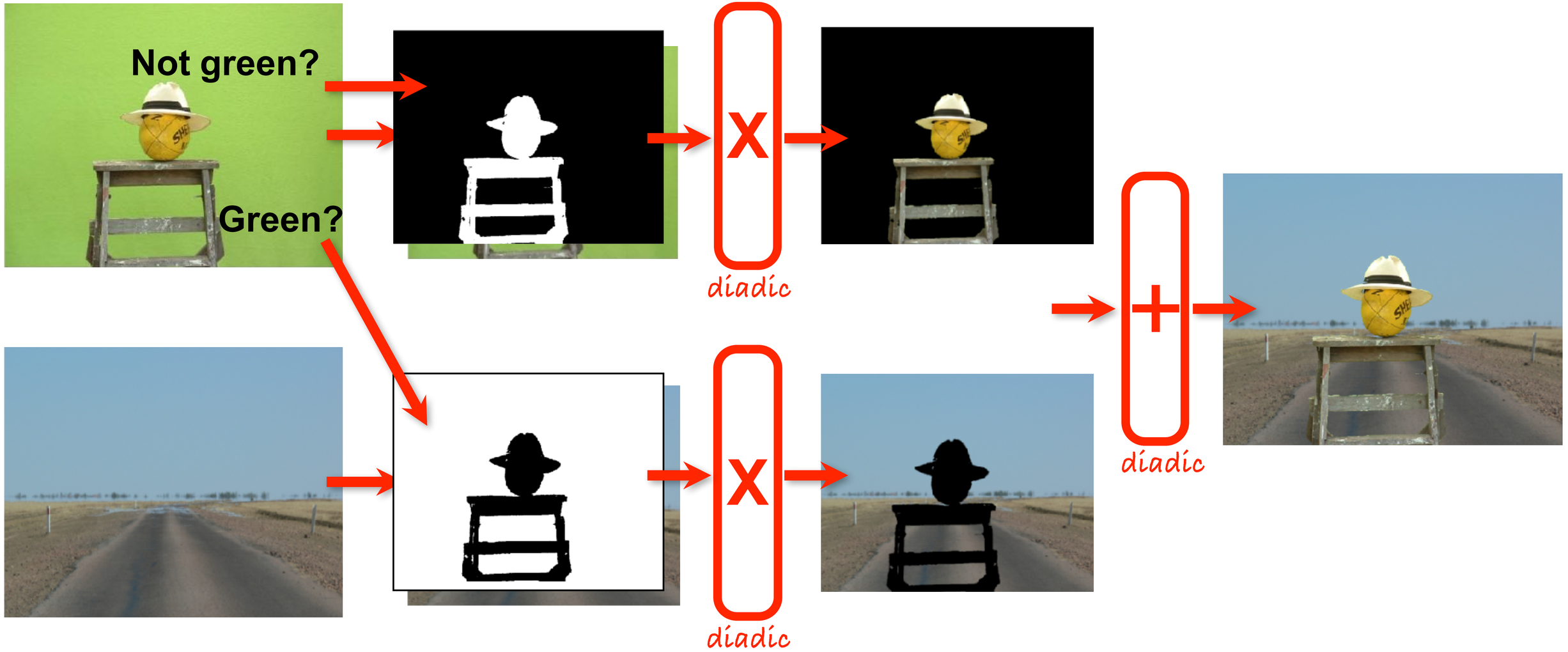
- Pixel by pixel difference highlights image change
- Be careful when subtracting uint8 types


SONY

Green screen effect

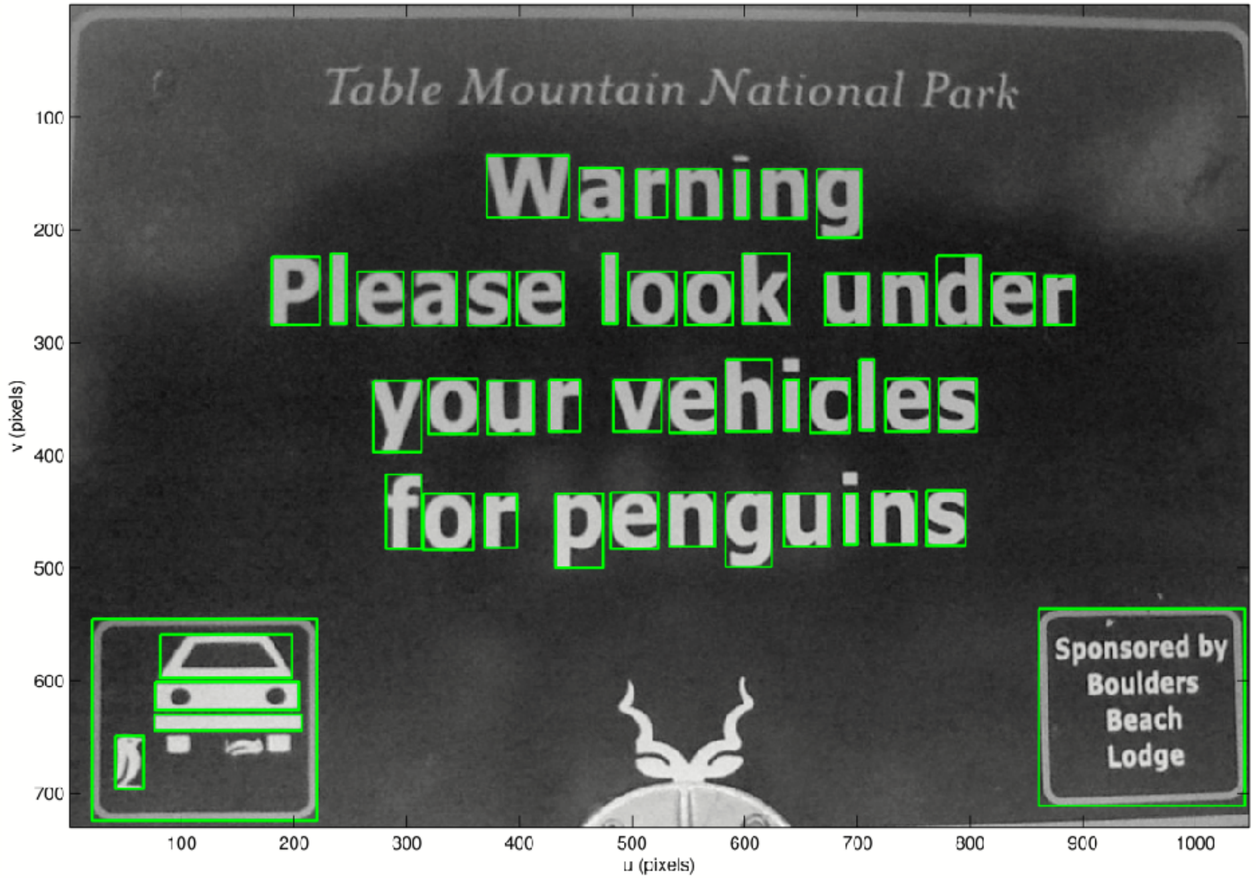
- Merge two images to create a composite image
- Green pixels are substituted for background pixels

Green screen effect



Section 2 Binary blobs

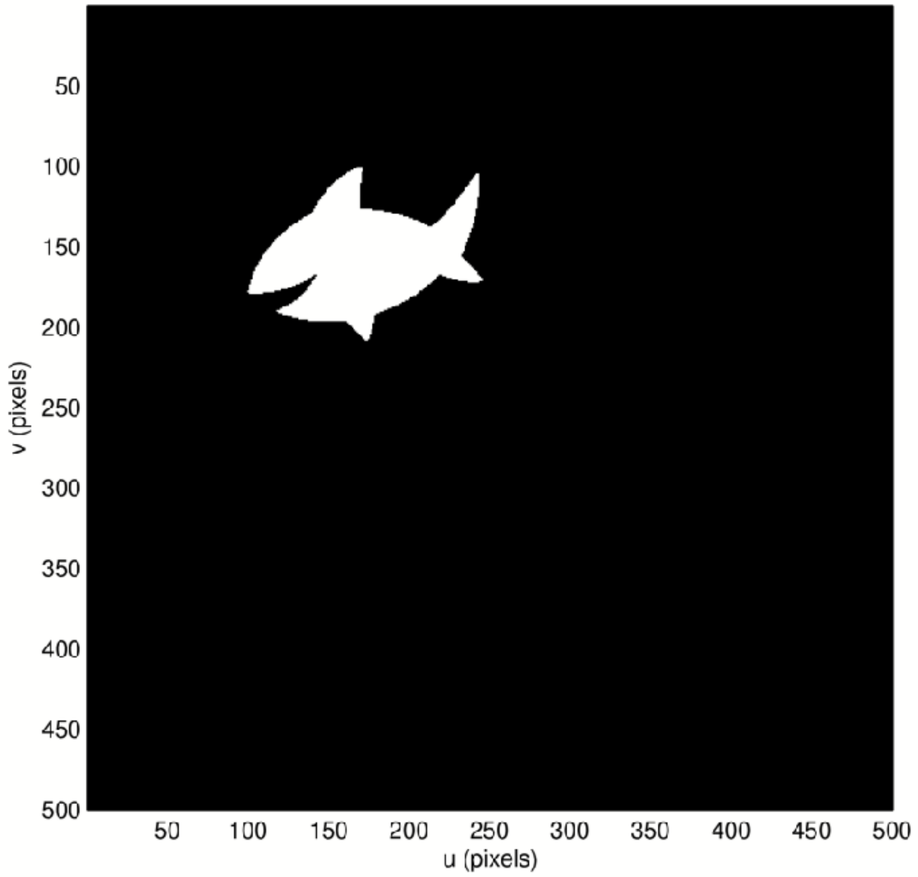
Region features



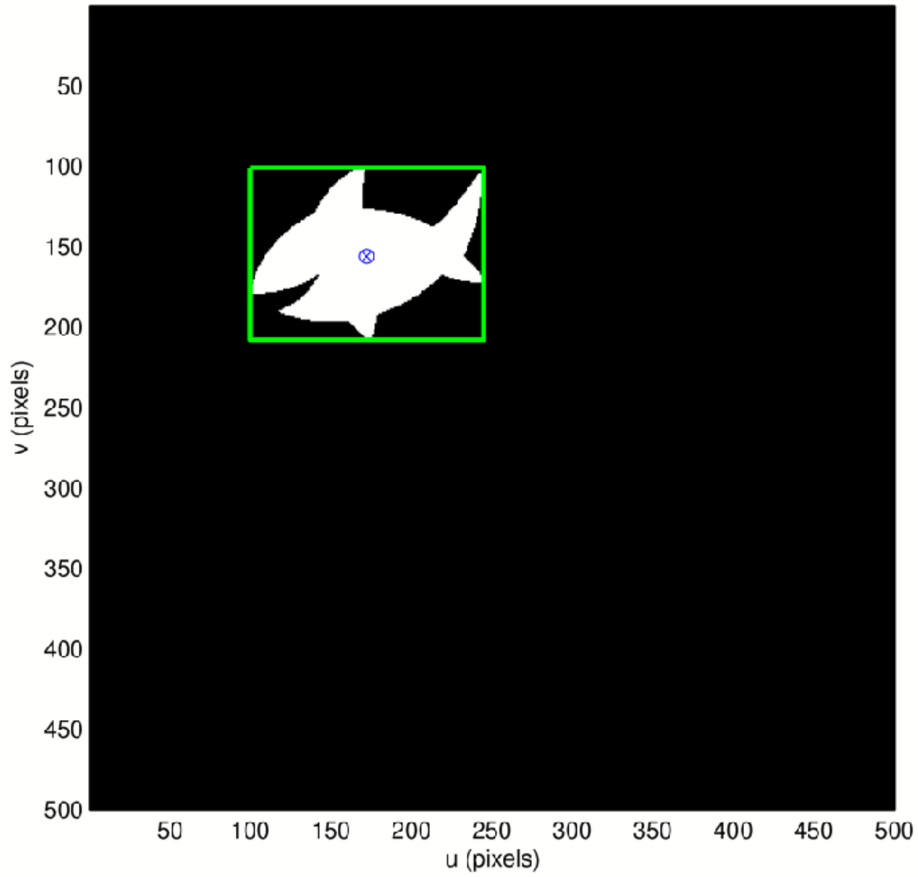
Where are the symbols on this sign?

Finding regions

http;//sweetclipart.com



a bunch of pixels



a region

Moments

The p-qth moment of an image is given by $m_{pq} = \sum_{(u,v) \in \mathbf{I}} u^p v^q \mathbf{I}[u, v]$

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

The zeroth (p=q=0) moment of an image is simply $m_{00} = \sum_{(u,v) \in \mathbf{I}} \mathbf{I}[u, v]$

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

First moments

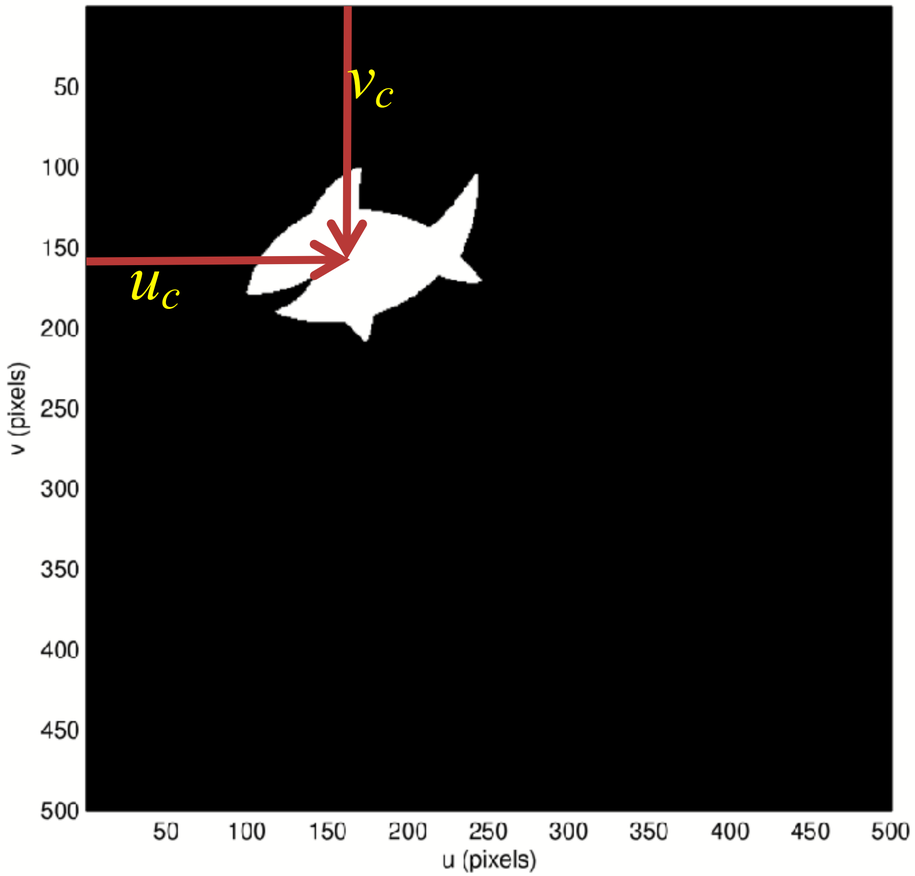
$$m_{10} = \sum_{(u,v)\in\mathbf{I}} u\mathbf{I}[u,v]$$

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

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

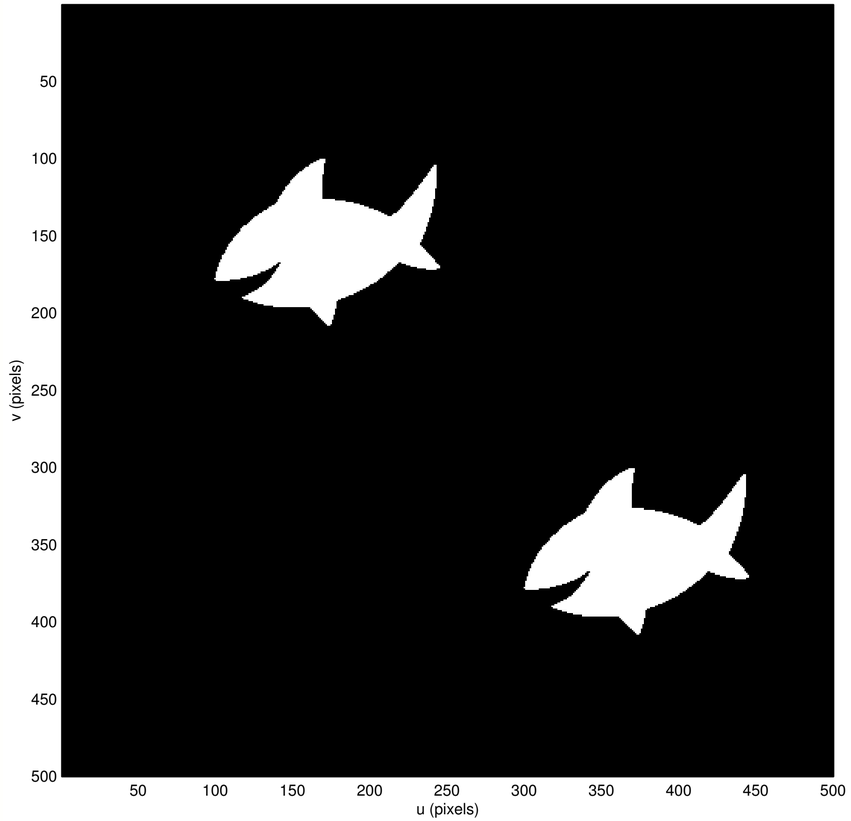
$$u_c = \frac{m_{10}}{m_{00}}, \quad v_c = \frac{m_{01}}{m_{00}}$$

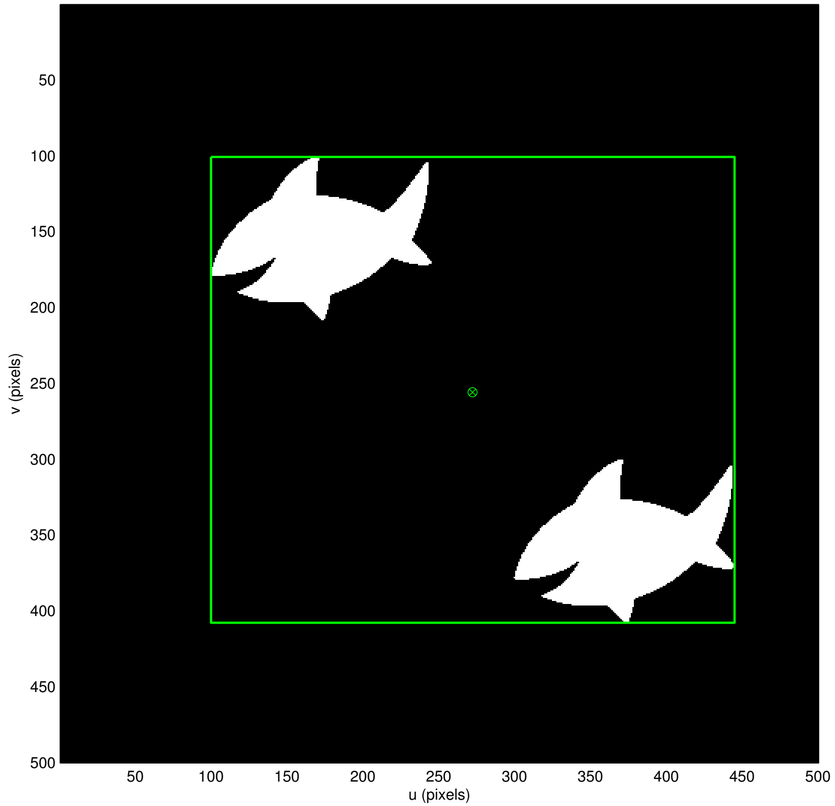
http;//sweetclipart.com



Multiple blobs

http;//sweetclipart.com

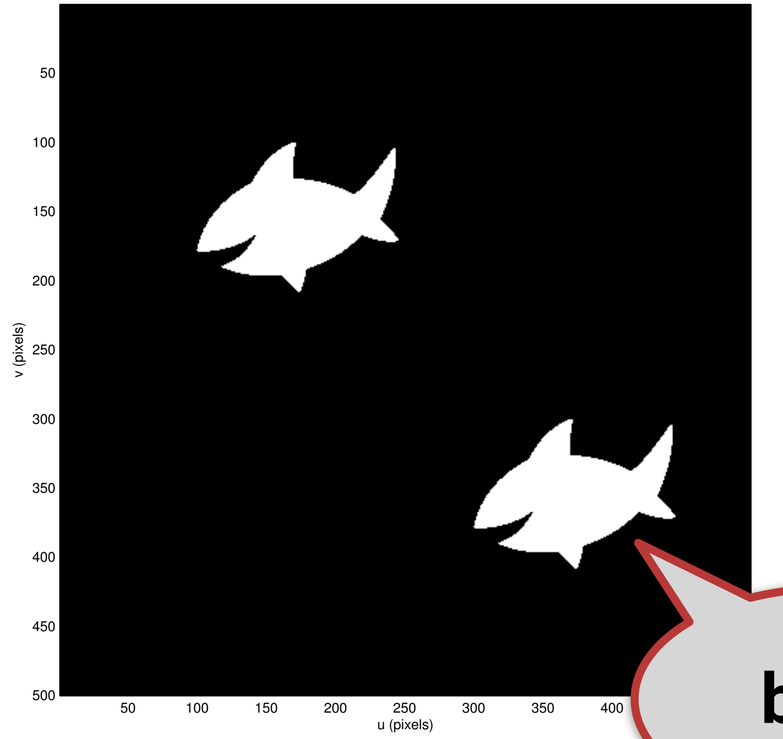




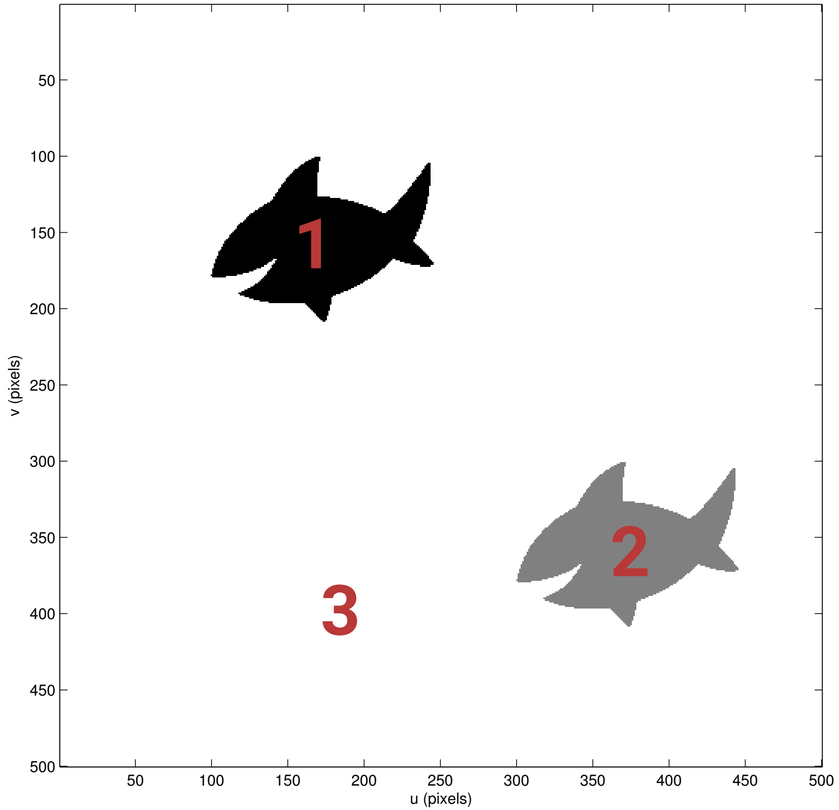
- We can see two distinct regions
- but the moment computation has no sense of this

Transform the image

http;//sweetclipart.com



blob



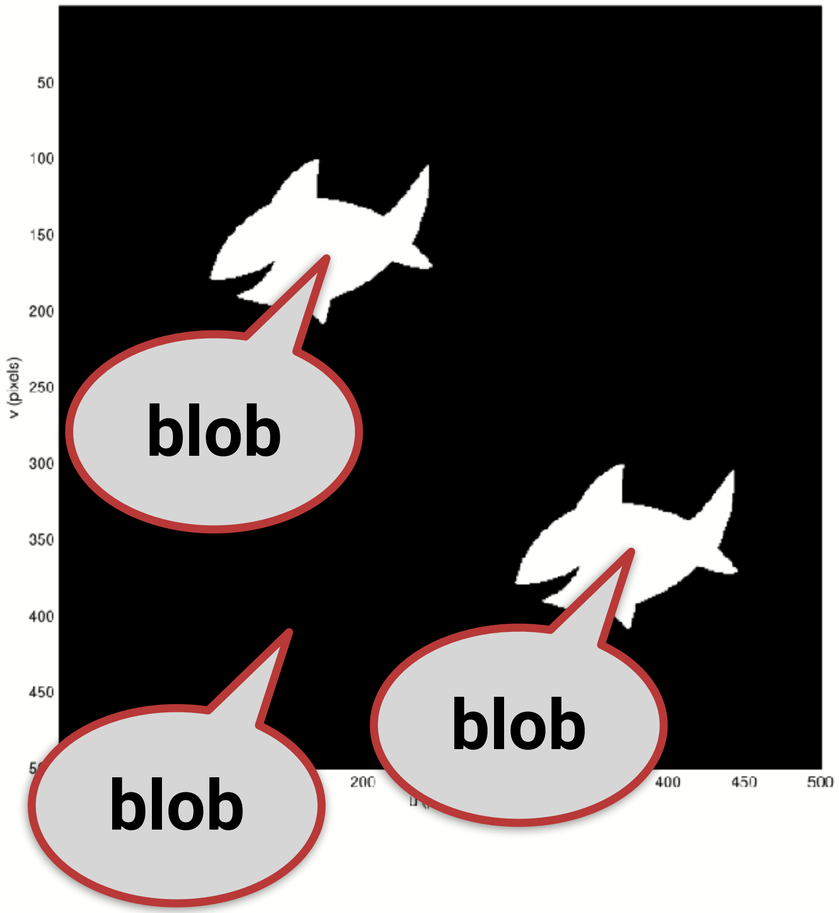
Label image

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

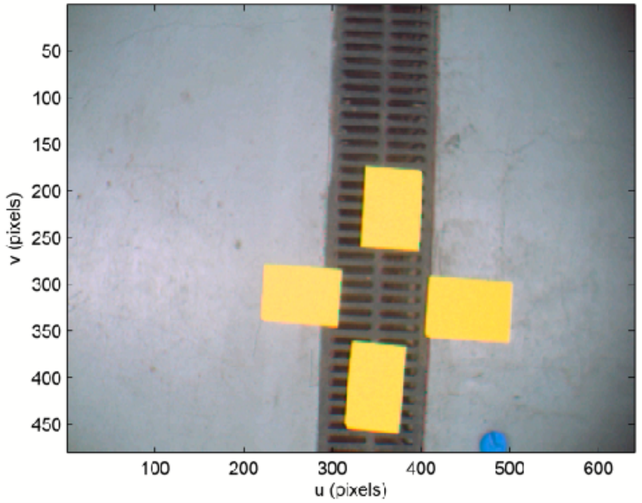
What is a blob?

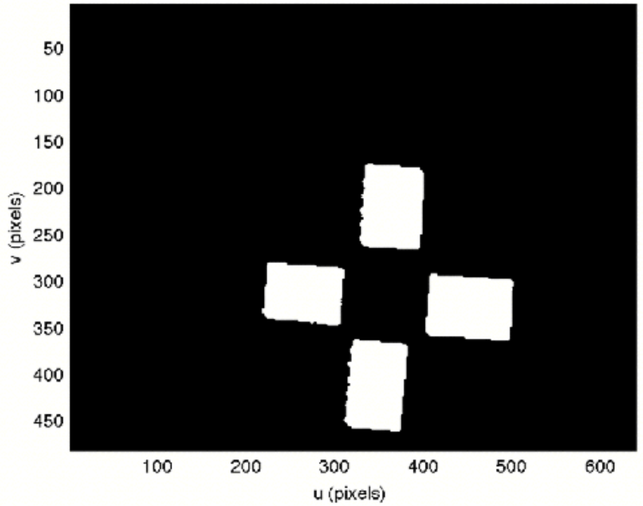
- aka region, connected component
- a set of **contiguous** pixels of the same color (value)

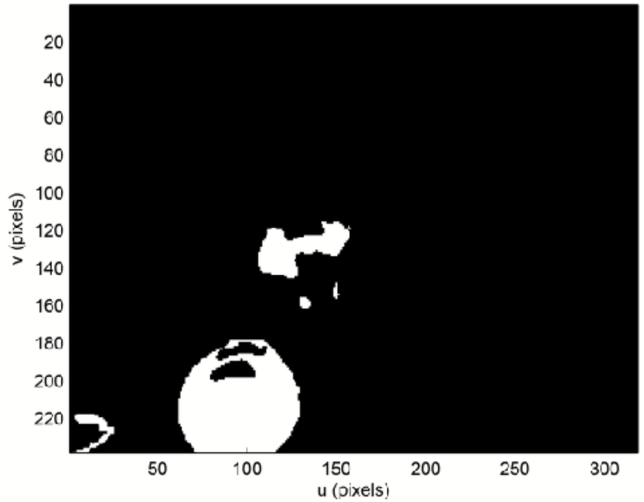
http;//sweetclipart.com



Pixel classification





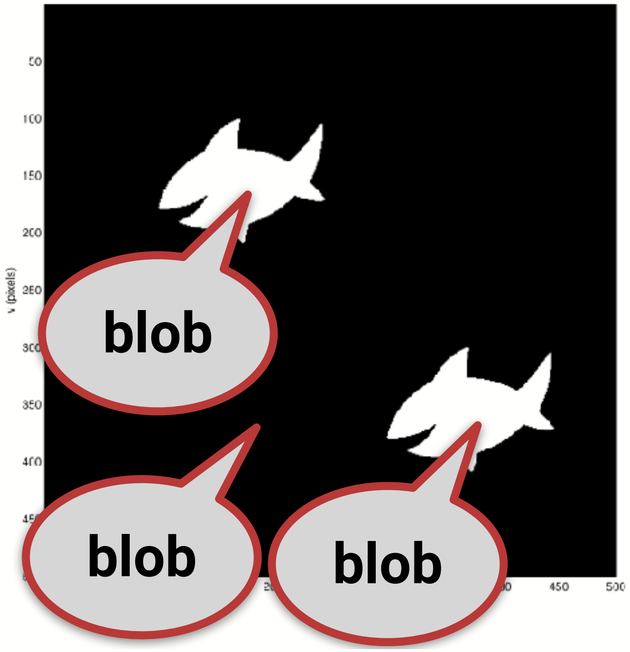


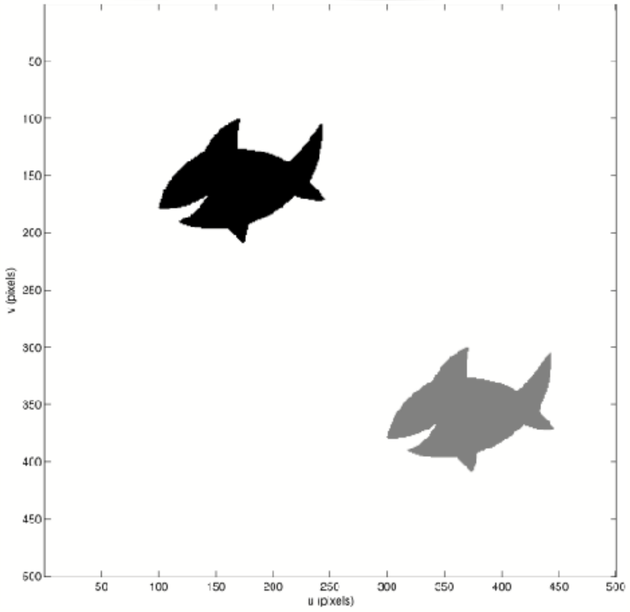
All images with kind permission of **Springer Science+Business Media**

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

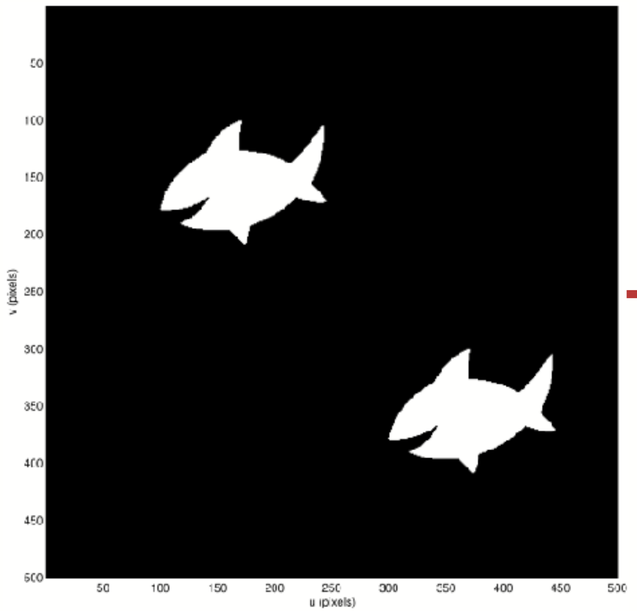
http;//sweetclipart.com



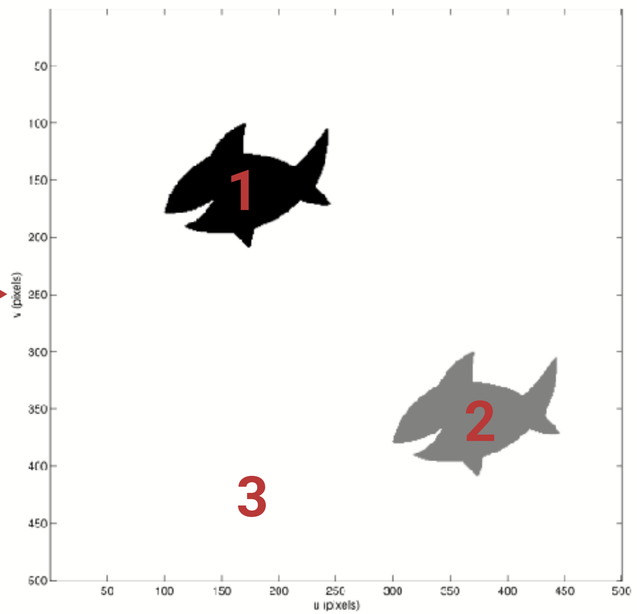


Select individual blobs

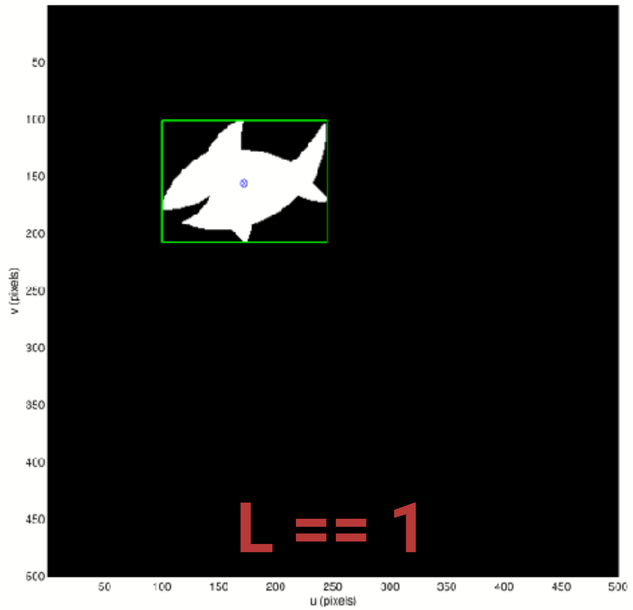
http;//sweetclipart.com

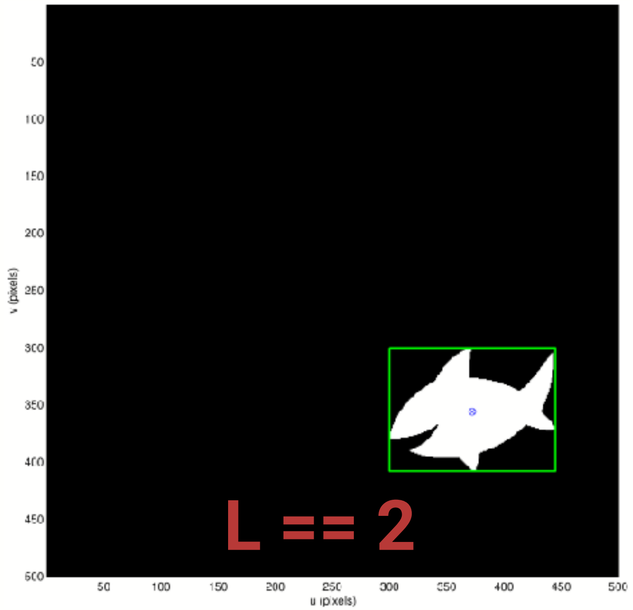


Original binary image



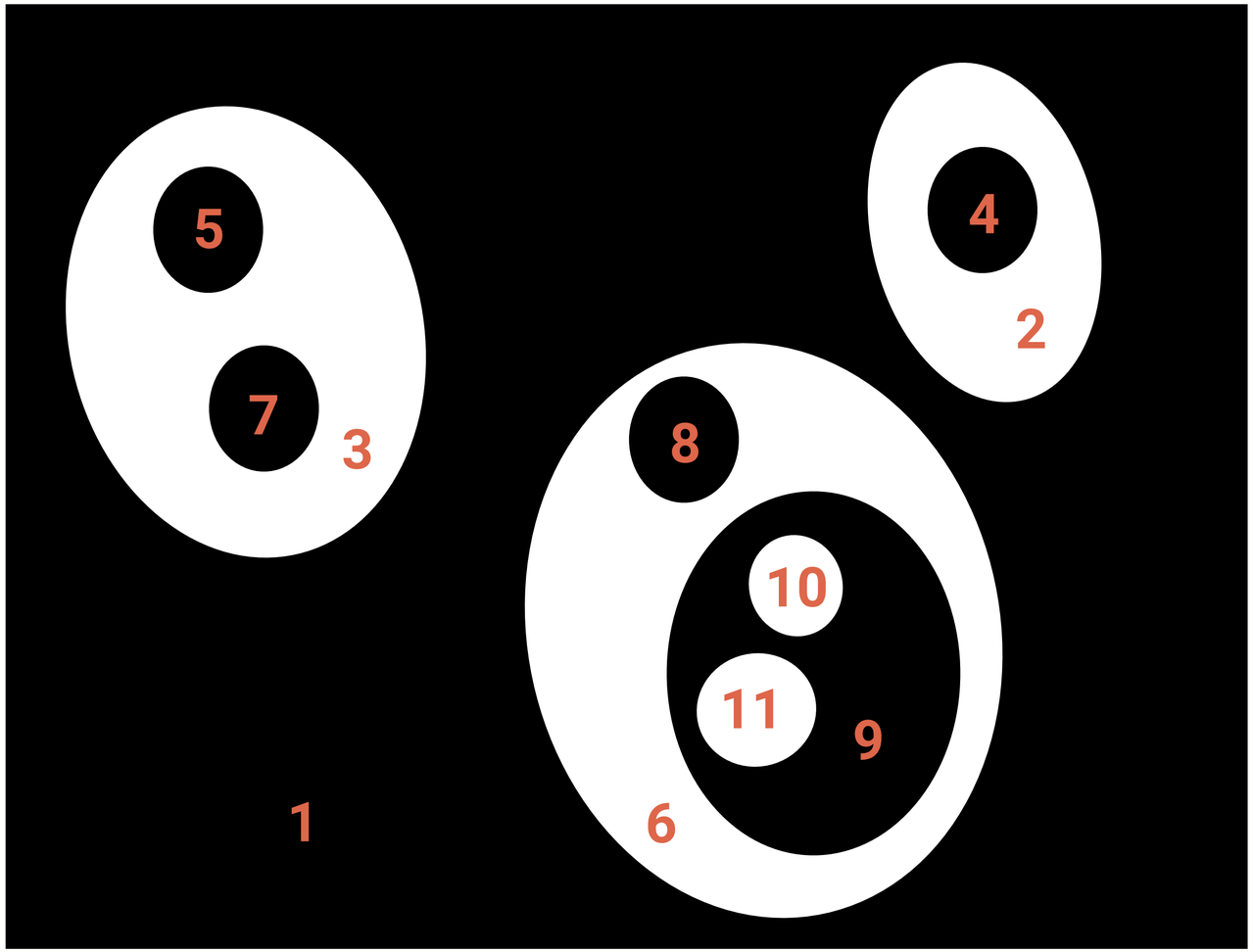
L = label image

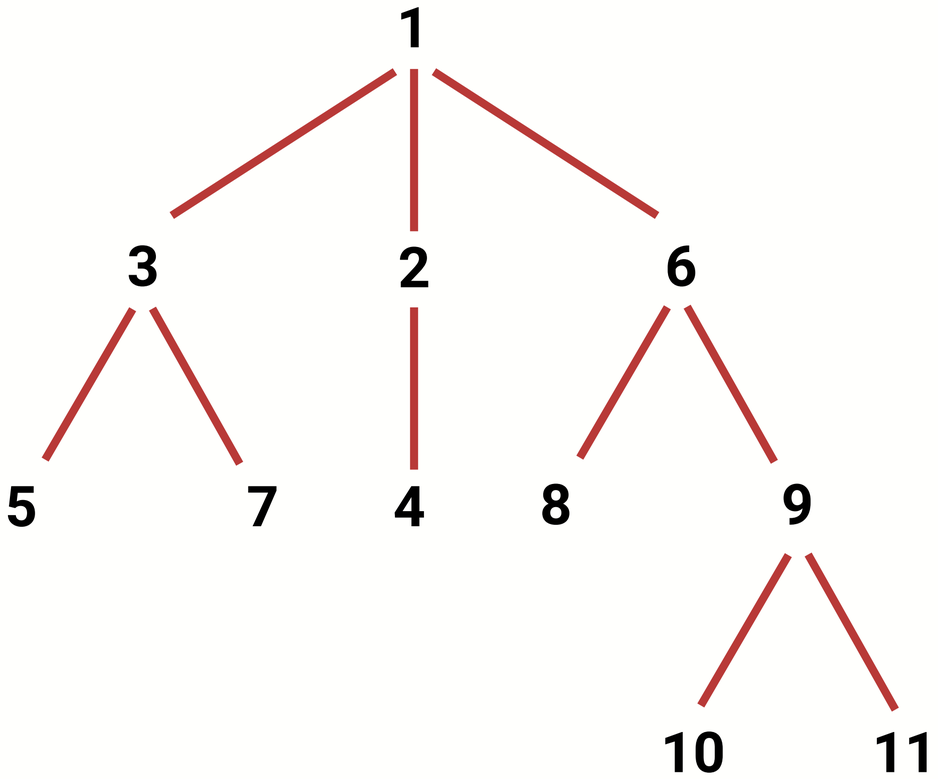




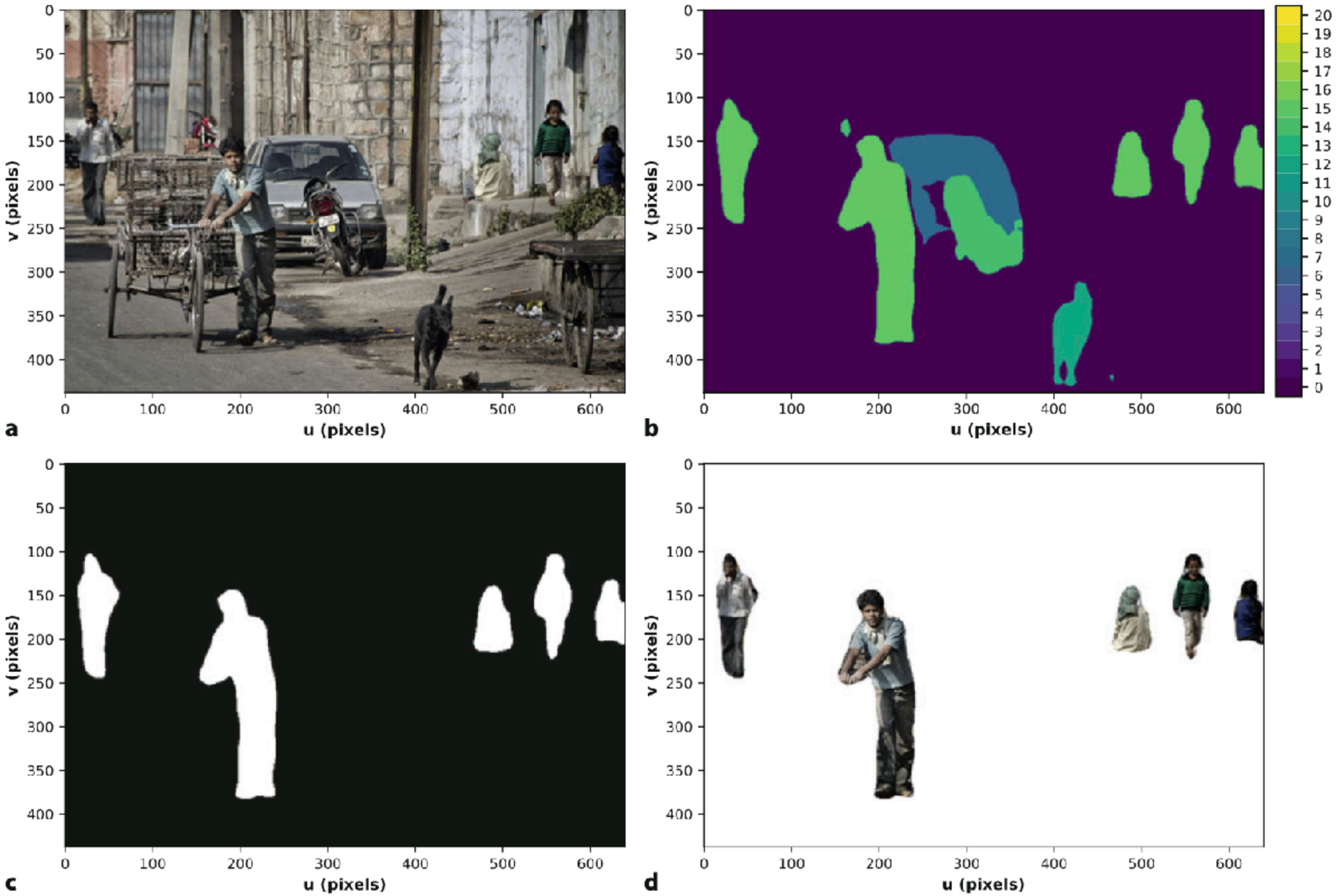
Individual blob binary image

Blob hierarchy



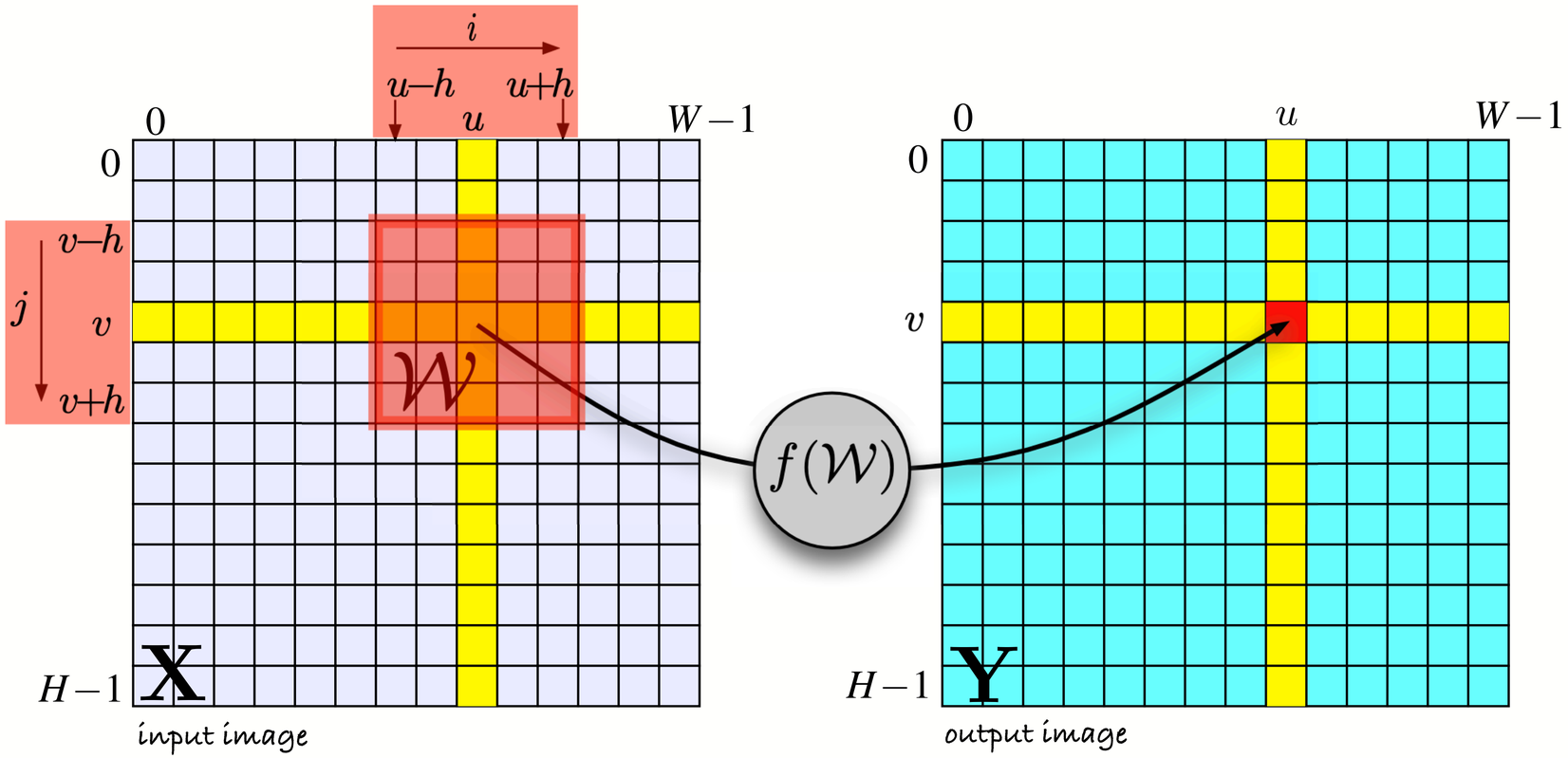


Deep networks allow us to classify pixels based on their semantic meaning (semantic segmentation) even if they have a range of brightness and color



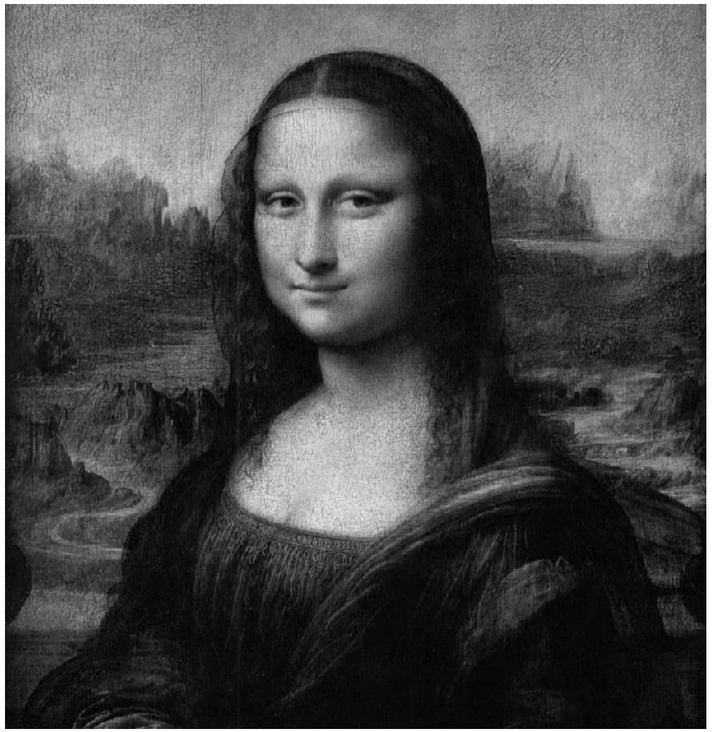
1	background
2	aeroplane
3	bicycle
4	bird
5	boat
6	bottle
7	bus
8	car
9	cat
10	chair
11	cow
12	diningtable
13	dog
14	horse
15	motorbike
16	person
17	pottedplant
18	sheep
19	sofa
20	train
21	tvmonitor

Spatial operators



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

f(·) could be an average

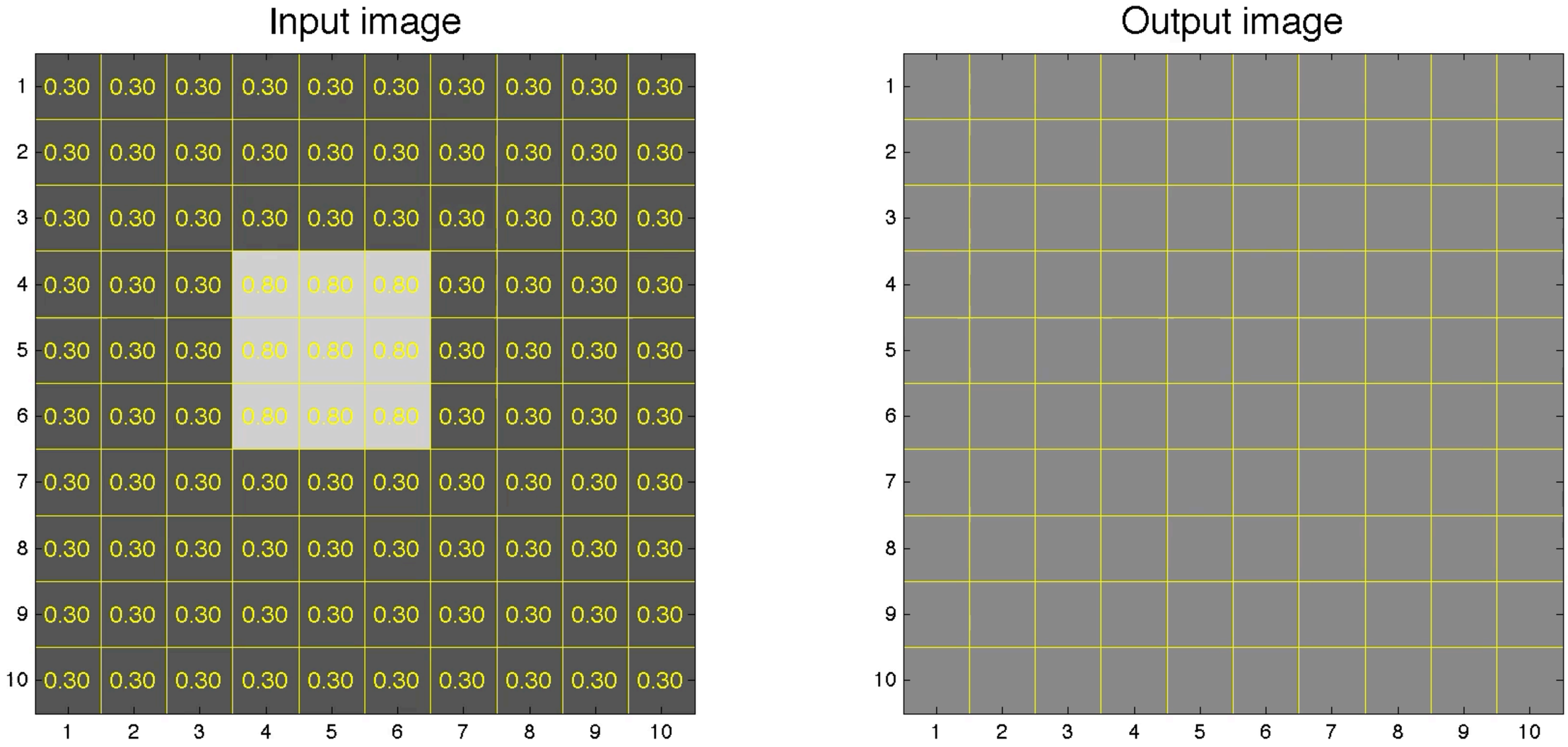


⇦Original image

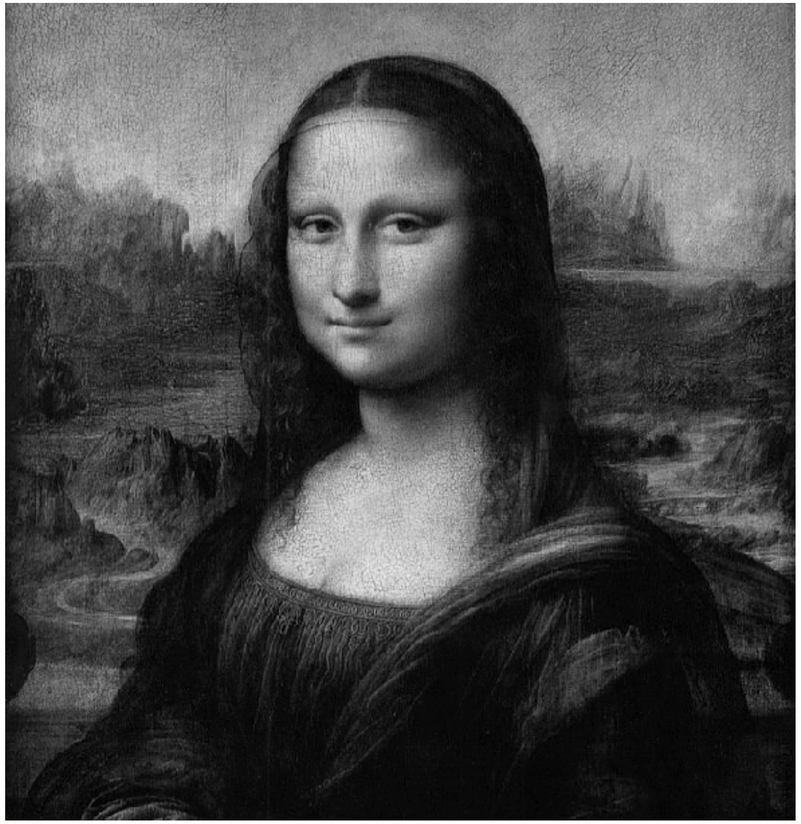
Average each pixel over a 7x7 window⇨

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

Moving window process



Effect of window size



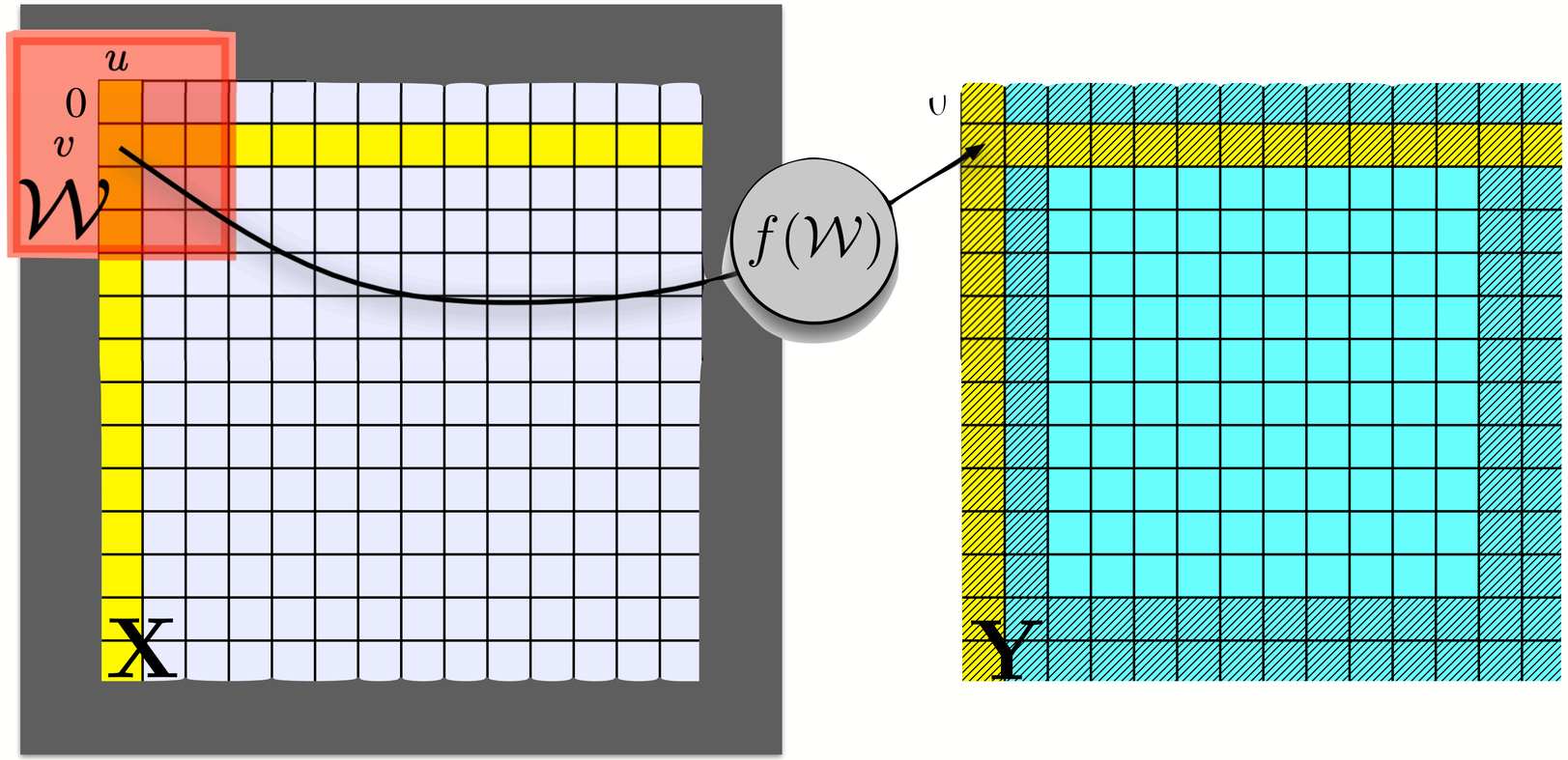
Original image

Average each pixel over a 7x7 window

Average each pixel over a 21x 21 window

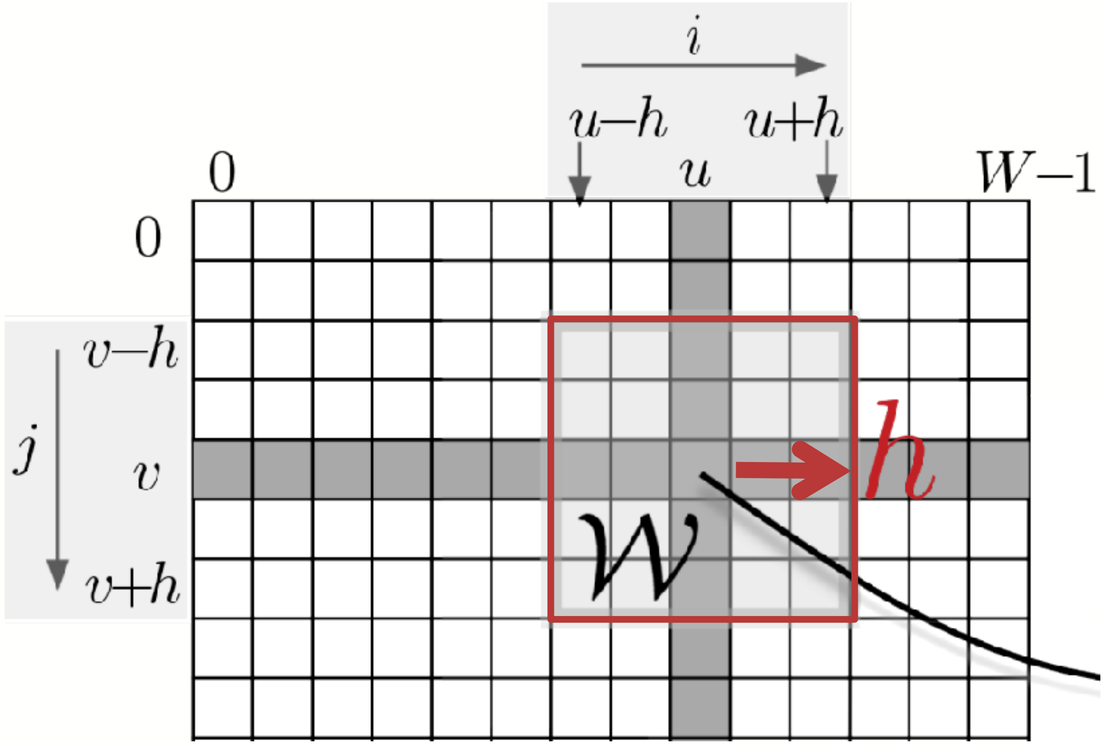
- Features smaller than the window size will be strongly attenuated
  - ↪ we say the image is blurry, fuzzy, lower resolution etc.

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

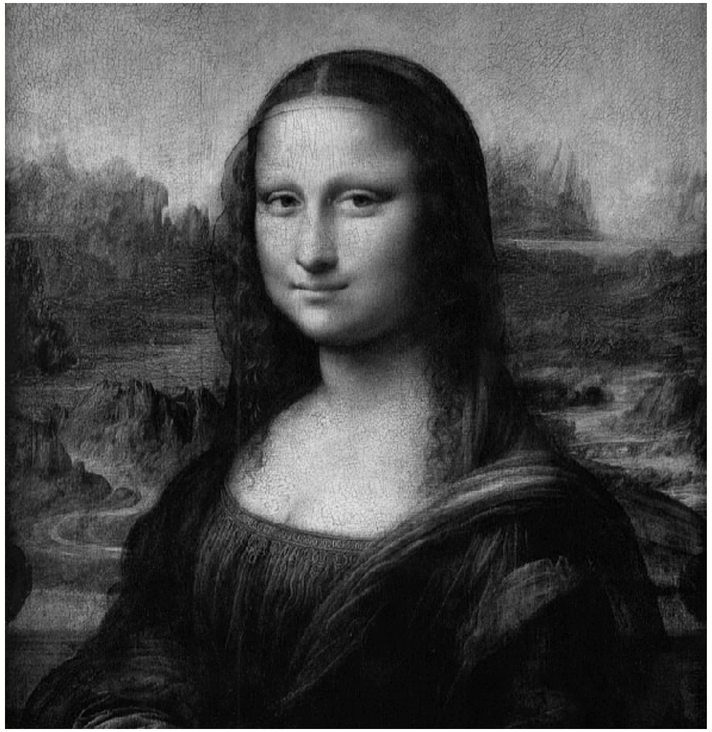
Window size is always odd



With kind permission of **Springer Science+Business Media**

- 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



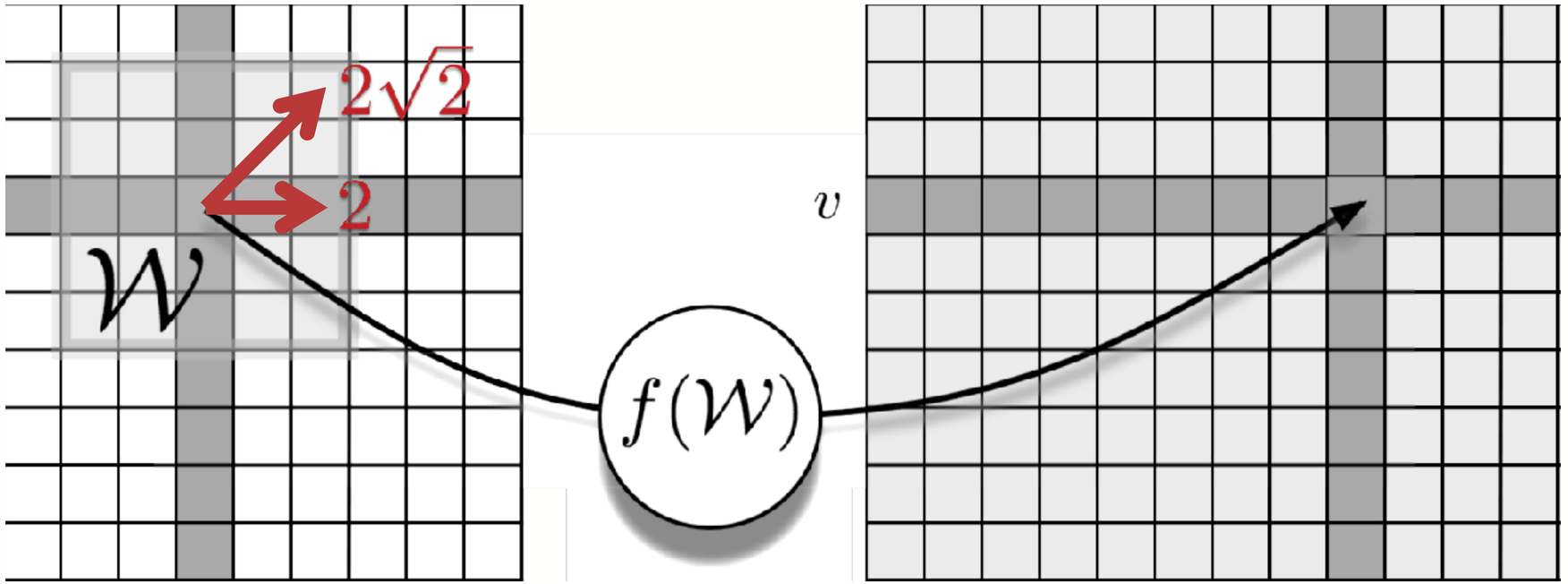
⇦Original image

Average each pixel over a 21x 21 window⇨

- Can lead to *ringing*

↪ faint vertical & horizontal lines are introduced

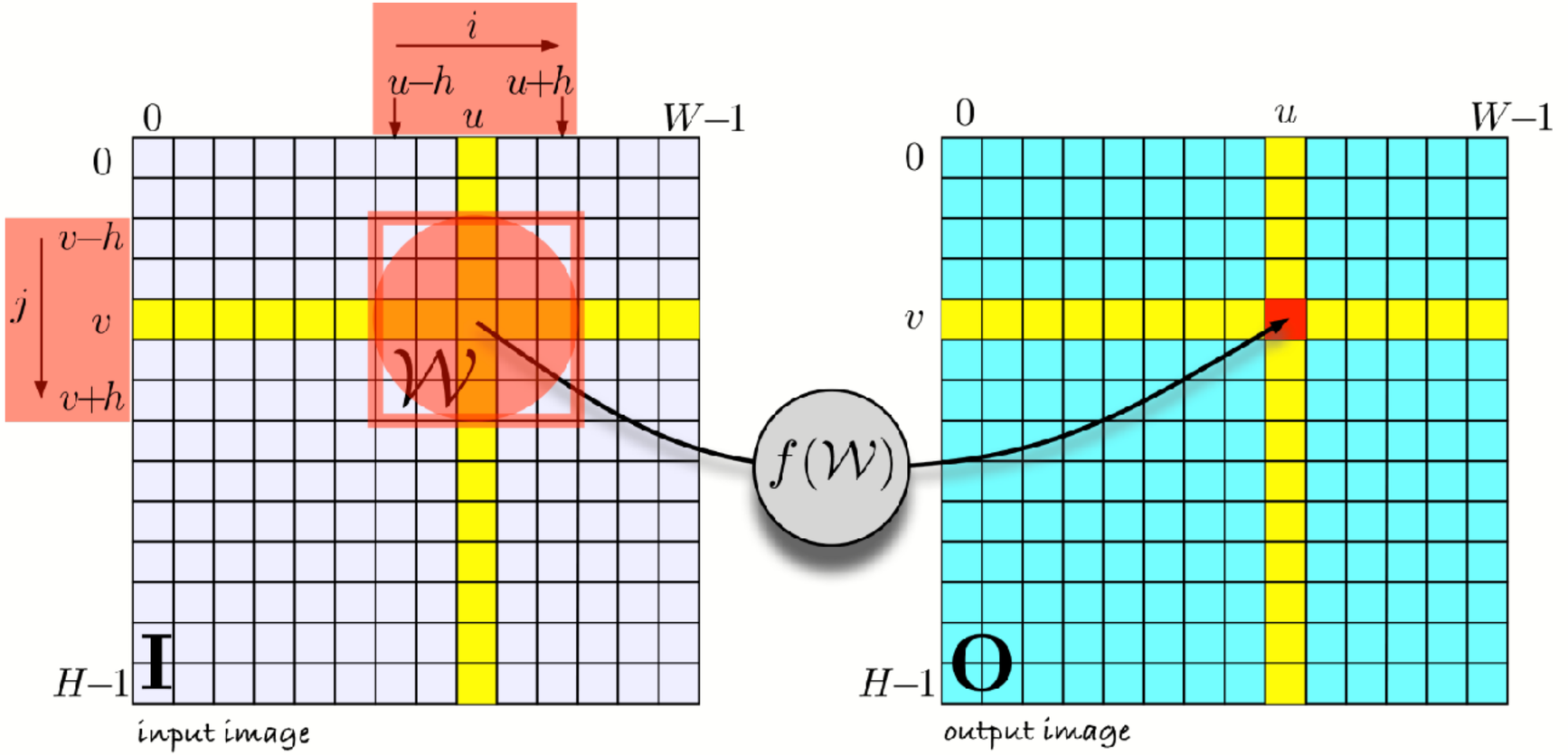
Averaging over a square is not isotropic



With kind permission of **Springer Science+Business Media**

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

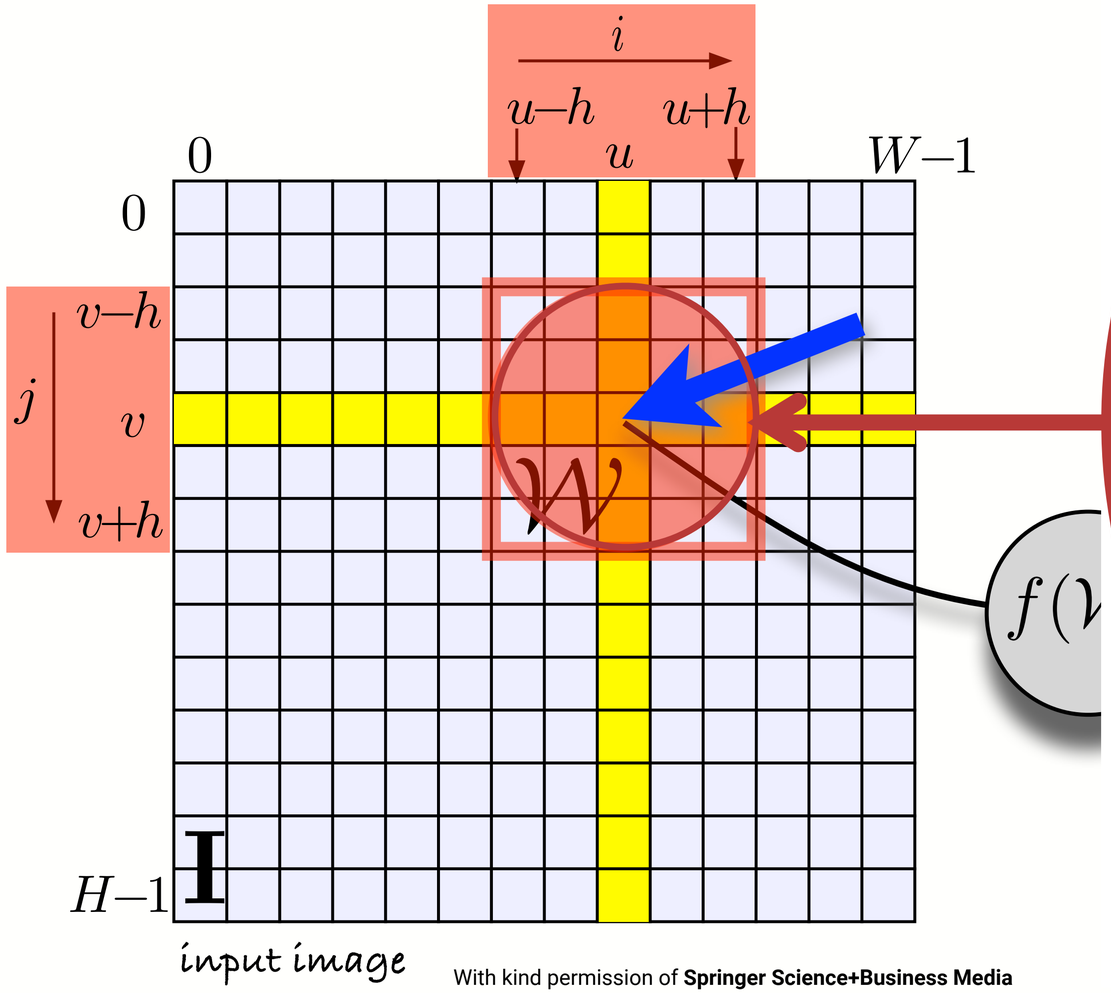
Going isotropic



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

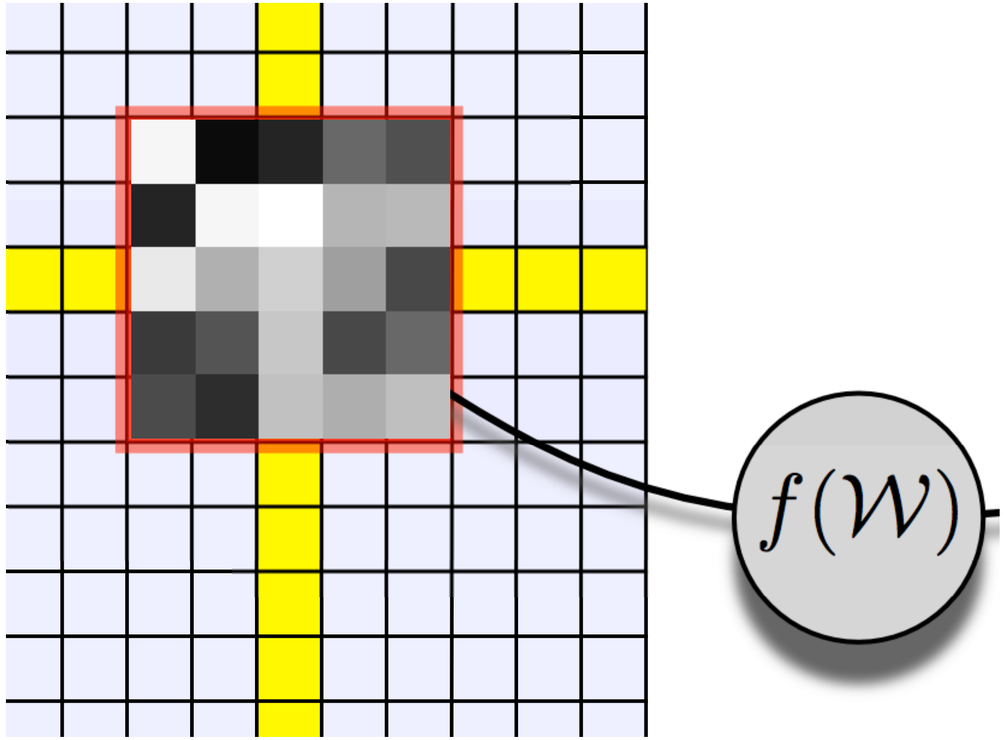


With kind permission of **Springer Science+Business Media**

0.14	0.77	0.98	0.77	0.14
0.77	1.00	1.00	1.00	0.77
0.98	1.00	1.00	1.00	0.98
0.77	1.00	1.00	1.00	0.77
0.14	0.77	0.98	0.77	0.14

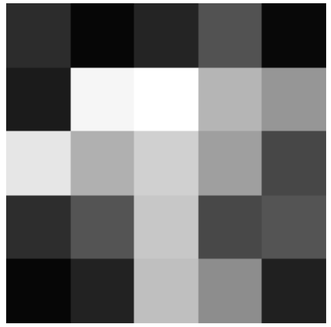
- Circle of radius 2.5 pixels

Weighted kernel



Hadamard product

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



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

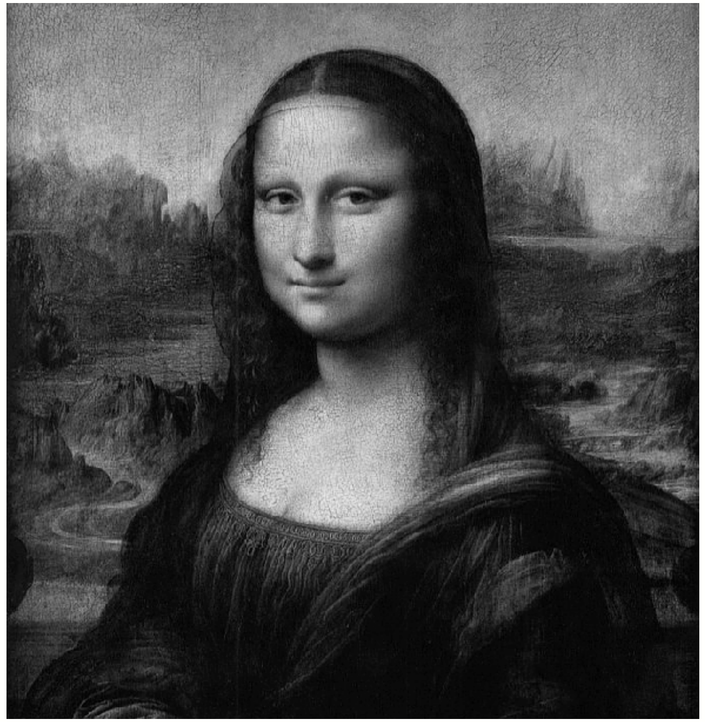
- The scale factor is $S = \Sigma \left| \mathbf{K}_{i,j} \right|$

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 $S = \Sigma \left|\mathbf{K}_{i,j}\right|$
- Typically make S = 1 to keep grey levels the same as the input image

Simple averaging is also a kernel



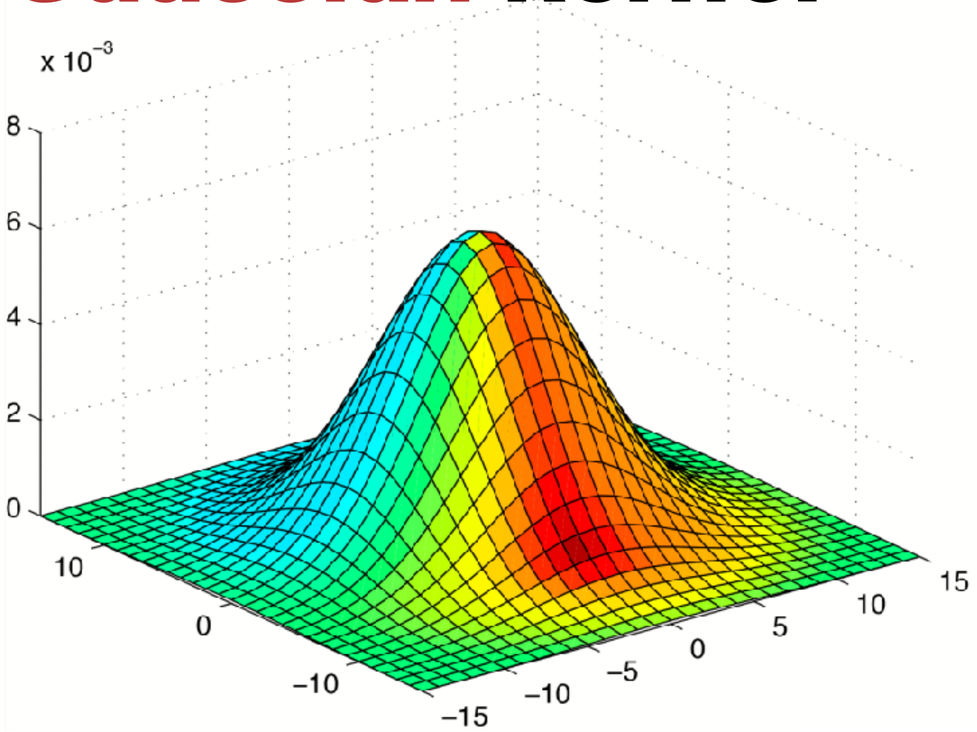
Original image

Average each pixel over a 7x7 window

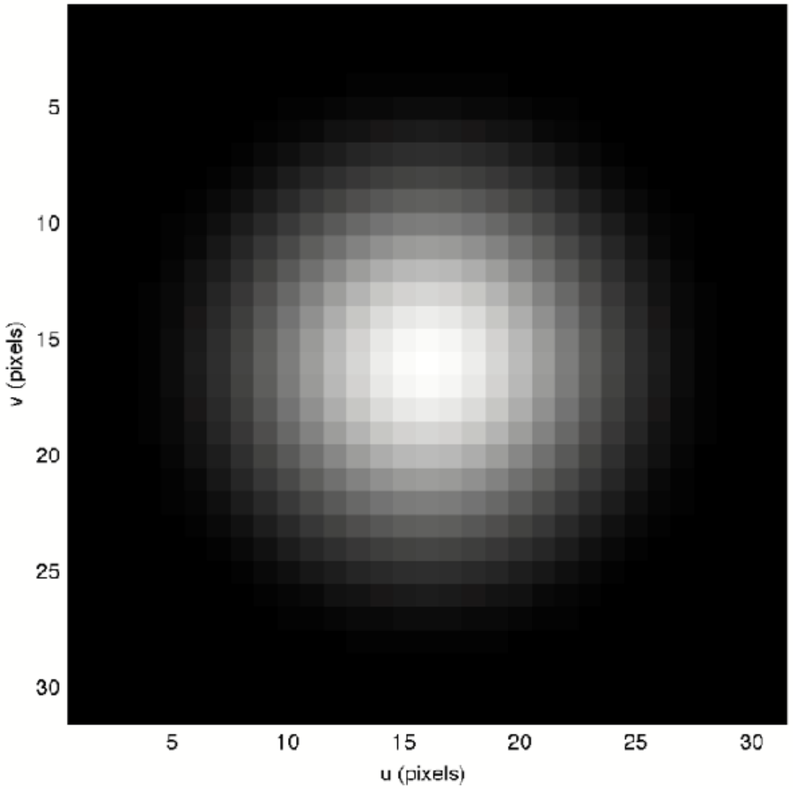
$$\mathbf{K} = \frac{1}{49}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

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

Gaussian kernel



With kind permission of **Springer Science+Business Media**



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

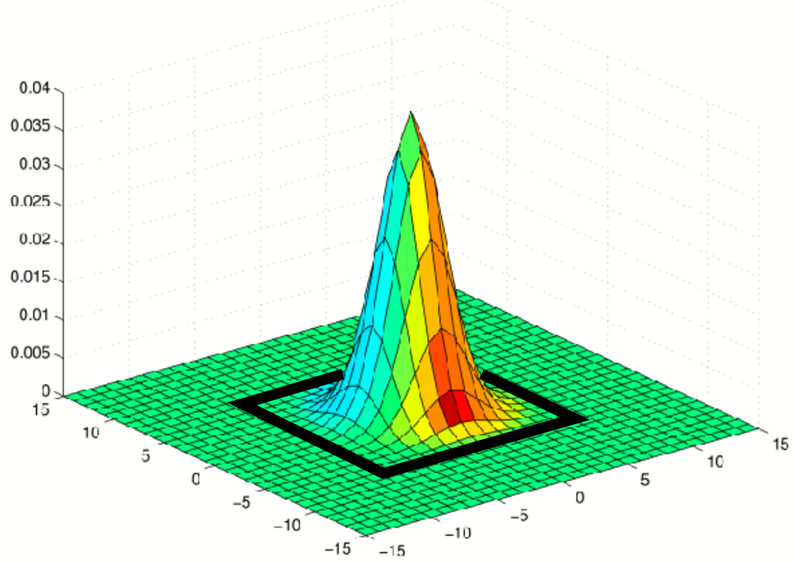
- Isotropic
- Decreasing weight away from centre

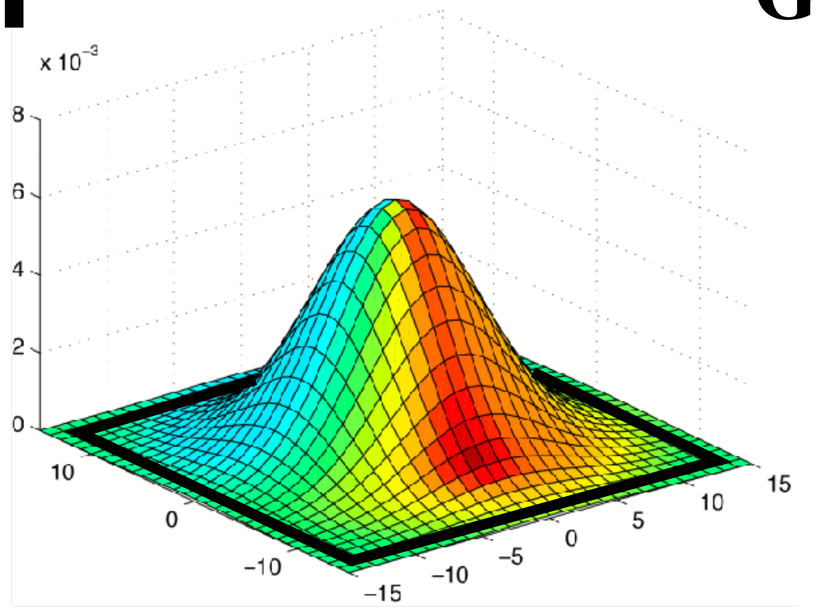
Carl-Friedrich Gauss
1777–1855

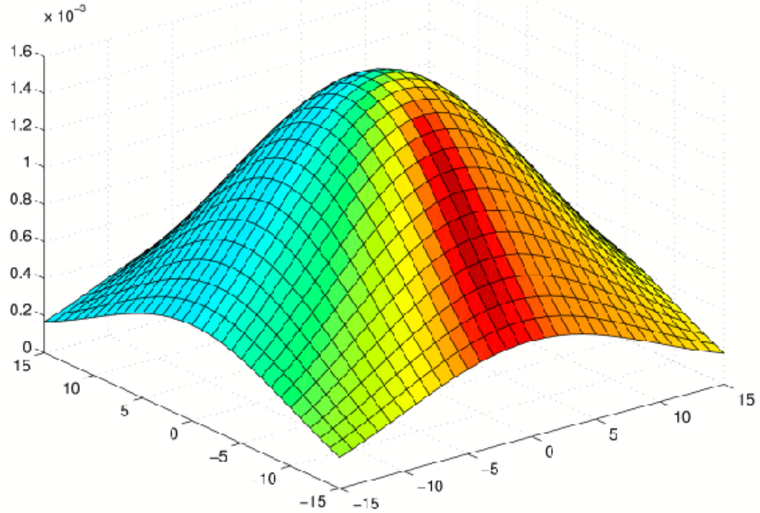
By **Gottlieb Biermann A. Wittmann** (photo) [Public domain], via Wikimedia Commons

Gaussian width

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







$\sigma = 2$

```
Kernel.Gauss(2, 15)
```

$\sigma = 5$

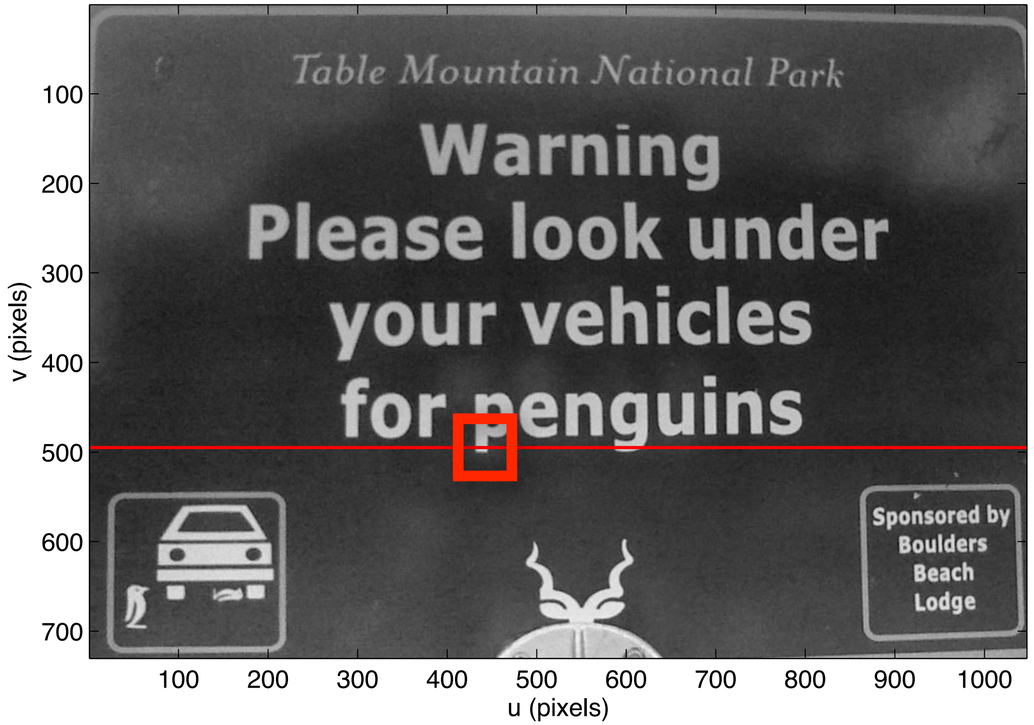
```
Kernel.Gauss(5, 15)
```

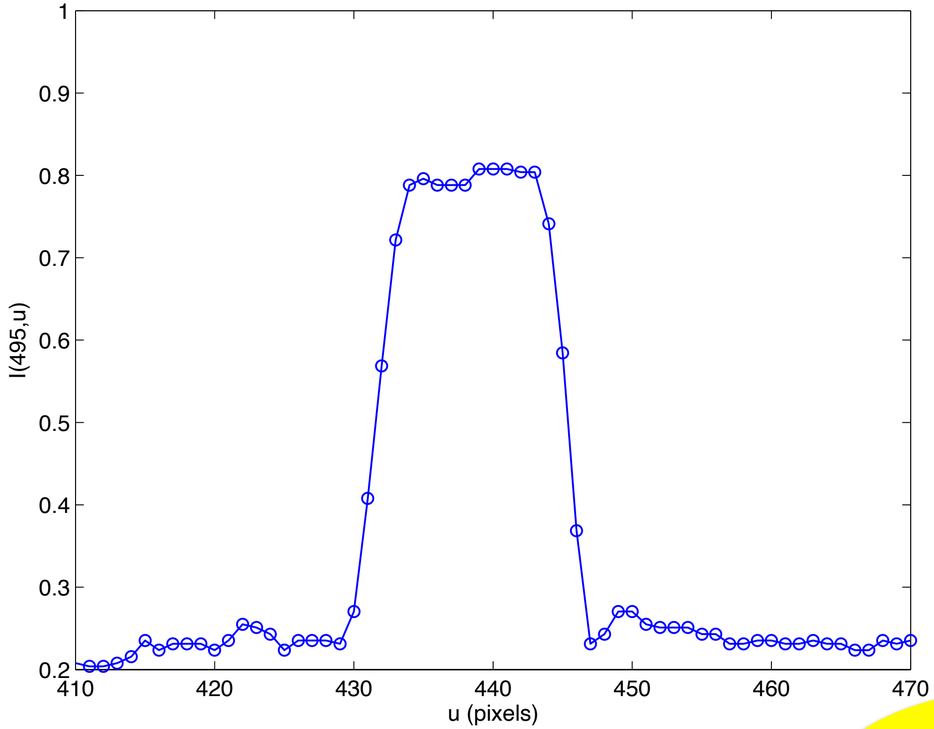
$\sigma = 10$

```
Kernel.Gauss(10, 15)
```

- Choose the size of the square kernel to fit the Gaussian
- Rule of thumb h=3σ

The intensity function





- Horizontal gradient can be approximated:

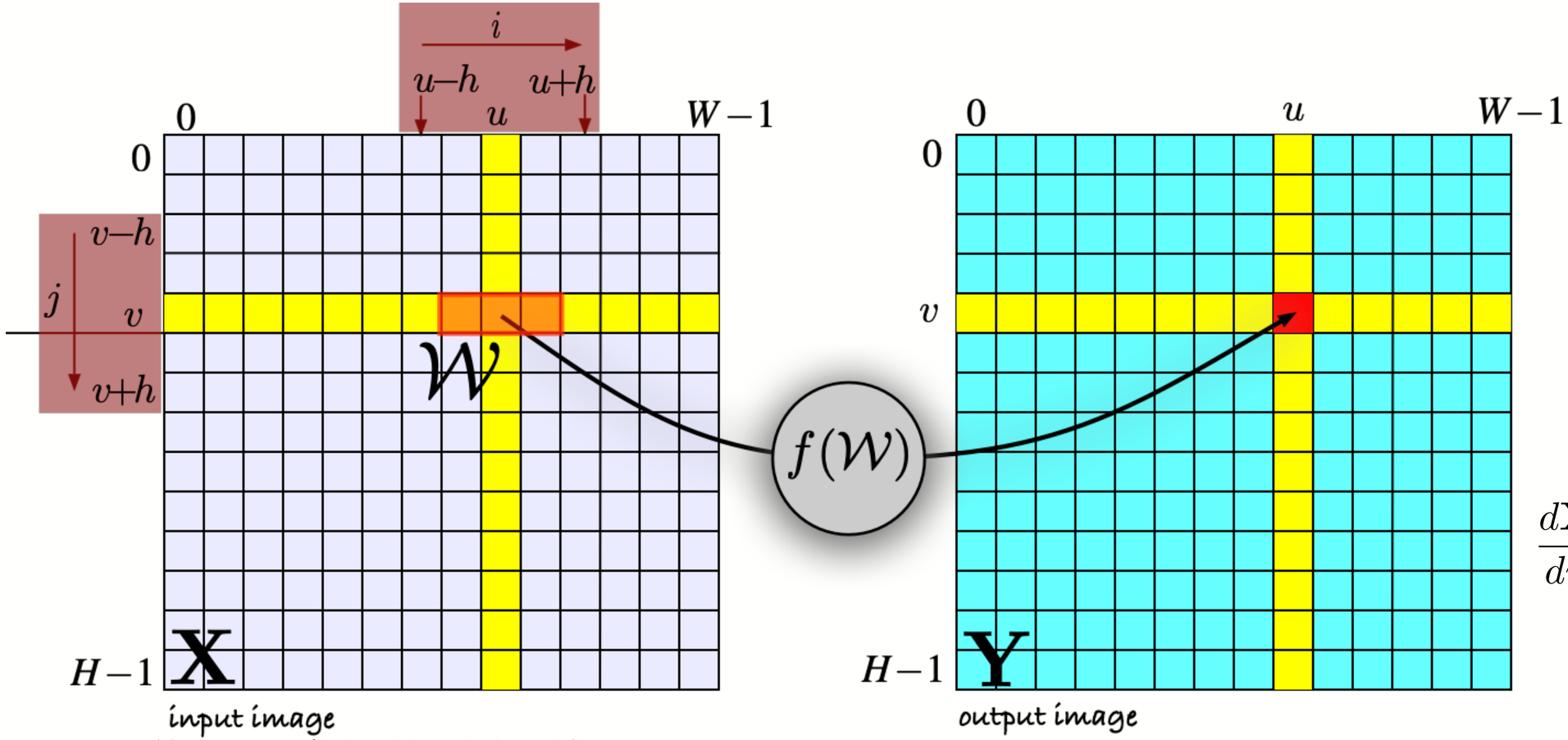
$$\frac{dI}{du} \approx I(u+1) - I(u)$$

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

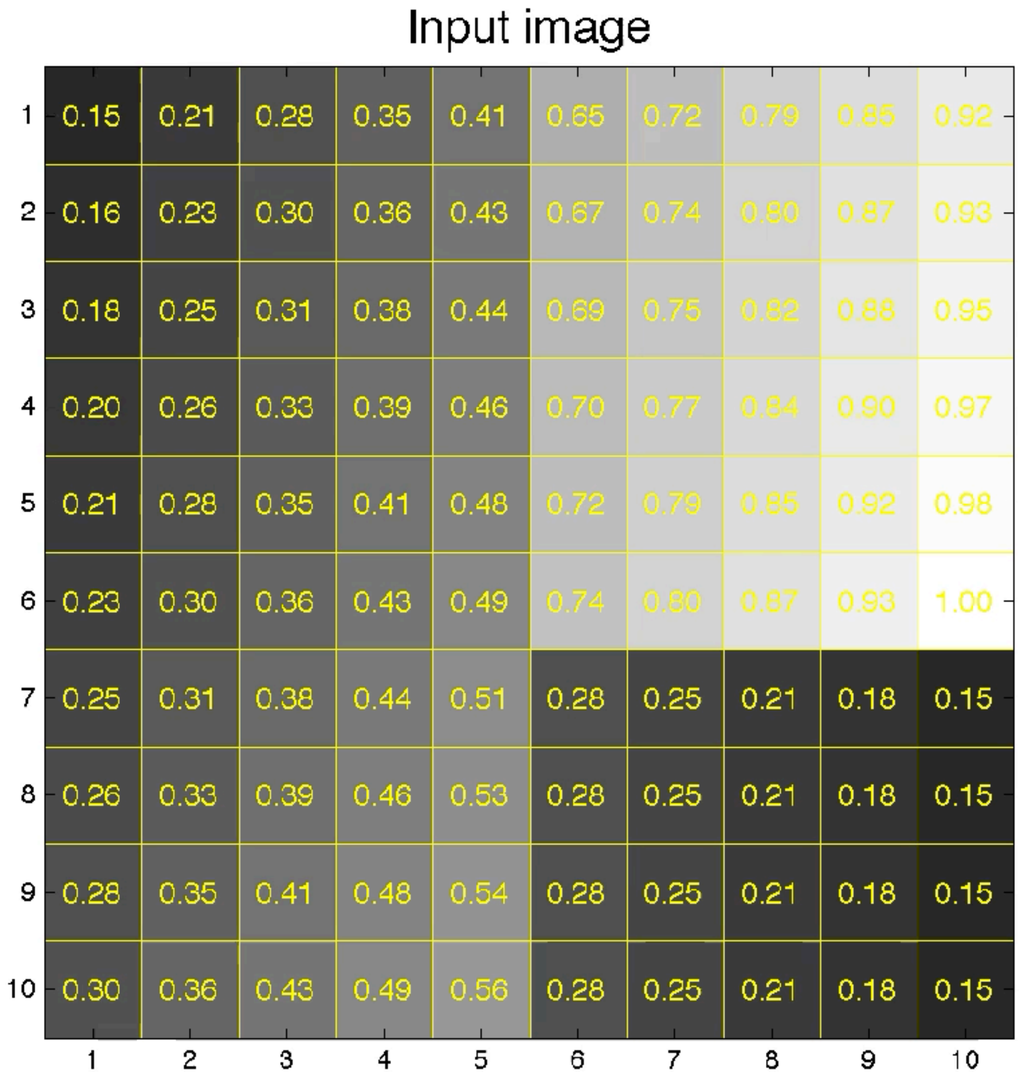


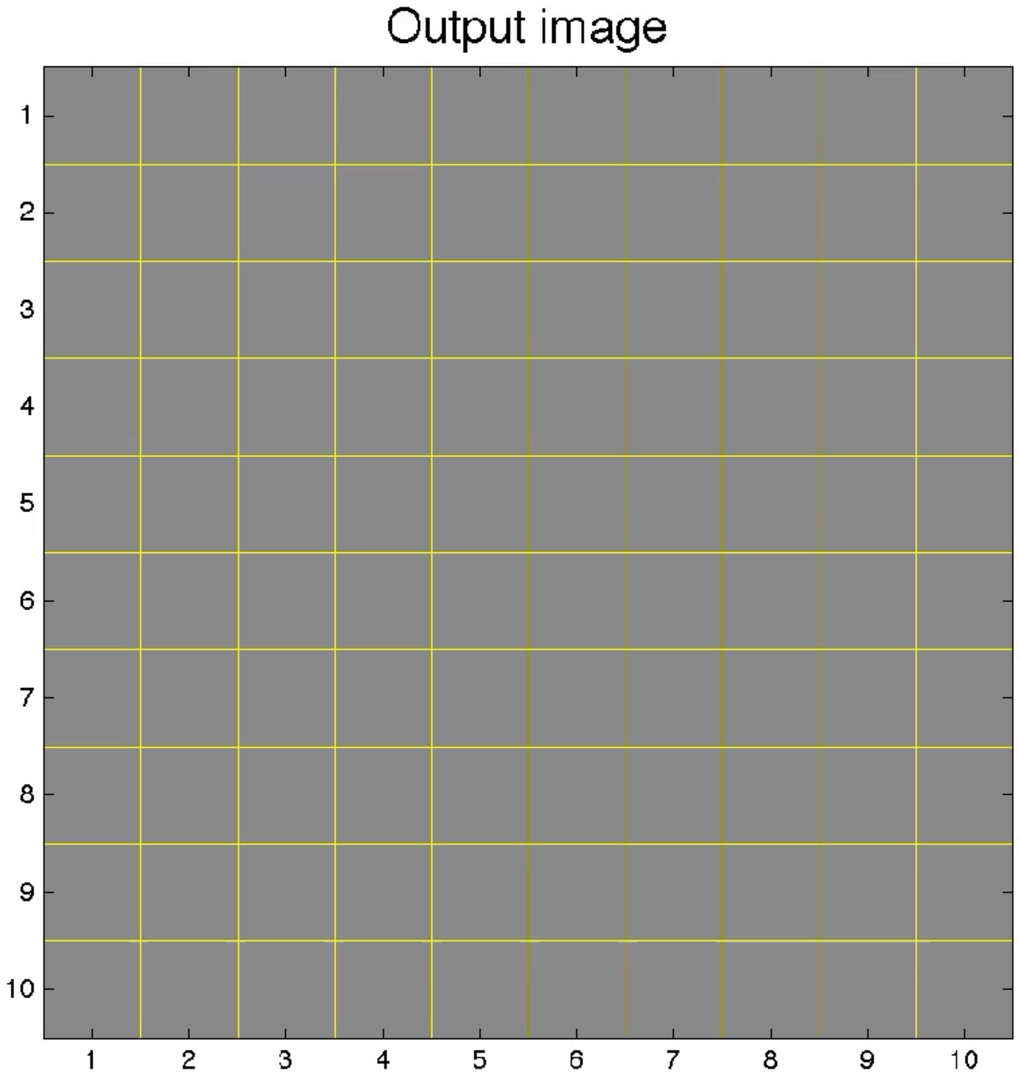
With kind permission of **Springer Science+Business Media**

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

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

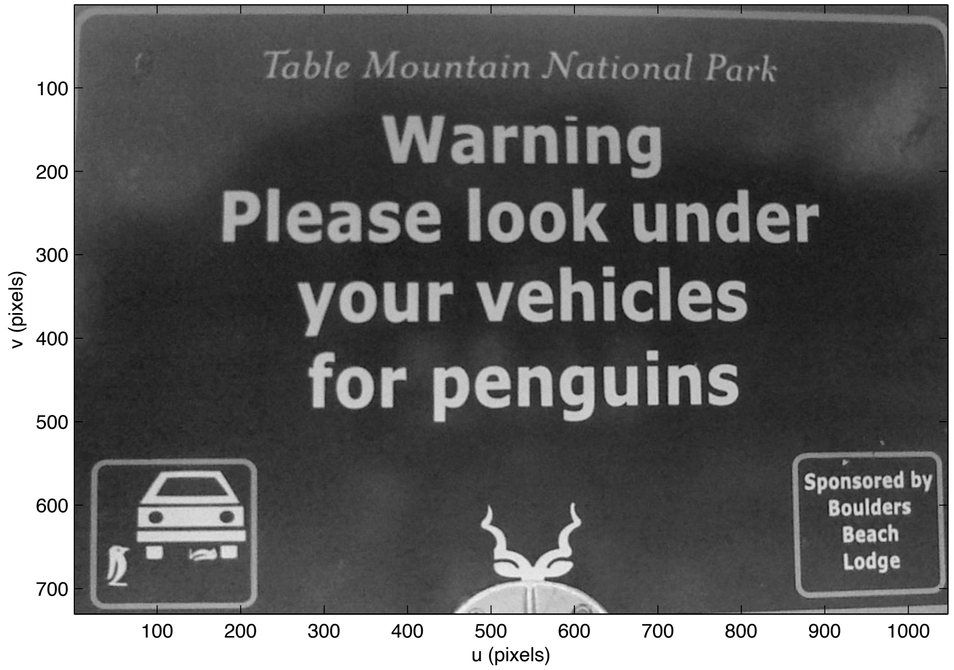
Convolution for edge detection

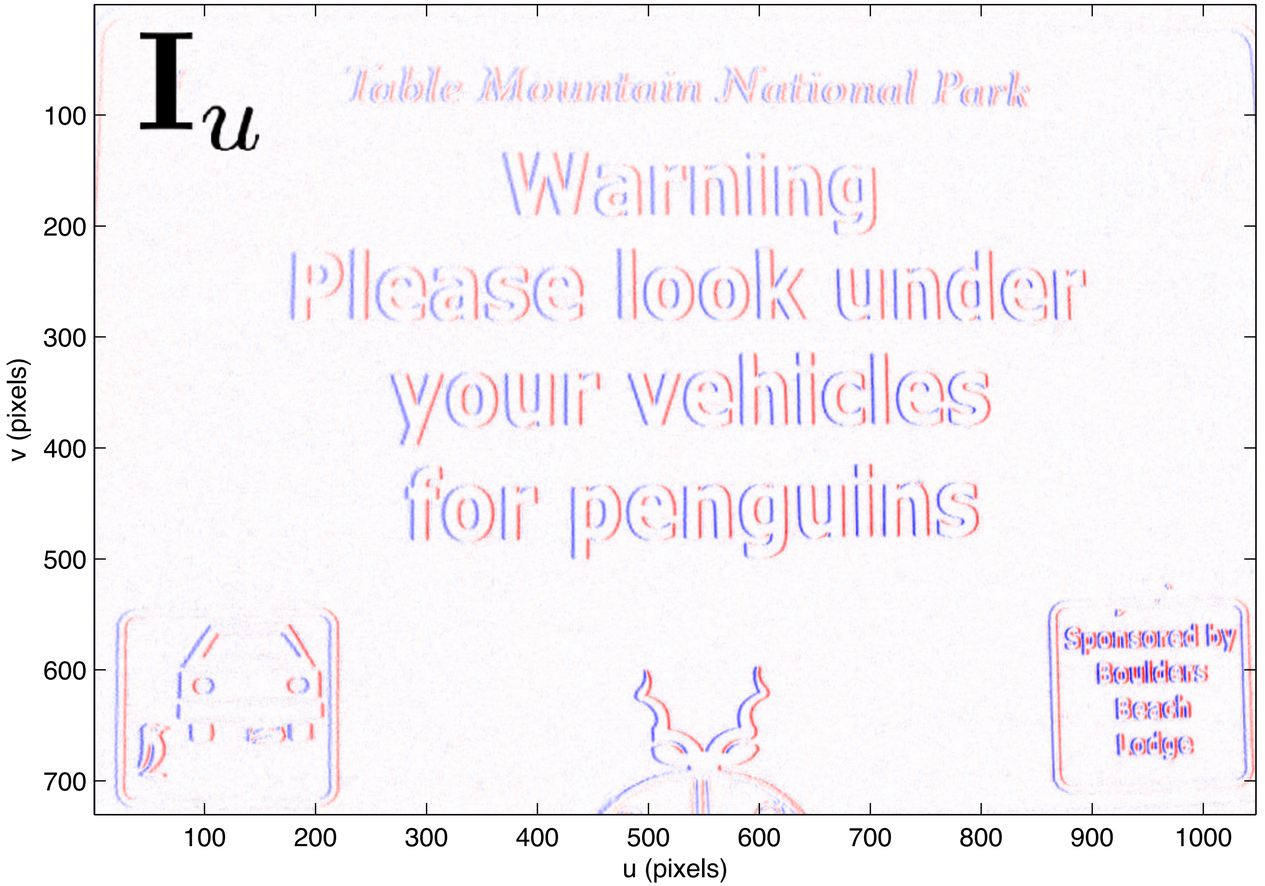




$$\mathbf{K} = \frac{1}{2}\begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$$

Horizontal gradient image

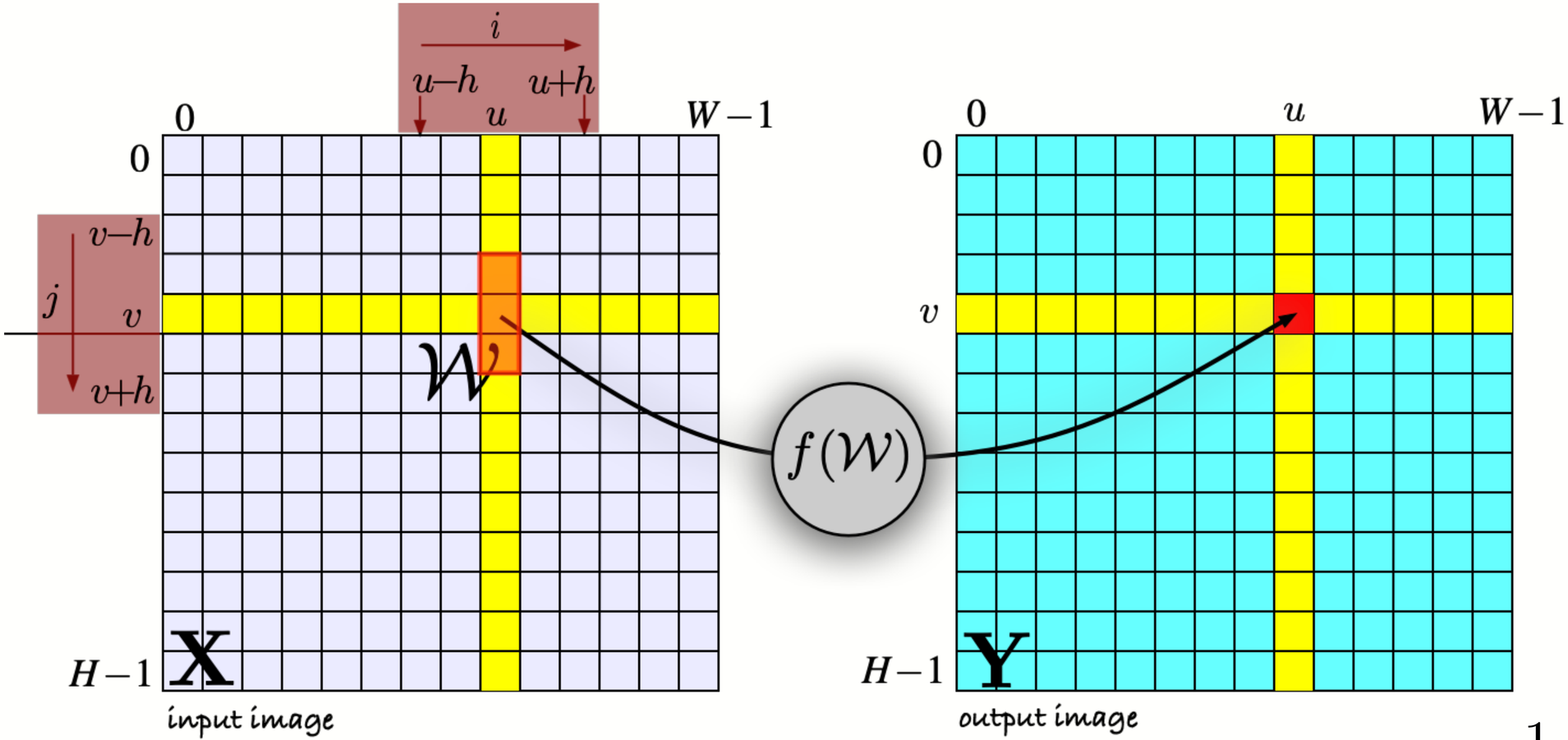




$$\mathbf{K} = \frac{1}{2}\begin{pmatrix}1 & 0 & -1\end{pmatrix}$$

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

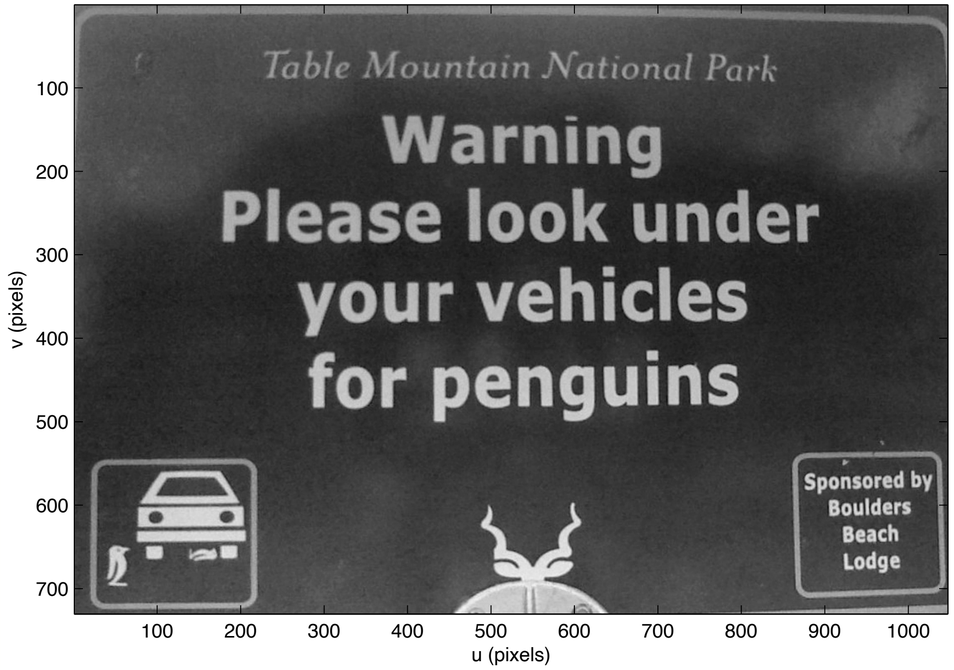
Vertical gradient as correlation with a kernel



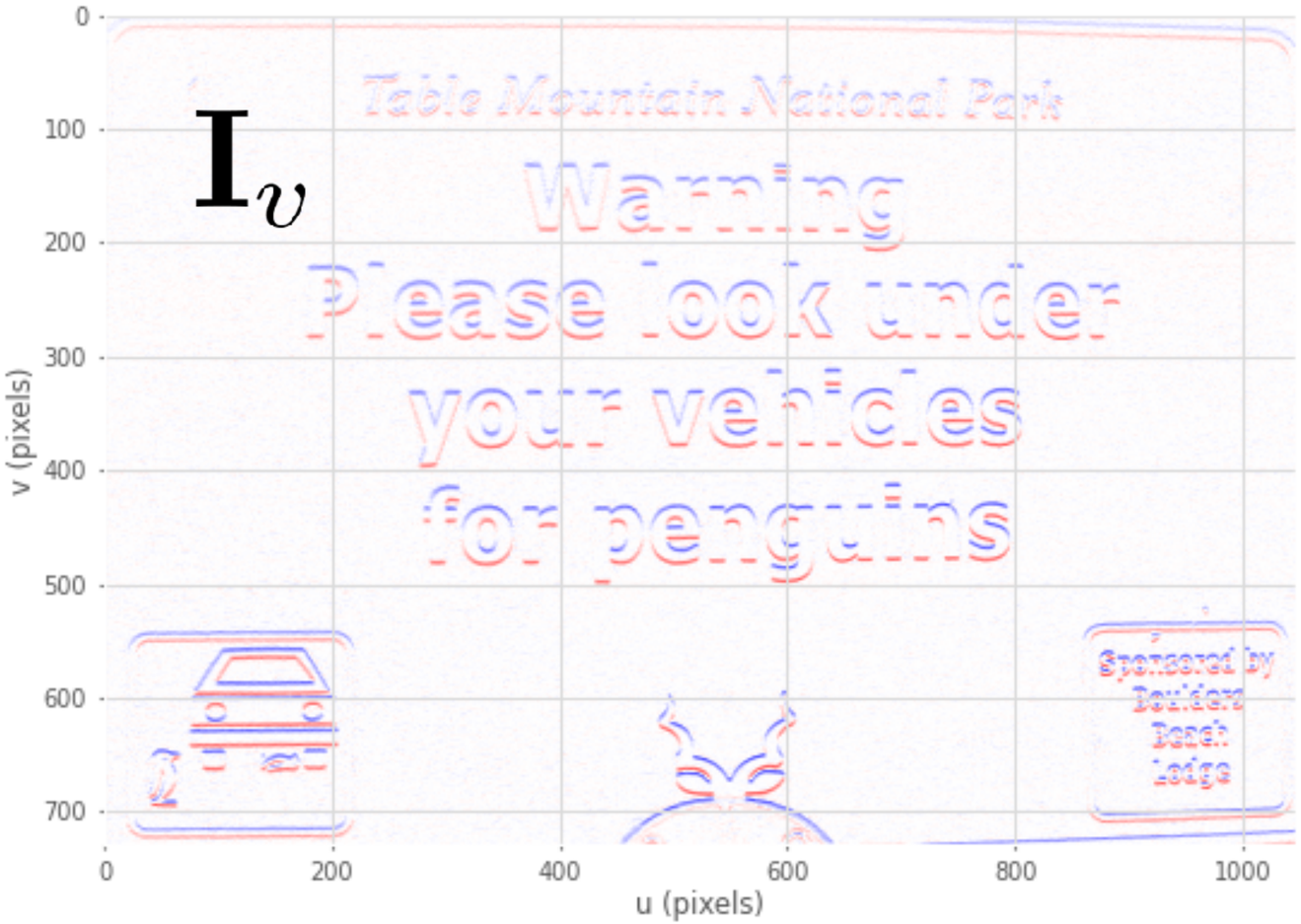
With kind permission of **Springer Science+Business Media**

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

Vertical gradient image, use the transposed kernel



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




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

2D convolution

- Correlation is closely related to convolution

$$\mathbf{O}[u,v] = \sum_{(i,j)\in\mathcal{W}} \mathbf{I}[u-i, v-j]\mathbf{K}[i,j], \forall (u,v) \in \mathbf{I}$$

- **Convolution is the same as correlation if the kernel is symmetric**
  - smoothing kernels are always symmetric
  - derivative kernels are anti-symmetric
- Often written in operator form $\mathbf{O} = \mathbf{K} \otimes \mathbf{I}$

Properties of convolution

- Commutative

$$A \otimes B = B \otimes A$$

- Associative

$$A \otimes B \otimes C = (A \otimes B) \otimes C = A \otimes (B \otimes C)$$

- Distributive

$$A \otimes (\alpha B) = \alpha (A \otimes B)$$

- and Linear

$$A \otimes (B + C) = A \otimes B + A \otimes C$$

Properties of Gaussians

$$\mathbf{G}(\sigma_1) \otimes \mathbf{G}(\sigma_2) = \mathbf{G}(\sqrt{\sigma_1^2 + \sigma_2^2})$$

$$\mathbf{G}(\sigma) \otimes \mathbf{G}(\sigma) = \mathbf{G}(\sqrt{2}\sigma)$$

Advantages of associativity

- Convolving an image with a Gaussian kernel twice

$$\mathbf{I} = \mathbf{G} \otimes (\mathbf{G} \otimes \mathbf{I})$$

- Is the same as convolving the image with a kernel that is the Gaussian convolved with itself

$$\begin{aligned} \mathbf{I} &= (\mathbf{G} \otimes \mathbf{G}) \otimes \mathbf{I} \\ &= \mathbf{G}' \otimes \mathbf{I} \end{aligned}$$

Derivative with smoothing

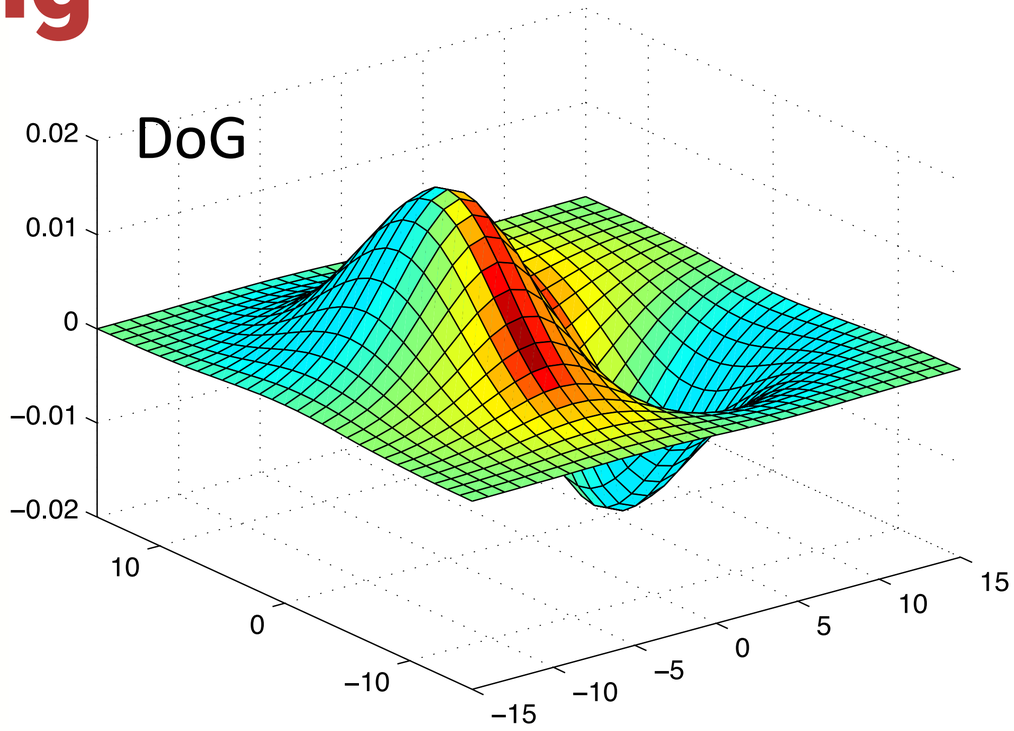
With kind permission of **Springer Science+Business Media**

- Convolve image with the derivative kernel $\mathbf{D}$

$$\nabla \mathbf{I} = \mathbf{D} \otimes \mathbf{I}$$

- If we smooth the image first then

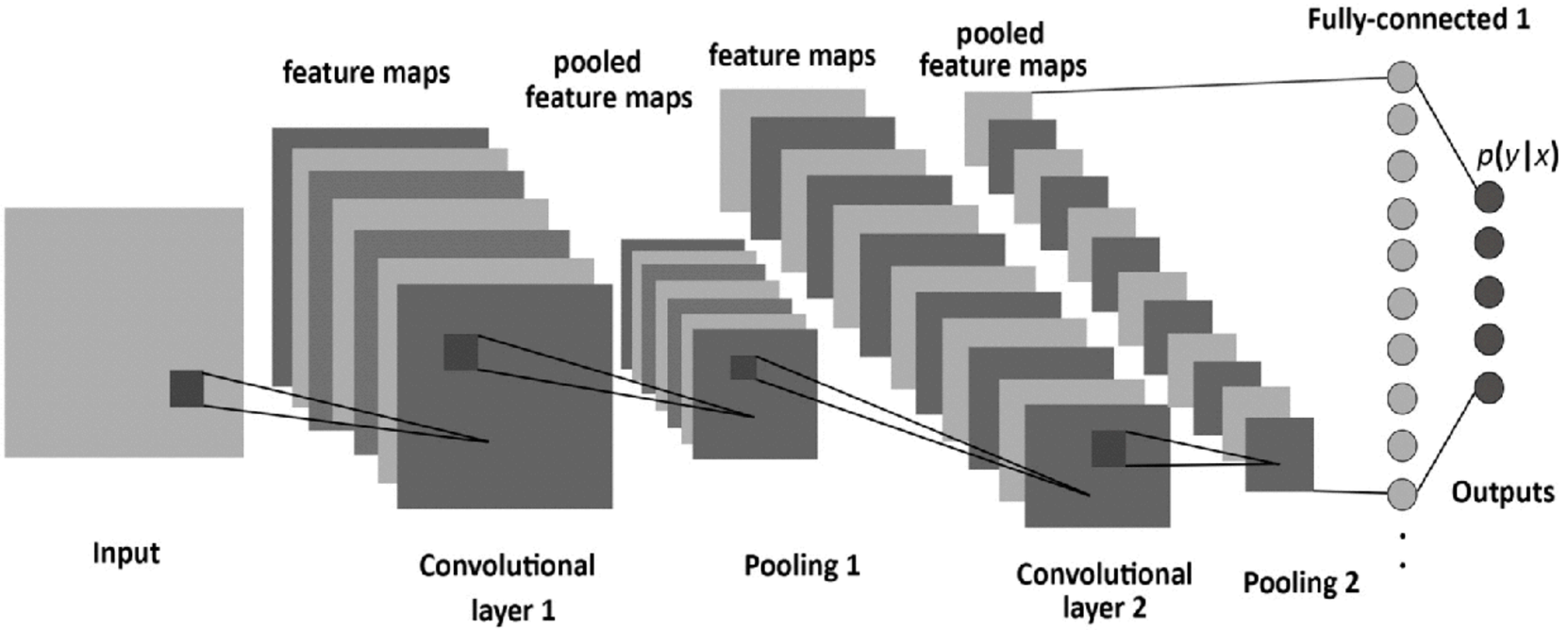
$$\nabla \mathbf{I} = \mathbf{D} \otimes (\mathbf{G}(\sigma) \otimes \mathbf{I}) = \underbrace{(\mathbf{D} \otimes \mathbf{G}(\sigma))}_{DoG} \otimes \mathbf{I}$$



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

```
>>> Image.kdgauss(5, 15)
```

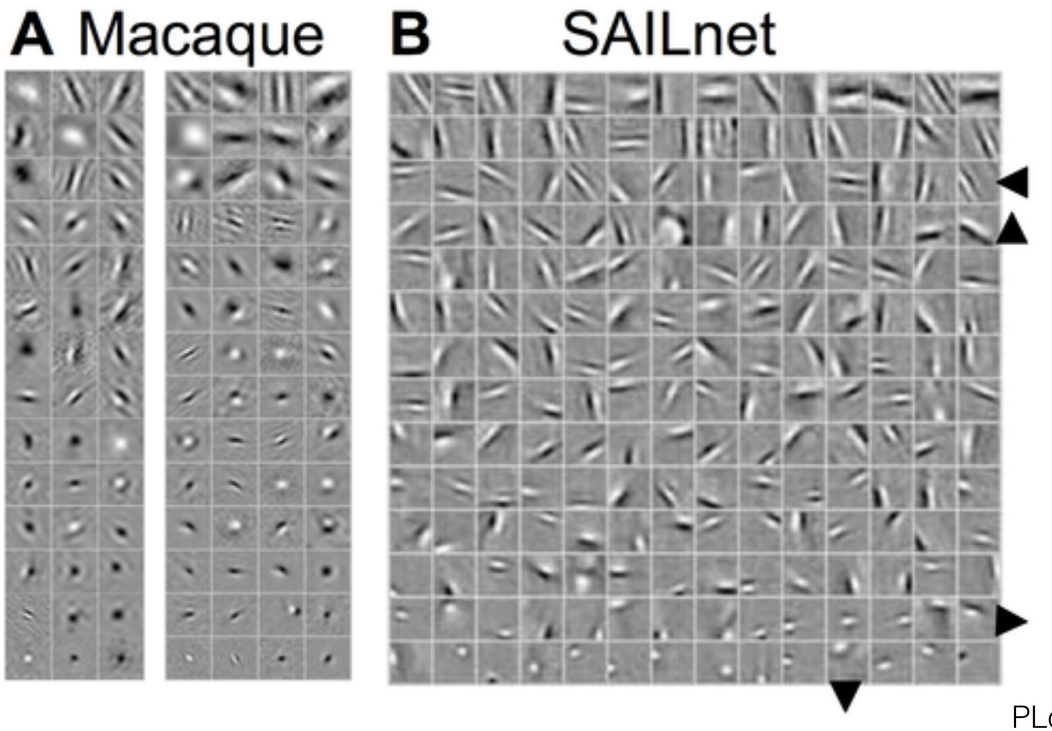
Convolutional neural networks



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

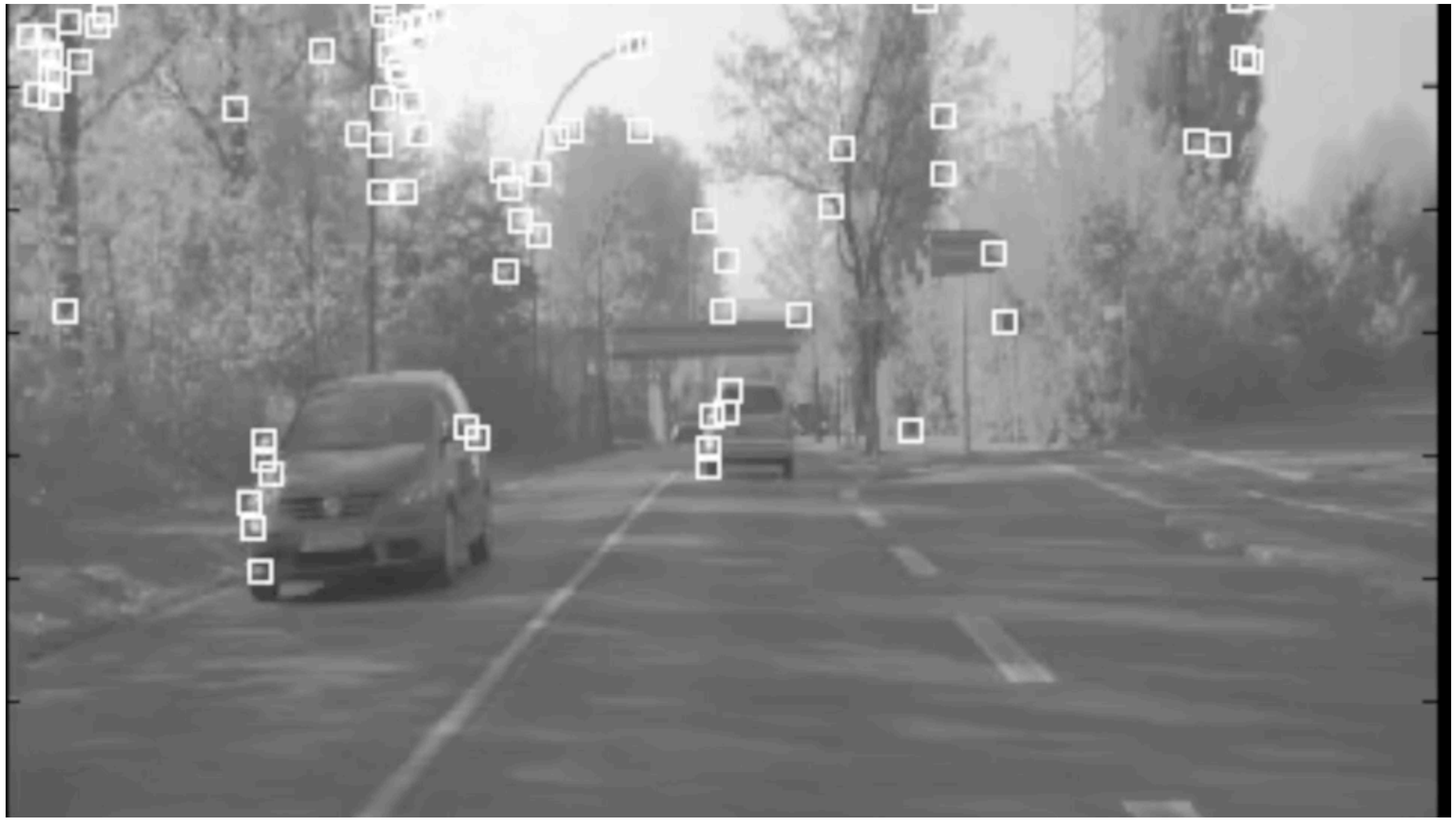


PLoS computational biology. 7. e1002250. 10.1371/journal.pcbi.1002250

- 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



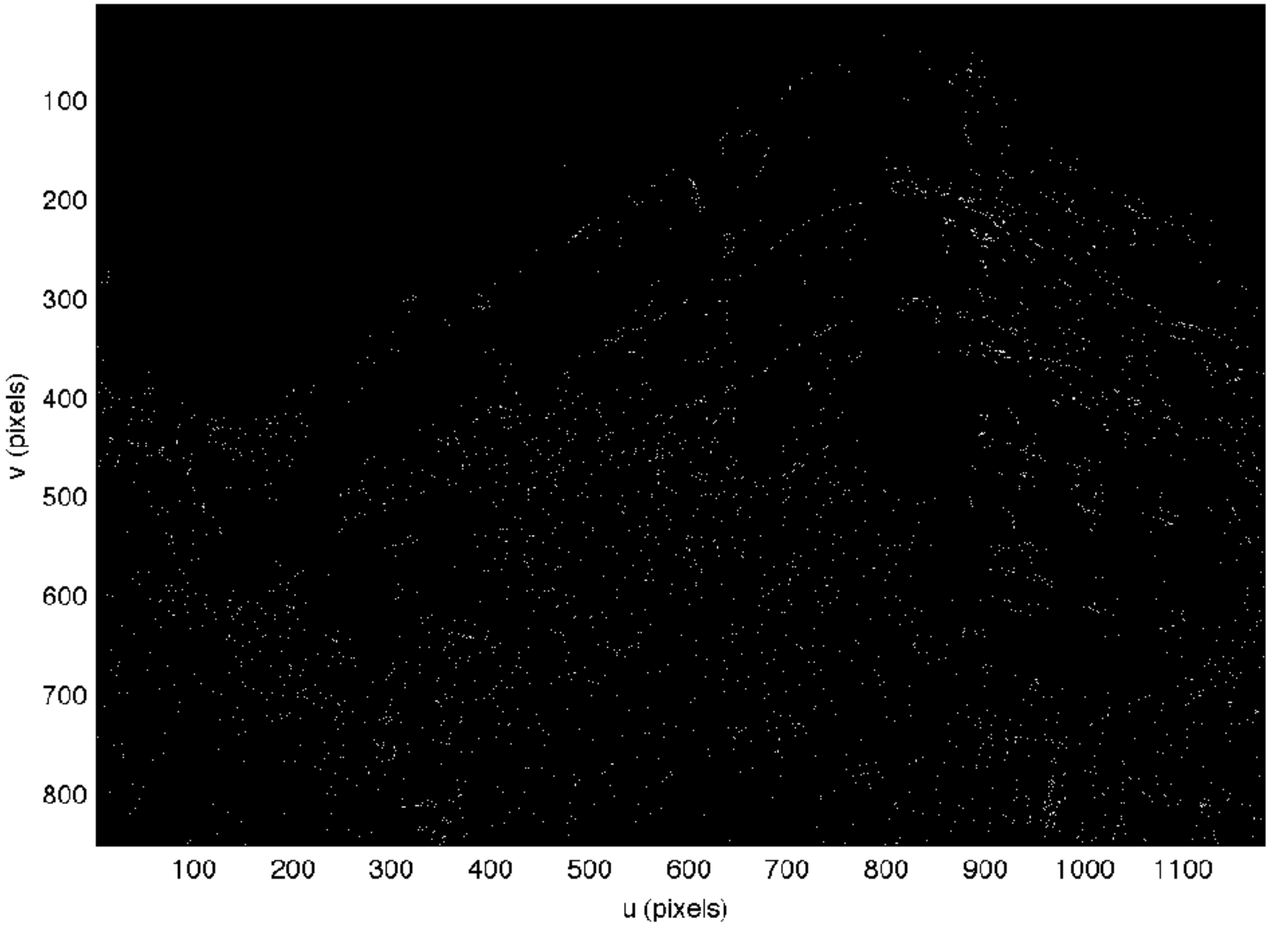
- between different views captured at the same time
- across consecutive views

Finding corresponding points

(818,164)

(619,184)

the pixel value is **51**



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

Finding corresponding points

(818,164)

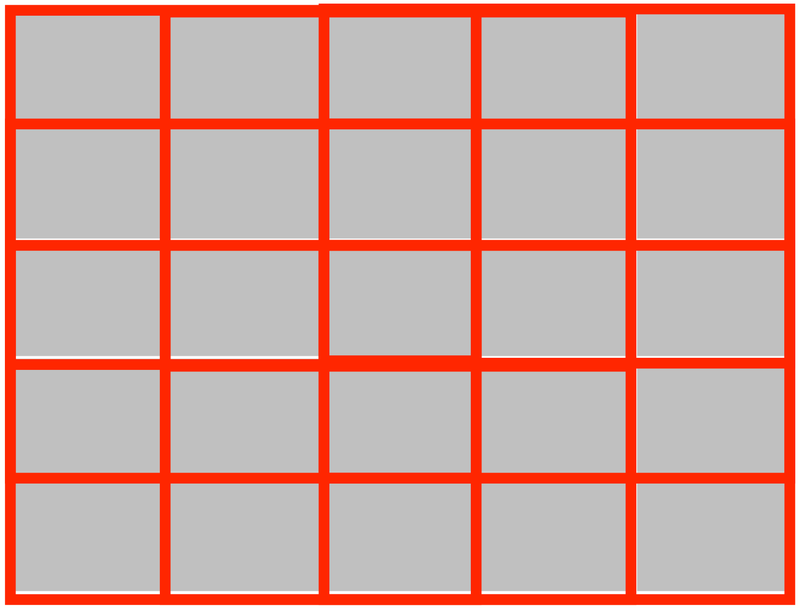
pixel value is **51**

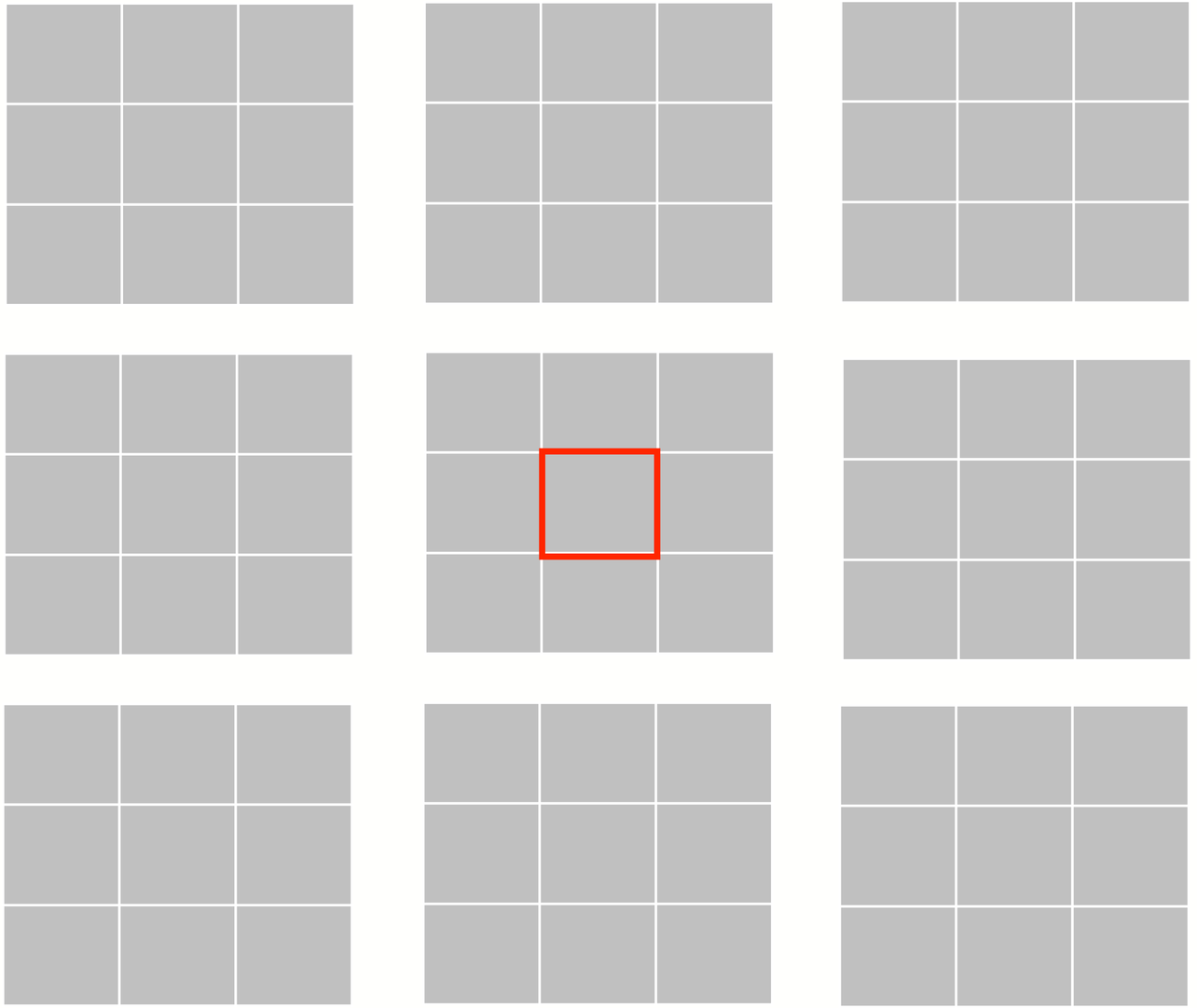
(619,184)

pixel value is **47**

Finding corresponding points

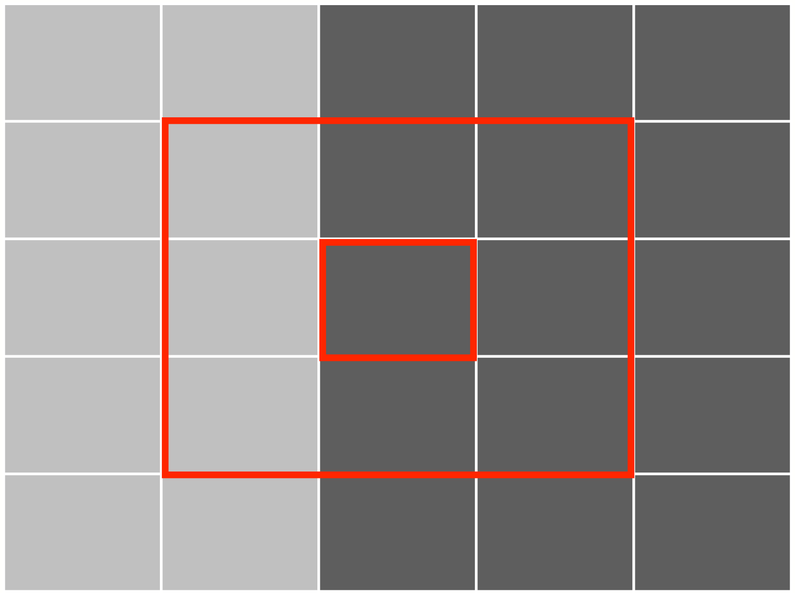
Corner detector intuition

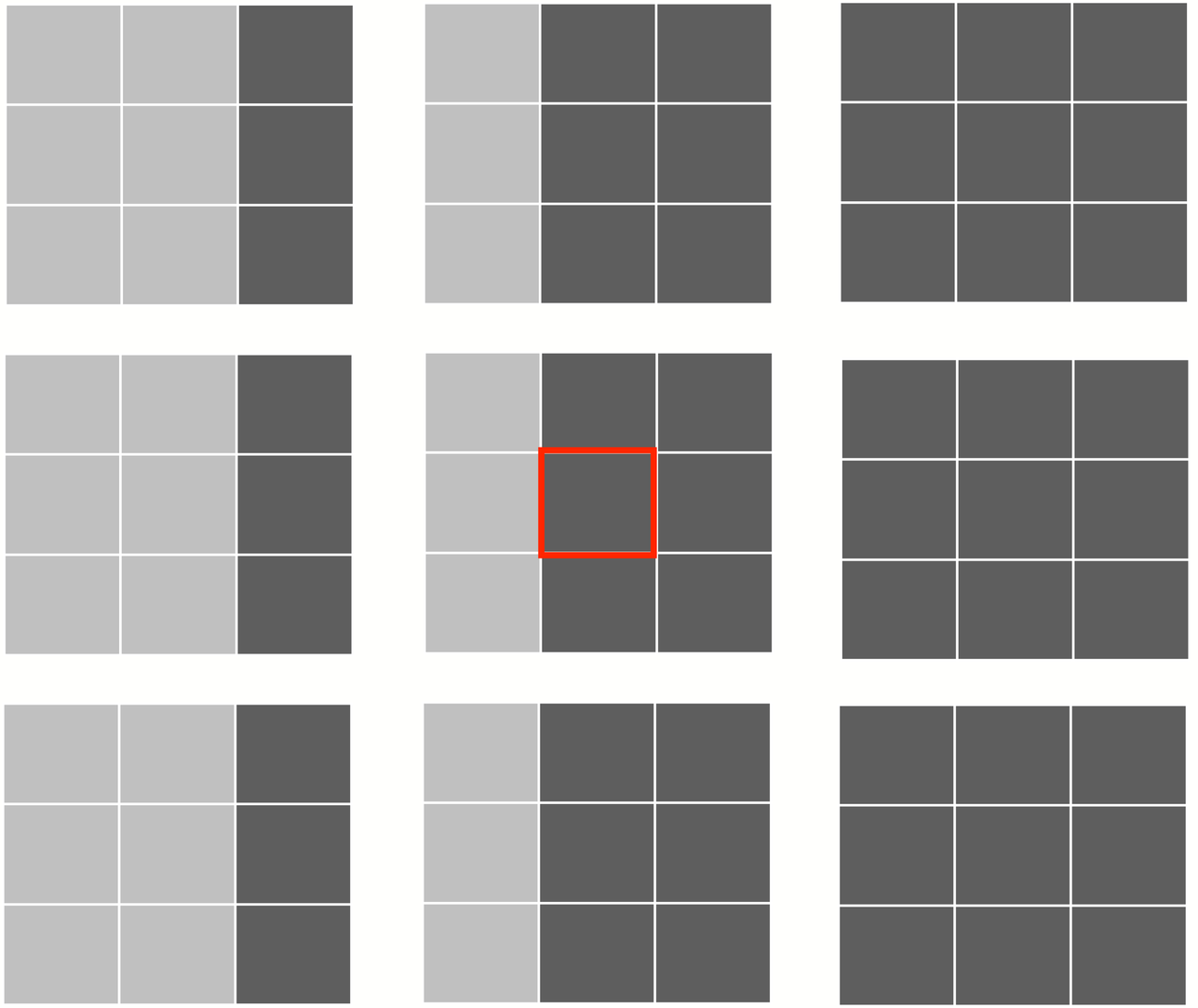




- Similar at 8 other locations
  - ➥ not very unique

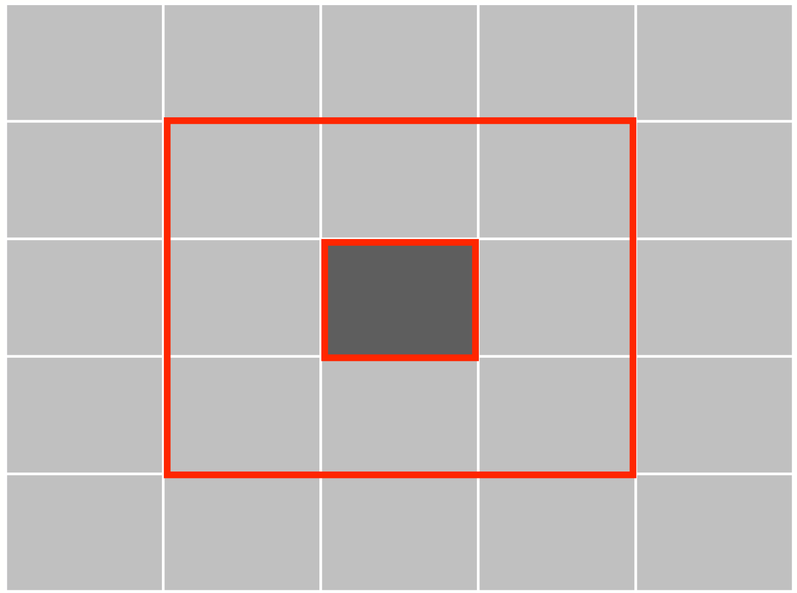
Corner detector intuition

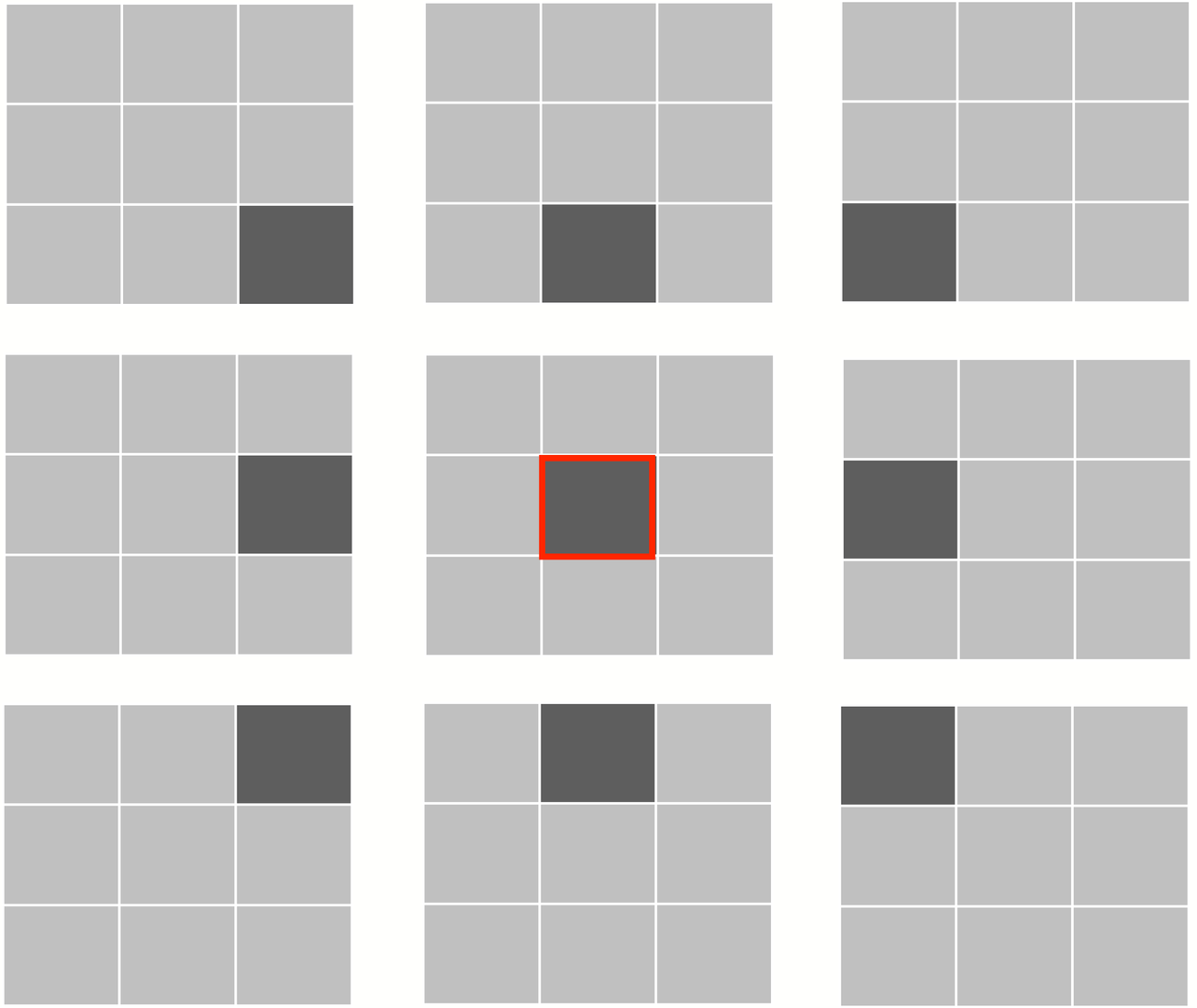




- Similar at 2 other locations
- vertical motion causes no change
  - ➥ parallel to the line

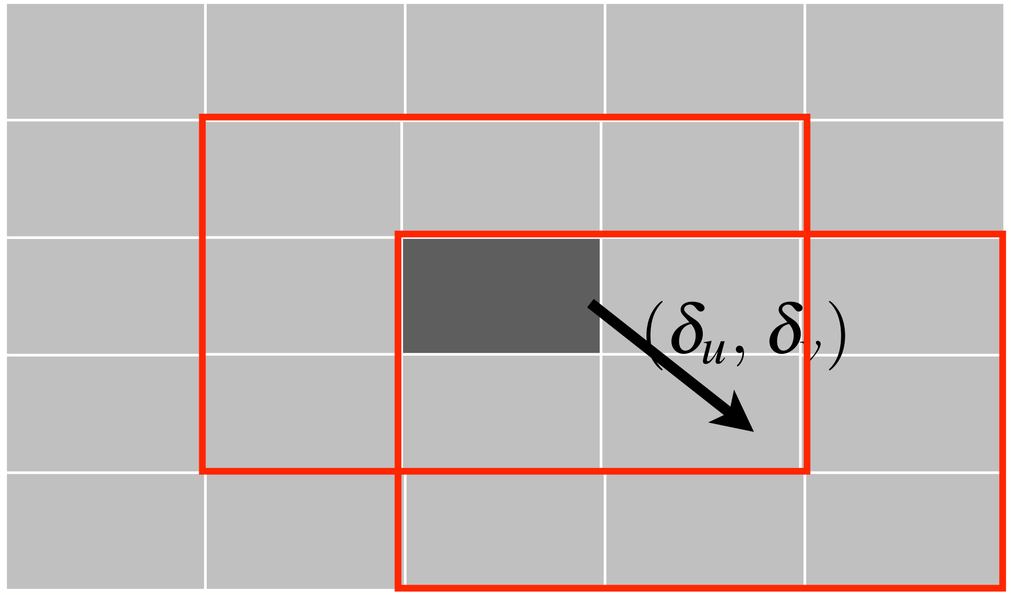
Corner detector intuition





- Similar at no other locations
  - ➥locally unique

...Corner detector intuition



- Similar at 0 other locations (locally unique)

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

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

Corner detector

- More general approach
  - ➥ add a Gaussian weighting matrix

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

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

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

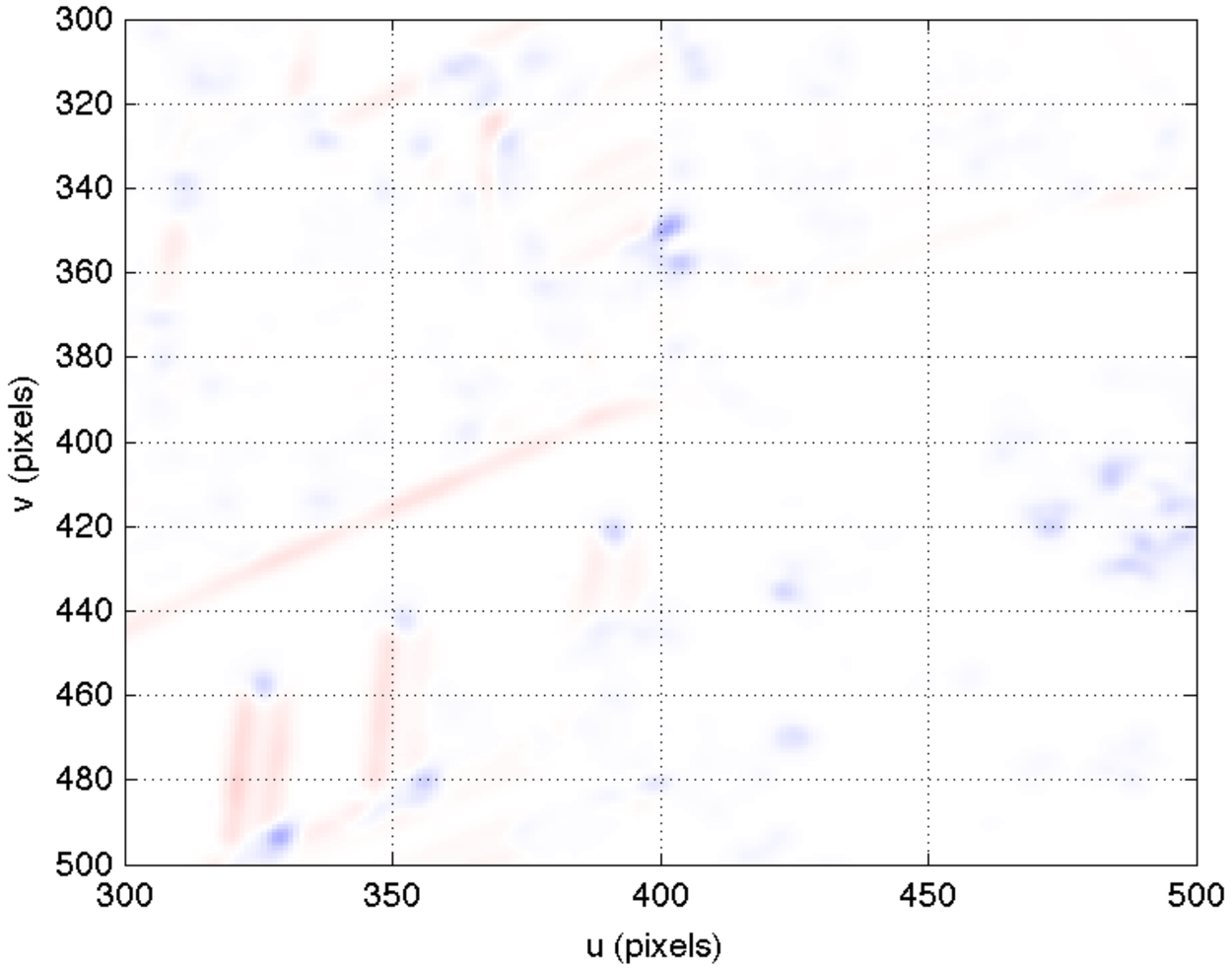
- The eigenvalues of $\mathbf{A}$ contain important information

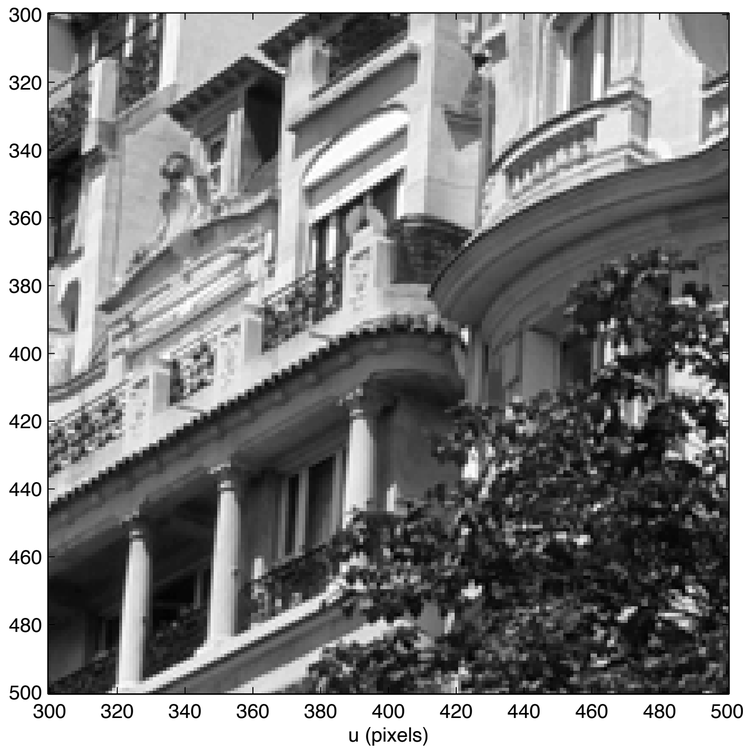
	λ_2 small	λ_2 large
λ_1 small	**constant**	**edge**
λ_1 large	**edge**	**peak**

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

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

Harris corner value



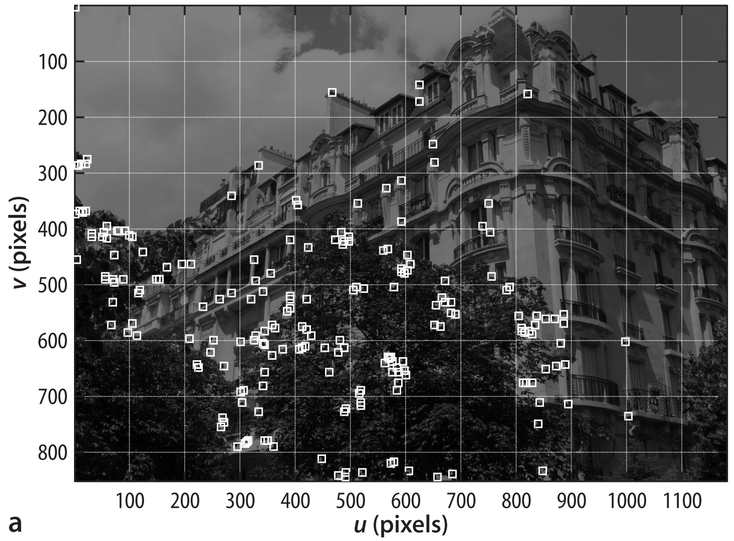


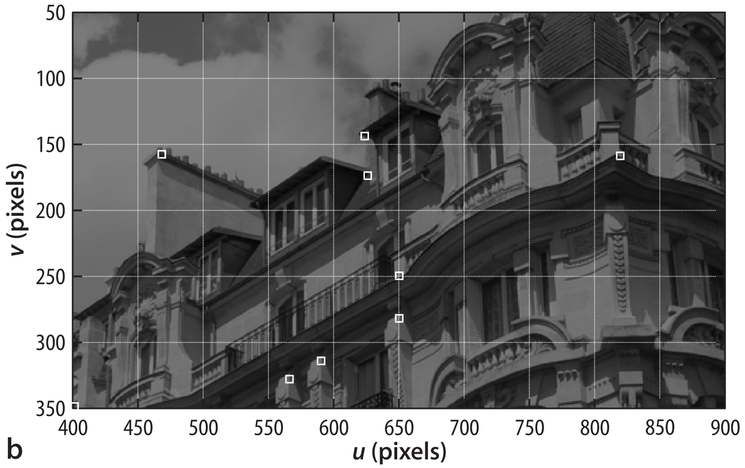
blue is positive,

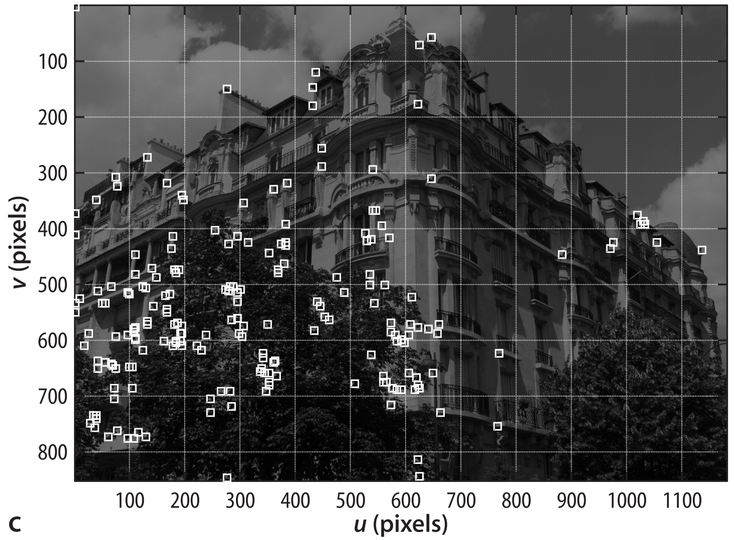
red is negative

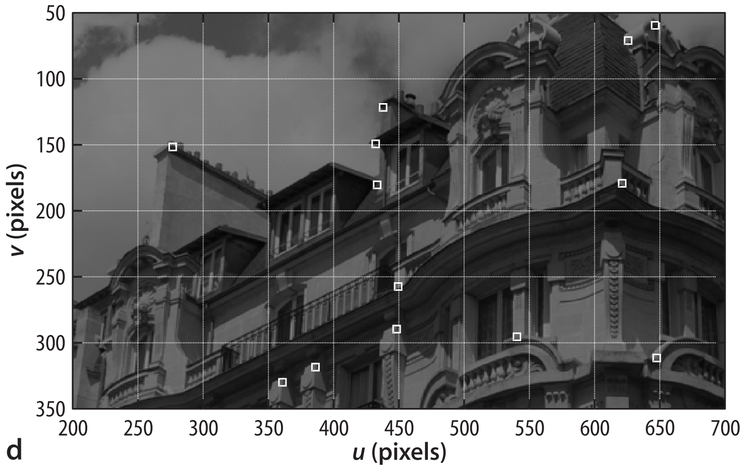
Harris CG, Stephens MJ (1988) A combined corner and edge detector.Proceedings of the Fourth Alvey Vision Conference. Manchester, pp 147–151

Harris features for 2 views









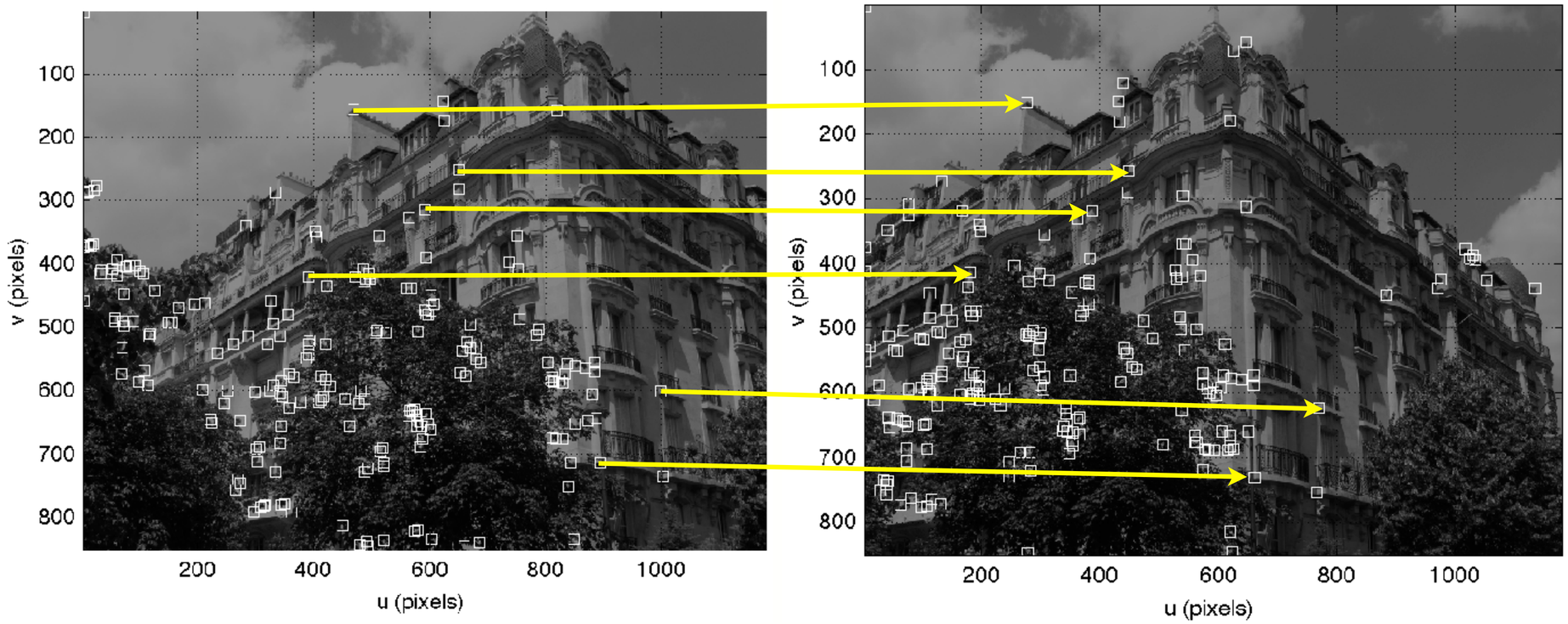
- Some features are clearly the same point in the scene (good)
- Unduly excited about the trees at the lower left (not good

Comparing features

- We've found the coordinates of some interesting points in each image:

$$\{ {}^{1}\mathrm{p}_i, i \in 1 \cdots N_1 \} \qquad \{ {}^{2}\mathrm{p}_j, j \in 1 \cdots N_2 \}$$

- Now we need to determine the correspondence, which ${}^{1}\mathrm{p}_i \leftrightarrow {}^{2}\mathrm{p}_j$



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 ${}^1\mathrm{p}_i$ we test the similarity against all the points $\{ {}^2\mathrm{p}_j,\ j \in 1 \cdots N_2 \}$ in the other image
- This is an $N_1 \times N_2$ search problem

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

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
 .
 .
```

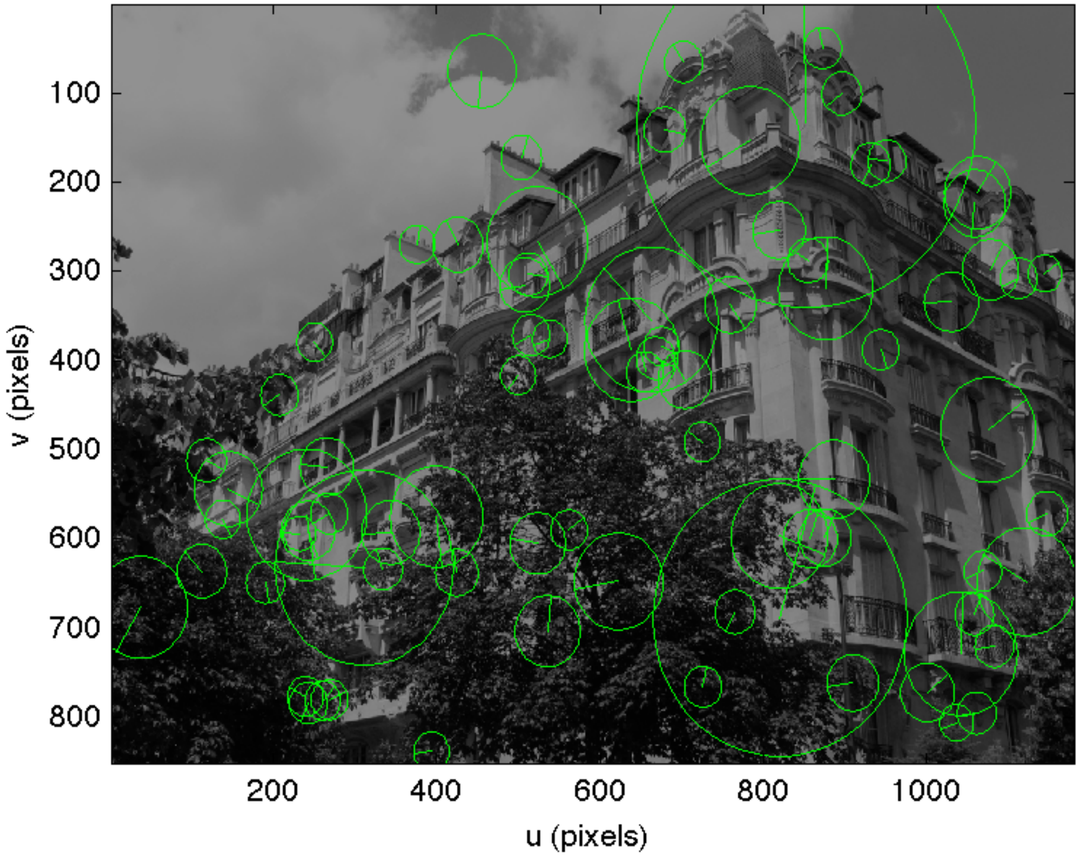
Harris feature recap

- 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

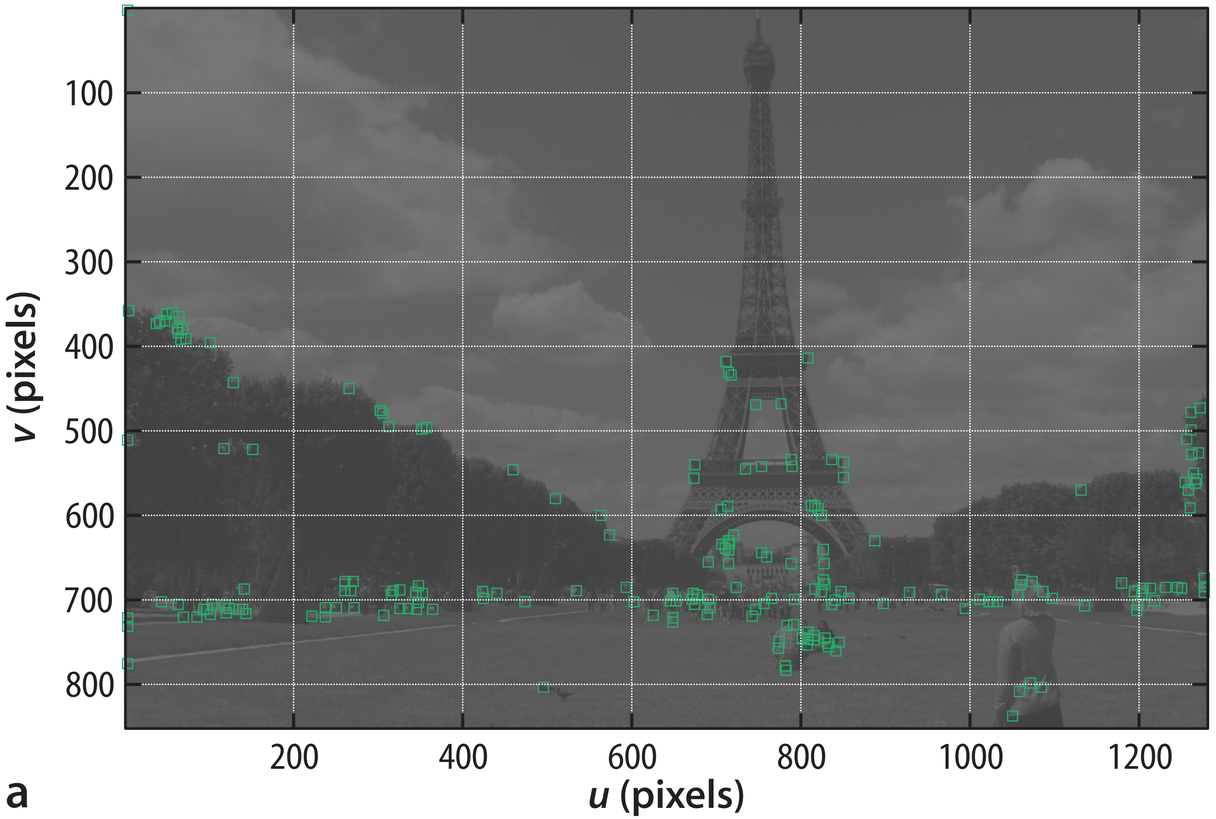
The problem of orientation

Scale Invariant Feature Transform

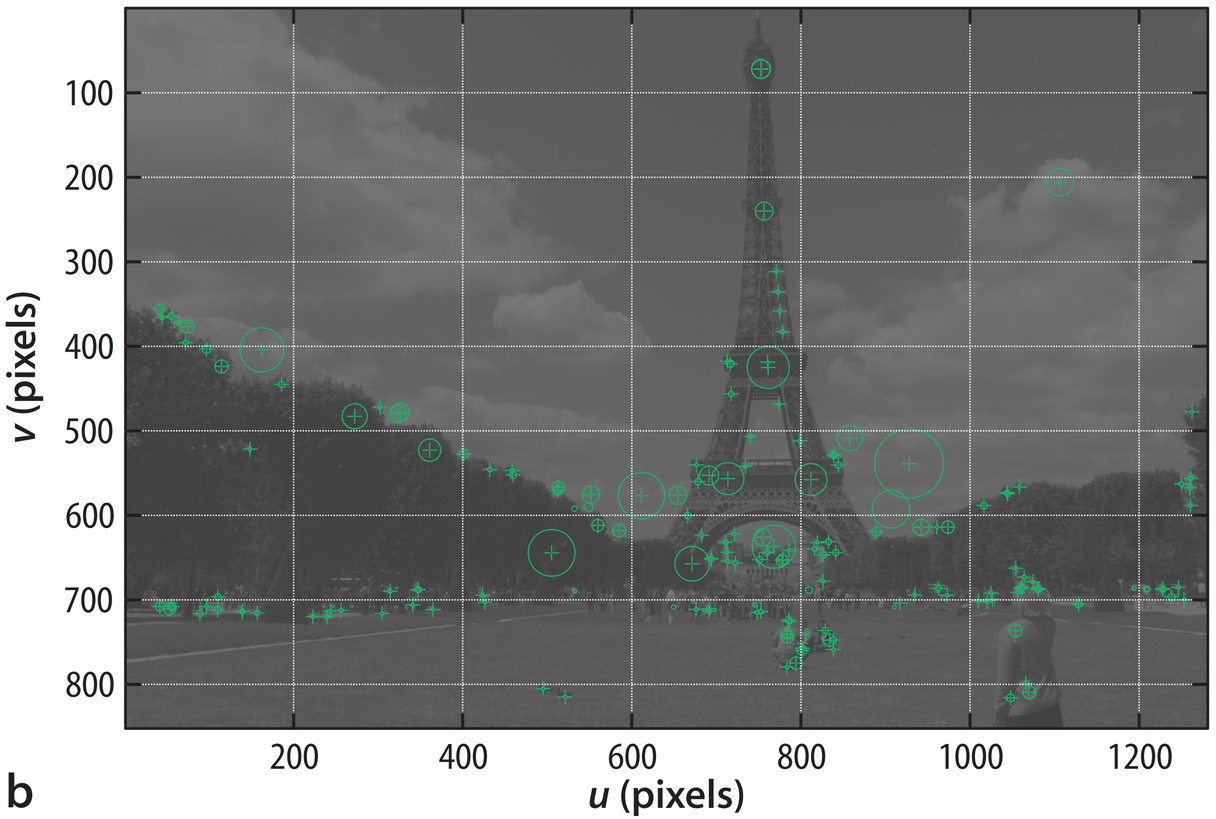


SIFT detector (Lowe, 2004)

Harris vs SIFT



a



b