Filtri

17634 | VISIONE ARTIFICIALE

Operazioni sui pixel

- In precedenza abbiamo esplorato le operazioni sui pixel, in cui il valore di ciascun pixel nell'immagine risultato dipende solo dal corrispondente pixel dell'immagine di partenza

$$\mathbf{I}'[y,x] = f(\mathbf{I}[y,x])$$

x

255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	0	0	0	255	255	255	255	255	255
255	255	255	255	255	255	0	0	0	0	0	0	0	255	255	255
255	255	255	255	255	0	0	0	168	168	0	168	255	255	255	255
255	255	255	255	0	0	168	0	0	168	168	168	168	255	255	255
255	255	255	255	0	0	168	168	168	168	0	0	168	255	255	255
255	255	255	255	255	255	168	168	168	168	168	168	255	255	255	255
255	255	255	255	0	0	0	0	96	0	255	255	255	255	255	255
255	255	168	168	0	0	0	0	96	0	0	168	168	168	255	255
255	255	168	168	255	0	0	0	255	96	0	0	168	168	255	255
255	255	255	255	96	96	96	96	96	96	255	255	0	255	255	255
255	255	255	0	96	96	96	96	96	96	96	0	0	255	255	255
255	255	255	0	0	96	255	255	255	96	96	0	0	255	255	255
255	255	255	255	0	0	0	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

y

$\mathbf{I}$

x

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0
0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0
0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0
0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0
0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0
0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0
0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

y

$\mathbf{I}'$

Operazioni locali

- Un operatore locale utilizza invece un insieme di pixel nelle vicinanze (intorno) del pixel dell'immagine di partenza per determinare il valore del pixel nell'immagine risultato

$$\mathbf{I}'[y,x] = f\left(w_{xy}(\mathbf{I})\right)$$

x

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0
0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0
0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0
0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0
0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0
0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0
0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

y

$\mathbf{I}$

$w_{xy}(\mathbf{I})$

x

0	0	0	0	0	0	255	255	255	255	255	0	0	0	0	0
0	0	0	0	0	255	255	0	0	0	255	255	255	255	0	0
0	0	0	0	255	255	0	0	0	0	0	0	0	0	0	0
0	0	0	255	255	0	0	0	255	255	0	255	255	255	0	0
0	0	0	255	0	0	255	0	0	255	168	249	0	87	0	0
0	0	0	255	0	0	249	168	168	168	0	0	0	87	0	0
0	0	0	255	255	255	255	240	255	153	249	0	255	87	0	0
0	87	87	255	0	0	0	0	48	0	255	255	255	87	87	0
0	87	0	0	0	0	0	0	255	0	0	0	0	0	87	0
0	87	0	153	255	0	0	0	255	0	0	0	0	0	87	0
0	87	255	246	120	0	192	0	96	0	255	255	0	255	87	0
0	0	255	0	0	0	0	0	0	0	0	0	0	255	0	0
0	0	255	0	0	192	255	255	255	0	0	0	0	255	0	0
0	0	255	255	0	0	0	0	159	159	255	255	255	255	0	0
0	0	0	255	255	255	255	255	0	0	0	0	0	0	0	0

y

$\mathbf{I}'$

Filtri lineari

- Sono le operazioni locali più comuni: il valore di ciascun pixel è calcolato come somma pesata dei valori dei pixel nell'intorno considerato: $\mathbf{I}'[y,x] = \sum_{i,j} w_{xy}[i,j] \cdot \mathbf{F}[i,j]$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0
0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0
0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0
0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0
0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0
0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0
0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	0	0	0	0	0	26	153	204	204	128	76	76	51	0	
	0	0	0	0	26	178	255	238	179	170	187	145	51	0	
	0	0	0	26	178	238	221	205	154	171	154	69	17	0	
	0	0	0	128	255	188	154	171	137	121	129				

$$\mathbf{I}' = \mathbf{I} \otimes \mathbf{F}$$

0.0	0.1	0.1
0.2	0.3	0.0
0.2	0.1	0.0

$\mathbf{F}$

$\bullet$

255	87	0
87	87	87
255	255	87

$w_{xy}(\mathbf{I})$

$=$ 0.0*255 + 0.1*87 + 0.1*0
+ 0.2*87 + 0.3*87 + 0.0*87
+ 0.2*255 + 0.1*255 + 0.0*87

$= 129$

$\mathbf{I}'[y,x]$

Filtri lineari

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0
0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0
0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0
0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0
0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0
0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0
0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0.0	0.1	0.1
0.2	0.3	0.0
0.2	0.1	0.0

•

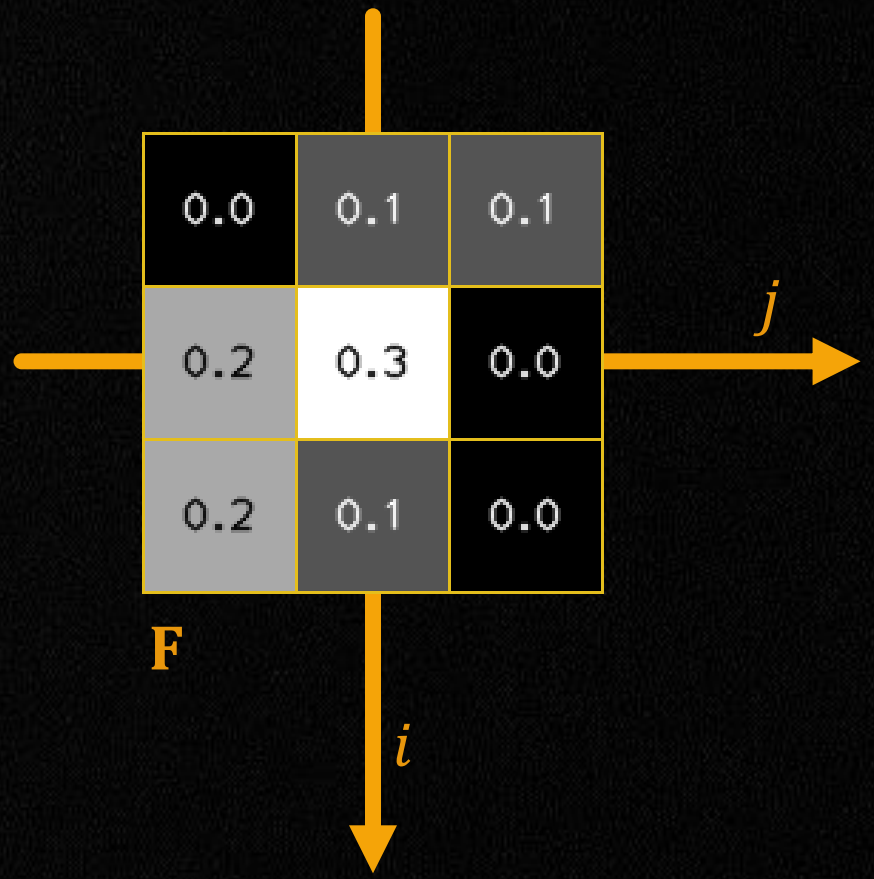
0	0	0
0	0	0
0	0	0

$$= 0.0*0 + 0.1*0 + 0.1*0 + 0.2*0 + 0.3*0 + 0.0*0 + 0.2*0 + 0.1*0 + 0.0*0 = 0$$

Il filtro

- $\mathbf{F}$ è chiamato *filtro*, *maschera*, o *kernel*: di norma è quadrato, con dimensioni $m \times m$, m dispari
  - Gli elementi di $\mathbf{F}$ sono chiamati *coefficienti del filtro* o *pesi*
- Di solito si fa riferimento ai coefficienti del filtro considerando due assi cartesiani (con l'asse verticale diretto vero il basso) con origine in corrispondenza del centro di $\mathbf{F}$.
  - La formula per il calcolo del valore di un pixel nell'immagine destinazione può essere quindi scritta come:

$$\mathbf{I}'[y,x] = \sum_{i=-d}^{d} \sum_{j=-d}^{d} \mathbf{F}[i,j] \cdot \mathbf{I}[y+i, x+j], con\ d = \left\lfloor \frac{m}{2} \right\rfloor$$



Gestione dei bordi

- Come comportarsi vicino ai bordi, dove il filtro non può essere completamente sovrapposto all'immagine?
- Analogamente a quanto visto per le trasformazioni geometriche, ci sono varie possibilità:
  - Usare un valore di "background" costante
  - Copiare il valore del pixel più vicino
  - Riflessione dei pixel dell'immagine

?	?	?	?	?									
?	?	?	?	?									
?	?	0	0	0	0	0	255	255	255	0	0	0	0
?	?	0	0	0	0	255	255	255	255	255	255	255	0
?	?	0	0	0	255	255	255	87	87	255	87	0	0
		0	0	255	255	87	255	255	87	87	87	87	0
		0	0	255	255	87	87	87	87	255	255	87	0
		0	0	0	0	87	87	87	87	87	87	0	0
		0	0	255	255	255	255	159	255	0	0	0	0
		87	87	255	255	255	255	159	255	255	87	87	87
		87	87	0	255	255	255	0	159	255	255	87	87
		0	0	159	159	159	159	159	159	0	0	255	0
		0	255	159	159	159	159	159	159	159	255	255	0
		0	255	255	159	0	0	0	159	159	255	255	0
		0	0	255	255	255	0	0	0	0	0	0	0

Gestione dei bordi in OpenCV

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0	0
0	0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0	0
0	0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0	0
0	0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0	0
0	0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0	0
0	0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0	0
0	0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0	0
0	0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0	0
0	0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0	0
0	0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0	0
0	0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0	0
0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

cv.BORDER_CONSTANT

0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0	0
0	0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0	0
0	0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0	0
0	0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0	0
0	0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0	0
0	0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0	0
87	87	87	87	87	255	255	255	255	159	255	255	87	87	87	87	87	87
87	87	87	87	87	0	255	255	255	0	159	255	255	87	87	87	87	87
0	0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0	0
0	0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0	0
0	0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0	0
0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0	0

cv.BORDER_REPLICATE

0	0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0	87
0	0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	255	255
0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255	255	255	255	255	0	0	255	255
0	0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0	87
255	0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	87	87
255	0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	87	255
0	0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0	87
255	0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0	0
255	87	87	87	87	255	255	255	255	159	255	255	87	87	87	87	87	87
0	87	87	87	87	0	255	255	255	0	159	255	255	87	87	87	87	255
159	0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	255	0
159	255	0	0	255	159	159	159	159	159	159	159	255	255	0	0	255	255
255	255	0	0	255	255	159	0	0	0	159	159	255	255	0	0	255	255
255	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0	0
255	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0	0
255	255	0	0	255	255	159	0	0	0	159	159	255	255	0	0	255	255
159	255	0	0	255	159	159	159	159	159	159	159	255	255	0	0	255	255

cv.BORDER_REFLECT

Visione Artificiale - Raffaele Cappelli - raffaele.cappelli@unibo.it

Tipi di dati e normalizzazione dei valori del filtro

- Tipi di valori dei pixel dell'immagine di partenza:
  - Il caso più comune è un *byte* per pixel [0,255]
  - Altri tipi comuni sono *interi con o senza segno* (a 16 o 32 bit) e numeri *floating point* (a 32 o 64 bit): di solito questo avviene quanto l'immagine è a sua volta il risultato di precedenti applicazioni di filtri
- Tipi dei valori dei pixel dell'immagine risultato:
  - Se ci sono coefficienti negativi, i pixel del risultato dovranno essere di tipo *intero con segno* o *floating point*
- Normalizzazione del filtro:
  - Se i coefficienti del filtro sono tutti ≥ 0 e si vuole che i pixel dell'immagine risultato abbiano lo stesso range di valori dell'immagine di partenza (ad esempio [0,255]), si normalizza il filtro dividendo per la somma dei coefficienti

0	2	1
2	5	0
3	1	0

$$\frac{1}{14} \cdot$$

0	2	1
2	5	0
3	1	0

0.0	0.14	0.07
0.14	0.36	0.0
0.21	0.07	0.0

Esempi di filtri

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0
0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0
0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0
0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0
0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0
0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0
0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I

$\otimes$

0	0	0
0	1	0
0	0	0

F

$=$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	255	255	255	0	0	0
0	0	0	0	0	[illegible]	[illegible]	[illegible]	255	87	[illegible]	[illegible]	87	0	0	0
0	0	0	0	255	255	87	255	255	87	[illegible]	[illegible]	87	0	0	0
0	0	0	0	255	255	87	87	87	87	[illegible]	[illegible]	87	0	0	0
0	0	0	0	0	0	87	87	87	87	[illegible]	[illegible]	0	0	0	0
0	0	0	0	255	255	255	255	159	[illegible]	[illegible]	[illegible]	0	0	0	0
0	0	87	87	255	255	255	[illegible]	[illegible]	255	87	87	87	0	0	0
0	0	87	87	0	255	255	[illegible]	[illegible]	159	255	255	87	87	0	0
0	0	0	0	159	159	159	[illegible]	[illegible]	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	[illegible]	[illegible]	159	159	255	255	0	0	0
0	0	0	0	255	255	255	[illegible]	[illegible]	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I′

Esempi di filtri

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0
0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0
0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0
0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0
0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0
0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0
0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I

$\otimes$

0	1	0
0	0	0
0	0	0

F

$=$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	0	0
0	0	0	0	0	[illegible]	[illegible]	255	255	[illegible]	255	255	0	0	0	0
0	0	0	0	0	255	255	255	87	87	[illegible]	0	0	0	0	0
0	0	0	0	255	255	87	255	255	87	[illegible]	87	0	0	0	0
0	0	0	0	255	255	87	87	87	87	[illegible]	87	0	0	0	0
0	0	0	0	0	0	87	87	87	[illegible]	87	0	0	0	0	0
0	0	0	0	255	255	255	[illegible]	[illegible]	0	0	0	0	0	0	0
0	0	87	87	255	255	255	[illegible]	255	255	87	87	87	0	0	0
0	0	87	87	0	255	255	[illegible]	159	255	255	87	87	0	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	[illegible]	159	159	255	255	0	0	0	0
0	0	0	255	255	159	0	[illegible]	159	159	255	255	0	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0

I′

Visione Artificiale - Raffaele Cappelli - raffaele.cappelli@unibo.it

Esempi di filtri

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0
0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0
0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0
0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0
0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0
0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0
0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I

$$\otimes \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

F

$$=$$

0	0	0	0	0	0	28	57	85	57	28	0	0	0	0	0
0	0	0	0	0	28	85	142	170	142	113	85	57	28	0	0
0	0	0	0	28	85	?	?	?	?	161	123	66	28	0	0
0	0	0	28	85	?	?	218	199	?	?	152	86	38	0	0
0	0	0	57	142	189	199	162	143	143	?	?	67	19	0	0
0	0	0	57	113	142	133	124	124	124	?	?	67	19	0	0
0	0	0	57	113	161	152	132	132	123	?	?	48	10	0	0
0	10	19	76	123	180	189	178	178	?	?	67	39	19	10	0
0	19	39	95	161	227	255	?	?	?	169	114	77	39	19	0
0	19	39	85	140	195	223	?	?	156	158	142	105	67	19	0
0	10	48	83	137	163	191	?	?	134	156	169	133	76	10	0
0	0	57	120	173	152	124	106	124	124	145	177	142	85	0	0
0	0	57	131	195	184	127	?	?	88	127	149	113	57	0	0
0	0	28	85	131	131	74	?	?	35	64	74	57	28	0	0
0	0	0	28	57	85	57	28	0	0	0	0	0	0	0	0

I′

Visualizzare immagini con pixel di tipo int o float

- Di solito i valori dei pixel sono normalizzati in [0,255] prima della visualizzazione
- Nelle immagini visualizzate in questo corso, i valori sono normalizzati come segue
- Immagini senza valori negativi: $\mathbf{I}'[y,x] = \frac{255}{\max(\mathbf{I})} \cdot \mathbf{I}[y,x]$
  - Il valore 0, anche se non presente nell'immagine, corrisponde al nero, il valore massimo presente nell'immagine corrisponde al bianco.
- Immagini che contengono valori negativi: $\mathbf{I}'[y,x] = 128 + \frac{127}{\max(\mathbf{abs}(\mathbf{I}))} \cdot \mathbf{I}[y,x]$
  - Valore 0 → livello di grigio 128, valori negativi → livelli di grigio più scuro (fino al nero), valori positivi → livelli di grigio più chiaro (fino al bianco).

0	10	21	31	42
53	63	74	85	95
106	116	127	138	148
159	170	180	191	201
212	223	233	244	255

0	208	416	625	833
1041	1250	1458	1666	1875
2083	2291	2500	2708	2916
3125	3333	3541	3750	3958
4166	4375	4583	4791	5000

3000	3083	3166	3250	3333
3416	3500	3583	3666	3750
3833	3916	4000	4083	4166
4250	4333	4416	4500	4583
4666	4750	4833	4916	5000

−600	−562	−525	−487	−450
−412	−375	−337	−300	−262
−225	−187	−150	−112	−75
−37	0	37	75	112
150	187	225	262	300

0.0	0.04	0.08	0.12	0.17
0.21	0.25	0.29	0.33	0.38
0.42	0.46	0.5	0.54	0.58
0.62	0.67	0.71	0.75	0.79
0.83	0.88	0.92	0.96	1.0

−1.41	−1.06	−0.71	−0.35	0.0
0.35	0.71	1.06	1.41	1.77
2.12	2.47	2.83	3.18	3.54
3.89	4.24	4.6	4.95	5.3
5.66	6.01	6.36	6.72	7.07

Esempi di filtri

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	255	255	255	255	0	0	0	0	0
0	0	0	0	255	255	255	255	255	255	255	0	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	0	255	255	255	255	255	255	255	0	0	0	0
0	0	0	0	0	255	255	255	255	255	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I

$\otimes$

−1.0	−1.0	−1.0
−1.0	8.0	−1.0
−1.0	−1.0	−1.0

F

$=$

0	0	−255	−510	−765	−765	−765	−765	−765	−765	−765	−510	−255	0	0
0	0	−510	1275	765	765	765	765	765	765	765	1275	−510	0	0
0	0	−765	765	0	0	0	0	0	0	0	765	−765	0	0
0	0	−765	765	0	0	0	0	0	0	0	765	−765	0	0
0	0	−510	1275	765	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	765	1275	−510	0	0
0	0	−255	−510	−765	−765	−765	−765	−765	[illegible]	[illegible]	−510	−255	0	0
0	0	0	0	−255	−510	−765	−765	−765	[illegible]	[illegible]	0	0	0	0
0	0	0	−255	−765	1020	765	765	765	[illegible]	[illegible]	−255	0	0	0
0	0	−255	−765	765	255	0	0	[illegible]	[illegible]	765	−765	−255	0	0
0	−255	−765	765	255	0	0	[illegible]	[illegible]	0	255	765	−765	−255	0
0	−510	1020	255	0	0	0	[illegible]	0	0	0	255	1020	−510	0
0	−765	765	0	0	0	0	[illegible]	0	0	0	0	765	−765	0
0	−765	765	0	0	0	0	0	0	0	0	0	765	−765	0
0	−765	765	0	0	0	0	[illegible]	0	0	0	0	765	−765	0
0	−510	1020	255	0	0	0	[illegible]	0	0	0	255	1020	−510	0
0	−255	−765	765	255	0	0	0	0	0	255	765	−765	−255	0
0	0	−255	−765	765	255	0	0	0	255	765	−765	−255	0	0
0	0	0	−255	−765	1020	765	765	765	1020	−765	−255	0	0	0
0	0	0	0	−255	−510	−765	−765	−765	−510	−255	0	0	0	0

I′

Visione Artificiale - Raffaele Cappelli - raffaele.cappelli@unibo.it

Esempi di filtri

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	255	255	255	255	0	0	0	0	0
0	0	0	0	255	255	255	255	255	255	255	0	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	0	255	255	255	255	255	255	255	0	0	0	0
0	0	0	0	0	255	255	255	255	255	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I

$\otimes$

0.0	0.0	0.0
0.0	−1.0	0.0
0.0	1.0	0.0

F

$=$

0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	−255	−255	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	−255	−255	0	0	0
0	0	0	0	0	0	0	0	0	[illegible]	[illegible]	0	0	0	0
0	0	0	0	0	255	255	255	255	[illegible]	[illegible]	0	0	0	0
0	0	0	0	255	0	0	0	0	[illegible]	[illegible]	0	0	0	0
0	0	0	255	0	0	0	0	[illegible]	[illegible]	0	255	0	0	0
0	0	255	0	0	0	0	[illegible]	[illegible]	0	0	0	255	0	0
0	0	0	0	0	0	0	[illegible]	0	0	0	0	0	0	0
0	0	0	0	0	0	0	[illegible]	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	[illegible]	0	0	0	0	0	0	0
0	0	−255	0	0	0	0	[illegible]	0	0	0	0	−255	0	0
0	0	0	−255	0	0	0	0	0	0	0	−255	0	0	0
0	0	0	0	−255	0	0	0	0	0	−255	0	0	0	0
0	0	0	0	0	−255	−255	−255	−255	−255	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I′

Esempi di filtri

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	255	255	255	255	0	0	0	0	0
0	0	0	0	255	255	255	255	255	255	255	0	0	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	255	255	255	255	255	255	255	255	255	255	255	0	0
0	0	0	255	255	255	255	255	255	255	255	255	0	0	0
0	0	0	0	255	255	255	255	255	255	255	0	0	0	0
0	0	0	0	0	255	255	255	255	255	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I

$\otimes$

0.0	0.0	0.0
1.0	−1.0	0.0
0.0	0.0	0.0

F

$=$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	−255	0	0	0	0	0	0	0	0	255	0	0
0	0	0	−255	0	0	0	0	0	0	0	0	255	0	0
0	0	0	−255	0	0	0	0	0	0	0	0	255	0	0
0	0	0	−255	0						0	0	255	0	0
0	0	0	0	0	0	0	0	0			0	0	0	0
0	0	0	0	0	0	0	0	0			0	0	0	0
0	0	0	0	0	−255	0	0	0			0	0	0	0
0	0	0	0	−255	0	0	0			0	255	0	0	0
0	0	0	−255	0	0	0			0	0	0	255	0	0
0	0	−255	0	0	0	0		0	0	0	0	0	255	0
0	0	−255	0	0	0	0		0	0	0	0	0	255	0
0	0	−255	0	0	0	0	0	0	0	0	0	0	255	0
0	0	−255	0	0	0	0		0	0	0	0	0	255	0
0	0	−255	0	0	0	0		0	0	0	0	0	255	0
0	0	0	−255	0	0	0	0	0	0	0	0	255	0	0
0	0	0	0	−255	0	0	0	0	0	0	255	0	0	0
0	0	0	0	0	−255	0	0	0	0	255	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I′

Filtri in Python e OpenCV


```python
# Applicazione di un filtro lineare a un'immagine con un solo canale
# Implementazione esemplificativa in Python
# Il risultato è memorizzato su un'immagine di interi con segno a 16 bit

def filter2D_py(img, f):
    h, w = img.shape
    kh, kw = f.shape
    dy, dx = kh//2, kw//2
    imgb = cv.copyMakeBorder(img, dy, dy, dx, dx, cv.BORDER_DEFAULT)
    res = np.zeros(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            s = 0
            for i in range(kh):
                for j in range(kw):
                    # N.B. sarebbe imgb[(dy+y)+(i-dy), (dx+x)+(j-dx)]
                    s += imgb[y+i, x+j]*f[i, j]
            res[y, x] = s
    return res
```

Filtri in Python e OpenCV


```python
# Applicazione di un filtro lineare a un'immagine con un solo canale
# Implementazione esemplificativa in Python con calcolo del valore
# di ciascun pixel mediante operazioni NumPy
# Il risultato è memorizzato su un'immagine di interi con segno a 16 bit

def filter2D_np(img, f):
    h, w = img.shape
    kh, kw = f.shape
    dy, dx = kh//2, kw//2
    imgb = cv.copyMakeBorder(img, dy, dy, dx, dx, cv.BORDER_DEFAULT)
    res = np.zeros(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            res[y, x] = (imgb[y:y+kh, x:x+kw]*f).sum()
    return res
```

Filtri in Python e OpenCV


```python
# Applicazione di un filtro lineare a un'immagine con OpenCV
# Il risultato è memorizzato su un'immagine di interi con segno a 16 bit
def filter2D_cv(img, f):
    return cv.filter2D(img, cv.CV_16S, f)


# Misura del tempo di esecuzione delle tre implementazioni
# con un filtro 9x9 con tutti i coefficienti pari a uno.
img = cv.cvtColor(cv.imread('esempi/bolts.png'), cv.COLOR_BGR2GRAY)
f = np.ones((9,9))

implementazioni = [filter2D_py, filter2D_np, filter2D_cv]
for filter2D in implementazioni:
    print(filter2D.__name__, end=': ')
    %timeit filter2D(img, f)
```

Metodo	Tempo
filter2D_py	52.1 s
filter2D_np	1.48 s
filter2D_cv	2.62 ms

Filtri separabili

- $\mathbf{I}'[y,x] = \sum_{i=-d}^{d} \sum_{j=-d}^{d} \mathbf{F}[i,j] \cdot \mathbf{I}[y+i, x+j]\,, con\ d = \left\lfloor \frac{m}{2} \right\rfloor$
  - Complessità computazionale: m^2 moltiplicazioni e somme per ciascun pixel del risultato
- Se la matrice $\mathbf{F}$ può essere ottenuta come *prodotto di un vettore colonna per un vettore riga* ($\mathbf{F} = \mathbf{F}_y \cdot \mathbf{F}_x^\top$), il filtro è *separabile* e si ha: $\mathbf{I}'[y,x] = \sum_i \sum_j \mathbf{F}_y[i] \cdot \mathbf{F}_x[j] \cdot \mathbf{I}[y+i, x+j] = \sum_i \mathbf{F}_y[i] \cdot \sum_j \mathbf{F}_x[j] \cdot \mathbf{I}[y+i, x+j]$ e quindi:

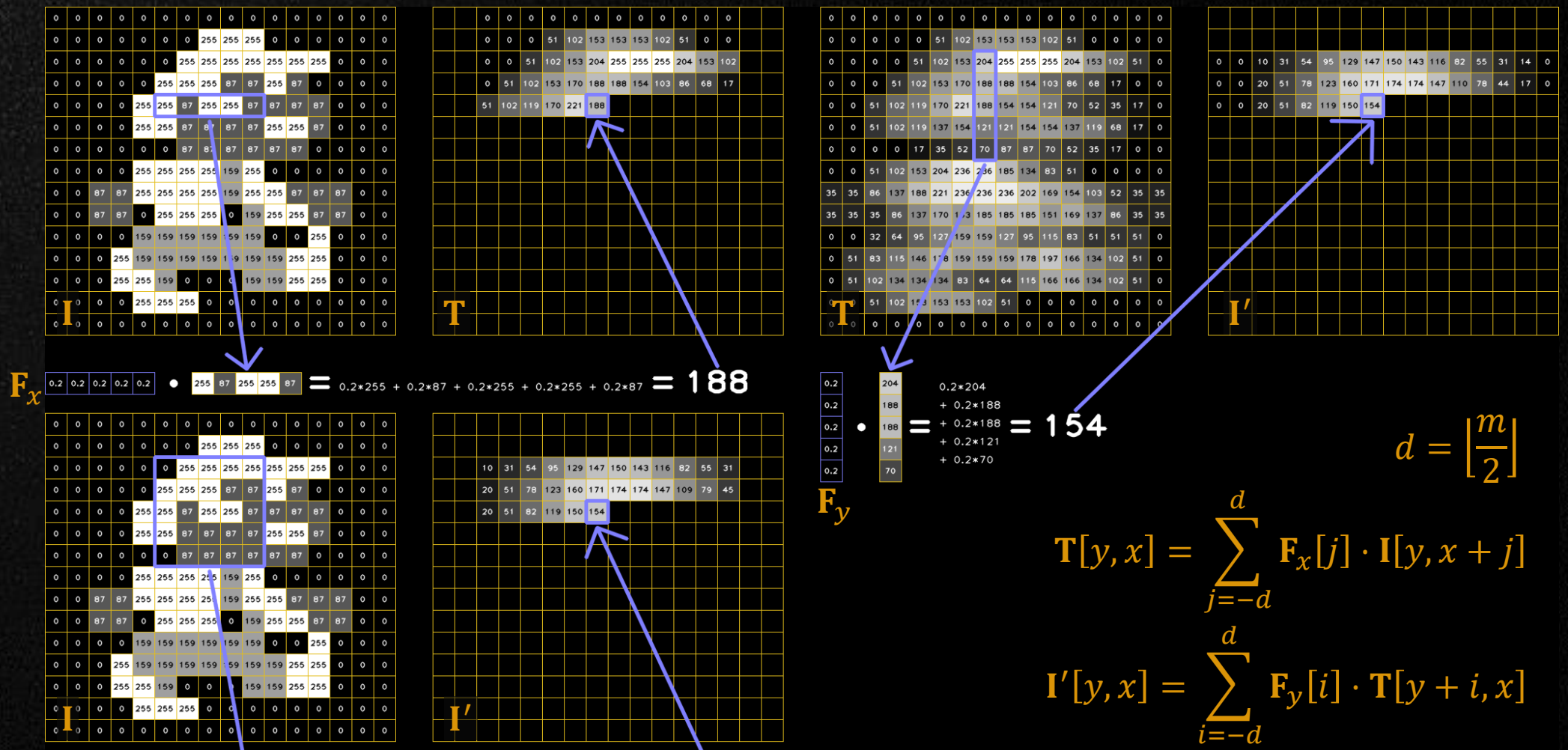
$$\begin{bmatrix} -3 & -6 & 3 \\ 1 & 2 & -1 \\ -2 & -4 & 2 \end{bmatrix} = \begin{bmatrix} -3 \\ 1 \\ -2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & -1 \end{bmatrix}$$

$$\mathbf{F} \qquad \mathbf{F}_y \qquad \mathbf{F}_x^\top$$

$$\mathbf{I}'[y,x] = \sum_{i=-d}^{d} \mathbf{F}_y[i] \cdot \mathbf{T}[y+i, x]\,, \textbf{con}\ \ \mathbf{T}[y,x] = \sum_{j=-d}^{d} \mathbf{F}_x[j] \cdot \mathbf{I}[y, x+j]$$

- Si può quindi applicare il filtro $\mathbf{F}_x$ (dimensioni $1 \times m$) a tutta l'immagine, ottenendo un'immagine intermedia $\mathbf{T}$, quindi applicare $\mathbf{F}_y$ (dimensioni $m \times 1$) a $\mathbf{T}$ per ottenere il risultato $\mathbf{I}'$.
  - Complessità computazionale: $2m$ moltiplicazioni e somme per ciascun pixel del risultato

Filtri separabili: esempio



$$d = \left\lfloor \frac{m}{2} \right\rfloor$$

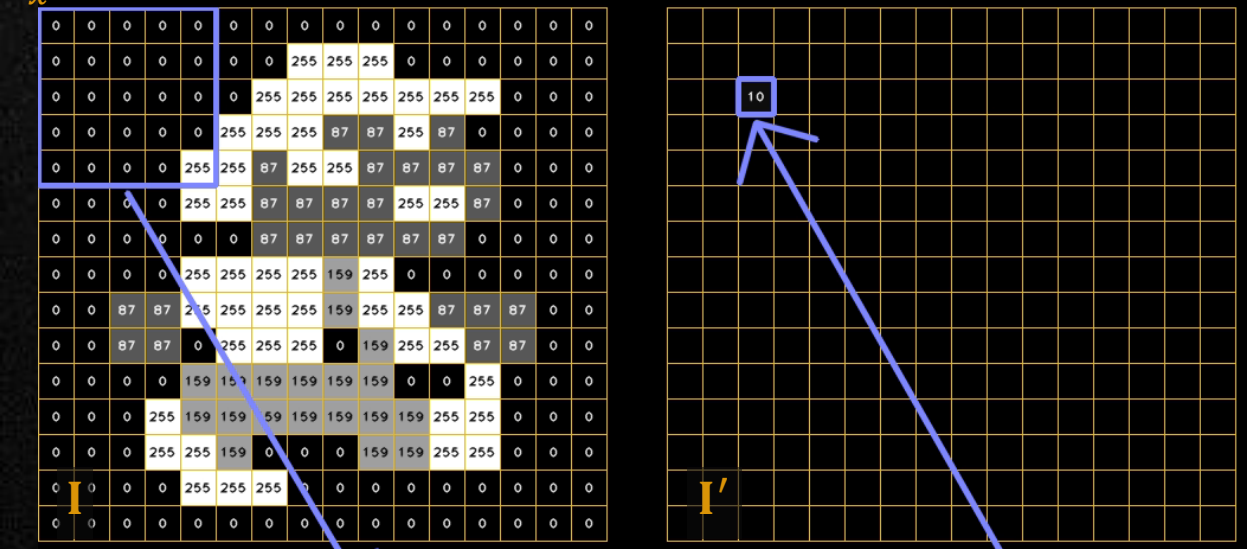
$$\mathbf{T}[y,x] = \sum_{j=-d}^{d} \mathbf{F}_x[j] \cdot \mathbf{I}[y, x+j]$$

$$\mathbf{I}'[y,x] = \sum_{i=-d}^{d} \mathbf{F}_y[i] \cdot \mathbf{T}[y+i, x]$$

$$\mathbf{I}'[y,x] = \sum_{i=-d}^{d} \sum_{j=-d}^{d} \mathbf{F}[i,j] \cdot \mathbf{I}[y+i, x+j]$$

$\mathbf{F}_x$ $\begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix} \bullet \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} = 0.2*0 + 0.2*0 + 0.2*0 + 0.2*0 + 0.2*0 = 0$

$\mathbf{F}_y$ $\begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix} \bullet \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 51 \end{bmatrix} = \begin{matrix} 0.2*0 \\ + 0.2*0 \\ + 0.2*0 \\ + 0.2*0 \\ + 0.2*51 \end{matrix} = 10$



$\mathbf{F}$ $\begin{bmatrix} 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \end{bmatrix} \bullet \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 255 \end{bmatrix} = \begin{matrix} 0.0*0 + 0.0*0 + 0.0*0 + 0.0*0 + 0.0*0 \\ + 0.0*0 + 0.0*0 + 0.0*0 + 0.0*0 + 0.0*0 \\ + 0.0*0 + 0.0*0 + 0.0*0 + 0.0*0 + 0.0*0 \\ + 0.0*0 + 0.0*0 + 0.0*0 + 0.0*0 + 0.0*0 \\ + 0.0*0 + 0.0*0 + 0.0*0 + 0.0*0 + 0.0*255 \end{matrix} = 10$

$$d = \left\lfloor \frac{m}{2} \right\rfloor$$

$$\mathbf{T}[y,x] = \sum_{j=-d}^{d} \mathbf{F}_x[j] \cdot \mathbf{I}[y, x+j]$$

$$\mathbf{I}'[y,x] = \sum_{i=-d}^{d} \mathbf{F}_y[i] \cdot \mathbf{T}[y+i, x]$$

$$\mathbf{I}'[y,x] = \sum_{i=-d}^{d} \sum_{j=-d}^{d} \mathbf{F}[i,j] \cdot \mathbf{I}[y+i, x+j]$$

Filtri separabili in Python e OpenCV


```python
# Applicazione di un filtro lineare separabile a un'immagine con un solo canale
# Implementazione esemplificativa in Python
# Il risultato è memorizzato su un'immagine di interi con segno a 16 bit

def sepFilter2D_py(img, fX, fY):
    fX, fY = fX.ravel(), fY.ravel()
    h, w = img.shape
    kw, kh = fX.size, fY.size
    dy, dx = kh//2, kw//2
    imgb = cv.copyMakeBorder(img, 0, 0, dx, dx, cv.BORDER_DEFAULT)
    t = np.empty(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            t[y, x] = sum(imgb[y, x+j]*fX[j] for j in range(kw))
    tb = cv.copyMakeBorder(t, dy, dy, 0, 0, cv.BORDER_DEFAULT)
    res = np.empty(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            res[y, x] = sum(tb[y+i, x]*fY[i] for i in range(kh))
    return res
```

Filtri separabili in Python e OpenCV


```python
# Applicazione di un filtro lineare separabile a un'immagine con un solo canale
# Implementazione esemplificativa in Python con calcolo del valore
# di ciascun pixel mediante operazioni NumPy
# Il risultato è memorizzato su un'immagine di interi con segno a 16 bit
def sepFilter2D_np(img, fX, fY):
    fX, fY = fX.ravel(), fY.ravel()
    h, w = img.shape
    kw, kh = fX.size, fY.size
    dy, dx = kh//2, kw//2
    imgb = cv.copyMakeBorder(img, 0, 0, dx, dx, cv.BORDER_DEFAULT)
    t = np.empty(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            t[y, x] = (imgb[y, x:x+kw]*fX).sum()
    tb = cv.copyMakeBorder(t, dy, dy, 0, 0, cv.BORDER_DEFAULT)
    res = np.empty(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            res[y, x] = (tb[y:y+kh, x]*fY).sum()
    return res
```

Filtri separabili in Python e OpenCV


```python
# Applicazione di un filtro lineare separabile a un'immagine con OpenCV
# Il risultato è memorizzato su un'immagine di interi con segno a 16 bit
def sepFilter2D_cv(img, fx, fy):
    return cv.sepFilter2D(img, cv.CV_16S, fx, fy)

# Misura del tempo di esecuzione delle tre implementazioni
# con un filtro 9x9 con tutti i coefficienti pari a uno.
img = cv.cvtColor(cv.imread('esempi/bolts.png'), cv.COLOR_BGR2GRAY)
f = np.ones((9,9))
fx, fy = np.ones((9,1)), np.ones((1,9))
print("Verifica: ", np.array_equal(fx*fy, f))
implementazioni = [sepFilter2D_py, sepFilter2D_np, sepFilter2D_cv]
for sepFilter2D in implementazioni:
    print(sepFilter2D.__name__, end=': ')
    %timeit sepFilter2D(img, fx, fy)
```

Metodo	Tempo
sepFilter2D_py	12.4 s
sepFilter2D_np	2.54 s
sepFilter2D_cv	0.89 ms

Correlazione e Convoluzione

- L'applicazione di un filtro mediante la formula

$$\mathbf{I}'[y,x] = \sum_{i=-d}^{d} \sum_{j=-d}^{d} \mathbf{F}[i,j] \cdot \mathbf{I}[y+i, x+j]$$

è chiamata *correlazione* di $\mathbf{I}$ con $\mathbf{F}$ e può essere indicata con la notazione:

$$\mathbf{I}' = \mathbf{I} \otimes \mathbf{F}$$

- Cambiando il segno degli offset nella formula si ottiene

$$\mathbf{I}'[y,x] = \sum_{i=-d}^{d} \sum_{j=-d}^{d} \mathbf{F}[i,j] \cdot \mathbf{I}[y-i, x-j]$$

che è chiamata *convoluzione* di $\mathbf{I}$ con $\mathbf{F}$ e può essere indicata con la notazione:

$$\mathbf{I}' = \mathbf{I} * \mathbf{F}$$

Convoluzione

- Si può semplicemente ottenere ribaltando il filtro sui due assi:

$$\mathbf{F}'[i,j] = \mathbf{F}[-i,-j]$$

$\mathbf{F}$

0	0.1	0.1
0.2	0.3	0
0.2	0.1	0

$\mathbf{F}'$

0	0.1	0.2
0	0.3	0.2
0.1	0.1	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0
0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0
0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0
0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0
0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0
0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0
0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	76	178	178	128	51	51	51	26	0	
0	0	0	0	76	230	255	238	170	162	162	85	0	0	
0	0	0	76	230	238	205	171	188	188	120	43	9	0	
0	0	51	204	221	188	188	137	121	121	103				

$$\mathbf{I}' = \mathbf{I} * \mathbf{F} = \mathbf{I} \otimes \mathbf{F}'$$

0	0.1	0.2
0	0.3	0.2
0.1	0.1	0

$\mathbf{F}'$

$\bullet$

255	87	0
87	87	87
255	255	87

$w_{xy}(\mathbf{I})$

$=$ 0.0*255 + 0.1*87 + 0.2*0
+ 0.0*87 + 0.3*87 + 0.2*87
+ 0.1*255 + 0.1*255 + 0.0*87

$= 103$

$\mathbf{I}'[y,x]$

Correlazione vs Convoluzione

- Se il filtro è simmetrico rispetto all'origine, $\mathbf{I} * \mathbf{F} = \mathbf{I} \otimes \mathbf{F}$
- Proprietà algebriche comuni a entrambe
  - Proprietà distributiva rispetto all'addizione: $\mathbf{I} \otimes (\mathbf{F} + \mathbf{G}) = (\mathbf{I} \otimes \mathbf{F}) + (\mathbf{I} \otimes \mathbf{G})$

    $\mathbf{I} * (\mathbf{F} + \mathbf{G}) = (\mathbf{I} * \mathbf{F}) + (\mathbf{I} * \mathbf{G})$
  - Proprietà associativa rispetto a un fattore scalare: $(a \cdot \mathbf{I}) \otimes \mathbf{F} = a \cdot (\mathbf{I} \otimes \mathbf{F})$

    $(a \cdot \mathbf{I}) * \mathbf{F} = a \cdot (\mathbf{I} * \mathbf{F})$
- Solo la convoluzione è *commutativa* e *associativa*:

$$\mathbf{I} * \mathbf{F} = \mathbf{F} * \mathbf{I}$$

$$(\mathbf{I} * \mathbf{F}) * \mathbf{G} = \mathbf{I} * (\mathbf{F} * \mathbf{G})$$

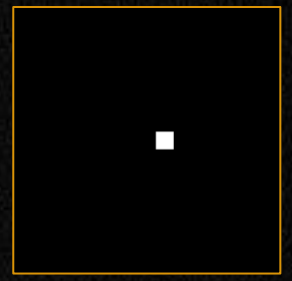
- In OpenCV: la funzione cv.filter2D implementa la correlazione; per ottenere la convoluzione è sufficiente prima ribaltare il filtro con cv.flip(filtro, -1)

Verifica empirica della proprietà associativa e commutativa

I

0	0	0
0	0	1
0	0	0

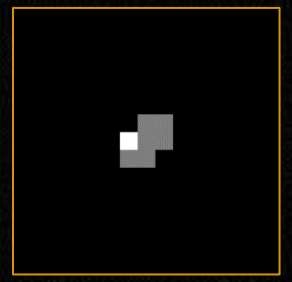
F



F con aggiunta bordo, stesse dimensioni di **I**

0	1	1
2	1	1
1	1	0

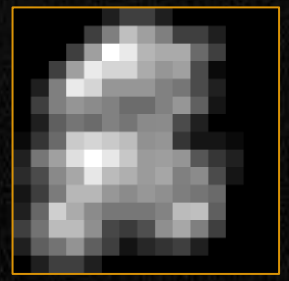
G

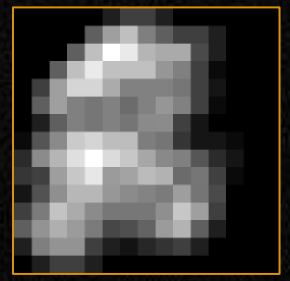


G con aggiunta bordo, stesse dimensioni di **I**

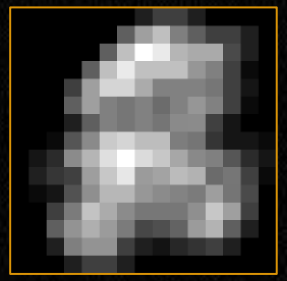
$\mathbf{I} \otimes \mathbf{F} \neq \mathbf{F} \otimes \mathbf{I}$

$\mathbf{I} * \mathbf{F} = \mathbf{F} * \mathbf{I}$





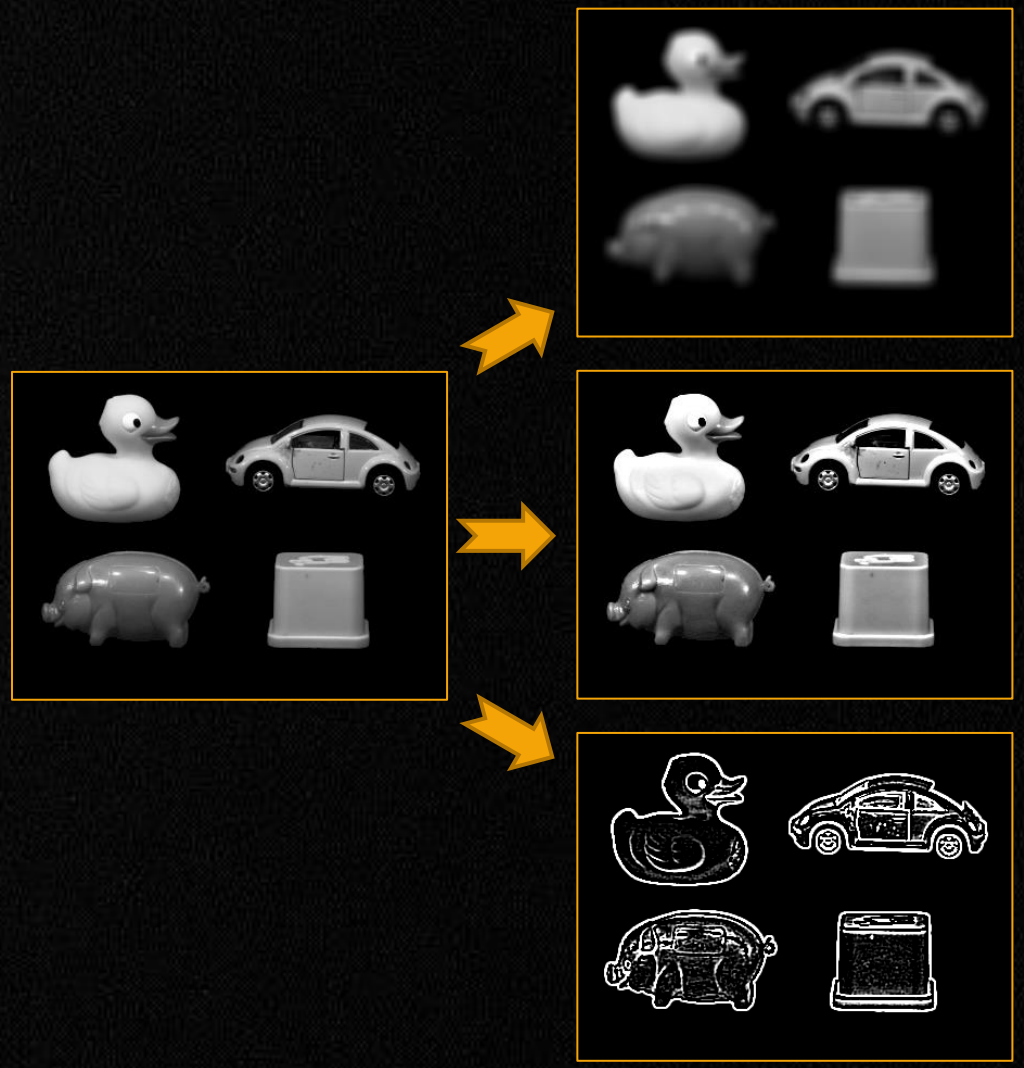
$(\mathbf{I} \otimes \mathbf{F}) \otimes \mathbf{G} \neq \mathbf{I} \otimes (\mathbf{F} \otimes \mathbf{G})$



$(\mathbf{I} * \mathbf{F}) * \mathbf{G} = \mathbf{I} * (\mathbf{F} * \mathbf{G})$

Alcune applicazioni dei filtri lineari

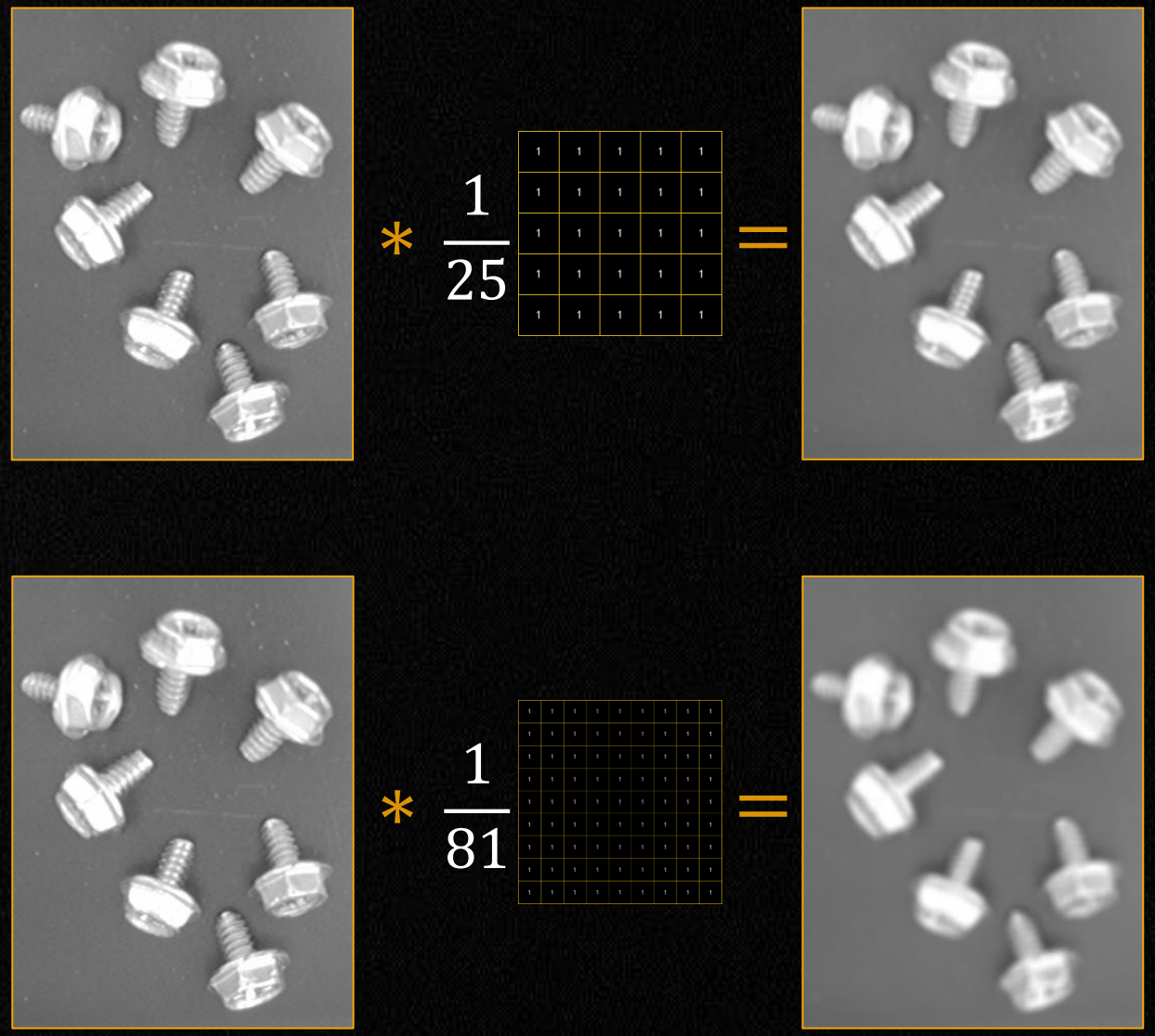
- Riduzione del rumore (regolarizzazione)
  - Box filter (media mobile)
  - Gaussian filter
- Aumento del contrasto locale
  - Filtro di sharpening
- Bordi e derivate dell'immagine
  - Difference of Gaussians (DoG)
  - Filtri derivativi (Prewitt, Sobel, Scharr)
  - Laplacian of Gaussians (LoG)



Box filter

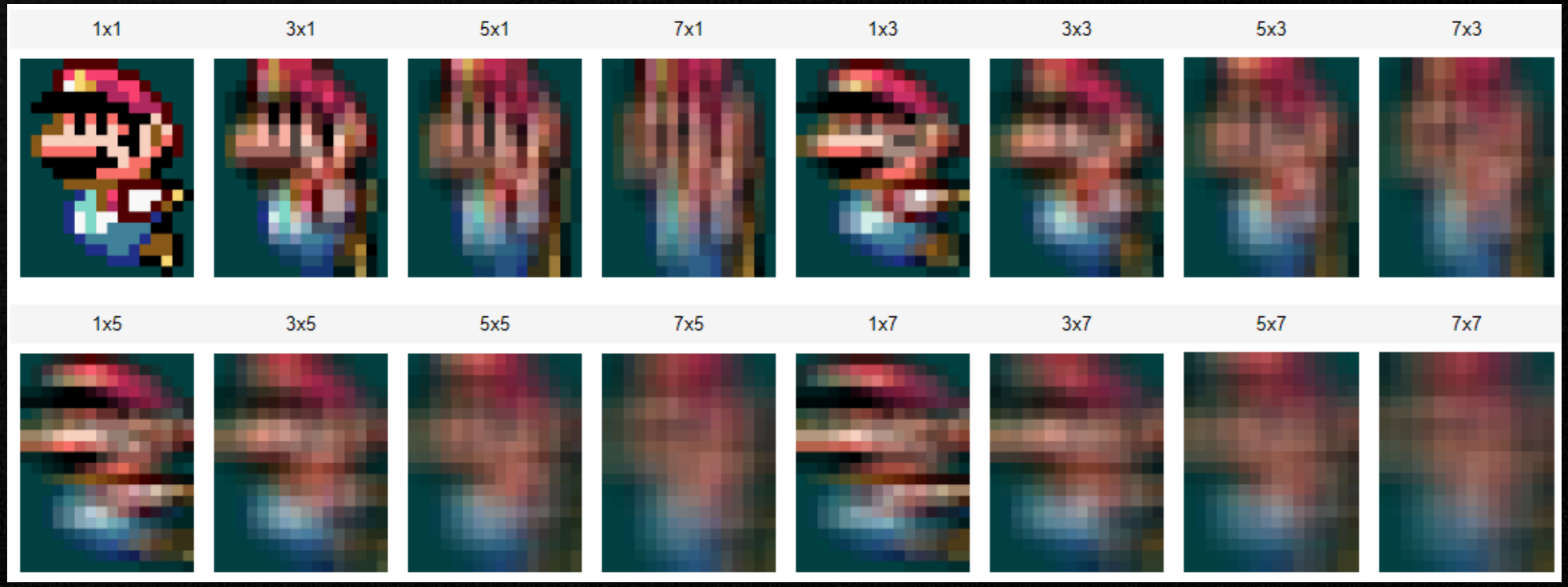
- Box filter
  - Tutti i coefficienti a uno
  - Di solito normalizzato dividendo per il numero di elementi → media mobile
- Effetto "sfocatura" (blur): attenua il contrasto locale, riduce il rumore
- In OpenCV è disponibile la funzione cv.boxFilter: un'implementazione specifica per questo filtro che è più efficiente di cv.filter2D e cv.sepFilter2D

Metodo	Tempo
cv.filter2D	2.62 ms
cv.sepFilter2D	0.89 ms
cv.boxFilter	0.17 ms



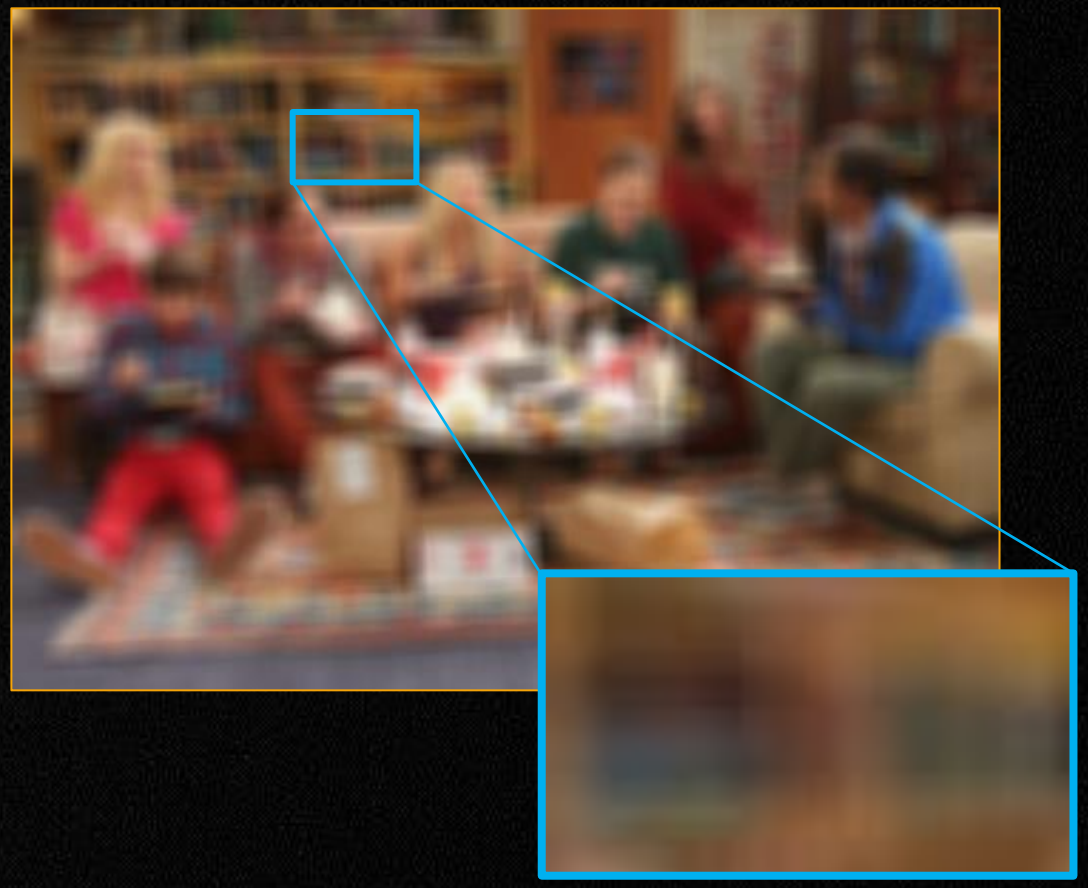
Filtri su immagini a colori


```
# Nelle immagini a colori, OpenCV applica lo stesso filtro a ciascun canale
img = cv.imread('esempi/mario-c.png')

# Box filter con normalizzazione
smoothed = cv.boxFilter(img, -1, (7,7), normalize = True)

# La funzione cv.blur eqivale a chiamare boxFilter con normalize=True
test = [cv.blur(img, (sx, sy)) for sx in range(1, 10, 2) for sy in range(1, 10, 2)]
```



Box filter: semplice ed efficiente, ma…

- Può produrre artefatti nell'immagine, che possono essere visibili all'aumentare della dimensione del filtro

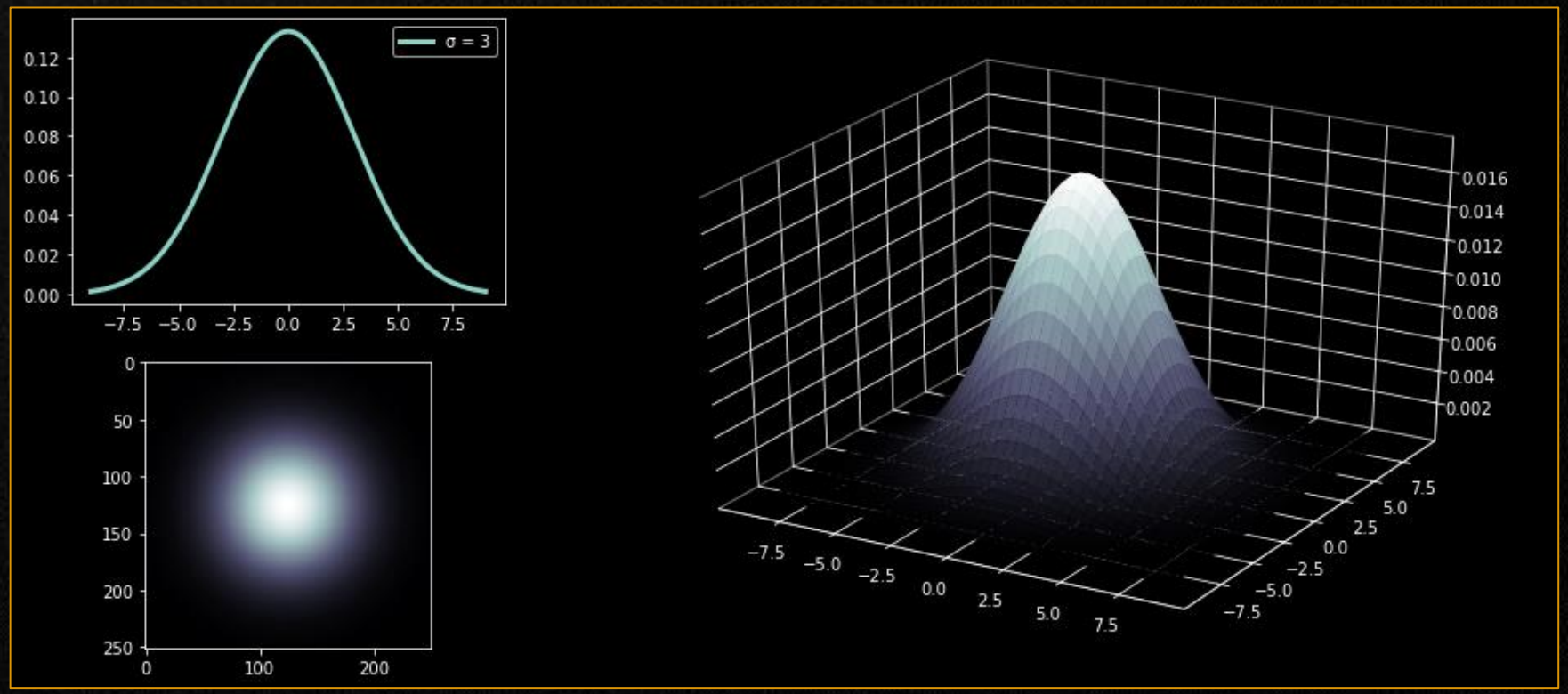


Filtro gaussiano

- Per risolvere il problema, servirebbe un filtro con pesi che diminuiscono spostandosi dal centro verso l'esterno
- La funzione gaussiana in due dimensioni è l'ideale (e produce filtri separabili!)

$$G_{1D}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

$$G_{2D}(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Creare un filtro gaussiano

- La funzione gaussiana 2D $G_{2D}(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ può essere scritta come

$$G_{2D}(x,y) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^2 e^{-\frac{x^2}{2\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} = G_{1D}(x) \cdot G_{1D}(y)$$

  - Un filtro ottenuto da G_{2D} è il prodotto di due filtri 1D identici e quindi separabile

- Per creare un filtro gaussiano $m \times m$, è necessario scegliere la dimensione m e il valore di σ
  - Per un'approssimazione discreta si può moltiplicare per un fattore opportuno e arrotondare i coefficienti

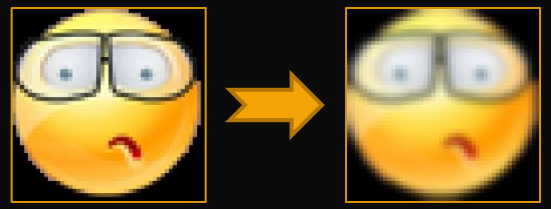
m=5, $\sigma = 1$

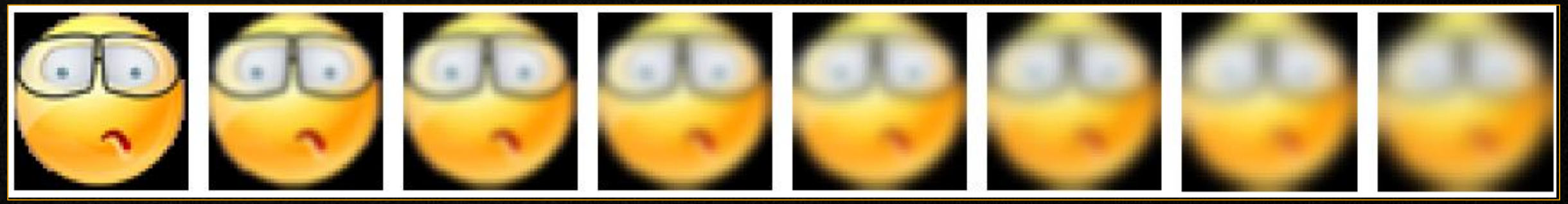
$$\underbrace{\begin{bmatrix} 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \\ 0.013 & 0.06 & 0.098 & 0.06 & 0.013 \\ 0.022 & 0.098 & 0.16 & 0.098 & 0.022 \\ 0.013 & 0.06 & 0.098 & 0.06 & 0.013 \\ 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \end{bmatrix}}_{\mathbf{F}} = \underbrace{\begin{bmatrix} 0.054 \\ 0.24 \\ 0.4 \\ 0.24 \\ 0.054 \end{bmatrix}}_{\mathbf{F}_1} \cdot \underbrace{\begin{bmatrix} 0.054 & 0.24 & 0.4 & 0.24 & 0.054 \end{bmatrix}}_{\mathbf{F}_1^\top}$$

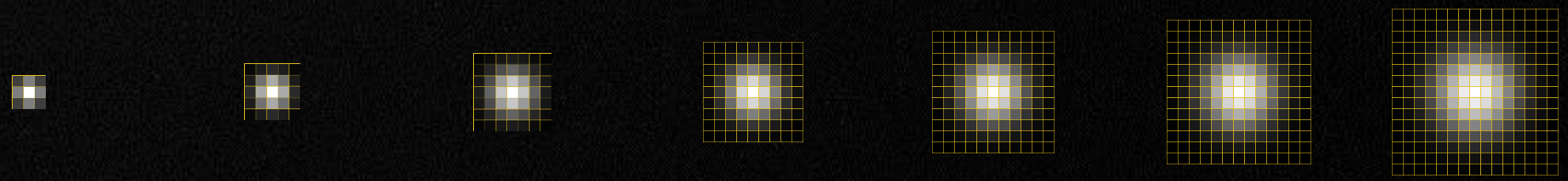
$$\underbrace{\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 28 & 16 & 4 \\ 7 & 28 & 49 & 28 & 7 \\ 4 & 16 & 28 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}}_{\mathbf{F}} = \underbrace{\begin{bmatrix} 1 \\ 4 \\ 7 \\ 4 \\ 1 \end{bmatrix}}_{\mathbf{F}_1} \cdot \underbrace{\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \end{bmatrix}}_{\mathbf{F}_1^\top}$$

Regolarizzazione con un filtro gaussiano (Gaussian blur)


```
img = cv.imread('esempi/study.png')

# Crea un filtro gaussiano 1D e lo passa a sepFilter2D
g = cv.getGaussianKernel(5, 1) # m=5, σ=1
img_blurred = cv.sepFilter2D(img, -1, g, g)

# Ancora più semplice utilizzare direttamente l'apposita funzione GaussianBlur
img_blurred2 = cv.GaussianBlur(img, (5,5), 1) # m=5, σ=1
# N.B. se σ<=0, la funzione calcola σ da m come σ = 0.3*((m-1)*0.5 - 1) + 0.8
list_img_blurred = [cv.GaussianBlur(img, (m,m), 0) for m in range(3,17,2)]
```







Box filter

Gaussian filter

Sharpening

- I filtri appena visti, oltre all'effetto "blur", possono essere impiegati per ottenere un risultato opposto: una sorta di "miglioramento della messa a fuoco", chiamato sharpening.
- Si procede calcolando un'immagine "blurred" mediante convoluzione con un filtro $\mathbf{F}_\mathrm{b}$ normalizzato (es. box filter o gaussiano) e sottraendola all'originale:

$$\mathbf{M} = \mathbf{I} - \mathbf{I} * \mathbf{F}_\mathrm{b}$$

- L'immagine $\mathbf{M}$ (pixel con valori interi) è infine sommata all'immagine originale, dopo essere stata moltiplicata per un parametro k che controlla l'intensità dell'effetto:

$$\mathbf{I}' = \mathbf{I} + k \cdot \mathbf{M}$$

$\mathbf{I} * \mathbf{F}_\mathrm{b} = \mathbf{I} * \frac{1}{81} \mathbf{F}_\mathrm{b}$

$\mathbf{M} = \mathbf{I} - \mathbf{I} * \mathbf{F}_\mathrm{b}$

$\mathbf{I}' = \mathbf{I} + k \cdot \mathbf{M}$

Immagine originale

Risultato sharpening

Immagine originale

Risultato sharpening

Filtro di sharpening

- Grazie alle proprietà algebriche della convoluzione, possiamo costruire un singolo filtro che esegua l'intera operazione di sharpening.

$$\mathbf{M} = \mathbf{I} - \mathbf{I} * \mathbf{F}_{\mathrm{b}} = \mathbf{I} * \mathbf{F}_{id} + \mathbf{I} * (-\mathbf{F}_{\mathrm{b}}), \text{dove } \mathbf{F}_{id} \text{ è un filtro con solo un 1 al centro}$$

- Applicando (al contrario) la proprietà distributiva $\mathbf{I} * (\mathbf{F} + \mathbf{G}) = (\mathbf{I} * \mathbf{F}) + (\mathbf{I} * \mathbf{G})$, si ha:

$$\mathbf{M} = \mathbf{I} * \mathbf{F}_{id} + \mathbf{I} * (-\mathbf{F}_{\mathrm{b}}) = \mathbf{I} * (\mathbf{F}_{id} - \mathbf{F}_{\mathrm{b}}) \Rightarrow \mathbf{M} = \mathbf{I} * \mathbf{F}_L, \text{con } \mathbf{F}_L = \mathbf{F}_{id} - \mathbf{F}_{\mathrm{b}}$$

- Il risultato dello sharpening $\mathbf{I}' = \mathbf{I} + k \cdot \mathbf{M}$ può essere quindi scritto come:

$$\mathbf{I}' = \mathbf{I} + k \cdot (\mathbf{I} * \mathbf{F}_L)$$

- Applicando la proprietà associativa rispetto a uno scalare e la commutativa, si ottiene:

$$\mathbf{I}' = \mathbf{I} + k \cdot (\mathbf{I} * \mathbf{F}_L) = \mathbf{I} + \mathbf{I} * (k \cdot \mathbf{F}_L)$$

- Infine applicando nuovamente la proprietà distributiva:

$$\mathbf{I}' = \mathbf{I} + \mathbf{I} * (k \cdot \mathbf{F}_L) = \mathbf{I} * \mathbf{F}_{id} + \mathbf{I} * (k \cdot \mathbf{F}_L) = \mathbf{I} * \left(\mathbf{F}_{id} + (k \cdot \mathbf{F}_L)\right)$$

$$\mathbf{I}' = \mathbf{I} * \mathbf{F}_s, \text{con } \mathbf{F}_s = \mathbf{F}_{id} + (k \cdot \mathbf{F}_L) = \mathbf{F}_{id} + k \cdot (\mathbf{F}_{id} - \mathbf{F}_{\mathrm{b}})$$

Costruzione di un filtro di sharpening: esempio con $m = 3$ e $k = 1.5$

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

$\mathbf{F}_{id}$

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

$\mathbf{F}_{\mathrm{b}}$

$$\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$\mathbf{F}_L = \mathbf{F}_{id} - \mathbf{F}_{\mathrm{b}}$

$$\frac{1}{6}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 14 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$\mathbf{F}_s = \mathbf{F}_{id} + (1.5 \cdot \mathbf{F}_L)$

$\mathbf{I}$

$\mathbf{I} * \mathbf{F}_{\mathrm{b}}$

$\mathbf{M} = \mathbf{I} * \mathbf{F}_L$

$\mathbf{I}' = \mathbf{I} * \mathbf{F}_s$

Filtro di sharpening in OpenCV


```
# Metodo 1: filtro di blur e operazioni aritmetiche fra immagini
def sharpen(img, gaussian_blur, m, k):
    blurred = cv.GaussianBlur(img, (m,m), 0) if gaussian_blur else cv.blur(img, (m,m))
    mask = img.astype(np.int16) - blurred
    return np.clip(np.round(img.astype(float)+k*mask), 0, 255).astype(np.uint8)

# Metodo 2: creazione di un unico filtro che fa tutto
def create_sharpen_filter(gaussian_blur, m, k):
    if gaussian_blur:
        g = cv.getGaussianKernel(m, 0); F_b = g.reshape(1,-1)*g
    else:
        F_b = np.ones((m,m), np.float32)
    F_id = np.zeros_like(F_b); F_id[m//2, m//2] = 1
    F_b /= F_b.sum() # Normalizza il filtro di blur
    return F_id + k*(F_id - F_b)

img = cv.imread('esempi/kernel.png', cv.IMREAD_GRAYSCALE)
f = create_sharpen_filter(False, 5, 2)
res1 = sharpen(img,  False, 5, 2)
res2 = cv.filter2D(img, -1, f)
```

```
9.78 ms ± 290 µs per loop
576 µs ± 11.3 µs per loop
```

Bordi

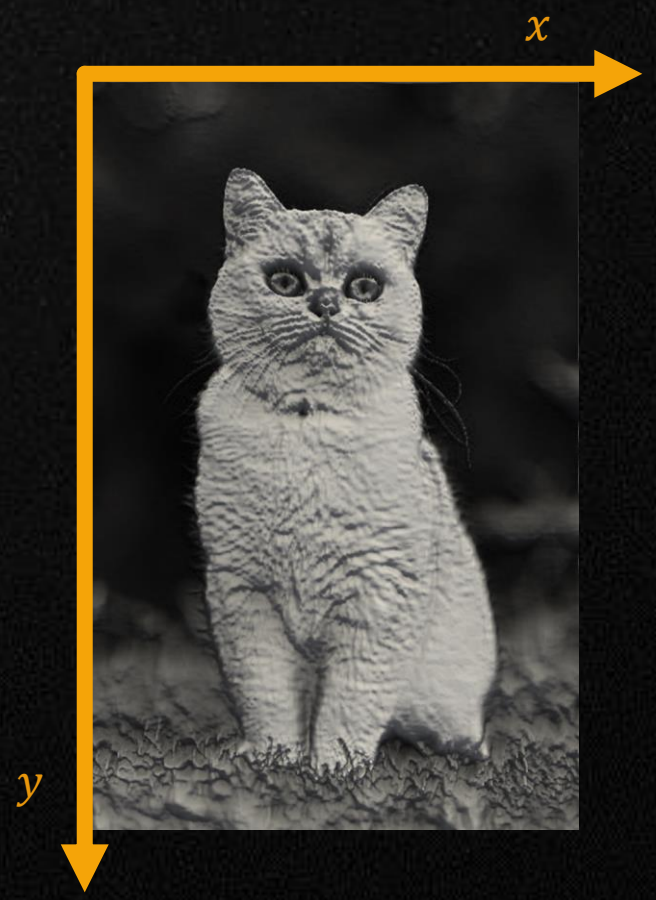
- Bordo di un oggetto
  - La separazione tra l'oggetto e lo sfondo o tra l'oggetto e altri oggetti
  - In genere molto più stabile rispetto alle informazioni di colore e tessitura, rispetto a illuminazione, rumore, ...
  - È utile per poterne interpretare forma geometrica
- Estrazione dei bordi (edge detection)
  - Un'operazione molto importante, spesso il primo passo per l'individuazione dell'oggetto

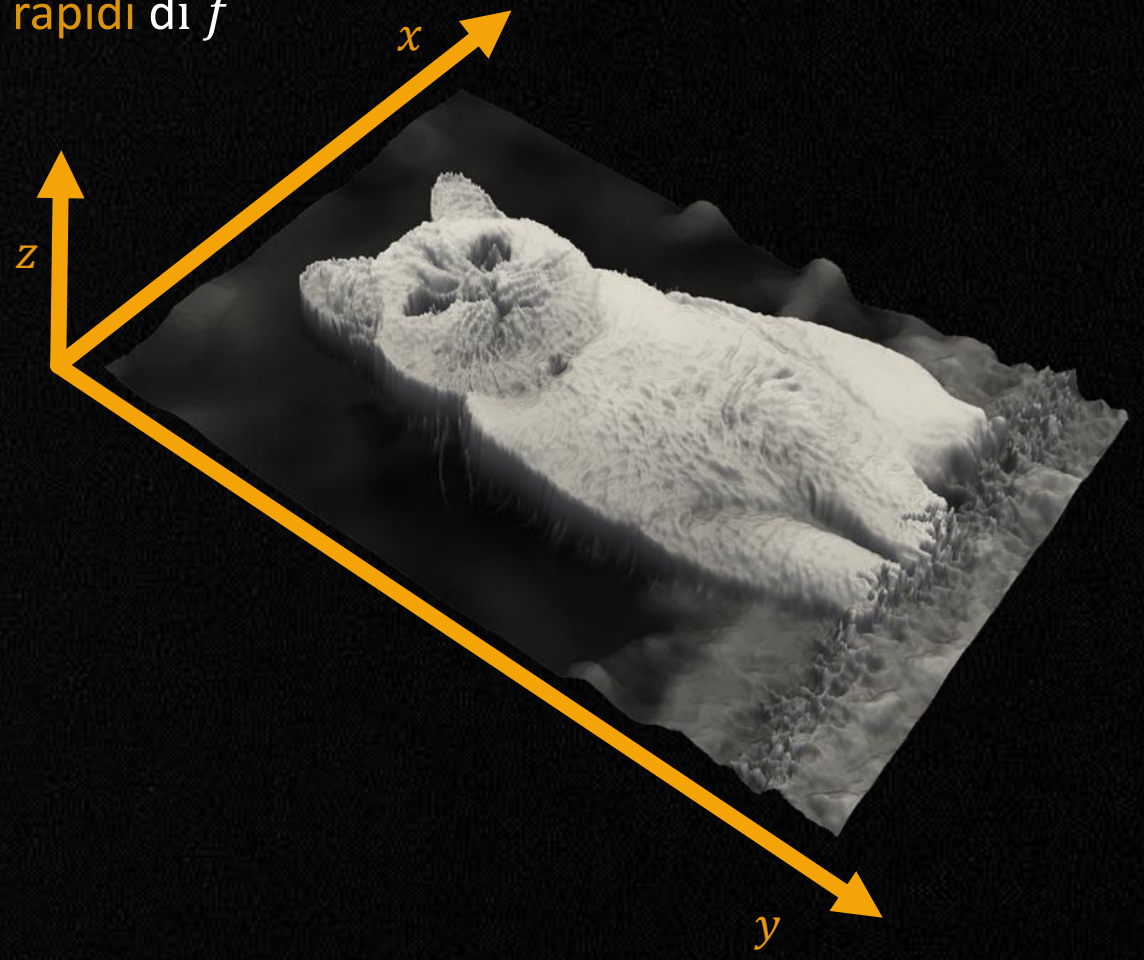
Cosa sono i bordi?

- Se consideriamo un'immagine grayscale $\mathbf{I}$ come una funzione

$$f: \mathbb{R} \times \mathbb{R} \to \mathbb{R}, \text{con } f(x, y) = \mathbf{I}[y, x]$$

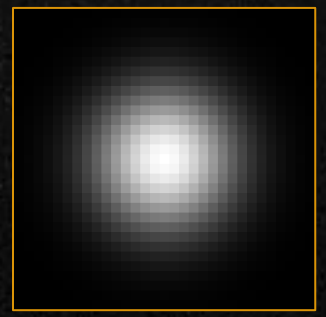
- i bordi si trovano in corrispondenza di cambiamenti rapidi di f





Differenza fra immagine e risultato di un Gaussian blur

- Abbiamo già osservato che sottraendo il risultato dello smooth di un'immagine dall'immagine originale, si possono evidenziare le zone caratterizzate da cambiamenti rapidi fra valori chiari e scuri dei pixel.



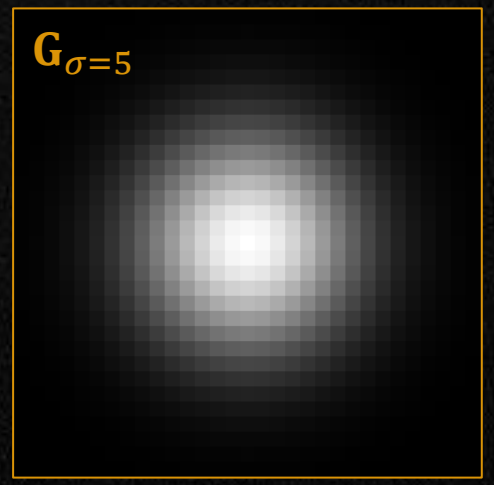
$\mathbf{G}_{\sigma=5}$

$\mathbf{I}$

$\mathbf{I} * \mathbf{G}_{\sigma=5}$

$\mathbf{I} - (\mathbf{I} * \mathbf{G}_{\sigma=5})$

Differenza fra immagine e risultato di un Gaussian blur



Difference of Gaussians (DoG)

- Il filtro gaussiano riduce le alte frequenze nell'immagine
  - $\mathbf{I} * \mathbf{G}_\sigma$ contiene sostanzialmente le basse frequenze dell'immagine
  - $\mathbf{I} - (\mathbf{I} * \mathbf{G}_\sigma)$ contiene le alte frequenze
- Dati due filtri $\mathbf{G}_{\sigma_1}$ e $\mathbf{G}_{\sigma_2}$, la differenza $(\mathbf{I} * \mathbf{G}_{\sigma_1}) - (\mathbf{I} * \mathbf{G}_{\sigma_2})$ evidenzia uno specifico range di frequenze nell'immagine
  - Con un'opportuna normalizzazione si possono evidenziare i bordi di un determinato spessore (nell'esempio a fianco sono stati considerati solo i valori positivi, riscalati fra 0 e 255).

$\mathbf{I} * \mathbf{G}_\sigma$

$\mathbf{I} - (\mathbf{I} * \mathbf{G}_\sigma)$

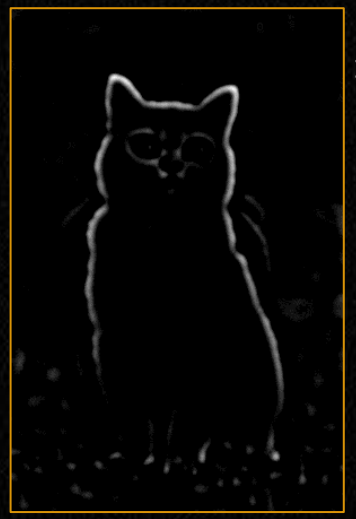
$\mathbf{G}_{\sigma_1}$

$\mathbf{G}_{\sigma_2}$

$\mathbf{I} * \mathbf{G}_{\sigma_1}$

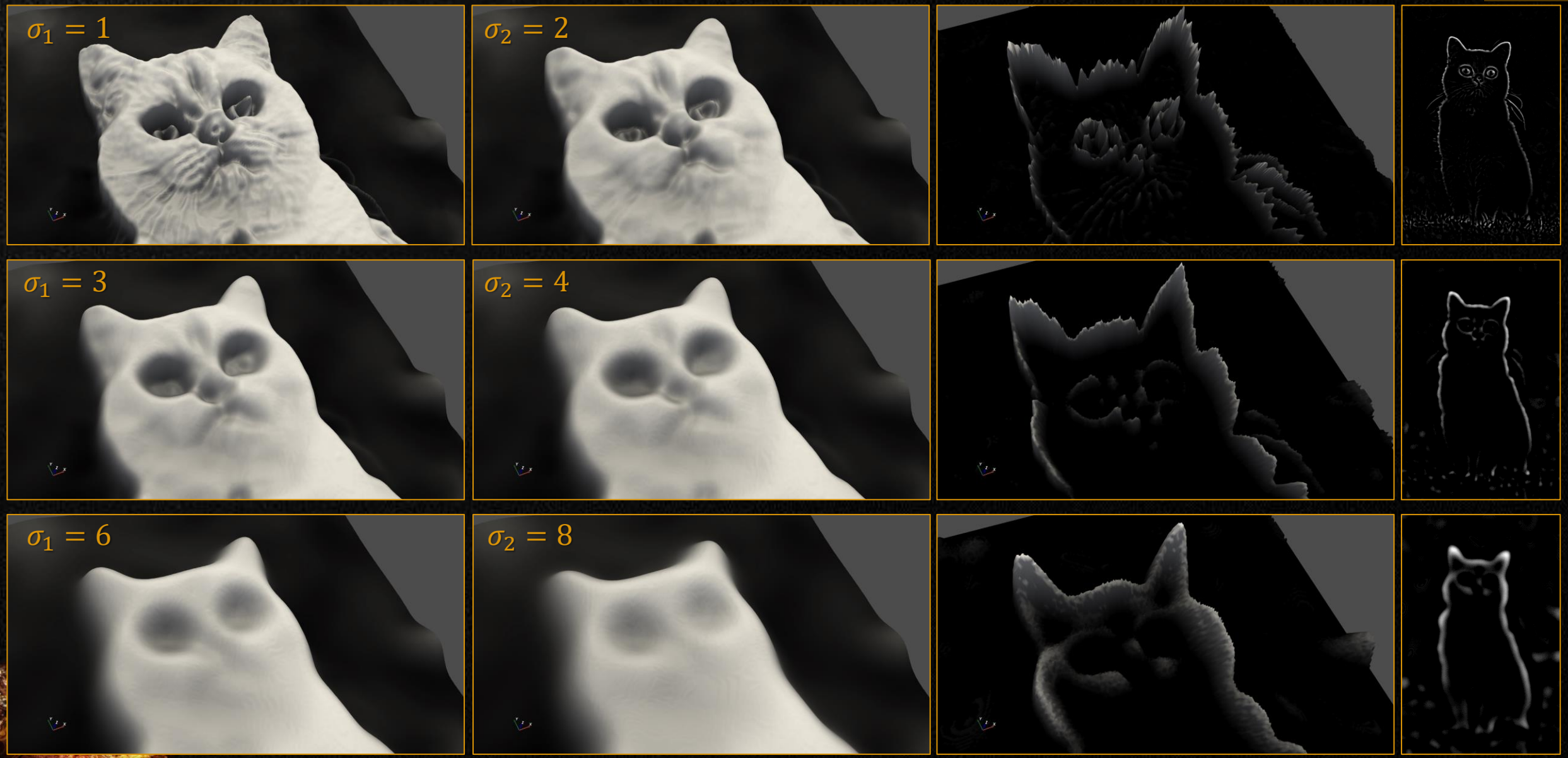
$\mathbf{I} * \mathbf{G}_{\sigma_2}$

$\mathbf{I} * \mathbf{G}_{\sigma_1} - \mathbf{I} * \mathbf{G}_{\sigma_2}$



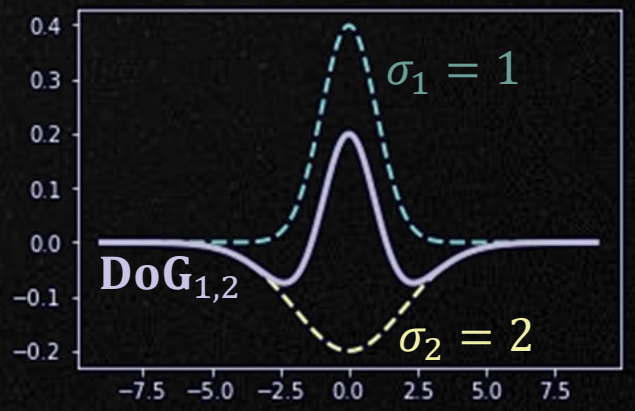
Normalizzazione

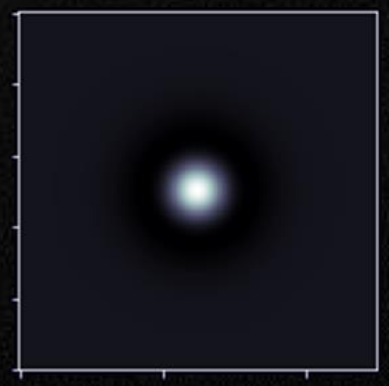
Difference of Gaussians (DoG) - Esempi

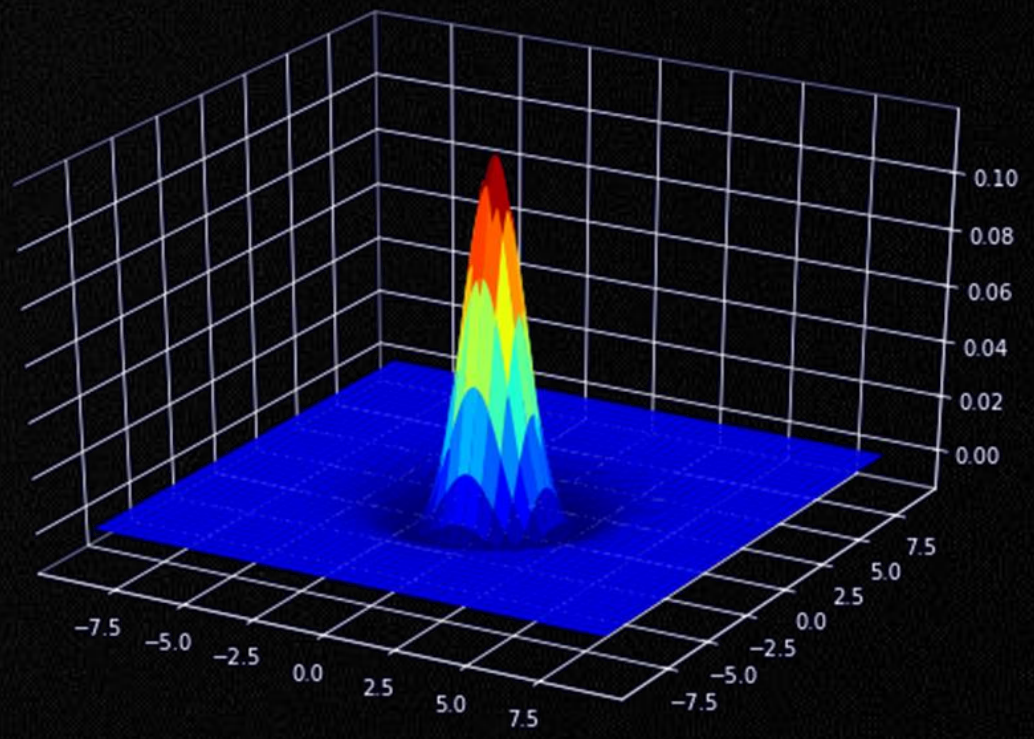


Difference of Gaussians: costruzione del filtro DoG

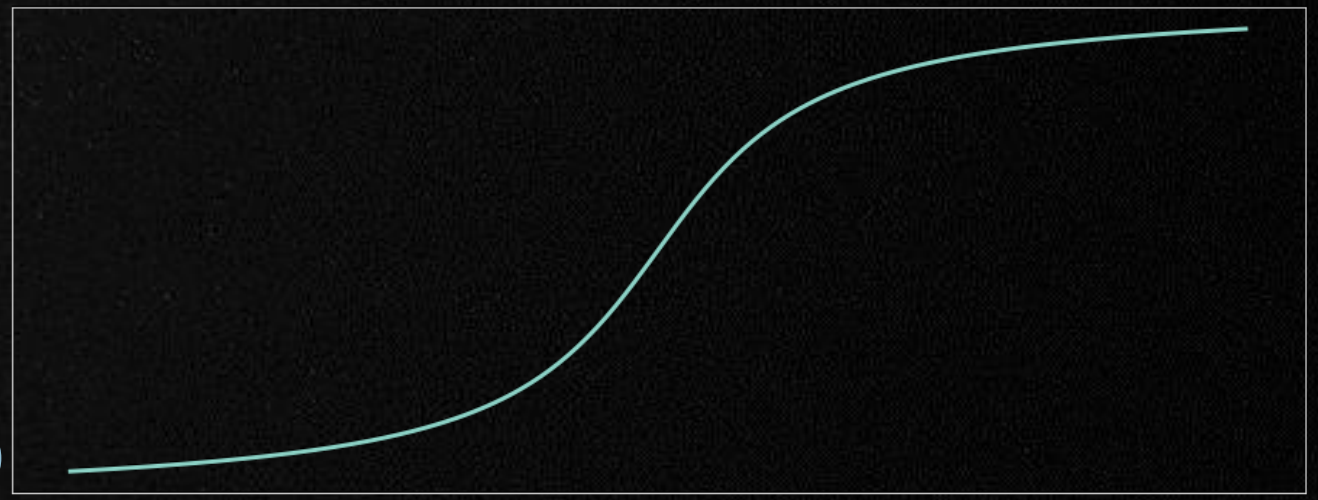
- Applicando (al contrario) la proprietà distributiva $\mathbf{I} * (\mathbf{F} + \mathbf{G}) = (\mathbf{I} * \mathbf{F}) + (\mathbf{I} * \mathbf{G})$, si ha:
  - $(\mathbf{I} * \mathbf{G}_{\sigma_1}) - (\mathbf{I} * \mathbf{G}_{\sigma_2}) = \mathbf{I} * (\mathbf{G}_{\sigma_1} - \mathbf{G}_{\sigma_2}) = \mathbf{I} * \mathbf{DoG}_{\sigma_1,\sigma_2}$, con $\mathbf{DoG}_{\sigma_1,\sigma_2} = \mathbf{G}_{\sigma_1} - \mathbf{G}_{\sigma_2}$
- L'operazione si può quindi ottenere applicando un unico filtro, ottenuto dalla differenza di due filtri gaussiani

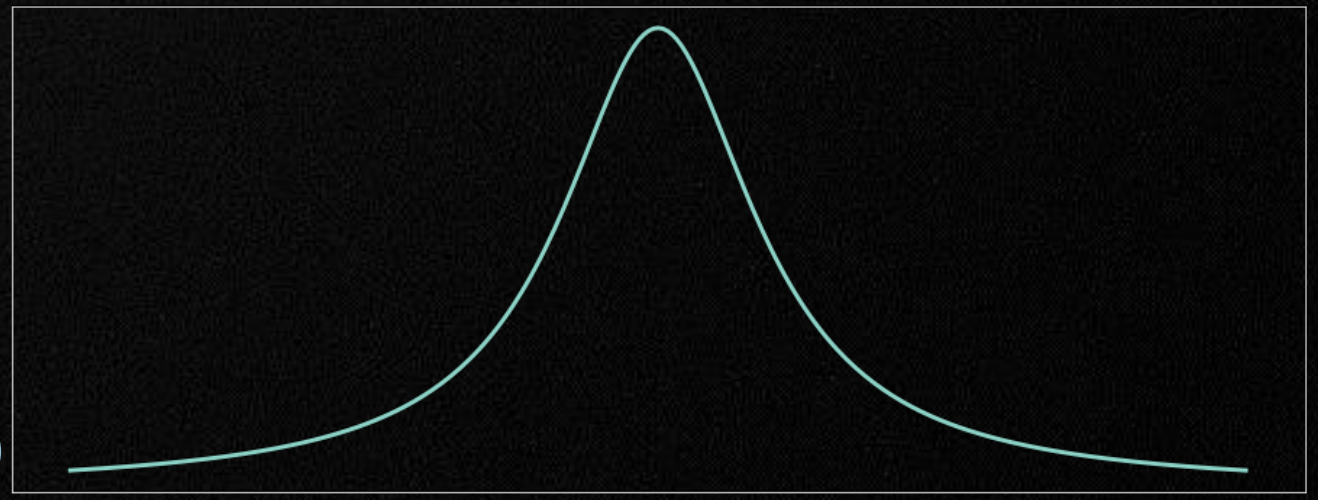






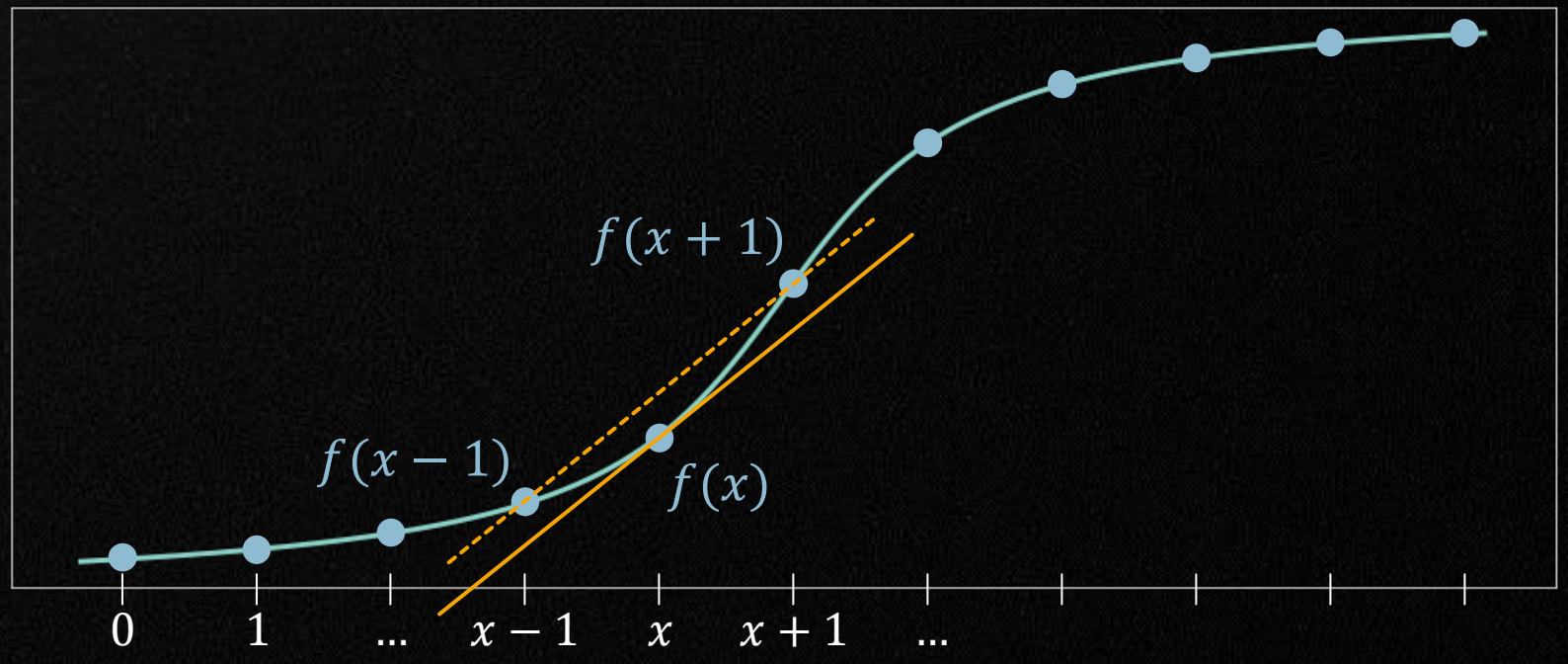
Derivata di una funzione





Derivata di una funzione discreta

- Limite del rapporto incrementale: $f'(x) = \lim_{t \to 0} \frac{f(x+t) - f(x)}{t}$
- Immagini digitali: i valori sono discreti, non è possibile calcolare esattamente il limite, ma si può stimare per un valore piccolo di t.
- Stima mediante differenza finita "centrata" (più accurata di altre): $f'(x) \approx \frac{f(x+1) - f(x-1)}{2}$

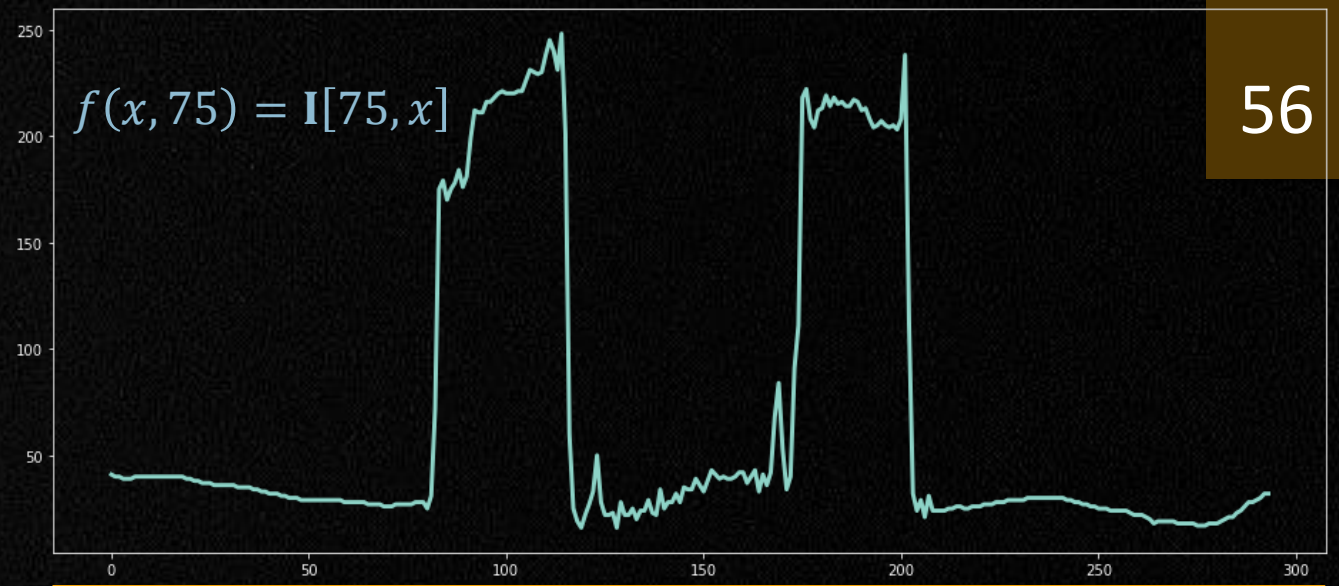


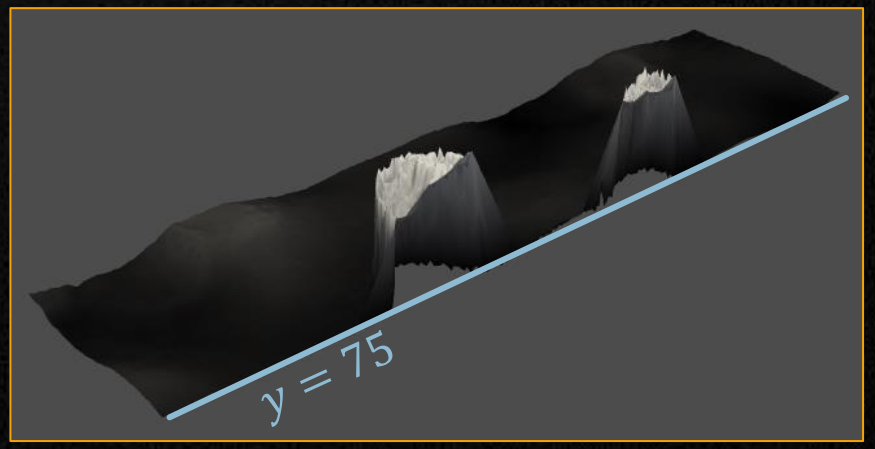
Derivate parziali

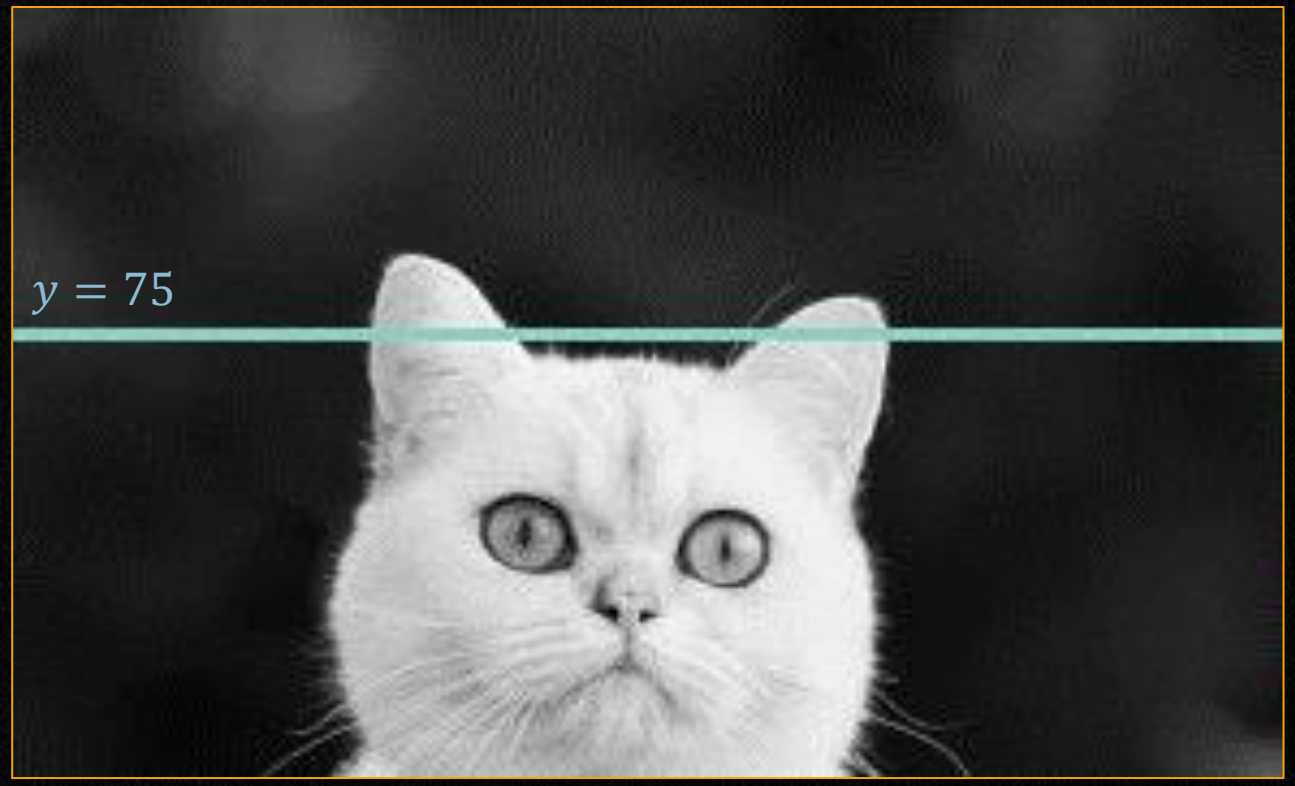
- L'immagine è una funzione discreta di due variabili
- Stima derivate parziali di $f(x, y)$:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x-1,y)}{2}$$

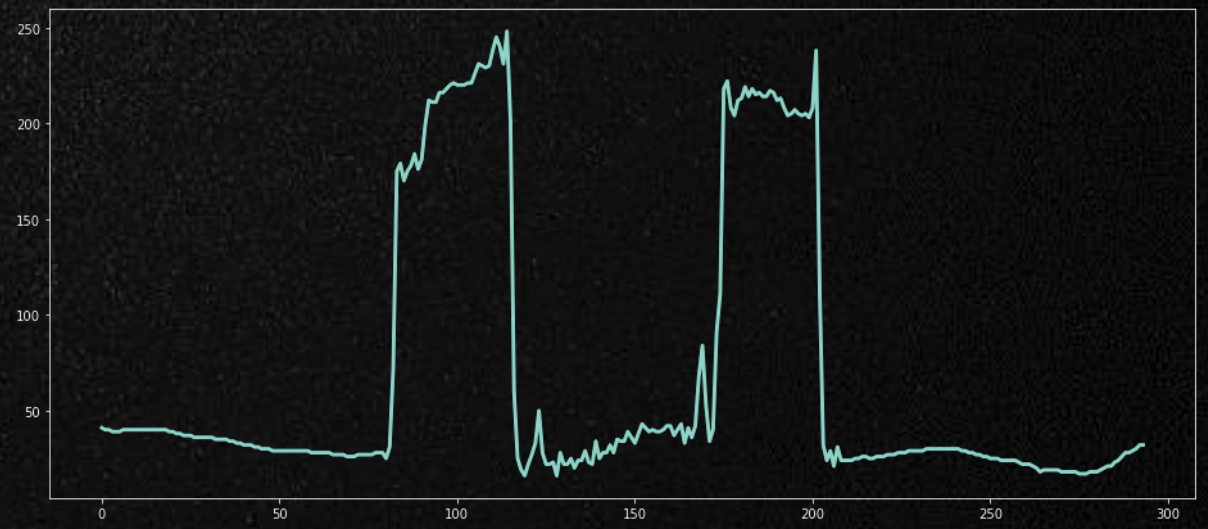
$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+1) - f(x,y-1)}{2}$$

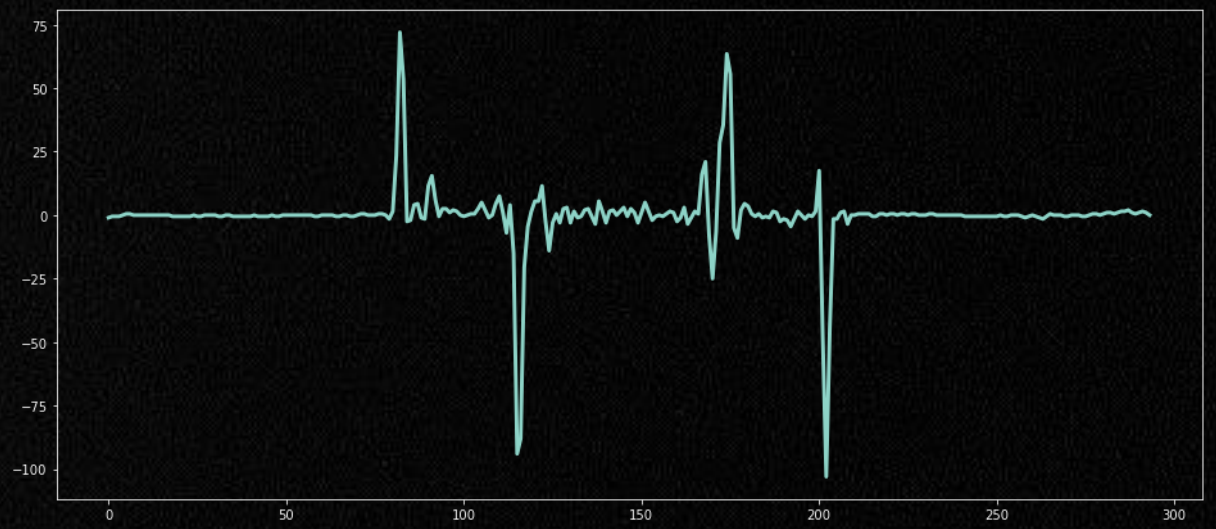






Come calcolare le derivate parziali sull'immagine?





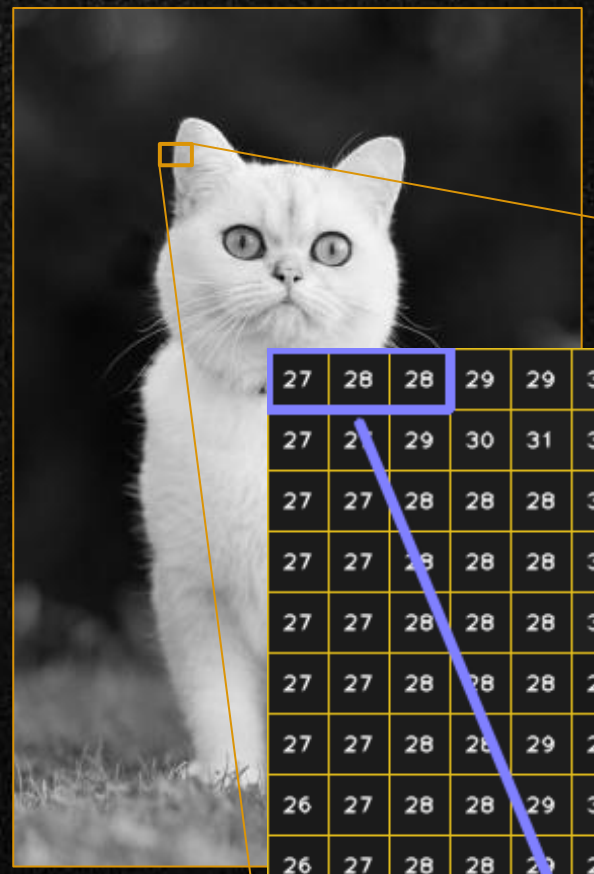
$$f(x, 75) = \mathbf{I}[75, x]$$

$$\frac{\partial f(x,75)}{\partial x} = \frac{\mathbf{I}[75, x+1] - \mathbf{I}[75, x-1]}{2}$$

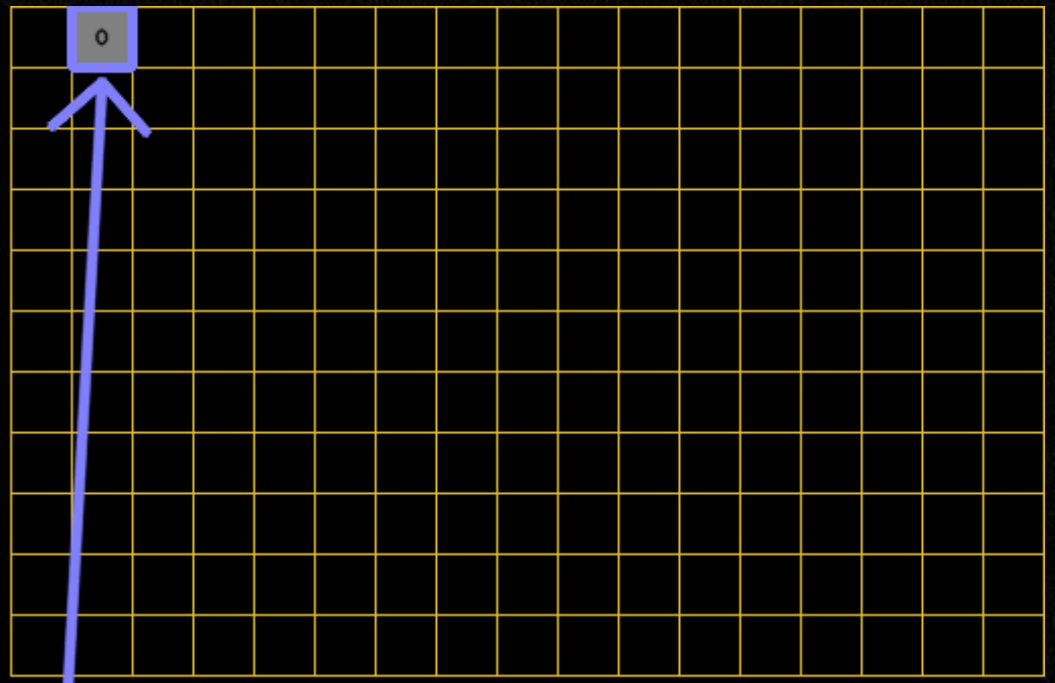
- Costruiamo un filtro che calcoli $\frac{\partial f(x,y)}{\partial x}$:

$$\otimes \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Come calcolare le derivate parziali sull'immagine?



27	28	28	29	29	34	39	67	132	192	171	181	189	185	186	187	191
27	27	29	30	31	33	35	65	138	195	185	198	182	187	187	187	192
27	27	28	28	28	31	47	67	148	191	184	187	183	186	186	186	190
27	27	28	28	28	31	44	70	159	192	181	185	186	189	182	182	189
27	27	28	28	28	32	40	74	174	191	178	183	188	183	187	187	186
27	27	28	28	28	25	31	72	175	179	170	175	178	184	176	181	199
27	27	28	28	29	26	31	78	176	174	171	176	173	169	182	191	194
26	27	28	28	29	30	36	88	178	176	173	179	173	175	178	188	199
26	27	28	28	29	28	33	91	175	175	166	173	170	172	185	192	190
26	27	28	28	29	31	36	98	180	180	166	174	174	175	178	184	189
26	27	27	28	28	33	47	116	189	172	171	175	167	175	176	180	185



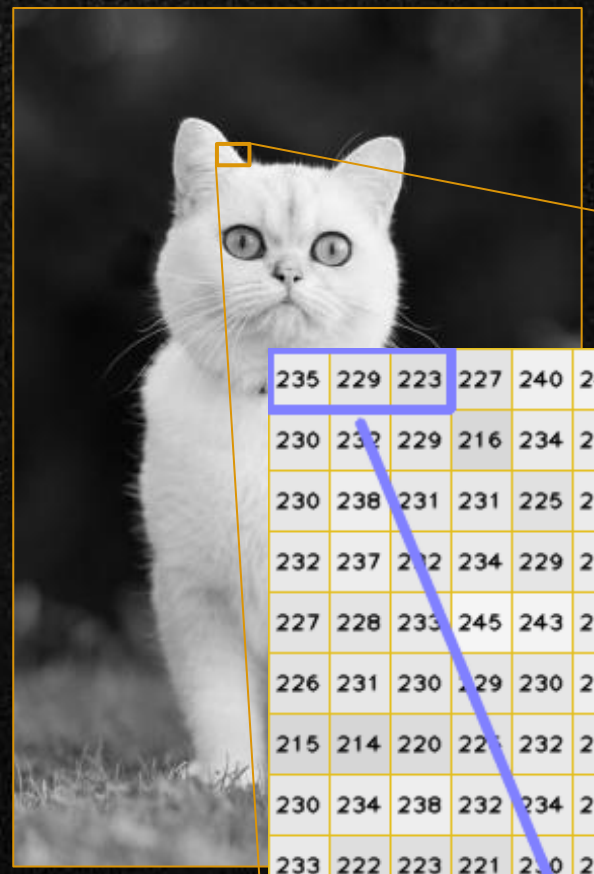
-0.5	0	0.5

$\bullet$

27	28	28

$= -0.5*27 + 0.0*28 + 0.5*28 = 0$

Come calcolare le derivate parziali sull'immagine?



235	229	223	227	240	242	159	76	24	29	22	22	21	28	23	24	23
230	232	229	216	234	234	242	172	51	37	15	32	27	25	22	23	22
230	238	231	231	225	235	240	235	148	46	30	37	25	15	24	21	21
232	237	232	234	229	235	236	246	232	92	76	42	21	29	22	20	21
227	228	233	245	243	236	224	245	249	194	150	31	22	22	19	22	26
226	231	230	229	230	238	245	240	231	248	201	60	25	19	16	22	27
215	214	220	226	232	231	229	228	253	242	234	142	34	26	29	24	24
230	234	238	232	234	234	239	230	245	242	254	220	120	41	35	28	23
233	222	223	221	230	231	234	242	217	240	245	255	229	114	57	46	43
233	235	239	228	228	225	230	228	236	236	231	250	249	211	120	83	85
221	220	219	221	225	227	227	234	233	237	238	238	248	241	214	151	159

−6		

−0.5	0	0.5

$\bullet$

235	229	223

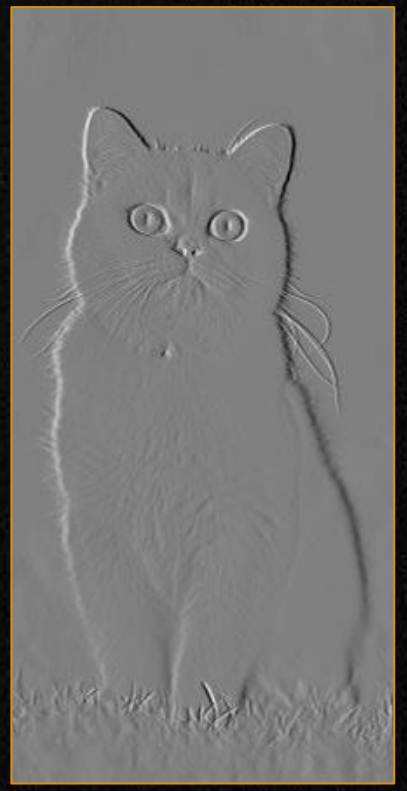
$= -0.5*235 + 0.0*229 + 0.5*223 = -6$

Derivate parziali di un'immagine

$\mathbf{I}$

$$\otimes \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} =$$

$\mathbf{F}_x$



$\frac{\partial \mathbf{I}}{\partial x}$

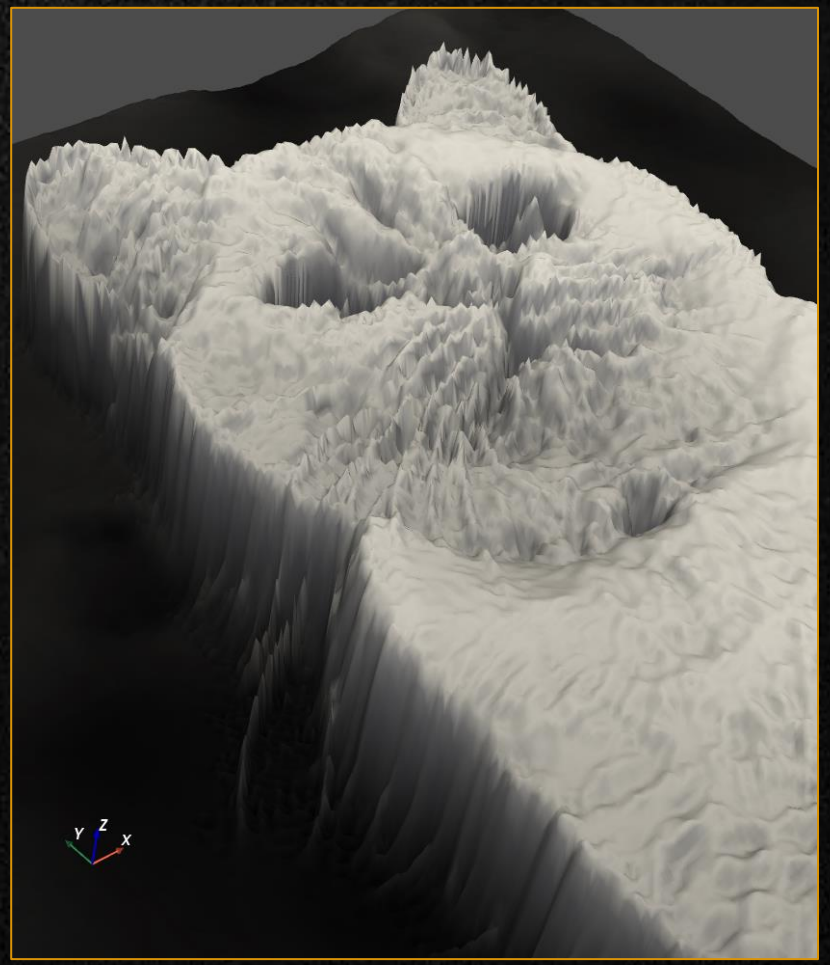
$\mathbf{I}$

$$\otimes \frac{1}{2}\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} =$$

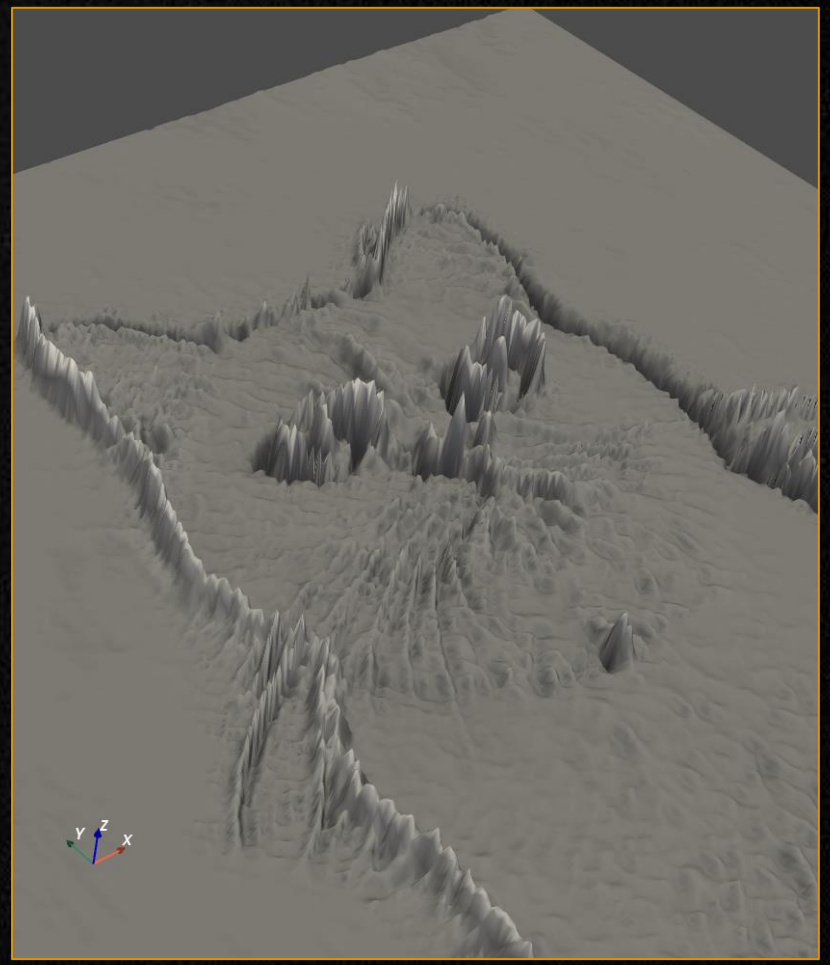
$\mathbf{F}_y$

$\frac{\partial \mathbf{I}}{\partial y}$

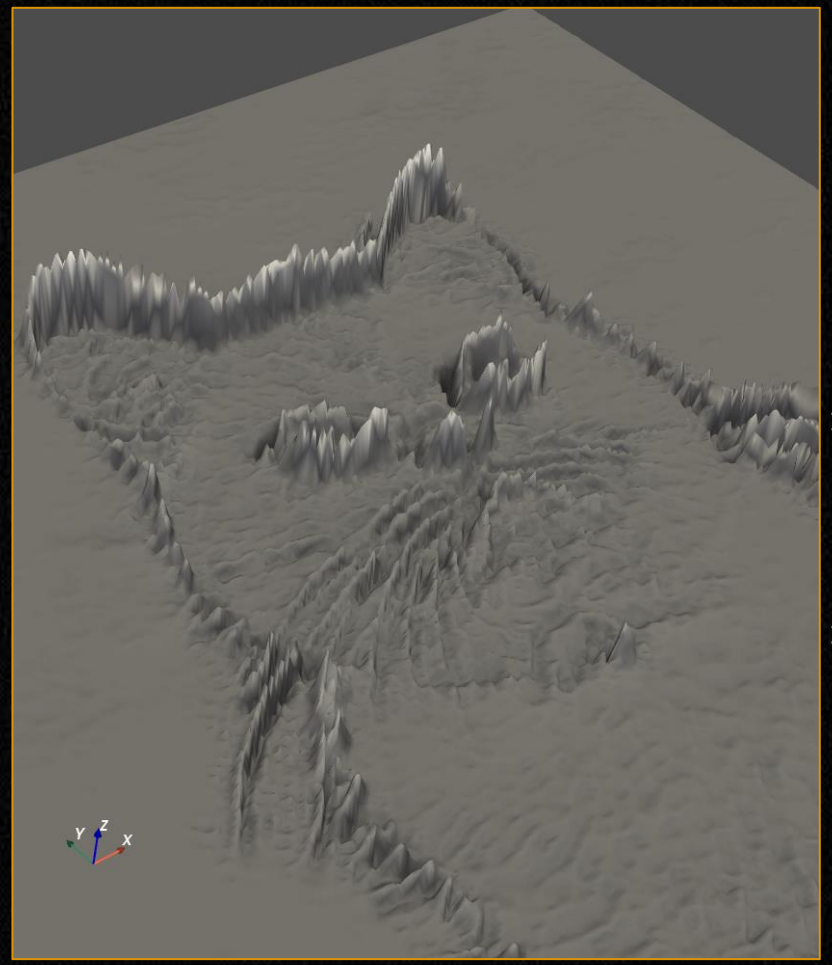
Derivate parziali di un'immagine



$\mathbf{I}$



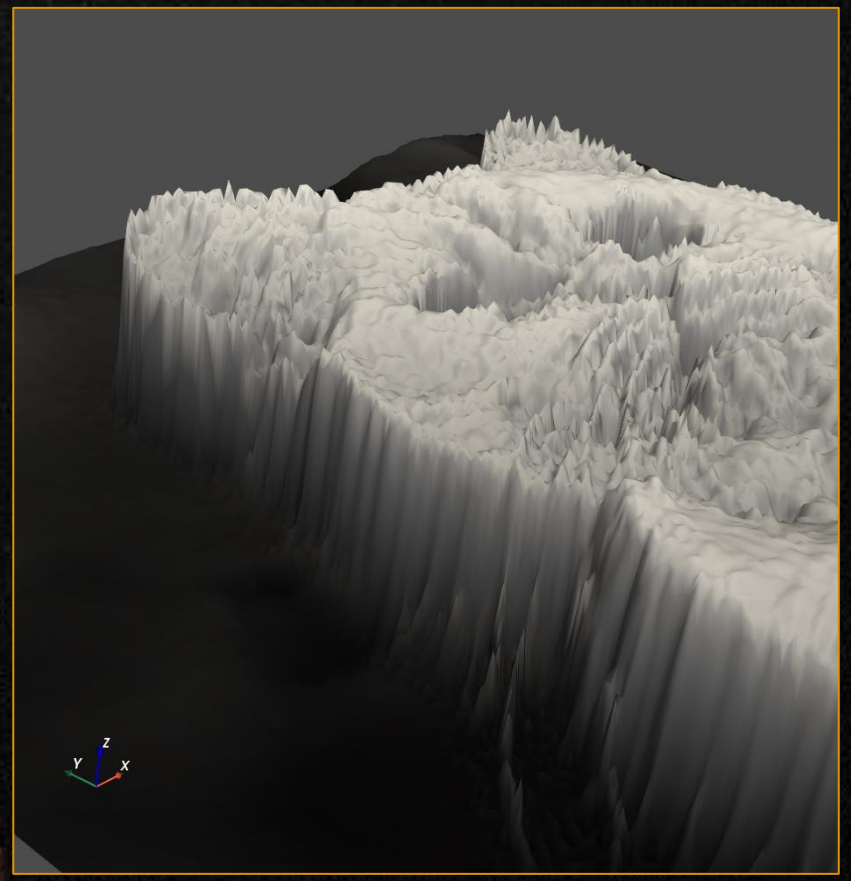
$\frac{\partial \mathbf{I}}{\partial x}$



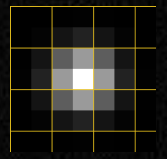
$\frac{\partial \mathbf{I}}{\partial y}$

Problema: rumore nelle immagini

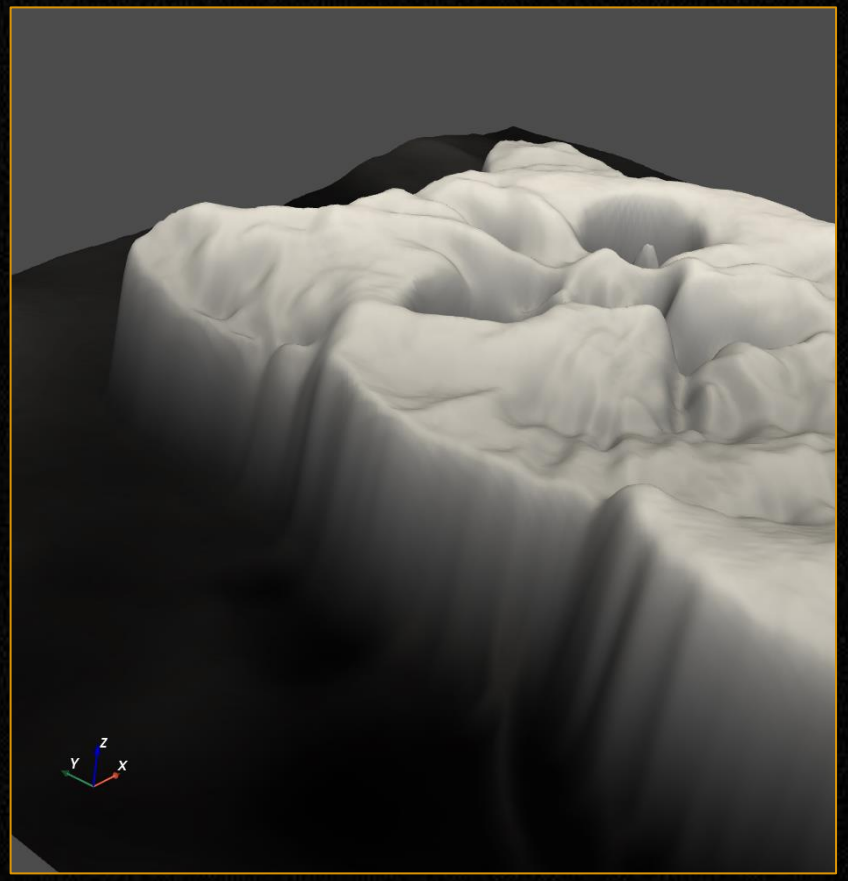
- La stima delle derivate parziali può essere influenzata da rumore nell'immagine
- Abbiamo già visto come è possibile applicare un filtro di smooth per ridurre il rumore



$\otimes$



$=$



Riduzione rumore e calcolo derivate parziali

- Prima di applicare il filtro derivativo $\mathbf{D}$, si applica un filtro di smooth $\mathbf{S}$:

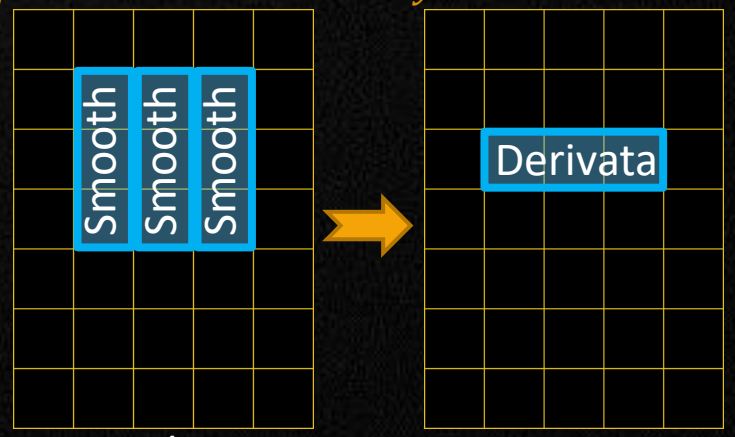
$$\mathbf{I}' = (\mathbf{I} * \mathbf{S}) * \mathbf{D}$$

- Applicando il filtro con la convoluzione e sfruttiamo la proprietà associativa: la convoluzione del filtro $\mathbf{D}$ con il filtro $\mathbf{S}$ permette di ottenere un singolo filtro che esegue entrambe le operazioni:

$$\mathbf{I}' = \mathbf{I} * \mathbf{F}, \text{con } \mathbf{F} = \mathbf{S} * \mathbf{D}$$

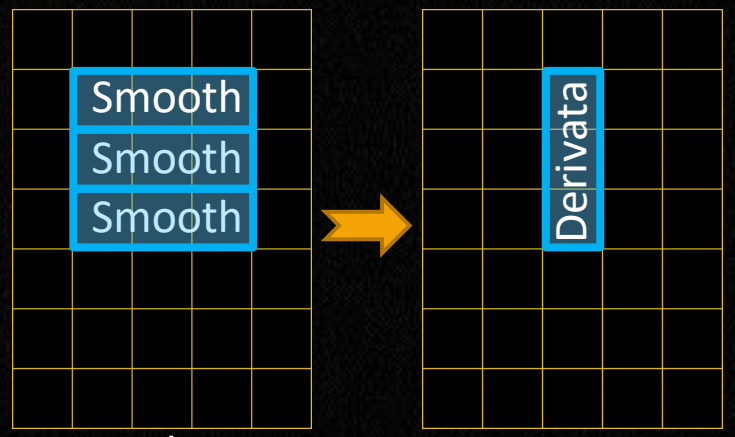
- Nei filtri più utilizzati in questo contesto, lo smooth avviene nella direzione ortogonale a quella di derivazione.

$$\frac{\partial \mathbf{I}}{\partial x} = \mathbf{I} * \mathbf{F}_x, \text{con } \mathbf{F}_x = \mathbf{S}_y * \mathbf{D}_x$$



Smooth su y Derivata x

$$\frac{\partial \mathbf{I}}{\partial y} = \mathbf{I} * \mathbf{F}_y, \text{con } \mathbf{F}_y = \mathbf{S}_x * \mathbf{D}_y$$



Smooth su x Derivata y

I principali filtri derivativi 3x3

$$\frac{\partial \mathbf{I}}{\partial x} = \mathbf{I} * \mathbf{F}_x, \text{con } \mathbf{F}_x = \mathbf{S}_y * \mathbf{D}_x = \mathbf{S}_y * \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix}$$

$$\frac{\partial \mathbf{I}}{\partial y} = \mathbf{I} * \mathbf{F}_y, \text{con } \mathbf{F}_y = \mathbf{S}_x * \mathbf{D}_y = \mathbf{S}_x * \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix}$$

Nome	$\mathbf{S}_y$	$\mathbf{S}_x$	$\mathbf{F}_x$	$\mathbf{F}_y$	Osservazioni
Prewitt	$\frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix}$	$\frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix}$	$\frac{1}{6}\begin{bmatrix}1 & 0 & -1\\1 & 0 & -1\\1 & 0 & -1\end{bmatrix}$	$\frac{1}{6}\begin{bmatrix}1 & 1 & 1\\0 & 0 & 0\\-1 & -1 & -1\end{bmatrix}$	Il più efficiente dei tre
Sobel	$\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix}$	$\frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}$	$\frac{1}{8}\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix}$	$\frac{1}{8}\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix}$	Oggi il più utilizzato nella pratica
Scharr	$\frac{1}{16}\begin{bmatrix}3\\10\\3\end{bmatrix}$	$\frac{1}{16}\begin{bmatrix}3 & 10 & 3\end{bmatrix}$	$\frac{1}{32}\begin{bmatrix}3 & 0 & -3\\10 & 0 & -10\\3 & 0 & -3\end{bmatrix}$	$\frac{1}{32}\begin{bmatrix}3 & 10 & 3\\0 & 0 & 0\\-3 & -10 & -3\end{bmatrix}$	Migliore invarianza rotazionale

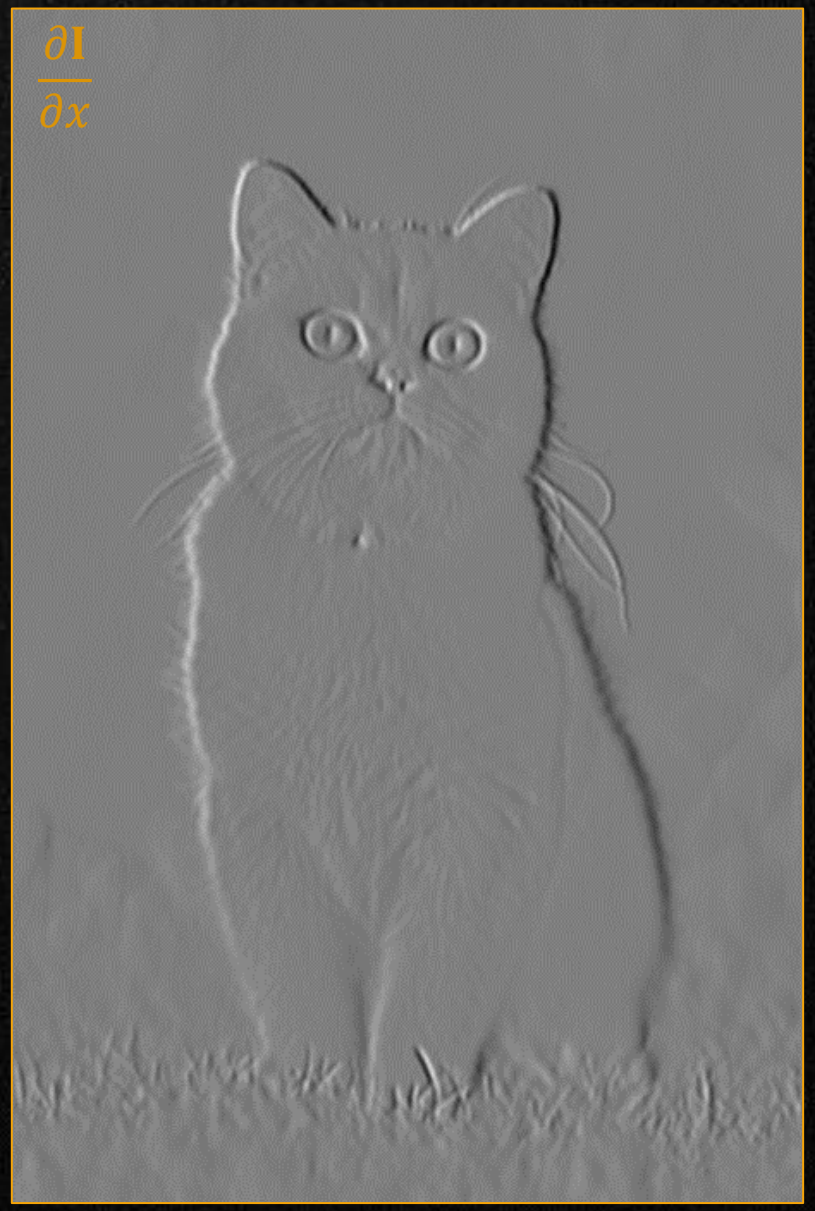
Note
- Sono stati ottenuti come filtri di convoluzione ($*$): per utilizzarli con la correlazione ($\otimes$) è sufficiente ribaltarli sugli assi
- Sono tutti filtri separabili (ovviamente!)

Creazione e applicazione di filtri derivativi in OpenCV


```
def derivata(img, t, d):
    s = {'Prewitt': [1,1,1], 'Sobel': [1,2,1], 'Scharr': [3,10,3]}[t]
    D = np.float32([-1,0,1])/2
    S = np.float32(s)/np.float32(s).sum()
    fx, fy = (D, S) if d=='x' else (S, D)
    return cv.sepFilter2D(img, cv.CV_32F, fx, fy)

img = cv.imread('esempi/cat.jpg', flags=cv.IMREAD_GRAYSCALE)
res = [(derivata(img,t,d), f'{t} ({d})')
       for t in ('Prewitt', 'Sobel', 'Scharr')
           for d in ('x', 'y')]
```

- Note:
  - Per applicare i filtri di Sobel e Scharr utilizzare cv.Sobel() e cv.Scharr()
  - Si possono ottenere i filtri di Sobel e Scharr con cv.getDerivKernels()
  - Infine cv.spatialGradient() restituisce entrambe le derivate parziali calcolate con i filtri di Sobel non normalizzati

Prewitt


$\frac{\partial \mathbf{I}}{\partial x}$

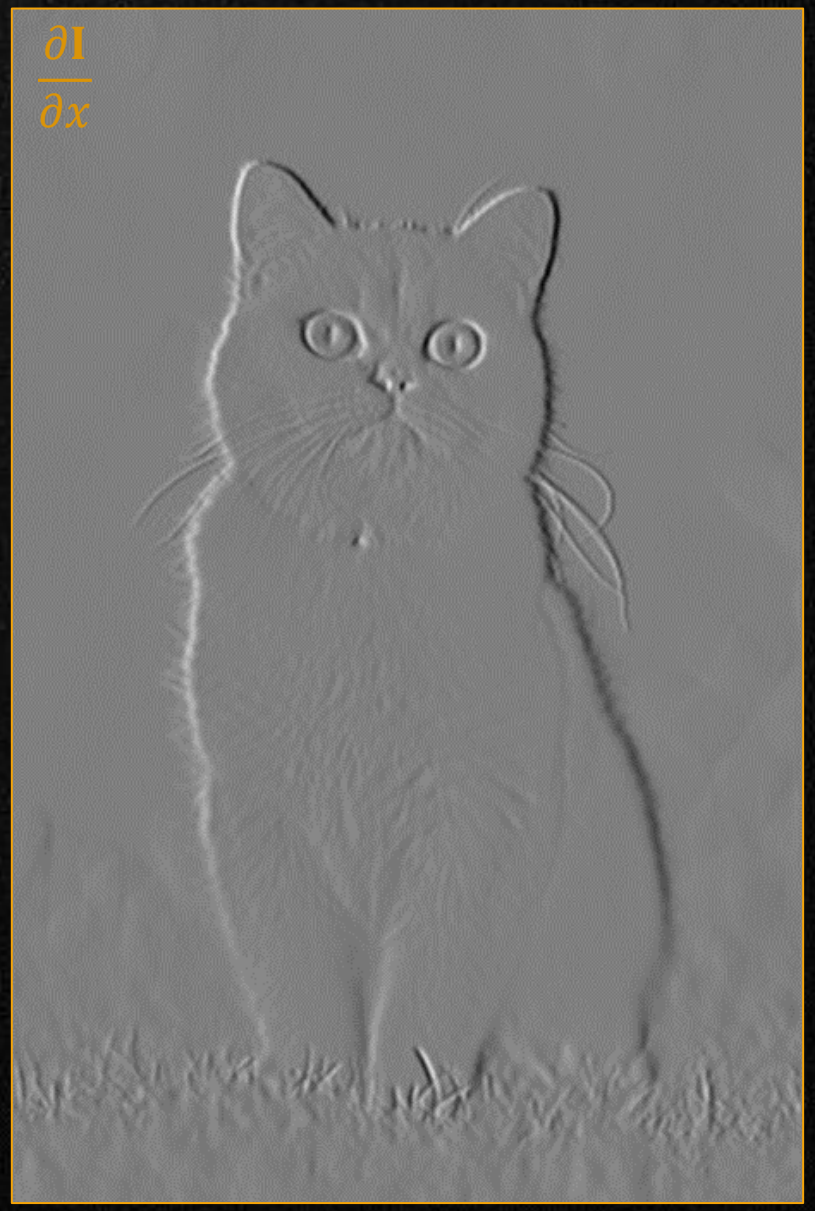
$\frac{\partial \mathbf{I}}{\partial y}$

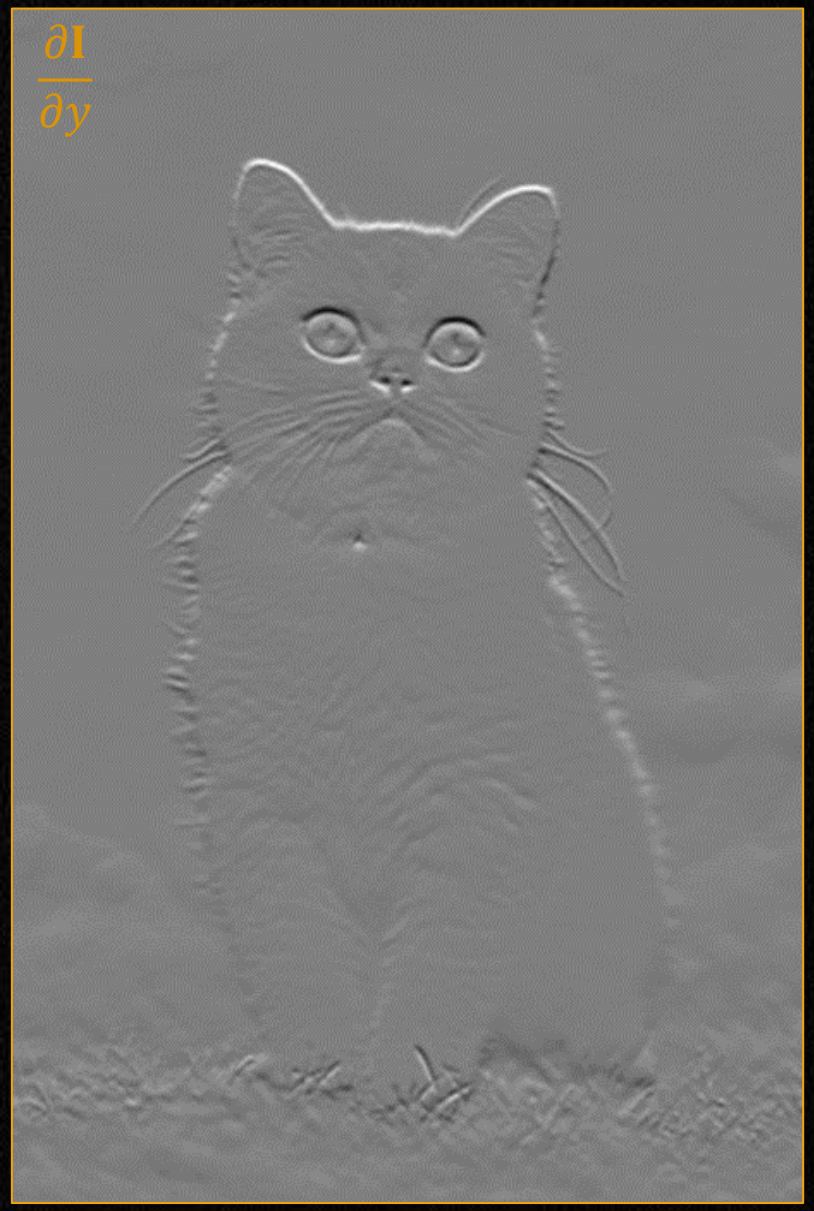
Sobel

$\frac{\partial \mathbf{I}}{\partial x}$

$\frac{\partial \mathbf{I}}{\partial y}$

Scharr


$\frac{\partial \mathbf{I}}{\partial x}$

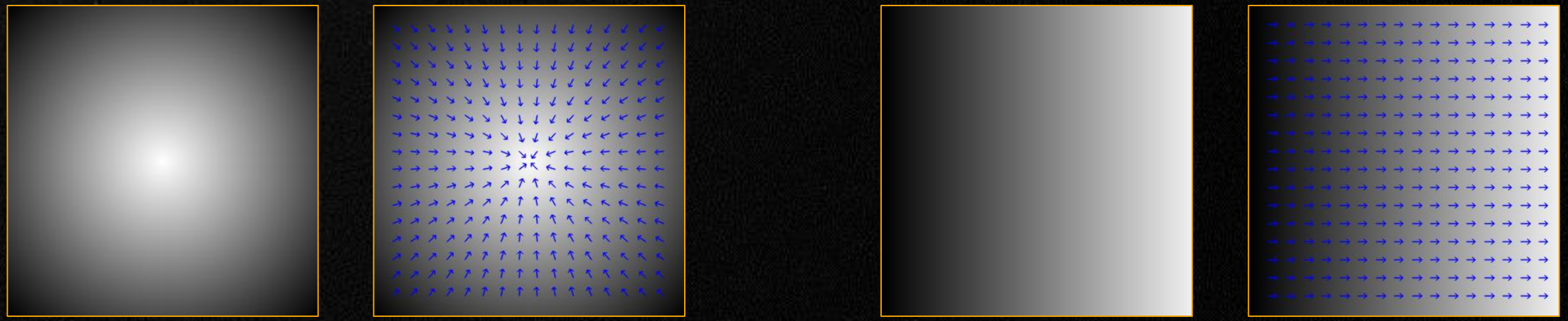

$\frac{\partial \mathbf{I}}{\partial y}$

Il gradiente

- Il gradiente di un'immagine in un punto (x, y) è il vettore che ha per componenti le due derivate parziali:

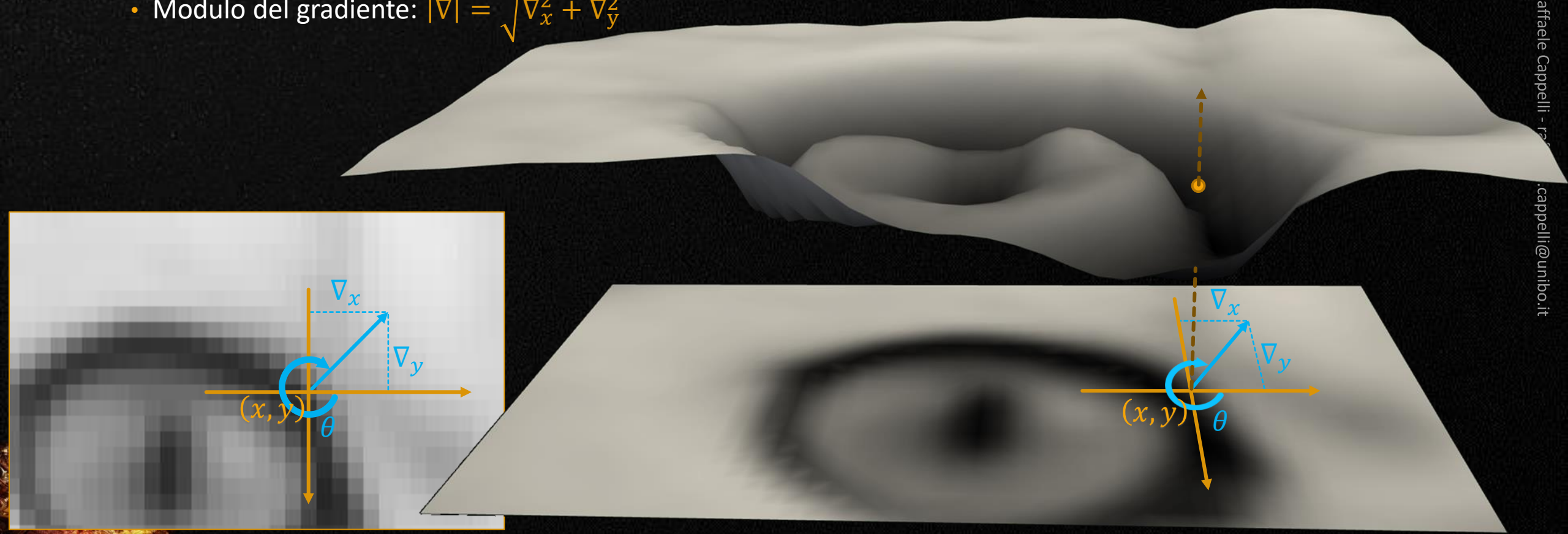
$$\nabla \mathbf{I}[y, x] = \left[\frac{\partial \mathbf{I}}{\partial x}[y, x], \frac{\partial \mathbf{I}}{\partial y}[y, x]\right]^{\top}$$

- Indica la direzione di maggior variazione dell'immagine nel punto (x, y).



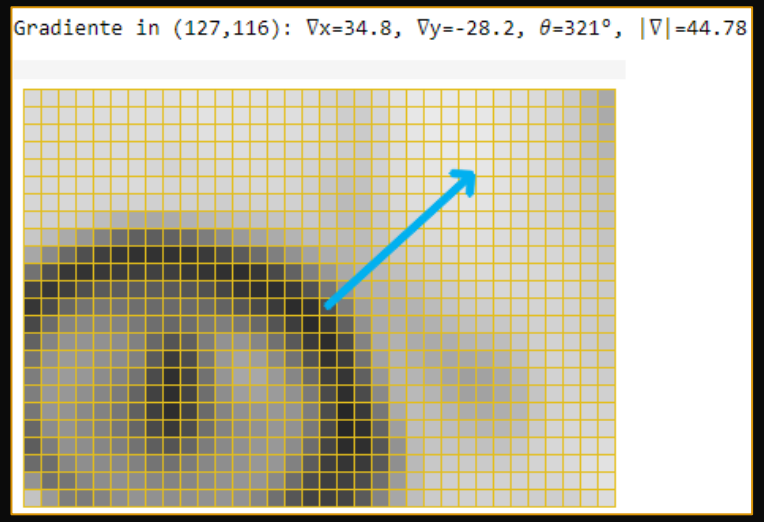
Modulo e orientazione del gradiente

- Sia $\nabla = [\nabla_x, \nabla_y]^\top$ il gradiente nel punto (x, y): $\nabla = \left[\frac{\partial \mathbf{I}}{\partial x}[y,x], \frac{\partial \mathbf{I}}{\partial y}[y,x]\right]^\top$
  - Angolo del gradiente: $\theta = atan2(\nabla_x, \nabla_y)$. Attenzione: misurato in senso orario (asse y verso il basso)
  - Modulo del gradiente: $|\nabla| = \sqrt{\nabla_x^2 + \nabla_y^2}$



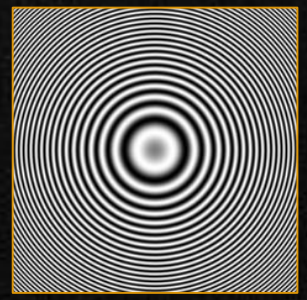
Orientazione e modulo del gradiente in OpenCV

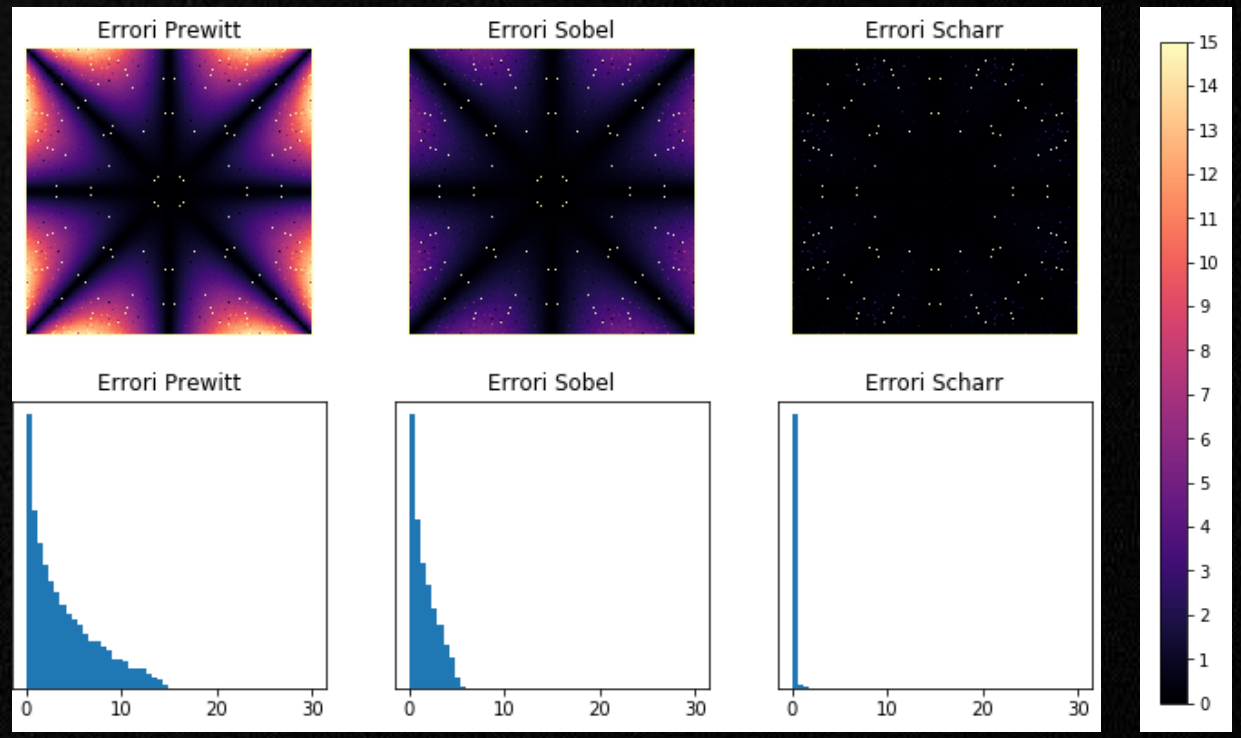

```
img = cv.imread('esempi/cat.jpg', flags=cv.IMREAD_GRAYSCALE)
img = cv.GaussianBlur(img, (3,3), 0)
# Calcolo derivate parziali per ogni pixel
dx,dy = cv.Sobel(img,cv.CV_32F,1,0,scale=1/8), cv.Sobel(img,cv.CV_32F,0,1,scale=1/8)
# Calcolo modulo e orientazione del gradiente per ogni pixel
mod = cv.magnitude(dx, dy)
ang = cv.phase(dx, dy, angleInDegrees=True)

# Disegna e stampa i dati del gradiente in un punto
x, y = 127, 116
w, h = 35, 25
scale, mult = 11, 1/4
roi = img[y-h//2:y+h//2, x-w//2:x+w//2]
# roi_large è l'immagine ingrandita dei pixel in roi
px, py = w//2, h//2
p1 = (px*scale+scale//2, py*scale+scale//2)
p2 = (int(round((px+dx[y,x]*mult)*scale)), int(round((py+dy[y,x]*mult)*scale)))
cv.arrowedLine(roi_large, p1, p2, (240,176,0), 3, cv.LINE_AA)
print(f'Gradiente in ({x},{y}): ∇x={dx[y,x]:.1f}, ∇y={dy[y,x]:.1f}, ' +
      f'θ={ang[y,x]:.0f}°, |∇|={mod[y,x]:.2f}')
```



Precisione nella stima dell'angolo del gradiente

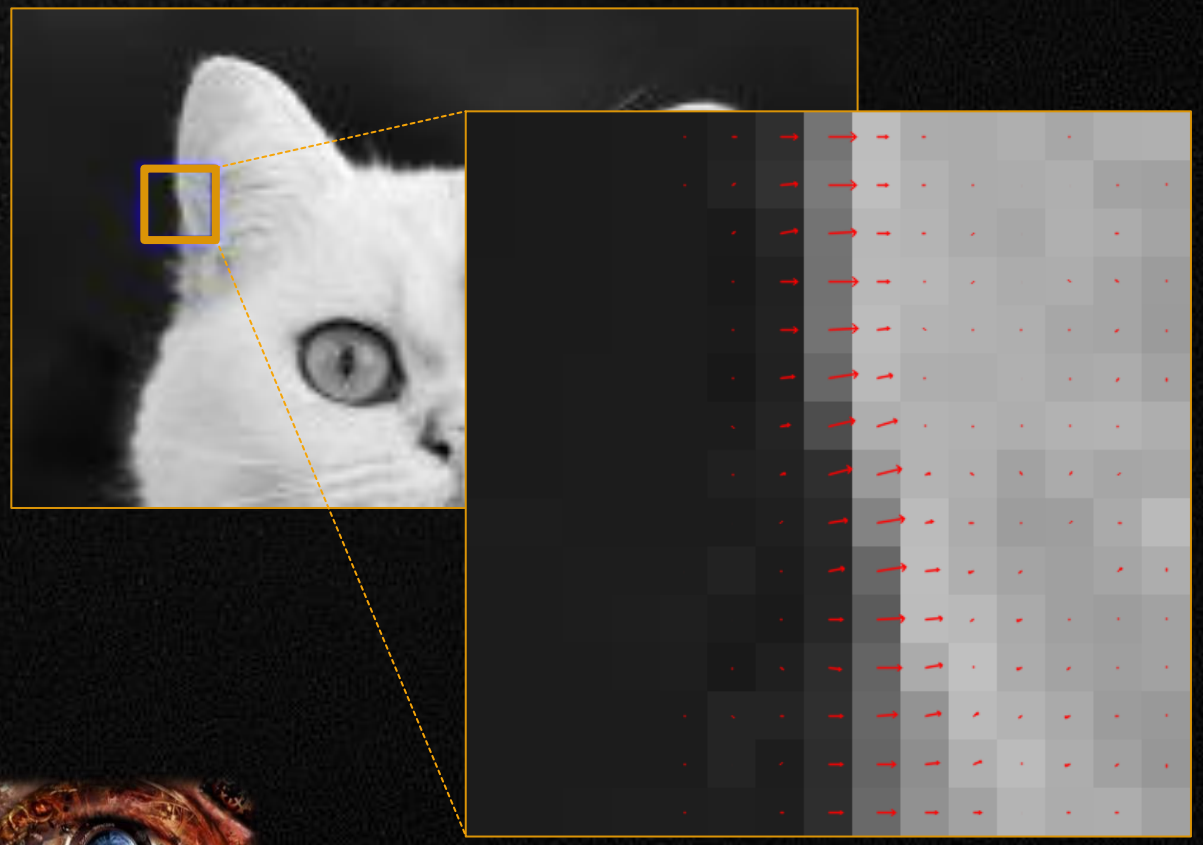
- Consideriamo l'immagine data dalla funzione $f(x, y) = \sin(x^2 + y^2)$
- Calcoliamo l'angolo del gradiente dalle derivate parziali ottenute con Prewitt, Sobel e Scharr
- Misuriamo l'errore in gradi, per ogni pixel, rispetto al gradiente ottenuto dalle derivate analitiche della funzione: $\frac{\partial f}{\partial x}(x, y) = 2x \cdot cos(x^2 + y^2), \frac{\partial f}{\partial y}(x, y) = 2y \cdot cos(x^2 + y^2)$

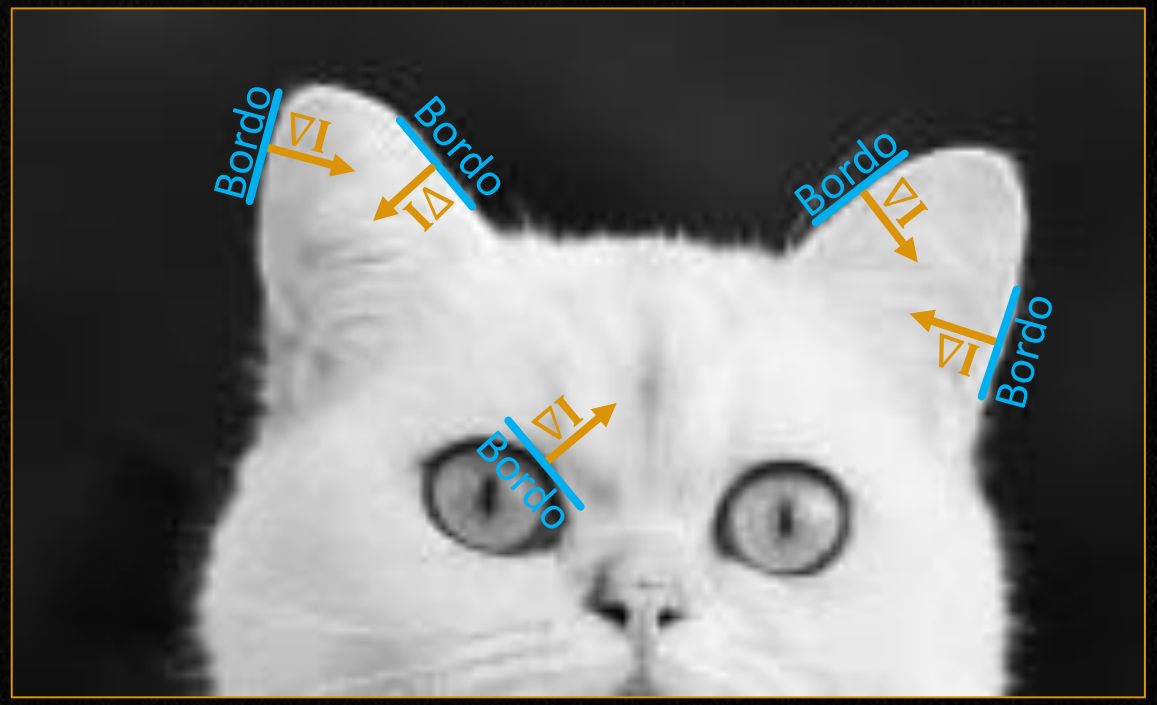




Orientazione del gradiente e bordi

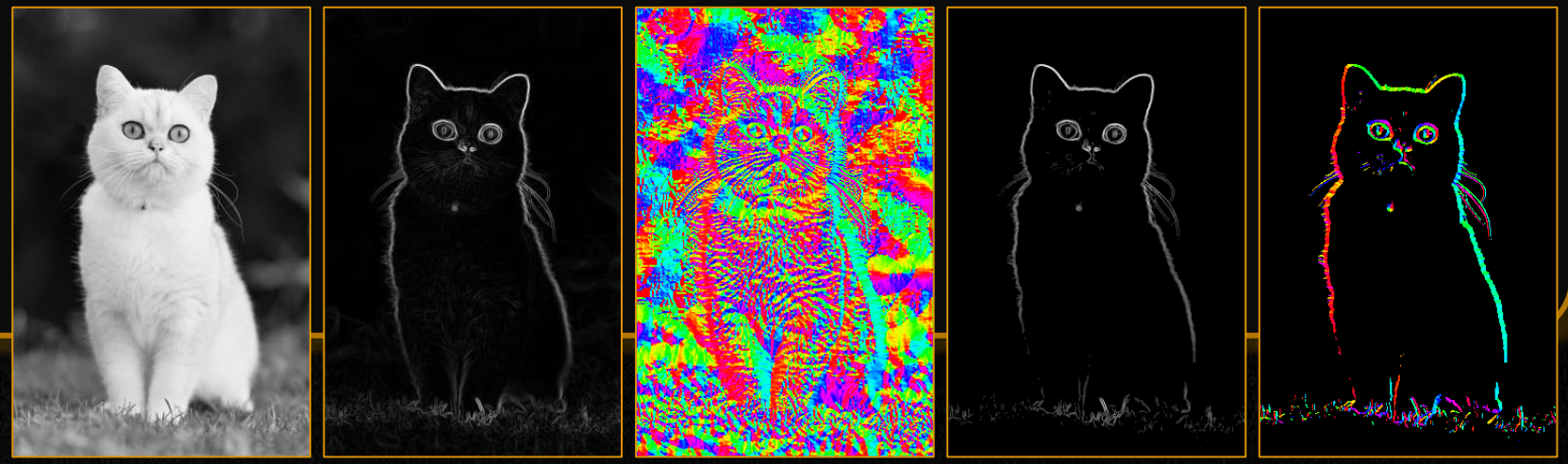
- Analizzando l'orientazione del gradiente, si può osservare che il bordo di un'immagine è ortogonale al gradiente.





Visualizzare modulo e angolo del gradiente in OpenCV


```
img = cv.imread('esempi/cat.jpg', flags=cv.IMREAD_GRAYSCALE)
dx, dy = cv.spatialGradient(img)
dx, dy = dx.astype(np.float32), dy.astype(np.float32)
# Calcola il modulo del gradiente per ciascun pixel
m = cv.magnitude(dx, dy)
# Calcola l'angolo del gradiente per ciascun pixel (in radianti)
a = cv.phase(dx, dy)
# Converte i moduli nel range [0,255]
m = cv.normalize(m, None, 0, 255, cv.NORM_MINMAX, cv.CV_8U)
# Converte gli angoli da [0,2pi] a [0,255]
a = (a/(2*np.pi)*255).round().astype(np.uint8)
a = cv.applyColorMap(a, cv.COLORMAP_HSV) # Angolo -> Hue
# Visualizza solo i gradienti con modulo superiore a una soglia
t = 40
a1 = a.copy(); a1[m<t] = 0
m1 = m.copy(); m1[m<t] = 0
```



Individuare i bordi con il modulo del gradiente?

- Il modulo del gradiente indica quanto è "importante" il corrispondente bordo:
  - Si potrebbe pensare di applicare una soglia al modulo del gradiente per selezionare i bordi, scartando così i "falsi positivi" dovuti a rumore o altro.
- Nella pratica questo approccio è difficilmente applicabile, in quanto produce:
  - frammenti di bordi non connessi
  - bordi spuri (dovuti al rumore)
  - bordi più spessi di un pixel

Immagine di partenza

Modulo del gradiente

Risultato ottenuto

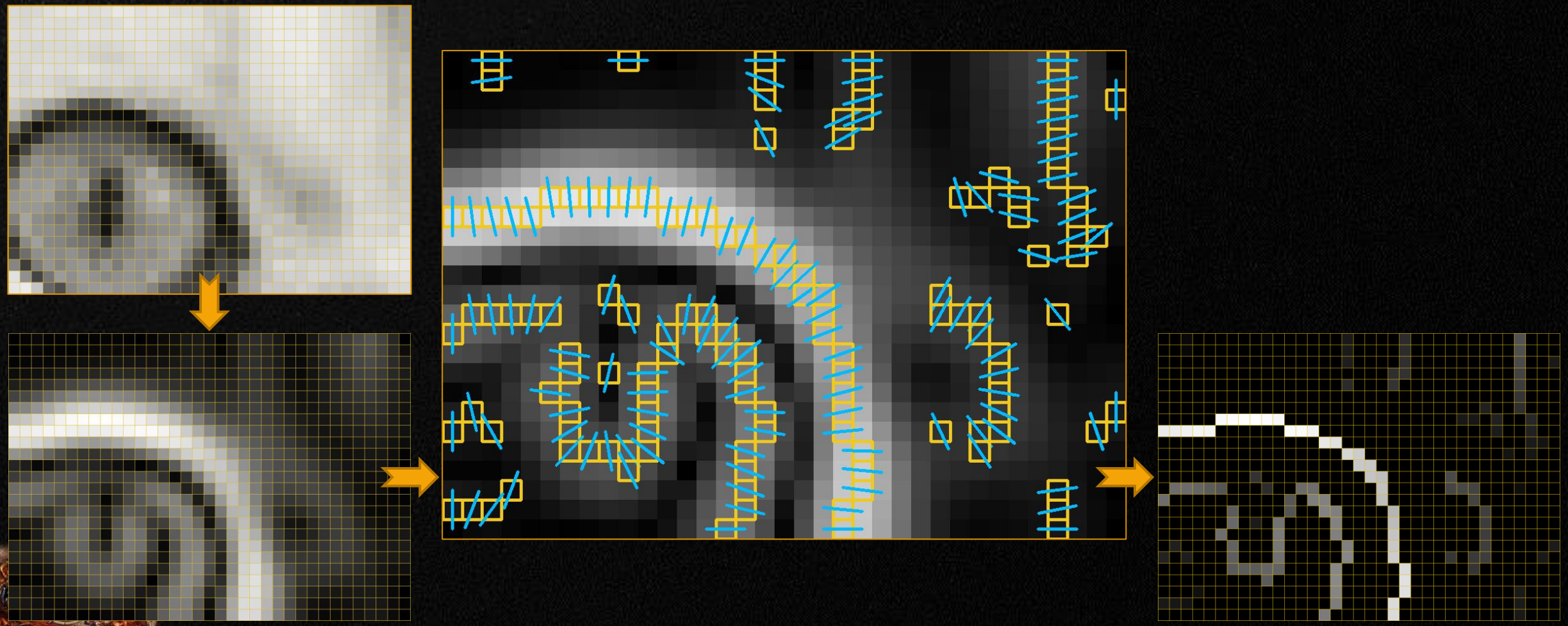
Risultato desiderato

Canny edge detector

- Algoritmo proposto da John F. Canny nel 1986. Consiste in quattro step:
  1. Smooth gaussiano dell'immagine
  2. Calcolo del gradiente per ogni pixel
  3. Soppressione dei non-massimi in direzione ortogonale al bordo
  4. Selezione dei bordi significativi mediante isteresi

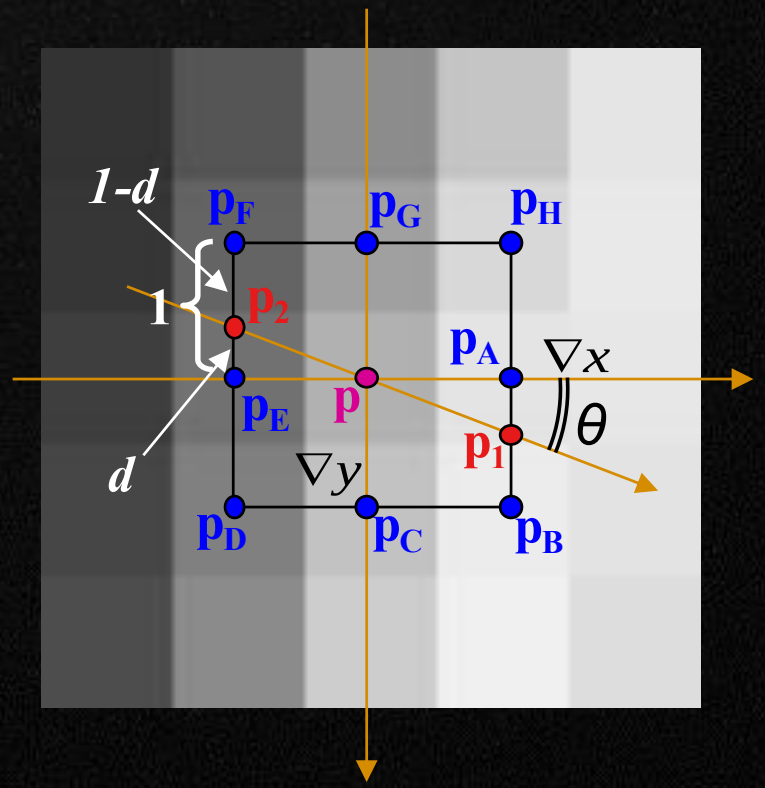
Canny edge detector: soppressione dei non-massimi

- Obiettivo: eliminare dall'immagine del modulo del gradiente tutti i pixel in cui il modulo del gradiente non è massimo locale rispetto all'orientazione del gradiente.



Canny edge detector: soppressione dei non-massimi

- Come verificare la condizione di massimo locale nell'intorno 3x3:
  - Si stima il modulo del gradiente nei punti $\mathbf{p}_1$ e $\mathbf{p}_2$ mediante interpolazione lineare
  - La figura mostra l'interpolazione nel caso l'orientazione del gradiente appartenga al primo ottante; gli altri casi sono analoghi
  - Il pixel **p** viene conservato solo se $\|\nabla[\mathbf{p}]\| \geq \|\nabla[\mathbf{p}_1]\| \;\wedge\; \|\nabla[\mathbf{p}]\| \geq \|\nabla[\mathbf{p}_2]\|$



$$\|\nabla[\mathbf{p}_1]\| \cong d \cdot \|\nabla[\mathbf{p}_\mathbf{B}]\| + (1-d)\|\nabla[\mathbf{p}_\mathbf{A}]\|$$

$$\|\nabla[\mathbf{p}_2]\| \cong d \cdot \|\nabla[\mathbf{p}_\mathbf{F}]\| + (1-d)\|\nabla[\mathbf{p}_\mathbf{E}]\|$$

$$d = \frac{\nabla y[\mathbf{p}]}{\nabla x[\mathbf{p}]}$$

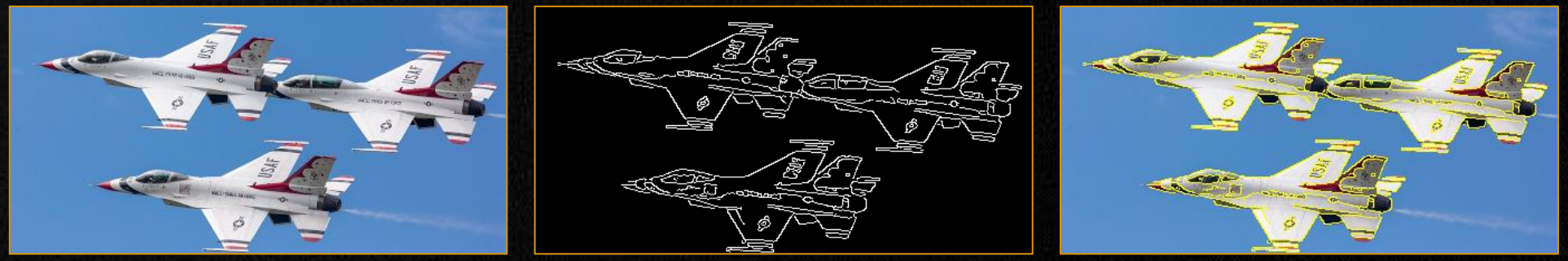
Canny edge detector: selezione finale dei bordi mediante isteresi

- Al fine di selezionare solo gli edge significativi, cercando allo stesso tempo di ottenere bordi continui, l'immagine ottenuta al passo precedente viene binarizzata utilizzando due soglie T1 e T2, con T1>T2:
  - sono inizialmente considerati validi solo i pixel in cui il modulo del gradiente è superiore a T1;
  - i pixel il cui modulo è inferiore a T1 ma superiore a T2 sono considerati validi solo se adiacenti a pixel validi.
- Una corretta scelta di T1 e T2, così come un'adeguata scelta dei parametri dello smooth gaussiano nella prima fase, sono molto importanti per ottenere gli effetti desiderati.
- La scelta dipende solitamente dall'applicazione e dalle caratteristiche dell'immagine; sono tipicamente necessari vari esperimenti per giungere ai valori ottimali dei parametri.

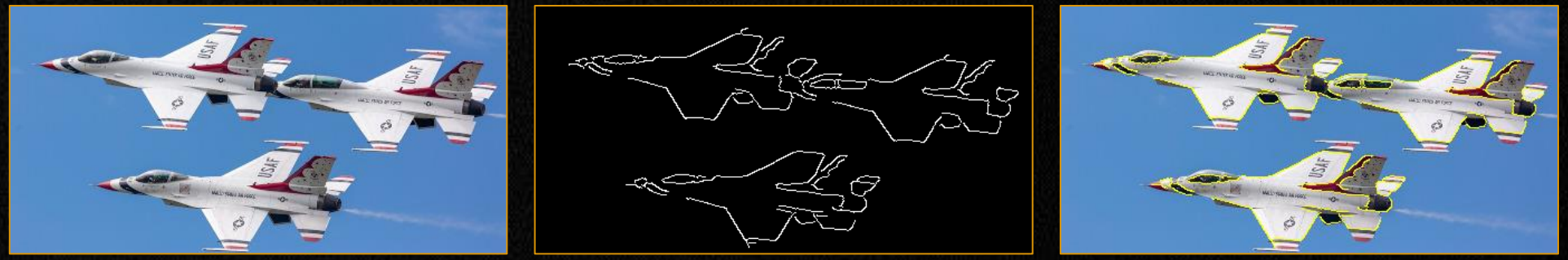
Canny edge detector in OpenCV


```
# Caricamento immagine
img = cv.imread('filtri/thunderbirds.jpg')
# Smooth gaussiano con filtro sxs e sigma calcolato da OpenCV
blurred = cv.GaussianBlur(cv.cvtColor(img, cv.COLOR_BGR2GRAY), (s, s), 0)
# Algoritmo di Canny con soglie t1 e t2
edges = cv.Canny(blurred, t1, t2)
img_e = img.copy()
img_e[edges!=0] = (0,255,255)
```

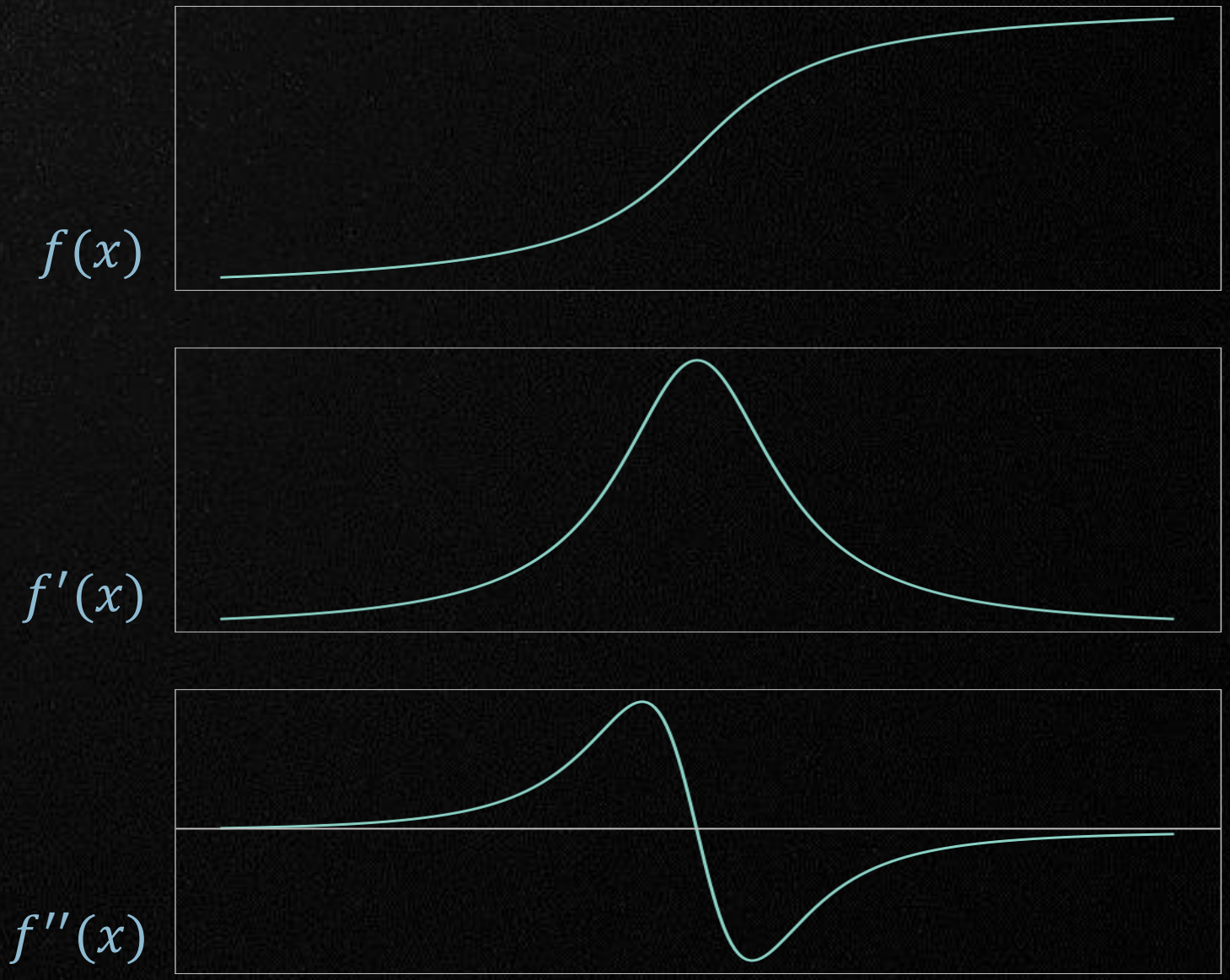
$s = 3, t_1 = 200, t_2 = 50$



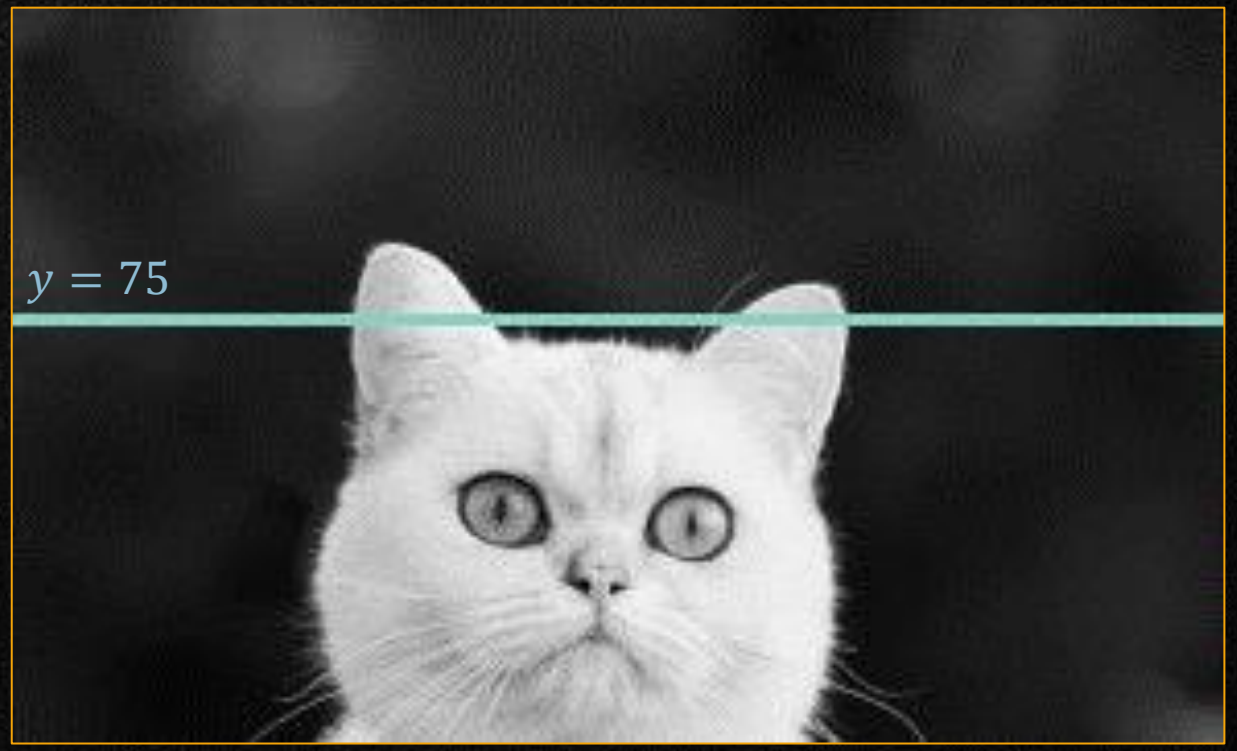
$s = 9, t_1 = 200, t_2 = 100$

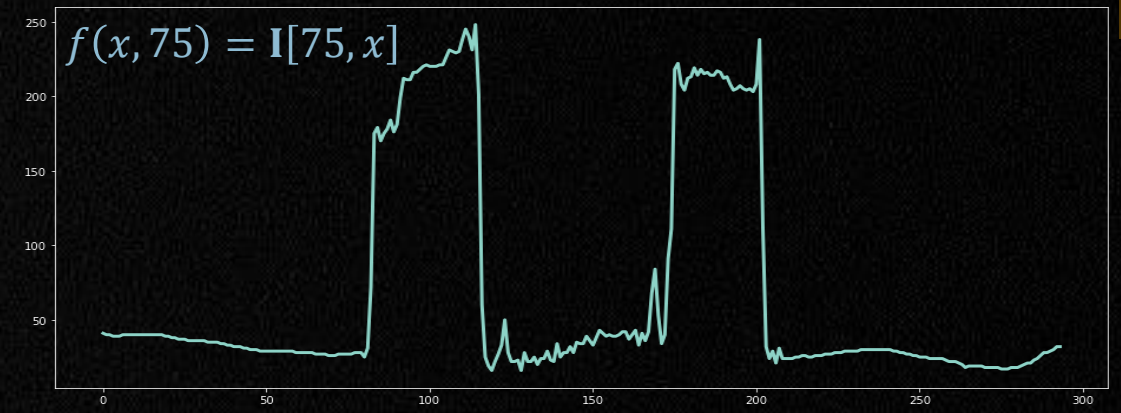


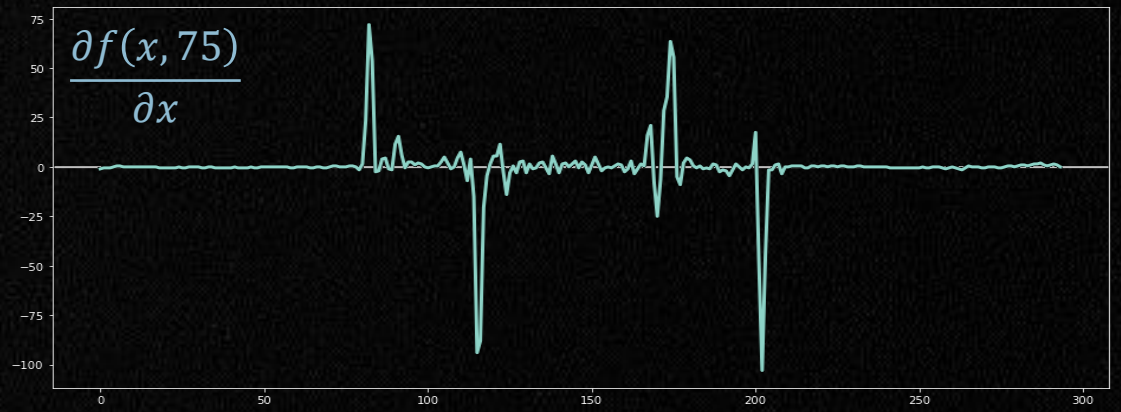
Utilizzo della derivata seconda

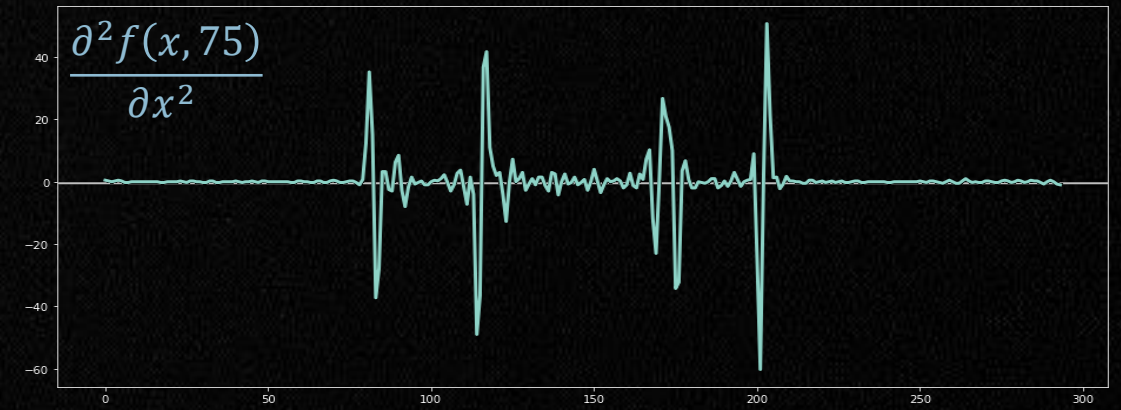


Seconda derivata parziale su un'immagine









Derivata seconda di una funzione discreta

- Stima a partire da $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h} \approx f\left(x+\frac{1}{2}\right) - f\left(x-\frac{1}{2}\right)$

$$f''(x) \approx f'\left(x+\frac{1}{2}\right) - f'\left(x-\frac{1}{2}\right)$$

$$f''(x) \approx \left(f\left(\left(x+\frac{1}{2}\right)+\frac{1}{2}\right) - f\left(\left(x+\frac{1}{2}\right)-\frac{1}{2}\right)\right) - \left(f\left(\left(x-\frac{1}{2}\right)+\frac{1}{2}\right) - f\left(\left(x-\frac{1}{2}\right)-\frac{1}{2}\right)\right)$$

$$f''(x) \approx f(x+1) - f(x) - f(x) + f(x-1) = f(x+1) - 2f(x) + f(x-1)$$

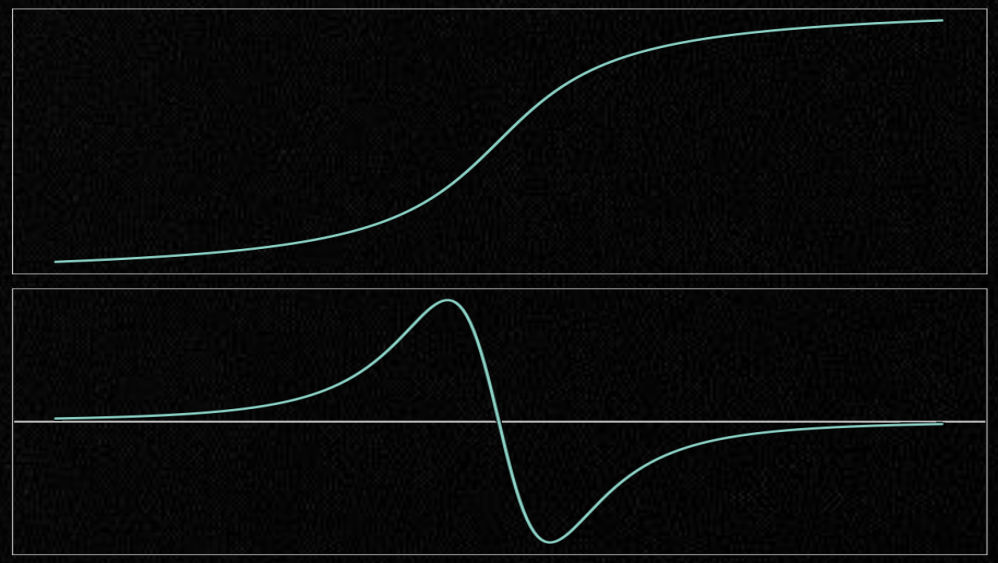
$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx f(x+1,y) - 2f(x,y) + f(x-1,y) \Rightarrow \mathbf{F}_x^2 = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx f(x,y+1) - 2f(x,y) + f(x,y+1) \Rightarrow \mathbf{F}_y^2 = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

Laplaciano

$$\Delta f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

- Il Laplaciano è definito come la somma delle due derivate parziali seconde in un punto.
- In un'immagine $\mathbf{I}$, in presenza di un bordo sufficientemente contrastato, il laplaciano $\Delta\mathbf{I}$ è:
  - zero lontano dal bordo;
  - positivo vicino al bordo (dal lato dove pixel più scuri diventano più chiari muovendosi verso il bordo);
  - negativo vicino al bordo (dal lato opposto) ;
  - zero in corrispondenza del bordo.



Costruiamo un filtro per calcolare il laplaciano

- Applicando la proprietà distributiva della convoluzione rispetto all'addizione:

$$\Delta \mathbf{I} = \frac{\partial^2 \mathbf{I}}{\partial x^2} + \frac{\partial^2 \mathbf{I}}{\partial y^2} = (\mathbf{I} * \mathbf{F}_x^2) + \left(\mathbf{I} * \mathbf{F}_y^2\right) = \mathbf{I} * \left(\mathbf{F}_x^2 + \mathbf{F}_y^2\right) = \mathbf{I} * \mathbf{F}_\Delta$$

$$\underbrace{\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}}_{\mathbf{F}_x^2} + \underbrace{\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}}_{\mathbf{F}_y^2} = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\mathbf{F}_\Delta}$$

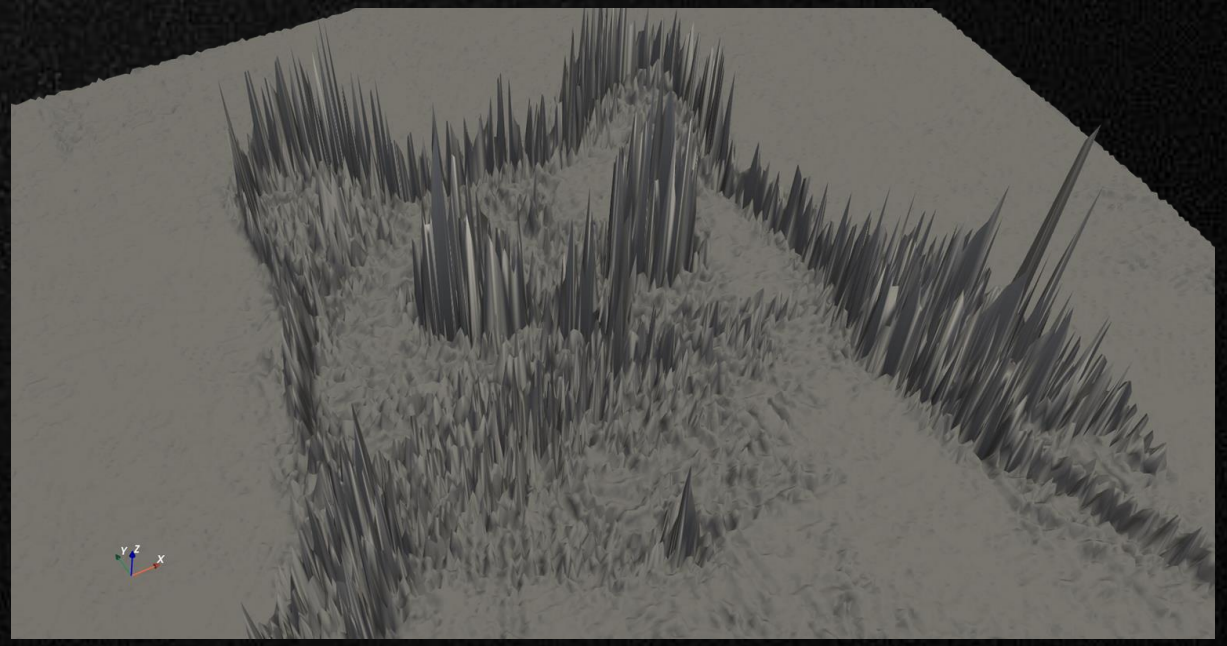
$\mathbf{I}$

$$* \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} =$$

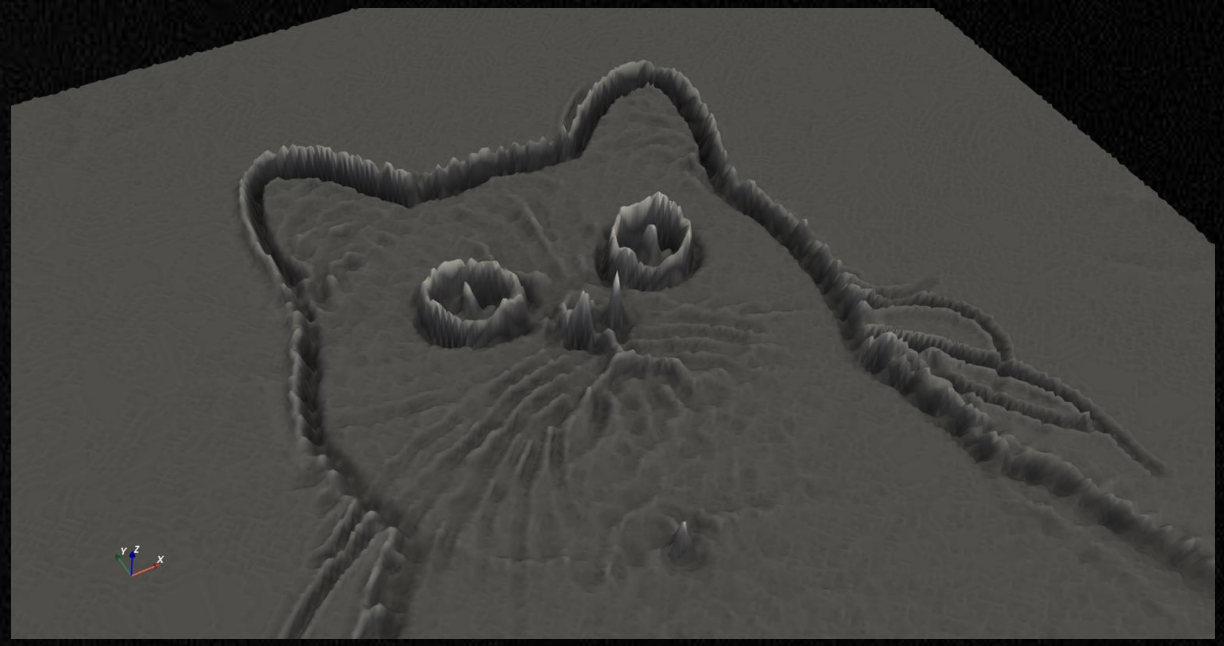
$\mathbf{F}_\Delta$

$\Delta\mathbf{I}$

Il laplaciano è molto sensibile al rumore: necessario uno smooth



Laplaciano sull'immagine originale



Laplaciano dopo aver applicato uno smooth gaussiano

Laplaciano con smooth gaussiano: Laplacian of Gaussian (LoG)

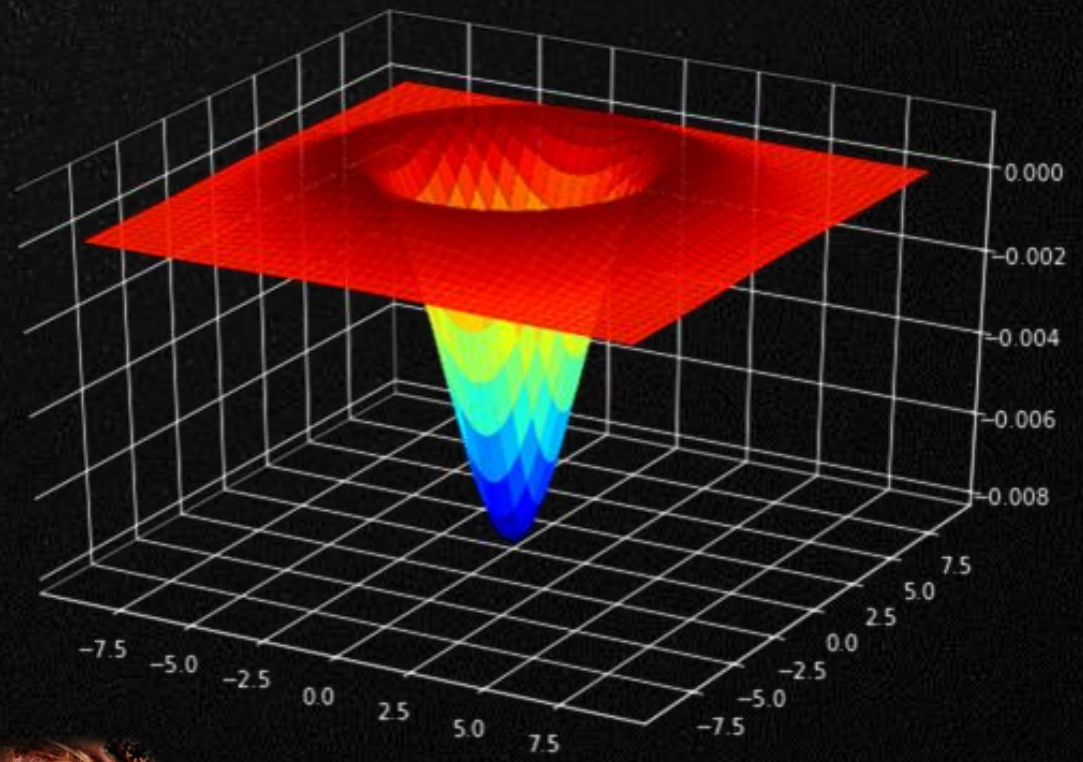
- Il laplaciano di un'immagine a cui è stato applicato uno smooth gaussiano si può ottenere mediante convoluzione con un filtro gaussiano $\mathbf{S}_G$ seguita da convoluzione con il filtro laplaciano $\mathbf{F}_\Delta$: $\Delta(\mathbf{I} * \mathbf{S}_G) = (\mathbf{I} * \mathbf{S}_G) * \mathbf{F}_\Delta$
- Per la proprietà associativa della convoluzione: $(\mathbf{I} * \mathbf{S}_G) * \mathbf{F}_\Delta = \mathbf{I} * (\mathbf{S}_G * \mathbf{F}_\Delta) = \mathbf{I} * \mathbf{LoG}$
- Il filtro $\mathbf{LoG}$ esegue quindi entrambe le operazioni e si può ottenere calcolando il laplaciano della funzione gaussiana $G_{2D}(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$:

$$\Delta\big(G_{2D}(x,y)\big) = \frac{\partial^2 G_{2D}(x,y)}{\partial x^2} + \frac{\partial^2 G_{2D}(x,y)}{\partial y^2} = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) G_{2D}(x,y) + \left(\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2}\right) G_{2D}(x,y)$$

$$\Delta\big(G_{2D}(x,y)\big) = \left(\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2}\right) G_{2D}(x,y)$$

Laplacian of Gaussian (LoG)

- $\Delta\big(G_{2D}(x,y)\big) = \left(\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2}\right) G_{2D}(x,y)$
- Può essere molto simile, ma non identico, al filtro DoG (per opportuni valori di σ_1 e σ_2)

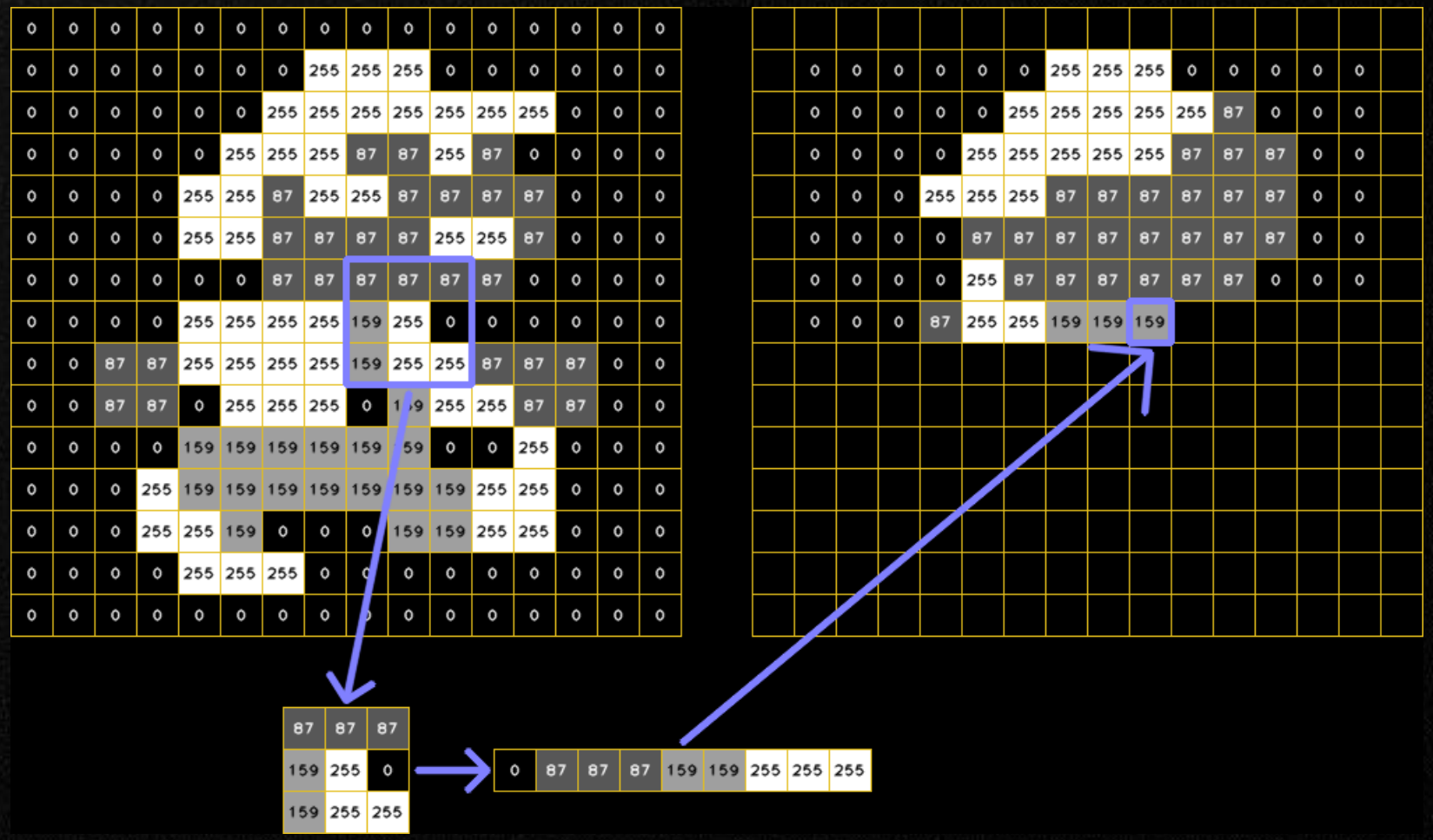


1	2	3	4	3	2	1
2	4	2	0	2	4	2
3	2	−8	−16	−8	2	3
4	0	−16	−27	−16	0	4
3	2	−8	−16	−8	2	3
2	4	2	0	2	4	2
1	2	3	4	3	2	1

Un esempio di LoG discretizzato su griglia 7x7

Un esempio di filtro non lineare: median filter

- Operazione locale in cui il valore di ciascun pixel è la mediana dei valori dei pixel nell'intorno considerato: $\mathbf{I}'[y,x] = \mathrm{median}(w_{xy})$. Può ridurre alcuni tipi di rumore *preservando maggiormente i bordi* rispetto ai filtri di smooth lineari.



Median filter

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0
0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0
0	0	0	0	0	255	255	255	87	87	255	87	0	0	0	0
0	0	0	0	255	255	87	255	255	87	87	87	87	0	0	0
0	0	0	0	255	255	87	87	87	87	255	255	87	0	0	0
0	0	0	0	0	0	87	87	87	87	87	87	0	0	0	0
0	0	0	0	255	255	255	255	159	255	0	0	0	0	0	0
0	0	87	87	255	255	255	255	159	255	255	87	87	87	0	0
0	0	87	87	0	255	255	255	0	159	255	255	87	87	0	0
0	0	0	0	159	159	159	159	159	159	0	0	255	0	0	0
0	0	0	255	159	159	159	159	159	159	159	255	255	0	0	0
0	0	0	255	255	159	0	0	0	159	159	255	255	0	0	0
0	0	0	0	255	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0
0	0	0
0	0	0

0	0	0	0	0	0	0	0	0

	0														

Median filter in OpenCV


```
# Carica un'immagine con rumore nello sfondo (alcuni piccoli punti più chiari)
img = cv.imread('esempi/bolts.png', cv.IMREAD_GRAYSCALE)

# Applica un filtro gaussiano della dimensione minima per poter eliminare il rumore
noise_reduction_gaussian_filter = cv.GaussianBlur(img, (25,25), 0)

# Applica un filtro mediano della dimensione minima per poter eliminare il rumore
noise_reduction_median_filter = cv.medianBlur(img, 9)
```

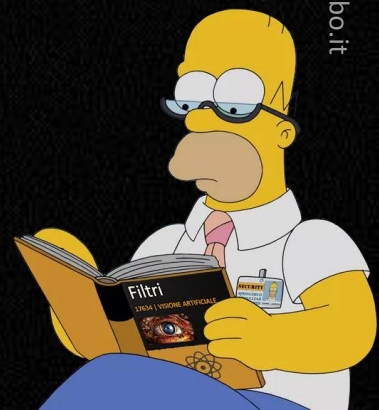
Immagine originale

Filtro gaussiano

Filtro mediano

Riepilogo

- Filtri lineari
  - Gestione dei bordi, tipi di dati, normalizzazione, implementazione
  - Filtri separabili, complessità computazionale, implementazione
  - Correlazione e convoluzione: proprietà comuni e differenze
- Riduzione del rumore
  - Box filter e Gaussian filter
- Aumento del contrasto locale
  - Filtro di sharpening: costruzione del filtro applicando le proprietà algebriche della convoluzione
- Estrazione dei bordi
  - Difference of Gaussians
  - Filtri derivativi e loro implementazione, modulo e orientazione del gradiente
  - Canny edge detector
  - Derivata seconda, Laplaciano e Laplacian of Gaussian
- Filtri non lineari: median filter



Per approfondire

- Nel libro [Szeliski, Computer Vision: Algorithms and Applications, 2022]:
  - Sezione 3.2: Linear filtering
  - Sezione 7.2.1: Edge detection

- Documentazione OpenCV del modulo «Image processing - Image Filtering»:
  - https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html