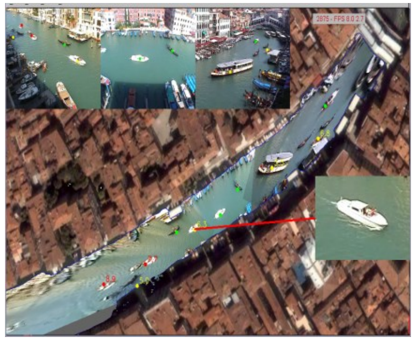
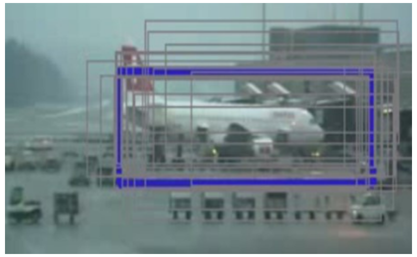
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

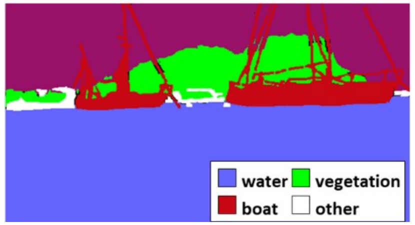


Corso di Sistemi Informativi
A.A. 2018/19

Docente
Domenico Daniele Bloisi

Filtri

Marzo 2019

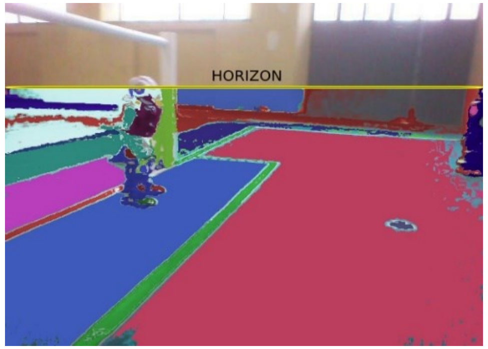
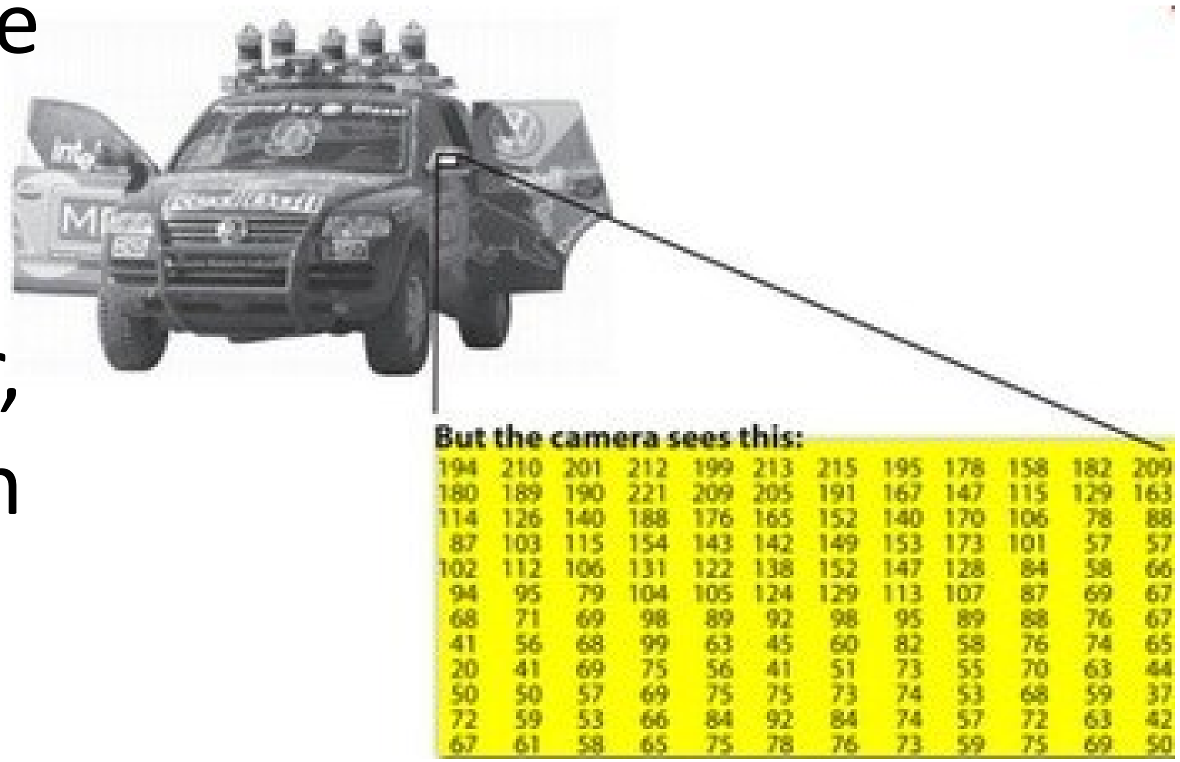




Immagine Digitale

- 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)



opencv.org

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 range predefinito
  $f: [a,b]x[c,d] \rightarrow [0,1]$

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

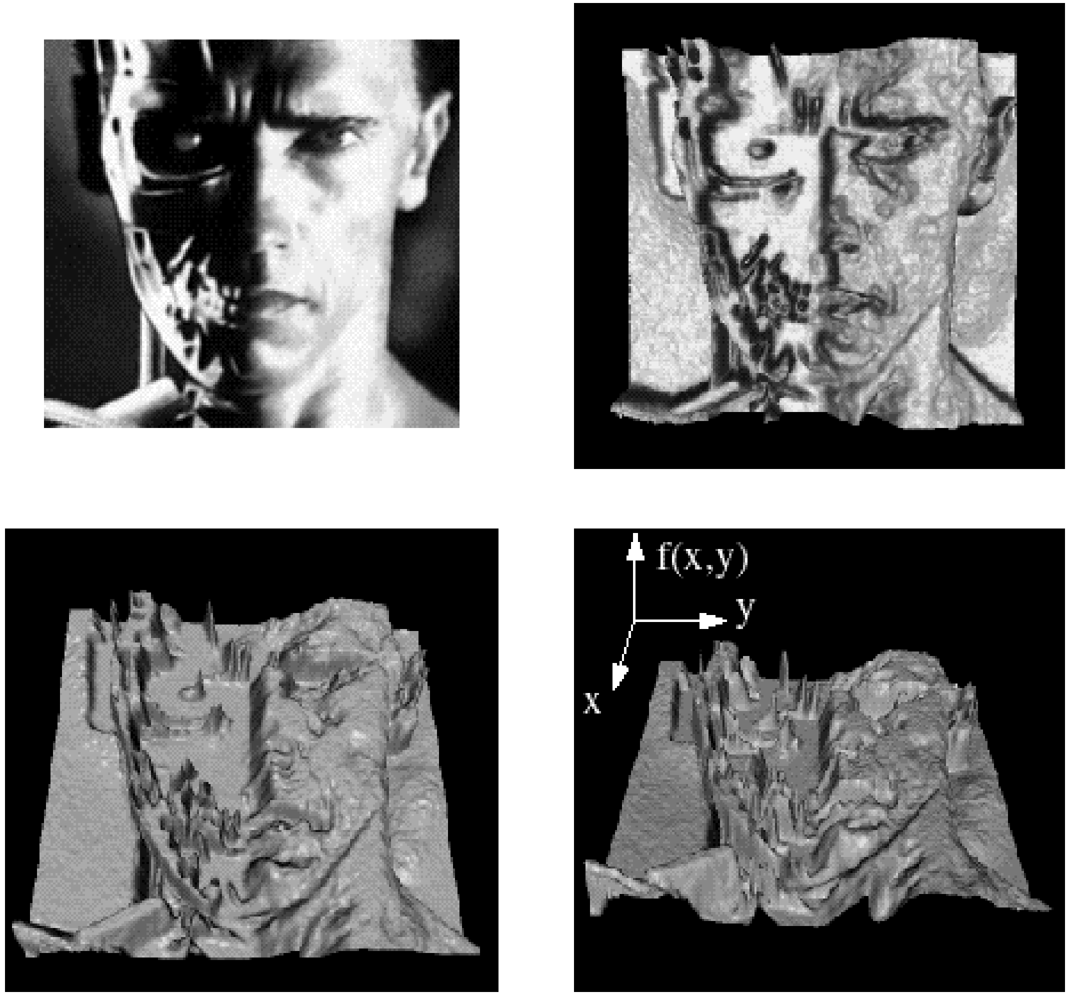
Immagine a colori

Una immagine a colori potrà essere rappresentata come l’unione tra tre funzioni, una per ogni canale 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

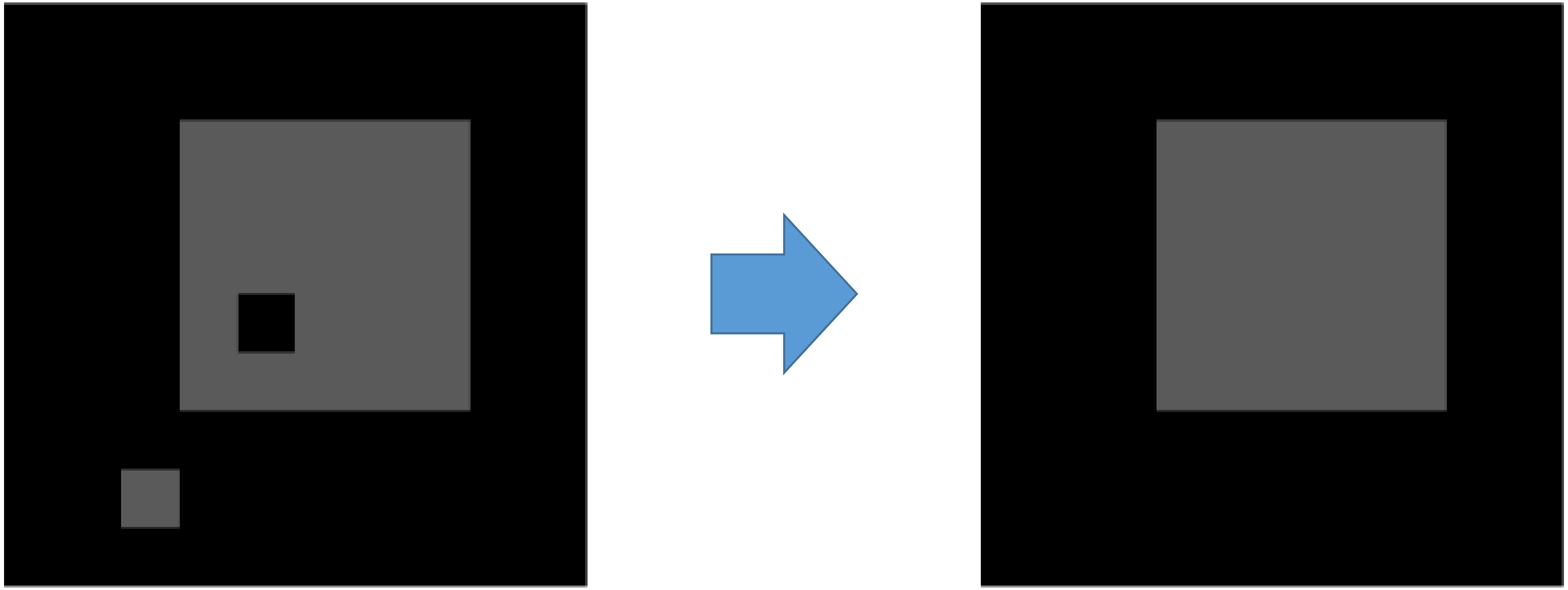
Esempio



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

Mean filter

Vogliamo limitare il rumore presente nell’immagine sotto a sinistra per trasformarla in quella a destra

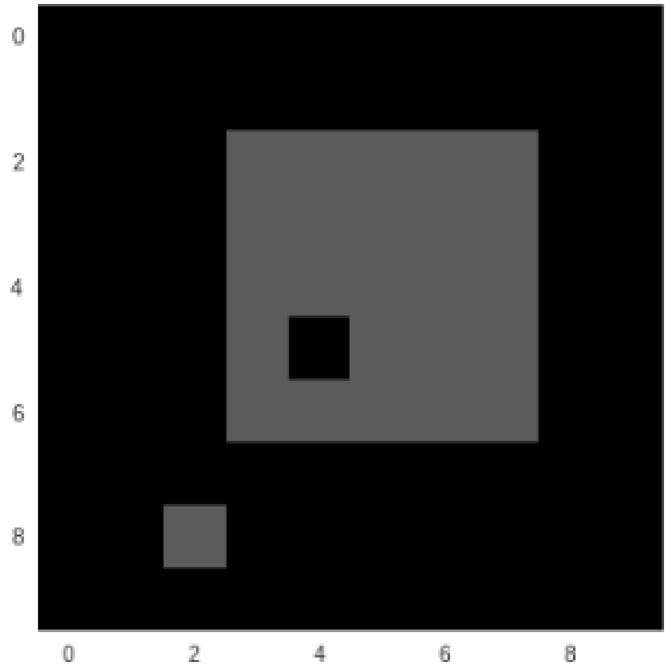


Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt

Mean filter: creiamo l’immagine

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

```
<matplotlib.image.AxesImage at 0x7f0d51201d68>
```



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

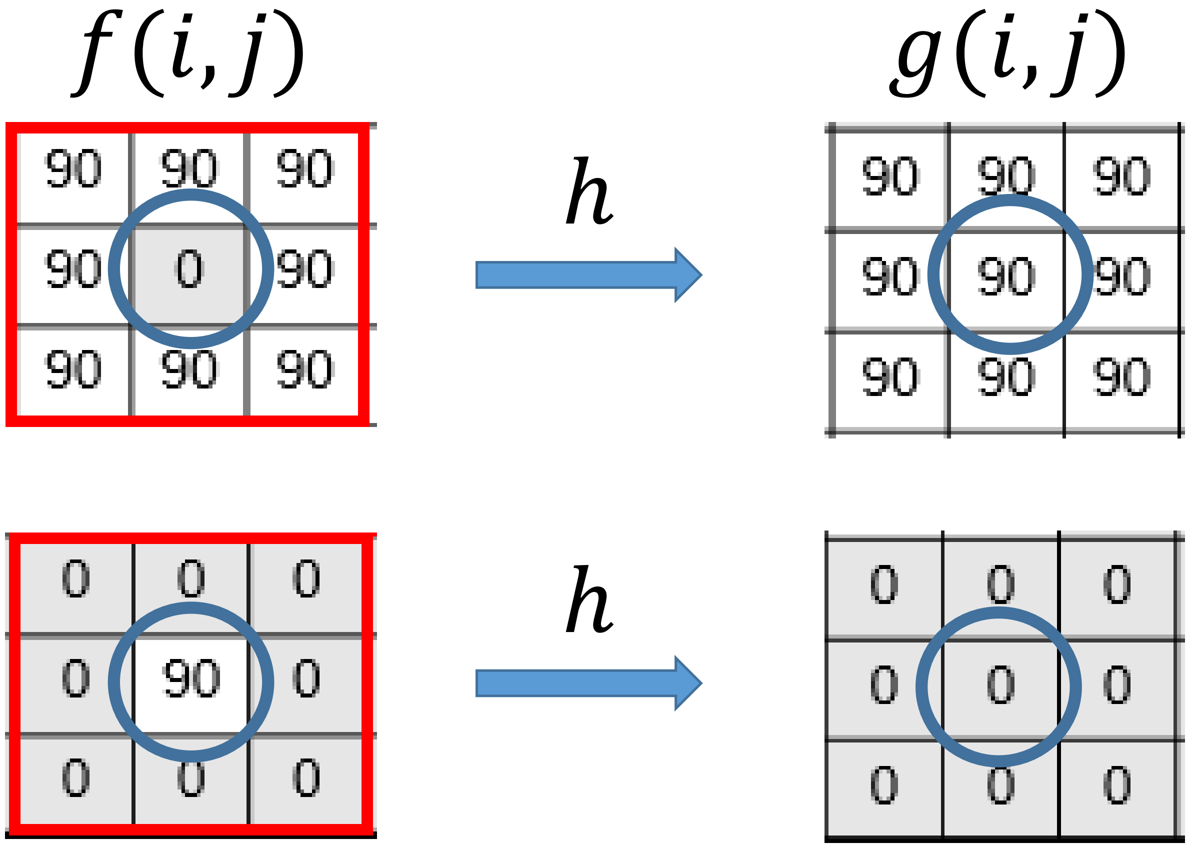
Mean filter

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](https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt)

Mean 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](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

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

- *h(k,l)* prende il nome di kernel o maschera (mask)
- Questa trasformazione è detta correlazione e può essere denotata con $g = f \otimes h$

[Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt](https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt)

Mean kernel

$$\overset{f(i,j)}{\begin{bmatrix}90&90&90\\90&0&90\\90&90&90\end{bmatrix}} \otimes \frac{1}{9}\overset{h}{\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}} \Rightarrow \overset{g(i,j)}{\begin{bmatrix} & & \\ & 80 & \\ & & \end{bmatrix}}$$

$$\begin{bmatrix}0&0&0\\0&90&0\\0&0&0\end{bmatrix} \otimes \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \Rightarrow \begin{bmatrix} & & \\ & 10 & \\ & & \end{bmatrix}$$

[Source: https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt](https://courses.cs.washington.edu/courses/cse576/08sp/lectures/filter.ppt)

Convoluzione

Una variante della formula di correlazione è la seguente

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

dove il segno degli offset in *f* è stato invertito.

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

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

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

$h(x,y)$

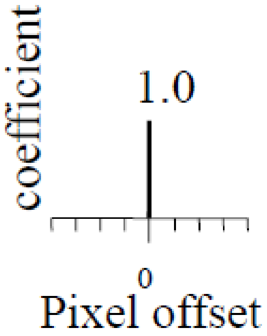
=

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

$g(x,y)$

Convoluzione

original



$$\delta$$

Filtered
(no change)

$$f = f \otimes \delta$$

Source: Convolution - Fredo Durand - MIT EECS 6.815/6.865

Convoluzione

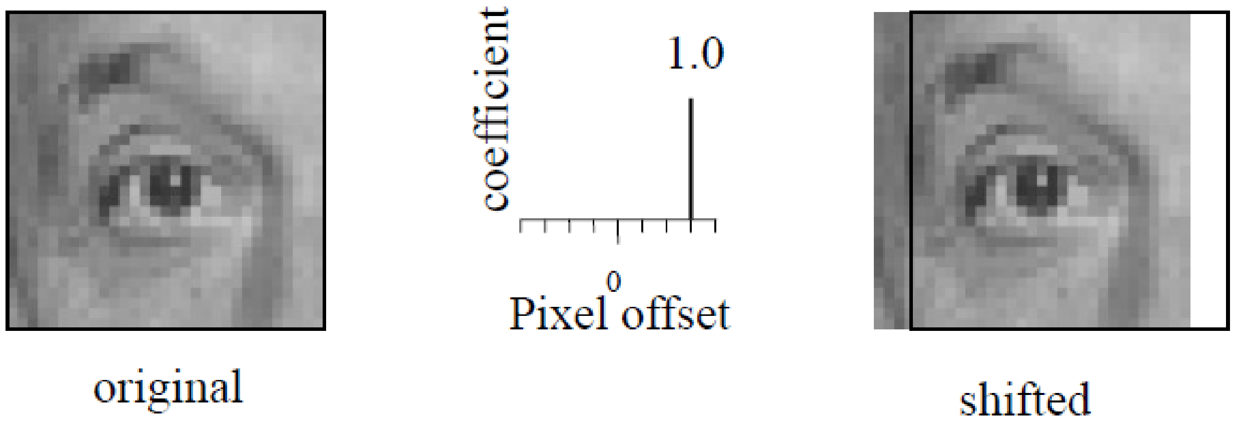
La convoluzione è commutativa e associativa

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

Inoltre,

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

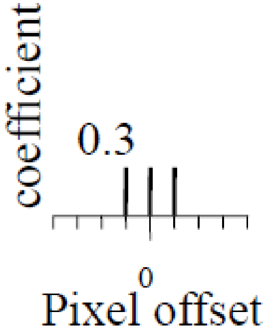
Shift



Source: Convolution - Fredo Durand - MIT EECS 6.815/6.865

Blurring

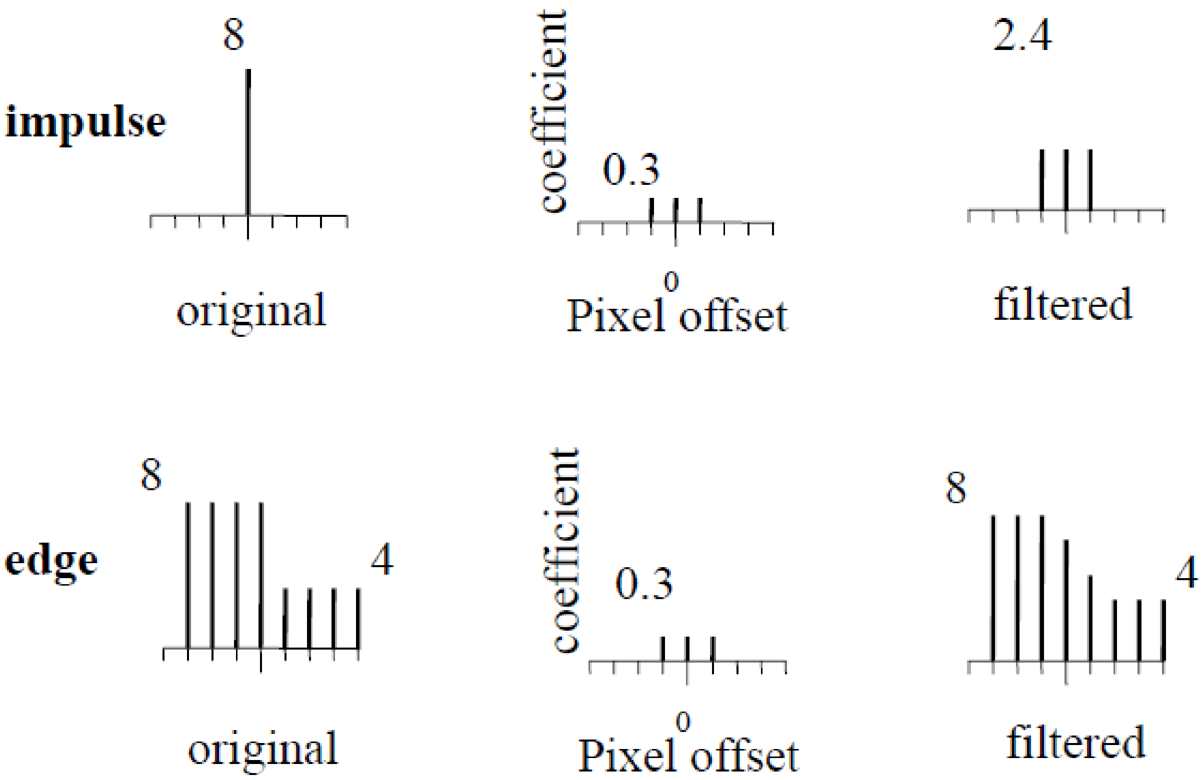
original



Blurred (filter applied in both dimensions).

Source: Convolution - Fredo Durand - MIT EECS 6.815/6.865

Blur examples

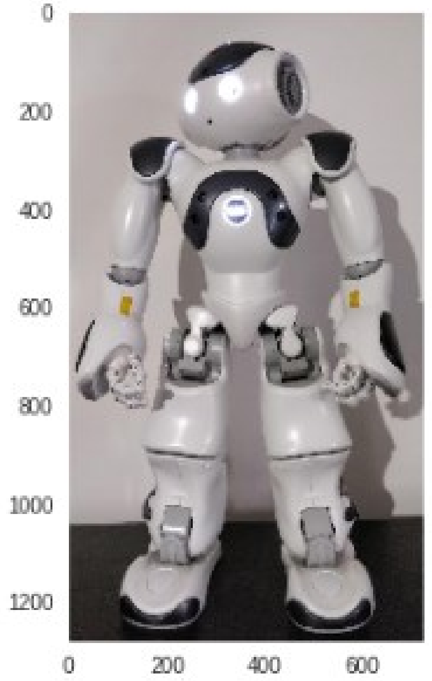


Source: Convolution - Fredo Durand - MIT EECS 6.815/6.865

How to read an image from url

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

import urllib.request

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

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

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

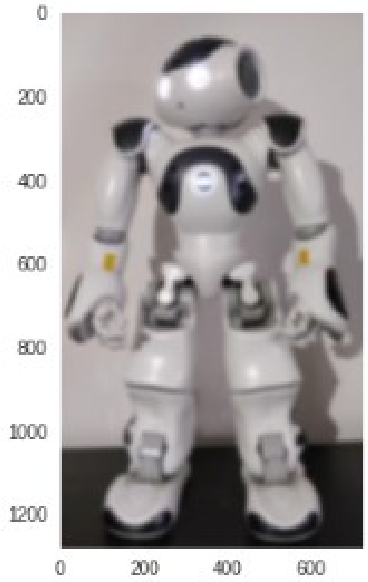
```
<matplotlib.image.AxesImage at 0x7fd4f5933048>
```



Gaussian blurring

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

from PIL import Image
from PIL import ImageFilter

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

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

blur_img = img.filter(ImageFilter.GaussianBlur(5))

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

```
<matplotlib.image.AxesImage at 0x7f0d50882828>
```



Gaussian blurring

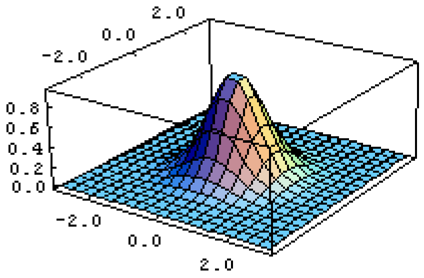
Un kernel Gaussian darà meno peso ai pixel distanti dal centro della finestra

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

$\frac{1}{16}$

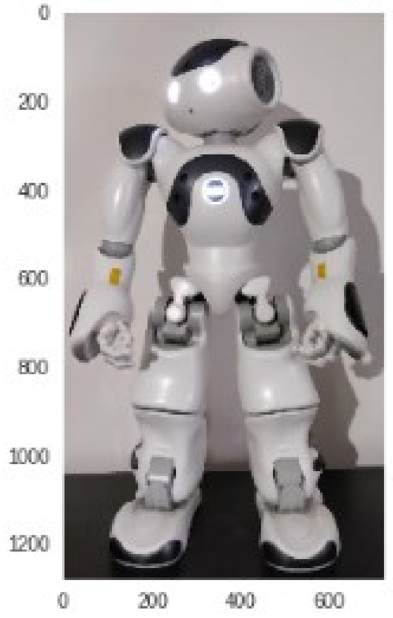
1	2	1
2	4	2
1	2	1



Median filter

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

from PIL import Image
from PIL import ImageFilter

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

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

blur_img = img.filter(ImageFilter.MedianFilter(7))

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

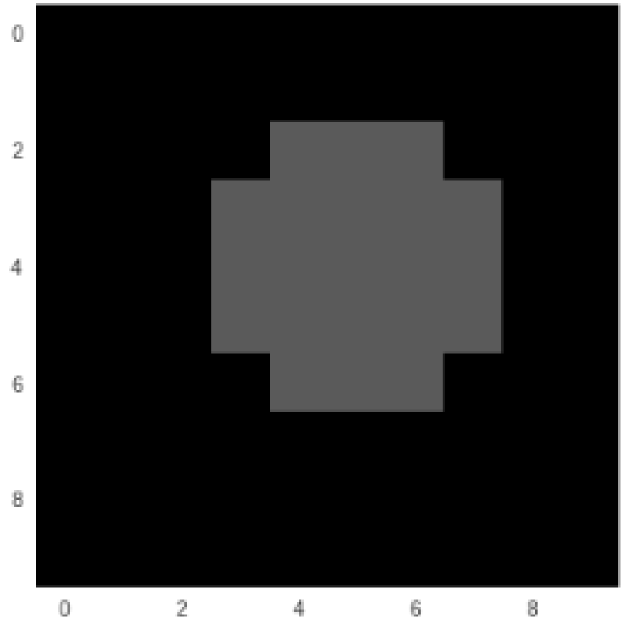
```
<matplotlib.image.AxesImage at 0x7f0d4efd2240>
```

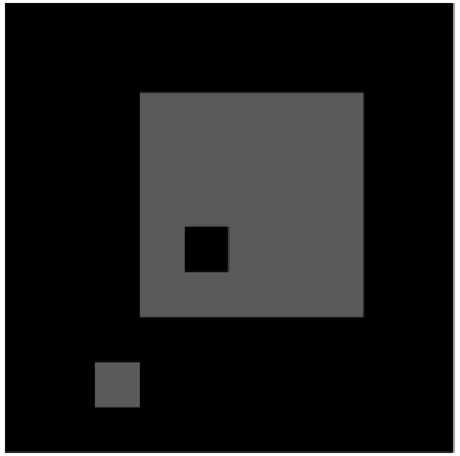


Median filter

```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.MedianFilter(3))

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

```
<matplotlib.image.AxesImage at 0x7f0d4eebaa20>
```

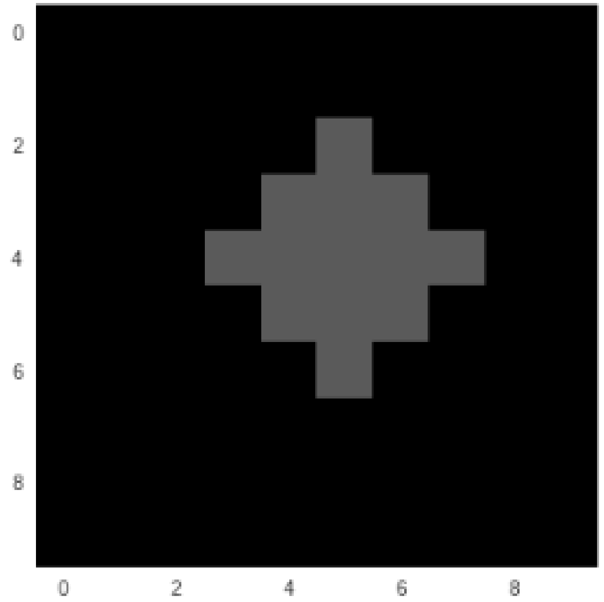


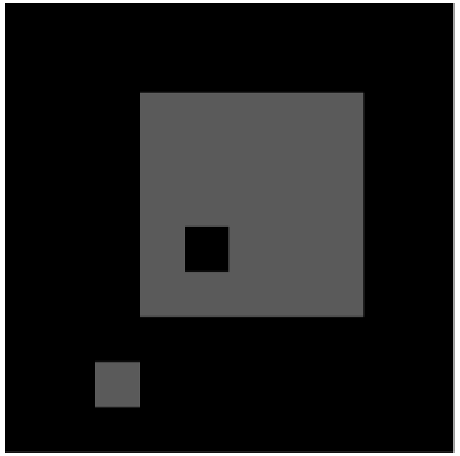


Median filter

```
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)

blur_img = img.filter(ImageFilter.MedianFilter(5))

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

```
<matplotlib.image.AxesImage at 0x7f0d4eeedbe0>
```

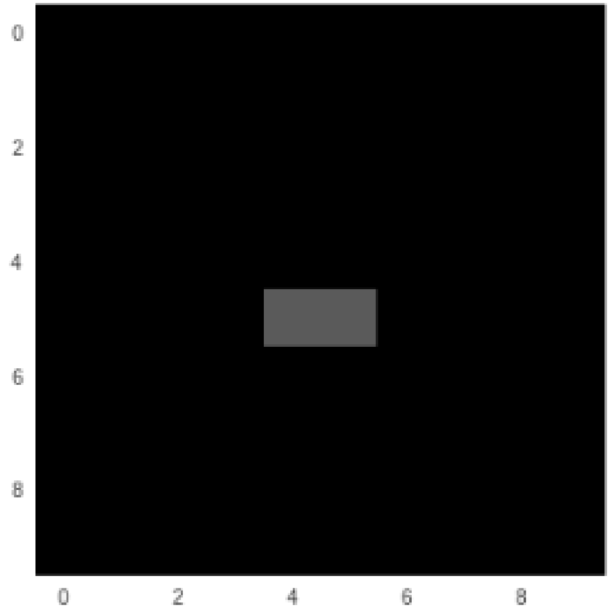


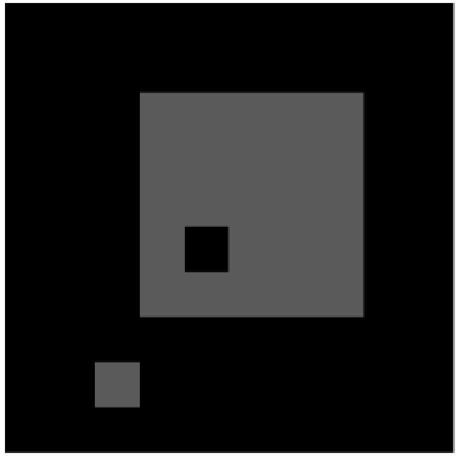


Median filter

```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.MedianFilter(7))

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

```
<matplotlib.image.AxesImage at 0x7f0d4ef9dd68>
```

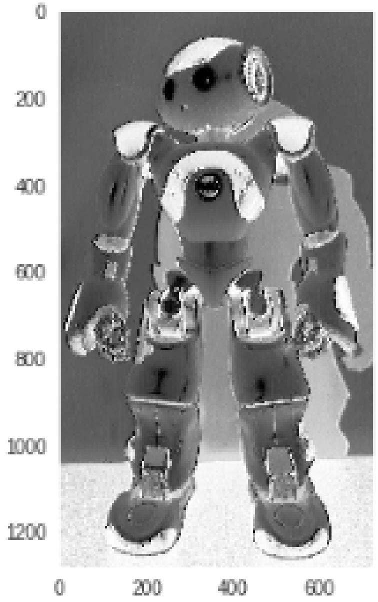




Custom filters

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

from PIL import Image
from PIL import ImageFilter

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

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

img = img.convert("L")
new_img = img.filter(ImageFilter.Kernel((3,3),[1,0,-1,5,0,-5,1,0,1]))

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

```
<matplotlib.image.AxesImage at 0x7f0d4ee8d5c0>
```



Gradiente

La derivata di una immagine è definita come la variazione nei valori di intensità dei pixel nell'immagine. Il tasso di variazione può essere calcolato come:

$$\boxed{\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}}$$

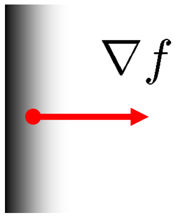
che diventa la differenza finita nel caso di immagini digitali

$$\frac{\partial f}{\partial x}[x, y] \approx F[x+1, y] - F[x, y]$$

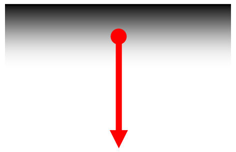
Gradiente

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

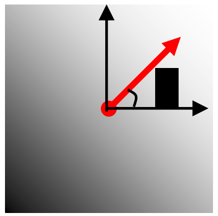
It points in the direction of most rapid change in intensity



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



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



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

Gradiente

The gradient direction is given by:

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

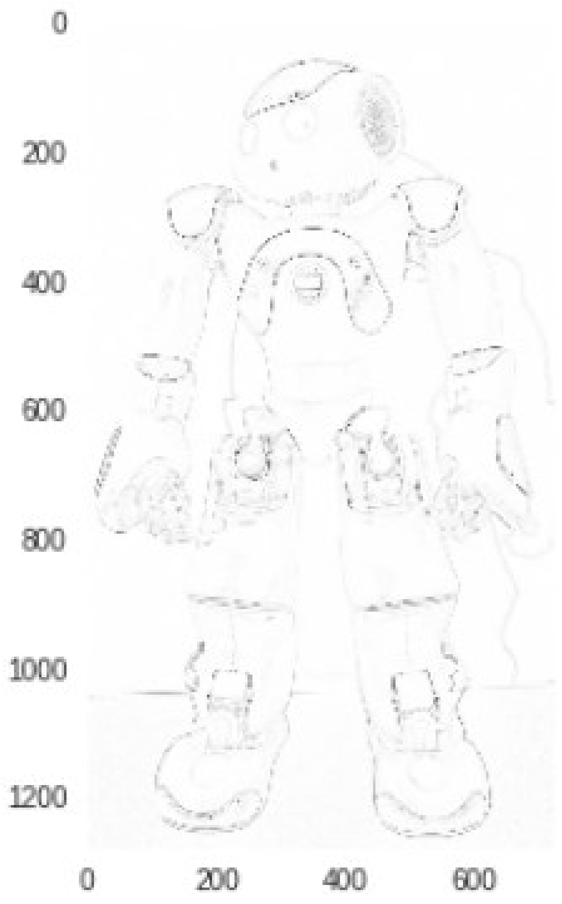
The *edge strength* is given by the gradient magnitude

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

Skimage (scikit-image)

Sobel edge detection

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

from PIL import Image

from skimage import io
from skimage import filters
from skimage import color

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

img = io.imread(urllib.request.urlopen(url))

img = color.rgb2gray(img)
edge = filters.sobel(img)

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



Canny edge detection

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

from PIL import Image

from skimage import io
from skimage import feature
from skimage import color

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

img = io.imread(urllib.request.urlopen(url))

img = color.rgb2gray(img)
edge = feature.canny(img,3)

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

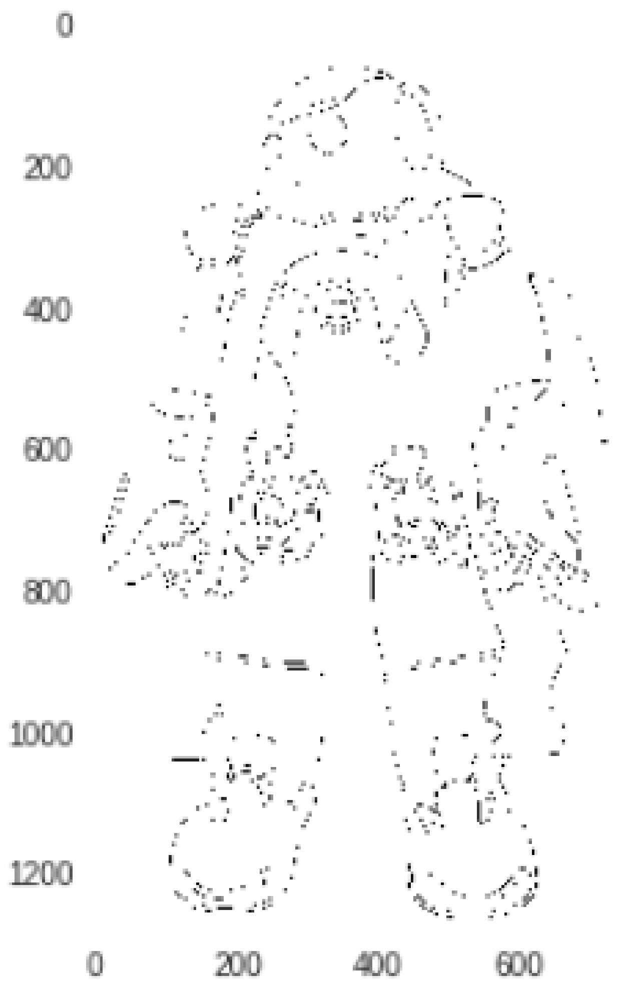
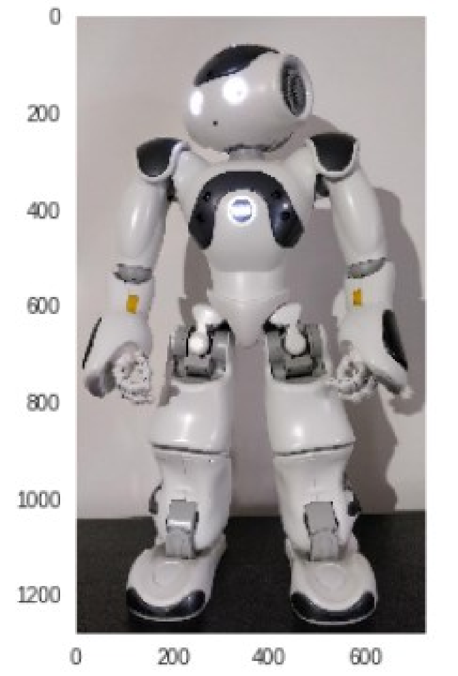
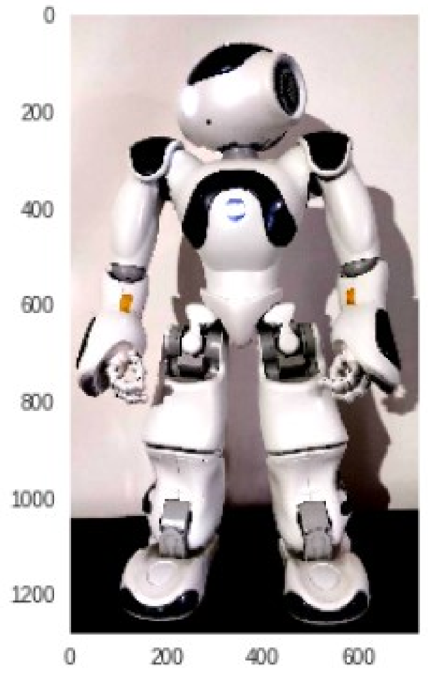


Image enhancement

```python
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

Applicare i filtri di Sobel e Canny sull'immagine https://dbloisi.github.io/corsi/images/nao-v6-spqr.jpg modificata tramite il contrast enhancement

Esercizio

Provare a modificare l’immagine
https://dbloisi.github.io/corsi/images/nao-v6-spqr.jpg
tramite cambio della **brightness**

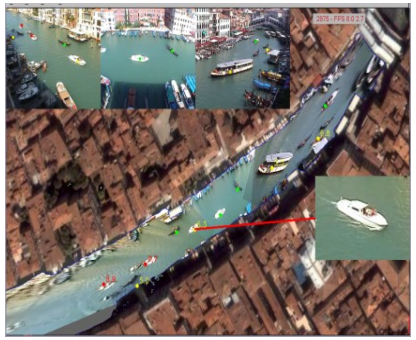
https://pillow.readthedocs.io/en/stable/reference/ImageEnhance.html
#PIL.ImageEnhance.PIL.ImageEnhance.Brightness

UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA

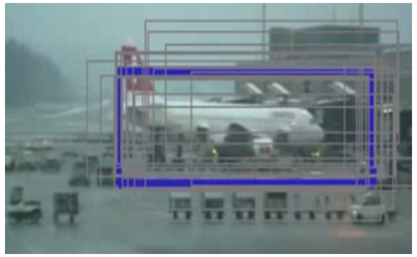


Corso di Sistemi Informativi
A.A. 2018/19

Docente
Domenico Daniele Bloisi



Filtri



Marzo 2019