

II - Operazioni locali e convoluzione

Abbiamo visto precedentemente le operazioni che cambiano il contenuto del pixel, vediamo ora le operazioni locali che si occupano invece di un intero intorno di esso.

Dove le operazioni puntuali erano del tipo $g(x, y) = f(x, y)$

Al fine di applicare queste operazioni sull'intorno del pixel sarà quindi necessaria introdurre la **convoluzione**:

Convoluzione

Partendo da un vettore di dimensione n esso può essere scomposto attraverso la **base canonica**, il vettore sempre di dimensione n dove a giro avremo una caratteristica del tipo:

$$2[1, 0, 0] \ 3[0, 1, 0] \ 4[0, 0, 1]$$

Ciò si applica non solo ai vettori ma specialmente **alle matrici**, chiaramente da una matrice $m \times n$ avremo anche una base canonica di dimensione $m \times n$

3	5	7
9	1	3
2	1	5

$$\begin{aligned} &= 3 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 5 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 7 \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ &\quad + 9 \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 1 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 3 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\ &\quad + 2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} + 1 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} + 5 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

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

Consideriamo quindi la formula di linearità di una funzione

$$af(x_1, y_1) + bf(x_2, y_2) = f(ax_1 + bx_2, ay_1 + by_2)$$

nelle slide il calcolo per verificare se una funzione data è lineare

Dobbiamo verificare la linearità della funzione:

Un esempio di funzione non lineare è: $f(x, y) = (255 - x, 255 - y)$ non è lineare perché i due membri alla fine del calcolo non sono uguali difatti:

$$(255 - (ax_1 + bx_2), 255 - (ay_1 + by_2)) = (255 - ax_1 - bx_2, 255 - ay_1 - by_2))$$

Per la convoluzione è essenziale che la funzione sia lineare, più avanti vedremo come anche le funzioni non lineari possono essere utili ma non in questo caso.

Il terzo tassello consiste nel trovare un operatore che è invariante per traslazione:

Operatore invariante per traslazione

Abbiamo visto che l'operatore negativo non è un operatore lineare ma possiamo affermare che è invariante per traslazione (shift invariant)

Dunque è sempre il medesimo indipendentemente dalla posizione del pixel, è rilevante dunque solo il valore del pixel e non la sua posizione

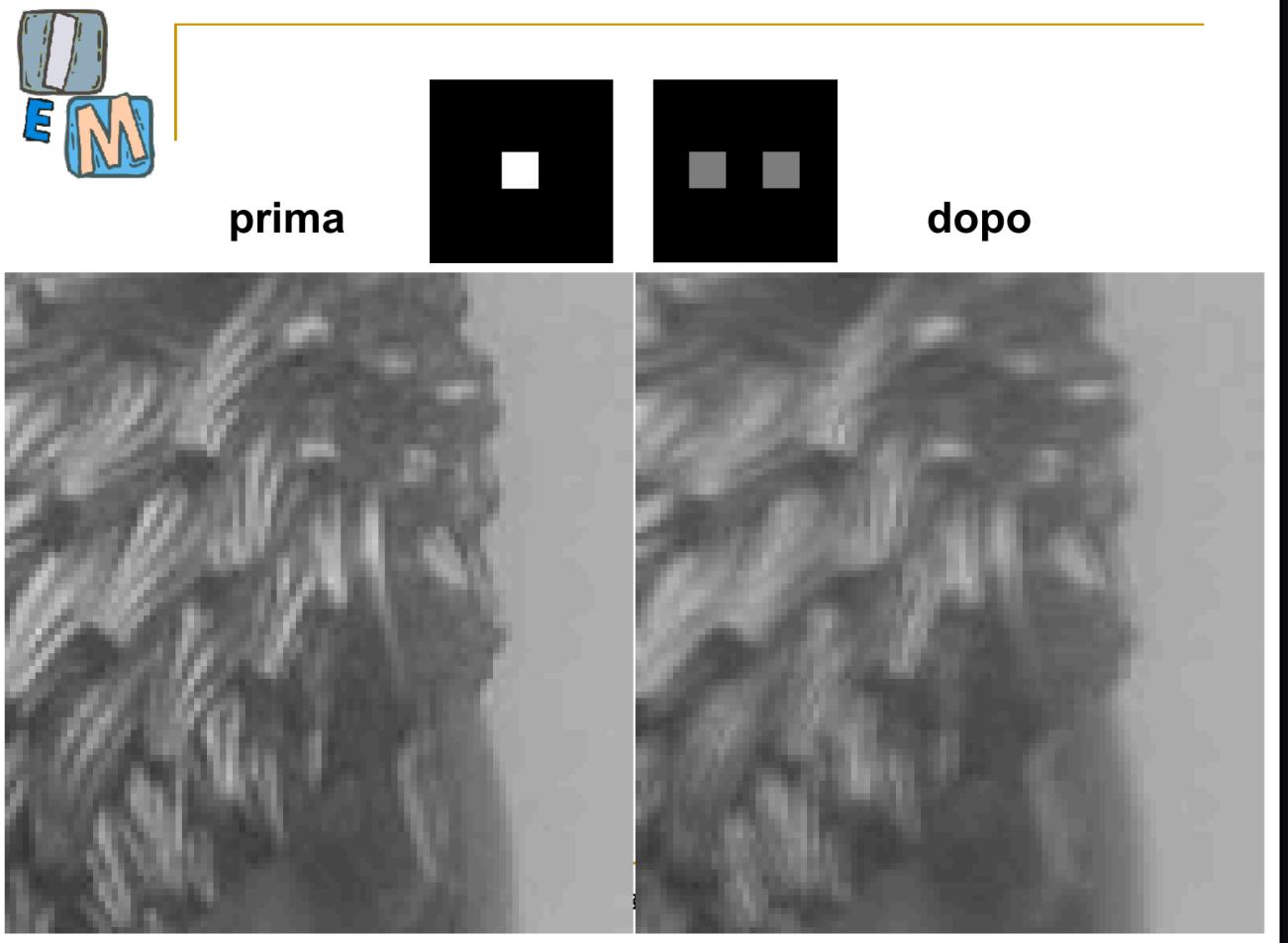
Introduciamo anche il concetto di kernel (nucleo) esso è necessariamente una matrice quadrata con dimensione dispari dato che è assolutamente necessario trovarne il centro

Il kernel della funzione sarà il risultato di aver applicato l'operatore alla base canonica con l'1 al centro della matrice (dall'immagine):

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & \textcolor{brown}{1} & 1 \\ 0 & 1 & 0 \end{bmatrix} = \textcolor{brown}{\lambda} \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & \textcolor{brown}{1} & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Mediante un operatore di convoluzione che dimezza due pixel all'interno

della matrice possiamo ottenere un risultato del genere:



Formula di convoluzione

Definiamo dunque la formula della convoluzione, vanno definiti due casi:

- Indici i, j partono da -1 -> il centro della matrice avrà coordinata $P(0, 0)$
-> formula (1)
- Indici i, j partono da 0 -> la formula diventa più complessa -> formula (2)

$$g_{m,n} = \sum_{i=-\lfloor s/2 \rfloor}^{\lfloor s/2 \rfloor-1} \sum_{j=-\lfloor t/2 \rfloor}^{\lfloor t/2 \rfloor-1} (h_{i,j} \cdot f_{m+i,n+j}) \quad (1)$$

$$g_{m,n} = \sum_{i=1,j=1}^{s,t} h_{i,j} \cdot f_{m+(i-s+\lfloor s/2 \rfloor),n+(j-t+\lfloor t/2 \rfloor)} \quad (2)$$

Rimane il problema su come fare la convoluzione e il filtraggio ai bordi, come possiamo fare?

Ci sono diversi metodi più o meno efficaci:

- Filtrare solo le zone centrali dell'immagine
- Supporre che tutto attorno all'immagine ci sia o
- Assumere una topologia "toroidale" quando si sfora in tutte le direzioni
- Aggiungere una riga all'inizio uguale alle riga precedente, una riga alla fine uguale all'ultima riga, una colonna all'inizio uguale alla colonna iniziale e una colonna alla fine uguale alla colonna finale.

L'ideale come al solito è il ***non fare nulla***

Esempi di filtri

Filtro binomiale

È un filtro di smussamento derivato dalla distribuzione binomiale, tale distribuzione è una approssimazione discreta della *distribuzione gaussiana*. Ha il pregio di smussare in modo eguale su tutte le direzioni

3-binomiale

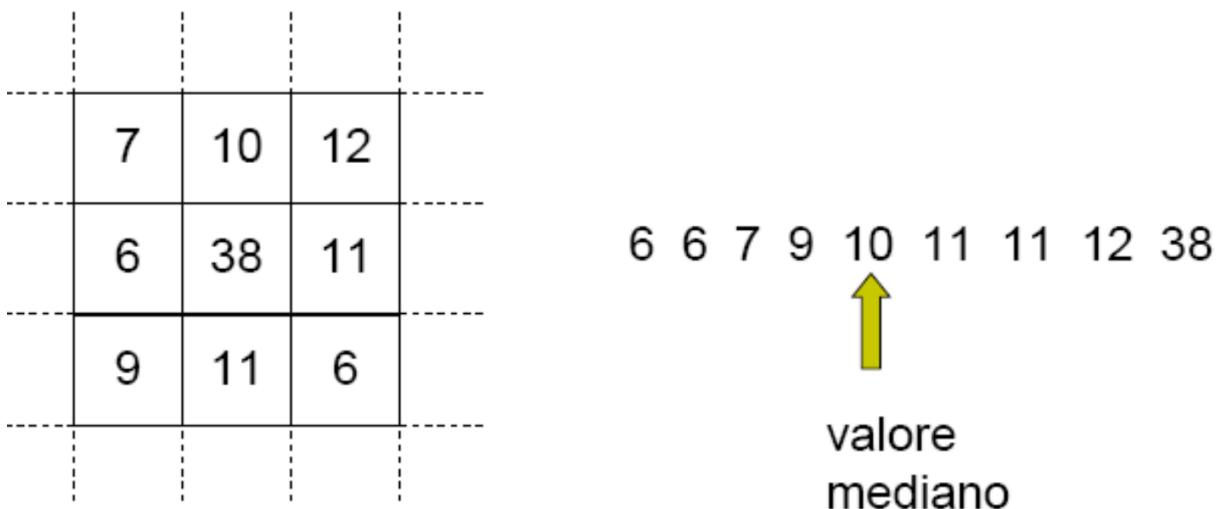
1/16 *	1	2	1
	2	4	2
	1	2	1

5-binomiale

1/256 *	1	4	6	4	1
	4	16	24	16	4
	6	24	36	24	6
	4	16	24	16	4
	1	4	6	4	1

Filtro mediano

Un filtro non lineare che da in uscita il valore mediano dell'intorno del pixel



Notiamo come il risultato di questo filtro è comunque percettibile ma non troppo

Filtro di minimo e massimo

Come è prevedibile il filtro di minimo prende il valore minore di tutto l'intorno e conseguentemente il massimo prende il valore più grande, il risultato quindi sarà di una immagine più scura (rimuovendo eventuali punti neri) per il filtro di minimo e viceversa un'immagine più chiara (rimuovendo eventuali macchioline chiare)



Noise cleaning and smoothing

I filtri appena visti possono essere usati anche per ridurre il rumore all'interno dell'immagine, in questo caso più grande è il kernel migliore sarà il risultato

I filtri N-box e N-binomiali invece hanno il potere di sfocare l'immagine (smoothing), più sarà grande il kernel, più sarà accentuata la sfocatura e si ridurrà il rumore

Rumore

Esistono diversi tipi di rumore, ne vedremo due in questo caso:

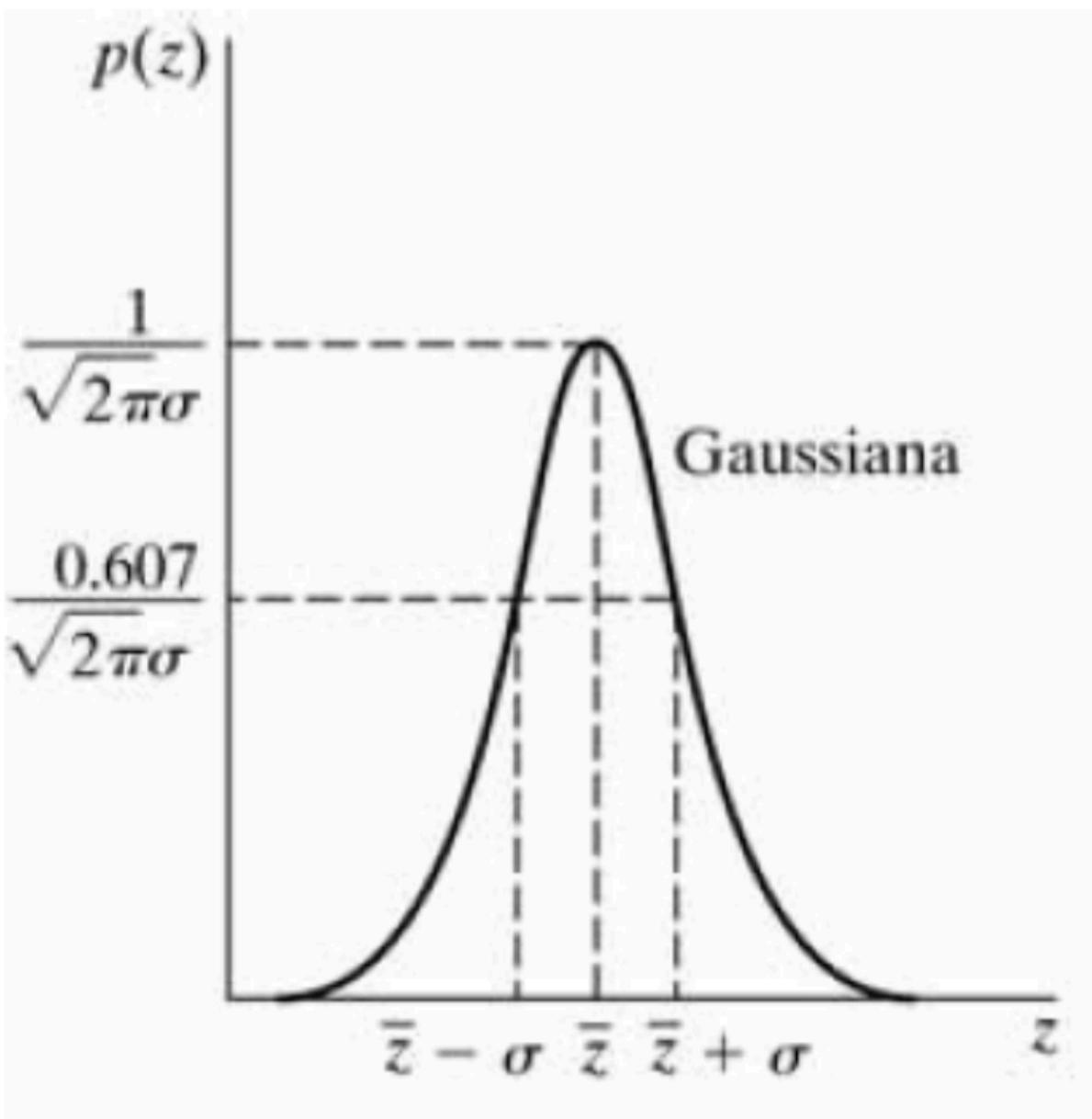
- **Rumore Gaussiano**

Rumore che viene caratterizzato dalla media e dalla varianza, caratterizzato dalla probabilità che qualcosa accada

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\bar{z})^2/2\sigma^2}$$

la probabilità che un certo pixel sia danneggiato dal rumore, gaussiana dalla distribuzione delle probabilità, quindi una forma a campana

Il rumore di tipo gaussiano *non sostituisce il valore del pixel ma lo modifica*



Nelle slide vediamo un esempio di rumore gaussiano al variare di media e di varianza, quest'ultima ha una discreta importanza nel risultato finale

- **Rumore sale e pepe**

Per il rumore sale e pepe dove a random vengono coperti pixel, pepe perché a random andiamo a coprire il pixel di nero, sale perché copriamo l'immagine intera

$$p(z) = \begin{cases} P_a & \text{per } z = a \\ P_b & \text{per } z = b \\ 0 & \text{altrimenti} \end{cases}$$

In base alla percentuale di pixel danneggiati nell'immagine vediamo come l'effetto di rumore diventa molto visibile, anche con piccole percentuali il risultato di danneggiamento è estremamente visibile, vediamo un esempio per 1% e 20%:





Con il filtro mediano posso ripulire in modo efficace l'immagine affetta da rumore sale e pepe, questo però comporta una sfocatura, applicandolo 2 volte avrò a tutti gli effetti un'immagine più pulita ma più sfocata, conviene però applicare il filtro direttamente con un kernel più grande ma una sola volta? No, rimane sempre meglio applicarlo due volte



Gaussiano



Filtro Mediano 5x5



Filtro 5-box

Altri filtri per la riduzione del rumore

- **Outliner**

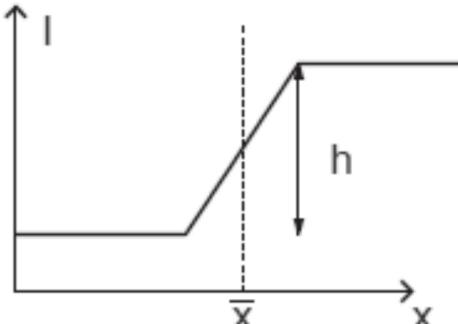
Il valore del pixel centrale viene confrontato con la media degli 8 pixel più vicini. Se il valore assoluto della differenza è maggiore di una certa soglia allora il valore viene sostituito se no si lascia così com'è

- **Olimpico**

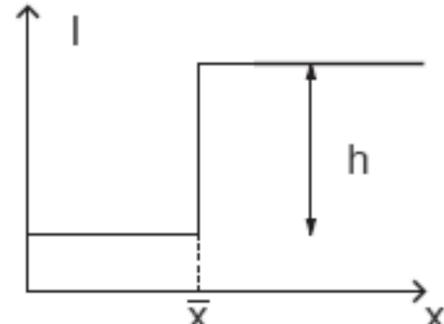
Dall'intorno si scartano il valore massimo e minimo, dunque si fa la media del resto

Estrazione dei contorni

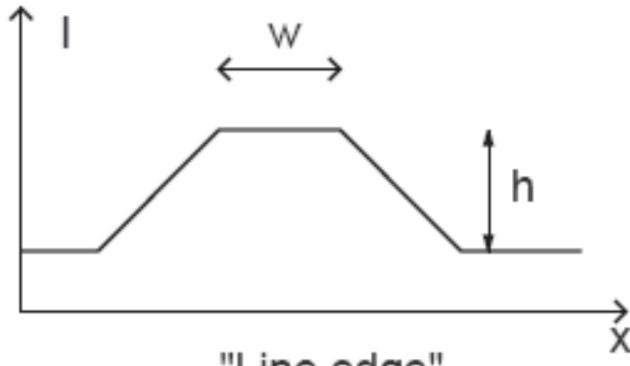
Gli operatori locali ci aiutano ad estrarre i contorni da una immagine, i contorni sono definiti come delle discontinuità locali della luminanza. Gli edge detector si occupano di fornire immagini dove vengono preservate solo le variazioni di luminanza e vengono rimosse altre informazioni, i vari edge possono essere di vario tipo, in questo caso osserviamo degli esempi per una sola dimensione, (quindi una sola riga dell'immagine):



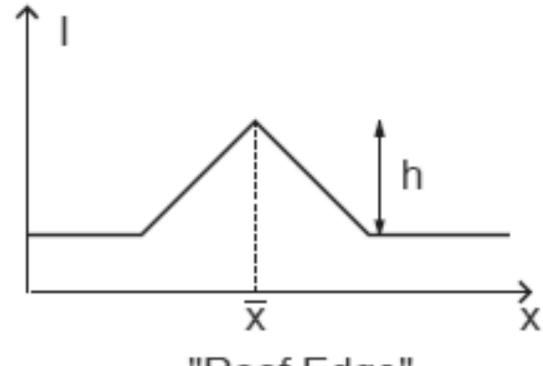
"Ramp Edge"



"Step Edge"



"Line edge"



"Roof Edge"

Facendo la derivata prima notiamo come in ogni corrispondenza del lato la derivata pr' ci darà un massimo, il nostro obiettivo quindi sarà di cercare di svolgere questa operazione e ricavarne il valore, ma noi non sappiamo fare la derivata prima di una immagine, utilizzeremo quindi due kernel notevoli:

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

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

L'immagine risultante sarà quindi tutta nera ma in corrispondenza del lato troveremo dei valori non nulli (approssimati al massimo), tutti i valori

vicini allo 0 non saranno automaticamente valori di massimo perché molto vicini allo 0, andrà quindi fatta una schermatura di questi valori.

Notiamo come nel *Prewitt* hanno tutte lo stesso valore sopra e sotto

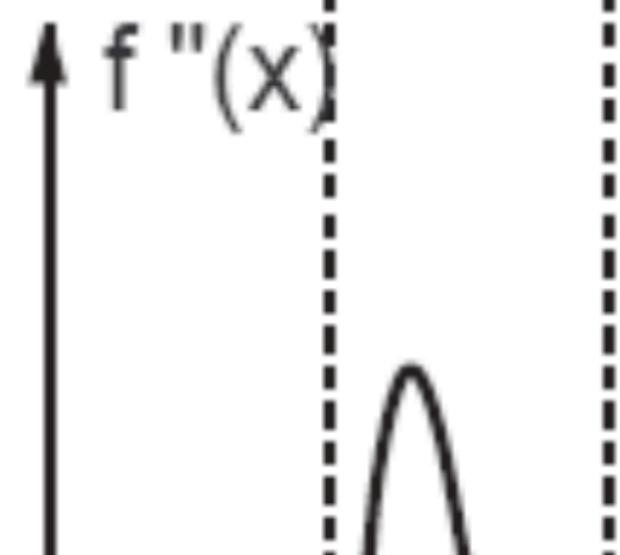
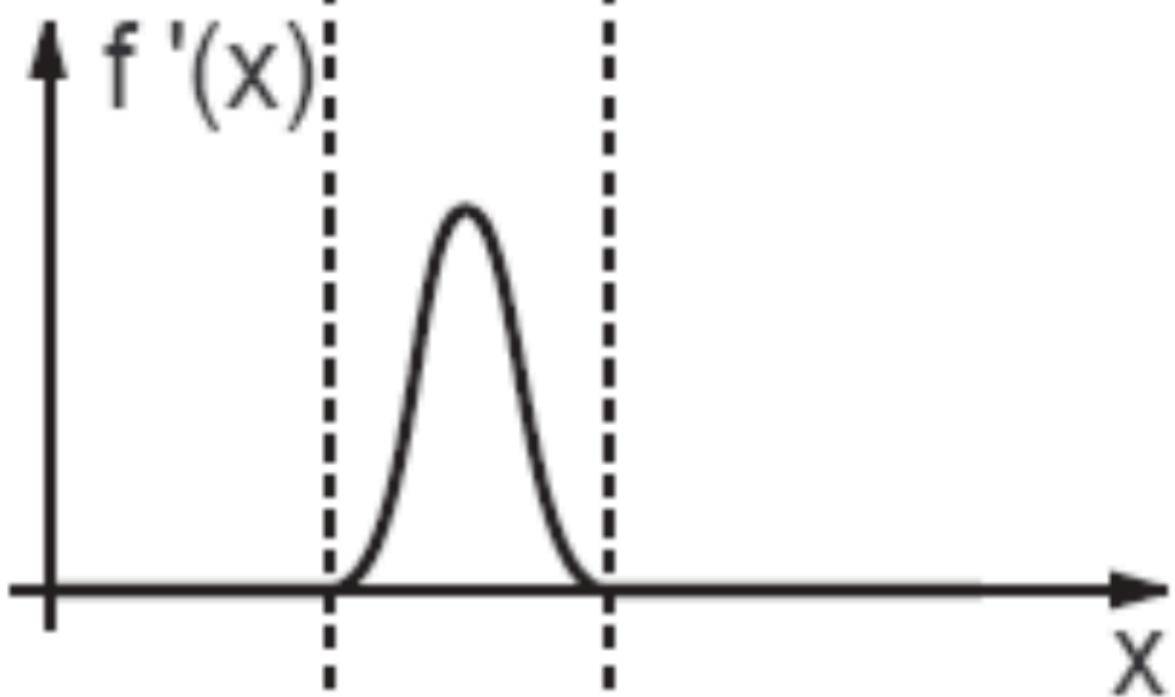
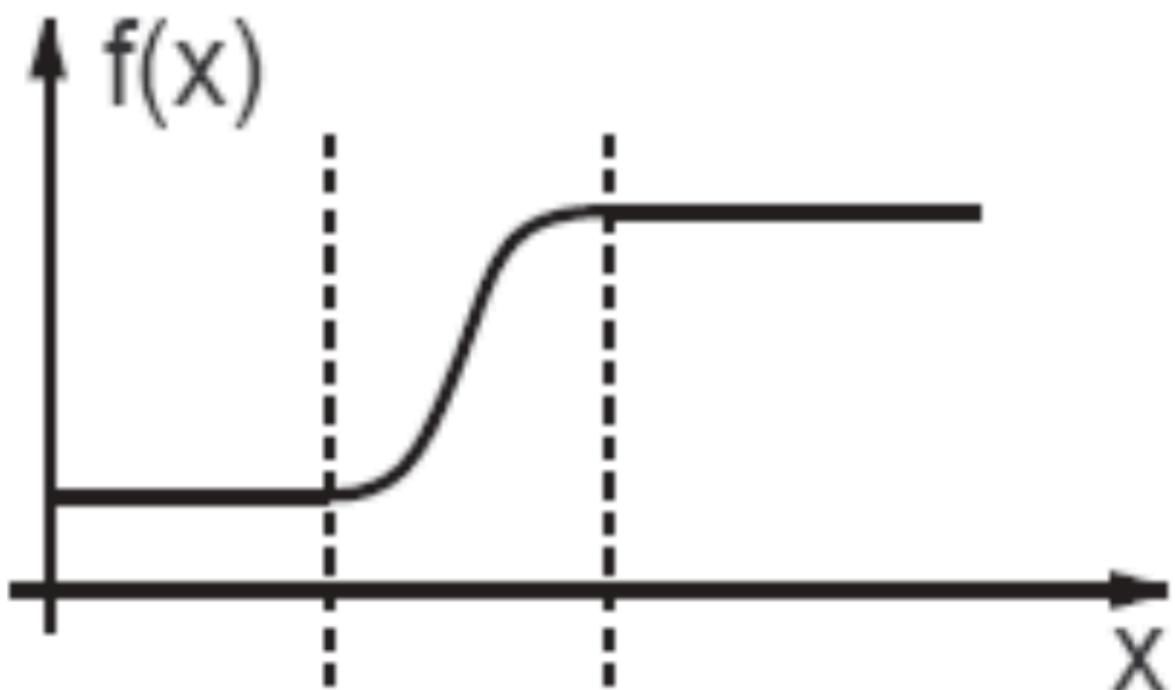
Del *Sobel* e del *Prewitt* esistono due varianti anche per il verticale:

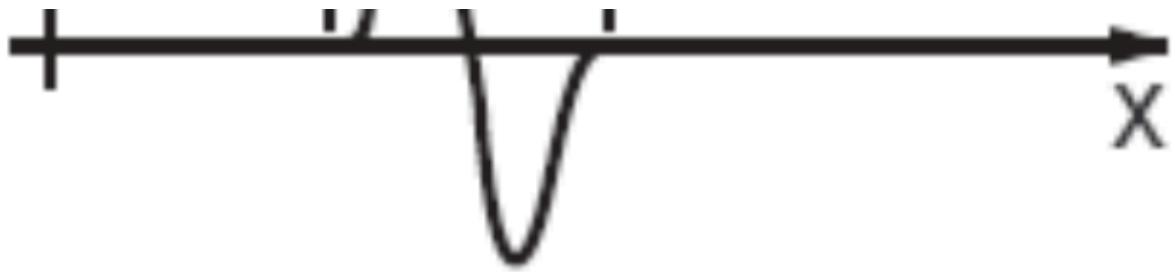
$$Sobel_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

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

I migliori risultati si ottengono con specifici algoritmi e strategie più "intelligenti" che vanno oltre però gli obiettivi del corso.

Utilizzando un edge detector basato sulla derivata seconda, scopriamo che la corrispondenza del lato passa per lo zero

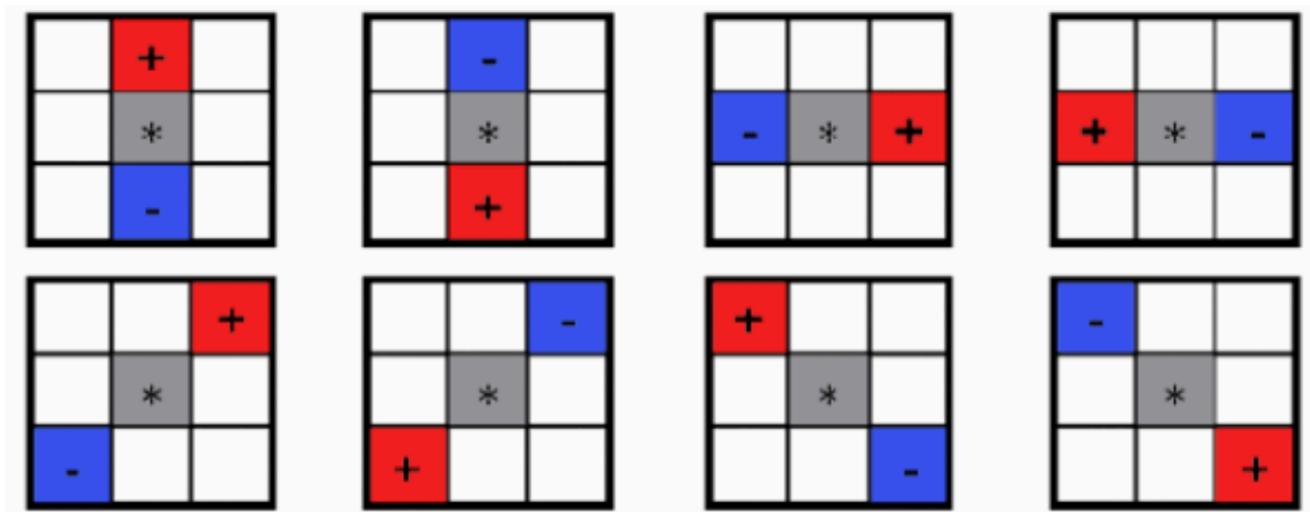




Per calcolare la derivata seconda abbiamo bisogno, come per la derivata prima, di utilizzare un kernel specifico, detto kernel laplaciano, definito dalla maschera:

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

Una volta applicato questo operatore è necessaria la condizione di **zero-crossing** ovvero rispetto al punto in questione vi sia sempre un valore positivo e un valore negativo



Filtro di sharpening

I filtri di sharpening hanno il ruolo di rendere più nitida l'immagine di partenza, è sostanzialmente l'opposto dello sfocamento, per ottenere tale effetto possiamo adottare una maschera che deriva dal laplaciano

rinforzando i lati presenti nell'immagine, vediamo quindi un esempio:



-1	0	-1
0	5	0
-1	0	-1



Filtro di Enhancing

