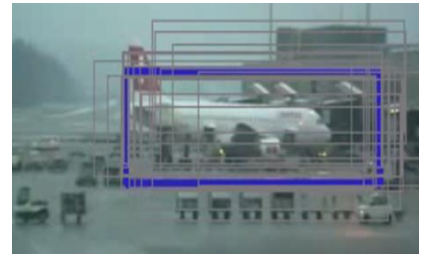
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

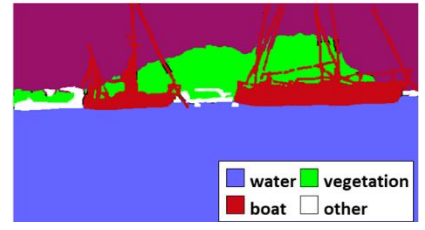


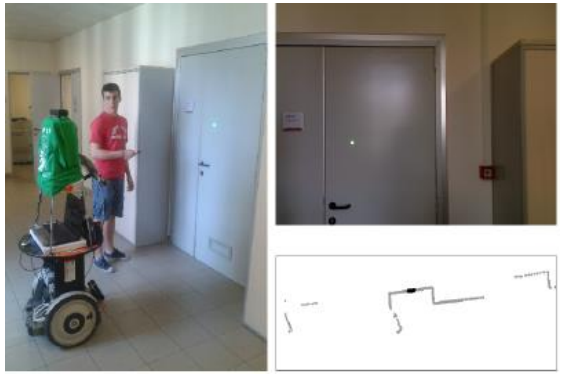
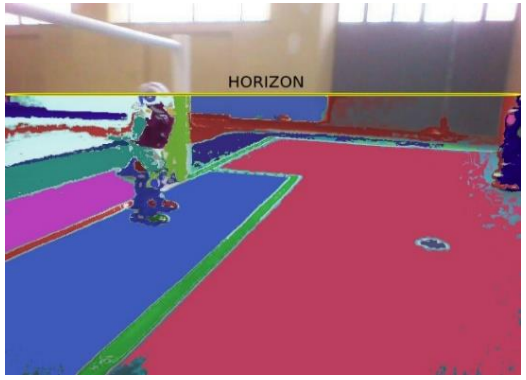
Corso di Visione e Percezione

Docente
Domenico D. Bloisi

Filtri

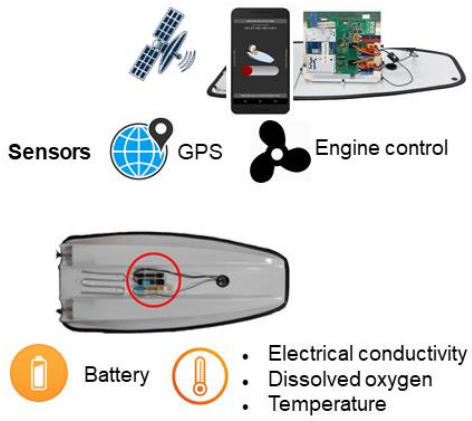


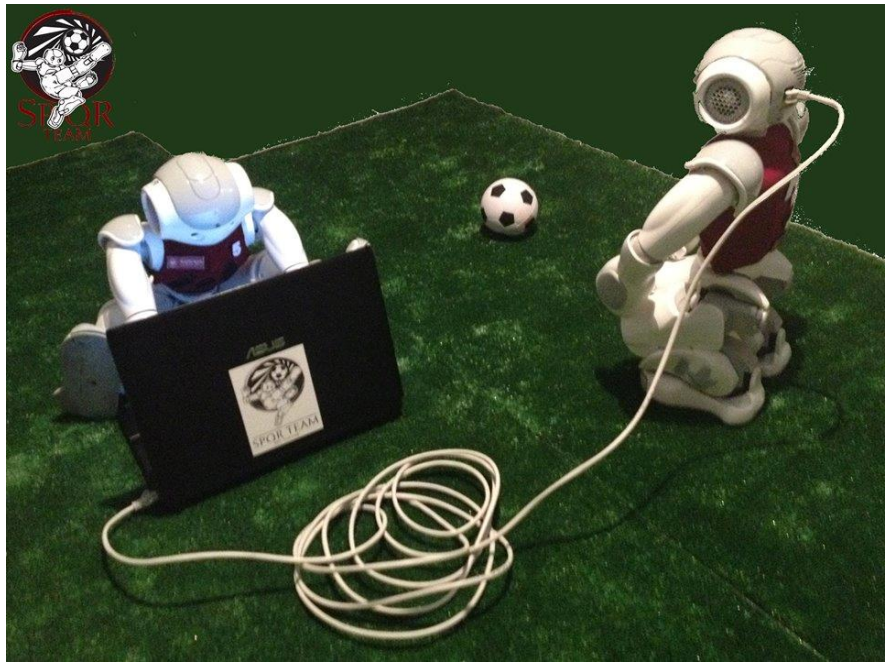




Domenico Daniele Bloisi

- Professore Associato
  Dipartimento di Matematica, Informatica ed Economia
  Università degli studi della Basilicata
  http://web.unibas.it/bloisi

- SPQR Robot Soccer Team
  Dipartimento di Informatica, Automatica e Gestionale Università degli studi di Roma “La Sapienza”
  http://spqr.diag.uniroma1.it





UNIBAS Wolves https://sites.google.com/unibas.it/wolves

- UNIBAS WOLVES is the robot soccer team of the University of Basilicata. Established in 2019, it is focussed on developing software for NAO soccer robots participating in RoboCup competitions.

- UNIBAS WOLVES team is twinned with SPQR Team at Sapienza University of Rome.

Informazioni sul corso

- Home page del corso:
  https://web.unibas.it/bloisi/corsi/visione-e-percezione.html
- Docente: Domenico Daniele Bloisi
- Periodo: II semestre marzo 2022 – giugno 2022
  - Martedì dalle 15:00 alle 17:00 (Aula Copernico)
  - Mercoledì dalle 8:30 alle 10:30 (Aula Copernico)

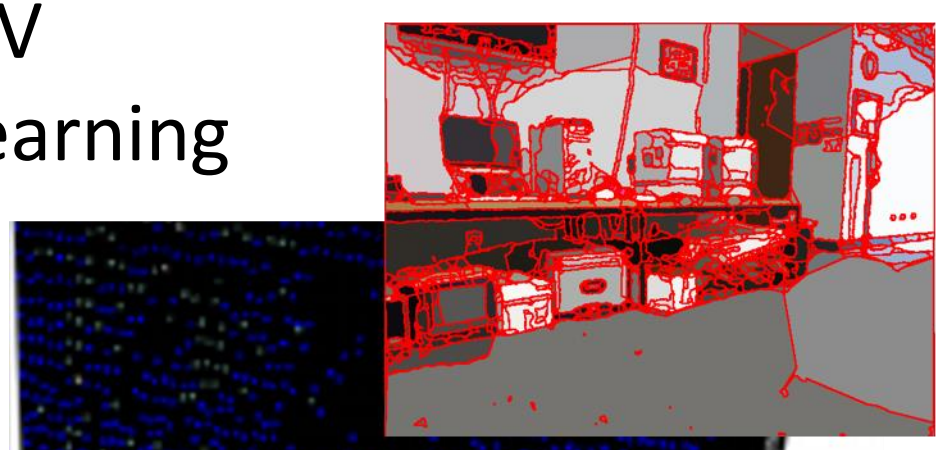
Ricevimento

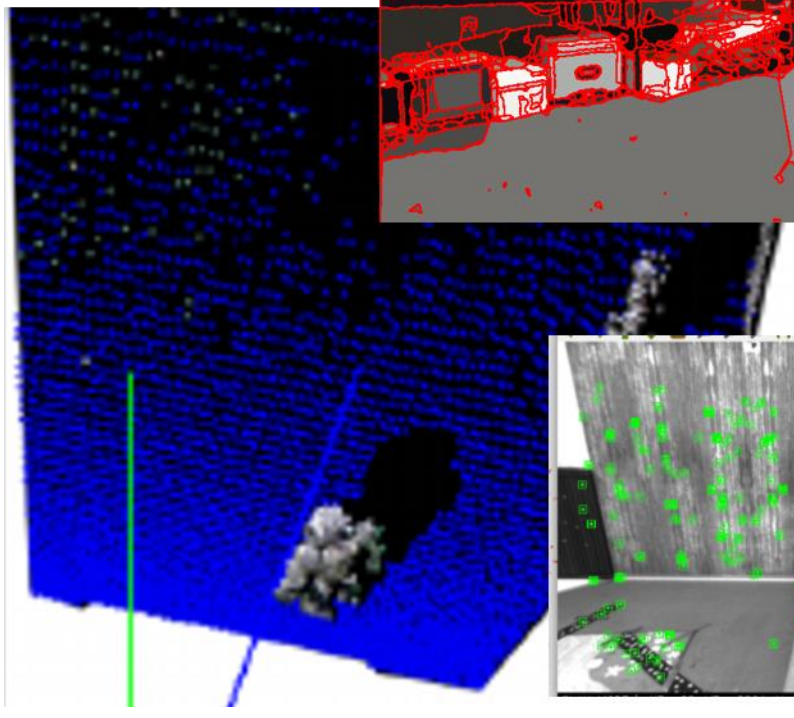
- Durante il periodo delle lezioni:
  Mercoledì dalle 11:00 alle 12:30 → Edificio 3D, II piano, stanza 15
  Si invitano gli studenti a controllare regolarmente la bacheca degli avvisi per eventuali variazioni
- Al di fuori del periodo delle lezioni:
  da concordare con il docente tramite email

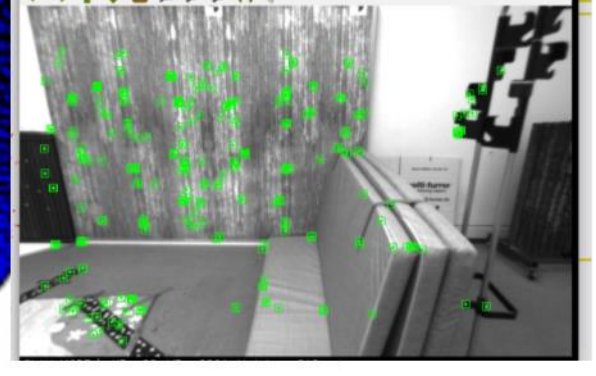
Per prenotare un appuntamento inviare una email a
domenico.bloisi@unibas.it

Programma – Visione e Percezione

- Introduzione al linguaggio Python
- Elaborazione delle immagini con Python
- Percezione 2D – OpenCV
- Introduzione al Deep Learning
- ROS
- Il paradigma publisher and subscriber
- Simulatori
- Percezione 3D - PCL







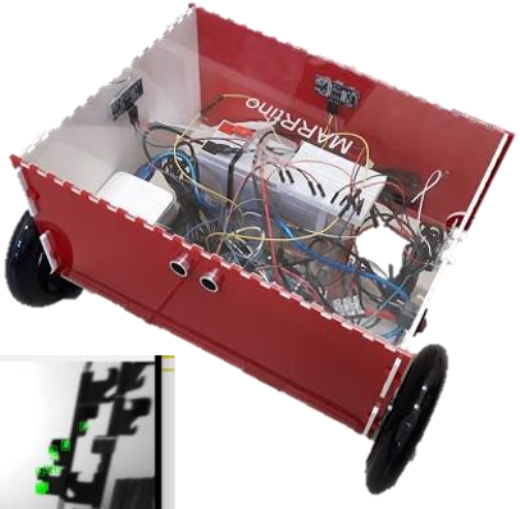


Immagine Digitale

- Una immagine digitale è una matrice di pixel
- Il termine pixel deriva da *picture element*
- Il pixel contiene l'informazione relativa alla rappresentazione della realtà che è stata catturata tramite uno scanner, una macchina fotografica o un frame grabber (per i video)

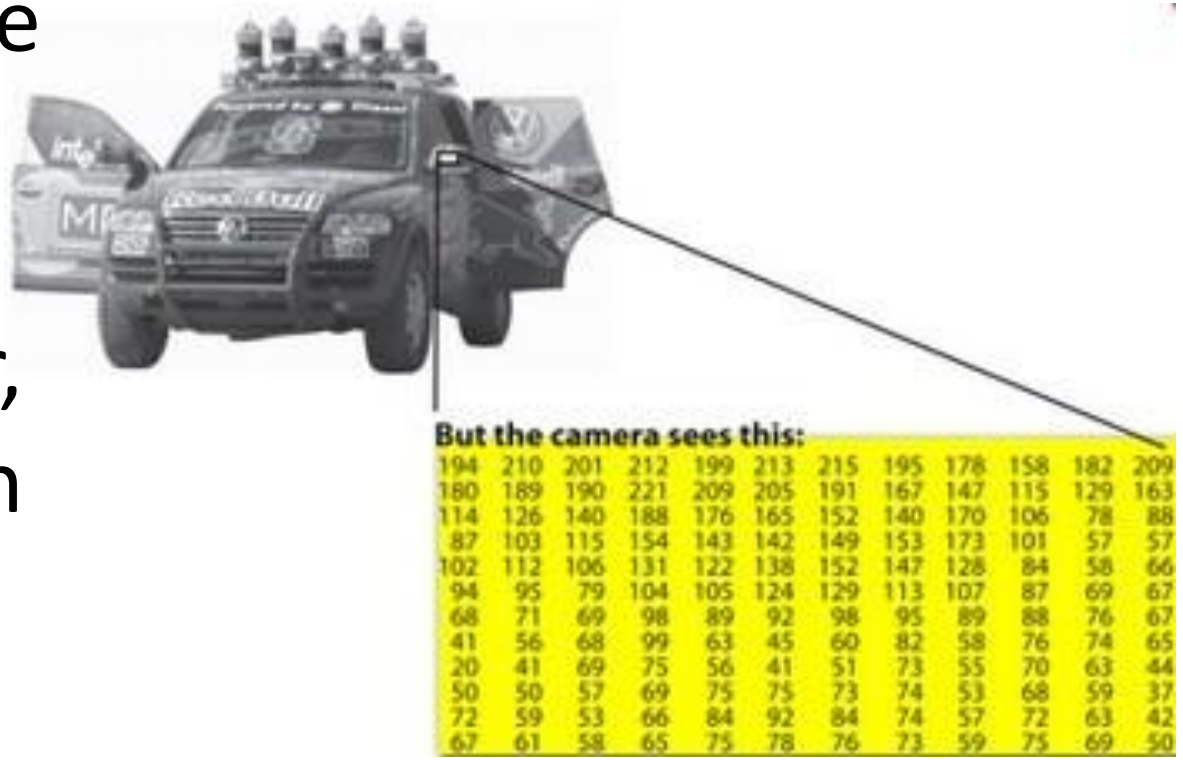


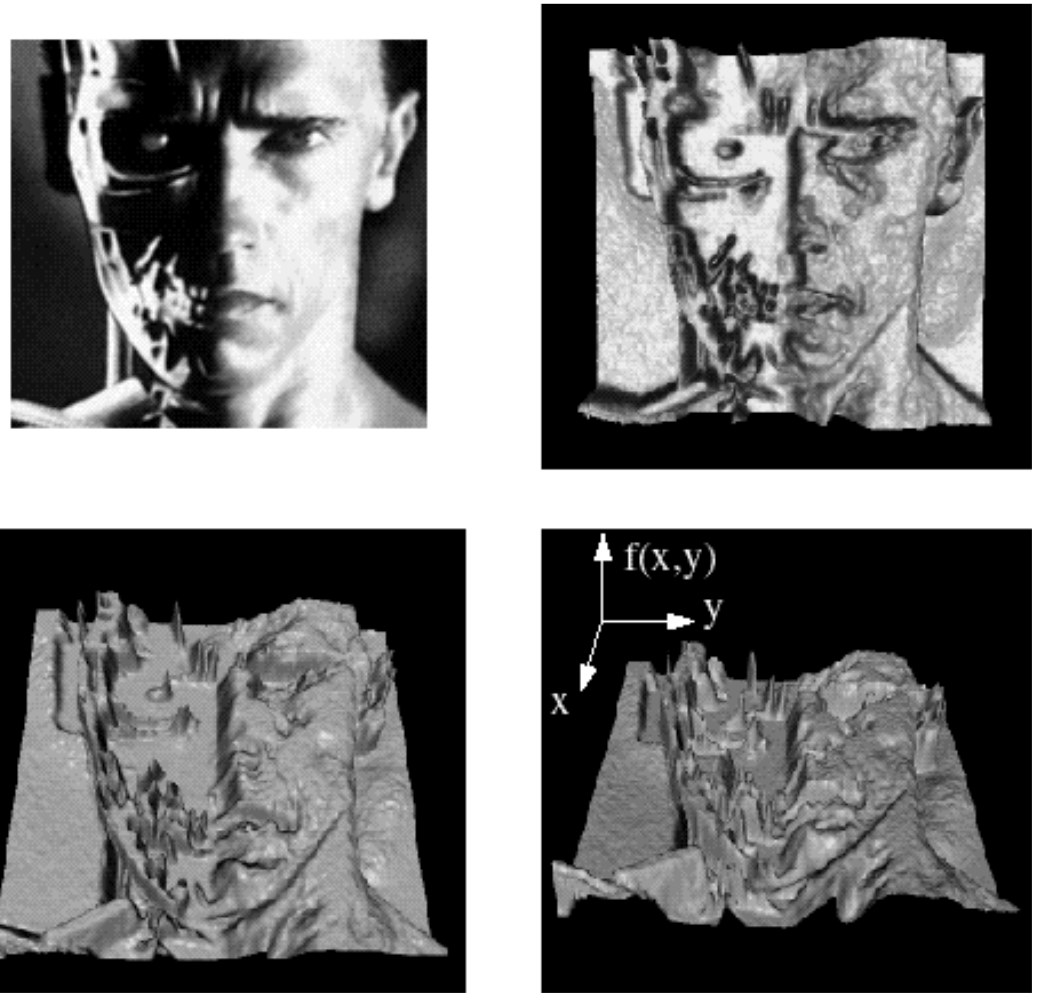
Immagine come funzione

Possiamo pensare ad una immagine come ad una funzione f da R^2 a R

- $f(x, y)$ sarà l'intensità nella posizione (x, y)
- L'immagine sarà definita all'interno di un rettangolo e ogni elemento potrà assumere valori in un range predefinito
  $f:[a,b]x[c,d] \rightarrow [0,1]$

Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Esempio



Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Immagine a colori

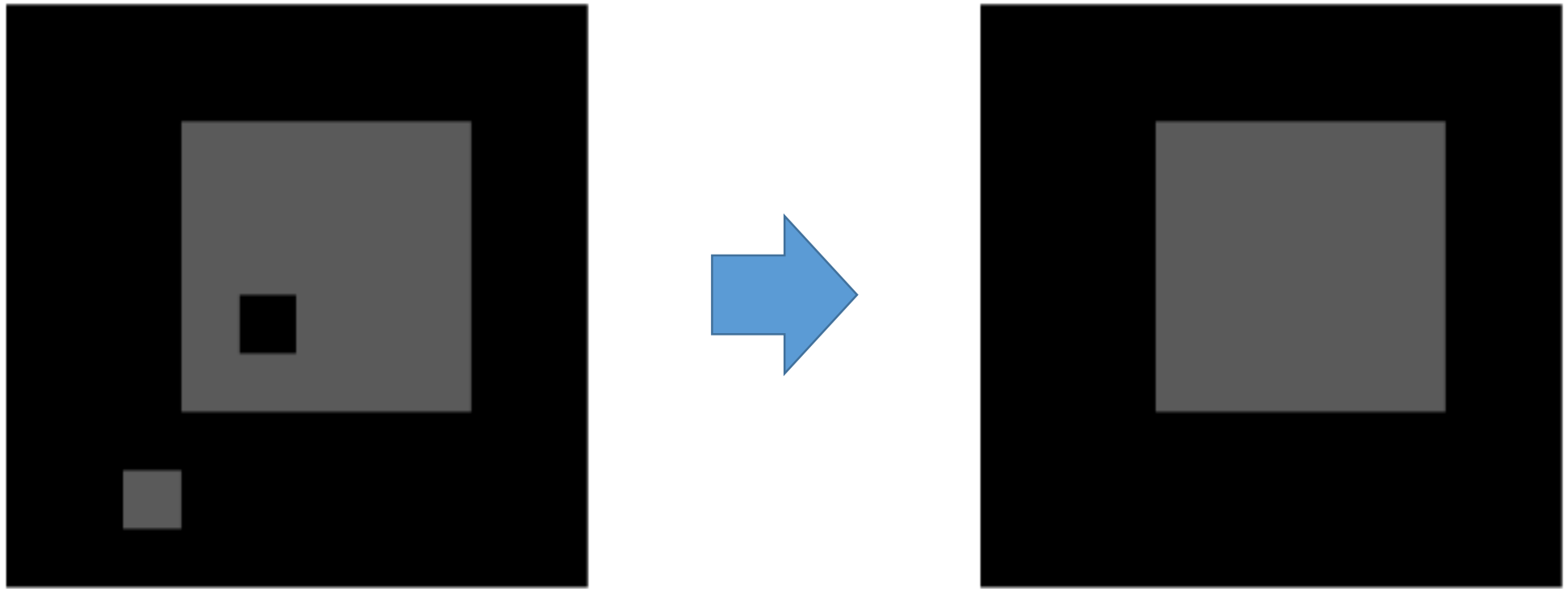
Una immagine a colori potrà essere rappresentata come l'unione tra tre funzioni, una per ognuno dei canali denominati red, green, blue

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Image filtering

Goal: Vogliamo limitare il rumore presente nell'immagine a sinistra per trasformarla in quella a destra

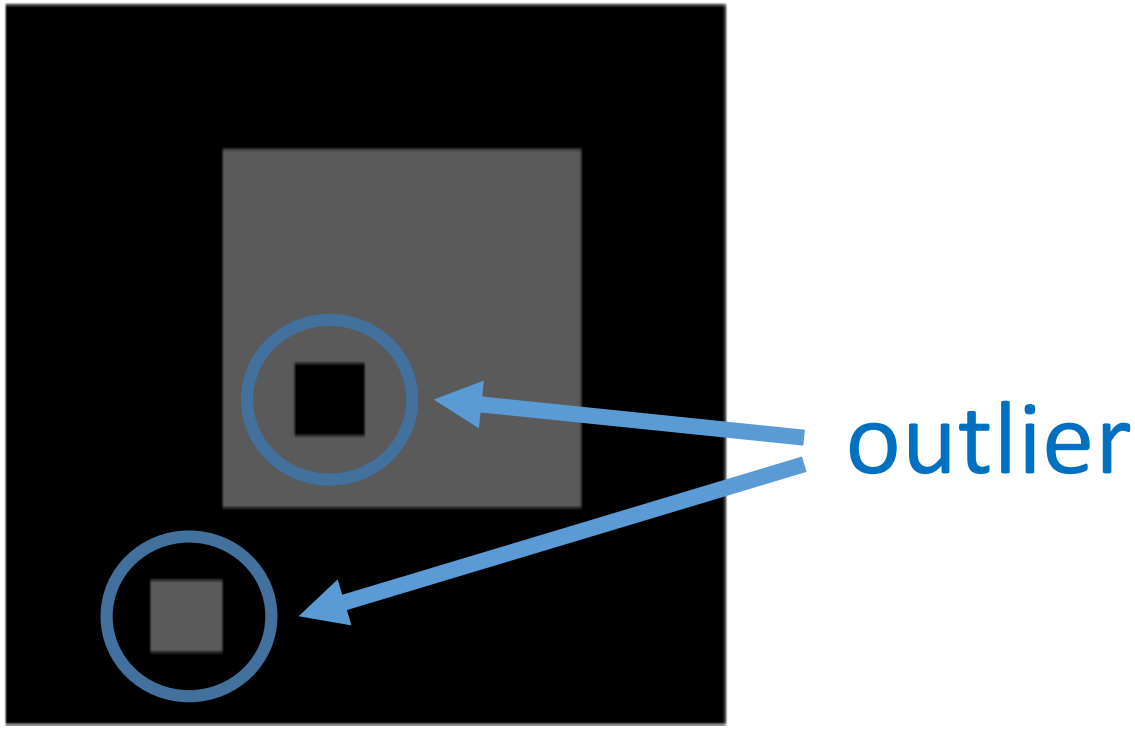


Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Outlier

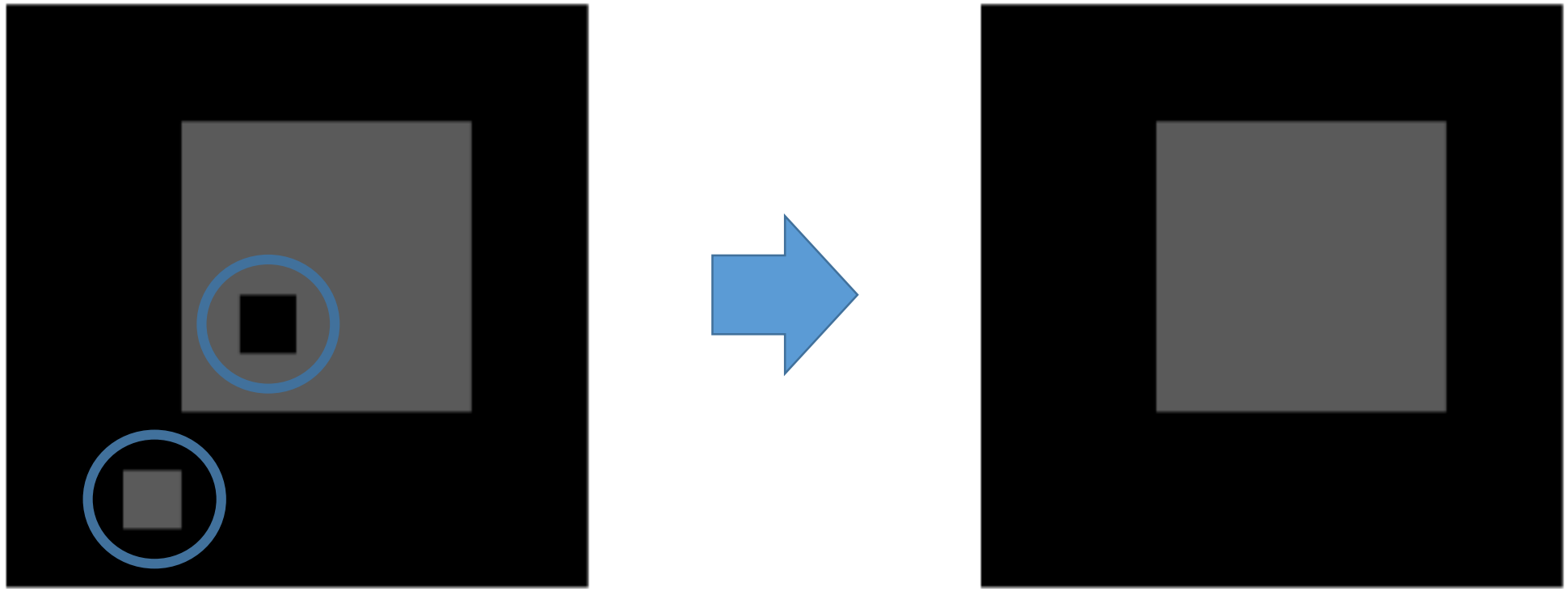
A person, thing, or fact that is very different from other people, things, or facts, so that it cannot be used to draw general conclusions

Source: https://dictionary.cambridge.org/it/dizionario/inglese/outlier



Smoothing

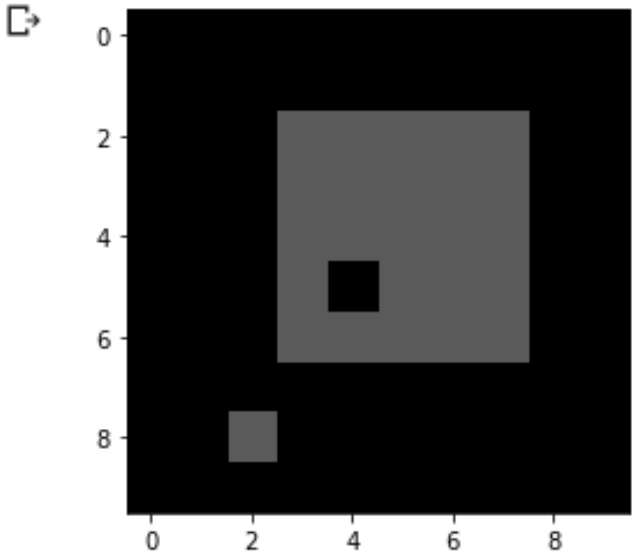
Una possibilità è quella di "smussare" l'immagine per ridurre il contrasto e nascondere gli outlier



Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Creiamo l’immagine in Colab

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

numpy_img = np.ones([10,10,3], dtype=np.uint8)*0
numpy_img[2:7,3:8] = 90
numpy_img[5,4] = 0
numpy_img[8,2] = 90

img = Image.fromarray(numpy_img)
_ = plt.imshow(img)
```



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

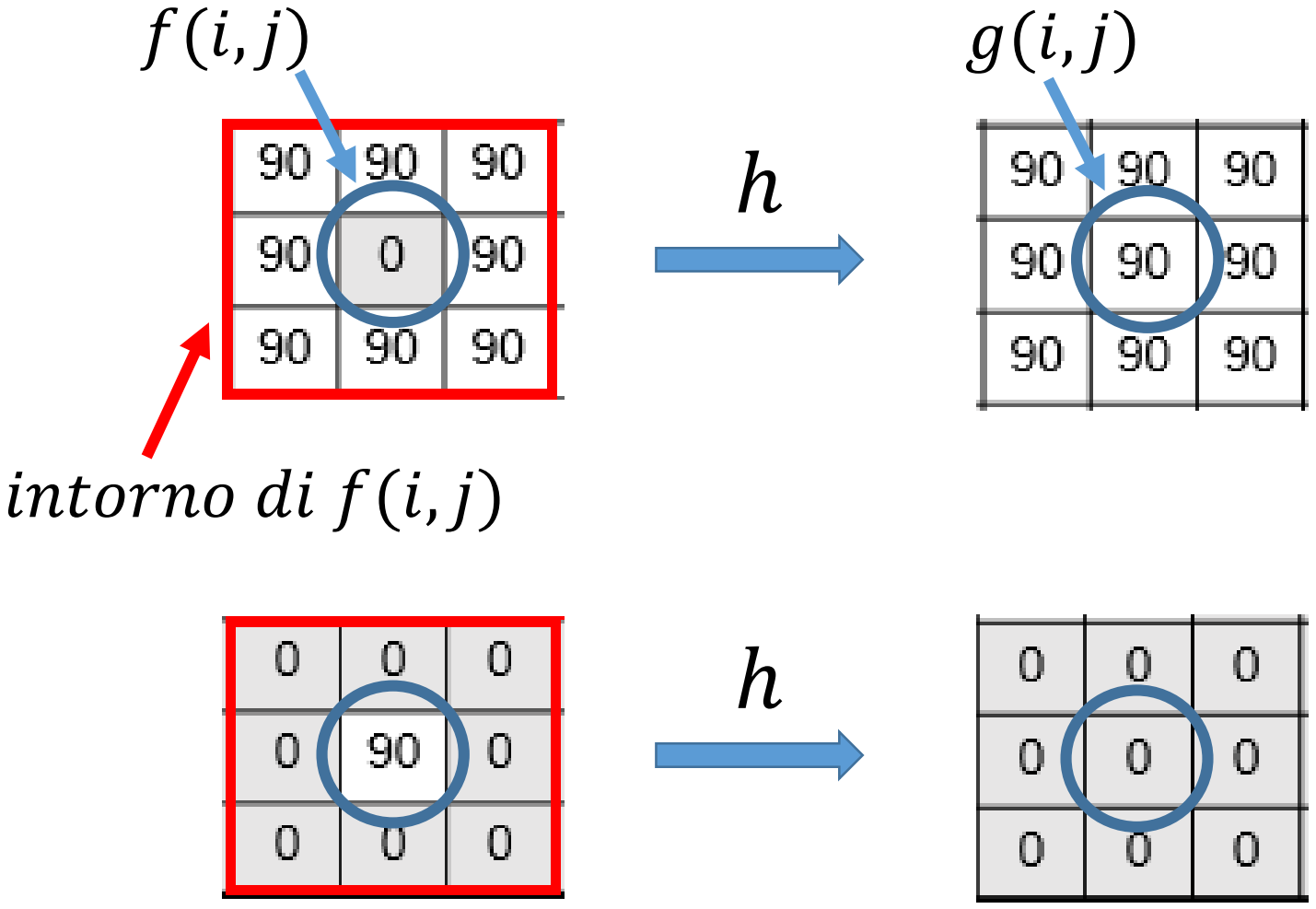
Filtering

Per smussare possiamo utilizzare una trasformazione *h* dell'immagine che modifichi i valori di intensità dei pixel in modo da renderli simili ai valori dei loro "vicini"

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Filtering



Il pixel f(i,j) verrà modificato attraverso una trasformazione locale h che coinvolgerà un intorno di f(i,j) per produrre il nuovo valore g(i,j)

Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Linear filtering

L’operazione di trasformare i valori del pixel *p* utilizzando una combinazione lineare pesata dei valori dei pixel in un intorno (piccolo) di *p* prende il nome di linear filtering

Cross-correlation filtering

- Si assuma di avere una finestra di filtraggio di dimensione (2k+1) x (2k+1)
- L'immagine filtrata g sarà

$$g(i,j) = \sum_{u,v=-k}^{k} f(i+u, j+v)h(u,v)$$

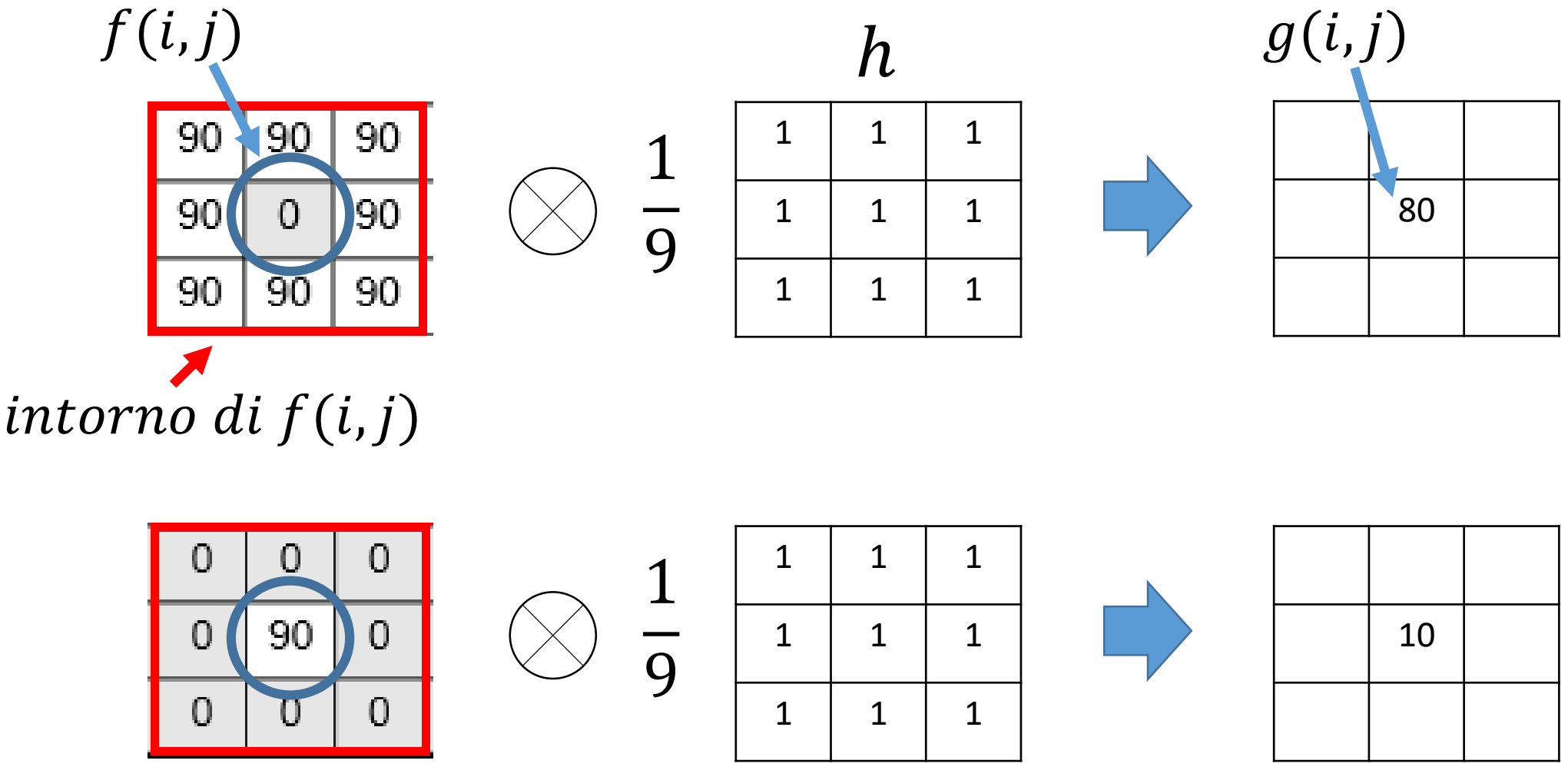
- Questa trasformazione è detta (cross-)correlazione e viene denotata con

$$g = f \otimes h$$

- $h(u,v)$ prende il nome di kernel o maschera (mask)

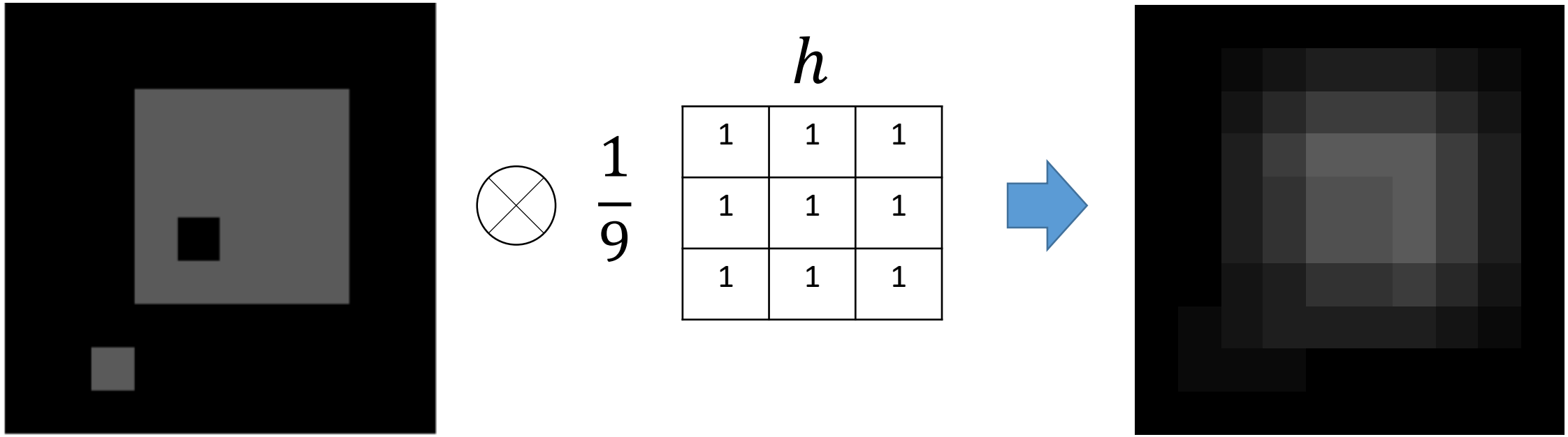
Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Mean kernel



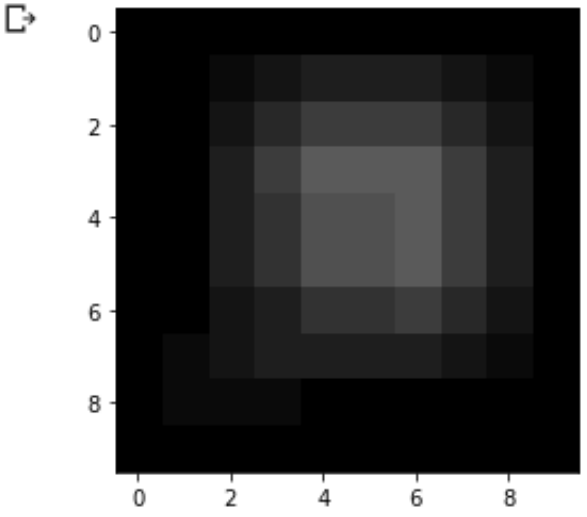
Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Mean filtering



Esempio mean filtering in Colab

```
f = numpy_img
g = np.ones([10,10,3], dtype=np.uint8)*0
k = 1
h = np.ones([2*k+1,2*k+1])*(1/pow(2*k+1,2))
for i in range(1,f.shape[0]-1):
  for j in range(1,f.shape[1]-1):
    sum_uv = 0
    for u in range(-k,k+1):
      for v in range(-k,k+1):
        sum_uv += (f[i+u,j+v] * h[u+k,v+k])
    g[i,j] = sum_uv

filtered = Image.fromarray(g)
_ = plt.imshow(filtered)
```



Abbiamo usato indici negative nella formula della correlazione. Nell'implementazione utilizziamo h(u+k,v+k) invece di h(u,v)

Convoluzione

Una variante della formula di correlazione è la seguente:

$$g(i,j) = \sum_{u,v=-k}^{k} f(i-u, j-v)h(u,v)$$

dove il segno degli offset in f è stato invertito.

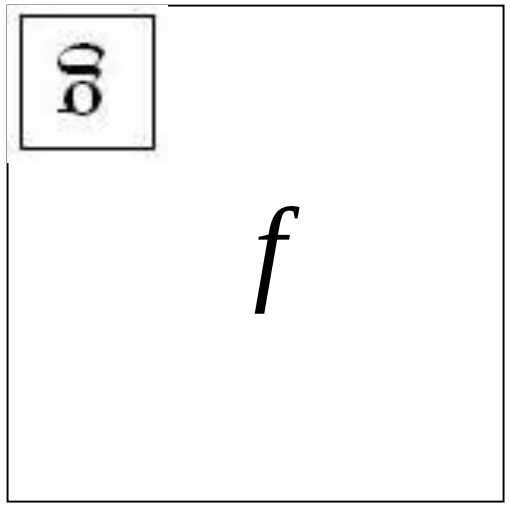
Questa operazione prende il nome di convoluzione ed è indicata come $g = f * h$

Convoluzione

Equivalentemente, otteniamo una convoluzione se applichiamo la formula

$$g(i,j) = \sum_{u,v=-k}^{k} f(i+u, j+v)h(-u,-v)$$

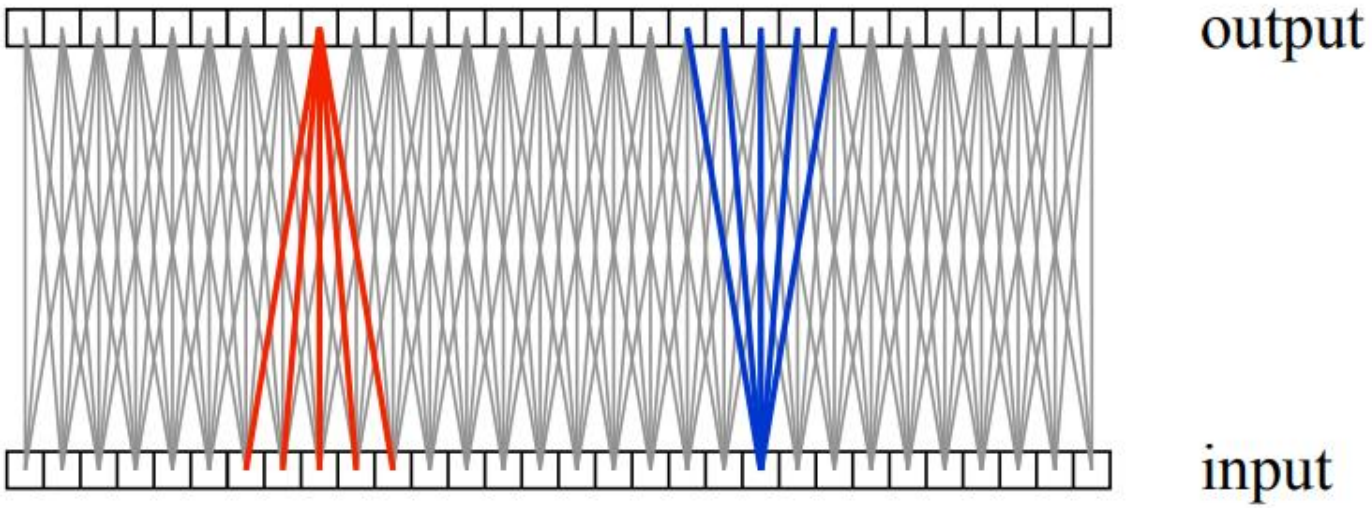
In pratica, andiamo a capovolgere sottosopra da sinistra a destra il kernel



Perché usare la convoluzione

Two pixels are connected by a gray edge if one contributes a value to the other. This is a symmetric view of convolution.

The blue view follows physics for image blur, and the view in red lends itself best to mathematical expression, because it reflects the definition of function.



Source: https://www2.cs.duke.edu/courses/spring19/compsci527/notes/convolution-filtering.pdf

Proprietà della convoluzione

La convoluzione è commutativa e associativa

$a * b = b * a$

$a * (b * c) = (a * b) * c$

Inoltre,

$(((a*b)*c)*d) = a*(b*c*d)$

Convoluzione: esempio

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

$*$

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

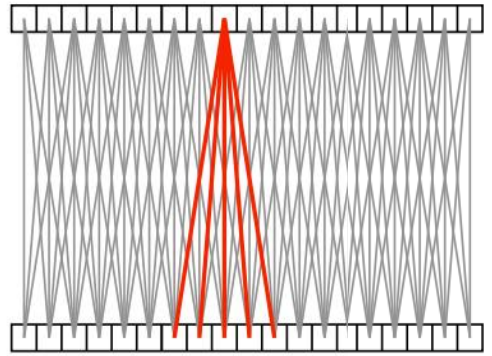
$=$

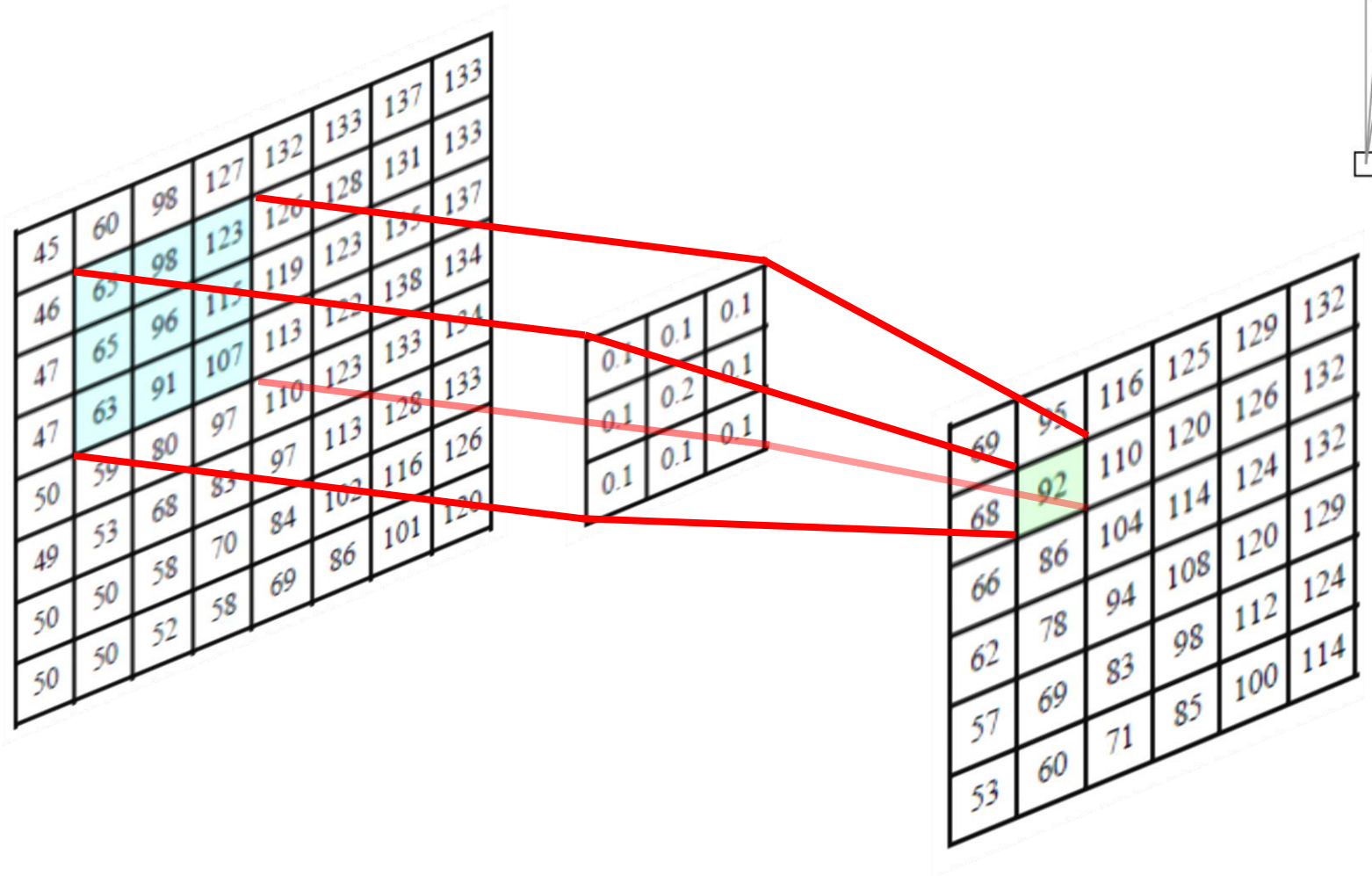
69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Convoluzione: esempio





Convoluzione

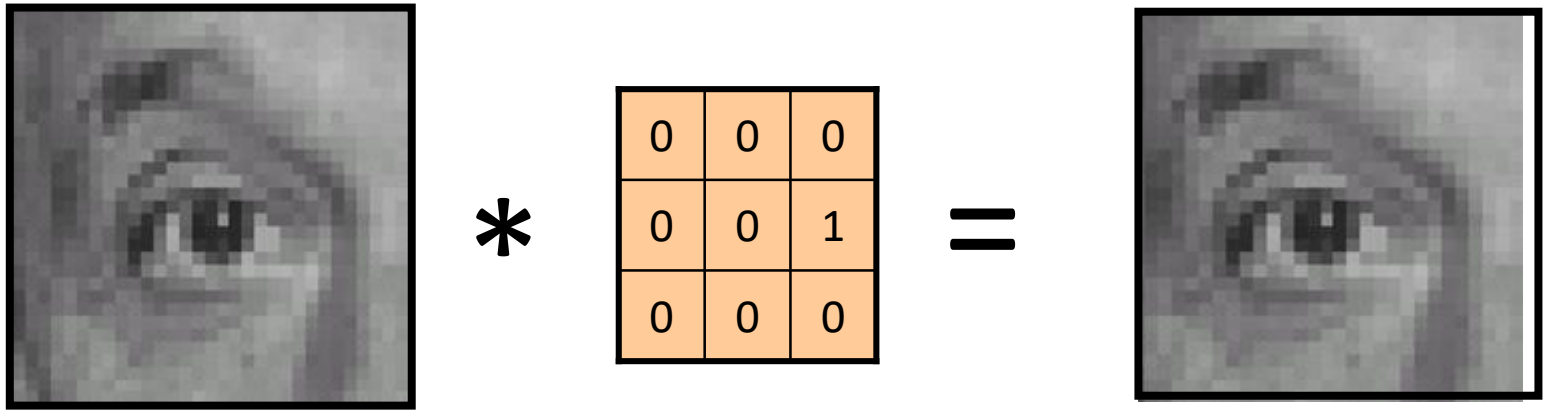
Original

$$*\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} =$$

Filtered
(no change)

Source: D. Lowe

Shift



Original

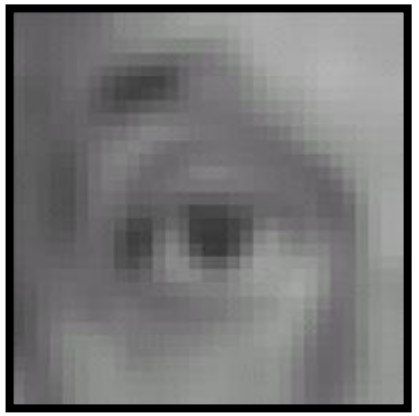
Shifted left
by 1 pixel

Source: D. Lowe

Blurring

Original

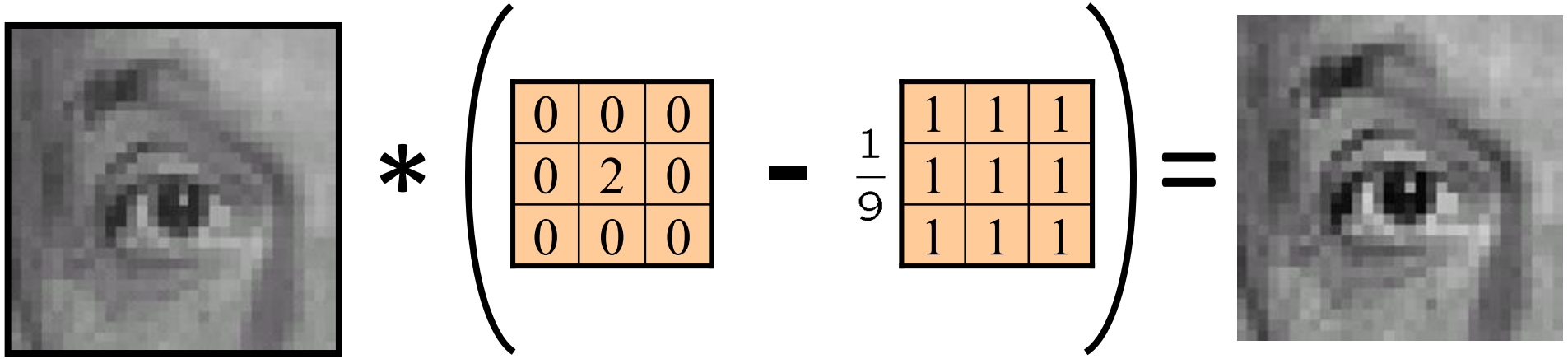
$$* \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$



Blur (with a box filter)

Source: D. Lowe

Sharpening



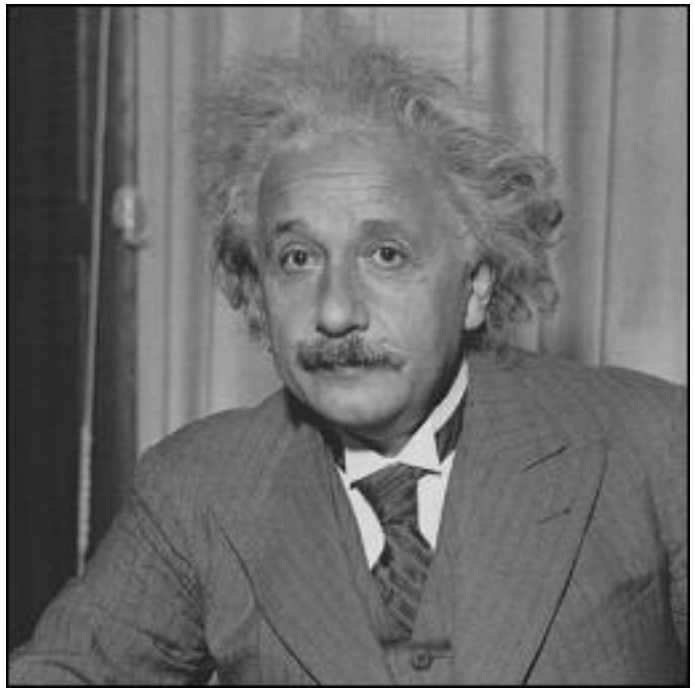
Original

Sharpening filter

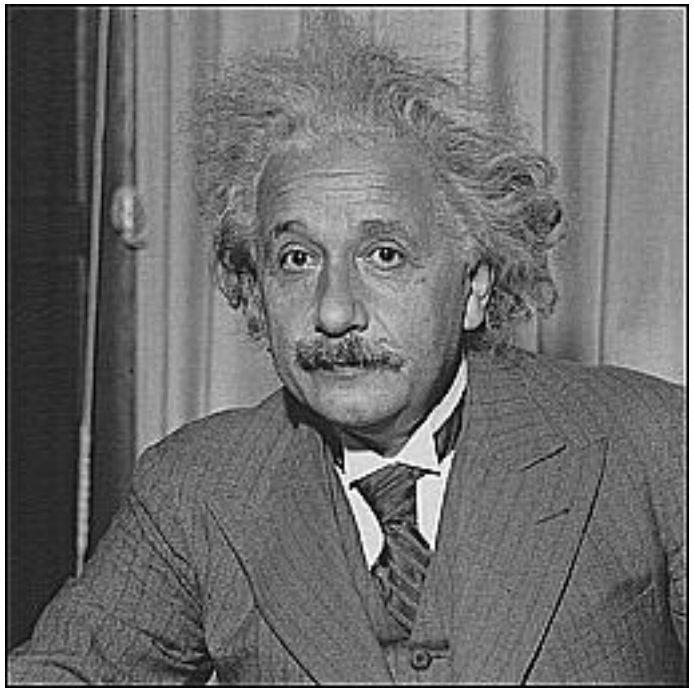
- Accentuates differences with local average
- Also known as Laplacian

Source: D. Lowe

Sharpening



before

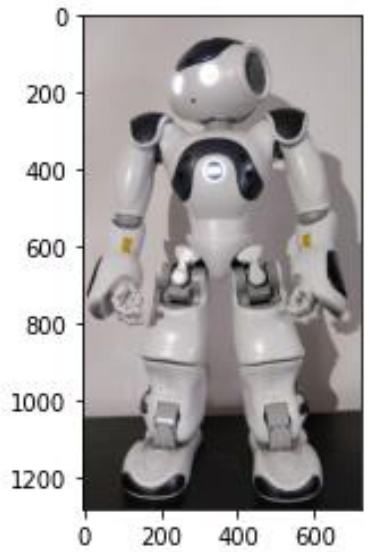


after

Slide credit: Bill Freeman

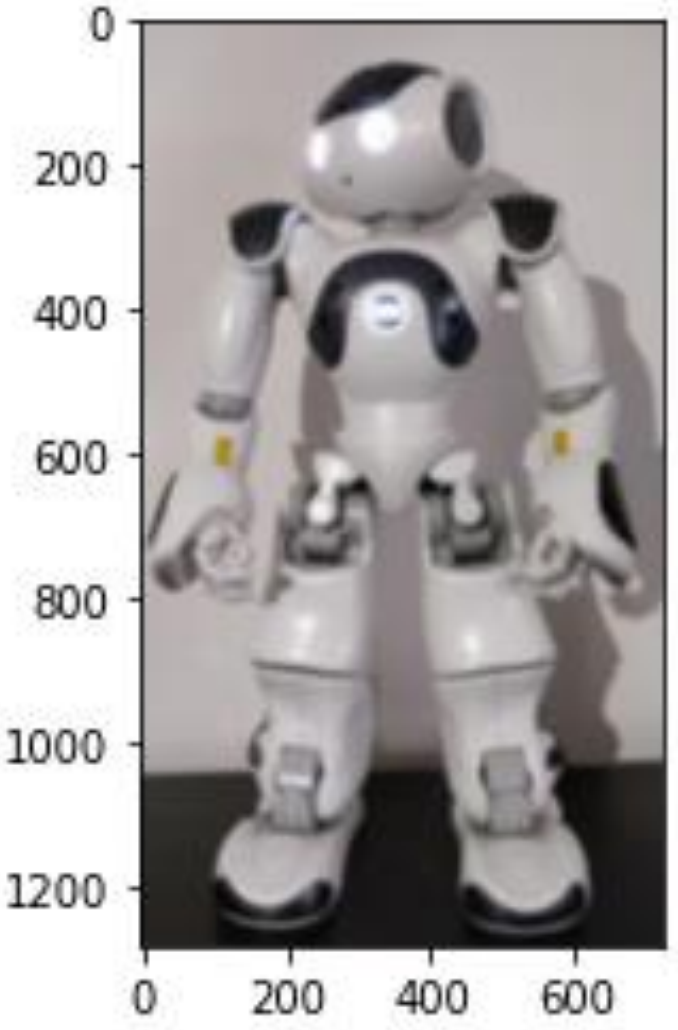
How to read an image from url

```python
from PIL import Image
import matplotlib.pyplot as plt
from urllib.request import urlopen

url = "https://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg"

img = Image.open(urlopen(url))

plt.grid(False)

_ = plt.imshow(img)
```



BoxBlur

```
from PIL import Image, ImageFilter
import matplotlib.pyplot as plt
from urllib.request import urlopen

url = "https://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg"

img = Image.open(urlopen(url))

box_blur_img = img.filter(ImageFilter.BoxBlur(5)) # 11x11 kernel

_ = plt.imshow(box_blur_img)
```



https://pillow.readthedocs.io/en/5.1.x/reference/ImageFilter.html

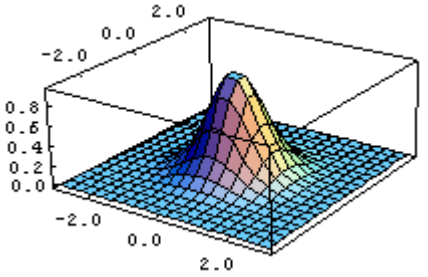
Gaussian blurring

Un kernel Gaussiano darà meno peso ai pixel distanti dal centro della finestra

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Questo kernel è una approssimazione della funzione Gaussiana: $h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$

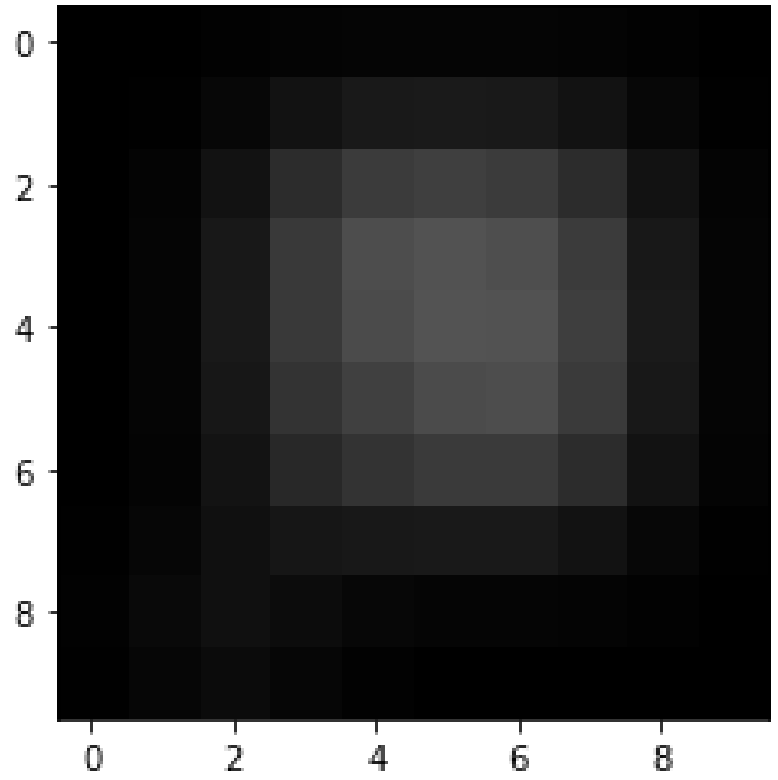
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

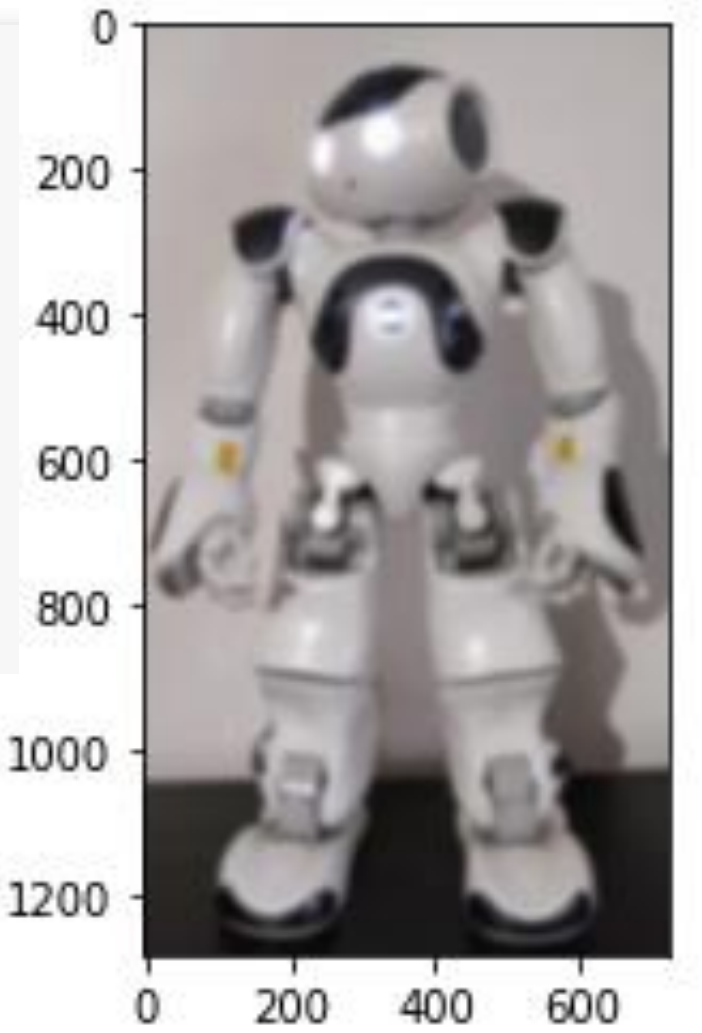
Gaussian blurring

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

numpy_img = np.ones([10,10,3], dtype=np.uint8)*0
numpy_img[2:7,3:8] = 90
numpy_img[5,4] = 0
numpy_img[8,2] = 90

img = Image.fromarray(numpy_img)
blur_img = img.filter(ImageFilter.GaussianBlur(1))
_ = plt.imshow(blur_img)
```



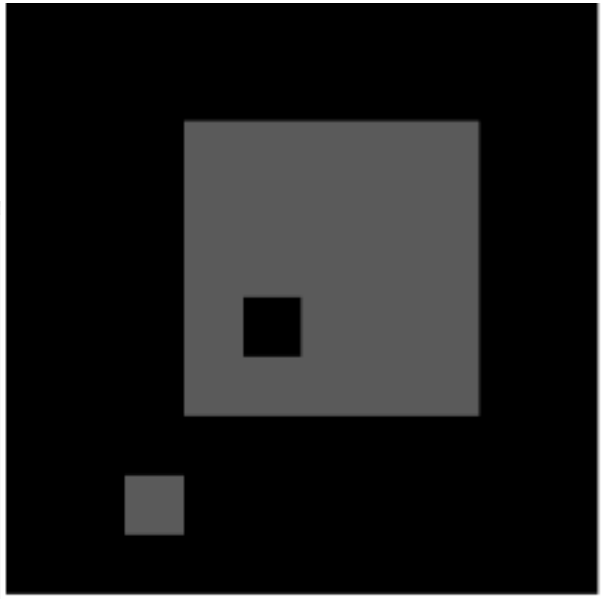
Gaussian blurring

```
from PIL import Image, ImageFilter
import matplotlib.pyplot as plt
from urllib.request import urlopen

url = "https://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg"

img = Image.open(urlopen(url))

G_blur_img = img.filter(ImageFilter.GaussianBlur(5)) # 11x11 kernel

_ = plt.imshow(G_blur_img)
```



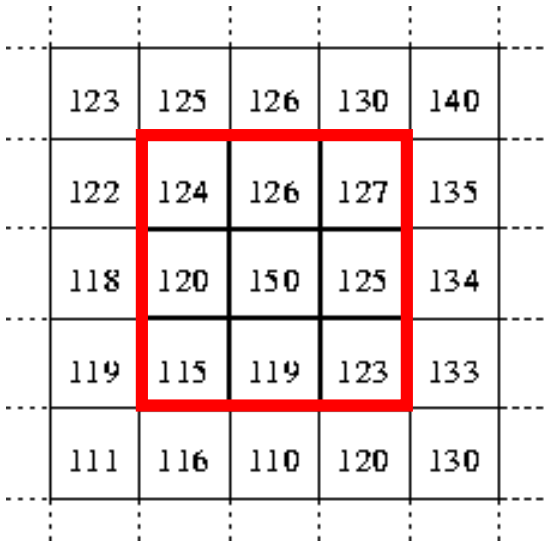
Median filter

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

numpy_img = np.ones([10,10,3], dtype=np.uint8)*0
numpy_img[2:7,3:8] = 90
numpy_img[5,4] = 0
numpy_img[8,2] = 90

img = Image.fromarray(numpy_img)
filtered_img = img.filter(ImageFilter.MedianFilter(3))
_ = plt.imshow(filtered_img)
```



Median filter: esempio

Per realizzare il median filter andremo ad ordinare tutti i valori presenti nell'intorno del pixel su cui è applicato il filtro e poi prenderemo il valore che si trova nel mezzo del vettore ordinato



Neighbourhood values:

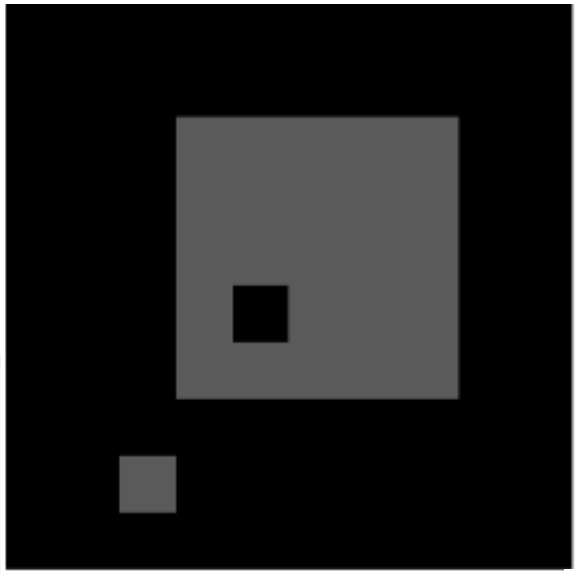
115, 119, 120, 123, 124, 125, 126, 127, 150

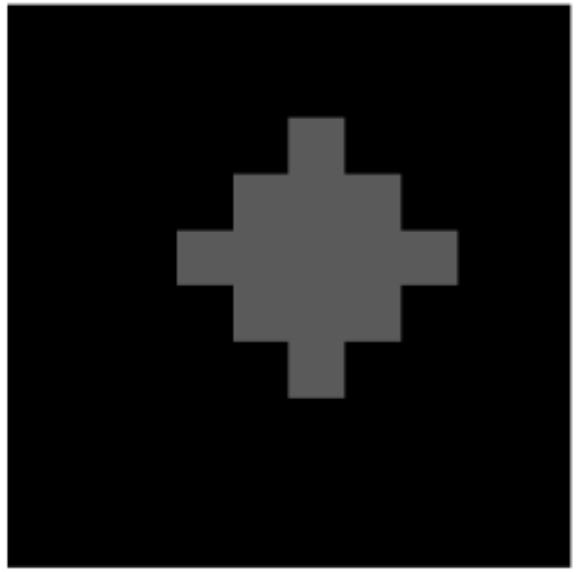
Median value: 124

Souce: https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm

Median filter

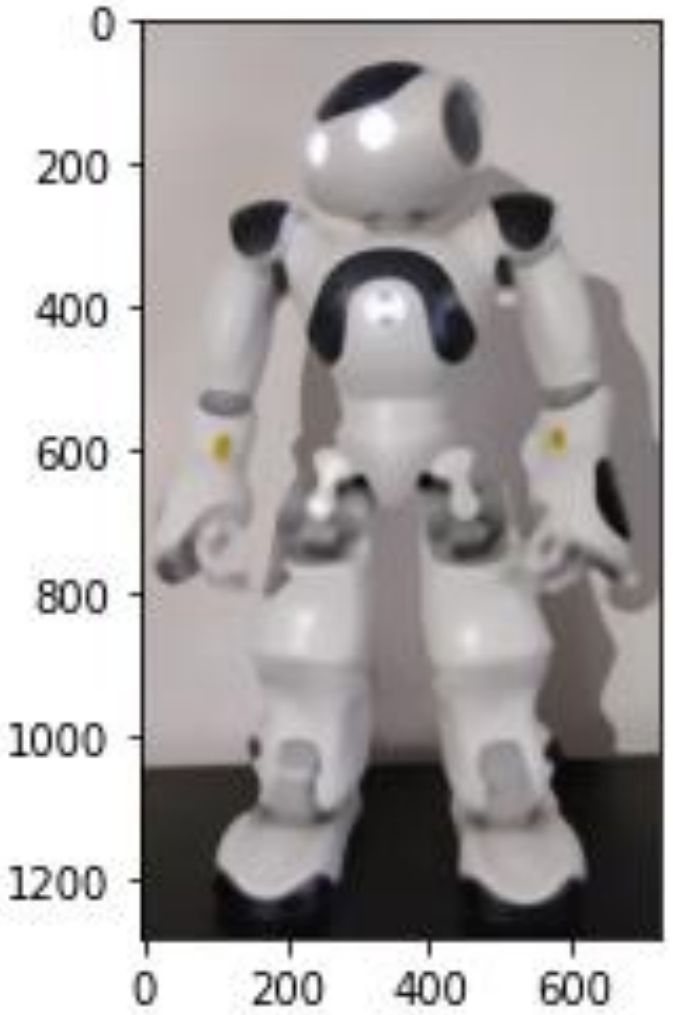
```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

numpy_img = np.ones([10,10,3], dtype=np.uint8)*0
numpy_img[2:7,3:8] = 90
numpy_img[5,4] = 0
numpy_img[8,2] = 90

img = Image.fromarray(numpy_img)
filtered_img = img.filter(ImageFilter.MedianFilter(5))
_ = plt.imshow(filtered_img)
```





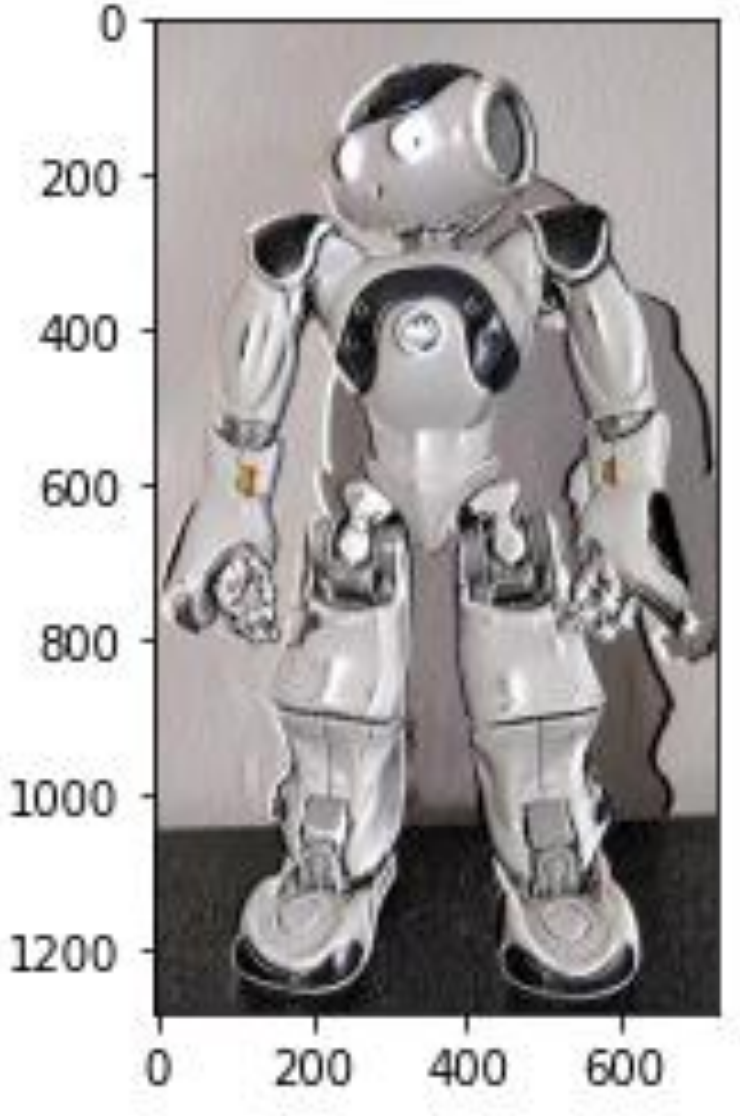
Median filter

```python
from PIL import Image, ImageFilter
import matplotlib.pyplot as plt
from urllib.request import urlopen

url = "https://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg"

img = Image.open(urlopen(url))

median_img = img.filter(ImageFilter.MedianFilter(21))

_ = plt.imshow(median_img)
```



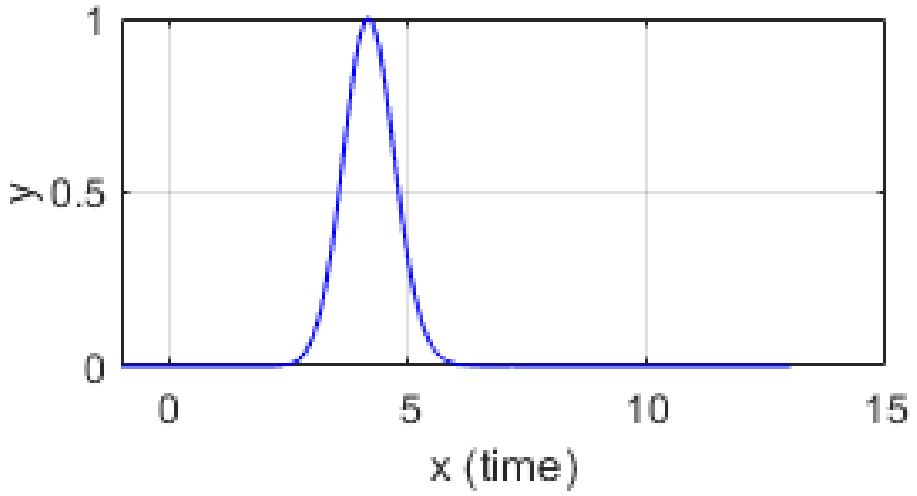
Custom filters

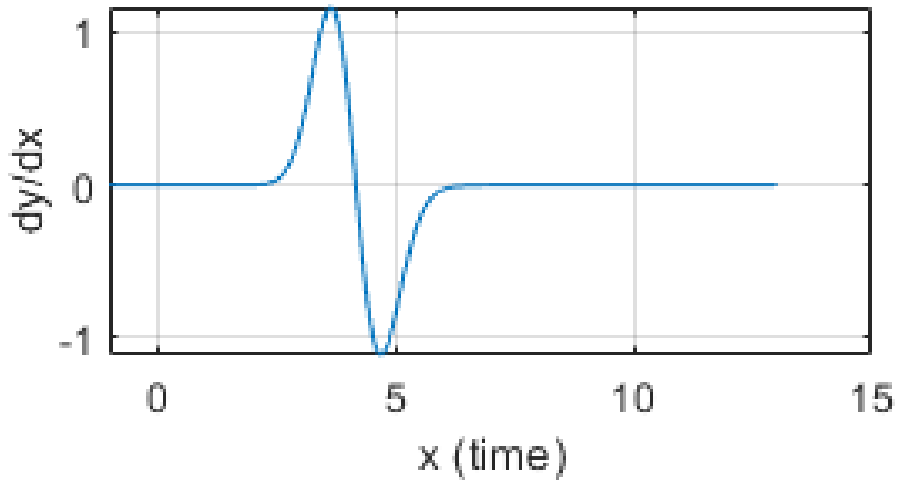
```
from PIL import Image, ImageFilter
import matplotlib.pyplot as plt
import urllib.request

url = "http://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg"
img = Image.open(urllib.request.urlopen(url))

filtered_img = img.filter(ImageFilter.Kernel((3,3),[11,0,-11,15,0,-15,1,0,1]))
_ = plt.imshow(filtered_img)
```



Derivata di un segnale monodimensionale

- La derivata di un segnale monodimensionale (per esempio, la voce) permette di misurarne la variabilità
  - forti variazioni locali → valori elevati per la derivata
  - nessuna variazione (segnale costante) → derivata nulla





Derivata di un segnale monodimensionale

- Per calcolare la derivata in x_0 di un segnale monodimensionale *f(x)*, possiamo scrivere:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

- Nel caso di un segnale discreto, avremo:

$$f'(x_0) \approx \frac{f(x_0 + 1) - f(x_0)}{1}$$

Derivata di una immagine

- Le immagini sono segnali bidimensionali discreti
- Andremo quindi a considerare le derivate parziali:

$$\frac{\partial f}{\partial x}[i,j] \approx f[i+1,j] - f[i,j]$$

$$\frac{\partial f}{\partial y}[i,j] \approx f[i,j+1] - f[i,j]$$

Derivata come convoluzione

Per una funzione 2D $f(x,y)$ avremo:

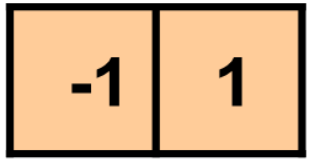
$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \left(\frac{f(x+\varepsilon, y)}{\varepsilon} - \frac{f(x,y)}{\varepsilon} \right)$$

che è una operazione lineare invariante rispetto allo shift (così come la convoluzione)

Possiamo utilizzare una approssimazione

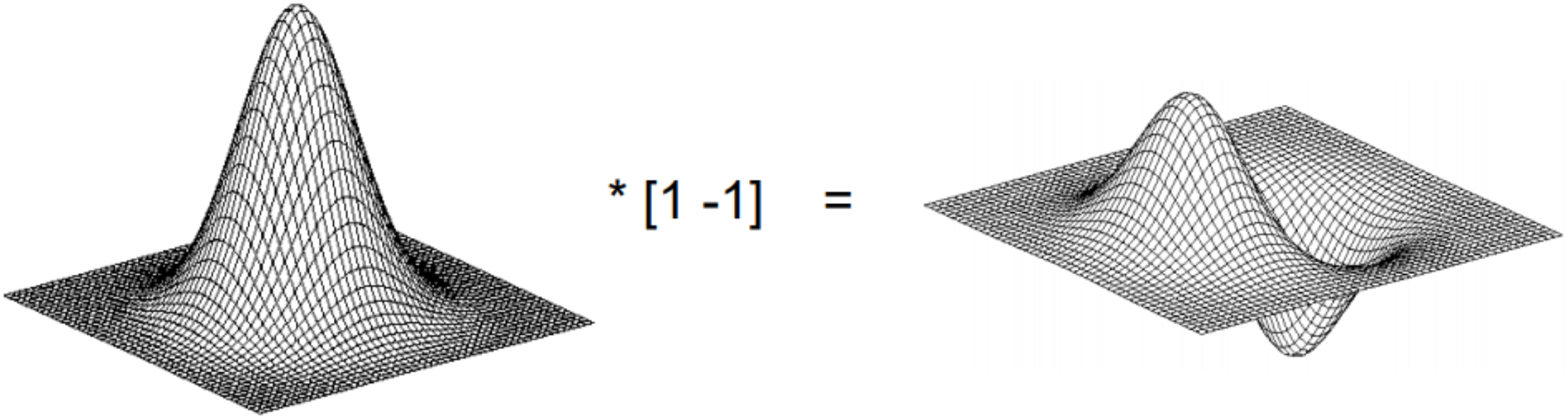
$$\frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

che può essere calcolata con una convoluzione



Source: D. Forsyth, D. Lowe

Derivata come convoluzione



https://inst.eecs.berkeley.edu/~cs194-26/fa17/Lectures/ConvEdgesTemplate.pdf

Gradiente

- Il gradiente è il vettore le cui componenti sono le derivate parziali nelle diverse direzioni

- Nel caso di immagini, avremo un gradiente a due componenti

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

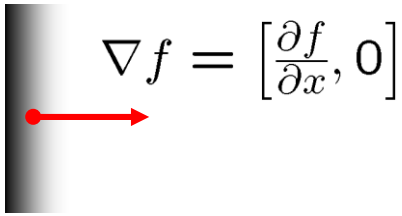
Gradiente

Gradiente dell'immagine:

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

$$\frac{\partial f}{\partial x}[i,j] \approx f[i+1,j] - f[i,j]$$

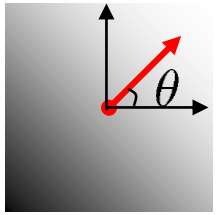
$$\frac{\partial f}{\partial y}[i,j] \approx f[i,j+1] - f[i,j]$$



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

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

Il gradiente punta nella direzione del più rapido cambio di intensità

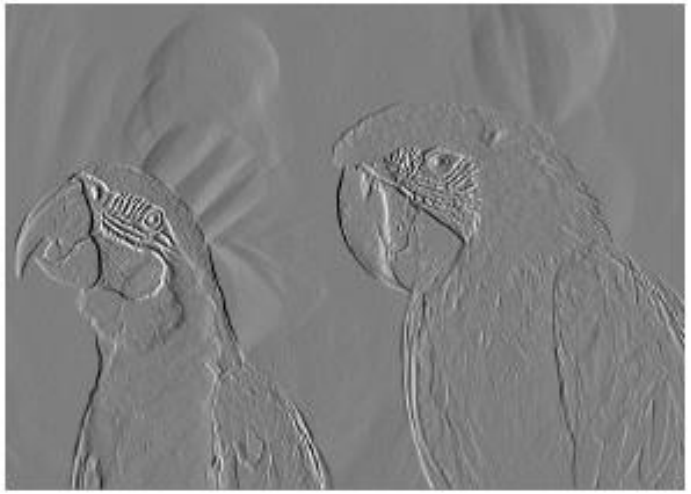


$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

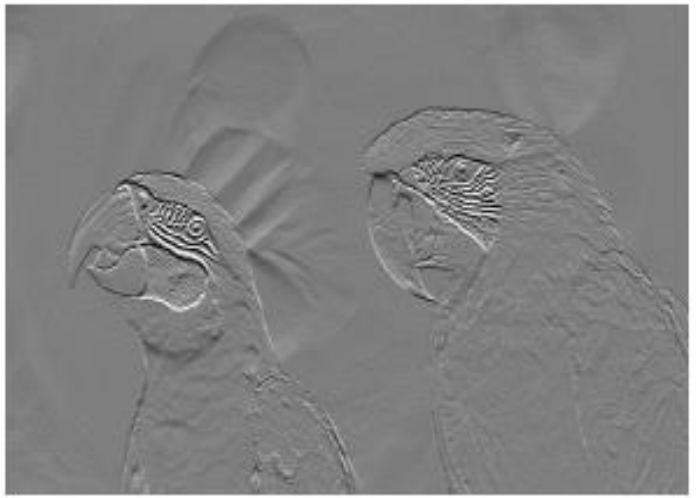
Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Esempio gradiente

f



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



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

Source: https://www.uio.no/studier/emner/matnat/its/UNIK4690/v16/forelesninger/lecture_2_1_filtering.pdf

Direzione del gradiente

- La direzione del gradiente è data da:

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

- Il modulo del gradiente è dato da:

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

il modulo fornisce una misura della “forza” di un bordo

Gradient magnitude

f

$||\nabla f||$

Source: https://www.uio.no/studier/emner/matnat/its/UNIK4690/v16/forelesninger/lecture_2_1_filtering.pdf

Sobel operator

$$S_x \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad S_y \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

L’operatore di Sobel fa uso di due kernel 3x3 per calcolare, tramite convoluzione, il valore approssimato delle derivate in direzione orizzontale e in direzione verticale

Skimage (scikit-image)

scikit-image.org

scikit-image
image processing in python

Download Gallery Documentation Community Guidelines Source Search documentation ...

Stable (release notes)
0.16.2 - October 2019

Download

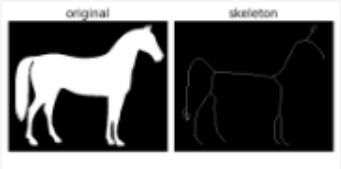
Development
pre-0.17

Download

GitHub *source & bug reports*
Contribute *get involved*
Mailing List *dev. discussion*
Forum *advice & community*
StackOverflow *code help*

Image processing in Python

scikit-image is a collection of algorithms for image processing. It is available free of charge and free of restriction. We pride ourselves on high-quality, peer-reviewed code, written by an active community of volunteers.

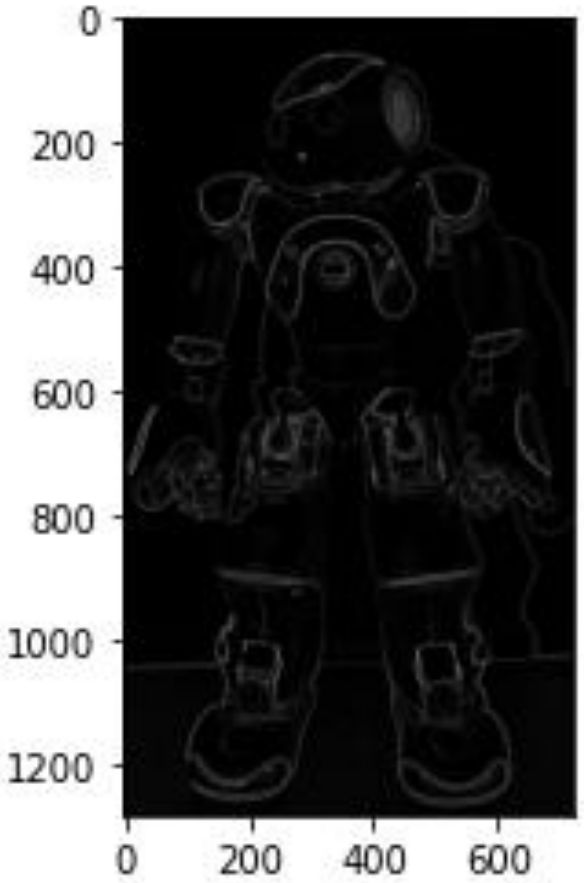


If you find this project useful, please cite: [BiBTeX]

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors. **scikit-image: Image processing in Python**. PeerJ 2:e453 (2014) https://doi.org/10.7717/peerj.453

Esempio Sobel edge detection

```
from PIL import Image
import matplotlib.pyplot as plt
from urllib.request import urlopen
from skimage import filters

url = "http://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg"
img = np.array(Image.open(urlopen(url)).convert('L'));

edges = filters.sobel(img)

_ = plt.imshow(edges, cmap="gray")
```



Canny edge detection

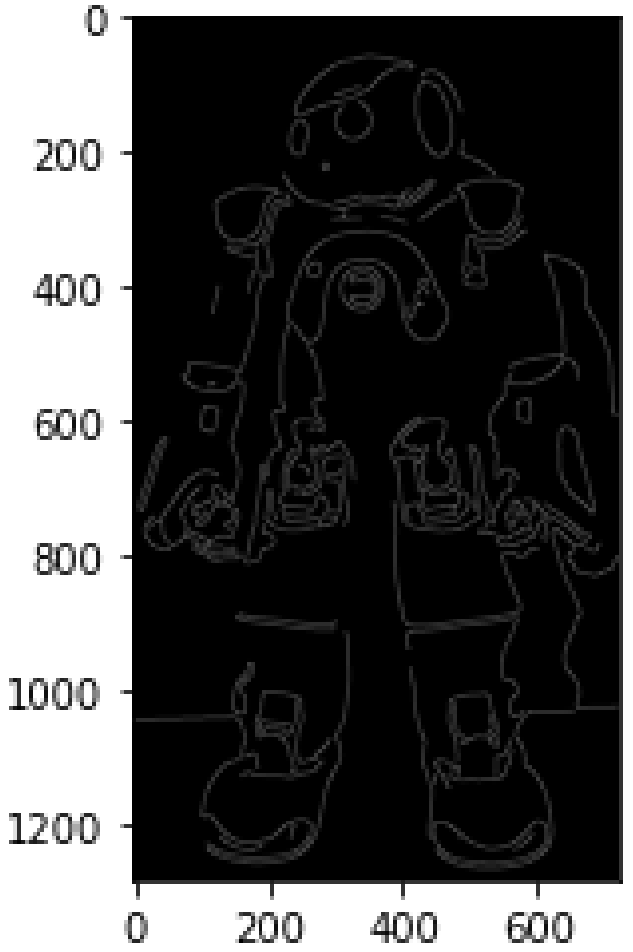
- Smooth image *I* with 2D Gaussian $G * I$
- Find local edge normal directions for each pixel $\bar{\mathbf{n}} = \dfrac{\nabla(G * I)}{|\nabla(G * I)|}$
- Compute edge magnitudes $|\nabla(G * I)|$
- Locate edges by finding zero-crossings along the edge normal directions (**non-maximum suppression**)

$$\frac{\partial^2(G * I)}{\partial \bar{\mathbf{n}}^2} = 0$$

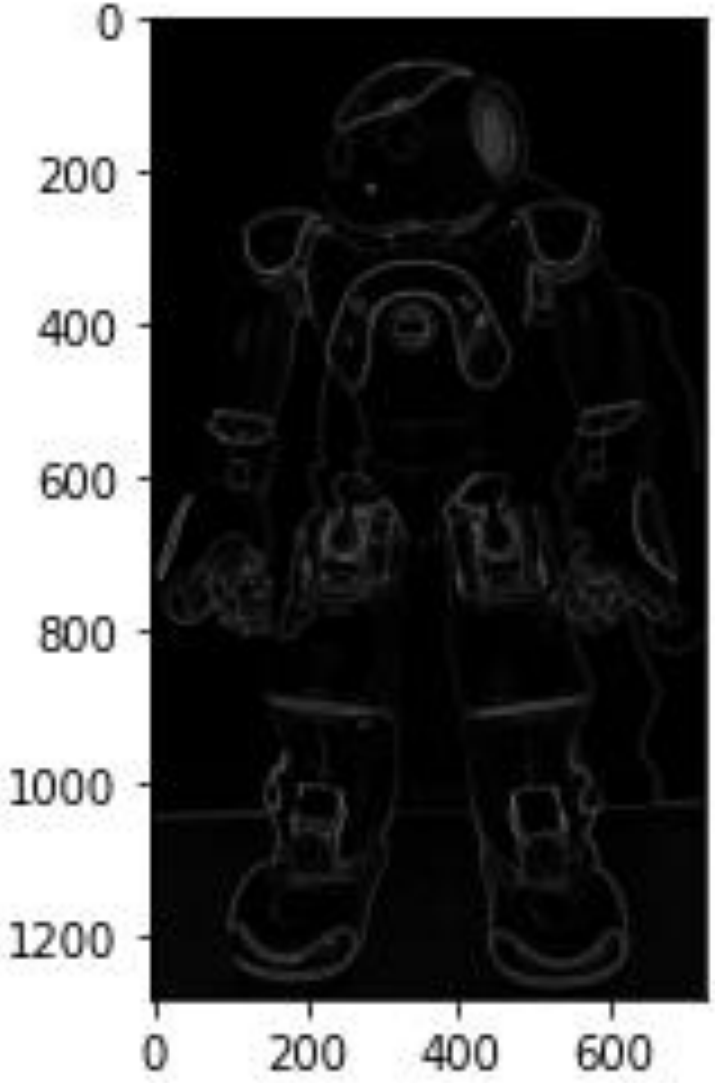
https://www.cs.cmu.edu/afs/cs/academic/class/15385-s06/lectures/ppts/lec-7.ppt

Esempio Canny edge detection

```
from PIL import Image
import matplotlib.pyplot as plt
from urllib.request import urlopen
from skimage import feature

url = "http://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg"
img = np.array(Image.open(urlopen(url)).convert('L'));

edges = feature.canny(img, sigma=3)

_ = plt.imshow(edges, cmap="gray")
```



Sobel vs. Canny



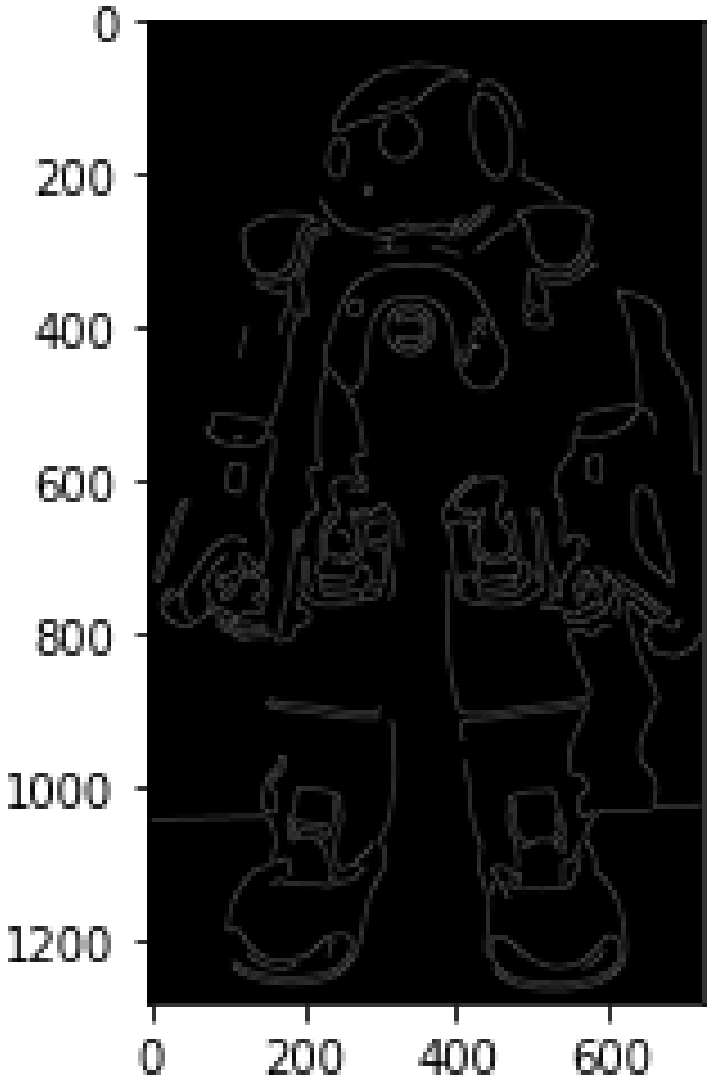
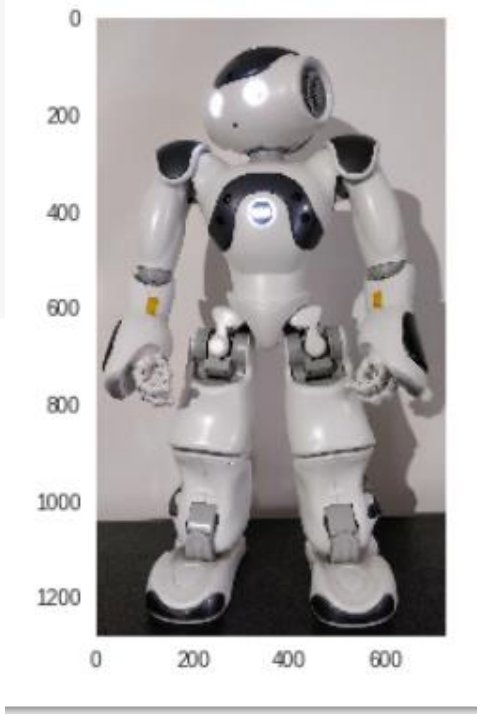
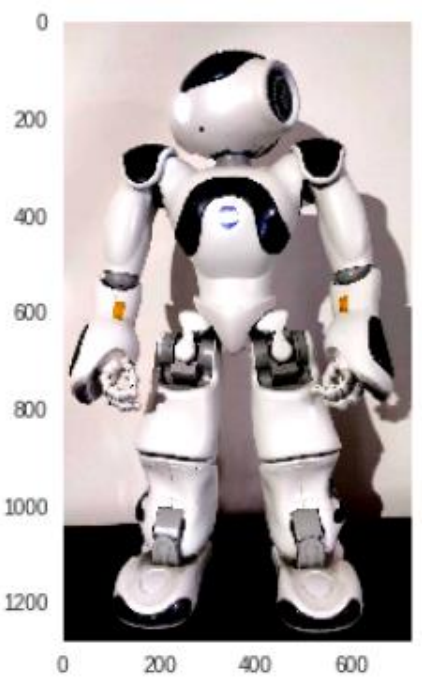


Image enhancement

```
import matplotlib.pyplot as plt
import urllib.request

from PIL import Image
from PIL import ImageEnhance

url = "https://dbloisi.github.io/corsi/images/nao-v6-spqr.jpg"

img = Image.open(urllib.request.urlopen(url))

enhancer = ImageEnhance.Contrast(img)
new_img = enhancer.enhance(2)

plt.grid(b=False)
plt.imshow(new_img)
```





Esercizio 1

Applicare i filtri di Sobel e Canny sull'immagine
http://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg
modificata tramite il contrast enhancement

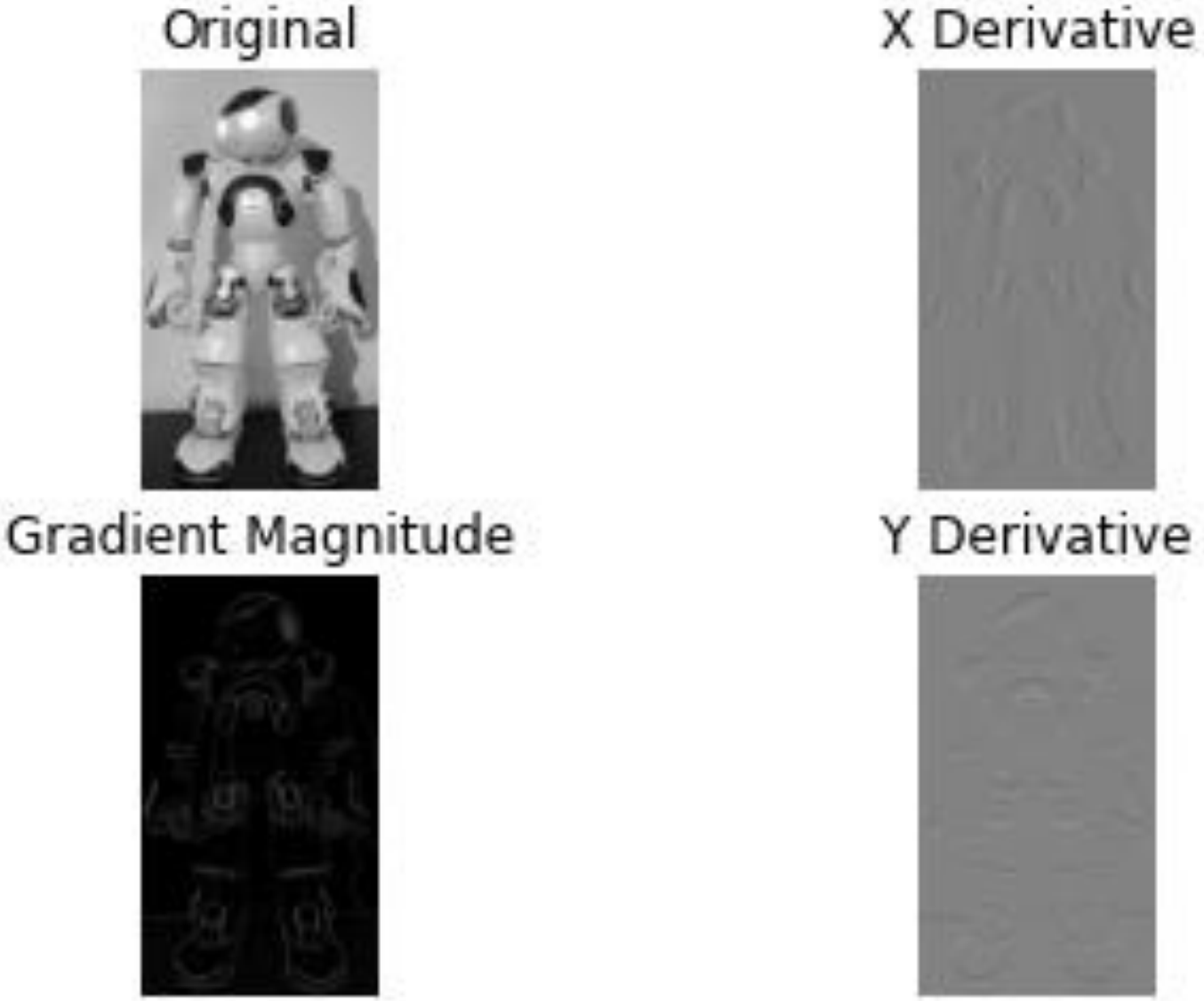
Esercizio 2

Provare a modificare l'immagine
http://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg
tramite cambio della **brightness**

https://pillow.readthedocs.io/en/stable/reference/ImageEnhance.html#PIL.ImageEnhance.PIL.ImageEnhance.Brightness

Esercizio 3

Usando l’immagine originale http://web.unibas.it/bloisi/corsi/images/nao-v6-spqr.jpg generare l’output a lato



UNIVERSITÀ DEGLI STUDI DELLA BASILICATA



Corso di Visione e Percezione

Docente
Domenico D. Bloisi

Filtri