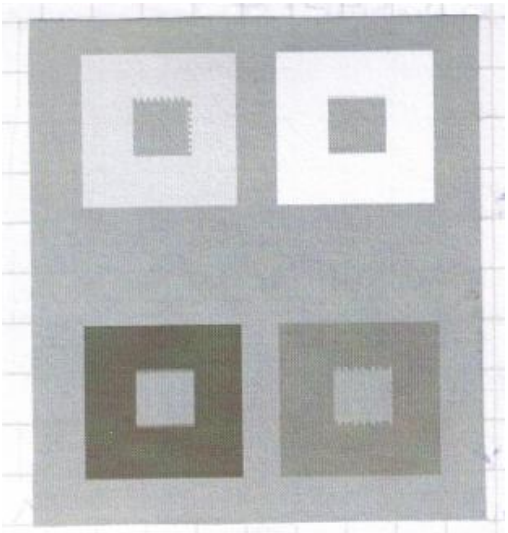
Introduzione alla Visione Artificiale

La **Visione Artificiale (Computer Vision)** è la scienza che tenta di rendere possibile che le macchine possano "vedere", nel senso che non solo possano acquisire immagini, ma sappiano anche interpretare il contenuto dell'immagine stessa, ovvero dare un significato a quell'immagine. Bisogna trovare un metodo che permetta al calcolatore di analizzare le immagini, ottenendo dei risultati "decenti", cioè paragonabili (anche lontanamente) a quello che potrebbe fare il nostro cervello. L'obiettivo è quindi quello di percepire le immagini attraverso una macchina, in modo simile a come le percepisce l'occhio umano.



La **percezione** del colore di un oggetto non dipende tanto dall'intensità della sorgente luminosa, quanto dal contrasto tra l'immagine e il suo sfondo.
In questo esempio infatti si nota come la stessa tonalità di grigio assuma sfumature diverse se lo sfondo cambia.
La **luminanza** è il rapporto tra l'intensità luminosa emessa da una sorgente verso una superficie perpendicolare alla direzione del flusso e l'area della superficie stessa.
La grandezza è indicativa dell'"abbagliamento" che può indurre una sorgente e l'unità di misura è espressa in cd/m^2 (candele su metro quadro).
La luminanza è importante per sorgenti estese in quanto ci dà un'idea di quanto è concentrata la sorgente.
Inoltre il rapporto tra la luminanza di una sorgente e quella dello sfondo è detto **fattore di contrasto**.

Analogico: ciò che si riferisce ad un insieme non numerabile, cioè non analizzabile entro un insieme discreto di elementi. Ciò che è analogico viene modellizzato con la matematica del continuo, che tratta un'infinità (numerabile e non) di elementi.

Digitale: indica ciò che può essere rappresentato con i numeri e può essere contato.
Al contrario di analogico, ciò che è digitale viene modellizzato con la matematica del discreto, che tratta un insieme finito di elementi. Il passaggio da analogico a digitale è chiamato **digitalizzazione**.

Pixel (picture element): elemento puntiforme che compone la rappresentazione di un'immagine nella memoria di un computer.
Un'**immagine digitale** D è una funzione definita sullo spazio discreto bidimensionale, detto **retina**, i cui valori discreti sono detti livelli di grigio g (luminosità):

$$D = \{ (i, j, g) \text{ tale che } i \in \{0, \ldots, w-1\}, j \in \{0, \ldots, h-1\} \text{ e } g \in \{0, \ldots, G-1\} \}$$

dove i e j sono le coordinate del punto nella struttura digitale (width = larghezza e height = altezza).
Per convenzione associamo i G livelli di grigio in modo tale che g = 0 equivale a nero, g = G-1 equivale a bianco e g = G/2 equivale a grigio.
La quantità di informazioni necessaria per rappresentare un'immagine digitale dipende dalla precisione con cui si codificano le coordinate spaziali e le intensità luminose.
1 byte corrisponde a 256 sfumature di grigio; il nostro occhio non riesce a percepirle tutte, ma soltanto una trentina circa. In ogni caso percepiamo meglio le sfumature di grigio che quelle relative ai colori (se la luce ad esempio diminuisce). Tra questi a loro volta percepiamo meglio le sfumature di verde, poi il rosso e infine il blu.

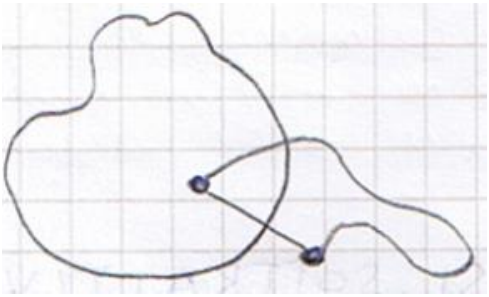
La conversione di un segnale analogico in segnale digitale si divide in due fasi:

- Il **campionamento**: consiste nel "leggere" nello spazio il valore del segnale.
- La **quantizzazione**: consiste nel codificare i valori assegnando ad ognuno di essi un valore discreto (nella luce). La quantizzazione introduce un errore nella sequenza di campioni. Dopo la quantizzazione non è più possibile effettuare l'esatta ricostruzione del segnale analogico.
  [x e y indicano le coordinate dell'immagine analogica, mentre l il valore della luminosità]
  $i = \min \{ \llcorner\ w \cdot (x - x_{min}) / (x_{max} - x_{min})\ \lrcorner , w-1\}$
  $j = \min \{ \llcorner\ h \cdot (y - y_{min}) / (y_{max} - y_{min})\ \lrcorner , h-1\}$
  $g = \min \{ \llcorner\ G \cdot (l - l_{min}) / (l_{max} - l_{min})\ \lrcorner , G-1\}$

Se consideriamo Δ la minima variazione apprezzabile di luminosità, il numero totale di livelli di grigio è dato da $G = (l_{max} - l_{min}) / \Delta$

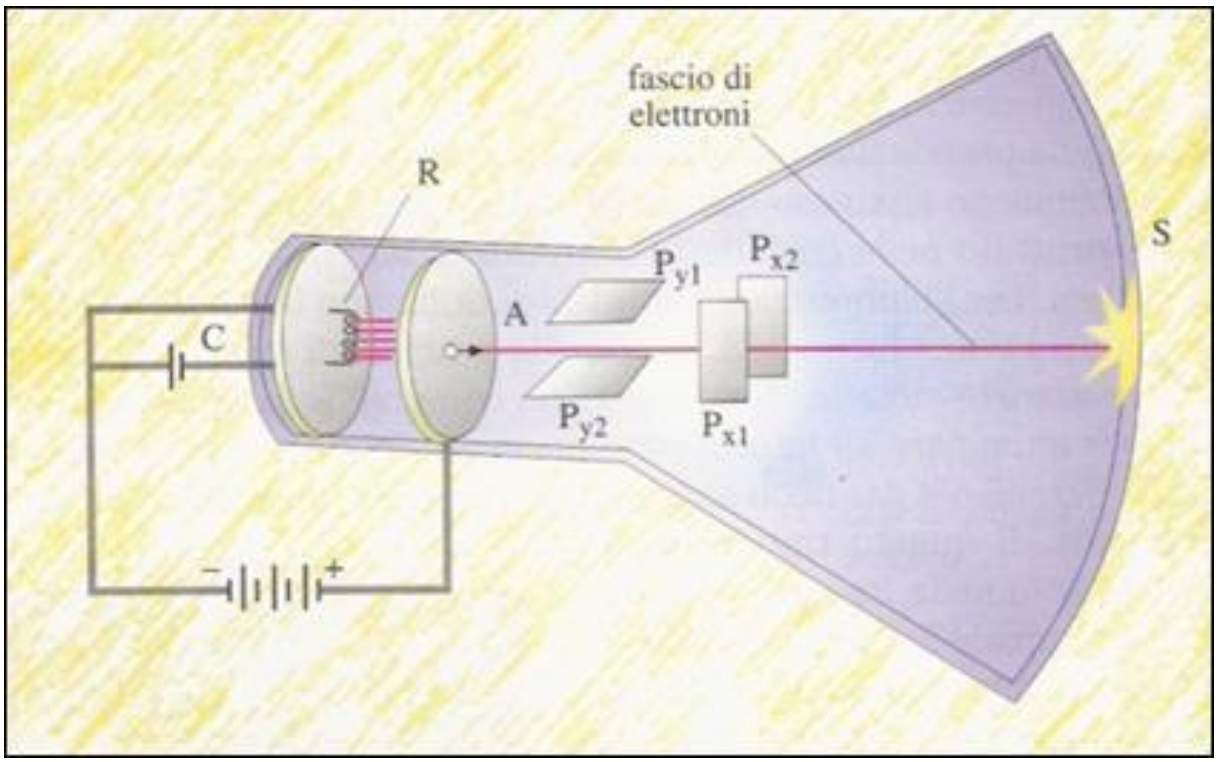
Teorema del campionamento: mette in relazione il contenuto di informazione di un segnale campionato con la frequenza di campionamento e le componenti minime e massime di frequenza del segnale analogico originale, definendo così la minima frequenza necessaria per campionare un segnale analogico senza perdere informazioni e per poter quindi ricostruire il segnale analogico originario. In particolare il teorema afferma che in una conversione analogico-digitale la minima frequenza di campionamento necessaria per evitare perdita di informazione nella ricostruzione del segnale analogico originario è pari al doppio della sua frequenza massima.

Possiamo rappresentare la retina digitale in due modi:
Intorno forte (4-connettivtà): 4 pixel adiacenti sul lato comune. Contiene 5 pixel totali.
Intorno debole (8-connettività): 8 pixel adiacenti (fanno parte dell'intorno anche i pixel che hanno solo l'angolo in comune). Contiene 9 pixel in totale.



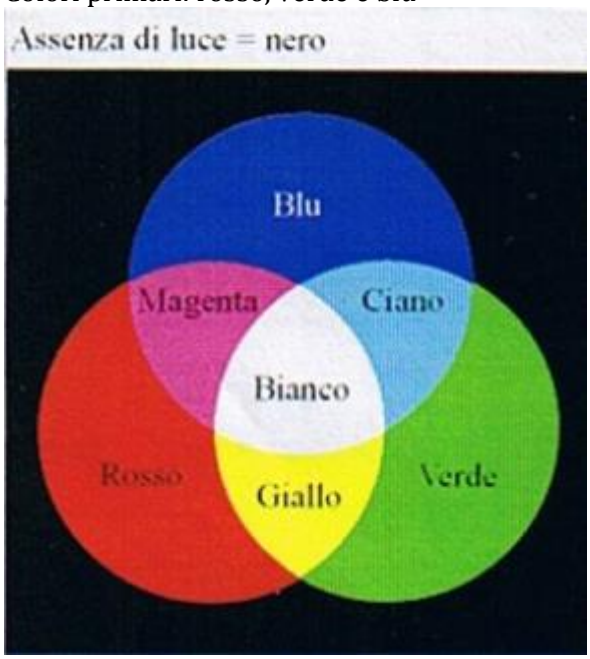
Paradosso di Jordan: Una curva chiusa delimita 2 aree. Se prendo un punto interno e un punto esterno, qualunque linea che li unisce taglia la curva. Questo teorema vale nel continuo. Se si utilizza la 4-connetività vale anche per il discreto, invece non vale se si utilizza la 8-connettività.

Il **tubo a raggi catodici** (**CRT**) viene usato per produrre su uno schermo (per esempio di una tv) un'immagine. Consiste nell'emissione di elettroni da parte di una lastra detta **catodo**, che riscaldato dalla corrente elettrica, genera un flusso di elettroni che si dirigono in una lastra carica positivamente detta **anodo**. Un foro praticato nell'anodo consente il passaggio di un fascio di elettroni, che prosegue il suo percorso nel vuoto, senza incontrare resistenza, fino a colpire lo schermo. Questo è ricoperto da granelli di fosforo che colpiti da elettroni si illuminano.



L'intensità luminosa è proporzionale al numero di elettroni che colpiscono lo schermo, ossia alla corrente del fascio. La posizione del punto luminoso è controllata da due coppie di placchette magnetiche di deflessione che deviano il fascio di elettroni in orizzontale e in verticale spostando il punto luminoso dello schermo. Il fascio di elettroni viene spostato rapidamente, attraversando lo schermo in righe. In ogni istante, un solo punto dello schermo viene colpito dagli elettroni emessi dal catodo, ma la successione è talmente rapida che lo schermo sembra illuminato contemporaneamente. Con il meccanismo descritto si ottengono delle immagini "in bianco e nero": dove arriva il fascio si ha il bianco e dove non arriva si ha il nero. In un televisore a colori vi sono tre fasci di elettroni che operano contemporaneamente, ognuno dei quali incide su strati differenti dello schermo, che producono punti colorati rispettivamente di rosso, verde e blu. Dalla composizione e distribuzione di questi punti si ottengono molte sfumature di colore, che rendono le immagini create sullo schermo simili a quelle reali.

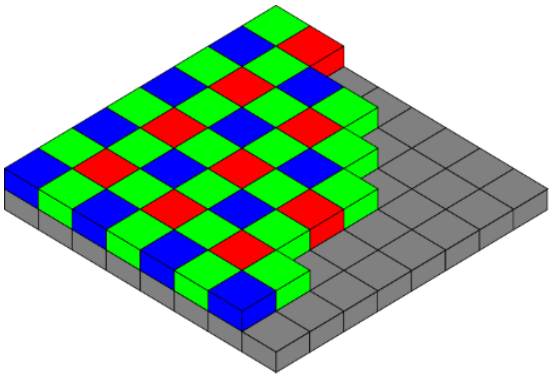
Sintesi additiva
Colori primari: rosso, verde e blu



Sintesi sottrattiva
Colori primari: ciano, magenta e giallo.

RGB è il nome di un modello di colori, che diversamente dalle immagini a livelli di grigio, si basa sui tre colori rosso (*Red*), verde (*Green*) e blu (*Blue*), da cui appunto il nome RGB. Un'immagine può infatti essere scomposta in questi colori base che miscelati tra loro danno tutto lo spettro dei colori visibili. L'RGB è un modello additivo: abbiamo 256 sfumature per ogni colore rosso, verde e blu; unendo i tre colori con la loro intensità massima (255, 255, 255) si ottiene il bianco, mentre (0, 0, 0) corrisponde al nero. La rappresentazione di un'immagine digitale a colori richiede quindi la conoscenza di tre parametri per ogni pixel.

Un **CCD** (**Charge Coupled Device**) consiste in un circuito integrato formato da una griglia di elementi semiconduttori in grado di trasformare il flusso luminoso che li colpisce in una determinata quantità di carica elettrica (*charge*). Questi elementi sono accoppiati (*coupled*) in modo che ognuno di essi possa trasferire la propria carica ad un altro elemento adiacente.
Inviando al dispositivo (*device*) una sequenza temporizzata d'impulsi, si ottiene in uscita un segnale elettrico grazie al quale è possibile ricostruire la matrice dei pixel che compongono l'immagine proiettata sulla superficie del CCD stesso.



Il **Bayer pattern** (o filtro di Bayer) è uno schema per la disposizione degli elementi sensibili ai diversi colori nei sensori usati per l'acquisizione di immagini digitali. La sua caratteristica è quella di raggruppare i sensori per i tre colori fondamentali (RGB) necessari per la sintesi additiva. Ogni cella 2x2 contiene due elementi verdi, uno rosso e uno blu. Il filtro di Bayer prevede che nelle otto cellule adiacenti ad ogni cella, ve ne siano almeno due di ognuno degli altri colori. In questo modo è possibile ricostruire il valore della luminosità, ad esempio, del rosso in corrispondenza di un elemento verde o blu, deducendolo dagli elementi rossi circostanti.

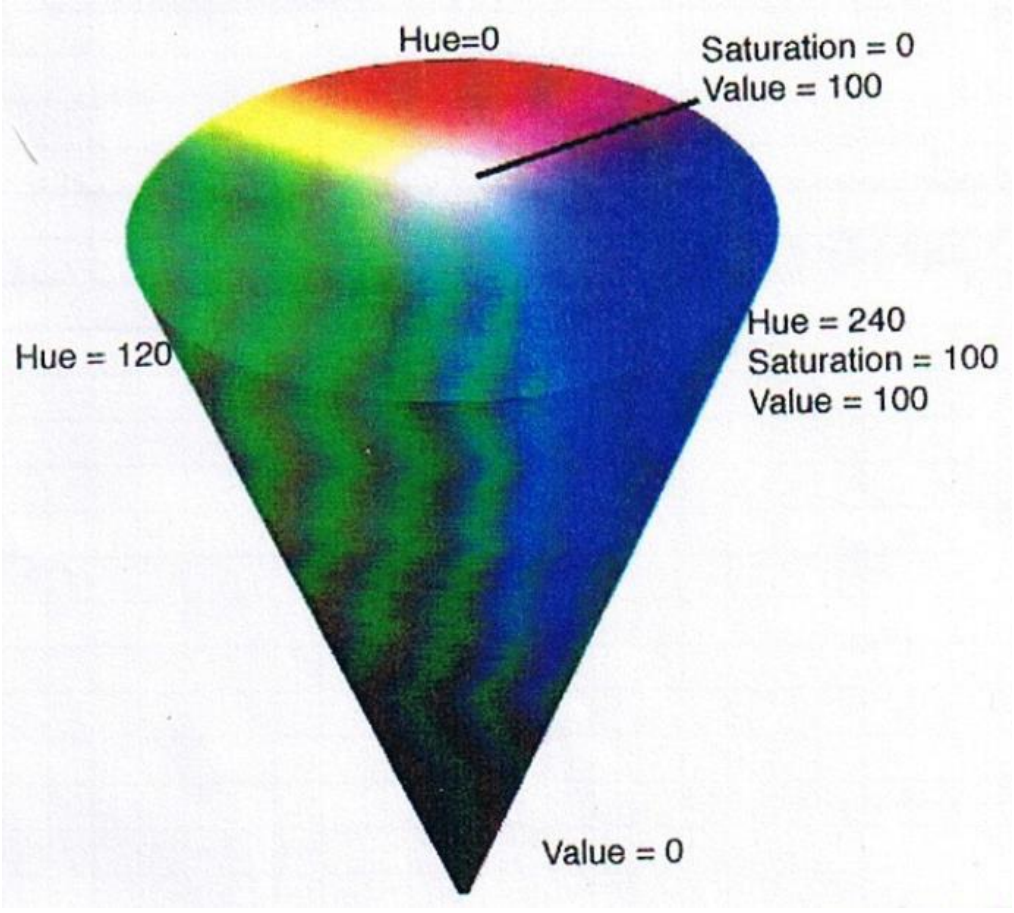
Lo standard **YUV** è un modello di rappresentazione del colore dedicato al video analogico.
Si basa su una modalità di trasmissione video a componenti separate che usa tre cavi diversi per far transitare le informazioni di luminanza (luminosità rappresentato dal parametro Y) e le due componenti di crominanza (colore) rappresentati dai parametri U e V.
Questo modello è stato messo a punto per permettere di trasmettere delle informazioni colorate ai televisori a colori, assicurandosi che le televisioni in bianco e nero esistenti continuano a visualizzare un'immagine nei toni del grigio. Ecco le relazioni che legano YUV a RGB:

$Y = 0.299R + 0.587\ G + 0.114\ B$

$U = -0.147R - 0.289\ G + 0.436B = 0.492(B - Y)$

$V = 0.615R - 0.515G - 0.100B = 0.877(R-Y)$

HSV (**H**ue **S**aturation **V**alue, ovvero tonalità, saturazione e valore) indica sia un metodo additivo di composizione dei colori, sia un modo per rappresentarli in un sistema digitale. Questo spazio di colori può essere rappresentato graficamente come un cono: dei tre parametri, H corrisponde all'angolo e parte dal rosso a 0°, passando per il verde a 120° e il blu a 240°; la saturazione S indica l'intensità della singola tonalità e va dal centro al bordo; mentre V è il valore della luminosità e va da 0 a 100, cioè dal nero (punta del cono) al bianco (centro della base).



Per poter passare da un'immagine a colori ad una a scala di grigio occorre passare dal modello RGB al modello HSV considerando solo V (la luminosità) ed eliminando H e S.
Oppure si potrebbe sommare i valori di rosso, verde e blu e dividere per 3: (R + G + B)/3
Ma questo non è molto corretto in quanto diamo ai tre colori lo stesso peso e sappiamo che non è così perché li percepiamo in modo diverso.
Il modo più opportuno per ottenere la luminosità è trovare la Y nel modello YUV:

$Y = 0.299R + 0.587\ G + 0.114\ B$

Osserviamo che il valore del verde è maggiore (perché lo percepiamo meglio) e che la somma fa 1.

Osserviamo come varia il seguente fiore modificando i parametri H, S e V

Angolo: ad ogni 60° cambia colore

Saturazione:

Luminosità:

Le varie fasi dell'analisi di un'immagine possono essere raggruppate in:

- **Basso livello** (o preanalisi):
  - le strutture dati mantengono le informazioni contenute nei pixel
  - gli operatori agiscono sui pixel o sui loro intorni
  - il risultato è una nuova immagine B : I → I'
  (esempio: riduzione del rumore e aumento del contrasto)
- **Medio livello**:
  - le strutture dati mantengono le caratteristiche dell'immagine
  - gli operatori, in genere globali, estraggono delle proprietà
  - il risultato è un insieme di proprietà dell'immagine M : I' → F
  (esempio: estrazione contorni e segmentazione)
- **Alto livello**:
  - le strutture dati codificano il senso dell'immagine (interpretazione)
  - gli operatori classificano le proprietà
  - il risultato è una descrizione dell'immagine A : F → R
  (esempio: interpretazione simbolica)

Operazioni fra immagini
Immagine nera → $0 = \{ (i, j, g) : g = 0 \}$
Immagine bianca → $255 = \{ (i, j, g) : g = 255 \}$

Prodotto di una immagine per uno scalare: occorre mantenere $g \leq G-1$
$I \times k = \{ (i, j, g) : g = (g \times k) \bmod G \}$ usando il modulo oppure
$I \times k = \{ (i, j, g) : g = \min\{ G-1, g \times k \} \}$ tutti i valori maggiori di 255 li forziamo a 255

Somma di un immagine con uno scalare:
$I + k = \{ (i, j, g) : g = \min\{ G-1, g + k \} \}$

Minimo e massimo tra due immagini: si prende pixel per pixel quello con il valore di g minore o maggiore
$\min(I_1, I_2) = \{ (i, j, g) : g = \min\{ g_1, g_2 \} \}$
$\max(I_1, I_2) = \{ (i, j, g) : g = \max\{ g_1, g_2 \} \}$

Somma tra due immagini:
$I_1 + I_2 = \{ (i, j, g) : g = \min\{ G-1, g_1 + g_2 \} \}$ oppure
$I_1 + I_2 = \{ (i, j, g) : g = (g_1 + g_2) \bmod G \}$ oppure
$I_1 + I_2 = \{ (i, j, g) : g = (g_1 + g_2) / 2 \}$

Differenza tra due immagini:
$I_1 - I_2 = \{ (i, j, g) : g = \max\{ 0, g_1 - g_2 \} \}$ oppure
$I_1 - I_2 = \{ (i, j, g) : g = |g_1 - g_2| \}$

Prodotto tra due immagini:
$I_1 \times I_2 = \{ (i, j, g) : g = (g_1 \times g_2)/(G-1) \}$

Quadrato di una immagine
$sqr(I) = \{ (i, j, g) : g = g^2/(G-1) \}$

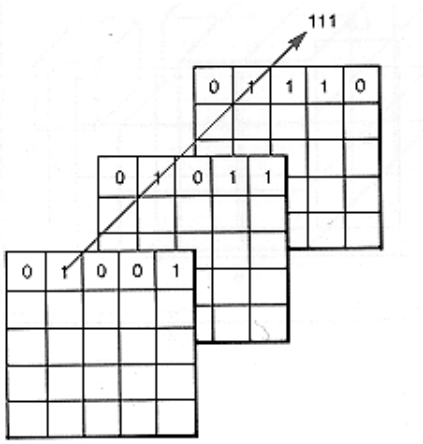
Radice quadrata di una immagine
$sqrt(I) = \{ (i, j, g) : g = sqrt((G-1)\, g) \}$

Logaritmo di una immagine
$log(I) = \{ (i, j, g) : g = log(\varepsilon+g) \times (G-1) / log(G-1) \}$

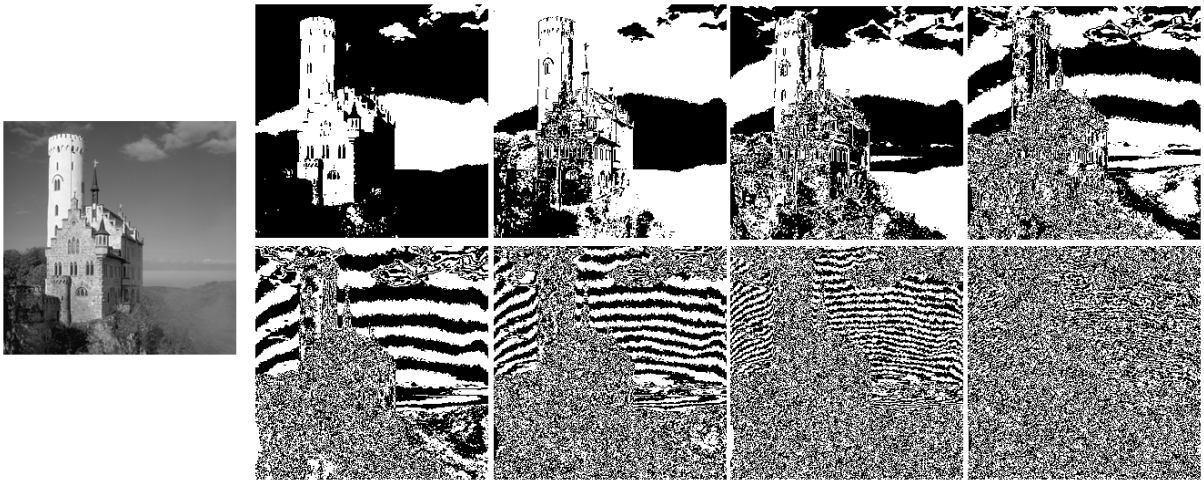
Operazioni booleane: operatori bitwise, cioè procedono bit per bit
$and(I_1, I_2) = \{ (i, j, g) : g = and(g_1, g_2) \}$
$or(I_1, I_2) = \{ (i, j, g) : g = or(g_1, g_2) \}$
$not(I) = \{ (i, j, g) : g = not(g) \}$ si fa il complemento a 255, cioè si esegue 255 - g
$shift(I, k) = \{ (i, j, g) : g = shift(g, k) \}$ shift di k valori: a sinistra moltiplico, a destra raddoppio

Un'immagine è codificata come una matrice di **pixel**, in cui ogni pixel codifica il colore del punto corrispondente dell'immagine. Il numero di bit usati per codificare un colore è detta **profondità** dell'immagine. La larghezza e altezza dell'immagine, in pixel, danno la **risoluzione** dell'immagine. L'occupazione di memoria di un'immagine è legata alla sua risoluzione e alla sua profondità. Più è alta la risoluzione, più è definita l'immagine; più è grande la profondità e più colori ci sono. Ovviamente maggiore è la qualità dell'immagine, maggiore sarà l'occupazione di memoria.

Occupazione in bit = larghezza (in pixel) × altezza (in pixel) × profondità colore (in bit).
L'organizzazione in memoria dei dati che codificano un'immagine può essere di vari tipi.
Uno fra questi è l'organizzazione a **bitplane**, in cui l'immagine è scomposta in tanti "piani" quanti sono i bit di profondità colore. I bit che occupano la stessa posizione in piani diversi formano il codice numerico del colore.



In un immagine a livelli di grigio un pixel è di 1 byte (256 livelli di grigio), quindi avremmo 8 bitplane uno per ogni bit, partendo dal meno significativo al più significativo. Osserviamo che il bitplane del bit meno significativo non dà molte informazioni sull'immagine; procedendo fino ad arrivare al bitplane del bit più significativo l'immagine diventerà sempre più chiara.



La **convoluzione** è un processo mediante il quale si possono applicare filtri a immagini. Date due matrici bidimensionali di cui la prima rappresenta l'immagine originale e la seconda, detta **kernel**, rappresenta il filtro da applicare, otteniamo una matrice che rappresenta l'immagine modificata. Consideriamo la matrice I contenente i valori di grigio di tutti i pixel dell'immagine originale e la matrice K che rappresenta il kernel. Sovrapponiamo il kernel K alla matrice I in modo che il centro della matrice K sia in corrispondenza del pixel della matrice I da elaborare.
Il valore di ciascun pixel della matrice I viene ricalcolato come la somma dei prodotti di ciascun elemento del kernel con il corrispondente pixel della matrice I sottostante.
Inoltre il kernel solitamente ha dimensioni 3x3, 5x5, 7x7 e così via e deve essere formato in modo tale che ogni prodotto scalare abbia come risultato un valore compreso fra 0 e 255.

Problema del bordo: cosa succede nel momento in cui bisogna calcolare il nuovo valore di un pixel situato sul bordo? Per tutti i pixel sul confine (e nel caso di matrici di ordine superiore al terzo anche per elementi più interni) alcuni elementi della matrice di convoluzione si trovano nell'area al di fuori dell'immagine. In questi casi si può agire in vari modi:

1. Si ignora la parte esterna al kernel
2. Si sceglie di non applicare la matrice ai bordi
3. La parte del kernel esterna all'immagine verrà applicata a pixel del bordo opposto cosicché pixel che scompaiono da un lato ricompaiono dall'altro

Deconvoluzione
Generalmente le immagini presentano delle deformazioni dovute al sistema di acquisizione.
La deconvoluzione è l'esatto opposto della convoluzione ovvero l'operazione che permette di ottenere l'immagine originale f a partire da una immagine degradata. Il procedimento di deconvoluzione tenta di recuperare un segnale distorto e rendere le immagini il più possibile fedeli alle originali "restaurandole" (problema della distorsione a botte, gatto dietro acquario).
Ad esempio se l'immagine è sfocata possiamo osservare una piccola regione denotata con $g_s(x, y)$ dell'immagine contenente strutture campione, come parte di un oggetto o lo sfondo.
Il passaggio successivo sarà elaborare la sottoimmagine per arrivare a un risultato che sia il meno sfocato possibile e la sottoimmagine elaborata, indicata con $f_s(x, y)$ sarà la nostra stima dell'immagine originale in quell'area.
La funzione di degrado sarà data da: H = g_s/f_s
Grazie alla funzione di degrado applicata all'immagine degradata sarò in grado di ricostruire l'immagine originale.
La **point spread function** (**PSF**) è una funzione che descrive la distorsione in termini di percorso della sorgente puntiforme di luce che avviene attraverso lo strumento. Di solito, tale sorgente puntiforme contribuisce una piccola area di sfocatura all'immagine finale. Se questa funzione può essere determinata, basta allora calcolare la sua inversa o la funzione complementare, e fare la convoluzione dell'immagine acquisita con quella. Il risultato è l'immagine originale non distorta. In pratica però trovare il vero PSF è impossibile e di solito si utilizza una sua approssimazione, calcolato sulla base di alcune stime sperimentali.
Il più comune algoritmo iterativo per la deconvoluzione è l'algoritmo di Richardson-Lucy.

Principali filtri di convoluzione

I filtri posso essere divisi in 3 grandi categorie:

- Filtri passa basso (o di smoothing)
- Filtri passa alto (evidenziano i dettagli)
- Filtri per la rilevazione dei bordi

I **filtri passa basso** sono utilizzati per conservare le basse frequenze e smussare le alte. Rendono il colore più uniforme ma causano sfocatura e possono essere usati per la rimozione del rumore dalle immagini. Per **rumore** si intende la presenza di un disturbo nelle immagini. Uno di quelli più comuni è il cosiddetto rumore **sale e pepe** che è legato alla presenza di pixel che hanno un colore notevolmente diverso da quelli dei pixel vicini solitamente pixel bianchi o neri quindi con valori 255 o 0.
I **filtri di smoothing** (lisciamento) quindi eliminano i picchi e le increspature, ma possono sfocare eccessivamente l'immagine.

Il kernel del **filtro media** è fatto in modo tale da effettuare una media tra i pixel sottostanti il kernel. Quindi per esempio se abbiamo un kernel 3x3 una possibile configurazione della matrice kernel potrebbe essere quella dove tutti gli elementi sono 1/9. Un altro kernel potrebbe essere quello dove al centro mettiamo zero mentre tutti gli altri valori sono 1/8.
Utilizzando la media per ridurre il rumore oltre a sfocare l'immagine, sparpagliamo anche il segnale; dunque se prima i pixel affetti da rumore erano pochi, adesso lo sono un po' tutti. In pratica sparpagliamo il rumore.

Il **filtro mediano** invece è un filtro di non convoluzione in quanto non è un operazione lineare. Esso infatti non fa uso di una matrice per effettuare le operazioni di convoluzione descritte prima, ma considera una maschera di pixel.
Considerati infatti n valori di pixel li ordina in una lista a secondo del loro livello di grigio e prende il valore mediano (centrale) e lo attribuisce al pixel della nuova immagine alla posizione centrale della maschera. Se la maschera che si considera ha numero di elementi pari si prende come posizione centrale la posizione per eccesso o per difetto.
La differenza con il filtro media sta nel fatto che mentre questo modifica i valori, il filtro mediano non inserisce nuovi valori nell'immagine e per questo motivo l'immagine risulta meno sfocata. Questo filtro riesce ad eliminare quasi completamente il rumore sale e pepe. Infatti ordinando i valori secondo i livelli di grigio di ogni pixel, i valori 0 e 255 del rumore sale e pepe si troveranno lontani dal centro della lista di ordinamento, dunque saranno eliminati. Tuttavia può anche capitare che vi sia un'alta concentrazione di rumore all'interno dei pixel considerati dal kernel, quindi a volte può accadere che al centro si trovino dei pixel neri o bianchi.

Il **filtro motion** crea un effetto di movimento. Ad esempio specificando un angolazione di 45 gradi si farà una media lungo una diagonale a 45 gradi del kernel. A zero gradi da come effetto il movimento dell'immagine in orizzontale.

Il **filtro gaussiano** definisce i coefficienti della maschera di convoluzione secondo la funzione Gaussiana, ovvero nel kernel i valori sono maggiori verso il centro e più piccoli verso l'esterno.
Il rumore gaussiano, a differenza del sale e pepe che è completamente casuale, ha una distribuzione d'ampiezza gaussiana in cui ogni pixel risente di un cambiamento rispetto il suo valore originale. In altre parole, i valori che il rumore può assumere, ponendo in ascissa la frequenza e in ordinata la densità di probabilità, sono distribuiti secondo una gaussiana.
La *densità di probabilità* è la probabilità che un certo valore x si possa presentare in ogni punto dello spazio campionario. Il filtro gaussiano fa uso della *deviazione standard* che è un indice statistico che consente di misurare la dispersione delle singole osservazioni intorno alla media aritmetica.
Quindi per valori piccoli di σ il filtro produce uno smoothing lieve, eliminando poche alte frequenze, al contrario se ha valori alti lo smoothing è maggiore, eliminando un numero maggiore di alte frequenze.
L'eliminazione delle alte frequenze comporta una maggiore attenuazione del rumore.

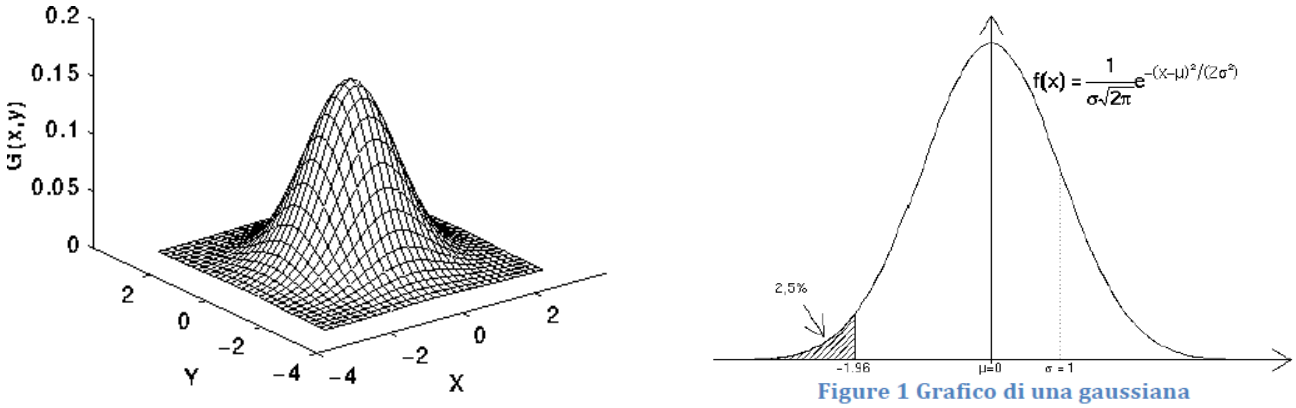
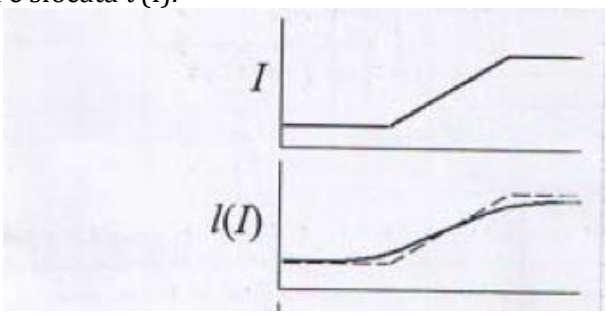


Figure 1 Grafico di una gaussiana

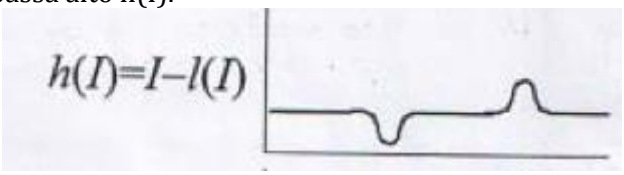
La gaussiana è una funzione a **variabili separabili**, ovvero una funzione che rispetta la proprietà per cui $f(x, y) = f(x) \times f(y)$
Questo vuol dire che invece di utilizzare la funzione a 2 variabili (bidimensionale) con la matrice kernel, possiamo utilizzare la funzione ad una sola variabile (monodimensionale) con un vettore colonna o un vettore riga.
Può essere implementato come convoluzione di due filtri gaussiani monodimensionali (orizzontale e verticale). Ciò è fattibile utilizzando un singolo filtro monodimensionale gaussiano e trasponendo l'immagine dopo ogni convoluzione (con la stessa maschera monodimensionale).
Se facciamo la gaussiana bidimensionale con un kernel 5x5 faremo 25 prodotti e 24 somme, mentre nel caso monodimensionale i calcoli si riducono perché faremo prima la convoluzione con il vettore colonna e al risultato applicheremo la convoluzione per il vettore riga quindi faremo 5+5 prodotti e 4+4 somme.

I **filtri passa alto** aumentano il contrasto dell'immagine, ovvero evidenziano i picchi, cioè le alte frequenze nel segnale, evidenziando così i dettagli.
Per esempio il **filtro di Sharpen** aumenta il contrasto dell'immagine e di solito ha un kernel il cui valore centrale è positivo, mentre i valori esterni sono negativi.

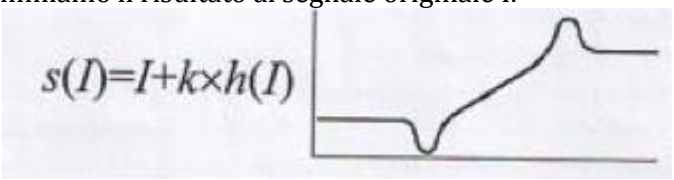
Per far ciò, dato un segnale monodimensionale I, applichiamo un filtro passa basso al segnale e avremo così una versione smussata e sfocata $\ell(I)$.



Facciamo la differenza tra il segnale originale I e il segnale che abbiamo smussato $\ell(I)$, ottenendo così la maschera che è il filtro passa alto h(I).



Infine moltiplichiamo il filtro passa alto h(I) per un certo scalare k, detto peso (che indica quanto deve essere alta la curva) e sommiamo il risultato al segnale originale I.



In questo modo evidenziamo i dettagli dell'immagine aumentandone il contrasto.

Ponendo k = 1 abbiamo che s(I) = I + h(I) = I + I - $\ell(I)$ = 2I - $\ell(I)$
Consideriamo per esempio $\ell(I)$ come il filtro media (passa-basso) quindi abbiamo:

$$2 \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} - 1/9 \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = 17/9 \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

In questo modo abbiamo creato un nuovo kernel che aumenta il contrasto dell'immagine.
Se k > 3 si evidenzia troppo il contrasto e il segnale ha dei picchi eccessivi che sgranano l'immagine.

I **filtri per la rilevazione dei bordi** non modificano l'immagine, ma restituiscono solo una evidenziazione dei bordi dell'immagine.
Abbiamo diversi modelli di edge che vengono classificati a seconda della loro intensità:

- Un *edge a gradino* implica una transizione tra due livelli di intensità che avviene idealmente alla distanza di 1 pixel. Gli edge a gradino sono degli edge ideali e si presentano ad esempio in immagini generate al computer.
- *Edge a rampa*. Nella pratica le immagini digitali hanno degli edge che sono sfocati e rumorosi con una sfocatura determinata dalla messa a fuoco del dispositivo di acquisizione.
  In questa situazione gli edge sono come se avessero un profilo di intensità a rampa: non si ha una linea sottile di 1 pixel, ma ogni punto contenuto nella rampa è un punto di egde.
- *Roof edge* (edge a tetto) è il bordo associato a una regione la cui larghezza dipende dallo spessore della linea. Si presentano ad esempio nella digitalizzazione di disegni di linee.

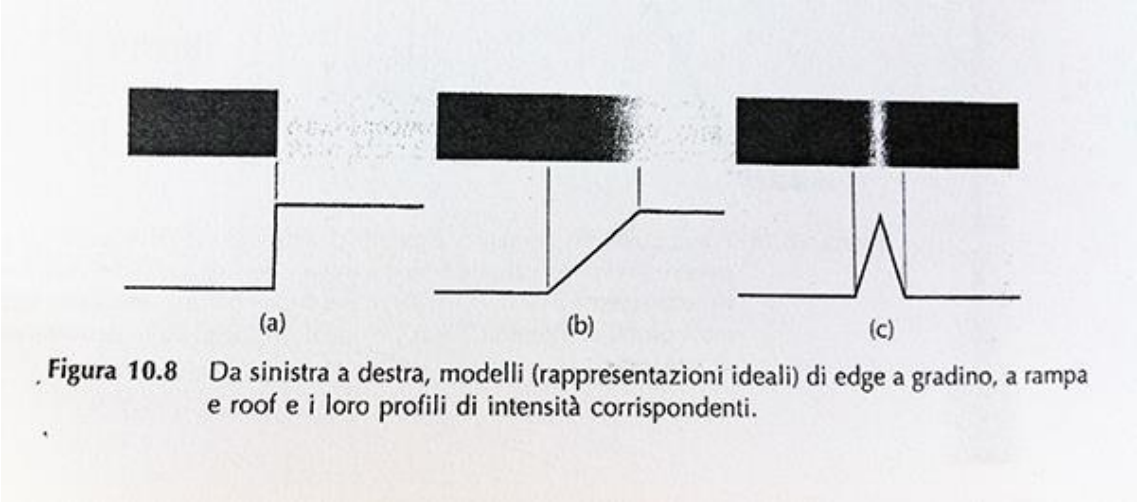


Figura 10.8 Da sinistra a destra, modelli (rappresentazioni ideali) di edge a gradino, a rampa e roof e i loro profili di intensità corrispondenti.

Uno strumento adatto per definire l'intensità e la direzione dell'edge nella posizione (x, y) di una immagine f è il **gradiente** definito come la somma delle derivate parziali $\Delta_x + \Delta_y$. Il gradiente è un vettore che punta nella direzione della maggiore variazione di I nella posizione (x, y).
Il **modulo** del vettore è dato da $\sqrt{(\Delta^2_x + \Delta^2_y)}$ ed è il valore in (x, y) del cambiamento di direzione del vettore gradiente. La **direzione** del vettore è data dall'angolo calcolato rispetto all'asse x con l'equazione $\alpha(x, y) = arctan(\Delta_y/\Delta_x)$.
Calcoliamo la derivata rispetto all'asse x partendo dalla definizione di derivata come limite del rapporto incrementale:
$\Delta_x = (g_x - g_{x0})/(x - x_0)$ dove g è la luminosità e x, x_0 due pixel. Avendo a che fare con quantità discrete si richiede quindi un'approssimazione digitale delle derivate parziali, quindi il limite di x che tende a x_0 significa che questi pixel sono adiacenti. Essendo adiacenti allora si ha che $x - x_0 = 1$, quindi possiamo scrivere $\Delta_x = (g_x - g_{x0})$, cioè la differenza fra i livelli di grigio di due pixel adiacenti.
Ottenere il gradiente di un'immagine quindi richiede il calcolo delle derivate parziali per ogni pixel dell'immagine quindi $f'(x) = f(x + 1, y) - f(x, y)$ e $f'(y) = f(x, y + 1) - f(x, y)$.
Queste due equazioni possono essere implementate per tutti i valori di x e y filtrando f(x, y) con i kernel

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix} \text{ e } [-1 \quad 1]$$

Quando però ci interessa la direzione diagonale di un edge è necessaria una matrice (sobel, prewitt etc).

Il filtro **point like edge** è un filtro che ha un valore negativo al centro e bordi uniformi e positivi.
L'effetto è quello di evidenziare i bordi luminosi delle immagini.

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

Filtro di Prewitt

Queste matrici evidenziano le componenti orizzontali e verticali; in questo caso da pixel scuri a pixel chiari:

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

Queste matrici evidenziano le componenti diagonali. Possiamo calcolare per il filtro di Prewitt tutti i possibili kernel che si ottengono effettuando rotazioni di 45°.

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

Filtri di Sobel

Le maschere di Sobel hanno migliori prestazioni rispetto a Prewitt nella gestione e nella rimozione di eventuale rumore presente nell'immagine e le rende di gran lunga preferibili a Prewitt.

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

Il filtro di Sobel ha la proprietà, come per il filtro gaussiano, di essere a variabili separabili:

1	2	1

-1
0
1

Filtri di Laplace

A differenza dei due precedenti filtri, sfrutta le derivate seconde. Infatti mentre la derivata prima produce edge piuttosto spessi, la derivata seconda dà luogo a edge più sottili ma "doppi".

- Le derivate prime danno luogo a contorni più spessi
- Le derivate seconde hanno una risposta più forte ai dettagli fini
- Le derivate prime hanno una risposta più forte agli edge ripidi
- Le derivate seconde producono una risposta doppia agli edge

Da queste considerazioni, emerge come l'uso della derivata seconda appaia in realtà più adeguato per operazioni di miglioramento di qualità, soprattutto se l'immagine contiene dettagli fini.
Calcoliamo la derivata seconda rispetto all'asse x nel caso discreto (quindi $x-x_0 = 1$):

$$\Delta^2_x = f'(f(x+1,y) - f(x,y))$$
$$= f'(f(x+1,y)) - f'(f(x,y))$$
$$= [f(x+2,y) - f(x+1,y)] - [f(x+1,y) - f(x,y)]$$
$$= f(x+2,y) - 2f(x+1,y) + f(x,y)$$
$$\Delta^2_y = f(x,y+2) - 2f(x,y+1) + f(x,y)$$

Trasliamo i risultati in modo da centrare il pixel (x, y):

$$\Delta^2_x = f(x+1,y) - 2f(x,y) + f(x-1,y)$$
$$\Delta^2_y = f(x,y+1) - 2f(x,y) + f(x,y-1)$$

Avendo centrato (x, y), possiamo esprimere queste espressioni con i filtri di convoluzione:

$$\partial^2 x = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad e \quad \partial^2 y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Sommando i 2 kernel otteniamo il filtro di Laplace:

0	1	0
1	-4	1
0	1	0

Line detection è un filtro per il rilevamento di linee orizzontali e verticali.

Rotazione e ridimensionamento

Esiste un sistema di equazioni che se vogliamo ruotare un'immagine di coordinate (x, y) di un certo angolo α, allora dobbiamo applicare un prodotto matriciale per cui otterremo due nuove coordinate (X, Y) che saranno date da

$$X = x \cos \alpha - y \operatorname{sen} \alpha \qquad Y = x \operatorname{sen} \alpha + y \cos \alpha$$

Nel ruotare l'immagine tramite queste equazioni nascono dei problemi:

1. L'immagine una volta ruotata sta in un riquadro più grande dell'immagine di partenza e questo verrà riempito con un colore predefinito, per esempio nero. Dobbiamo quindi trovare la dimensione del riquadro di output e collocare correttamente l'immagine in questo riquadro.
2. Inoltre c'è un pattern ripetuto di pixel neri (neri perché il riquadro più grande dentro cui mettiamo l'immagine ruotata la inizializziamo a zero) dovuto al fatto che questi pixel neri non sono stati toccati durante la rotazione e quindi rimangono a 0.
   Questo nasce dal fatto che stiamo facendo degli arrotondamenti perché le coordinate (x, y) sono interi, le coordinate (X, Y) sono pure interi però arrotondati.

Per risolvere questi problemi facciamo la scansione dell'immagine grande per ogni pixel (X, Y) e applichiamo la formula inversa ricavando le coordinate dell'immagine piccola; i livelli di grigio li andiamo a mettere nell'immagine grande: siccome stiamo facendo la scansione dell'immagine grande, non salteremo mai nessun pixel e non ci saranno puntini neri, ma solo zone vuote laddove l'immagine non è definita.

Questo stesso metodo lo possiamo applicare per ridimensionare l'immagine. Quando vogliamo ingrandire l'immagine, anziché fare la scansione dell'immagine piccola e proiettarla su quella grande, facciamo la scansione dell'immagine grande e ci domandiamo da dove provengono i pixel, andandoli a prendere dall'immagine piccola. Anche in questo caso avremo il compito di riposizionare i pixel in modo da ottenere l'immagine di partenza ringrandita o rimpicciolita. Nell'immagine di output ci saranno dei pixel non definiti nell'immagine originale a cui dobbiamo attribuire un valore. Esistono vari metodi per determinare il valore dell'intensità luminosa di questi pixel.

Con il termine **interpolazione** si intende il processo attraverso il quale è possibile ottenere immagini digitali ad alta risoluzione dalle controparti a bassa risoluzione. Il processo delle interpolazioni delle immagini digitali consiste nel disporre i pixel dell'immagine originale su una matrice a più alta risoluzione (con più righe e più colonne) e di stimare il valore dei pixel mancanti utilizzando quelli noti.

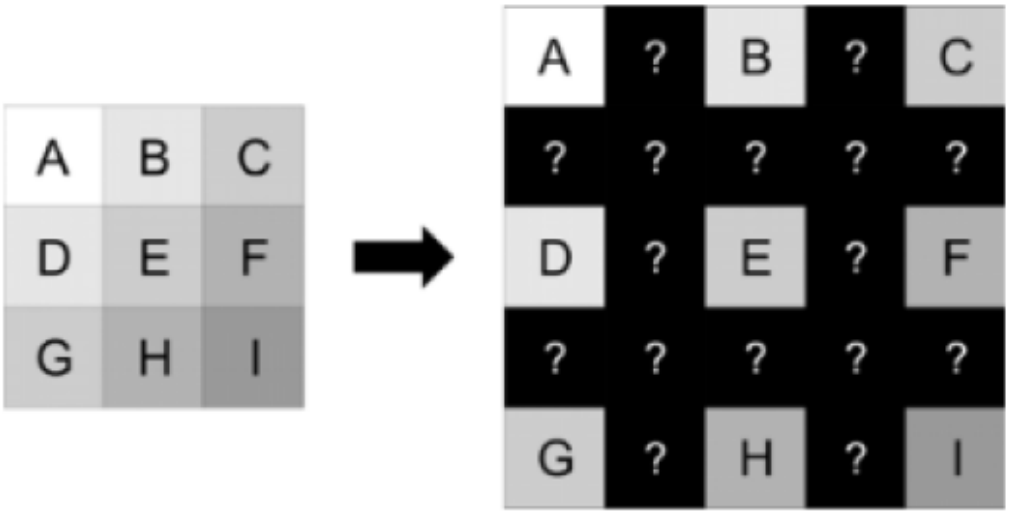


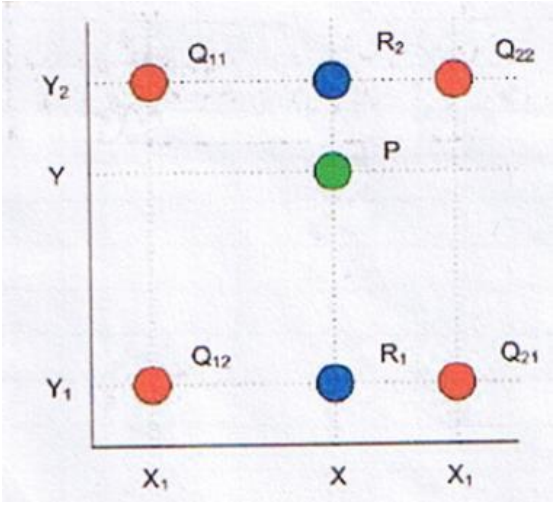
Figura 1 - Passaggio da bassa ad alta risoluzione.

I **metodi lineari** utilizzano dei filtri lineari che applicati all'immagine a bassa risoluzione, generano i pixel ad alta risoluzione. Data la loro semplicità ed efficienza, i metodi lineari sono quelli maggiormente utilizzati.
Fanno parte di questa categoria i metodi nearest neighbor, bilineare e bicubico.

Figura 3 - Comparazione tra alcuni metodi lineari (ingrandimento 4x).

Legenda
- *NN = nearest neighbor (pixel replication);*
- *BL = bilineare;*
- *BC = bicubico;*
- *OR = immagine originale da cui è stata ottenuta l'immagine sottocampionata di partenza;*

Il metodo **nearest neighbor** (il vicino più vicino) è il metodo di interpolazione più semplice e rapido. Questo metodo si limita a porre il valore di ogni pixel sconosciuto uguale al valore del pixel noto più vicino. L'immagine interpolata appare come se ogni pixel fosse cresciuto proporzionalmente al fattore di ingrandimento.
Un fattore di scala S è applicato all'immagine originaria per aumentare (S > 1) o diminuire (S < 1) le dimensioni. Facciamo una scansione dell'immagine di output in cui ad ogni pixel (X, Y) è assegnato un valore a partire dal valore del pixel (x, y) più vicino dell'immagine originaria, dove x = X / S e y = Y / S.



L'interpolazione **bilineare** stima il valore del pixel sconosciuto facendo la media dei valori dei 4 pixel noti diagonalmente adiacenti. Il risultato è un'immagine tanto più sfocata quanto maggiore è il fattore d'ingrandimento.

Questo metodo si basa sull'applicazione del metodo lineare rispetto alle due direzioni: supponiamo di voler conoscere il valore del pixel P(x, y), assumendo di conoscere i valori nei punti $Q_{11}(x_1, y_1)$, $Q_{12}(x_1, y_2)$, $Q_{21}(x_2, y_1)$ e $Q_{22}(x_2, y_2)$. Si compie prima l'interpolazione nella direzione x ottenendo R_1 ed R_2, e poi nella direzione y ottenendo il valore di P(x, y).

Il metodo **bicubico** utilizza lo stesso principio del metodo bilineare, ma anziché 4 pixel, utilizza 16 pixel noti che circondano il pixel da interpolare in una griglia 4x4. Questo metodo genera immagini più nitide rispetto al metodo bilineare, ma ha una complessità computazionale maggiore.

La **quantizzazione** consiste nel codificare i valori di un segnale analogico assegnando ad ognuno di essi un valore digitale, ovvero passando dal continuo al discreto. Questa operazione introduce un errore nella sequenza di campioni e quindi una perdita di informazione.
Nelle immagini a scala di grigio abbiamo i pixel definiti con valori da 0 a 255; la quantizzazione è la ridistribuzione dei livelli di grigio nell'intervallo 0 - 255. Nelle immagini a colori RGB a ogni pixel attribuiamo 3 byte, uno per ogni componente fondamentale rosso, verde, blu: ogni componente può assumere valori tra 0 e 255. Ogni pixel ha quindi 24 bit, e possiamo avere più di 16 milioni di colori (2^{24}); queste immagini vengono chiamate **true color**.
Riduciamo il numero di bit per pixel da 24 a 16 bit, attribuendo 5 bit per il rosso, 6 per il verde e 5 per il blu; otteniamo più di 65 mila colori (2^{16}), le immagini sono ancora true color e l'occhio umano non percepirebbe alcuna differenza.
Riduciamo ancora il numero di bit da 16 a 8 bit per pixel, attribuendo 3 bit per il rosso, 3 per il verde e 2 per il blu; otteniamo così 256 colori (2^8) visibilmente distinguibili dall'occhio umano; queste immagini vengono chiamate **indexed color**. Queste immagini, dette anche immagini indicizzate, fanno uso di una **tavolozza** di colori, in cui ad ogni colore è associato un *indice* che lo identifica. La rappresentazione dei pixel di un'immagine attraverso indici nella tavolozza permette un drastico risparmio di memoria e di tempo computazionale, ma limita la gamma di colori utilizzabili a quelli presenti nella tavolozza. Questi ultimi possono essere fissi, se determinati ad esempio dall'hardware utilizzato, oppure modificabili a seconda della fattispecie di un'immagine.
Lo spazio occupato dalla tavolozza è trascurabile e si calcola moltiplicando il numero dei colori della tavolozza per 3 byte (un byte per ogni componente RGB). Quindi una tavolozza di 256 colori occupa 256x3 = 768 byte.

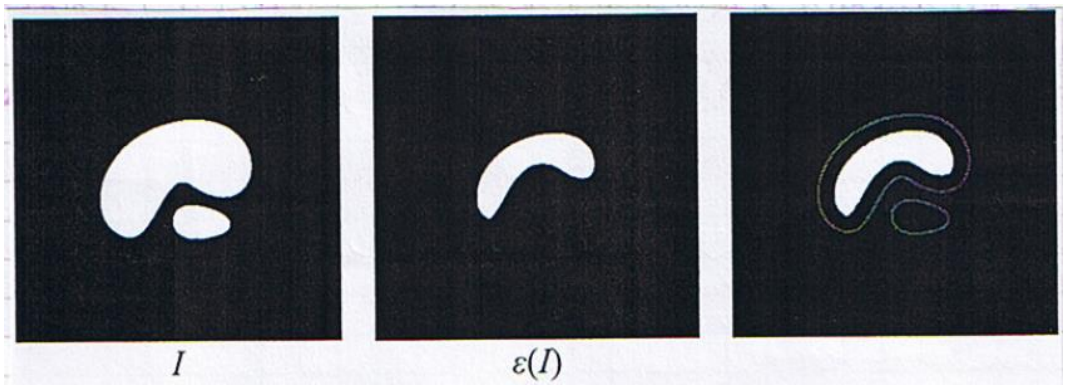
Immagine	Descrizione	Numero di colori	Memoria occupata (in byte)
True color	Ogni pixel è identificato dalla sua posizione e dalla terna RGB: (i, j, r, g, b)	Più di 16 milioni (24 bpp) Più di 65 mila (16 bpp)	w*h*3 w*h*2
Indexed Color	Ogni pixel è identificato dalla sua posizione e dall'indice che rappresenta il suo colore nella tavolozza: (i, j, k)	256 (8 bpp) 16 (4 bpp) 4 (2 bpp) 2 (1 bpp)	w*h*1 + 256*3 w*h*(1/2) + 16*3 w*h*(1/4) + 4*3 w*h*(1/8) + 2*3

Morfologia matematica

Il termine **morfologia matematica** denota lo studio della struttura geometrica dell'immagine. E' uno strumento utile per la rappresentazione e la descrizione della forma di una regione.
Obiettivo: Distinguere le informazioni significative sulla forma da quelle irrilevanti, cioè mettere in evidenza determinate strutture nelle immagini.
La struttura dell'immagine viene "sondata" con un insieme di forma definibile dall'utente (**elemento strutturante**) solitamente codificato da una piccola immagine raster (3×3 o 5×5).
Dunque attraverso la morfologia matematica si estraggono informazioni da un'immagine A ⊆ E, dove E è l'insieme di tutte le immagini di dimensione nota, attraverso l'uso di un'immagine più piccola B, detta elemento strutturante.
INPUT: immagine con strutture da evidenziare;
elemento strutturante per evidenziare queste strutture.
OUTPUT: strutture evidenziate.

Vengono definiti 4 operatori principali: dilatazione, erosione, apertura e chiusura, che combinati con diversi elementi strutturanti B trasformano un "oggetto" A in vario modo. Erosione e dilatazione sono gli operatori elementari: operatori più complessi sono definiti come combinazioni di questi ultimi.

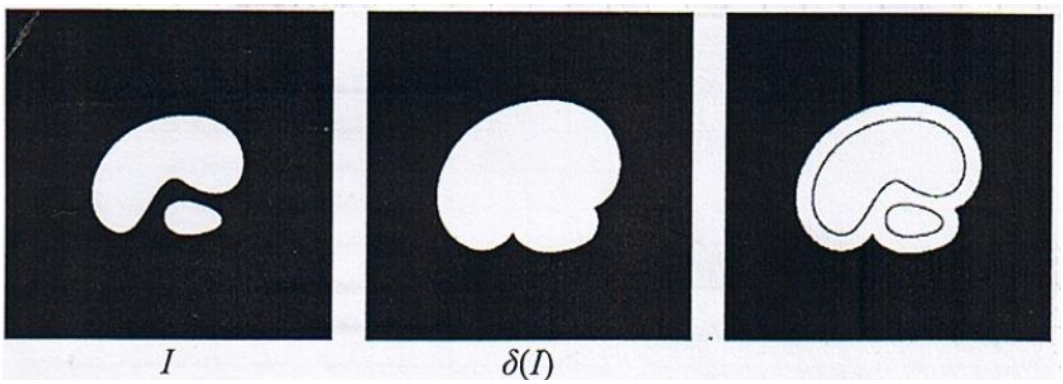
Erosione: L'operazione di erosione assottiglia gli oggetti.



$$\varepsilon_r\left(I\left(\underline{p}\right)\right)=\min_{\underline{q}\in S_r} I\left(\underline{p}+\underline{q}\right)$$

Sia I l'immagine di partenza e S l'elemento strutturante (che in questo caso ha forma circolare). Si scansiona l'immagine I con S, prendendo i valori dei pixel di I che stanno sotto S; si calcola il **minimo** di tali valori e si assegna al pixel centrale di riferimento.
Il risultato è un inscurimento dell'immagine e un assottigliamento delle strutture tanto grande quanto le dimensioni di S; inoltre le strutture più piccole di S verranno eliminate.

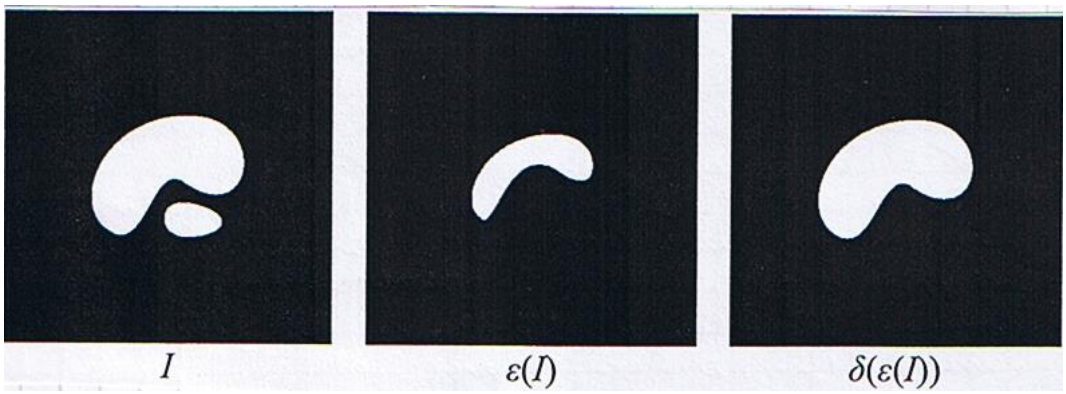
Dilatazione: L'operazione di dilatazione espande gli oggetti.



$$\delta_r\left(I\left(\underline{p}\right)\right)=\max_{\underline{q}\in S_r} I\left(\underline{p}+\underline{q}\right)$$

Sia I l'immagine di partenza e S l'elemento strutturante (che in questo caso ha forma circolare). Si scansiona l'immagine I con S, prendendo i valori dei pixel di I che stanno sotto S; si calcola il **massimo** di tali valori e si assegna al pixel centrale di riferimento.
Il risultato è uno schiarimento dell'immagine e un ingrandimento delle strutture tanto grande quanto le dimensioni di S; inoltre le strutture più piccole di S verranno inglobate in una grande.

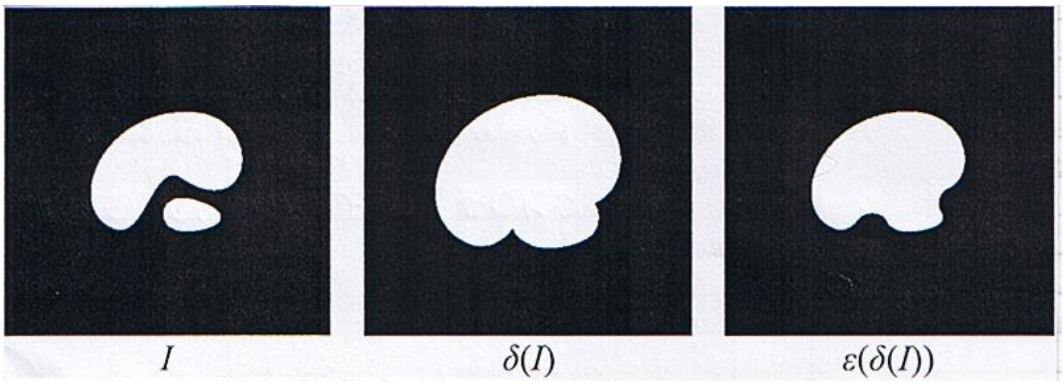
Apertura



$$\gamma\left(I\right)=\delta\left(\varepsilon\left(I\right)\right)$$

L'operazione di apertura rimuove piccoli oggetti in I, preservando la forma e la dimensione di oggetti più grandi.
Essa consiste in un'operazione di erosione seguita da una dilatazione utilizzando lo stesso elemento strutturante.

Chiusura



$$\varphi(I) = \varepsilon(\delta(I))$$

L'operazione di chiusura riempie piccoli buchi in I.
Essa consiste in un'operazione di dilatazione seguita da una erosione utilizzando lo stesso elemento strutturante.

$$\mathcal{E}(I) = \overline{\mathcal{J}(\bar{I})}$$
$$\mathcal{J}(I) = \overline{\mathcal{E}(\bar{I})}$$

Proprietà: erosione e dilatazione sono operatori **duali**, cioè l'erosione di un'immagine è equivalente al complemento della dilatazione dell'immagine complementata con lo stesso elemento strutturante, e viceversa.

Possiamo definire un **intensificatore di bordi** morfologico:

$$\rho(I) = \delta_1(I) - \varepsilon_1(I)$$

Dato un elemento strutturante di raggio 1, eseguiamo la dilatazione dell'immagine I; eseguiamo a parte l'erosione dell'immagine I attraverso lo stesso elemento strutturante.
Facendo la differenza otteniamo i bordi dell'immagine.
Osserviamo che i risultati sono migliori dell'applicazione del filtro di Sobel.

Gli operatori γ e φ possono essere combinati per ottenere gli operatori **top-hat** (th) e **bottom-hat** (bh):

$$th(I) = I - \gamma(I) \qquad bh = \varphi(I) - I$$

Miglioriamo il **contrasto** I tramite la formula:

$$K(I) = \max\{0, \min\{255, I + th(I) - bh(I)\}\} = \max\{0, \min\{255, 3 \times I - \gamma(I) - \varphi(I)\}\}$$

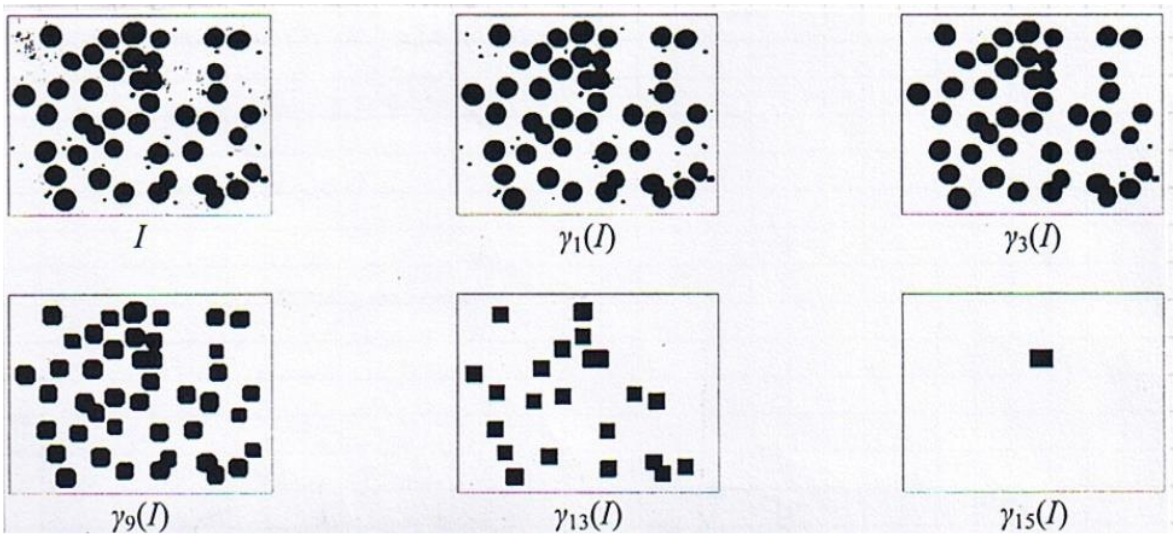
la funzione max serve per assicurarci che non scendiamo sotto lo zero.
Aumentando il raggio dell'elemento strutturante si aumenta il contrasto.

Il **minimo rettangolo di ricoprimento** è il più piccolo rettangolo che contiene gli oggetti.
Si applica 4 volte l'erosione all'immagine I con i seguenti elementi strutturanti:

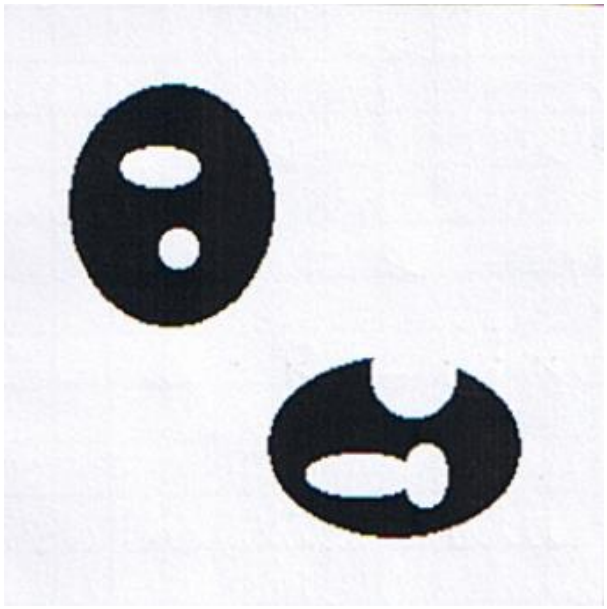
$$S_1 = \begin{bmatrix} 1 & 1 \\ 1 & \boxed{0} \end{bmatrix} \quad S_2 = \begin{bmatrix} 1 & 1 \\ \boxed{0} & 1 \end{bmatrix} \quad S_3 = \begin{bmatrix} \boxed{0} & 1 \\ 1 & 1 \end{bmatrix} \quad S_4 = \begin{bmatrix} 1 & \boxed{0} \\ 1 & 1 \end{bmatrix}$$

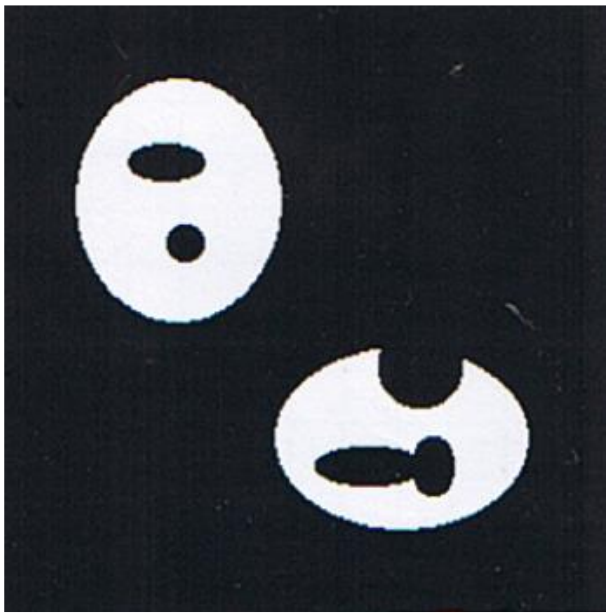
Unendo i risultati allarghiamo l'immagine fino ad ottenere il rettangolo.

La **granulomentria** si utilizza in ambito medico ad esempio per vedere il numero di globuli rossi o che forma hanno. Consiste nell'applicare all'immagine l'apertura utilizzando un elemento strutturante con raggio via via più grande.



Il **numero di connettività** NC di un'immagine binaria restituisce una prima informazione sulla struttura delle sue componenti. Si definisce NC come la differenza tra il numero di **componenti connesse** e il numero di **buchi**: **NC = #C - #B**





In questo caso ho 2 componenti connesse e 3 buchi.
I buchi si possono vedere come le componenti connesse della negazione dell'immagine.

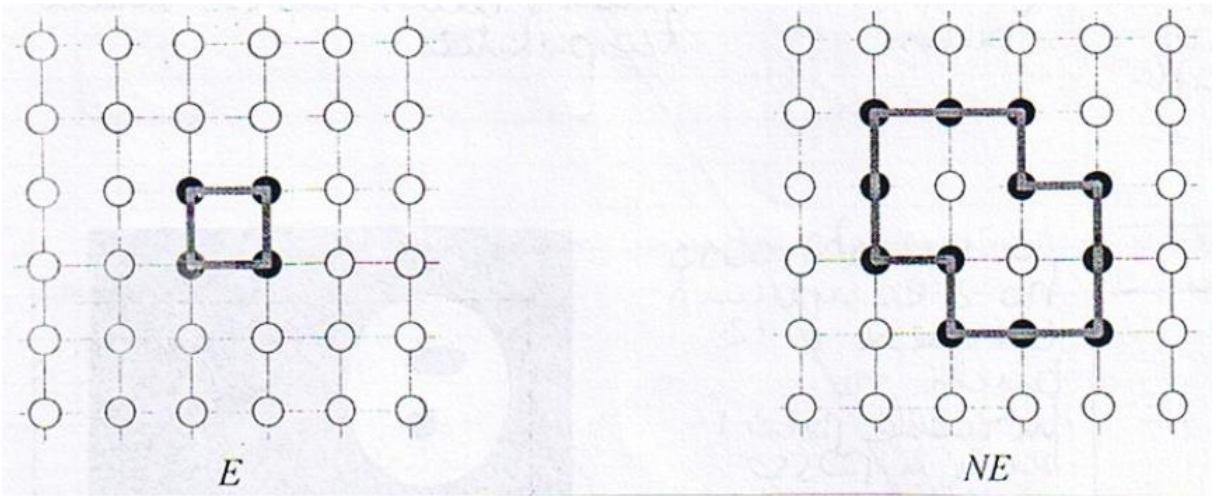
Un grafo planare è un grafo che può essere raffigurato in un piano in modo che non si abbiano archi che si intersecano. Il numero delle componenti connesse è data dalla formula di Eulero:

$$\#C = \#V + \#F - \#A \qquad \text{(Componenti connesse = Vertici + Facce – Archi)}$$

In particolare, nelle immagini binarie possiamo considerare *elementari* le facce delimitate da archi diretti e *non elementari* tutte le altre facce: $\#F = \#E + \#NE$ quindi si ha che

$$\#C = \#V + \#E + \#NE - \#A$$

Possiamo inoltre osservare che le facce elementari sono i gruppi di 4 pixel fortemente connessi a due a due, mentre le facce non elementari sono i buchi delle componenti connesse ($\#B = \#NE$):



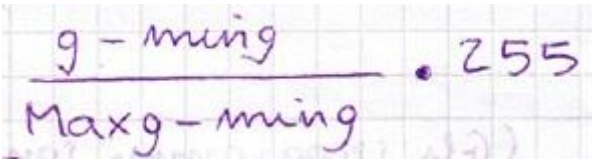
Quindi possiamo scrivere: **$\#NC = \#V + \#E + \#NE - \#A - \#NE = \#V + \#E - \#A$**

Istogrammi e Thresholding

L'**istogramma** è un grafico che rappresenta la distribuzione dei valori di ogni pixel di un'immagine (ascissa = valore del pixel da 0 a 255; ordinata = numero di pixel).
Quindi l'istogramma ci dice quanti pixel ci sono con un determinato valore.
Possiamo creare l'istogramma della luminanza considerando i livelli di grigio dei pixel, ma anche i livelli delle tre componenti RGB.
Uniamo i tre istogrammi e *normalizziamo*, ovvero ci concentriamo sui punti più significativi, tagliando sostanzialmente i picchi (outlier).

Gli istogrammi possono servire ad esempio per le basi di dati di immagini; infatti se ci ricaviamo gli istogrammi di queste immagini potremmo cercare di confrontarli ad esempio nelle ricerche, riducendo così il confronto tra immagini in confronto tra istogrammi.

Lo **stretching** è una tecnica che dilata l'istogramma in modo da coprire l'intera gamma dei grigi. In questo modo si introducono dei "buchi" nel nuovo istogramma perché ci sono livelli di grigio non utilizzati.
Inoltre in output l'immagine presenta un contrasto maggiore.
In pratica prendiamo il massimo valore e lo portiamo a 255, mentre il minimo lo portiamo a 0.
Gli altri valori si calcolano considerando la proporzione: $min_g : 0 = max_g : 255$
Quindi ogni livello di grigio g si calcola facendo



L'**equalizzazione** è una tecnica che distribuisce meglio i livelli di grigio per sfruttare al meglio la luminosità. Questa operazione produce un nuovo istogramma più uniformemente distribuito, aumenta il contrasto e migliora la visualizzazione dell'immagine.
Come lo stretching, modifica i livelli di grigio dell'immagine secondo una LUT che comprende l'intero range di variabilità [0, 255].

La **sogliatura** o **thresholding** è un metodo per segmentare un'immagine.
Data un'immagine a livello di grigio, la sogliatura restituisce un'immagine binaria.
Durante questo processo si creano due insiemi:

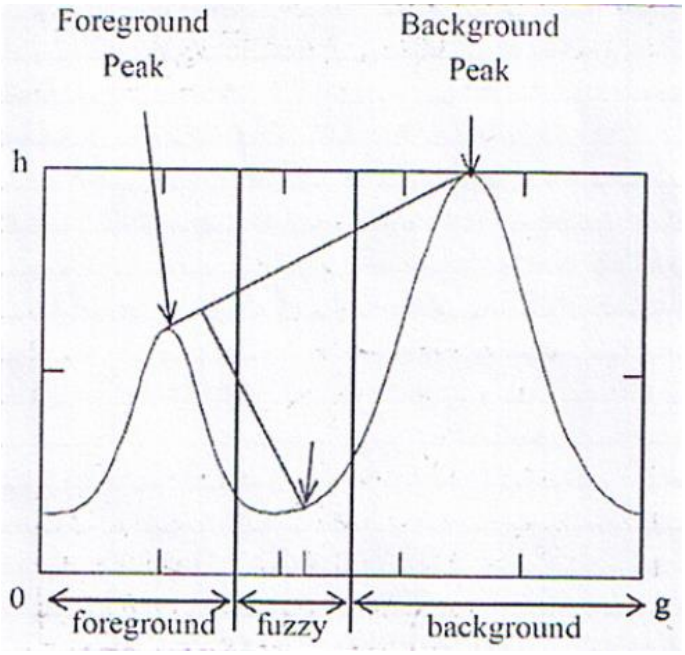
- Il primo piano (*foreground*) contiene i pixel il cui valore è minore di una certa soglia t
- Lo sfondo (*background*) contiene i pixel il cui valore è maggiore di t

In generale il threshold t dipende non solo dal livello di grigio g del singolo pixel p, ma anche dalla posizione (x, y) e dall'intorno N del pixel.

Esistono diversi metodi per la scelta del threshold:

- La media o la mediana sono un buon valore in un'immagine senza rumore
- Ottimale
- Iterativo
- Dinamico

Threshold ottimale: un threshold per istogrammi bimodali (cioè con 2 picchi) può essere ricavato geometricamente, partendo dal segmento di massima distanza tra la curva dell'istogramma e la congiungente i suoi picchi.



Si traccia prima il segmento che unisce i due picchi. Poi si traccia la perpendicolare a questo segmento: ci sarà solo un punto in cui questa perpendicolare avrà lunghezza massima (se non fosse bimodale ce ne sarebbero più di uno).
Dopodiché basta fare la proiezione di questo punto di intersezione sui valori di grigio g e troveremo il valore del threshold ottimale.

Threshold iterativo: un threshold per un dato istogramma, relativo ad un'immagine con due oggetti, può essere facilmente ricavato minimizzando iterativamente la varianza all'interno dei due insiemi (foreground e background). Inizialmente l'immagine è segmentata arbitrariamente in due parti e sono calcolati i rispettivi valori medi x_1 e x_2. Il nuovo threshold t è posto uguale alla media di x_1 e x_2 e la segmentazione è ricalcolata.
Il procedimento termina quando il valore di t si stabilizza o dopo un numero massimo di iterazioni.

Threshold dinamico: l'algoritmo è descritto nel seguente modo:

1. Suddividere l'immagine in blocchi di 7x7 pixel. Per i blocchi che presentano un istogramma bimodale può essere applicato l'algoritmo precedente per ricavare un insieme di valori di sogliatura.
2. I valori di sogliatura dei blocchi che non presentano un istogramma bimodale sono ricavati per interpolazione da quelli già calcolati.
3. I valori così ottenuti per tutti i blocchi sono assegnati ai pixel nel centro dei blocchi.
4. Ai restanti 48 pixel di ciascun blocco sono assegnati valori di sogliatura ricavati ancora una volta tramite interpolazione.

Spesso l'istogramma non è bimodale oppure è bimodale, ma mascherato da rumore; in questi casi possiamo calcolare una **sogliatura adattiva**, che consiste nello scegliere per differenti regioni nell'immagine un threshold diverso.

Compressione e segmentazione

La **compressione** è una tecnica preposta alla riduzione del numero di bit necessari per immagazzinare un'informazione e ridurre quindi le dimensioni di un determinato file. Le varie tecniche di compressione cercano di organizzare in modo più efficiente le informazioni al fine di ottenere una memorizzazione che richieda minor uso di risorse.
Gli algoritmi di compressione si dividono in due categorie:

- **Lossless**: algoritmi che comprimono i dati senza perdita di informazioni, sfruttando le ridondanze nella codifica dei dati. Questi algoritmi si preoccupano di preservare il messaggio originale durante la compressione.
  Un esempio è il formato *zip* per i file e *gif* per le immagini; a partire da un file di uno di questi formati, è sempre possibile ricostruire esattamente il file d'origine.
- **Lossy**: algoritmi che comprimono i dati con perdita di informazioni, sfruttando le ridondanze nell'utilizzo dei dati. Questi algoritmi sono più efficaci in termini di compressione, a scapito però dell'integrità del file; infatti il file compresso è simile, ma non identico al file originale. Di solito vengono utilizzati per comprimere file multimediali, che spesso sono troppo grandi per essere facilmente trasferiti o memorizzati; quindi si preferisce avere una piccola riduzione della qualità, ottenendo però file molto più leggeri. Esempi sono il formato *jpeg* (per le compressioni d'immagini) e il formato *mpeg* (per le compressioni di audio e video).

Per quanto riguarda le immagini **non compresse**, queste richiedono un'elaborazione minima, non essendo necessari algoritmi di compressione (in fase di scrittura) e decompressione (in fase di lettura). Tuttavia, mancando di compressione, risultano particolarmente voluminose, in termini di spazio occupato, rispetto agli altri formati.
Un esempio è il formato *raw* usato da alcune fotocamere digitali, che memorizzano l'immagine proveniente dal sensore, senza perdita di informazioni.

Gli algoritmi di compressione per immagini digitali devono tener conto non solo del fattore di compressione, ma anche dell'errore eventualmente introdotto.
Le misure di compressione normalmente usate misurano:

- **Bit per pixel**: $bpp = C/N$
  $bpp = 1.00$ → l'immagine occupa 1/8 rispetto all'originale
  $bpp = 0.50$ → per ogni pixel ho mezzo bit, ovvero ogni bit codifica 2 pixel
  $bpp = 0.25$ → ogni bit codifica 4 pixel
- **Rapporto di compressione**: $ratio = kN/C$

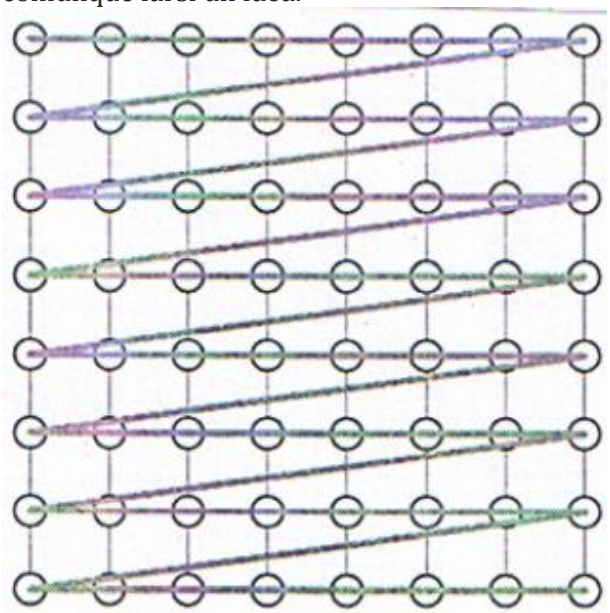
dove **C** indica la **dimensione in bit** del file compresso, **N** indica il **numero di pixel** e **k** è il **numero di bit per pixel** nell'immagine originale.
Le informazioni aggiuntive come la dimensione dell'immagine, la risoluzione ecc. che sono indicate nell'intestazione (**header**) del file occupano uno spazio trascurabile e solitamente non sono incluse nei calcoli.

L'errore (distorsione) introdotto dagli algoritmi di compressione lossy è quantitativamente misurabile, per esempio, tramite l'**errore quadratico medio** (**MSE**, mean square error) che è pari a

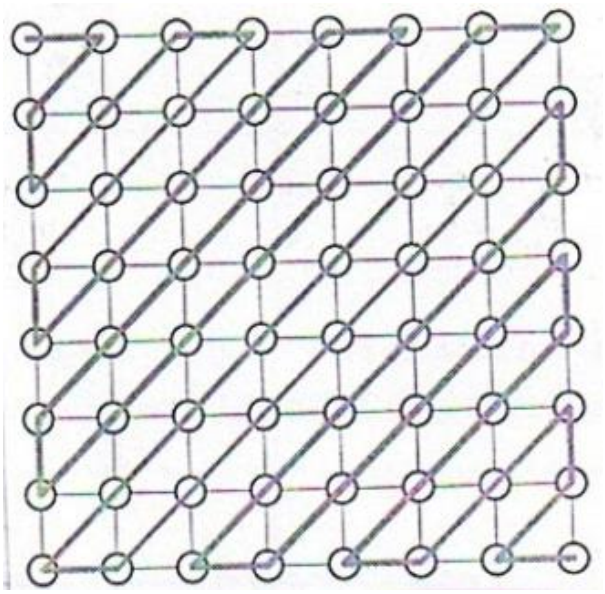
$$\frac{1}{N}\sum_{i=1}^{N}(g_i - g_i')^2$$

dove g_i' è il livello di grigio del pixel corrispondente nell'immagine compressa.
Purtroppo questa misura non coincide con la normale valutazione personale, infatti l'occhio umano non confronta i singoli pixel, ma permette una stima qualitativa della distorsione globale.
Inoltre queste misure non confrontano artefatti come i blocchi o le sfocature.
Se due immagini differiscono per un certo numero k di pixel abbastanza piccolo, il nostro occhio le percepisce come uguali.

Molto importante è il modo in cui le immagini vengono lette per decomprimerle; molto dipende dal modo in cui vengono scanditi i pixel dell'immagine. Chi riceve un'immagine vuole via via già vederla o comunque farsi un'idea.

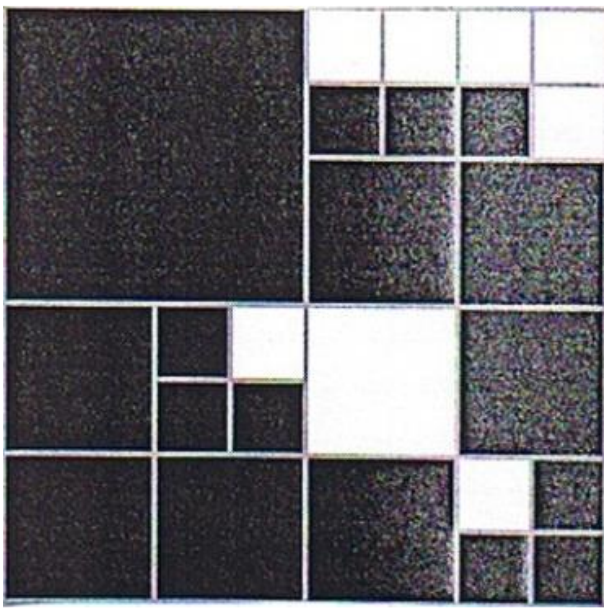


Con la scansione raster (riga per riga) l'immagine avrà senso una volta scaricata tutta.



Con la scansione Jpeg si ha un andamento a zig zag. In questo modo possiamo capire di che immagine si tratta già all'8,3% della scansione.

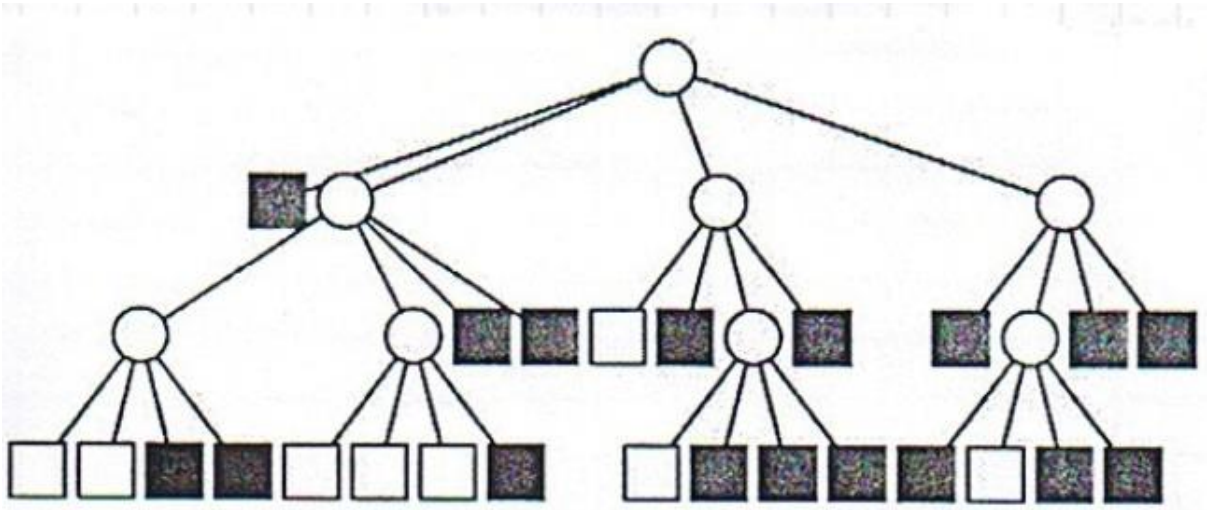
La **segmentazione di un'immagine** è il processo di partizionare un'immagine in regioni significative. Viene utilizzata per ottenere una rappresentazione più compatta, estrarre degli oggetti o comunque come strumento per l'analisi delle immagini e permette di partizionare le immagini digitali in insiemi di pixel. Lo scopo della segmentazione è semplificare la rappresentazione delle immagini in qualcosa che è più significativo e facile da analizzare.
La segmentazione è di solito utilizzata per localizzare oggetti e bordi (linee, curve, ecc.). Più precisamente, la segmentazione è il processo con il quale si classificano i pixel dell'immagine che hanno caratteristiche comuni, pertanto ciascun pixel in una regione è simile agli altri della stessa regione per una qualche proprietà o caratteristica (colore, intensità). Regioni adiacenti sono significativamente differenti rispetto ad almeno una di queste caratteristiche. Il risultato di un'immagine segmentata è un insieme di segmenti che, collettivamente, coprono l'intera immagine.



Un **quadtree** è una struttura dati ad albero non bilanciata nella quale tutti i nodi interni hanno esattamente 4 nodi figli.
Sono usati per partizionare un'immagine suddividendola ricorsivamente in 4 quadranti.

I quadtree sono strutture piramidali in cui l'immagine è divisa in 4 quadranti; per ogni quadrante si controlla se è uniforme: se non lo è si ripete il procedimento per quel quadrante fino al raggiungimento di zone uniformi (al massimo fino ad arrivare al singolo pixel).
Per suddividere i quadranti si parte da quello in alto a sinistra e si prosegue in senso orario.

Esiste una corrispondenza tra i quadtree e i 4-alberi:

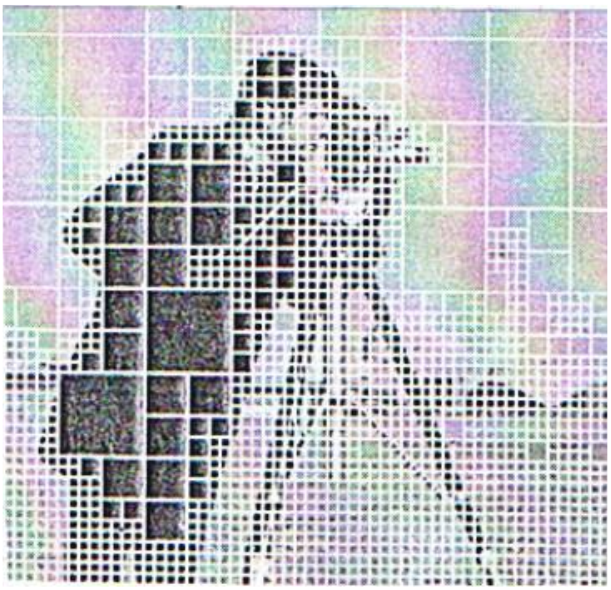


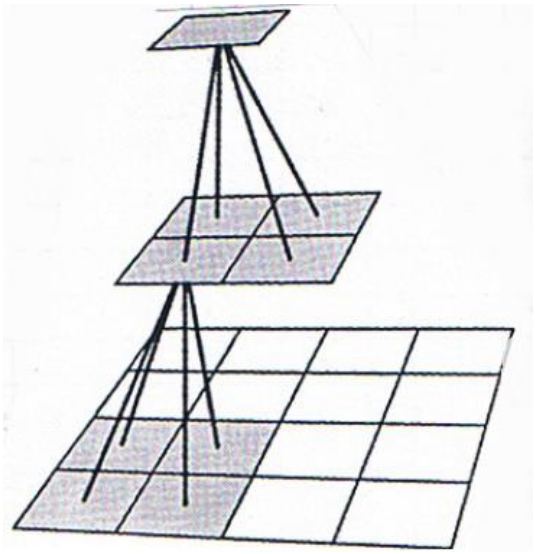
Quando giungiamo alle foglie (quadrati) allora non dobbiamo più suddividere e il colore della foglia indica quello del quadrante uniforme, che quindi non ha più bisogno di essere suddiviso.

I quadtree possono essere usati per la segmentazione. Una volta suddivisa (*split*) l'immagine in quadranti, possiamo fondere (*merge*) i quadranti che hanno le stesse proprietà secondo la 4-connettività o la 8-connettività.

I quadtree possono essere usati per rappresentare le immagini binarie con un codice in genere non particolarmente efficiente. Per esempio data un'immagine binaria precedente di 8x8 pixel (64 bit). Una volta creato il quadtree, codifichiamo l'albero in modo che un nodo interno sia rappresentato da 0, una foglia nera da 10 e una bianca da 11. Analizzando l'albero con una visita BFS (per ampiezza) otteniamo un codice prefisso, cioè un codice non ambiguo di una sola parola. In questo modo possiamo rappresentare l'immagine con un codice di 58 bit (compressione esigua!).

Per estendere i quadtree alle immagini a livelli di grigio, si definisce un *criterio di uniformità*, basato sulla varianza dei pixel nel medesimo blocco: se superiore a una data soglia, il blocco deve essere suddiviso. In pratica si fissa un valore di soglia: se la varianza è al di sopra di questa soglia, allora il quadrante non è uniforme e va suddiviso. All'aumentare della soglia, diminuiscono i blocchi e quindi saranno più evidenti gli artefatti.





I quadtree sono utili anche per rappresentare le immagini con differenti risoluzioni. Nella **struttura piramidale** i dettagli sono memorizzati alla base della piramide, mentre i livelli superiori permettono un'analisi più veloce. Salendo di livello ogni 4 pixel ne otteniamo uno e i livelli di grigio passano dai figli al padre tramite ad esempio le funzioni and, xor, min, max, media, mediano.

Proprietà:

- le foglie rappresentano i pixel
- ogni nodo di livello l contiene le coordinate del primo pixel in alto a sinistra
- indichiamo con (i, j, 2^l) la sottomatrice di dimensione 2^l identificata dal pixel (i, j)

Problemi:

- dipendenza dalla dimensione dell'immagine
- perdita di connettività
- immagini simili possono avere quadtree completamente diversi; basta avere un solo pixel diverso e i quadtree saranno diversi, perché il blocchetto cha cambia si propaga a cascata.

Il formato JPEG

JPEG specifica come una immagine può essere trasformata in una sequenza di byte. Il metodo di compressione jpeg è basato sull'uso della trasformata discreta del coseno (DCT) con compressione di tipo **lossy**, cioè con perdita di informazione.
Il JPEG opera in 3 passi fondamentali per trasformare un'immagine raster in una JPEG e viceversa:

- Rappresentazione in ambito frequenziale tramite DCT (trasformata discreta del coseno).
- Quantizzazione effettuata tramite opportune matrici di quantizzazione.
- Codifica entropica ed eliminazione delle ridondanze di tipo statistico tramite codifica RLE e codici di Huffman.

L'obiettivo è sfruttare le ridondanze nei dati dell'immagine per fornire la compressione.
In altre parole, si riduce l'entropia, che nel nostro caso significa diminuzione del numero medio di bit necessari per rappresentare l'immagine.
Nuovi metodi lossy, in particolare basati sulle DWT (Discrete Wavelet Transform), garantiscono migliori risultati in alcuni casi. Il comitato JPEG ha creato un nuovo standard basato su wavelet, JPEG 2000, con la prospettiva di sostituire nel tempo lo standard JPEG.

La **Trasformata Discreta del Coseno** (**DCT**) è la procedura matematica in grado di decomporre l'immagine in un insieme di forme d'onda elementari, ognuna con una particolare frequenza
La DCT si basa sul presupposto che i pixel in un'immagine mostrano un certo livello di correlazione con i pixel vicini. Di conseguenza, queste correlazioni possono essere sfruttate per prevedere il valore di un pixel dai suoi rispettivi vicini. La trasformata definisce quindi un modo per mappare questi dati spaziali (correlati) in coefficienti trasformati (non correlati). Infatti la DCT scorrela i dati dell'immagine, riducendo la ridondanza fra i pixel, in modo da *impacchettare* l'informazione, sul minimo numero possibile di coefficienti. L'obiettivo è quello di rappresentare il segnale in un dominio in cui la maggior parte dei coefficienti sia prossima a zero. Questa è un'operazione senza perdita di informazioni, dunque la trasformazione inversa rende una perfetta ricostruzione dell'immagine originale.

La definizione più comune di DCT di una sequenza 1-D di lunghezza N è

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{\pi(2x+1)u}{2N}\right]$$

per u = 0, 1, 2, … , N -1.
Analogamente, la trasformata inversa (IDCT) è definita come

$$f(x) = \sum_{u=0}^{N-1} \alpha(u) C(u) \cos\left[\frac{\pi(2x+1)u}{2N}\right]$$

per x = 0, 1, 2, … , N -1.
In entrambe le equazioni α(u) è definita come

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & for \quad u = 0 \\ \sqrt{\frac{2}{N}} & for \quad u \neq 0. \end{cases}$$

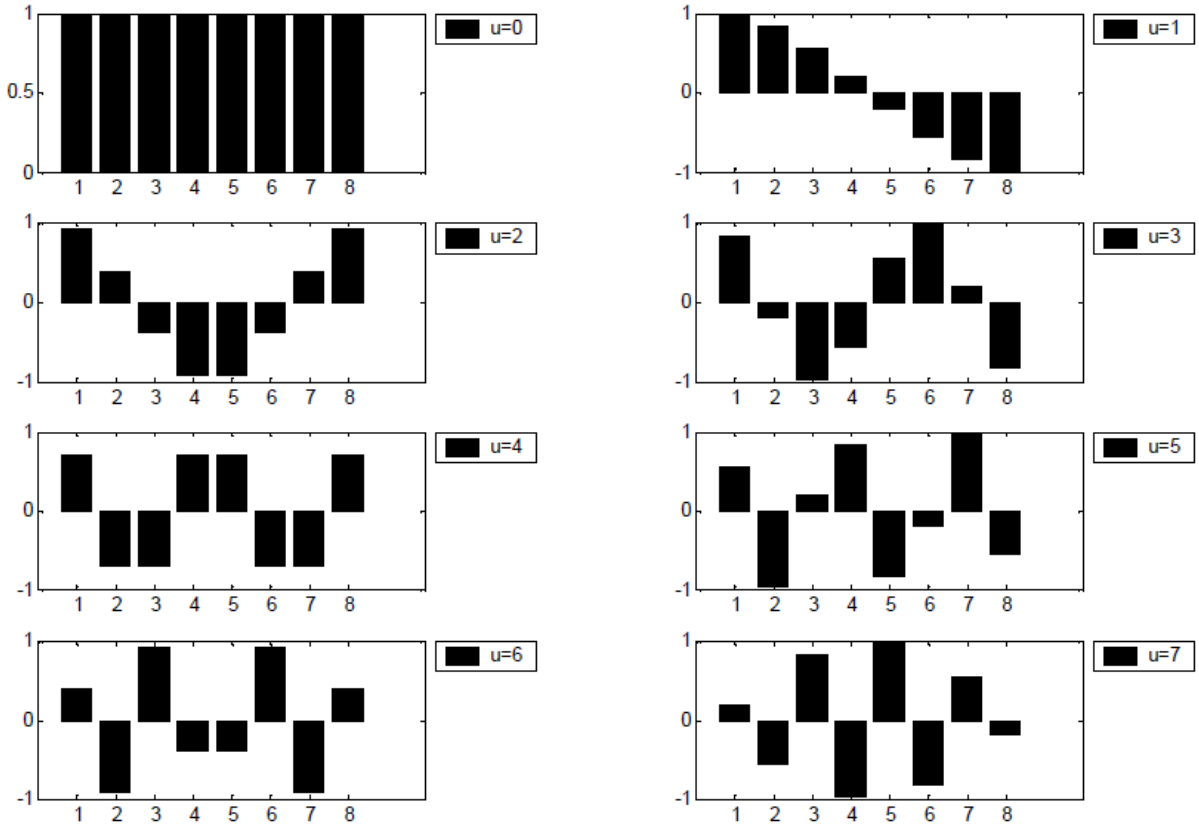
Per u = 0 si ha

$$C(u=0)=\sqrt{\frac{1}{N}}\sum_{x=0}^{N-1} f(x)$$

Così, il primo coefficiente della trasformata è il valore medio della sequenza campione.
Per fissare meglio le idee, ignoriamo *f*(x) e α(u):

$$\sum_{x=0}^{N-1}\cos\left[\frac{\pi(2x+1)u}{2N}\right]$$

Grafici per N = 8 variando i valori di *u*:



La prima forma d'onda in alto a sinistra (u = 0) rende un valore costante, mentre tutte le altre forme d'onda (u = 1, 2, … , 7) danno forme d'onda in progressivo aumento delle frequenze.
Queste forme d'onda sono chiamate **funzioni base coseno**. Si noti che queste funzioni base sono ortogonali. Quindi, la moltiplicazione di qualsiasi forma d'onda con un'altra forma d'onda seguita da una sommatoria su tutti i punti campione produce un valore zero (scalare), mentre la moltiplicazione di qualsiasi forma d'onda con sé stessa seguita da una sommatoria produce un valore costante.
Forme d'onda ortogonali sono indipendenti, cioè nessuna delle funzioni base può essere rappresentata come una combinazione di altre funzioni base.
Se la sequenza di ingresso ha più di N punti di campionamento, allora può essere suddivisa in sotto-sequenze di lunghezza N e la DCT può essere applicata a tali blocchi in modo indipendente. Qui, un punto molto importante da notare è che in ogni tale calcolo i valori dei punti funzione base non cambieranno. Solo i valori di *f*(x) cambieranno in ogni sotto-sequenza. Questa è una proprietà molto importante, poiché dimostra che le funzioni base possono essere pre-calcolate e dopo moltiplicate con le sotto-sequenze. Questo riduce il numero di operazioni matematiche (moltiplicazioni e addizioni) generando in tal modo l'efficienza di calcolo.

Esempio (slide): Consideriamo un segnale f(x) in 8 campioni adiacenti rappresentati su 8 bit, cioè con valori da 0 a 255. Dopo aver sottratto ai campioni il valore 128 (ottenendo valori tra -128 e 127), possiamo decomporre questa sequenza in un insieme di forme d'onda di frequenza spaziale crescente. Queste forme d'onda sono tra loro ortogonali, quindi indipendenti e possono essere tra loro utilizzate per ottenere gli 8 campioni tramite combinazione lineare; gli 8 coefficienti moltiplicativi (coefficienti del coseno) rappresentano l'uscita della DCT a 8 punti.
Estendiamo questi concetti ad uno spazio **bidimensionale**:

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

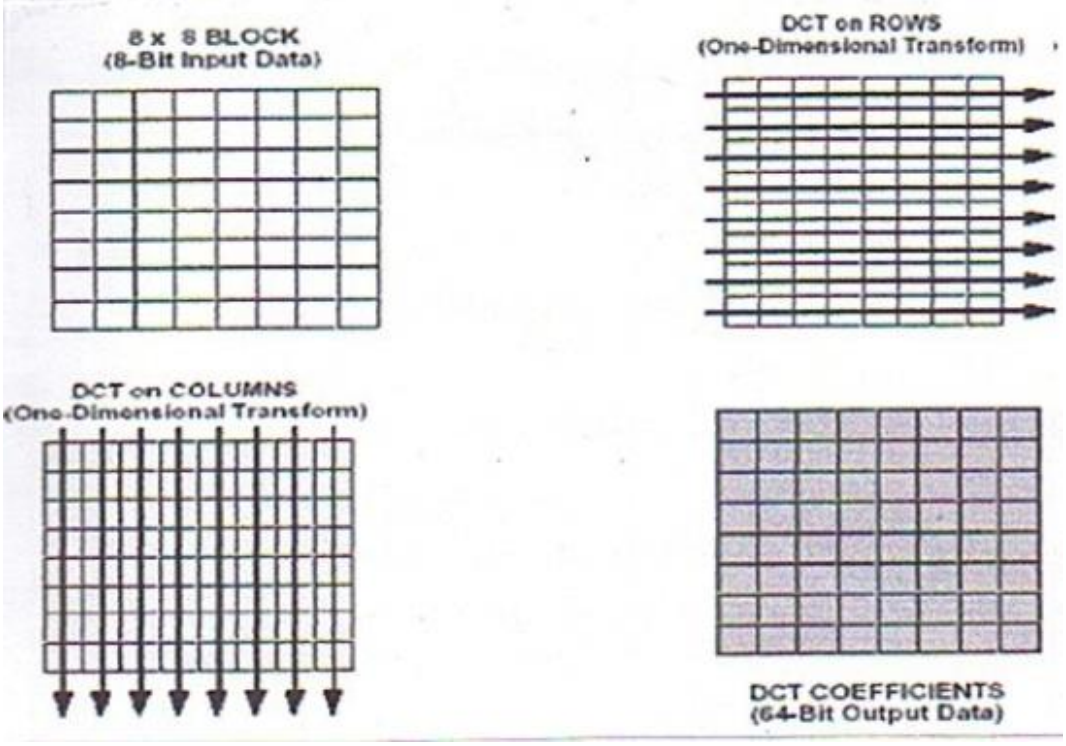
per u, v = 0, 1, 2, … , N -1. La trasformata inversa è definita come

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u,v)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

per x, y = 0, 1, 2, … , N -1. Le funzioni di base 2-D possono essere generate moltiplicando le funzioni base 1-D orientate orizzontalmente con l'insieme orientato verticalmente delle stesse funzioni.
Fruttando la proprietà di separabilità l'equazione della 2D DCT si può esprimere come

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\cos\left[\frac{\pi(2x+1)u}{2N}\right]\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

C (u, v) può quindi essere calcolato in due fasi successive: possiamo applicare una volta per x e una volta per y le formule del caso monodimensionale. La 2D DCT lavora scomponendo l'immagine in blocchi di 8x8 pixel e per ogni blocco si applica la trasformazione prima per righe e poi per colonne.



Quantizzazione: operazione che riduce l'accuratezza con la quale i coefficienti DCT sono rappresentati quando vengono arrotondati a numeri interi.
Dopo aver applicato la DCT a un blocco 8x8, ogni elemento del blocco viene diviso per un coefficiente e arrotondato all'intero più vicino: questo procedimento porta alla perdita di informazioni.
Le matrici contenenti i 64 coefficienti di quantizzazione pesano i coefficienti che rappresentano le basse frequenze in maniera più decisa, in quanto il sistema visivo umano percepisce maggiormente le basse frequenze rispetto alle alte frequenze.
In un'immagine jpeg le basse frequenze corrispondono a colori che cambiano in modo lento e graduale, le alte frequenze corrispondono invece a cambiamenti fini e particolareggiati, ovvero ai dettagli. Per questo conviene mantenere i coefficienti alle basse frequenze il più uguali possibile, mentre quelli alle alte frequenze possono essere modificati.
Quindi la tabella dovrà avere la parte in alto a sinistra con valori più bassi e in basso a destra con valori più alti, dato che in alto stanno le basse frequenze e in basso le alte frequenze.
Dopo alcuni studi sono state rese note le tabelle che danno i risultati migliori:

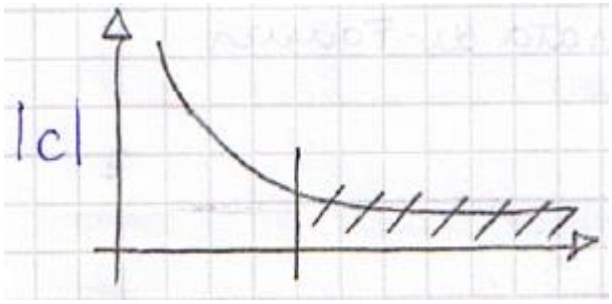
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

tabella di quantizzazione per luminanza

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

tabella di quantizzazione per crominanza

Dopo il processo di quantizzazione si ottengono dei numeri che nella loro rappresentazione binaria hanno gli ultimi bit posti a zero: più bit meno significativi posti a 0 ci sono, più si può comprimere l'immagine.
Possiamo immaginare un grafico dei coefficienti in cui l'ampiezza di C ha alti valori per le alte frequenze, mentre tende a zero per le basse frequenze:



Per comprimere introduciamo quindi un valore di soglia S e lì tagliamo in modo da trasmettere solo alcuni coefficienti e gli altri considerarli nulli.
Nel ricostruire il segnale introduciamo quindi un errore dato dalla perdita di informazioni nel processo di quantizzazione. Infatti un'immagine ricostruita usando un alto fattore di compressione mostra artefatti a quadratini dovuti proprio alla suddivisione in blocchi.

Entropia: per ridurre il numero di bit necessari per rappresentare l'immagine, i coefficienti del blocco 8x8 ottenuti applicando la DCT e la quantizzazione vengono codificati usando due tecniche lossless:

- Prima si applica l'algoritmo **RLE (Run Length Encoding)** che cerca nei dati una serie di elementi uguali e la sostituisce con un solo elemento, quindi un carattere speciale e infine il numero di volte che esso va ripetuto. Per esempio se abbiamo una sequenza di 100 zeri, l'RLE memorizzerà il primo 0, poi metterà il carattere speciale e in seguito memorizzerà il numero 100: così invece di occupare cento locazioni la prima riga ne occuperà solo 3. Il carattere speciale serve a distinguere un elemento normale da uno compresso.
  In questo modo si riescono a rappresentare in modo efficiente le lunghe sequenze di zeri che si ottengono inevitabilmente con la quantizzazione.
- Successivamente si applica l'**algoritmo di Huffman**: questo è usato per la compressione di dati, cercando di trovare il sistema ottimale per codificare stringhe, basato sulla frequenza di ciascun carattere. Si cerca di codificare i caratteri con maggior frequenza con dei codici binari più brevi rispetto a quelli utilizzati per caratteri con minor frequenza. In questo modo le lettere più frequenti in un testo verranno rimpiazzate da codici di bit più corti.
  L'algoritmo di Huffman calcola dei nuovi codici da assegnare alle lettere costruendo un albero binario in cui le lettere più frequenti siano posizionate più vicino alla radice rispetto a quelle con minore frequenza. Una caratteristica particolare di questo algoritmo è che il codice ottenuto per ogni singola lettera NON è un prefisso di un'altra lettera.

STFT (**Short Time Fourier Trasform**): Nei segnali non stazionari facendo la trasformata di Fourier otteniamo un solo picco che rappresenta il segnale, ma non dice nulla sulle diverse frequenze che lo compongono, cioè su com'è fatto questo segnale. Occorre quindi inserire nella trasformazione una dipendenza dal tempo; il modo più immediato consiste nel rendere locale la trasformata di Fourier, non operando su tutto il supporto del segnale, cioè da $-\infty$ a $+\infty$, ma su porzioni di esso (per questo si chiama "short time").
La trasformata di Fourier a tempo breve fornisce così una misura complessiva del peso delle diverse frequenze nel segnale in esame, in tutta la sua durata.
Nella STFT anziché fare la convoluzione con la curva del coseno, si fa la convoluzione con una gaussiana: in questo modo sapremo se una certa frequenza è presente e quando sarà presente nel segnale. Questa trasformata che utilizza una gaussiana è anche detta **trasformata di Gabor**.

Wavelet

Discrete Wavelet Transform (**DWT**): La trasformata discreta wavelet rappresenta un segnale mediante l'uso di una forma d'onda oscillante di lunghezza finita detta *wavelet madre*. Questa forma d'onda è scalata e traslata per adattarsi al segnale in ingresso.
La differenza principale tra le wavelet e la trasformata di Fourier è che le wavelet sono localizzate sia nel tempo che nella frequenza, mentre la trasformata di Fourier è localizzata solo in frequenza. Inoltre il più grande vantaggio della DWT è che non solo mi dà la frequenza, ma dice anche dov'è localizzata.

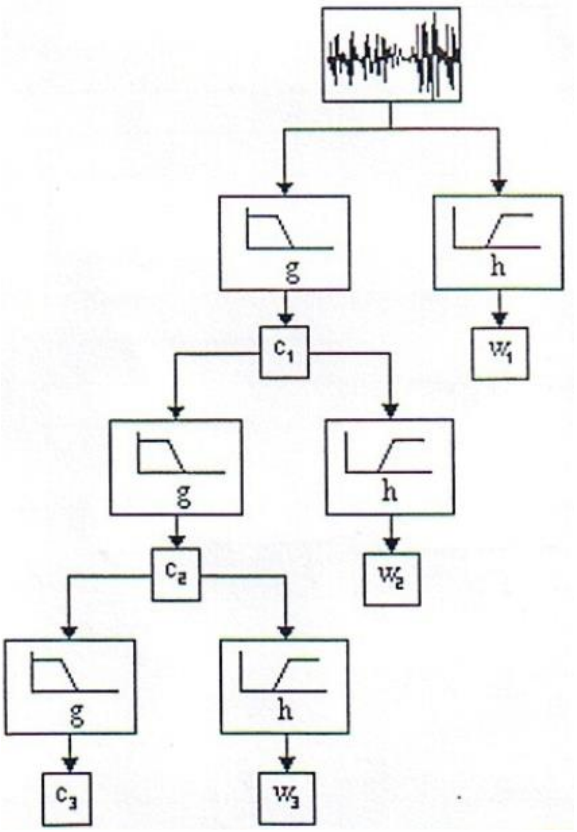
La **wavelet Haar** è stata la prima wavelet ed è anche la più semplice. Essa è veloce da realizzare perché si basa solo su semisomme e semidifferenze, però produce anche risultati scadenti.
La wavelet madre di Haar è la funzione:

$$\psi(t) = \begin{cases} 1 & 0 \le t < 1/2, \\ -1 & 1/2 \le t < 1, \\ 0 & \text{altrimenti.} \end{cases}$$

e la sua funzione padre o portante è

$$\phi(t) = \begin{cases} 1 & 0 \le t < 1, \\ 0 & \text{altrimenti.} \end{cases}$$

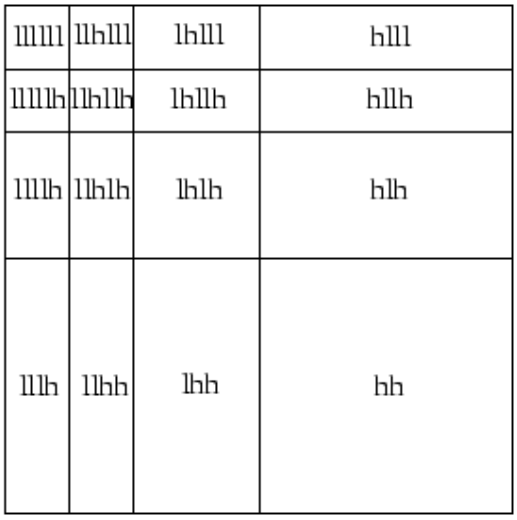
Il segnale d'ingresso è dato dalla somma della portante e delle trasformate di Haar. Dobbiamo prima però trovare i coefficienti da moltiplicare a queste funzioni; questi coefficienti si calcolano attraverso semisomme e semidifferenze (esempio nelle slide).



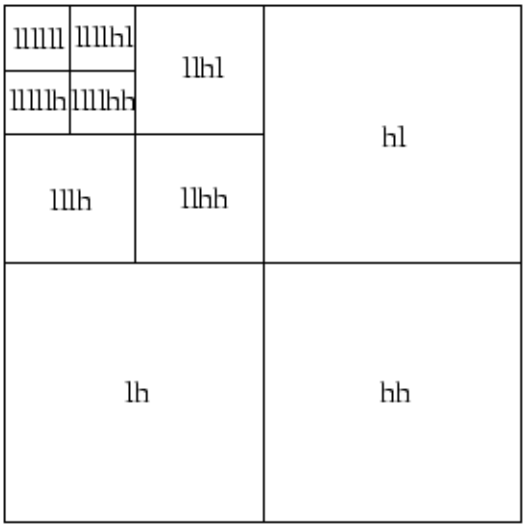
Ponendo le semisomme come filtri passa-basso e le semidifferenze come filtri passa-alto otteniamo il **Filter Bank**. Il segnale d'ingresso viene scomposto nei due filtri ottenendo con il filtro passa-alto h i coefficienti wavelet w_i, e con il filtro passa-basso g i coefficienti C_i, che sono la versione smussata del segnale d'ingresso.
Applicando la DWT a una immagine, moltissimi coefficienti sono prossimi a zero e quindi i quantizzatori a precisione variabile operano nel loro ambiente ideale. Il motivo di tale proprietà è che la DWT fornisce una versione approssimata del segnale dal ramo passa-basso e una versione di dettaglio dal ramo passa-alto; a causa della correlazione fra i campioni, la versione approssimata è molto simile all'originale, mentre la versione di dettaglio è costituita principalmente da coefficienti molto prossimi a zero. Iterando la decomposizione sul ramo passa-basso, al termine si ottengono molte sottobande di dettaglio i cui coefficienti richiedono pochi bit per essere rappresentati, e una sola sottobanda di approssimazione che, visivamente assomiglia molto all'immagine di partenza, che verrà quantizzata con il massimo numero di bit disponibile, ma che è molto più piccola dell'immagine originale.

La DWT per immagini (**2D-DWT**) può essere implementata seguendo fondamentalmente due strategie, che si differenziano in base al criterio con il quale l'immagine viene convertita in una stringa di campioni.

La prima soluzione, decisamente poco usata, consiste nell'applicare la trasformata prima per righe e successivamente per colonne.
La decomposizione risultante, nel caso esemplificato estesa fino al terzo livello, è:



L'altra soluzione consiste invece nell'alternare una decomposizione per righe e una per colonne, iterando solo sulla sottoimmagine passa-basso.
La decomposizione risultante è:



Si osservi la denominazione tipica delle sottobande: le lettere l o h informano attraverso quale filtro è stata eseguita l'analisi; in questo modo le sottoimmagini a due lettere corrispondono al primo livello di decomposizione, quelle a quattro lettere al secondo livello, quelle a sei lettere al terzo livello, e così via. La sottobanda di approssimazione risulta essere quindi identificata da tutte *l*.

Esempio di decomposizione di una fotografia digitalizzata estesa a due livelli:

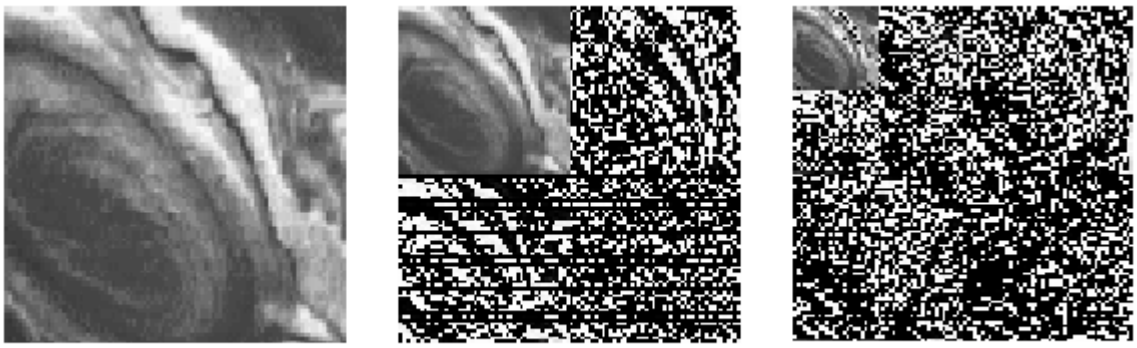


immagine originale — decomposizione a un livello — decomposizione a due livelli

Si osservi come la banda LL in alto a sinistra dell'immagine centrale corrisponda a una *piccola copia* dell'originale, mentre le altre tre sottoimmagini contengano solo le informazioni necessarie alla ricostruzione. Si ricordi che la vera trasformata DWT è costituita esclusivamente dalle sottobande di dettaglio, mentre la *piccola copia* altro non è che la proiezione nel sottospazio approssimazione; essa è tuttavia fondamentale nelle applicazioni di compressione e decompressione di immagini per permettere la successiva ricostruzione.

Algoritmo à trous e JPEG 2000

A differenza del metodo Haar con il quale l'immagine diventava via via più piccola, con l'algoritmo à trous riusciamo a mantenere l'immagine della stessa grandezza, poiché ad allargarsi è il kernel introducendo dei buchi. Ogni immagine viene ottenuta facendo la convoluzione tra l'immagine precedente e un kernel di dimensioni opportune.
Ad ogni iterazione la distanza dal centro (raggio) di un kernel varia secondo le potenze di 2:

Kernel	Raggio	Dimensione
k_0	1	3x3
k_1	2	5x5
k_2	4	9x9
k_3	8	13x13
…	…	…
k_i	2^i	$2^{i+1} + 1$

Applichiamo la convoluzione tra l'immagine originale I_0 e il kernel k_0 (filtro passa-basso), ottenendo una versione smussata I_1 dell'immagine. Successivamente andiamo a sottrarre all'immagine originale I_0, l'immagine smussata I_1, ottenendo i dettagli dell'immagine I_0, cioè è come se avessimo applicato un filtro passa-alto a I_0, ottenendo così W_1.
A questo punto estendiamo il kernel con valori nulli, passando al kernel k_1.
Facciamo la convoluzione di I_1 col nuovo kernel k_1 ottenendo una versione ancora più smussata I_2.
Facendo la differenza tra I_1 e I_2 otteniamo W_2, cioè i dettagli di I_1.
Si prosegue estendendo il kernel e trovando I_3 e W_3, e così via.
Questo modo di procedere non va a cambiare la complessità dell'algoritmo in quanto i coefficienti diversi da 0 sono sempre 9 e quindi nella convoluzione abbiamo sempre 9 prodotti e 8 somme.
Con l'algoritmo à trous i valori dei coefficienti ad ogni iterazione tendono velocemente a zero poiché facciamo delle differenze. Perciò è sufficiente fare soltanto 3 iterazioni:

I_0 = Immagine originale	
$I_1 = I_0 \circledast k_0$	$W_1 = I_0 - I_1$
$I_2 = I_1 \circledast k_1$	$W_2 = I_1 - I_2$
$I_3 = I_2 \circledast k_2$	$W_3 = I_2 - I_3$

Sommando i coefficienti della wavelet otteniamo:
$W_1 + W_2 + W_3 = I_0 - I_1 + I_1 - I_2 + I_2 - I_3 = I_0 - I_3 \rightarrow I_0 = W_1 + W_2 + W_3 + I_3$
Ciò significa che possiamo risalire all'immagine iniziale I_0 sommando i coefficienti wavelet con l'immagine smussata I_3.

JPEG 2000 è uno standard di compressione dell'immagine basato sulla trasformata wavelet discreta (DWT): questa è la principale differenza con lo standard JPEG che utilizza la trasformata coseno (DCT) per la compressione. JPEG 2000 non presenta artefatti a blocchi tipici del JPEG dovuti alla suddivisione a blocchi di 8x8 pixel. Infatti JPEG 2000 analizza l'immagine nella sua totalità, e l'immagine compressa ha un aspetto più o meno sfocato nei contorni degli oggetti.
Dunque a parità di compressione JPEG 2000 ottiene risultati migliori rispetto al JPEG.
Tuttavia ancora è poco utilizzato perché non supportato da molti software e browser.

Il formato Bitmap

Windows bitmap è un formato dati utilizzato per la rappresentazione di immagini raster sui sistemi operativi Microsoft Windows. Il formato di file Windows bitmap permette operazioni di lettura e scrittura molto veloci e senza perdita di qualità, ma richiede generalmente una maggior quantità di memoria rispetto ad altri formati analoghi.
Le immagini bitmap possono avere una profondità di 1, 4, 8, 16, 24 o 32 bit per pixel. Le bitmap con 1, 4 e 8 bit contengono una tavolozza per la conversione dei (rispettivamente 2, 16 e 256) possibili indici numerici nei rispettivi colori. Nelle immagini con profondità più alta il colore non è indicizzato bensì codificato direttamente nelle sue componenti cromatiche RGB.

Le immagini bitmap sono codificate utilizzando alcune strutture che ne descrivono le proprietà:
BITMAPFILEHEADER: Definisce l'intestazione di un file di bitmap indipendente dal dispositivo e contiene i dati che definiscono il tipo, la dimensione e la disposizione del file di bitmap.

BITMAPINFOHEADER: Qui sono indicate le dimensioni in pixel dell'immagine e il numero di colori utilizzati. Le informazioni sono relative al dispositivo sul quale la bitmap è stata creata.
Sempre in questa struttura sono indicate inoltre la risoluzione orizzontale e verticale del dispositivo di output: questi valori, uniti a quelli della larghezza e dell'altezza in pixel, determinano le dimensioni di stampa dell'immagine in grandezza reale. Alcuni campi più importanti sono:
biWidth e **biHeight**: indicano rispettivamente la larghezza e l'altezza in pixel della bitmap
biBitCount: indica il numero di bit richiesti per descrivere ogni pixel nella bitmap:

- Se *biBitCount* è 1, la bitmap è monocromatica, la tabella dei colori deve contenere due ingressi e ogni bit nella bitmap rappresenta un pixel; un bit a zero rappresenta il primo colore nella tabella, un bit a uno rappresenta il secondo colore.
- Se *biBitCount* è 4, la bitmap ha fino a 16 colori numerati da 0 a 15, ogni pixel nella bitmap richiede quattro bit per indicarne il colore; la tabella dei colori contiene 16 ingressi, ogni byte nella bitmap rappresenta due pixel, il primo nel mezzo byte superiore e il secondo nel mezzo byte inferiore.
- Se *biBitCount* è 8, la bitmap ha fino a 256 colori, ogni byte rappresenta un pixel, quindi ogni byte nella bitmap rappresenta un indice tra 0 e 255 nella tabella dei colori.
- Se *biBitCount* è 24, la bitmap ha fino a 224 colori, la tabella dei colori non esiste e ogni pixel è rappresentato da una terna di byte che indicano l'intensità del rosso, del verde e del blu nel pixel.

biSizeImage: indica la dimensione in byte dell'immagine della bitmap.

Una delle caratteristiche essenziali del formato bitmap è la velocità con cui le immagini vengono lette o scritte su disco, molto maggiore se paragonata a quella di altri tipi di file.
Le bitmap occupano più spazio su disco rispetto ad altri formati come GIF o PNG, e sono perciò meno adatte di questi alla trasmissione di immagini via Internet. Una limitazione grave del formato bitmap è quella di non supportare alcun tipo di trasparenza (canale alfa).
Veloci e ingombranti, le bitmap si rivelano adatte alla memorizzazione temporanea delle immagini che vengono modificate spesso, mentre appare poco adeguato ad Internet.

Il formato GIF

Il **GIF** (**Graphics Interchange Format**) è un formato per immagini digitali molto utilizzato nel web e perciò molto diffuso grazie a Internet.
Il numero massimo di colori visualizzabili è 256, ma tra i punti di forza di questo formato vi sono la possibilità di creare immagini animate; molto spesso viene usato per le animazioni e in secondo piano per le immagini fisse. Il formato GIF si diffuse perché utilizzava l'algoritmo di compressione lossless **LZW**, molto più efficiente dell'RLE adottato da altri formati immagine.
Anche la caratteristica opzionale di interlacciamento, che memorizza le linee in un ordine tale da rendere riconoscibile un'immagine solo parzialmente scaricata, contribuì ad incrementare la popolarità del GIF, permettendo agli utilizzatori di riconoscere anzitempo gli scaricamenti errati.
Il formato GIF prevede l'utilizzo di un numero massimo di 256 colori essendo basato sull'uso della tavolozza. Ogni colore all'interno della tavolozza è definito da una terna di valori RGB delle dimensioni di un byte (di valore compreso tra 0 e 255) consentendo quindi di definire, per ogni colore, 256×256×256 sfumature, ovvero circa 16,8 milioni di colori distinti. La tavolozza è formata da 256 colori scelti tra i 16,8 milioni di colori distinti, i quali vengono appunto numerati da 0 a 255; ciò permette di rappresentare ogni singolo pixel con un solo byte che fa riferimento alla posizione del colore nella tavolozza. Un singolo colore della tavolozza può essere, opzionalmente, definito come trasparente e quindi, in fase di visualizzazione, viene sostituito con il colore di sfondo o con l'immagine sottostante. Questa caratteristica differisce dal *canale alfa* in quanto non permette di rappresentare la semitrasparenza; Con questa opzione è possibile creare immagini non rettangolari.

Un file GIF è essere strutturato in vari blocchi. I più importanti sono:

- **Header**: La prima versione del GIF è denominata 87a. Nel 1989, si diffuse una versione migliorata, denominata 89a, che aggiunse il supporto per la trasparenza e le immagini multiple. Il blocco header è formato dai primi 6 byte del file è indicano la versione del GIF; infatti, se interpretati come caratteri ASCII, i primi 6 byte riportano le scritte *GIF87a* o *GIF89a*.
- **Logical Screen Description**: questo blocco definisce la grandezza dello schermo logico in cui l'immagine verrà visualizzata; in pratica indica le dimensioni in pixel dell'area in cui il file gif verrà posizionato. Se si vuole costruire un'animazione GIF è importante quindi considerare le dimensioni di tutti i frame e impostare il Logical Screen in modo che possa contenerli tutti. Questo blocco contiene inoltre una **Global Color Table**, cioè una palette globale di colori comune fra tutte le immagini che compongono il file GIF. Questa può avere 2, 4, 8, 16, 32, 64, 128 o 256 colori; uno di questi colori viene scelto come colore di sfondo del Logical Screen. Le palette sono molto importanti perché ogni colore presente nell'immagine deve provenire dalla palette. Se i colori di un'immagine singola differiscono molto da quelli presenti nella Global Color Table, conviene definire una **Local Color Table**, contenente i colori che si riferiscono ad un'unica immagine, per evitare di alterare l'immagine quando questa viene visualizzata.
- **Graphic Control Block**: questo blocco gestisce la modalità con cui l'immagine deve essere visualizzata in particolare definisce:
  - La presenza o meno di una palette locale
  - Se uno dei colori della palette locale è trasparente
  - Se per procedere con l'animazione è richiesto un input dall'utente
  - La durata della pausa (in centesimi di secondo) fra due immagini consecutive
- **Image Descriptor Block**: questo blocco contiene i dati di una singola immagine e altre informazioni come le sue dimensioni in pixel, la sua posizione nel Logical Screen e l'eventuale palette locale.
- **Trailer**: questo blocco serve per indicare la fine del file; esso deve essere sempre presente e non deve mai essere modificato.