

LEZIONE 1

7/03/07

Ci sono vari testi su cui potiamo fare riferimento -

Dobbiamo intanto fare una distinzione iniziale tra:

- Computer Graphics
- Computer Vision

COMPUTER GRAPHICS & COMPUTER VISION

con Computer Graphics si intende comunemente l'insieme delle tecniche per la generazione di immagini utilizzando un computer. Al giorno d'oggi infatti c'è un concetto che sta alla base di molte elaborazioni come videogiochi, animazione o retocco fotografico (e così via).

Le Computer Vision invece è conosciuta anche come "Visione Artificiale" ed è quella di cui ci occupiamo. È la scienza che tenta di rendere possibile che le macchine possa "vedere" - (significato dall'immagine). Vedere inteso non solo come l'acquisizione di immagini in tante varie prospettive come l'interpretazione del contenuto di quell'area. La Computer Vision è una branca dell'Intelligenza Artificiale AI, e comprende tutte le teorie, tecnologie e metodologie per la costruzione di sistemi per la computer vision.

Esempi di sistemi adibiti alle C.V.:

- controllo dei processi (Robot e veicoli autonomi)
- individuazione di Eventi (videowatchers)
- Modellazione di Oggetti o Ambienti (ispezioni in ambito medico o industriale)
- Interazione tra l'uomo e il computer

Ci sono fondamentalmente 3 tipi di C.V.:

- Basso livello
- Interpretativo (dal + basso al + alto)
- Alto livello

Può alto diviso intenderemo ad esempio che possiamo dare "un pasto" al computer un'immagine e questo mi lo restituiscia decifrata (ma noi in questo studio non ci spingeremo così oltre).

COMPUTER VISION

→ Analisi automatica delle immagini

Esistono 2 approcci a questo problema:

- Appuccio Antropomorfo (non più interessante)
- Appuccio Pragmatico (più rigenerativo)

In questo secondo aspetto l'importante è trovare un metodo che permetta al calcolatore di risolvere il problema, lo scopo resta comunque quello di ottenere dei risultati "d'acento", paragonabili anche contanamente a quelli che potrebbe fare il nostro cervello.

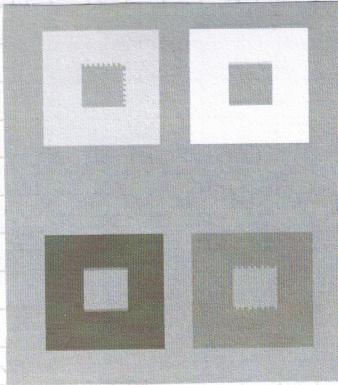
IL SISTEMA VISIVO

L'uomo ha una percezione visiva molto sviluppata - Il computer è

questo scopo si evolvono continuamente e anche al livello di "colore" in se però, abbiamo ormai pc con 16 milioni di colori (24 bit, 3 byte) ma anche 32 milioni (4 byte in modo che ci restano 8 bit).

La percezione del colore di un oggetto non dipende tanto dalla intensità della sorgente luminosa, quanto dal contesto che

c'immagine e il suo sfondo.



→ Contrasto simultaneo

In questo esempio infatti si nota subito come le stesse tonalità di grigio - come pure la luminosità - divisi se lo sfondo cambia.

(i quadrati nel grigio all'interno sono gli stessi ma appaiono diversissimi)

Oppure come altro esempio potremmo dire che un corpo rosso appare più rosso su sfondo verde il bianco è accentuato da un contorno nero; il nero scompare di più se si trova nel bianco;

Tornando brevemente al programma del corso, andiamo a vedere:

- Analisi Preliminari
- Estrazione delle caratteristiche fondamentali
- Segmentazione
- Classificazione
- Analisi strutturale
- Interpretazione

LUMINANZA = E' il flusso luminoso emesso da una superficie di area unitaria (1 m^2) della sorgente entro un angolo solido di 1 sr in direzione perpendicolare alla superficie.

E' quindi il rapporto tra l'intensità luminosa emessa da una sorgente verso una superficie normale alle direzioni 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 espresso in

cd/m^2 (candela in metro quadro)

La luminanza è importante per le sorgenti esterne, in quanto ciò de 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".

Tipiche luminanze sono:

- Sole	10^9
- Faro di Automobile	10^7
- Faro alogeno	$10^4 \div 10^6$
- Faro a LED	$10^3 \div 10^4$
- Luce piana	10
- Minilamp per visione fotografica	1
- Illuminazione stradale	$10^{-6} \div 10^{-3}$
- Cielo notturno senza luna	

Questa nostra percezione cingu non è stata ancora raggiunta dalle macchine digitali.

CCD = Charge Coupled Device.

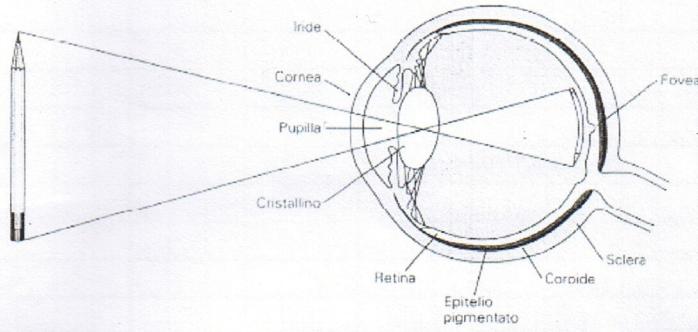
Dispositivo a trasferimento di carica - Rivelatore di fotoni: circuito integrato che comprende milioni di rivelatori.

Un CCD consiste quindi in un circuito integrato formato da una griglia (o griglia) di elementi semiconduttori, in grado di

accumulare una carica (charge) proporzionale all'intensità della radiazione ultramagnetica che li colpisce.
Questi elementi sono accoppiati (coupled) in modo che ogni uno di essi, sollecitato da un impulso elettrico, possa trasferire le proprie cariche ad un altro elemento adiacente.
Invianto al dispositivo (device) una sequenza temporizzata di impulsi, si ottiene in uscita un segnale digitale grazie al quale è possibile ricostruire la matrice dei pixel che compongono l'immagine proiettata sulle superfici del CCD stesso.

→ In pratica è insieme di lenti, segnali e fotoni strutturati in maniera simile al nostro occhio. Ma, allo stesso tempo, fatto per un determinato range di luminosità che non arriverà mai ad un intervallo di eccitazione come il nostro occhio (che vede già da 10^{-6} cd/m^2)

ANATOMIA DEL SISTEMA VISIVO:



Il cristallino di occhio proietta le immagini riverte sulla retina, analogamente a quanto fatto dall'obiettivo di una fotocamera.

(Famoso esperimento degli occhiali a specchio per vedere al contrario, il cervello dopo circa 1 stimolazione non abilita)

Nella Fovea troviamo 1 allissima densità di fotorecettori.

Inoltre potremmo parlare di vista e vista periferica, di campo visuale di messa a fuoco, della cosiddetta visione "con le code dell'occhio". Anche se a volte ci sembra di fissare un punto, l'occhio compie ugualmente dei movimenti concentrici impercettibili chiamati → Movimenti saccadi.

Inoltre tra le innumerevoli capacità dell'occhio umano troviamo quelle di "percepire oggetti per continuità" (quando guardiamo una persona di profilo il nostro cervello ne ricostruisce il viso per intero).

Il problema ovviamente è spiegare al pc come trovare queste immagini. Un altro esempio potrebbe essere l'individuazione di assi e centri di simmetria radiali, di zone di simmetria o di asimmetria nelle immagini (esempi nelle slide 9-10).

Ma come scrivere un programma che individua queste zone? Come dire ad es. ad una macchina di codificare i contorni? Potremmo (ad es.) ricordare il problema (codificando con l'elenco dell'alfabeto) al confronto di parole.

ANALOGICO / DIGITALE

Vediamo le fondamentali differenze.

Analogico = Ciò che si riferisce ad un sistema non numerabile, non analizzabile entro un insieme discreto di elementi. Ciò che è analogico viene modellizzato con le matematiche del continuo, che tratta una infinità (numerabile o non) di elementi.

→ Infiniti punti con infiniti livelli di luminosità presiede + valori.

Digitale = Deriva da digit che in inglese vuol dire "apre", e da digital che in latino vuol dire dito.
 E' quindi ciò che è rappresentabile con i numeri, e può essere contato (ad es. come digitali).
 E' il contrapposto di analogico, digitale vuol dire quando c'è riferito alla matematica me nel discreto, che lavora con un insieme finito di elementi.
 Il passaggio da analogico a digitale è chiamato → digitalizzazione.

Ha quindi un supporto fisico finito (dobbiamo effettuare 2 quantizzazioni sullo spazio: mettere i punti dentro uno spazio finito).

Pixel → Elemento puntiforme che compone la rappresentazione di una immagine raster nella memoria di un computer.
 Punto elementare → ole picture element.

1 Mega Pixel → Un milione di punti elementari.

CCD → Tanti elementi quanti sono i pixel, calcola le media dei fotoni per la luminosità.

Con "D" d'ore mi permettendiamo digitale.

RETINA DIGITALE

Una immagine digitale D è una funzione definita sullo spazio discreto bidimensionale, detto retina, i cui valori discrete sono detti livelli di grigio g (luminosità).

$$D = \{(i, j, g) : i \in \{0, \dots, W-1\}, j \in \{0, \dots, h-1\}, g \in \{0, \dots, G-1\}\}$$

dove i e j sono le coordinate del punto nella struttura digitale che vanno da zero a w (width) per i, e da 0 a h (height) per j (ma uno pochi punti da zero). W e h stanno per lunghezza e altezza, rispettivamente.

Assumendo di quantizzare (per convenzione) i G livelli di grigio, possiamo associare:

$$\begin{aligned} 0 &= \text{nero} \\ G-1 &= \text{bianco} \\ G/2 &= \text{grigio} \end{aligned}$$

(Slide 11-12)

Per semplificazione si impone che le dimensioni della retina digitale siano quadrate e che w e h siano potenze di 2.

Ma una risoluzione a 256×256 è ormai superata.
 Si usano ora le 512×512 o le 1024×1024 . Nel caso che dimensioni siano rettangolari si mantiene il rapporto 4/3.

La quantità di informazione 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 percepire tutti, ma ne percepisce all'incirca una trentina.

⇒ ≈ 30 : In ogni caso percepiamo migliò le sfumature di grigio che quelle relative ai colori (se le vediamo ad esempio al massimo).

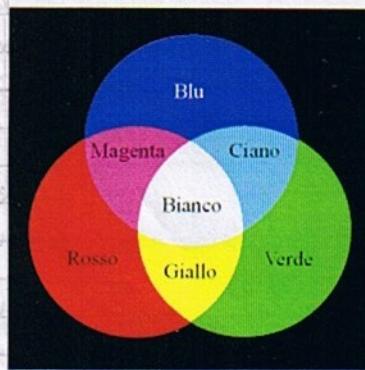
E tre questi a loro volta percepiamo meglio le sfumature di verde - (abbiamo per queste molte vettori nulla forse). Poi nell'ordine percepiamo il rosso e infine il blu.

SINTESI ADDITIVA E SOTTRATTIVA

SINTESI ADDITIVA →

(i colori primari in questo caso sono il rosso, il verde e il blu)

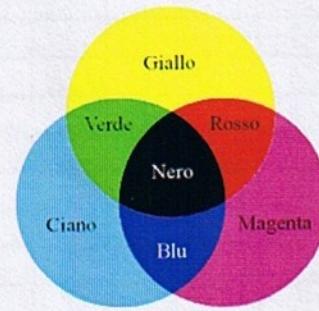
Assenza di luce = nero



SINTESI SOTTRATTIVA →

(i colori primari in questo caso sono il giallo, il ciano e il magenta)

Assenza di pigmento = bianco



Eperimenti a cura di "Boringò Pinna"

Dall'analogico al Digitale → Campionamento e Quantizzazione

Campionamento → nello spazio

* (Pag. 9)

Quantizzazione → nella luce

x_{min} etc delimitano le dimensioni dell'immagine analogica

$$i = \min \{ \lfloor w \times (n - n_{min}) / (n_{max} - n_{min}) \rfloor, w-1 \} \quad (\text{Camp.})$$

$$j = \min \{ \lfloor h \times (y - y_{min}) / (y_{max} - y_{min}) \rfloor, h-1 \} \quad (\text{Camp.})$$

luminosità dell'immag. analog.

$$g = \min \{ \lfloor G \times (l - l_{min}) / (l_{max} - l_{min}) \rfloor, G-1 \} \quad (\text{Quant.})$$

$L \rightarrow$ si approssima in quanto in digitale i e j sono interi.
Quantizzazione → i valori sono gli stessi come formule ma i processi prendono due nomi diversi in spazio e luce

Con " I " definiamo invece l'immagine analogica, ma non avendo approssimazioni.

Assumendo che l'immagine analogica I originale ammetta decomposizione in serie di Fourier con frequenze massime ω_x e ω_y

$$I(x, y) = \frac{1}{4\pi^2} \sum_{u=0}^{w_x} \sum_{v=0}^{w_y} T(u, v) e^{i(u\omega_x x - v\omega_y y)}$$

di coordinate ai numeri immaginari.
punto

(due sommatorie perché due variabili)

⇒

5

allora può essere completamente determinata da ampiezza e campionate ad intervalli spaziali dati da:

$$\Delta x = \frac{1}{2} v_x \quad \Delta y = \frac{1}{2} v_y$$

v_x e v_y

Sono uguali i Δ se voglio il pixel quadrato

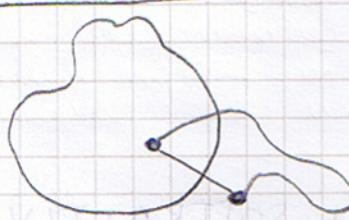
per cui un criterio di campionatura uniforme è: $\Delta = \min\{\Delta_x, \Delta_y\}$

Δ vuol dire variazione, lo possiamo trovare o fissare a priori; da qui andiamo al concetto di trasformata di Fourier (andremo)

Ho un'immagine D e la voglio sottocampionare
(2^{a} forma di Shannon o del Campionamento)

Torna del campionamento

(Per unire un punto vicino ad una curva chiusa con un punto vicino devo toccare la curva stessa in 10 più punti)



TEOREMA DI JORDAN

Ovviamente se voglio campionare l'immagine devo trovare un delta Δ che mi permette di mantenere i dettagli (se un'immagine è troppo piccola infatti si verifica una perdita di informazioni)

Poi con Fourier possiamo tirare fuori le frequenze (nel'immagine non potremo usare solo i coseni)

Alla fine trovo Δx e Δy (vedi valori sopra)

Ovviamente in una immagine quadrata i valori di Δx e Δy coincidono.

da frequenza con la quale vogliamo campionare il segnale non deve essere più minore al doppio della f. max che c'è nel segnale.

(più piccole di quelle frequenze non posso campionare per non perdere informazioni)

Frequenze piccole \rightarrow alte

Se invece usassi un delta più piccolo avrei "tutte le informazioni" ma sarebbe come "sporcare ad una mosca con un cannone".

INTORNO FORTE E DEBOLE

Possiamo rappresentare la retina digitale in due modi attraverso i concetti di intorno forte e intorno debole.

Possiamo vedere l'intorno attraverso pixel quadrati o tonoli (slide 15), noi useremo quelli quadrati.

Intorno forte \rightarrow 4 pixel adiacenti sul lato comune
(le linee rappresentano la connessione tra i pixel vicini dell'intorno)
(contiene 5 pixel totali) \rightarrow nel meglio 1

Intorno debole \rightarrow slide 16.
8 pixel adiacenti nella cosiddetta otto-connessione
virtuale (fanno parte dell'intorno anche i pixel
che hanno solo l'angolo in comune)
d'intorno in totale contiene 9 pixel.

Teorema di Jordan → (già scritto prima)
 (o PARADOSSO)

Una curva chiusa delimita 2 aree, se puoi 1 punto esterno e 1 interno, qualunque linea che li unisce taglia la curva.

A questo teorema vale nel disegno il CONTORNO.
 Si utilizza la 4-connectività → vale (Slide 17) anche per il disegno, & con le 8-conn. niente no- (non sempre vale per il contorno)

(detto anche Paradosso di Jordan)

INTORNO DIGITALE → Fondamentale uttanto il concetto di **RAGGIO**

Intorno di raggio 0 → è il pixel stesso

Intorno di raggio 1 → a seconda della connettività sono 4 o 8 deboli

Intorno di raggio 2 → Pixel di raggio 1 da quelli già di raggio 1

CITY BLOCK

distanza definita con intorni forti (4 connettività)

Slide 18

$d_4 = d_x + d_y$ → basta calcolare le distanze di tre i vari pixel dell'intorno

CHESSBOARD

distanza definita con intorni debolli (8 connettività), slide 19

$$d_8 = \max\{d_x, d_y\}$$

OTTAGONALE

la migliore. Prendiamo ad un passo le 4-conn. e all'altro le 8-conn., alternandole. Slide 20.

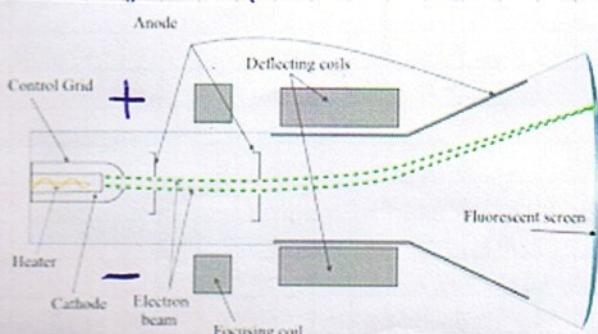
In questo modo avremo un intorno più simile ad un cerchio (ovvero come noi immaginiamo intuitivamente un intorno).

$$d_0 = \max\{d_8, \lfloor 2(d_4+1)/3 \rfloor\}$$

CRT - TUBO A RAGGI CATODICI

Il termine indica la tecnologia comunemente usata per la visualizzazione in monitor e televisori, che consiste nel ~~convogliare~~ ad hoc dei raggi catodici su di una superficie sensibile, che ricrea l'immagine visibile. La misura del monitor CRT è ~~attualmente~~ in pollici nelle diagonali.

Attualmente questa tecnologia sta "scendendo" a favore delle tecnologie a plasma o a cristalli liquidi.



Dentro la tv in pratica c'è un tubo a raggi catodici. Il catodo è una luce calice negativamente che viene scaldata in modo da accumulare sufficiente energia per poter emettere elettroni.

Gli elettroni vengono accelerati verso una testa canice puntiformemente, chiamata anodo - da forza attrice fra anodo e catodo produce il fascio di elettroni. Questo colpisce l'interno dello schermo televisivo, che è ricoperto con granuli di fosforo - (Il fosforo si colpisce da elettroni avendo luminescenza) → quindi lo schermo si illumina - (i puntini sullo schermo)

Il fascio di elettroni, per mezzo del fosforo, produce le immagini che vediamo -

Ma questo fascio, è focalizzato in tempi diversi in punti differenti dello schermo, cioè attraversa lo schermo in linee. Dopo che tutte le linee relative ad un'immagine sono state analizzate, il raggio passa alle linee dell'immagine seguente. Ma la persistenza dell'immagine nella nostra memoria fa sì che le immagini si sovrappongono e creino l'impressione del moto movimento -

→ da nostra percezione di "immagine fluida" è dovuta alla persistenza dell'immagine in memoria.

Se guardiamo una foto alla tv infatti, ci accorgiamo di vedere solo le "fasce" illuminate in quel momento. Queste fasce si muovono in maniera "interallacciata" in modo che debba passare la metà del tempo prima che quella parte venga illuminata di nuovo.

Ma come far il fascio di elettroni a muoversi nelle diverse parti dello schermo?

- Il fascio viene inviato alle diverse parti dello schermo, dei magneti piegano il fascio in avanti verso le varie parti dello schermo.

Ma come ci vediamo i colori? (FILTO DI BAYER)

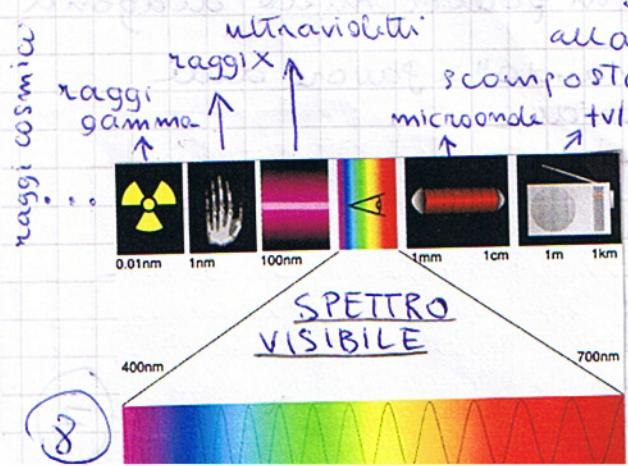
Davanti lo schermo possiamo immaginare tanti puntini colorati di rosso, blu e verde. In realtà non abbiamo un fascio solo matre fasce di elettroni e tre diversi tipi di granuli di fosforo. Ogni fascio di elettroni viene diretto da una griglia in modo da arrivare ad un granulo dello stesso colore.

In questo modo (slide 25 e 26) abbiamo i tre diversi colori da mescolare -

(A volte troviamo anche filtri verticali → Sony)

COLORI RGB

(Tricromia)



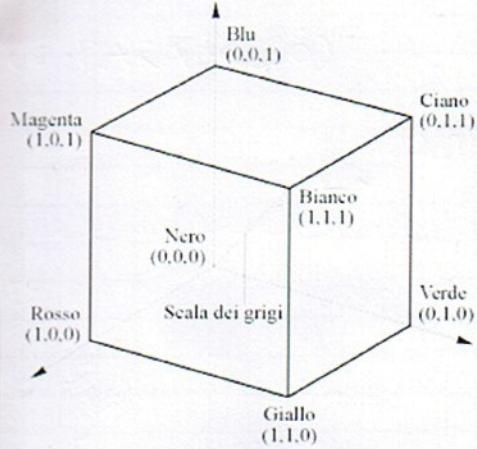
Se poniamo un prisma e lo esponiamo alla luce vedremo come questa viene scomposta dal prisma nei 7 colori.

(Slide 23)

Rid Green Blue sta appunto per RGB
Abbiamo 256 sfumature diverse per ogni colore

R G B
(0,0,0) → nero
(255,255,255) → bianco

tra la luce visibile e le microndine → marrone



E' come vedere il colore RGB come un cubo nello spazio, visto dalla parte del nero o del bianco -
 \rightarrow '1' sta anche per '1255' in RGB

SPAZIO DEI COLORI RGB (CCD)

Metiamo 1 filtro di bayer davanti al CCD.

Ho quindi dei quadratini verdi (R) rossi (R) e blu (B) come mi figura (Slide 32)

Come si può osservare, ogni 4 quadratini presi con tutti e 3 i colori (Visto che 1 colore due compare 2 volte scegliamo di reduplicare il verde, le cui tonalità sono meglio percepite dal nostro occhio)

Il valore di ogni filtro è dato dalla media pesata dei pixel adiacenti

Se un pixel ad esempio non ha filtro, per vedere il suo colore prendiamo quello dei 4 adiacenti così possiamo vedere ad es. che intensità di verde dare, e ne facciamo la media per fondo. (In questo modo ho l'impressione di avere 1 filtro verde su tutti i pixel, anche dove non c'è)

CCD \rightarrow array di sensori che si eccitano se vengono colpiti da光子

Molti davanti 1 filtro di Bayer (più verde) dove non ho pixel verdi vado per "interpolazione" - lo stesso faccio per blu e nero -

\rightarrow Facendo le combinazioni ho tutti e 16 milioni di colori - lo spazio dei colori in pratica è una rappresentazione matematica di un insieme di elementi chromatici. Questo spazio possiamo rappresentarlo tramite un sistema tridimensionale di coordinate cartesiane.

* Quando passiamo a digitale con i, j ottino il campionamento e riguarda lo spazio mentre con g ho la quantizzazione e riguarda la luce. Quindi queste due operazioni sono fondamentali per il passaggio da analogico/digitale. Se consideriamo Δ la minima variazione appetibile della luminosità il not. di livelli di grigio è dato da: $G = \Gamma(l_{\max} - l_{\min}) / \Delta T$

TEOREMA DEL CAMPIONAMENTO/SHANNON/NYQUIST

Questo teorema afferma che la frequenza con cui vogliamo campionare il segnale non deve essere inferiore al doppio delle frequenze massime che c'è nel segnale. Nello specifico, supponiamo di avere immagini analogiche e di volerle campionare rendendole più piccole (ma non troppo per non perdere dettagli). Supponiamo quindi che questa immagine analogica possa decomporre in serie di Fourier trovando 2 frequenze massime V_x, V_y , cioè ($I(x,y)$ pag. 5). Avendo trovato le 2 frequenze massime possiamo trovare 2 valori Δx che è la base del pixel e Δy che è l'altezza tale che mi dia le dimensioni corrette del pixel per l'immagine analogica I data in input. Δx e Δy pag. 6.

Diminuendo questi valori sfoco le immagini - per cui il criterio del campionamento uniforme è $\Delta = \min\{\Delta x, \Delta y\}$. Nasce dall'esigenza di archiviare in forme digitali e con la fedeltà necessaria tutte le info relative ad eventuali dati analogici.

LEZIONE 2

14/03/07

I più importanti modelli di colori sono l' RGB (utilizzato nell'ambito delle computer graphics), l' YUV (usato nei sistemi video) etc...

YUV

Questo spazio rappresenta una trasformazione lineare dello spazio RGB -

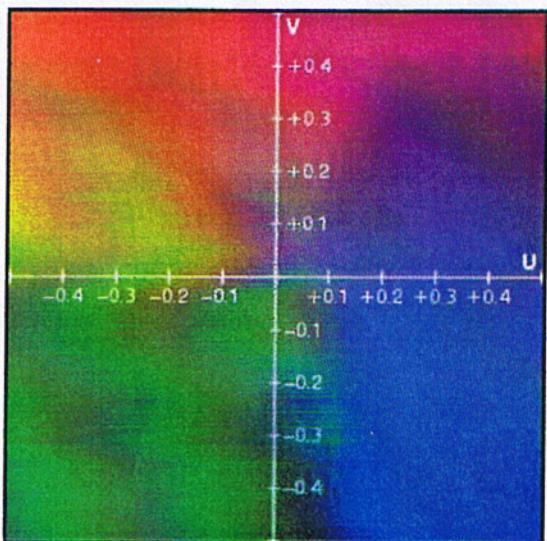
la Y rappresenta l'informazione sulla luminosità dei pixel mentre le componenti U e V contengono informazioni sul colore -

Ognuna delle componenti cromatiche dei due spazi (RGB e YUV) può assumere valori compresi nell'intervallo [0,255], ciò permette di poter salvare ogni componente cromatica in 8 bit -

YUV può contenere solo una componente che possiede le informazioni sulla luminosità dei pixel dell'immagine e non porta alcuna interdipendenza tra le varie componenti: in pratica modificando la luminosità il colore percepito dal pixel risulterà ~~essere~~ diverso e differente - Per questo motivo anche YUV risulta essere adatto per essere utilizzato in sistemi in cui l'università alle variazioni di luminosità rappresenta un aspetto ponderante del sistema -

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} .299 & .587 & .114 \\ -.147 & -.289 & .436 \\ .615 & -.515 & -.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

In pratica si passa da RGB a YUV tramite prodotto matriciale



HSV

= Hue Saturation Value (colore saturazione luminosità)
In questo spazio

S = la saturazione, rappresenta quanto è forte il colore

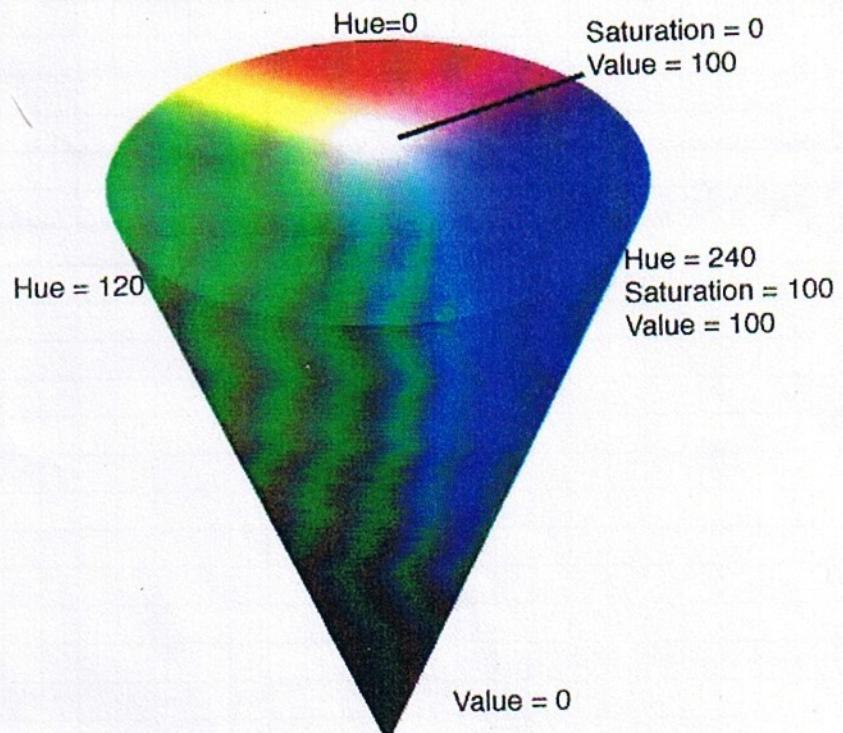
H = è il colore

V = E' la luminosità

La sua rappresentazione geometrica è in un sistema di coordinate cilindriche come il cono (o prisma a base ottagonale) con il vertice rivolto verso il basso -

Cioè è la seguente immagine:

Spazio dei colori HSV



Osserviamo che:

- da coordinata V che corrisponde alle luminosità assume valori nell'intervallo ~~0-100~~ de 0 (scuro) a 100 (brillante) cioè verso uscente = bianco verso entrante = nero
- da coordinata H che è il colore e corrisponde all'angolo cioè va da 0 a 360° cioè con
 - 0° = rosso
 - 120° = verde
 - 240° = blu
- da coordinata S , che è la saturazione, invece è la coordinata che va dal centro fino al bordo

Questo sistema è importante per estendere il canale V della luminosità rendendo questo a differenza degli altri due componenti.

Intuiamo quindi che per poter passare da un'immagine a colori ad una con scale di grigio occorre passare dal sistema

RGB → al sistema

HSV

→ eliminare H e S → mi rimane solo la luminosità V

C'è un altro modo per ottenere la luminosità?

Potrei fare

$$\frac{R+G+B}{3}$$

$$\text{oppure } \frac{1}{3}G + \frac{1}{3}B + \frac{1}{3}R$$

In questo modo non sarebbe del tutto corretto in quanto diamo ai 3 colori

lo stesso peso e sappiamo che non è così perché li percepiamo in modo diverso

In realtà il modo opportuno per ottenere la luminosità è trovando le Y nel sistema YUV facendo:

$$Y = .299 R + .587 G + 0.114 B$$

Usciti Vede che ho
+ verde

Osserviamo che sommando otteniamo 1 così come sommando $.33 + .33 + .33$ è fatto in modo che ottieniamo il valore massimo quando RGB sono massimi.

Osserviamo quindi come varia il seguente fiore modificando:

L'ANGOLO = Ad ogni 60° cambia colore



H=0°



H=60°



H=120°



H=180°



H=240°



H=300°

LA SATURAZIONE:



S=0 → abbiamo solo le luminosità senza H e G e B



S=20



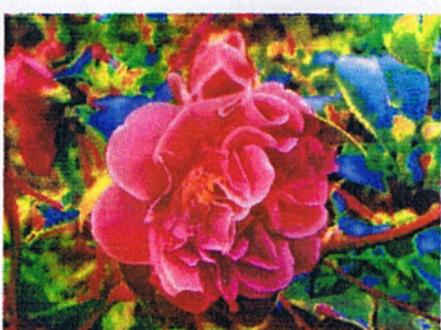
S=40



S=60



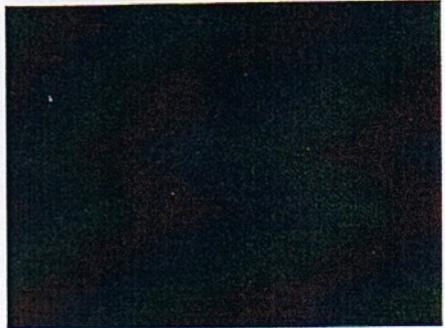
S=80



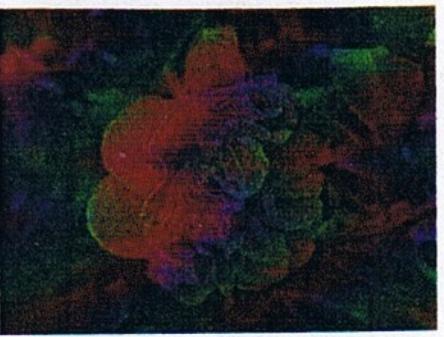
S=100

qui "spaniamo" tutti i colori

LA LUMINOSITA':



V=0 minime quando sono al vertice



V=20



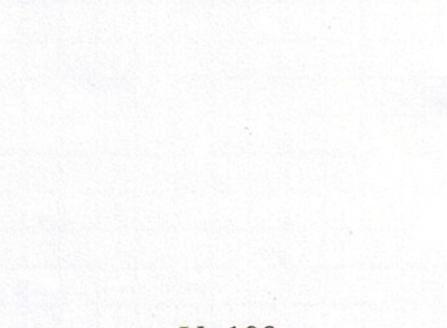
V=40



V=60



V=80



V=100

Questo si puo' vedere andando in photo shop, caricando l'immagine e selezionando

IMMAGINE → REGOLAZIONI → tonalita / saturazione

Eli ho le optioni:

- Tonalita
(che corrisponde al colore H)
- Saturazione
(S)
- Luminosita
(V)

ANALISI DI IMMAGINI

dei vari passi 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 (puntuali) o sui loro intorni (locali)
- il risultato è una nuova immagine $B: I \rightarrow I'$
esempio: riduzione del rumore e aumento del contrasto

- MEDIO LIVELLO

- le strutture dati mantengono le caratteristiche dell'immagine
- gli operatori, in genere globali, estraggono caratteristi che
- il risultato è un'insieme di proprietà dell'immagine
 $N: I' \rightarrow F$

esempio: extrazioni, 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 $A: F \rightarrow R$

esempio: interpretazione simbolica

Osserviamo alcuni OPERATORI ARITMETICO/LOGICO

$$0 = \{(i, j, g) : g = 0\} \quad \text{Immagine nera}$$

$$1 = \{(i, j, g) : g = 1\} \quad \text{Immagine a colori in binario e non}$$

$$255 = \{(i, j, g) : g = 255\}$$

$$K \times I = \{(i, j, g) : g = K \times g \bmod G\}$$

$$K \times I = \{(i, j, g) : g = \min\{G-1, [K \times g]\}\} \quad K \geq 0$$

$$K + I = \{(i, j, g) : g = \min\{G-1, [K+g]\}\} \quad K \geq 0$$

$$\min(I_1, I_2) = \{(i, j, g) : g = \min\{g_1, g_2\}\}$$

$$\max(I_1, I_2) = \{(i, j, g) : g = \max\{g_1, g_2\}\}$$

Prodotto di un'immagine per uno scalare
occorre poi mantenere sempre in $G-1$ perché
nel primo caso uso il
modulo nel secondo uso un
modo simile alla scalatura per tagli
cioè i valori tra 0 e 255 riman
gono gli stessi i valori
maggiori di 255 li
portiamo a 255.

In modo analogo faccia
mo per le somme

se minimo o il massimo
tra 2 immagini si fa
prendendo quella con il
colore maggiore o minore

$$\begin{aligned}
 I_1 + I_2 &= \{(i, j, g) : g = \min\{G-1, g_1 + g_2\}\} \text{ (saturation a 1)} \\
 I_1 + I_2 &= \{(i, j, g) : g = (g_1 + g_2) \bmod G\} \\
 I_1 + I_2 &= \{(i, j, g) : g = \lceil (g_1 + g_2) / 2 \rceil\} \\
 I_1 - I_2 &= \{(i, j, g) : g = \max\{0, g_1 - g_2\}\} \text{ (saturation 0)} \\
 I_1 - I_2 &= \{(i, j, g) : g = |g_1 - g_2|\} \\
 I_1 \times I_2 &= \{(i, j, g) : g = \lfloor g_1 \times g_2 / (G-1) \rfloor\} \\
 \text{sqrt}(I) &= \{(i, j, g) : g = \lfloor g^2 / (G-1) \rfloor\} \quad (\text{quadrato}) \\
 \text{sqrt}(I) &= \{(i, j, g) : g = \lfloor \sqrt{(G-1)g} \rfloor\} \quad (\text{radice quadrata}) \\
 \log(I) &= \{(i, j, g) : g = \lfloor \lg(\varepsilon + g) \times (G-1) / \lg(G-1) \rfloor\} \quad \text{Anche l'operazione log va protetta e lo faccio con la } \varepsilon
 \end{aligned}$$

Somma e sottrazione tra immagini

(8.1)

mentre

• Possiamo anche effettuare le operazioni booleane tra immagini

$$\begin{aligned}
 \text{and}(I_1, I_2) &= \{(i, j, g) : g = \text{and}(g_1, g_2)\} \quad \text{faccio and, or, pixel per pixel di ogni immagine -} \\
 \text{or}(I_1, I_2) &= \{(i, j, g) : g = \text{or}(g_1, g_2)\}
 \end{aligned}$$

$$\text{not}(I) = \{(i, j, g) : g = \text{not}(g)\}$$

$$\text{shift}(I, K) = \{(i, j, g) : g = \text{shift}(g, K)\}$$

Eseguire uno shift di K valori, se lo svolgo a sinistra, moltiplico per due redoppio

non

Nel caso in cui l'immagine è a colonne ogni pixel è rappresentato da 1 byte quindi avremo 8 bit (8 parti da 1) dal meno significativo al più significativo cioè

$$\begin{array}{ccccccc}
 \text{MSB} & & & & \text{LSB} & & \\
 b_8 & - & - & - & - & - & b_1
 \end{array}$$

In questo caso vengono or, and e not, bit by bit cioè sono operazioni BITWISE -

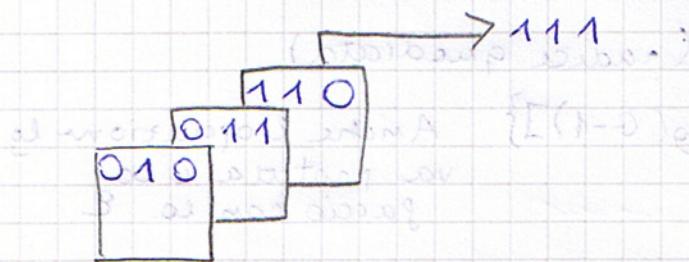
\Rightarrow

d'organizzazione in memoria dei dati che codificano l'immagine può variare a seconda dei criteri. Ci sono varie organizzazioni:

Ad esempio a BITPLANE

In questa organizzazione l'immagine è scomposta in tanti "piani" quanti sono i bit di profondità del colore.

I bit che occupano la stessa posizione in piani diversi formano il codice numerico del colore.



In pratica si vanno prendendo i singoli bit di ogni pixel dell'immagine.

Nel nostro caso poiché un pixel è 1 byte avremo 8 bit plane uno per ogni bit partendo dal meno significativo fino al più significativo.

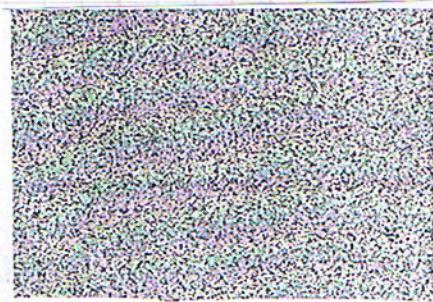
Osserviamo che il bit piano del bit meno significativo non dà molte informazioni riguardo l'immagine infatti avremo 50% di 0 e 50% di 1.

Procedendo con il bitplane fino ad arrivare a quello più significativo l'immagine diventerà sempre più chiara.

(LSB)

bitplane 1

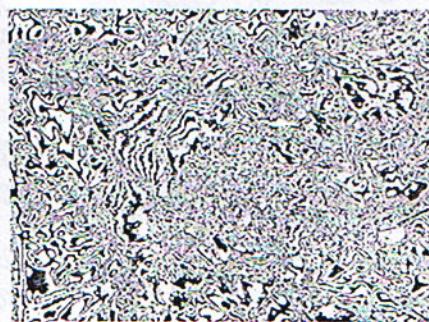
Si prendono i bit meno significativi di tutta l'immagine.



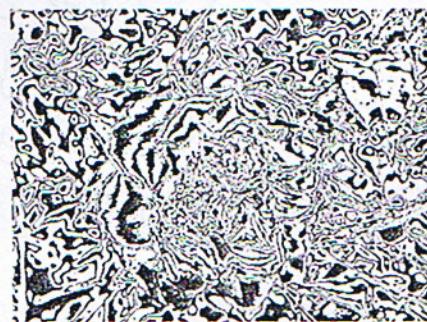
bitplane 2



bitplane #3



bitplane #4



bitplane #5



bitplane #6



bitplane #7



bitplane #8 (MSB)

ISTRUZIONI MATLAB PER LAVORARE CON LE IMMAGINI

> `imread` = Questa istruzione legge un'immagine a colori o in scale di grigio e i che puo' contenere sono:
tif, gif, JPG... e molti altri tenni i formati compressi.
Un esempio è:

> `mig = imread('fiore.tif')` cioè assegno a mig l'immagine fiore.tif
è indicata con il tasto (tasto per il percorso)

> `imshow` = serve per visualizzare un'immagine però osserviamo che con

`imshow(I)` = visualizzo semplicemente l'immagine con 256 grigi.

`imshow(I,n)` = usando n livelli di grigio

`imshow(I,[low,high])` = visualizza l'immagine con una scala di grigi indicata dal row. Il valore low appare come nero. Il valore high appare come bianco.

I valori che stanno in mezzo sono tonalità di grigio.

Quindi se volessimo visualizzare l'immagine di prima a colori avremmo:

`imshow(mig)`

Faccendo `whos` ci accorgiamo che l'immagine a colori è memorizzata come un array tridimensionale di tipo uint8.

> `RGB2gray` = Converte un'immagine a colori o RGB in 1 a scale di grigio. Quindi in questo caso ottengo solo la luminanza.

Ora se oppongo `con whos` l'immagine in bianco e nera notiamo che non ho più un vettore tridimensionale ma bidimensionale.

(de notare che Matlab produce una serie di warnings ma trascurabili poiché legati al fatto che non riusciamo a rappresentare tutte le varie sfumature del colore)

Utilizzando sia queste istruzioni che gli operatori osserviamo come otteniamo i due primi bitplane dei bit meno significativi

Bitplane #1

```
fiore=rgb2gray(imread('000.tif'));
i1=bitand(fiore,1); // 1 che corrisponde a 2° bit meno significativo
figure; imshow(i1,[0,1]);
```

Osserviamo che se avessi messo [0,255] avrei visto l'immagine quasi nera, mentre [0,2] viene sul grigio.



bitplane#2

```
i2 = bitshift(bitand(power, 2), -1);  
figure; imshow(i2, [0, 1]);
```

2 perché corrisponde a 2^1 cioè il 2° bit meno significativo
In questo modo pro' visualizzando l'immagine tra [0 e 2] la
vedremo grigia invece noi vogliamo vederla in bianco e nero quindi
le ripeto tra 0 e 1.

• $\text{bitshift}(\text{bitand}(x, 2), -1)$ visualizza l'immagine in bianco e nero
• $\text{bitshift}(x, 1)$ visualizza l'immagine in bianco e nero

l'immagine laterale corrispondente all'espressione $\text{bitand}(x, 2)$ è visualizzata
in bianco e nero.

• $\text{bitshift}(x, 2)$ visualizza l'immagine in bianco e nero

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \text{ xor } (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

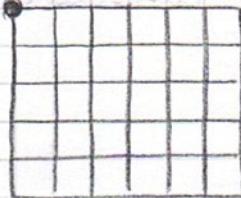
• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

• $\text{bitshift}(x, 1) \oplus \text{bitshift}(x, 2) = (x, 1) \oplus (x, 2)$

FILTRI DI CONVOLUZIONE

Supponiamo di avere un'immagine, che sappiamo che possiamo immaginare come una matrice

Origini



Supponiamo di avere l'altra matrice più piccola ad esempio 3×3 che chiamiamo Kernel

K ₁	K ₂	K ₃
K ₄	K ₅	K ₆
K ₇	K ₈	K ₉

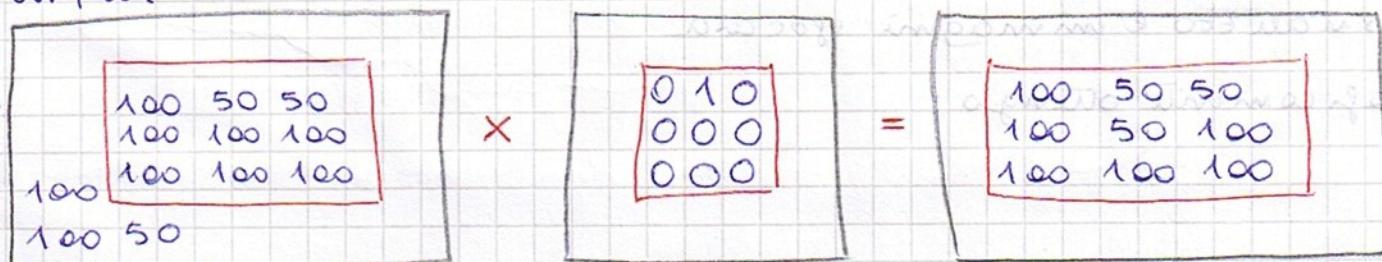
In pratica le convoluzioni è il
processamento di una matrice
attraverso il Kernel - la matrice da processare è proprio la
nostre immagine iniziale -
A seconda del Kernel che usiamo si ottengono effetti
diversi -

In pratica con il Kernel scorri tutta la matrice e riesco ad
individuare 9 pixel cioè g₁g₂... g₉ che avranno 9 tonalità
di grigio poiché l'immagine è grigia -
A questo punto faccio il prodotto scalare:

$$K_1g_1 + K_2g_2 + K_3g_3 + \dots + K_9g_9$$

Il valore che ottengo lo assegno non all'immagine iniziale
nella posizione corrispondente al pixel centrale del Kernel cioè -
K₅ mi viene creata l'altra matrice delle stesse dimensioni
dell'immagine iniziale ed è lì che andrà messa nelle stesse
coordinate -

Esempio:



100	50	50		
100	100	100		
100	100	100		
100	50			

X

0	10		
0	0		
0	0		

=

100	50	50		
100	50	100		
100	100	100		

Osserviamo che vogliamo restare nel range di luminosità tra 0 e 255 pertanto i valori di g sono tutti tra 0 e 255 e al massimo potrebbero essere tutti con 255 in questo caso dovrei dare a K dei valori opportuni affinché non esca da questi parametri.

In caso potrei dare a K₁ = $\frac{1}{9}$, K₂ = $\frac{1}{9}$... K₉ = $\frac{1}{9}$ cioè faccio la media dei g

In questo caso quindi ho ancora un valore tra 0 e 255, mi caso sforziamo effettuando l'arrotondamento quindi ho che

$$[< k, g >] \in [0, 255]$$

(cioè significa che la dimensione del kernel è quadrata) -
con questa operazione lineare facendo la media otengo

l'immagine sfocata -

OSSERVIAMO ALCUNE APPLICAZIONI

In Matlab del
fatto di:
convoluzione -

- Vediamo come possiamo sfocare l'immagine

Codice Matlab

```
lena=imread('lena.tif');  
figure; imshow(lena,[0,255]);  
mean3=ones(3); mean3(2,2)=0; mean3=mean3/sum(mean3(:));  
blurred3=conv2(lena, mean3, 'same');  
figure; imshow(blurred3,[0,255]);
```

- Allora

- carico l'immagine e la visualizzo
- uso il kernel 3×3 mettendo nel pixel centrale 0 e gli altri mettendo il valore della media cioè

1/8	1/8	1/8
1/8	0	1/8
1/8	1/8	1/8

In questo modo otengo $[k,g] \in [0,255]$

- il comando conv2 = restituisce la convoluzione bidimensionale della matrice A con il Kernel
da provare com'è
il Kernel
Osserviamo che il terzo parametro
Same restituisce la convoluzione mantenendo la stessa dimensione della matrice (immagine)
full restituisce tutta la convoluzione

Valid restituisce solo le parti delle convoluzioni che sono state calcolate -

- Visualizzo l'immagine sfocata

Graficamente otengo:



Osserviamo che possiamo ottenere una meggiore sfocatura creando un kernel più grande 7×7 oppure ripetendo 5 volte il kernel 3×3 . Cioè:

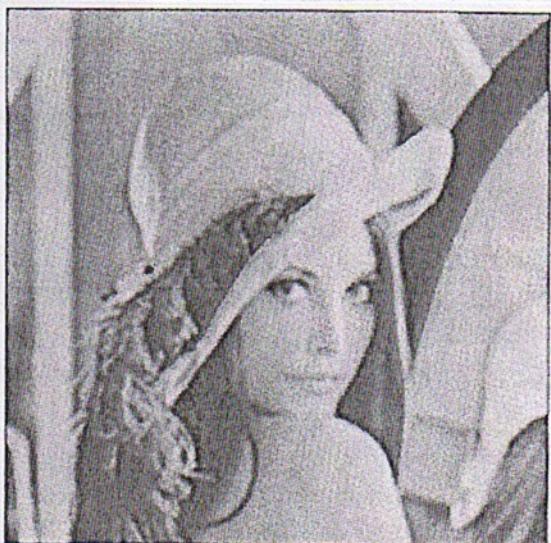
Codice:

```
mean7=ones(7); mean7(4,4)=0; mean7=mean7/sum(mean7(:));  
blurred7=conv2(lena, mean7, 'same');  
figure; imshow(blurred7,[0,255]);  
blurred3b=conv2(conv2(blurred3, mean3, 'same'), mean3, 'same');  
blurred3b=conv2(conv2(blurred3b, mean3, 'same'), mean3, 'same');  
figure; imshow(blurred3b,[0,255]);
```

Visivamente otengo



mean 7×7



mean 3×3 , ripetuto 5 volte

- Se volessimo far scomparire le rughe da l'immagine

Codice Matlab

```
imgfx=(blurred7+double(lena))/2;  
figure; imshow(imgfx,[0,255]);
```

In pratica si prende l'immagine sfocata gli si somma l'immagine di ingresso tramite l'operatore prima introdotto e si divide per due. Quello che ottengo è sempre l'immagine sfocata ma le rughe non mi notano più.

Immagine
grise =
una
immagine =
50 =



Immagine
grise con
effetto collage



- possiamo anche simulare il movimento

Codice Matlab

```
motion9h=fspecial('motion');
moved9h=conv2(lena, motion9h, 'same');
figure; imshow(moved9h,[0,255]);
```

il comando `fspecial` (type) ha un filtro bidimensionale cioè ha un Kernel di un tipo particolare.

I vari tipi possono essere: gaussian, sobel, prewitt, etc...
Nello specifico:

`fspecial('motion', len, teta)` = restituisce 1 filtro per ottenere un movimento lineare di `len` pixel con un angolo `teta` in senso orario.

← il filtro diventa un vettore per movimenti orizzontali e verticali.
Di default `len` è 9 e `teta` è 0 che corrisponde a un movimento orizzontale di 9 pixel.

cioè ottengo:

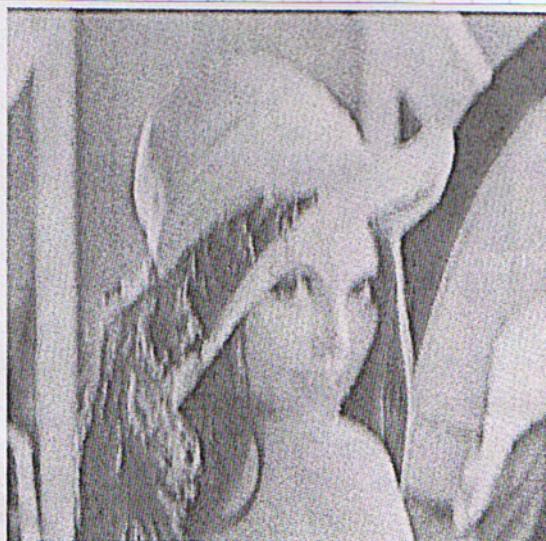
$\begin{matrix} 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 \end{matrix}$ oppure $\begin{matrix} 0 & 1/9 & 0 \\ 0 & 1/9 & 0 \\ 0 & 1/9 & 0 \end{matrix}$
che posso sempre rigiocare un
livello

- Una volta ottenuto questo filtro / kernel faccio la convoluzione con l'essa e lo visualizzo -

Graficamente:



motion: 9 pixels, 0°



motion: 15 pixels, 90°

Se volessi un movimento maggiore come nell'immagine a destra scrivrei:

```
motion15v=fspecial('motion',15,90)
moved15v=conv2(lena, motion15v, 'same');
figure; imshow(moved15v,[0,255]);
```

FILTRI DI CONVOLUZIONE (meglio)

In maggior parte dei filtri utilizzano matrici di convoluzione - con queste matrici è molto possibile costruire dei filtri personalizzati.

Ma cos'è una matrice di convoluzione?

E' possibile avere tante senza usare strumenti troppo specifici - la CONVOLUZIONE infatti è il processamento di una matrice attraverso un'altra matrice che viene chiamata **KERNEL**.

Il filtro "Matrice di convoluzione" utilizza come prima matrice l'immagine da processare (risolve bidimensionale di pixel in coordinate rettangolari) -

Il Kernel usato invece dipende dall'effetto che si vuole ottenere -

I Kernel (matrici) più utilizzati sono quelli 3×3 in quanto sufficienti per la maggior parte degli effetti -

Il filtro esamina in sequenza ogni pixel dell'immagine - Per ciascuno di essi, che denominiamo "pixel iniziale", moltiplica il valore di quest'ultimo e i valori degli 8 pixel confinanti per i valori corrispondenti nel Kernel - I risultati vengono poi sommati e il pixel iniziale viene impostato a questo risultato finale -

Un semplice esempio:

100	100	100	100	100
100	100	50	50	100
100	100	100	100	100
100	100	100	100	100
100	100	100	100	100

\times

0	1	0
0	0	0
0	0	0

=

100	100	100	100	100
100	100	50	50	100
100	100	50	100	100
100	100	100	100	100
100	100	100	100	100

La prima a sinistra è la matrice dell'immagine: ogni pixel è marcato con il suo valore. Il pixel iniziale ha un bordo rosso - L'area di azione del Kernel è quella con il bordo verde - Al centro sta il Kernel e a destra il risultato della convoluzione -

Ecco cosa accade: il filtro legge successivamente da sinistra a destra e dall'alto verso il basso tutti i pixel dell'area di azione del Kernel, moltiplica il valore di ciascuno per i corrispondenti valori del Kernel e poi somma i risultati -

$$\rightarrow (100 \cdot 0) + (50 \cdot 1) + (50 \cdot 0) + (100 \cdot 0) = 50$$

\rightarrow il pixel iniziale assume valore 50.
(il filtro non lavora sull'immagine ma su una copia)

Il risultato grafico è che il pixel si sposta di una posizione in basso.

P.S. I Kernel hanno dimensione quadrata!

la progettazione dei kernel è abbastanza complesse, ma n' trovano kernel già preconfigurati, ne vediamo alcuni esempi

- Aumento del contrasto
(SHARPEN)

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

Si indica

- Sfocature
(BLURRED)

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

- Evidenziazione di bordi
(EDGE)

0	0	0		
-1	1	0		
0	0	0		

- Individuazione di bordi
(EDGE DETECT)

0	1	0
1	-4	1
0	1	0

- Bassorilievo
(EMBOSS ED)

-2	-1	0
-1	1	1
0	1	2

... e molti altri...

$$\begin{aligned} &= (0 \cdot 0.01) + (0 \cdot 0.01) \\ &= 0.02 = (0 \cdot 0.01) + (0 \cdot 0.01) \end{aligned}$$

LEZIONE 3

21/03/07

8 7 6 5 4 3 2 1
128 64 32 16 8 4 2 1

$$\begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{array} = 2 \quad = 117 \quad = 119$$

$$2 \text{ and } 117 = 0$$

$$2 \text{ and } 119 = 2$$

Quindi nel 2° bit avremo 0 se nel 2° bit del 2° numero c'è 0, 2 altrimenti -

Dovranno effettuare degli shift per riportare il tutto nell'intervallo $[0, 1]$

DECONVOLUZIONE

Generalmente le immagini presentano delle deformazioni dovute allo stesso sistema di acquisizione. Il procedimento di deconvoluzione tenta di rendere le immagini il più possibile fedeli alle originali.

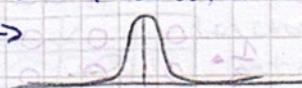
Con molta probabilità potremo applicare le tecniche di convoluzione che siamo vedendo per i segnali monodimensionali (esempi come i suoni) anche alle immagini (quasi tutti se non tutti).

Un esempio di filtro di deconvoluzione potrebbe essere applicato in una distorsione a botte (stile 79, immagine del gatto dietro una bocca di vetro).

Ovviamente per quanto riguarda le immagini, ci saranno scuse degli errori strumentali, e cercheremo quindi di costruire un software per correggere questi tipi di errore.

Un esempio potrebbe essere quello di una immagine inviata dal satellite, questo tipo di immagine è molto sfocata, per aumentare il contrasto vengono applicate ricorsivamente delle tecniche (quindi più volte) finché qualcosa nel cambia modo significativo. Spesso questi algoritmi hanno bisogno di essere applicati anche per ore, e sono detti Algoritmi di deconvoluzione di Lucy-Richardson.

Oppure potrai trovarmi ad avere a che fare con un segnale con 1 picco \rightarrow



che nel monitor è percepito come 1 pixel più smussato (tipo campana gaussiana)

allora dobbiamo aumentare il contrasto in modo da vedere questo picco (il metodo deve prendere in considerazione però anche la posizione del pixel che ha il picco).

Point Spread Function = Funzione che ci dà indicazioni circa quanto viene smussato un pixel (o)

(in genere più siamo vicini al bordo più il picco viene sparpagliato)

c'è anche da dire però che modificando il contrasto allo

stesso tempo migliori e peggiori le situazioni - Migliori (nel pixel stesso) e peggiori (intorno)

E 3/0/33

Generalmente le immagini presentano delle deformazioni dovute allo stesso sistema di acquisizione.

$$I'_D(i,j) = \sum_{x=-w/2}^{w/2} \sum_{y=-h/2}^{h/2} I_D(i+x, j+y) \cdot k(x, y) + N(i, j)$$

Ad esempio, in un kernel $3 \times 3 \rightarrow x = [i-1, i+1]$
(N dispari)

$$y = [j-1, j+1]$$

" " $7 \times 7 \rightarrow x = [i-3, i+3]$
 $y = [j-3, j+3]$

Il procedimento, in genere iterativo, tutta così di risolvere un sistema di $w \times h$ equazioni in $w \times h$ incognite.

Se N è zero, allora il sistema ammette soluzioni esatte.

Se l'operatore di deconvoluzione è omogeneo e lineare, allora può essere scritto in forma vettoriale:

$$D' = K \otimes D + N \rightarrow \text{costante additiva che viene aggiunta al risultato finale}$$

prodotto scalare \downarrow se $N=0 \rightarrow$ sistema omogeneo

Un sistema si dice lineare invece quando:

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

oppure:

$$f(\alpha x + \beta y, z) = \alpha f(x, z) + \beta f(y, z)$$

Ma facciamo esempio di applicazione delle formule D'
se dovo fare la media per le 4-connettività avrò:

0	114	0
114	0	114
0	114	0



$\frac{1}{4} \cdot$	0	1	0
	1	0	1
	0	1	0

(stessa cosa ma non dovrà trattare virgole mobili di continuo)

Oppure, nel caso del kernel seguente:

-1	-1	-1
0	0	0
0	0	0

0	0	0
0	0	0
1	1	1

Potrei ~~ottenerne~~ ottenere valori che vanno da -765 a 765
avendo nelle prime matrice -255 nelle prime riga, e nella 2^a, 255 nelle 3^a riga.

Ma con 765 riamo ancora molto lontani dal range [0, 255] che vogliamo.

\rightarrow divido per 3 e ottengo

$\frac{1}{3} \cdot$	-85	-85	-85
	0	0	0
	0	0	0

$\frac{1}{3} \cdot$	0	0	0
	0	0	0
	85	85	85

ho ottenuto quindi un range di valori che va da -255 a 255

Ma sono ancora lontano \rightarrow

divido ulteriormente per due $\rightarrow \frac{1}{6} \cdot$

$\frac{1}{6} \cdot$	-42,5	-42,5	-42,5
	0	0	0
	0	0	0

$\frac{1}{6} \cdot$	0	0	0
	0	0	0
	42,5	42,5	42,5

ho ottenuto un range di valori che va da -127,5 a 127,5

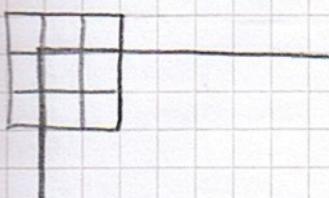
A questo punto aggiungo 128 (costante additiva) e, a meno di approssimazioni (dovendo ottenere un numero intero) otterro' il range che cercavo $\rightarrow [0, 255]$

la deconvoluzione è quindi l'esatto opposto della convoluzione, ma, a differenza di queste, non conduce sempre a risultati ottimali / accettabili.

IL PROBLEMA DEL BORDO

Ho un'immagine e una matrice kernel 3×3 .

Per come abbiamo definito l'applicazione del filtro di convoluzione con il kernel 3×3 , ci accorgiamo subito che c'è tutto un bordo attorno all'immagine, in cui NON SAPPIAMO calcolare il valore dell'immagine.



Per un kernel 3×3 questo bordo non calcolabile sarà grande 1 pixel.

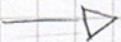
Per un kernel 7×7 " " 3 pixel di bordo.

Per ovviare a questo problema vengono in seguito riportate delle soluzioni che adottano approcci diversi risolvendo a modo loro meglio o peggio il problema in questione.

1 Il primo metodo proposto è quello di NON calcolare la convoluzione per i pixel del bordo, fermandosi quindi un po' prima.

Effettuo quindi la convoluzione esclusivamente sui pixel su cui ho la certezza di avere tutti i pixel dell'intorno che mi servono (8 in questo caso).

Nelle cornici andrò a mettere il valore 0 = nero.



È però un metodo un po' brutale

Il risultato è una cornice nera

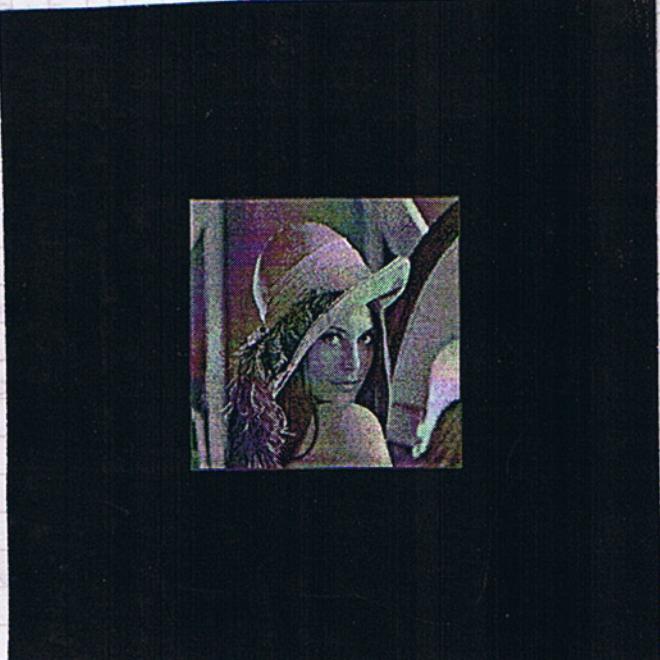
2 Applico il filtro all'immagine pensandole già con una cornice nera significativamente grande tutta intorno a sé. Applico quindi il filtro di convoluzione fino a 1 certo bordo, e all'origine questa da quanto poi sarebbe diminuita.

0	1	0
1	0	1
0	1	0

→ nelle parti di bordo il risultato non è corretto perché mi trovo ad avere zeri (nero) tutti intorno.

Noterò sempre nel risultato una cornice nera, ma adesso questa apparirà più sottilata.

Vediamo l'esempio →

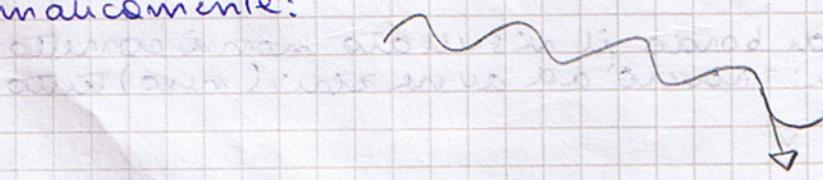


- 3 Il terzo metodo consiste nel propagare le ultime righe e colonne delle matrice dell'immagine (i pixel di bordo) per "rivestire" il risultato lungo il bordo, ovvero assegnare valori che n'assomigano a quelli reali.
Questo metodo è abbastanza buono per kernel di dimensioni piccole (come i 3×3), in quanto ci aspettiamo valori simili.

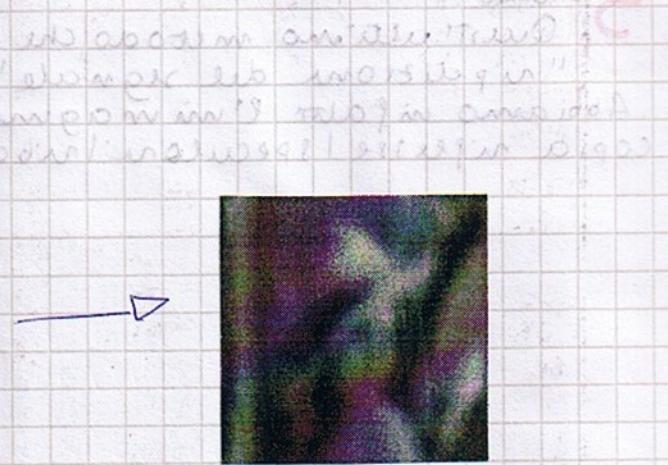


d'effetto al bordo come n'può notare sta' diminuendo ma abbiamo aggiunto informazioni che prima non c'erano.

Matematicamente:


tangente -
Ovvero rivesto 1 valore più opportuno

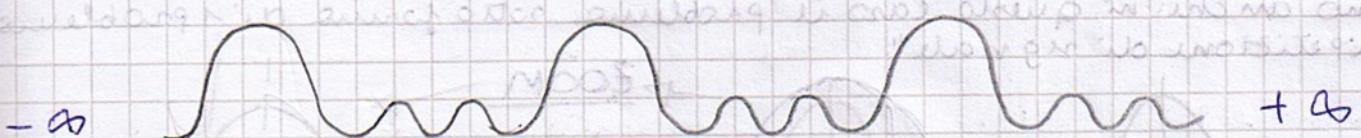
- 4 Una scelta ulteriore potrebbe essere quella di ripetere l'immagine attorno all'immagine stessa



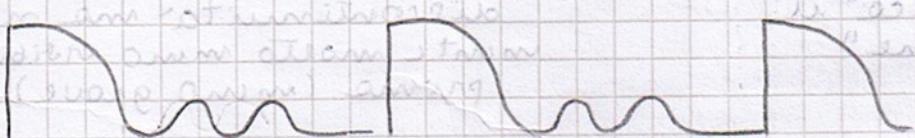
In questo modo però non
ci ha una continuità
"naturale".
Come si può notare dai
bordi per via dei contrasti
troppo repentina.

Questo metodo ci risulta alle mie trasformate di Fourier
dovute e applicate ai segnali periodici.
Per "segnali" intendiamo sul vago e intendiamo anche quelli sonori
oltre che visivi.

Vediamo quindi un esempio di segnale periodico:



(riimmagini che il segnale sia ripetuto all'infinito, altrimenti
non è possibile pensare di poter applicare Fourier)
Applico quindi i filtri di convoluzione, e ottengo segnali di
questo tipo:



ottengo continuamente un salto (de evitare) dovuto all'impiego
di cambi di bianco/nero (ad esempio).

Quindi questo metodo non è poi molto efficiente, in quanto dobbiamo
1 pixel chiaro/scurro va a mettere 1 pixel scuro chiaro rispettivamente
→ tiene poco conto della "continuità" del colore.

Nel caso di salti di questo tipo, ci vuole un grande (ma finito)
numero di coefficienti per costruire il segnale nel caso
discreto, mentre noi avevamo lavorato nel caso continuo.

5

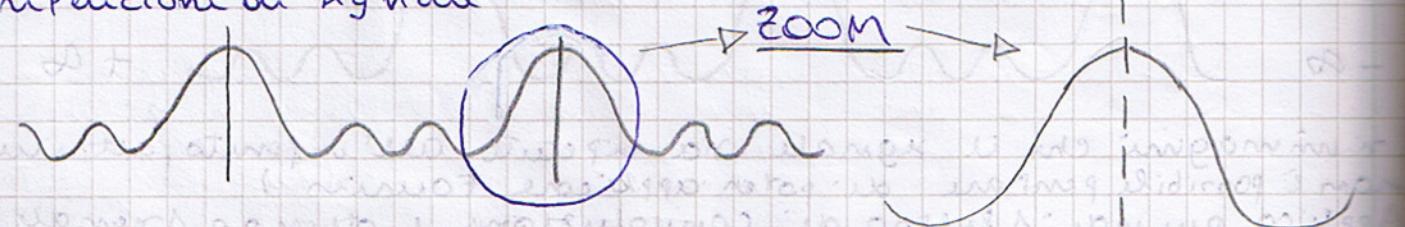
Quest'ultimo metodo che vediamo si basa sul concetto delle "ripetizioni del segnale".

Abbiamo infatti l'immagine principale affiancata da un'altra copia ripetuta specularmente.



questo metodo è il migliore per la risoluzione del problema del bordo.

Vediamo anche in questo caso il problema sotto forma di problema di "ripetizione di segnale".



Questo discorso in un certo senso "giustifica" il parametro "Same".

si nota che c'è sempre una discontinuità ma sicuramente molto meno visibile di prima (meno grave).

"Same" = Mantiene le dimensioni di output uguali.

- Scrivere un programma che fa le convolutioni applicando i metodi scritti prima per il problema del bordo.

(attualmente) non esiste un algoritmo che lo fa.

ALTKI TILIKI VI CONVOLUZIONE

Vediamo inoltre alcuni comandi:

» J = imnoise (I, type, ...)

Aggiunge un disturbo (noise) ad una immagine -
il tipo di disturbo "type" viene
aggiunto all'immagine I -

Type è una stringa che può assumere uno dei seguenti valori:

- 'gaussian' → disturbo gaussiano bianco con media e varianza variabili
- 'localvar' → disturbo bianco con varianza uguale a quella del pixel
- 'poisson' → disturbo poisson
- 'salt & pepper' → pixel accesi e spenti (On = OH)
- 'speckle' → disturbo moltiplicativo

Possono essere specificati dei parametri addizionali - Questi sarebbero dei numeri e corrispondono ad operazioni sull'immagine e variano da 0 a 1 -

Potrei mettere media e varianza per la 'Gaussian' o un secondo terzo parametro per salt & pepper che indica la densità del disturbo.

» h = fspecial (type)

Cioè dei filtri speciali a due dimensioni (cioè un filtro speciale che assegna col h, di tipo specifico)

Possibili valori di type sono:

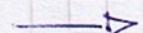
- 'average' → filtro delle medie
- 'disk' → filtro media circolare
- 'gaussian' → gaussiano
- 'laplacian' → approssima l'operatore laplaciano
- 'log' → filtro laplaciano di Gauss
- 'motion' → motion
- 'prewitt' → enfatizza linee orizzontali
- 'sobel' → sobel horizontal
- 'unsharp'

fspecial può avere molte parametri optionali -

possiamo specificare hsize, che mi dà un filtro h di dimensioni h size - Posso scrivere come [5 5] o come uno scolare (in questo caso viene visto di dimensioni quadrate come lo scolare) e se non scalo nulla di default hsize = 3x3.

Troviamo anche sigme, che specifica una rotazione con deviazione standard sigma (positiva). Può essere in valore o uno scolare allo stesso modo, e il suo valore di default è 0.5. Abbiamo anche len e Teta -

Ma vediamo qualche applicazione di questi comandi -



```
noised=imnoise(lena,'salt & pepper',0.1);  
figure; imshow(noised,[0,255]);
```



"salt & pepper" noise (10%)

```
denoised=conv2(noised, mean3, 'same');  
figure; imshow(denoised,[0,255]);
```

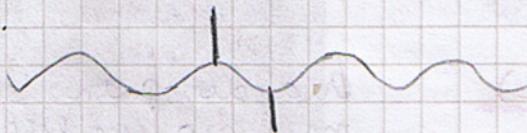


denoised (mean 3x3)

"noise" sta per disturbo - rumore -

Il rumore sale e pepe possiamo veduto come dei "picchi" a salire e a scendere.

SALE E PEPE



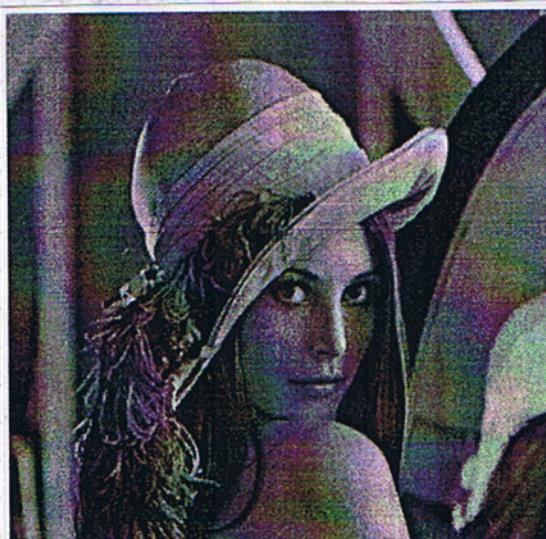
(FILTRI DI SMOUSSAMENTO)

Quinto tipo di rumore (detto anche rumore casuale indipendente) è dovuto alle presenze di pixel che presentano un colore molto diverso da quello dei pixel circostanti, il nome è legato al modo in cui il rumore si manifesta in una immagine in bianco e nero.

Il numero di pixel alterati è comunque molto inferiore rispetto al numero totale di pixel. Cause del rumore possono essere in genere di polvere su una lente o in alcuni elementi CCD difettosi in una fotocamera digitale.

denoised → risulta qui smussato, ma purtroppo per ridurre il disturbo viene smussato anche il resto -

```
gaussian3=fspecial('gaussian',3);  
blurred4=conv2(lena, gaussian3, 'same');  
figure; imshow(blurred4,[0,255]);
```

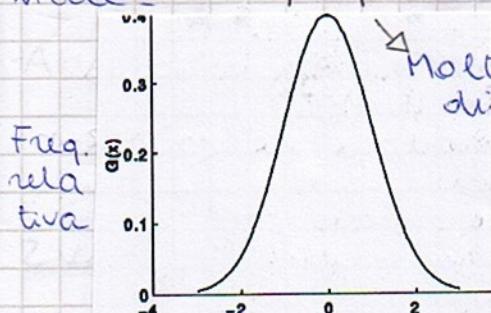


GAUSSIAN

```
gaussian9=fspecial('gaussian',9);  
denoised2=conv2(noised, gaussian9, 'same');  
figure; imshow(denoised2,[0,255]);
```



Il rumore gaussiano è detto anche "rumore dipendente" e migliora su tutti i pixel dell'immagine -
Ogni pixel risente di un cambiamento rispetto al suo valore originale -



$$(monodim) \quad \text{Gauss}(x) = \frac{e^{-\frac{x^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}$$

$$\mu=0, \sigma=1$$

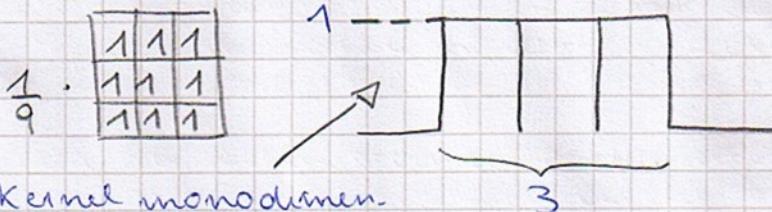
pochi pixel con distorsione ampia

Il filtro gaussiano è un "filtro di smoothing" o "smoothing = rendere una superficie più uniforme, liscia") - Questo tipo di filtri eliminano picchi e increspature ma di contro possono sfocare eccessivamente l'immagine - (oltre il filtro gaussiano possiamo menzionare il filtro media)

Il nuovo valore del pixel sarà la media pesata dei valori del suo vicinato, eletti per i quali saranno distribuiti appunto secondo una funzione gaussiana -

Più larga è la campana, maggiore sarà l'effetto di smoothing.

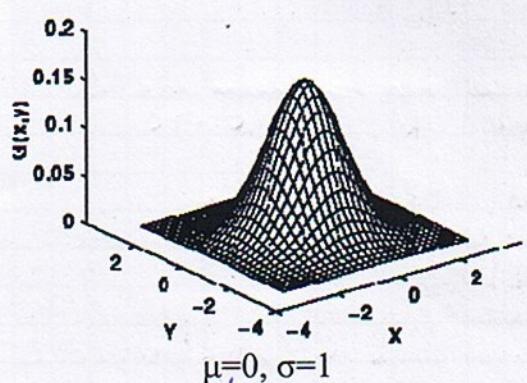
Nell'esempio, gaussian3



Ora, anche con questi picchi possiamo immaginare il kernel come un Kernel gaussiano , in questo caso quando sfoca l'immagine le sfocano di meno

Il filtro di Gauss è a variabili separabili - Inoltre, grandi finestre contengono molti valori nulli e possono essere approssimate con successivi filtri -

$$\text{Gauss}(x, y) = \frac{e^{-\frac{x^2+y^2}{2\sigma^2}}}{2\pi\sigma^2}$$



vuol dire "entrato nell'origine"

In questo caso però ci accorgiamo che per un'immagine nxn ci troviamo a dover effettuare n^2 prodotti, ovvero

Vediamo una strategia per migliorare le cose ossia

Il filtro di Gauss può essere monodimensionale e bidimensionale

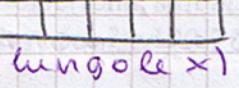
Per il caso monodimensionale vale le regole $\text{Gauss}(x)$ scritte sopra, e in questo caso il nostro kernel non sarà altro che un vettore riga (colonne) come ad esempio il seguente :

.006	.061	.242	.383	.242	.061	.006
------	------	------	------	------	------	------

CASO BIDIMENSIONALE

In questo caso vale le regole di $\text{Gauss}(x, y)$ scritte qui accanto, e il kernel in questa caso sarà una matrice come la seguente, ad esempio:

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

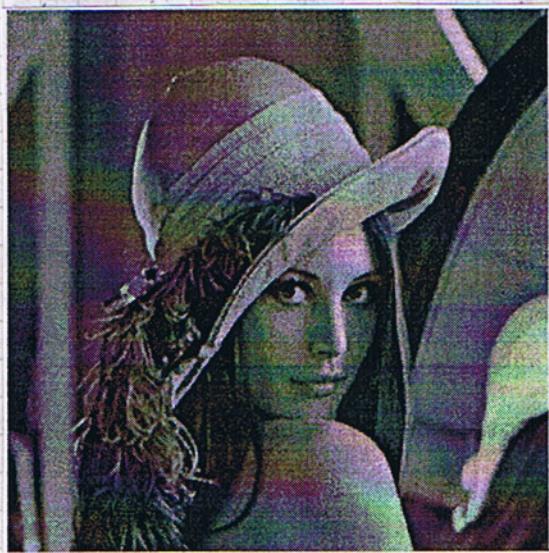
Ovvvero, niente di applicare il kernel quadrato applico il vettore
primo su  e poi su  (lungo le x) (lungo le y)

In questo modo dovremo fare due "passate" me i anche verso che avremo ridotto il numero dei prodotti da $n^2 \times n$ (nel caso del kernel 5×5 , da 25 a 10). Uno cresce al quadrato e l'altro cresce linearmente - (avere kernel separati).

Più grosse sono le dimensioni, più costoso.

FILTO MEDIAN

`median3=medfilt2(lena);
figure; imshow(median3,[0,255]);`



blurred (median 3×3)

`denoised3=medfilt2(noised);
figure; imshow(denoised3,[0,255]);`



denoised (median 3×3)

$\Rightarrow B = \text{medfilt2}(A)$

applica un filtro "media" sulla matrice A usando un Kernel 3×3 (di default), lo sostituisce nell'immagine B.

$\Rightarrow B = \text{medfilt2}(A,[M\ N])$

stessa cosa puoi in questo caso possiamo specificare le dimensioni $M \times N$ del Kernel.

Il filtro mediano rappresenta l'altra tecnica per eliminare il rumore, si è un'operazione volata nel insieme di vicinanza (locale).

Dato il pixel corrente, si considerano i pixel di tale vicinanza, si ordinano secondo i loro valori e si sostituisce al pixel corrente il loro mediano (livelli di grigio).

ESEMPIO Maschera 3×3 - Il pixel che vogliamo elaborare ha valore 55 e i circostanti hanno i seguenti valori:

10		
5	55	6
15		

Si ordinano i valori ottenendo la seguente lista:
5, 6, 10, 15, 55

Il mediano, ovvero centrale, è il valore che si trova al centro della lista $\rightarrow 10$.

d'uscita del filtro sarà dunque 10, che andrà a sostituire il valore 55 (probabilmente erroneo)

Ho quindi preso il valore di posizione [n12] in una lista di n elementi (o vettore)

Attenzione! mediana e media NON sono la stessa cosa

Nell'esempio precedente se avessi voluto la media avrei dovuto fare $(5+6+10+15+55)/5 = [18,2] = 18$ mentre con la mediana abbiamo 10.

- SVANTAGGI** → 1 - Con la mediana, il valore che userò per sostituire, appartiene già all'insieme dei valori di partenza, e quindi è un valore di grigio già presente nell'immagine
2 - non devo fare approssimazioni

Il filtro mediana quindi smussa l'immagine ma meno brutalmente di quello gaussiano - Ho quasi del tutto tolto il rumore e l'immagine somiglia molto a quella di partenza

FILTRO SHARPEN

(FILTRI DI RINFORZO)

```
sharpen=fspecial('unsharp');  
sharpened=conv2(lena, sharpen, 'same');  
figure; imshow(sharpened,[0,255]);
```

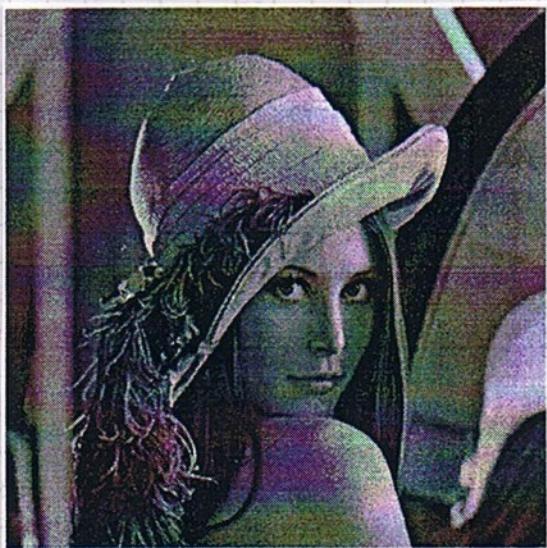


Immagine originale



Immagine sharpened

E' un filtro (kernel) di "rinforzo" (sharpening) che ha l'obiettivo di rendere più evidenti i picchi e le microspatature (ad es. sui bordi) e quindi di aumentare/migliorare il contrasto dell'immagine

"Sharp" → contrasto, rinforzo

Migliora quindi la percezione dei dettagli

Di solito sono composti da meschini (kernel) che hanno il valore centrale positivo (e alto) e i valori laterali negativi

-1	-1	-1
-1	8	-1
-1	-1	-1

→ ad esempio, (anche per evidenziare il rumore sale & pepe)

Per pixel scuri e isolati basterebbe invertire i segni



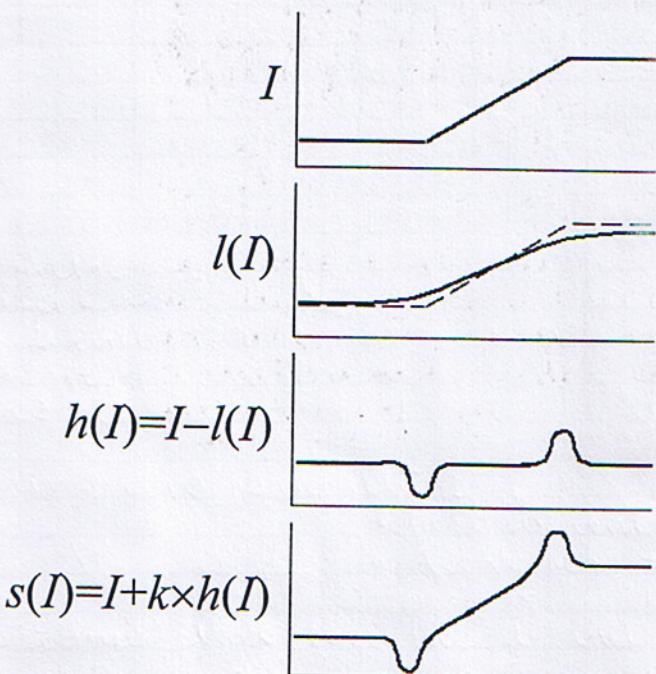
→ i picchi a salire rappresentano i pixel bianchi e i solletti

(→ con segni invertiti basterebbe fare la convolution per uno e moltiplicare il risultato per -1) visto che con il filtro sharpen vogliamo mettere in evidenza le alte frequenze, possiamo pensare di utilizzare una versione "smussata" (che ridecherà con $\rightarrow \text{low}$)

Definizione:

"Filtro passa basso" → è 1 filtro che fa passare le basse frequenze, cioè l'immagine (un esempio è il filtro mediano)

"Filtro passa alto" → è 1 filtro che fa passare le alte frequenze, cioè i dettagli (comuni al filtro ~~sharpen~~ sharpen)



(il filtro amplifica quindi le alte frequenze dell'immagine (mantiene i dettagli dell'immag.)

→ Immagini monodimensionali

Applico un filtro passa basso, ho

→ Versione smussata, $l = \text{low}$

→ Faccio le differenze - Annullo mantenendo le strutture più piccole e ho $h(I)$

→ sommo $h(I)$ all'immagine I (1° grafico) e ottengo $s(I)$ → filtro di sharpen. K mi dice quanto deve essere alto. (Più di 3 volte non conviene farlo perché si genera)

$K=1$
E.S.

$$2. \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{17}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(ma anche)

$$I + I - I(l) = 2I - I(l)$$

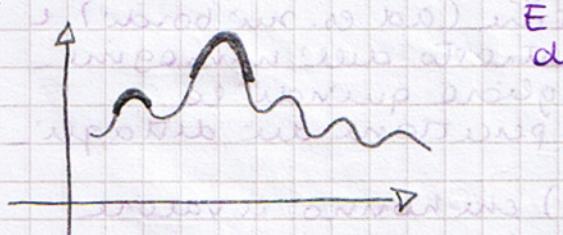
con la media 3×3 , 8 campioni

nuovo kernel che in 1 colpo solo aumenta il contrasto

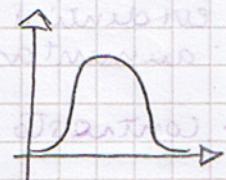
Abbiamo sfruttato anche la linearità del Kernel -

In generale possiamo dire che quando vogliamo mettere in evidente particolari strutture allora dobbiamo usare dei kernel che ne avvicinano a queste strutture (sono comunque concetti che approfondiremo più avanti parlando di "wavelet")

Ad esempio, se ho 1 segnale monodimensionale di questo tipo:



E uso un kernel di questo tipo →



la convoluzione fa che il prodotto scalare i due, e siccome il kernel "appatta" con i valori dell'immagine, m'inviano ad altre valori molto alti -

FILTRO POINTLIKE EDGE

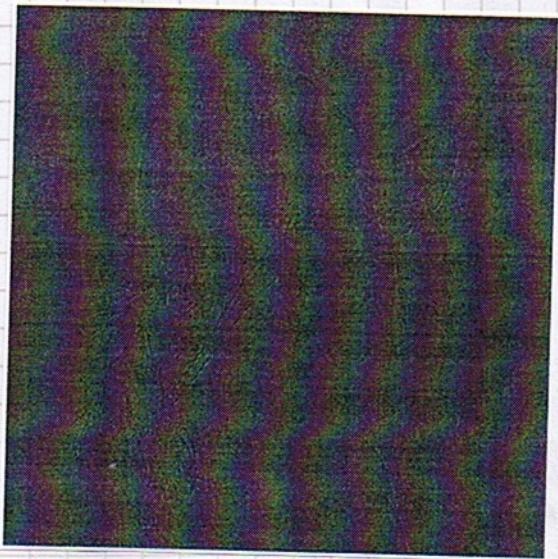
"Edges" → bordi/margini

Questo filtro evidenzia i bordi dell'immagine, li fa più chiari

```

dark=[1,1,1; 1,-8,1; 1,1,1];
darkedge=conv2(lena, dark, 'same');
figure; imshow(darkedge,[-2040,2040]);

```

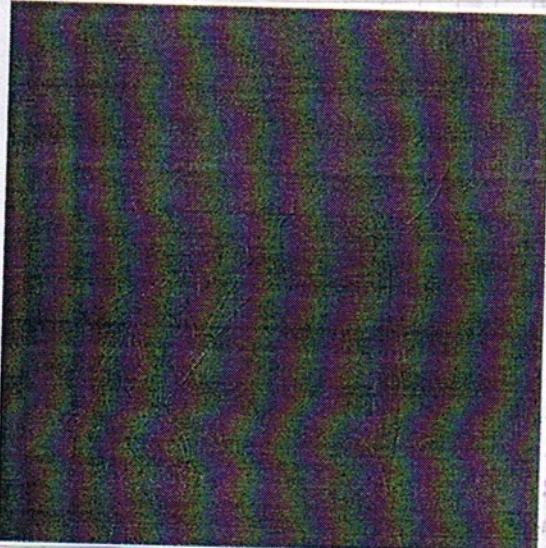


"dark edges"

```

brightedge=-darkedge;
figure; imshow(brightedge,[-2040,2040]);

```



"bright" edges

Dovrebbero (...) essere visibili i bordi "luminosi" dell'immagine

Nota: per migliorare le qualità di questi risultati si consiglia di specificare in imshow l'intervallo di grigio [-100,100]

FILTRO GRADIENTE (Sobel)

Un gradiente è un insieme di colori posti in ordine lineare -
Un filtro gradiente può permettere di "colorare" un'immagine a scale di grigi, sostituendo ogni sfumatura di grigio con un colore corrispondente presso ad un gradiente -

→ lo usiamo per l'individuazione dei contorni.
Effettuiamo la derivata prima dell'immagine:

$$\Delta I = \frac{\partial I}{\partial x} i + \frac{\partial I}{\partial y} j \rightarrow \text{gradiente di una funzione continua e derivabile di due variabili reali.}$$

ricordiamo che la derivata rappresenta il limite del rapporto incrementale

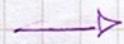
$$\lim_{x \rightarrow x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

Nel caso discreto le componenti diventano :

$$\frac{\partial D}{\partial x} \approx D(i, j+1) - D(i, j) = \Delta_x D \quad \frac{\partial D}{\partial y} \approx D(i+1, j) - D(i, j) = \Delta_y D$$

e il modulo del filtro è definito come:

$$\|I\| = \sqrt{\Delta_x^2 I + \Delta_y^2 I} \quad \alpha = \arctan \frac{\Delta_y I}{\Delta_x I}$$



X ₀		
X		

se a vado de sotto il più vicino è questo

Ma mi pare che poche, questo filtro,
cosa è? Che fa?

FILTRO LAPLACIANO

Il filtro gradiente e quello laplaciano sono un tipo di filtri passa-alto basati su approssimazioni di derivate prime (per il gradiente) e seconde (laplaciano) spaziali.
Servono per mettere in evidenza dove l'immagine varia fortemente
oppure per mettere in risalto i contorni per esempio.

In un filtro passa alto:

- Peso al centro = 1 - peso al centro del passabasso
- Altri pesi = 0 - pesi corrispondenti al passabasso

Definiamo il laplaciano di una funzione continua e derivabile due volte, di due variabili reali:

$$\Delta^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Nel caso di filtro le componenti diventano:

$$\begin{aligned}\frac{\partial^2 D}{\partial x^2} &\approx \frac{\partial(D(i, j+1) - D(i, j))}{\partial x} = \frac{\partial D(i, j+1)}{\partial x} - \frac{\partial D(i, j)}{\partial x} = \\ &= (D(i, j+2) - D(i, j+1)) - D(i, j+1) - D(i, j) = \\ &= D(i, j+2) - 2D(i, j+1) + D(i, j)\end{aligned}$$

$$\frac{\partial^2 D}{\partial y^2} \approx D(i+2, j) - 2D(i+1, j) + D(i, j)$$

pixel a destra pixel a sinistra

Effettuando la traslazione mi (*i*, *j*):

$$\frac{\partial^2 D}{\partial x^2} \approx D(i, j+1) - 2D(i, j) + D(i, j-1) = \Delta_x^2 D$$

$$\frac{\partial^2 D}{\partial y^2} \approx D(i+1, j) - 2D(i, j) + D(i-1, j) = \Delta_y^2 D$$

Effettuo la traslazione mi quanto la formula è corretta come mi sono spostato troppo rispetto al pixel (*i*, *j*) quindi traslo l'immagine per diminuire questo spostamento.

Da cui:

$$\Delta_x^2 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad \Delta_y^2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \Delta_{45^\circ}^2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \Delta_{135^\circ}^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Δ_{45° e 135° li riceviamo con una semplice rotazione-

deploriamo \rightarrow

0	-1	0
-1	4	-1
0	-1	0

FILTO GRADIENTE (SOBEL) E APPLICAZIONI

$$\Delta_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \Delta_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Vertical Edge

Horizontal Edge

Ma vediamo qualche applicazione -

Possiamo ricordare infatti che i picchi verso il basso rappresentano il passaggio da una zona scura ad una zona chiara e viceversa -

```
sobelk=[-1,-2,-1; 0,0,0; 1,2,1];
sobelx=conv2(lena, sobelk, 'same');
figure; imshow(sobelx,[-1020,1020]);
```

```
sobely=conv2(lena, sobelk', 'same');
figure; imshow(sobely,[-1020,1020]);
```



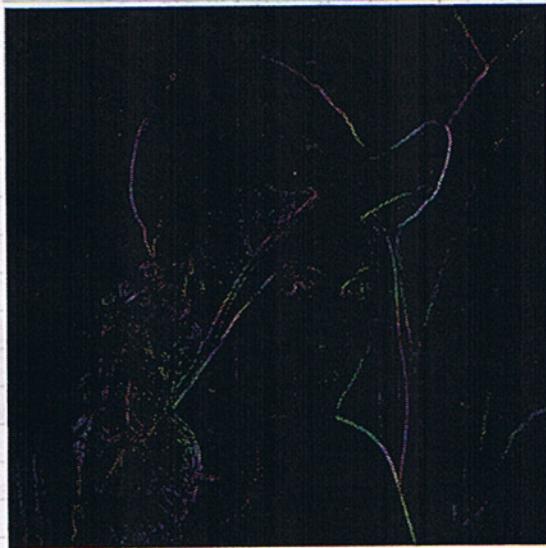
Sobelx (componente orizzontale)



Sobely (componente verticale)

Altre applicazioni:

```
sobelm=sqrt(sobelx.^2 + sobely.^2)./sqrt(32);
figure; imshow(sobelm,[0,255]);
```



Euclidean Magnitude

per riportare i valori tra 0 e 255

```
sobelm2=(abs(sobelx) + abs(sobely))./8;
figure; imshow(sobelm2,[0,255]);
```



"city block" magnitude

Prendiamo il valore assoluto del risultato.

Se vogliamo prendere le transizioni di $\Delta y \rightarrow |\Delta y|$
 " " " " " " " " " " $\Delta x \rightarrow |\Delta x|$

pixel più vicini pesano di più.

Se invece vogliamo prendere entrambe le transizioni allora siamo nel caso di:

$$\frac{|\Delta x| + |\Delta y|}{2}$$

\rightarrow faccio la media mettendo in evidenza
 le componenti sia orizzontali che
 verticali, e lo faccio sia da chiaro e
 scuro che viceversa

Quindi le condizioni dipendono da quello che vogliamo:

- destra \rightarrow sinistra
- sinistra \rightarrow destra
- chiaro \rightarrow scuro
- scuro \rightarrow chiaro

(si può vedere anche come
 discorsi su transizioni con
 la gaussiana)

$$|\Delta x| + |\Delta y| \approx \sqrt{|\Delta x|^2 + |\Delta y|^2}$$

\rightarrow quanto più possibile uguale a

Sobelx \rightarrow componente orizzontale
 Sobely \rightarrow componente verticale

$$S_{0^\circ} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, S_{45^\circ} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, S_{90^\circ} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, S_{135^\circ} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix},$$

$$S_{180^\circ} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, S_{225^\circ} = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}, S_{270^\circ} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, S_{315^\circ} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Orientazione del gradiente:

```
sobel = atan2(sobely, sobelx);
figure; imshow(sobel, [-pi, pi]);
```



Immagine Originale



Orientation

$\Rightarrow \text{atan2}$

«Orientamento del gradiente
(arctangente)

Gli algoritmi visti fin qui possono essere applicati sulle singole componenti colore e i risultati possono essere composti in un'immagine a colori.

FILTRO GRADIENTE (PREWITT)

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

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

(a)

(b)

(a) \rightarrow da scuro a chiaro evidenziando la componente verticale

(b) \rightarrow da chiaro a scuro evidenziando la componente orizzontale

$$P_{0^\circ} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, P_{45^\circ} = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, P_{90^\circ} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, P_{135^\circ} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix},$$

$$P_{180^\circ} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, P_{225^\circ} = \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}, P_{270^\circ} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, P_{315^\circ} = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Ovviamente possiamo calcolare per il filtro Prewitt tutti i possibili kernel che si ottengono effettuando rotazioni di 45° .

LINE DETECTION

→ rilevamento di linee (orizzontali o verticali)

$$l_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad l_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad l_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & -1 & 2 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad l_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$l_5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad l_6 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad l_7 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad l_8 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 2 & 2 & 2 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Con il 3×3 puoi avere delle limitazioni per quanto riguarda le rotazioni, che dovranno essere di 45° .

Per comodità estendo il filtro di Previtt a un Kernel 5×5 .

In questo modo si vedono meglio le rotazioni e posso usare anche angoli con multipli di $22,5^\circ$.

GRADIENTE → In generale sta per filtro direzionale

Con le line detection possiamo mappare il segnale ricordando che quando il Kernel "appoggia" con il segnale abbiamo valori alti. Se supponiamo di mappare le linee dirigate nell'immagine avremo

0	0	255
0	255	0
255	0	0

autore di Line detection

A questo punto il kernel per poter mettere in evidenza ciò come dovrà essere?

$$\begin{bmatrix} 0 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & 0 \end{bmatrix} \rightarrow -1 = 2 \times \text{compensano}$$

Nel caso di linee verticali l'immagine sarebbe:

(d)

0	255	0
0	255	0
0	255	0

→ il kernel

0	0	0	0	0
0	-1	2	-1	0
0	-1	2	-1	0
0	-1	2	-1	0
0	0	0	0	0

LEZIONE 4

28/03/07

CACHE MISS E PRINCIPIO DI LOCALITÀ'

Nelle simulazioni in Matlab al comando `conv2`, ovvero delle convolutioni lento usare funzioni predefinite, abbiamo dovuto di sicuro scorrere le nostre immagini in altezza e larghezza per applicare riguadagnando il kernel.

Abbiamo quindi di sicuro dovuto applicare 2 cicli for -

Potrò applicare nei 2 modi:

- `for j=1:h
for i=1:w`

→ detta "Scansione Raster"

- `for i=1:w
for j=1:h`

→ che è una scansione per righe

Ma in quest'ultimo caso, per spostare il kernel lungo l'immagine devo fare $h \cdot 8$

Se l'immagine ad esempio ha $h = 512$ (cosa) avrà $512 \cdot 8 = 4096$ (caso) che sono in pratica 4 mega -

Ma normalmente le istruzioni vengono eseguite in sequenza e dopo l'altro (a meno di jump) ed eseguire una scansione per righe, visto che le immagini sono normalmente munite come righe e colonne, vuol dire provocare una cache miss ad ogni iterazione. Le memorie cache come ricordiamo non ci mettono altro che le parti di memoria temporanea le cui caratteristiche principali sono le piccole dimensioni e la velocità (nel prossimo esempio) -

Ora, è ovvio che finché eseguo istruzioni nella cache anch'esse molto veloci, ma se ne esco ad ogni iterazione girandomi da "cache miss" falso perdere molto tempo alla macchina -

(un esempio riportato potrebbe essere i paletti di paglia per i blocchi di memoria, ma qui parliamo di cache la cui dimensione è molto più piccola)

Quanto continuo "saltare" diminuisce notevolmente le prestazioni del nostro algoritmo.

Cache Miss → Memoria Cache non colpita -

In pratica ad esempio viene richiesto un dato alla cache, ma questo non si trova ed è busi nelle memorie principale. Il fallimento succede quindi al tempo, in quanto richiede il trasferimento del dato dalla memoria principale, il cui tempo di risposta è molto maggiore di quello delle memorie cache -

IL PRINCIPIO DI LOCALITÀ': Afferma che, se la CPU sta eseguendo una data istruzione, vuol dire che con ogni probabilità le prossime istruzioni da eseguire saranno ubicate nelle vicinanze di quelle in corso. Vale a dire che, durante la normale esecuzione di un programma, la CPU passa molto tempo accedendo a zone di memoria strette e solo occasionalmente accede a locazioni molto lontane

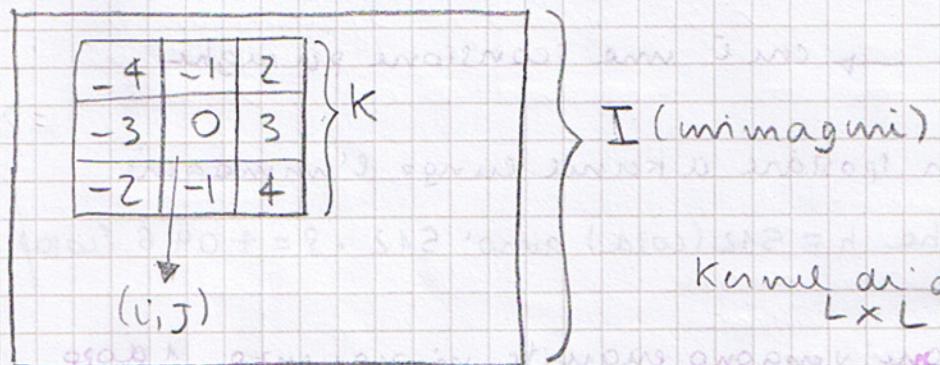
Quindi scriviamo il nostro algoritmo in modo da scorrere la matrice delle immagini per colonne e non per righe.
(molte meno cache miss)

Inoltre, dovremo cercare di scrivere i nostri algoritmi in modo da evitare l'uso di funzioni specifiche del Matlab ma di unire funzioni più intuite come celle in un linguaggio standard di programmazione.

METODO PER SCRIVERE LA CONVOLUZIONE (LookUpTable)

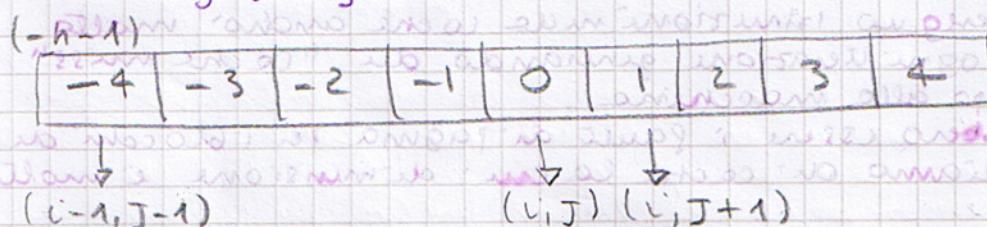
(nona fare uso delle funzioni specifiche come il prodotto puntuale in Matlab tra matrici)

Numero il Kernel che indicheremo con la lettera K .



creando una Look Up Table (LUT)

In informatica per Look Up Table si intende una struttura dati, che è generalmente un array, usata per sostituire operazioni di calcolo con una più semplice operazione di consultazione. Si usa anche per ottenere un guadagno in termini di velocità.



In pratica non faccio altro che precalcolarmi le coordinate di tutti questi pixel che mi servono per fare il prodotto puntuale.

Senza preoccuparmi dei bordi.

Ad esempio $a = 1 : (h \cdot w)$

for $b = 1 : L^2$

$I(a + lut(b)) \circ K(b)$

Ovviamente dovrò poi fare le opportune somme.

Chiamiamo P il puntatore al pixel centrale applicando il Kernel all'immagine.

ad esempio $P + lut(b)$

nel primo ciclo: $P + lut(1)$

$b = 1 \rightarrow P - h - 1$

$b = 2 \rightarrow P - h$

$b = 3 \rightarrow P - h + 1$

$b = 4 \rightarrow P - 1$

$b = 5 \rightarrow P$

$b = 6 \rightarrow P + 1$

$b = 7 \rightarrow P + h - 1$

$b = 8 \rightarrow P + h$

$b = 9 \rightarrow P + h + 1$

Facendo quindi questo for + nous questi elementi -
Abbiamo così ottenuto il prodotto puntuale senza ricorrere a
istruzioni specifiche del Matlab.

Riflettiamo su:

a	b	c
d	e	f
g	h	i

b	c	a
e	f	d
h	i	g

c	a	b
f	d	e
i	g	h

ROTAZIONE DI IMMAGINI

Vediamo ora che problemi incontriamo nelle rotazioni di una immagine -

Se mostriamo una immagine di un angolo α , il "quadro" che la contiene (visto come rettangolo) dovrà essere più grande dell'immagine di partente, mentre i pixel dell'immagine stessa dovranno essere "riempiti" per andare a formare le nuove immagini rotate. Saranno quindi le formule che circondano per combinare il sistema di riferimento.

$$\begin{cases} X = n \cos \alpha - y \sin \alpha \\ Y = n \sin \alpha + y \cos \alpha \end{cases}$$

$$(n \ y \ z) \begin{pmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Cioè prendo l'uso delle \rightarrow (modo sbagliato) per $y = 1 : h$

for $y = 1 : h$

for $n = 1 : w$

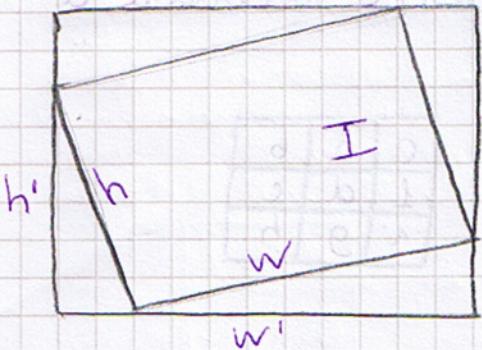
$$X = n \cos \alpha - y \sin \alpha$$

$$Y = n \sin \alpha + y \cos \alpha$$

X e Y saranno le nuove coordinate, ma sono dei valori che costituiscono approssimato durante le trasformazione. Andò a mettere in X e Y i valori di luminosità corrispondenti. I valori dei pixel corrispondenti di partente.

Problema: Per colpa proprio di questa approssimazione, ci saranno pixel ripetuti nell'immagine (slide 93) e quindi tra i punti (slide 94) \rightarrow pixel contati più volte. E' vero e proprio che vado ad un'immagine più grande ed uno più piccolo.

Allora scriviamo il codice.



I'

```
for Y=1:h
    for X=1:w
        i =
        j =

```

if $(x, y) \in I$ allora
prendiamo il pixel nelle
stesse coordinate (interpolato)
else

assegno zero (caso le
cornici)

In ogni caso esistono molti altri metodi per calcolare le rotazioni di un'immagine di un angolo α .

Alcuni di questi algoritmi sono scritti in un manuale "GEMS" che è una raccolta di metodi.

RIDIMENSIONAMENTO

(tecniche di zooming) o anche



Anche in questo caso avremo il solito compito di andare a riposizionare i pixel in modo da ottenere l'immagine di partenza rigonfiata e ripicciolata di quanto le vogliamo - o ammirata.

Dipende da quello che vogliamo, avremo dei "buchi" o delle ripetizioni di pixel.

In ogni caso dobbiamo essere sicuri di aver piazzato tutti i pixel.

Ci sono diversi metodi che riguardano il ridimensionamento. Il Nearest Neighborhood, metodi che usano le medie, e metodi di interpolazione come quelli bilineari e bicubici. Nella griglia di output infatti, ci saranno dei pixel non del tutto definiti dalle griglie originarie, ed i propri attributi attraverso questi metodi che sceglieremo come identificare il valore da attribuire ai pixel.

Vediamo quindi i principali metodi per la scelta dell'intensità luminosa.

INTERPOLAZIONE NEAREST NEIGHBORHOOD

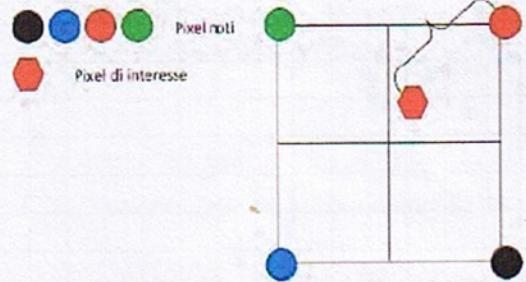
Determinare l'intensità luminosa delle componenti del pixel scegliendo i valori del pixel meno vicino al pixel in esame. Questo metodo prende una scansione di tutti i pixel dell'immagine da output attraverso l'iterazione nelle coordinate (x_i, y_i) del pixel di output e vuole calcolare le luminanze (nel caso di RGB si procede in base alle componenti per componente).

Disponendosi del fattore di scale S applicato all'immagine originale per aumentare ($S > 1$) o diminuire ($S < 1$) le dimensioni, è possibile dare le coordinate (x_i, y_i) , stabilire le coordinate corrispondenti nell'immagine da partenza (X_i, Y_i) dove $X_i = x_i \cdot \frac{1}{S}$

Ogni pixel interpolato dell'immagine di output è assegnato a $y_i = y_i \cdot \frac{1}{S}$ partire dal valore del pixel più vicino dell'immagine di output.

Quando pixel si calcola la coordinate intere (o intere) e si calcola il valore di tale pixel viene poi utilizzato come valore per il pixel nell'immagine che si sta creando.

Vantaggi: Facile da eseguire.
I pixel conservano valori di luminosità esistenti nell'immagine di partenza.



$$\begin{cases} X = S \cdot n \\ Y = S \cdot y \end{cases} \quad \begin{cases} n = X/S \\ y = Y/S \end{cases}$$

→ copiare il pixel più vicino è il metodo più veloce

Vediamo degli esempi grafici di ridimensionamento con questo metodo

d'esempio che passa da 256x256 a 128x128
poi da 128x128 a 64x64
quindi 16x16 è molto
poco definito



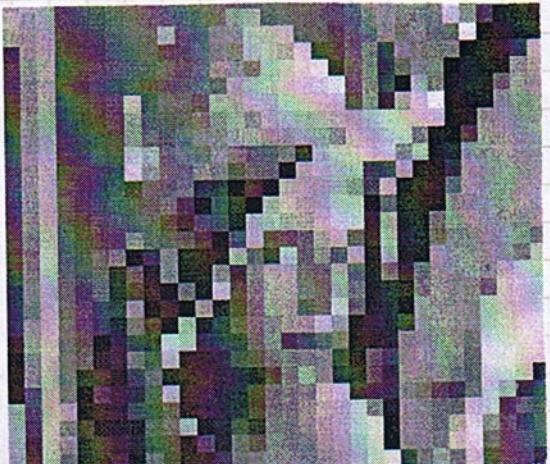
Da Immagine Originale
512x512

Nelle riduzioni trovano anche questi due esempi

Passando da 32x32 a 16x16
512x512 come si può vedere

vedere puoi molte informazioni

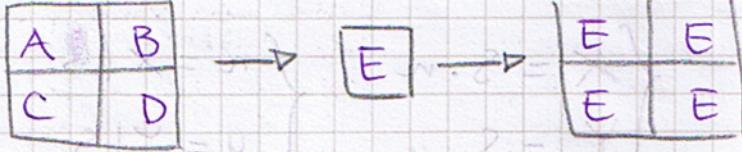
A 32x32



Uno studio ha dimostrato che 16 pixel sono sufficienti per riconoscere il viso di una persona nota.

Il N.N. è in totale un metodo veloce con risultati più precisi di altri metodi che vedremo.

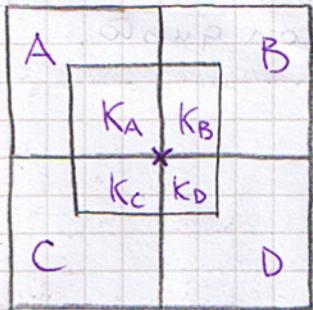
Nei nostri esempi intuitivamente si come se nei pixel ecco delle cose del genere



quindi è normale che perda molte informazioni.

"E" può avere un valore di intensità luminosa (di grigio) che sarà uguale ad A o a B, o a C o a D (insieme non nuovo).

● CON MEDIA PESATA



Potremmo considerare pixel a valori nulli anziché interi.

Vado a vedere il nuovo pixel e che pixel corrisponde dell'immagine di partenza e vedo che ne individua da 1 ad un massimo di 4, e sono formata da queste 4 sostanze (nel caso immagine figura).

(4 pixel adiacenti ovviamente)

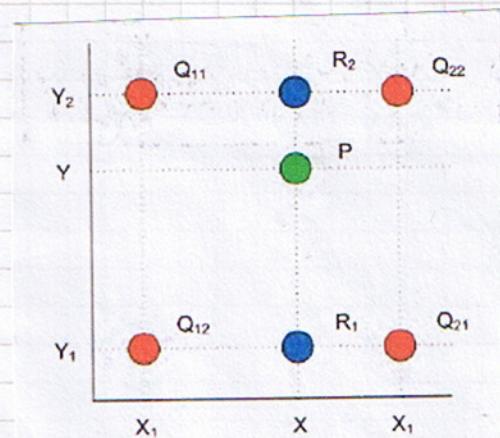
Il valore di luminosità del nuovo pixel viene attribuito attraverso la media pesata dei valori dei 4 pixel dell'immagine di partenza.

"Media pesata" nel senso che se K_A (anche di A facente parte del nuovo pixel) è maggiore di K_B (ad esempio) allora il valore del pixel A avrà più peso nella media da calcolare.

Con questo metodo avrò un'immagine di output meno sgranata ma più sfocata per via delle media.

Svantaggi: d'immagini multimedie avrò dei valori di grigio "nuovi"

● INTERPOLAZIONE BILINEARE



L'interpolazione bilineare è una estensione delle Nearest Neighbor, che interpola il valore di una funzione a una variabile come una media pesata rispetto alla distanza dei valori noti, adattata per operare su funzioni a due variabili e per analogia sulle immagini digitali.

Intuitivamente il metodo bilineare basa sull'applicazione del metodo lineare rispetto alle due direzioni.

Supponiamo di essere interessati al valore della funzione f nel punto $P = (x, y)$ assumendo di conoscere il valore di f

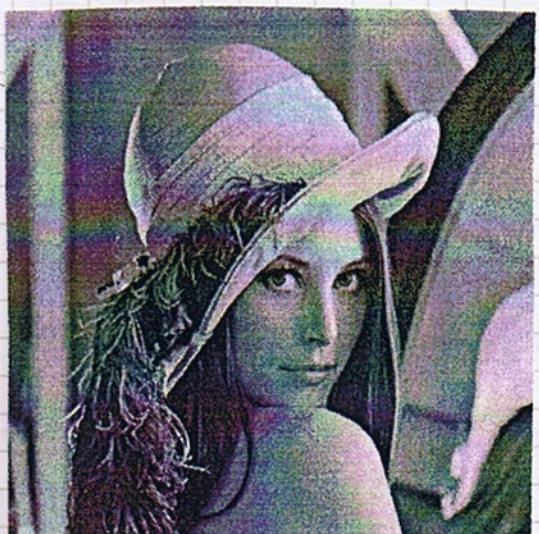
nei quattro punti $Q_{11} = (x_1, y_1)$ $Q_{12} = (x_1, y_2)$
 $Q_{21} = (x_2, y_1)$ $Q_{22} = (x_2, y_2)$

Si compie prima l'interpolazione nella direzione x , ottenendo R_1 ed R_2 . E permettendo di ricavare le formule per ottenere $f(n, y)$.

A 128×128
(con metodo
bilineare)



Torna a
 512×512
con N.N.



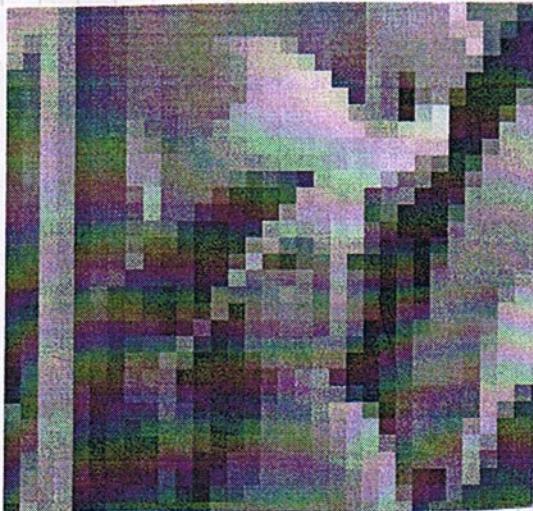
Da Immagine Originale
 512×512

A 64×64
(con metodo
bilineare)



Torna a
 512×512
con N.N.

A 32×32
(con metodo
bilineare)



Torna a
 512×512
con N.N.

Come possiamo notare, con questo metodo rispetto al metodo N.N. l'immagine di output migliora, ma quanto viene conservata più informazione. (più lento, ma dei risultati visibilmente migliori)

→ Questa interpolazione, è conosciuta anche come first-order, interpole bilineamente i pixel lungo ogni riga, poi interpole i pixel lungo ogni colonna -

Assegna ai pixel di destinazione ~~Da un valore che è una funzione bilineare dei 4 pixel vicini~~ alle seguenti ~~intensità~~ dell'immagine di input e fornisce un miglioramento di qualità rispetto alla replicazione, ma l'effetto di "smoothing" che produce più risultati più o meno "desiderabili"

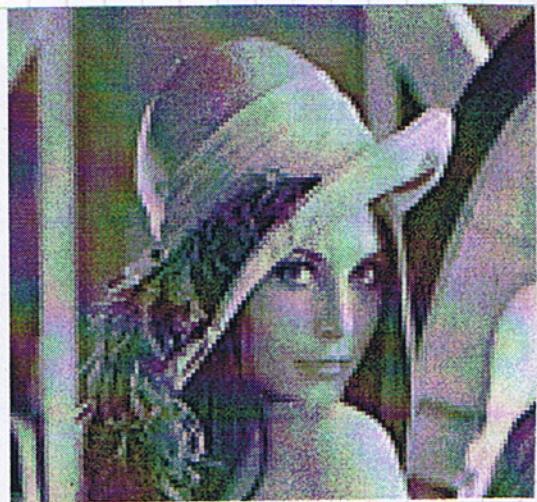
● INTERPOLAZIONE BICUBICA

d'interpolazione bicubica riduce gli artefatti introdotti dalle tecniche già esaminate e spese di un maggior calcolo computazionale.

Calcola il valore di un pixel dell'immagine di output facendo una media di 16 pixel che circondano il pixel corrispondente nell'immagine di input. A volte però i risultati del bicubico risultano addirittura peggiori del bilineare ma questo dipende molto dall'immagine di input.

In generale questo metodo

A 128×128
(con metodo bicubico)



Da: Immagine Originale
 512×512

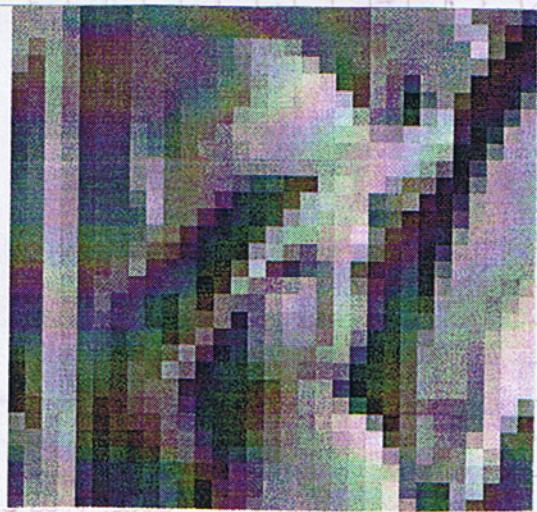
A 128×128
(con metodo bicubico)



In generale questo metodo risulta comunque essere più lento e solo "leggermente" migliore del bilineare. Anche se

d'immagine è meglio, è meno sfocata e fa meno mosaico, ma fa molti contatti più vicini (il gioco non vale le candele).

A 32×32
(con metodo bicubico)



QUANTIZZAZIONE

In generale, la quantizzazione è l'operazione con la quale si riduce il dettaglio di una informazione che non è all'interno del campo di informazione che esprime.

Riduciamo questo argomento attraverso un esempio.



"TrueColors" (24 bpp)

24 bpp saranno 8×3 cioè 8 bit per canale, ovvero avrò a disposizione 8 bit per il rosso, 8 per il verde e 8 per il blu.

"bpp" → bit per pixel. Ma se i colori non sono tutti gli stessi (R, G, B) allora abbiamo 24 bpp vuol dire 24 bit per pixel divisi appunto in 8×3 . Avrò 2²⁴ sfumature di colore nella tavolozza (che dobbiamo immaginare a colori).

Riduciamo il numero di bit per pixel a 16 (ancora TrueColors). Ma 16 non è multiplo di 3, quindi come dividere i bit per i 3 canali?

$$\begin{aligned} R &\rightarrow 5 \\ G &\rightarrow 6 \\ B &\rightarrow 5 \end{aligned}$$



In quanto ci ricordiamo sempre che il verde è meglio percepito dell'occhio umano. La tavolozza (immaginandole a colori) continua a sembrare uguale a quelle con 24 bpp se tieni conto che

Avrò 2¹⁶ sfumature di colore, ma l'immagine risulta identica a quelle con 24 bpp. Perché?

Perché lo schermo del nostro computer ha circa 800.000 pixel cioè 480.000 pixel, ma $480.000 < 2^{16}$ quindi in pratica 2¹⁶ sono sempre più sfumature di colore di quelle che possono visualizzare sul pc, e punto a fatto che anche l'occhio umano non percepisce le differenze.

TrueColors = Ogni pixel è identificato dalla sua posizione e dalle tre RGBC. quindi (i, j, R, G, B)

Dal continuo al discreto ho le quantizzazioni per quanto riguarda il colore.

Tavolozza:



Riduciamo ancora il numero di bit per pixel a 8 bpp che dividiamo in questo modo:

$$\begin{aligned} R &\rightarrow 3 \\ G &\rightarrow 3 \\ B &\rightarrow 2 \end{aligned}$$



La tavolozza ora (naturale bianco e nero...) contiene visibilmente i $2^8 = 256$ colori (sfumature di colore).

A questo punto il nostro occhio si accorge delle diminuzione di sfumature e l'immagine così digitata è detta midizzata.

"Index of Colors"

Notiamo infatti una tavolozza del tutto diversa.

Per identificare un colore nelle immagini infatti non andiamo più a mettere una terna RGB (come per i true colors) bensì faremo uso di una LookUpTable in cui andiamo a mettere l'indice corrispondente al colore nella tavolozza.

Index of Colors → Ogni pixel è identificato dalle sue posizioni e dall'indice che rappresenta il suo colore nella tavolozza (i, j, k)

(in questo caso perché $2^8 = 255$ con $0 < k < (255-1)$)
→ da $0 : (2^8 - 1)$

$k=0$ rappresenta il colore in alto a sinistra.

$k=1$ quello che gli sta accanto (sulla destra) e via così.

DOMANDA: Ma quali colori scegliere per la tavolozza?

Potremmo

- Pensare ad una tavolozza universale (...)
- Trovare la tavolozza più opportuna secondo diversi metodi e algoritmi

Siamo partiti da 16 milioni di colori e delle immagini True Colors che occupano $(w \cdot h \cdot 3)$ byte (dove $w \cdot h$ sono le dimensioni dell'immagine) e arrivati alla griglia di 3 puttoni ogni pixel ha le 3 componenti RGB) e arriviamo quindi alla tavolozza.

e siamo arrivati alle immagini midizzate (che occupano $(w \cdot h) \frac{N}{8} \text{ byte}$ + spazio occupato dalla tavolozza) quindi 113, visto che lo spazio occupato dalla tavolozza è trascurabile.

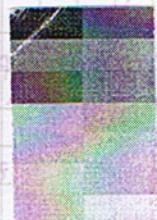
SPAZIO OCCUPATO DALLA TAVOLOZZA = Quello in questione.

Quindi passiamo da 8bpp a 3 byte per ogni pixel (8bpp) ad esempio con i suoi 256 colori occupa 3 byte per ogni suo colore (RGB) e quindi in totale occupa $256 \times 3 = 768$ byte

$768 \text{ byte} < 1 \text{ kb}$ (è poco meno di un kb quindi del tutto trascurabile)

Ma riduciamo ulteriormente la 4 bpp

o $R \rightarrow 1$ ossia la tavolozza ha differenze molto meno evidenti (non 16 colori ma solo 4) e $G \rightarrow 1$ e $B \rightarrow 1$ la tavolozza)



Quanti pixel occuperà $w \cdot h \cdot \frac{1}{2} + (16 \cdot 3)$

↓ dimensione tavolozza

mezzo byte = 1 Nibble

Scendiamo ancora di 2 bpp (4 colori nelle tavolozze) che
occupa $\left(\frac{w \cdot h}{4} + 12\right)$

E addirittura a 1 bpp (2 colori nelle tavolozze) che NON è binaria
attenzione, ma dicomatica -
(d'immagini ieri molto deteriorata)
(slide fino a 117)

MORFOLOGIA MATEMATICA

E' una branca dell'Image Analysis.

di cui si fonda la morfologia matematica è, rispetto
essenzialmente, l'analisi delle strutture geometriche di un'immagine
al fine di rendere evidenti le relazioni topologiche con i
elementi di confronto; tali connessioni dipendono, oltre che dalla
geometria delle strutture da evidenziare, anche dalla sua posizione
all'interno dell'immagine da esaminare.

Di recente inoltre la morfologia matematica ha acquisito dignità di
discipline autonome nell'ambito dell'elaborazione delle immagini.
Il suo insieme matematico si fonda principalmente sulla
teoria degli insiemi ed assume in se concetti di algebra, topologia
e geometria.

(I francesi ne sono gli scopritori nonché i maggiori utilizzatori)

Obiettivo: Mettere in evidenza determinate strutture nelle

2 (immagini)

che aiutano a

Ma come trovare una struttura?

all'obiettivo delle M.M. è infatti quello di estrarre informazioni
topologiche/geometriche da un'immagine binaria
S'ASCE, dove con E si indica l'insieme di tutti le possibili immagini
immagine di dimensione nota, attraverso l'utilizzo di un
seconda immagine B, più piccola, detta ELEMENTO STRUTTURANTE.

INPUT → Un'immagine (insieme) con delle strutture da evidenziare
un elemento strutturante (fonde) per evidenziare queste

strutture sono 2, 3, 4, 5, 7, 9, 13, ecc.
OUTPUT → Le strutture evidenziate (cioè sommate) non sono direttamente
osservabili e potrebbero essere ad esempio il calcolo di
loro area o periferia, comunque ci sono degli esempi nelle slide
(verde 119 a 123).

Per fare ad esempio potremmo calcolare un valore di soglia per
decidere di prendere o no un determinato elemento (ogni nello)

Se questa quantità è maggiore o minore di un certo valore di
soglia allora prendo e non prendo questa determinata struttura.

Per questi scopi potremmo usare algoritmi opposti o la
morfologia matematica.

- Alcune delle tecniche viste negli esempi delle slide sono:
- Filtraggio (mentre solo le strutture che hanno una determinata direzione)
 - Mappa Direzionale (colorare secondo la direzione)
 - Separazione di componenti connesse (contare ad es. delle cellule - numero anni di un albero -)
 - Segmentazione (individuare le singole componenti connesse - cellule - numero anni di un albero -)

Le operazioni elementari delle m.m. sono le



- DILATAZIONE (DILATATION)
- EROSIONE (EROSIONE)
- EROSIONE (EROSIONE)
- DILATAZIONE (DILATATION)

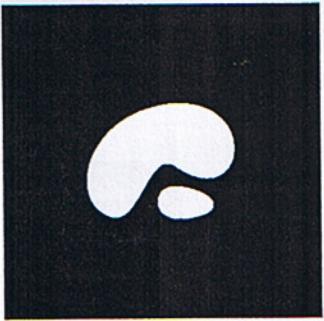
vengono indicate inoltre come operazioni elementari anche le trasformazioni di APERTURA e CHIUSURA ottenute dall'opportuna combinazione di trasformazioni di erosione e dilatazione.

Definiamo l'erosione e la dilatazione per un pixel $P = (i,j)$ dell'immagine I a scale di grigio in $[0,255]$ e per l'elemento strutturante S_r , ottenuto come disco discreto di raggio r .

EROSIONE

Gli operatori morfologici sono adatti maggiormente all'erosione e al ridimensionamento di immagini.

$$E_r(I(p)) = \min_{q \in S_r} I(p+q)$$



I



$E(I)$



Abbiamo l'immagine precedente chiamata I , che vediamo in figura inversa.

Abbiamo inoltre l'E.S. (Elemento Strutturante) S che in questo caso è un punto (anche se uniforme) abitato

come si può vedere più vicino alto. In parole povere dobbiamo immaginare che S "spazzi" tutta l'immagine I andando a prendere sempre il minimo (cioè, se S si trova ad identificare una zona tutta nera (quindi tutta di "zeri"), se ne prendiamo il minimo avremo ancora uno zero \rightarrow non cambierà nulla).

Stessa cosa avviene tutto bianco (tutto 255) nel minimo resterà 255 non andando a altro valore (ma non tutti i punti bianchi saranno spazzati) ma nelle zone di "confine", dove l'E.S. andrà a identificare pixel sia bianchi (anche uno solo basta) che neri (anche uno solo basta) varrà preso il minimo (e quindi il risultato sarà **IN SCURIMENTO** (nonché nimpicciamente) in questo caso tanto grande quanto le dimensioni dell'E.S. (meno 1 pixel)).

→ definizione troppo piccole scompaiono (più piccole dell'E.S.) (come si può vedere, meglio prendere sempre da come scegliere l'E.S. che puo' avere QUALSIASI FORMA) (tutti gli elementi troppo piccoli puo' contenere l'E.S. che li spazza via e cancella), solo se

→ è il centro dell'E.S. solo non permette mai in modo che puo' anche essere fuori dallo stesso. Bastano specificare mantenendo sempre uguale nella scrittura di I .

q spazzole s, p spazzole I.
Bisogna ricordarsi che s c'è a sua volta un'immagine binaria, per questo la circonferenza è solo l'esempio di E.S.

$q \in S_r$ (quando tutti i pixel che appartengono all'E.S.)

Poi per ogni pixel del disco considero i pixel dell'immagine e ne calcolo il minimo valore.

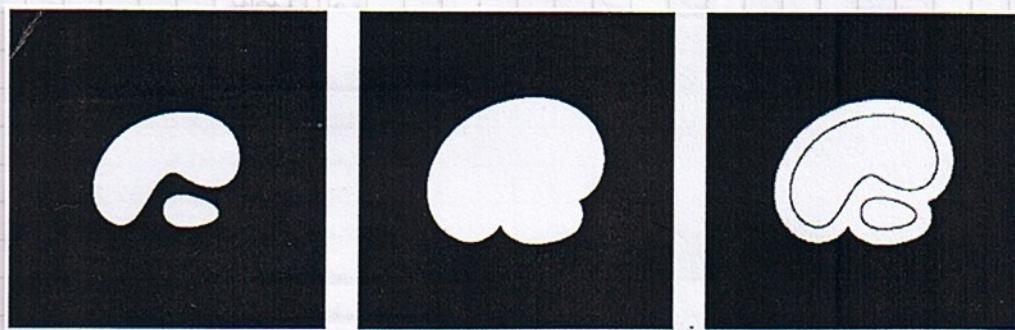
\rightarrow Valore da attribuire al pixel p nell'immagine

Le formule quindi non vuol dire che questo.

In questo modo abbiamo espanso le nostre formule anche per scale di raggio (è quindi estesa) mentre in rete si trovano formule che rappresentano l'erosione / dilatazione solo per immagini binarie.

\rightarrow Possiamo percepire la forma dell'E.S. dell'immagine risultante

DILATAZIONE



$$\delta_r[I(p)] = \max_{q \in S_r} I(p+q)$$

Il procedimento di spazzolamento è identico, ma la formula fa l'esatto opposto delle precedenti.

Vieni quindi adesso puoi il valore massimo che si trova all'interno dell'elemento strutturante.

Il risultato sarà uno SCHIARIMENTO (nonché ingrandimento in questo caso) tanto grande quanto le dimensioni dell'E.S.

\rightarrow se ritengo piccole probabilmente verranno "inglobate".

Di fatto lavorano quelle che custodiscono meno di un E.S.

Come mai può vedere dalle slide 127 se denotif l'effetto netto di erosione e dilatazione è rispettivamente scorrere e schiacciare.

E s / d vuol dire che si effettua l'una o l'altra cosa con un E.S. di raggio 5.

APERTURA E CHIUSURA

APERTURA γ :

(gamma)

(disegno manuale)

intervento

intervento

intervento

intervento

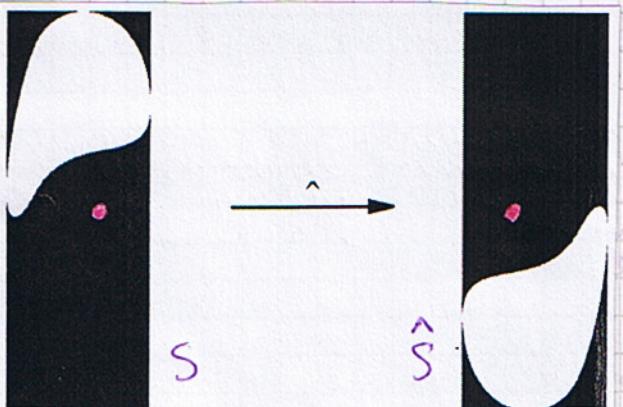
$$\gamma_S[I] = \delta_{\hat{S}}[\epsilon_S[I]]$$

Definiamo altri due operatori, anamorfologici, detti appunto apertura e chiusura, che servono a recuperare il più possibile l'immagine originale I_0 .

La normale applicazione di γ è l'eliminazione dei piccoli oggetti in I , preservando le forme e le dimensioni degli oggetti più grandi.

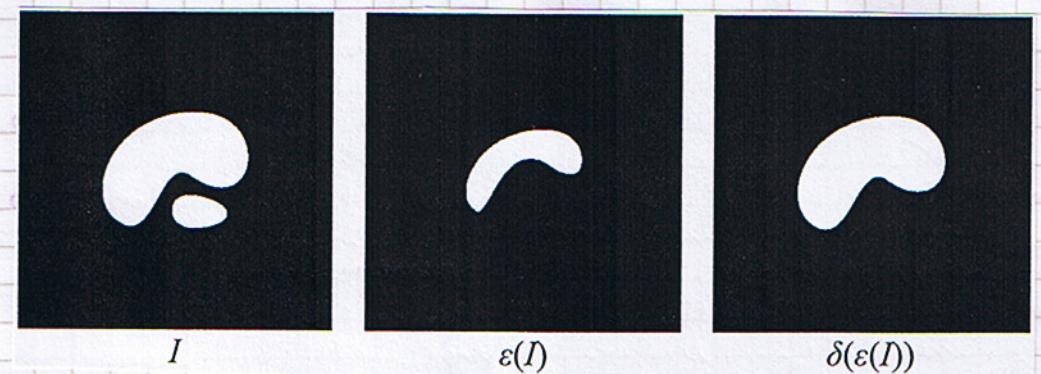
Viceversa, γ riempie i piccoli buchi nell'immagine.

$$(I)_3 = (I)_6$$

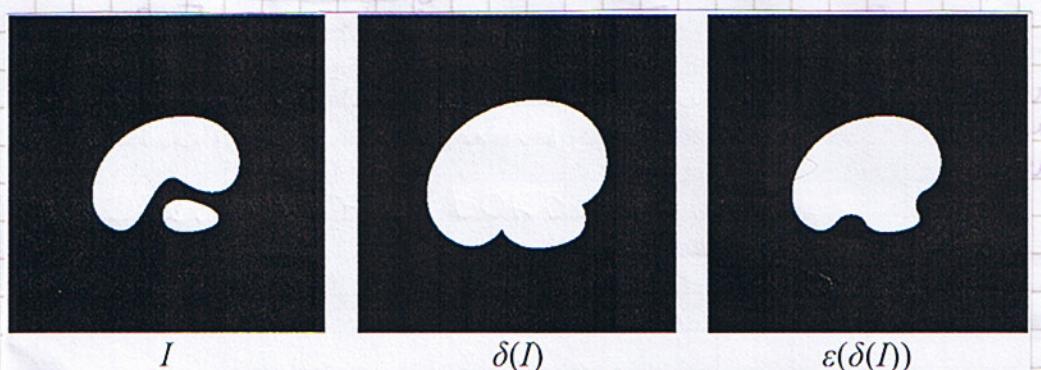


Si possono definire molte:

APERTURA: $\gamma(I) = \delta(\varepsilon(I))$



CHIUSURA: $\phi(I) = \varepsilon(\delta(I))$



Questi operatori morfologici vengono ad esempio usati nelle fabbriche per individuare guasti, nello campo di montaggio, nell'ambito medico, in circuiti stampati per individuare ponti fusi (dissaldature), per compiere misurazioni, in ambito topografico etc etc...

Apertura e Chiusura NON sono invertibili.

→ Erosione e Dilatazione SONO DUALI $\varepsilon(I) = \delta(\bar{I})$

(cioè l'erosione di un'immagine è equivalente alla dilatazione della complementazione della immagine)

de formula $\gamma_s(I)$ vuol dire che
che anche se si fa l'apertura e la
chiatura sulla stessa immagine
non rientra più nell'E.S. perché
l'apertura ha fatto sparire la
faccia davanti al cappello
e la chiusura ha fatto apparire la
faccia davanti al cappello.
Applichiamo consecutivamente
prima l'erosione, poi la dilata-
zione, come in
figura -

Il risultato è
quello di
mentre nere le
dimensioni delle
strutture più
grandi mantiene
cancelando del
tutto le strutture
troppo piccole pe-

Prima applichiamo la dilatazione
e poi l'erosione.

Il risultato è
di "unire" strut-
ture grandi e piccole
mentenendo le
dimensioni

LEZIONE 5

04/04/2007

Difiniamo delle operazioni sulle immagini attraverso la Morfologia Matematica

COMPLEMENTO: Nel complemento di un'immagine, se questa è binaria, ovviamente basta scambiare $0 \rightarrow 1$, $1 \rightarrow 0$.

Ma in Matlab rappresentiamo anche lo zero e l'uno con 1 byte e quindi avremo:

$$\begin{array}{l} 0 \rightarrow 0000000 \\ 1 \rightarrow 0000001 \end{array}$$

Procediamo con il complemento per scambiare i valori binari con:

$$\begin{array}{l} 0 \rightarrow 1111111 \\ 1 \rightarrow 1111110 \end{array}$$

e ciò non è più possibile, quindi sbagliato.

Per risolvere questo problema potremmo pensare i valori 0 e 1 dell'immagine binaria come 1 booleano, in modo da poter fare il contrario (avere solo 2 possibili valori).

Ma come ci comportiamo nel caso di scala di grigio?

Aviamo infatti valori compresi tra 0 e 255 e se facciamo ripetutamente il complemento come visto sopra non sempre otterremo valori che apparteranno ancora a quel range di valori per scala di grigio. Faremo quindi: $255 -$ (livello di grigio attuale)

Svantaggio: Dobbiamo effettuare una sottrazione per ogni pixel.

UNIONE: (di 2 elementi strutturanti)

slide 136.

nel quadretino indica la posizione "centrale" dell'E.S. (che non vuol dire letteralmente centrale, ma, come abbiamo visto, quella in cui abbiamo deciso di applicare i vari filtri).

Fanno quindi coincidere le posizioni "centrali" (immagini una sovrapposizione) e si effettua l'unione secondo le normali regole.

INTERSEZIONE: Stessa cosa ma con le regole per l'intersezione

(slide 137)

Inoltre, con due elementi strutturanti A e B e due immagini I e J valgono una granola quantità di proprietà che si trovano nelle slide 135 → 140.

Con le morfologia matematica possiamo anche vedere l'intensificazione di bordi (contorno) nelle slide 141 → 145

NUMERO DI CONNESSIONI
MINIMO RETTANGOLO DI RICOPRIMENTO
GRANULOMETRIA

} slide 146 → 154

- aggiustare double, bordi / angoli
- .. il fatto che sono + volte le stesse cose (nelle convolutioni)
- Metodo per non usare il prodotto puntuale minimo rettangolo di ricoprimento

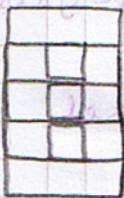
ESERCIZI

- Quinto esercizio effettua la convolutione su un'immagine senza l'utilizzo delle formule/commands predefinito CONV2. Applichiamo un filtro/kernel 7x7 di tipo blurred.

```
function imm=conv_senza_bordo_7x7
raul imread('RAUL_SCOGLIO.tif'); //cancilliamo l'immagine
figure; imshow(raul); title('The original RAUL');
thebar=waitbar(0,'Aspettando RAUL...');

kernel=ones(7); //creiamo il kernel da applicare
kernel(4,4)=0;
kernel=kernel/sum(kernel(:)); //stiamo usando il kernel blurred che conosciamo (sfocia)
len=length(raul)-length(kernel)+1; //calcoliamo quante volte il kernel può essere applicato
imm=zeros(length(raul)); //inizializziamo la matrice dell'immagine (sintesi distorsione) perché è imm. e quadrata
for j=1:len
    imm(i+((length(kernel)-1)/2),j+((length(kernel)-1)/2))=sum(aus2(:)); //troviamo il valore
end;
waitbar(i/len);
end;
imm=uint8(imm); //riconvertiamo
close(thebar);
figure; imshow(imm); title('Blurred RAUL');
```

In `imshow(I)` è raccomandabile mettere il range [0,255].
In questo esempio ci sono più delle imperfezioni.
Per prima cosa la matrice kernel sfocia l'immagine in questo modo:



Il secondo problema, è che non dobbiamo usare `sum()` per i limiti del possibile le funzioni predefinite di Matlab, e quindi dovremo evitare di usare il comando `*` per il prodotto punto a punto, e scrivere secondo il metodo di cui si discute a pag. 44 di questo quad.

In ultimo, non abbiamo bisogno di convettine e riconvertire continuamente in `uint8 -> double`, riflettendo in questo (meglio basta specificare il range).

● PROBLEMA DEL BORDO

(secondo metodo)

```
function imm=secondo_modo
raul2 imread('RAUL_SCOGLIO.tif');
raul=zeros(length(raul2)+(3*2)); //creiamo una matrice di dimensione aumentata del bordo rispetto al kernel
raul([4:(4+length(raul2)-1)], [4:(4+length(raul2)-1)])=raul2; //il bordo +1 (cioè length(kernel)-1)/2
raul=uint8(raul);
figure; imshow(raul); title('The original RAUL');
thebar=waitbar(0,'Aspettando RAUL...');

kernel=ones(7);
kernel(4,4)=0;
kernel=kernel/sum(kernel(:));
imm=zeros(length(raul2)); //inizializziamo la matrice di output
for i=1:(length(raul2)) //stavolta poniamo di considerare tutta l'immagine
    for j=1:(length(raul2))
        imm(i,j)=sum(aus2(:));
    end;
    waitbar(i/(length(raul2)));
end;
imm=uint8(imm);
close(thebar);
figure; imshow(imm); title('Blurred RAUL');
```

Il primo metodo, come ricordiamo, semplicemente non tiene conto del bordo, quindi è identico all'esercizio delle convolutioni, in cui non ci siamo posti il problema del bordo.

• (terzo metodo)

```
function imm=terzo_metodo
raul2=imread('RAUL_SCOGLIO.tif');
raul=zeros(length(raul2)+(3*2)); //come nel precedente
raul([4:(4+length(raul2)-1)], [4:(4+length(raul2)-1)])=raul2;
for i=1:3
    raul(i,[4:(4+length(raul2)-1)])=raul2(1,:); //parte alta
    raul([4:(4+length(raul2)-1)],[(4+length(raul2)-1)+i])=raul2(:,end); //parte destra
    raul([(4+length(raul2)-1)+i],[4:(4+length(raul2)-1)])=raul2(end,:); //parte bassa
    raul([4:(4+length(raul2)-1)],i)=raul2(:,1); //parte sinistra
end;
raul=uint8(raul);
figure; imshow(raul); title('The original RAUL');
thebar=waitbar(0,'Aspettando RAUL...');

for i=1:(length(raul2))
    for j=1:(length(raul2))
        .....
    end;
    waitbar(i/(length(raul2)));
end;
....
```

C'è da dire che in questo 3° metodo, e in quelli successivi a questo, abbiamo in certo senso trascurato i pixel che stanno ad angolo, ma questo è facile da corrreggere.
Si noti ad angolo infatti una leggera sfocatura nera.

Per il problema del bordo e i vari metodi vedi pag. 27

• (quarto metodo)

```
function imm=quarto_metodo
for i=1:3
    raul(i,[4:(4+length(raul2)-1)])=raul2((end-i+1),:); //parte alta
    raul([4:(4+length(raul2)-1)],[(4+length(raul2)-1)+i])=raul2(:,i); //parte destra
    raul([(4+length(raul2)-1)+i],[4:(4+length(raul2)-1)])=raul2(i,:); //parte bassa
    raul([4:(4+length(raul2)-1)],i)=raul2(:,(end-i+1)); //parte sinistra
end;
....
```

Bon i puntini nello 0
che le parti in
questioni sono
uguali.
all'esercizio
precedente

• (quinto metodo)

```
function imm=quinto_metodo
for i=1:3
    raul(i,[4:(4+length(raul2)-1)])=raul2((4-i),:);
    raul([4:(4+length(raul2)-1)],[(4+length(raul2)-1)+i])=raul2(:,(end-i+1));
    raul([(4+length(raul2)-1)+i],[4:(4+length(raul2)-1)])=raul2((end-i+1),:);
    raul([4:(4+length(raul2)-1)],i)=raul2(:,(4-i));
end;
....
```

//parte alta, destra,
bassa e sinistra.

Questo è l'ultimo
metodo. Resta il
problema degli
angoli.

• Questo esercizio ha un filtro sharpen e lo applica senza fare uso delle funzioni predefinita special(sharpen) ma con le procedure di pag. 35-36

function imm=sharpen

```
for i=1:len
    for j=1:len
        .....
        imm(i+((length(kernel)-1)/2),j+((length(kernel)-1)/2))=sum(aus2(:));
    end;
    waitbar(i/len);
end;
imm=uint8(imm); //niconvertiamo
close(thebar);
imm=double(imm);
imm2=raul-imm; //imm2 è h(I) ovvero la differenza che mantiene le strutture più piccole
imm2=uint8(imm2);
k=2; //lo stiamo moltiplicando noi
imm2=double(imm2);
sharpened=raul+k*(imm2); //otteniamo l'immagine contrastata secondo il filtro di sharpen
sharpened=uint8(sharpened);
figure; imshow(sharpened);
```

MORFOLOGIA (continuando)

Date una coppia di elementi strutturanti A e B e una coppia di immagini I e J, valgono le seguenti proprietà:

- $\Sigma_A(\Sigma_B(I)) = \Sigma_{\partial_A^\wedge B}(I)$

Fa fare 2 volte l'erosione su un'immagine I con 2 elementi strutturanti diversi così:

$$2 \times \epsilon 100^2 \text{ (su un'immagine } 100 \times 100 \times 10^2 \text{ cioè con un E.S. } 10 \times 10)$$

Potrei visualizzarla facendo

L'azione con B su un'immagine è stata fatta una dilatazione con

l'elemento strutturante A ribaltato cioè \bar{A} .

Osserviamo che è più veloce poiché $\epsilon \cdot 10^2 \times 10^2 + \epsilon \cdot 1000^2 \cdot 10^2$

- $\partial_A(\partial_B(I)) = \partial_{\partial_A^\wedge B}(I)$

la stessa cosa per la dilatazione

- $\Sigma_A(I) = \{P : A_P \subseteq I\}$

Un altro modo di vedere l'erosione è come l'insieme dei punti che appartengono all'immagine I cioè



- $\partial_A(I) = \{P : A_P \cap I \neq \emptyset\} = \{P : A_P \cap I \neq \emptyset\}$

E' l'insieme dei punti x che A colpisce l'immagine I quando le sue origini coincide con x.

- $\gamma_A(I) = \bigcup_P \{A_P : A_P \subseteq I\}$

(bisognerebbe scrivere)

- $\varphi_A(I) = \bigcap_P \{\bar{A}_P : \bar{A}_P \subseteq \bar{I}\}$

L'UNIONE TRA I DUE E.S.

Poiché non sempre hanno le stesse dimensioni e poiché non hanno sempre lo stesso pixel centrale, il loro confronto principale è nel fatto che coincidono infatti indichiamo con t la translation dell'elemento strutturante in modo da far coincidere le posizioni centrali.
Ad esempio,

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \cup \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$$A \cap B = A' \cap B = A \cap B'$$

INTERSEZIONE TRA E.S.

Esempio:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \cap \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Come prima ma con l'intersezione.

- $\left\{ \begin{array}{l} \mathcal{E}_{A \cup B}(I) = \mathcal{E}_A(I) \cup \mathcal{E}_B(I) \\ \mathcal{D}_{A \cup B}(I) = \mathcal{D}_A(I) \cup \mathcal{D}_B(I) \end{array} \right.$ Fare l'erosione o le dilatazioni con un elemento strutt. che è l'unione di 2 E.S. è uguale a fare l'erosione o dilatazioni con l'elemento strutturante A unione erosione/dilataz. con l'E.S. B.

- $\left\{ \begin{array}{l} \mathcal{E}_{A \cap B}(I) \geq \mathcal{E}_A(I) \cap \mathcal{E}_B(I) \\ \mathcal{D}_{A \cap B}(I) \geq \mathcal{D}_A(I) \cap \mathcal{D}_B(I) \end{array} \right.$ La stessa cosa non vale per l'intersezione.

- $\left\{ \begin{array}{l} \mathcal{D}_A(I \cup J) = \mathcal{D}_A(I) \cup \mathcal{D}_A(J) \\ \mathcal{E}_A(I \cap J) = \mathcal{E}_A(I) \cap \mathcal{E}_A(J) \end{array} \right.$ Applicare le dilatazioni con l'el.st. A sull'unione di 2 immagini è uguale a fare l'unione delle 2 dilatazioni applicate alle 2 immagini. Analogamente per l'intersezione.

- $\left\{ \begin{array}{l} \mathcal{E}_A(I \cup J) \geq \mathcal{E}_A(I) \cup \mathcal{E}_A(J) \\ \mathcal{D}_A(I \cap J) \leq \mathcal{D}_A(I) \cap \mathcal{D}_A(J) \end{array} \right.$ In questo caso invece non vale.

- $I \leq J \Rightarrow \left\{ \begin{array}{l} \mathcal{E}(I) \leq \mathcal{E}(J) \\ \mathcal{D}(I) \leq \mathcal{D}(J) \end{array} \right.$ se l'immagine I è \leq a J allora anche l'erosione e le dilatazioni su I saranno \leq a quelle su J

con $\mathcal{E}(I) \leq \mathcal{E}(J)$ $I \leq J \leq \varphi(I) \leq \varphi(J)$

con $\mathcal{D}(I) \leq \mathcal{D}(J)$ $I \leq J \leq \varphi(I) \leq \varphi(J)$

$$\gamma(I) \leq \gamma(\varphi(\gamma(I))) \leq \varphi(\gamma(I)) \leq \varphi(\gamma(\varphi(I))) \leq \varphi(\varphi(I))$$

Le operazioni base sono l'unione \cup e l'intersezione \cap - Per le immagini a scale di grigio l'unione diventa THE POINT-WISE OPERATOR \vee e l'intersezione THE POINT-WISE OPERATOR \wedge :

- UNION: $(f \vee g)(x) = \max [f(x), g(x)]$
- INTERSECTION: $(f \wedge g)(x) = \min [f(x), g(x)]$

ESTRATTORI DI CONTORNO

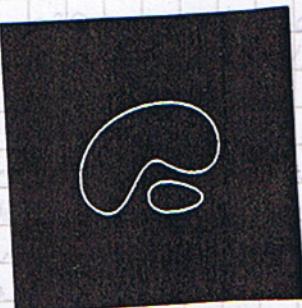
Con gli operatori morfologici possiamo definire l'estrazione di contorno nel seguente modo:

$$\text{p}(I) = \partial_1(I) - \varepsilon_1(I)$$

// elemento strutt. di reggis. 1
(proprio per il contorno)

In prendendo un po' l'area dell'immagine con la dilatazione, poi lo riduco ed effettua la differenza, in questo modo metto in evidenza le parti che non sono in comune.
Osserviamo che prima facciamo ∂ che prende i valori massimi e poi ε che punta quelli minimi.
Osserviamo che se avessi fatto al contrario avrei sempre avuto contorni ma con valori negativi.

Graphicamente:



Dalle suole possiamo vedere l'effetto in linea è molto simile al confronto con il filtro di convoluzione che effettua le stesse cose cioè Sobel, anche se Sobel ha una maggiore completezza.

(ρ da un risultato migliore)

TOP HAT E BOTTOM HAT

Osserviamo che gli operatori γ e φ possono essere combinati denotando ad altri operatori quelli:

(in realtà anche γ e φ potremmo ricevere zero venirebbe più lento)

$$- th(I) = I - \gamma(I)$$

In entrambi i casi $I > \gamma(I)$ e $\varphi(I) > I$ quindi non occorre prendere valori assoluti

$$- bh(I) = \varphi(I) - I$$

Miglioriamo il contrasto di I tramite le formule:

$$k(I) = \max \left[0, \min \left\{ 255, I + th(I) - bh(I) \right\} \right] = \max \left[0, \min \left\{ 255, 3 \times I - \gamma(I) - \varphi(I) \right\} \right]$$

(per il contrasto)

$L(k) \leq (k)$ invece per assicurarsi che non scendiamo sotto lo zero

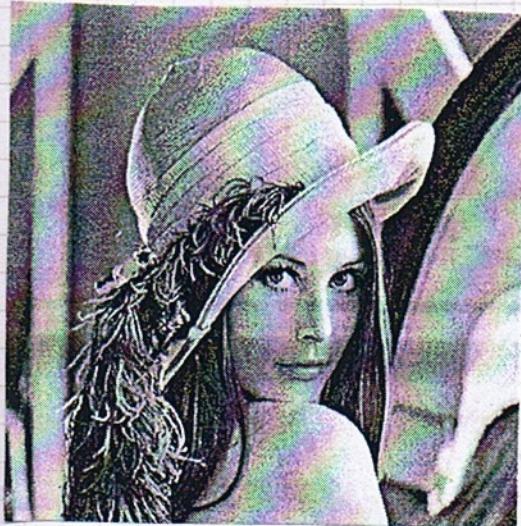
$$[L(x) \leq (x)] \text{ vero} \Rightarrow L(x) = (x) \wedge 1$$

Vediamo alcune applicazioni:

Dato è un'immagine:



Applichiamo le formule usando l'elemento strutturante con
raggio 1 e 5. Nel caso del sharpen varia il contrasto tramite K qui
tramite raggiò K



$K_1(I)$



$K_5(I)$

NUMERO DI CONNETTIVITÀ

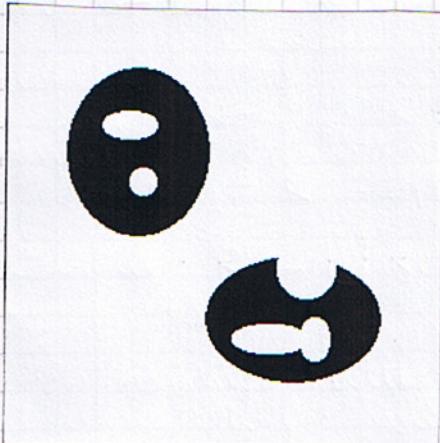
Il numero di "connettività" NC di un'immagine binaria restituisce una prima informazione sulla struttura delle sue componenti. Si definisce NC come la differenza tra il numero di componenti connesse e il numero di buchi. Cioè

$$NC = \#C - \#B \rightarrow \text{buchi}$$

\downarrow
componenti connesse

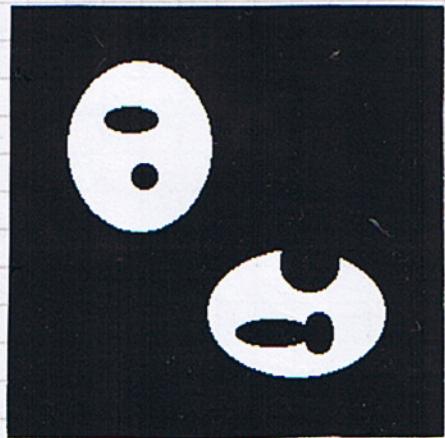
Questo numero può servire per il controllo della qualità industriale (es. nullo trasportatore)

Ad esempio:



In questo caso ho 2 componenti connesse e tre buchi.

In realtà però i buchi li posso vedere come le componenti connesse della negazione dell'immagine cioè:



Si può calcolare NC tramite la formula di Euler per i grafi planari:

$$\#C = \#V + \#F - \#A$$

V = vertici.

A = Archi.

F = Face

In particolare nelle immagini binarie possiamo considerare elementari le facce delimitate da archi diretti e non elementari tutte le altre facce. Quindi:

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

Facciamo l'ultimo passo indietro e definiamo

GRAFO PLANARE = si ammette una rappresentazione nel piano tale che nessuna coppia di spigoli si intersechi eccetto nei vertici comuni

Ese.

Due rappresentazioni piane di un grafo planare



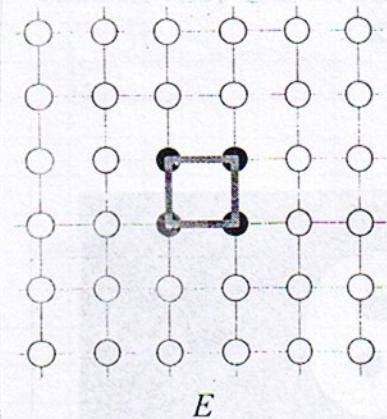
Rappresentazione non planare di un grafo planare



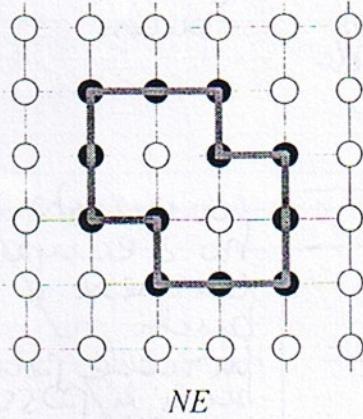
Grafo non planare



Per ~~ogni~~ faccia intendiamo le componenti connesse di un insieme -
ogni faccia è contornata da uno o più archi di spigoli di E e vertici V -
quindi le facce elementari sono i gruppi di 4 pixel strettamente connessi a 2 a 2, mentre le facce non elementari sono i buchi delle componenti connesse



E



NE

Quindi:

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

Possiamo esprimere questa relazione contando il numero di volte in cui particolari elementi strutturanti compaiono nell'immagine:

$$\#V = \# \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \#E = \# \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}; \#A = \# \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \# \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Possiamo però ora normalizzare gli elementi strutturanti in modo che abbiano tutti la stessa dimensione e posizione dell'elemento centrale -

Quindi:

$$\#V = \# \left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \right)$$

$$\#E = \# \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \# = \text{n° di volte che E.S. compare}$$

$$\#A = \# \left(\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right) + \# \left(\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right)$$

Semplificando il numero di connettività $NC = \#V + \#E - \#A$
lo possiamo ottenere nel seguente modo

$$\#NC = \# \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \# \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \# \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \rightarrow \text{oltre 9000 volte}$$

trovato questo pattern

Quindi applichiamo alle immagini binarie 3 volte un operatore morfologico.

MINIMO RETTANGOLO DI RICOPRIMENTO

E' il più piccolo rettangolo che contiene gli oggetti (esiste una cosa analoga che si chiama CONVEX HULL che è il più piccolo poligono che contiene il minimegno).

Si ha:

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

Input: I // prendiamo un input una minimegna e I

$R_{i,0} \leftarrow I$ // considerare il
num. di righe.

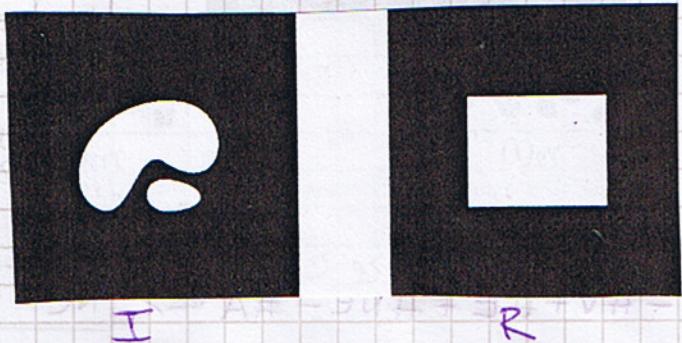
repeat

$R_{i,j+1} \leftarrow E_{S_i}(R_{i,j}) \cup R_{i,j}$

until

$$R_{i,j} = R_{i,j+1}$$

Output: $R \leftarrow \bigcup R_{i,j}$



Applichiamo 4 volte l'azione con i vari E.S. Su all'minimegna
il risultato è l'unione cioè vado allargando l'minimegna
finché non ottengo il risultato -
Fare un Matlab come esercizio.

$$A\# - B\# + V\# = DU$$

$$(A\# - B\#) + (B\# - C\#) + (C\# - D\#) = V\#$$

$$A\# - B\# + B\# - C\# + C\# - D\# = V\#$$

$$A\# - D\# = V\#$$

$$A\# - (A\# - V\#) = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

$$V\# = V\#$$

$$A\# - A\# + V\# = V\#$$

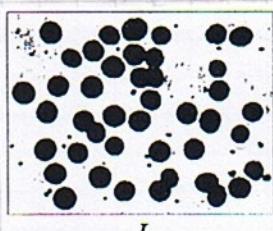
$$V\# = V\#$$

GRANULOMETRIA

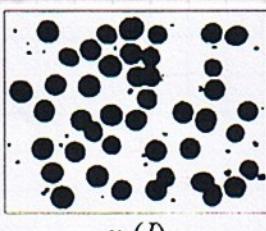
3.3.1

Si può utilizzare ad esempio per vedere quanti globuli rossi ci sono o che forma abbiano.

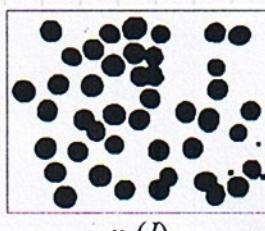
Ad esempio:



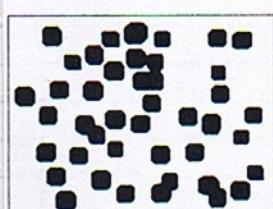
I



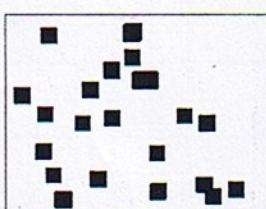
$\gamma_1(I)$



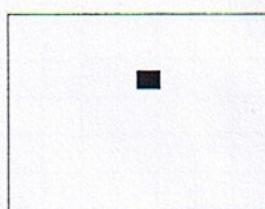
$\gamma_3(I)$



$\gamma_9(I)$



$\gamma_{13}(I)$

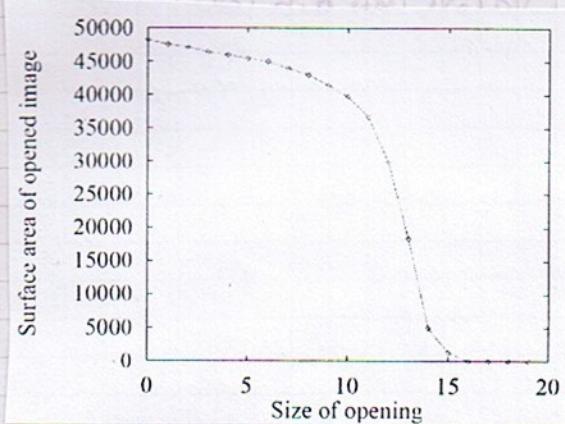


$\gamma_{15}(I)$

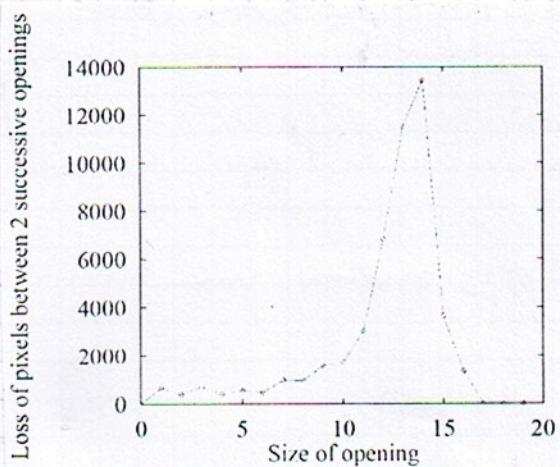
Applichiamo aperture basate all'immagine I in cui l'E.S. è via via più grande.

Ci fermiamo finché non abbiamo più nessun elemento. Poiché vediamo che vanno diminuendo.

Possiamo vederci con i seguenti grafici



lungo la x poniamo il raggio (dimensione di apertura) e lungo la y rappresentiamo il che come vediamo appena arriviamo al raggio 15 si è annullata naturalmente la dimensione.



Adesso lungo la y poniamo il numero di pixel persi tra due successive aperture.

Uso: patologie, cellule etc.

Le aperture sono delle sorta di filtri che fanno uscire solo certi pixel.

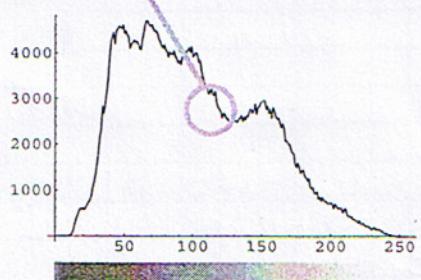
Per esempio se ho un pixel bianco e lo filtri con un apertura di 3x3, uscirà solo il pixel centrale.

(ogni pixel solo) osservere al reg imponeva in un modo giusto o errato).

LEZIONE 6

ISTOGRAMMA (luminanza)

18/04/07



g	#
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	14
11	46
12	112
...	...
114	3066
115	3160
116	2913
117	2806
...	...
242	41
243	18
244	8
245	9
246	2
247	4
248	0
249	0
250	0
251	0
252	0
253	0
254	0
255	0

Data un'immagine in scale di grigio, con i pixel i cui valori vanno da 0 a 255, ne costruiamo il corrispondente

ISTOGRAMMA

A destra dell'immagine possiamo vedere la tabella a partire della quale costruiamo l'istogramma. Per ogni valore di grigio scriviamo al lato il numero di pixel nell'immagine con quel valore di grigio.

In questo caso abbiamo evidenziato i pixel con valore medio, che sono quelli più numerosi nell'immagine.

dell'istogramma vero e proprio è quello sotto l'immagine, il grafico per l'appunto.

In ascissa troviamo g e in ordinata # (numero di pixel con quel valore g)

Gli histogrammi possono servire ad esempio per le barre dati di un'immagine, se di questi infatti ci ricaviamo gli histogrammi, potremo ancora di confrontare gli histogrammi ad esempio nelle ricerche.

In questo modo si potrebbe, nei limiti del possibile, ridurre il confronto tra immagini al confronto tra histogrammi o perché no, tabelle di ampiezza

La LOOKUP TABLE in questione è infatti molto semplice da costruire, basta fare scansione dell'immagine e incrementare i vari valori ogni volta che li riconosciamo.

Matlab ha un comando per plotare l'istogramma che presenta i problemi fondamentali che vedremo + avanti.

Ora tocca il caso delle scale di grigio.

Come si fa l'istogramma di un'immagine a colori?

- Si fanno separatamente i 4 plot:

- Rosso
- Verde
- Blu
- Luminanza

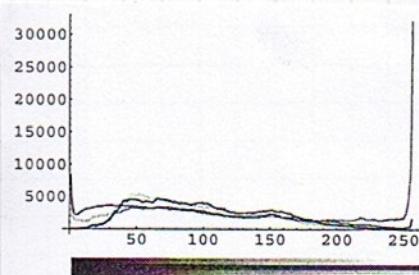
(68) Come avviene l'immagine per la luminanza (scale di grigio) da un'immagine a colori è molto semplice - Converti in scale

di grigio con HSV come avremmo visto nel precedente.
Avrò quindi, oltre all'istogramma già visto, altri tre istogrammi

→ Rosso, molto carico per questa immagine; le barre va' dal nero al rosso puro
→ Verde, non moltissimo; le barre arrivano al verde puro, come sopra
→ Blu, più uniforme (fino al blu puro)

(di slide in questione sono 158-159-160)
le percentuali di rosso è molto alta.
Per la luminanza le barre iniziano con il bianco.

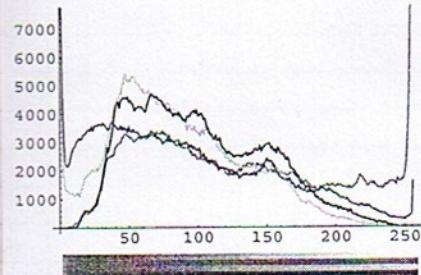
Sarà perciò dunque i 4 istogrammi e avremo (per queste figure)



(immaginando i 3 colori...)

→ notiamo subito questo picco

na mi compenso non riusciamo a vedere bene l'istogramma che non ha il picco. Allora:



→ NORMALIZZIAMO: Modifichiamo quindi il plot dell'istogramma "tagliandolo" dove sive e ingrandendo lo sui punti fondamentali per determinare i 4 adiacenti e viderne bene l'andamento.

Adesso però abbiamo perso che il picco arriva ad ordinata 3000 circa - possiamo risolvere il problema scrivendolo esplicitamente -
Il problema del quale accennavamo prima di Matlab sugli histogrammi (c'anche un problema anche di photoshop) è che taglia gli OUTLYER ma non dice niente sul livello/altezza del picco che è stato tagliato -

per immaginare un confronto tra i histogrammi, potremmo immaginare 2 foto che ritraggono gli stessi frammenti in posizione semplicemente diversa (slide 163).
Vediamo che i loro istogrammi (entrambi normalizzati al massimo valore comune) sono purtroppo uguali - Questo però allo stesso tempo ci fa notare che gli histogrammi non ci danno informazioni sulla forma dell'immagine (infatti l'esempio delle slide rappresenta le stessa cosa, i frammenti, in posizione diversi).

Dai valori di media e deviazione standard possiamo mettere in paragone a vedere le somiglianze tra gli istogrammi.

Possiamo quindi avere noi una procedura per calcolare l'istogramma di un'immagine, che mette visualmente pure le barre sotto l'istogramma, come abbiamo finora visto nei nostri esempi.

Personalizziamo quindi la funzione di Matlab per la visualizzazione dell'istogramma.

```
function h=plothist(img);
% esempio: plothist(img);
h=imhist(uint8(img),256)';
m=101-ceil(100*double(h)/double(max(h(:)))); % m=altezza istogramma
out=uint8(zeros(120,256)+255);
for i=1:256
    out(m(i):100,i)=0;
    out(102:120,i)=i-1;
end;
figure; imshow(out);
```

Questo codice non elimina gli outliers.

No digitare questo codice:

- non visualizzando gli outliers
- evitando di normalizzare l'istogramma sugli ~~non zero~~ outliers

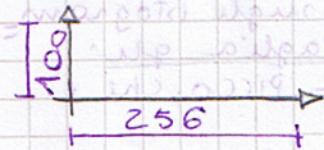
→ **imhist**

// scrive i valori dei vari pixel.

Visualizza l'istogramma di un'immagine data.
Il primo parametro è appunto l'immagine, il
secondo, cioè 256 sono i binari di intensità
che devono essere calcolati nell'istogramma (le
intensità si lasciate).

In questo modo otteniamo sempre istogrammi con uguale altezza
h, proprio per questo motivo al riguardo successivo facciamo una
propostione.

100 è un valore arbitrario che alterza delle ordinate che
abbiamo impostato noi per il grafico dell'istogramma, che sono:



(Valore Massimo sarà 100 ma siamo 5) (tipicamente decodifica che usavamo per RAYTIC
gli algoritmi genetici)

Si impostano il valore massimo a 1 tutti

eseguiamo su tutti gli altri veranno compresi tra 0 e 100

"Out" è proprio l'immagine (istogramma) e nel forzando a non
mettere proprio il valore dell'immagine nell'istogramma (che è
a sua volta un'immagine) Ogni pixel oltre ai 100 dei primi ci sono anche 200 pixel
per la barra.

→ barra

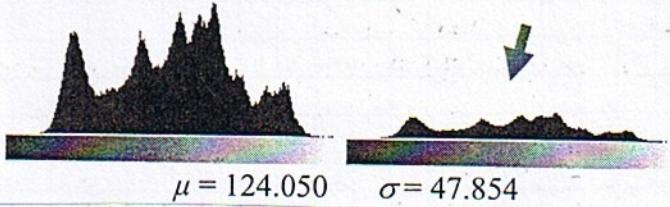
in 256 avremo 30 barre (impossibile è un libro; i valori si
riportano solo da un esempio il libro è molto

Ma vediamo altri esempi di immagini con relativi istogrammi:
Timiamo conto dalla slide 165 che l'istogramma di destra di
ogni immagine è normalizzato a quello relativo all'esempio di
"salt & pepper" (picchiatrice 0 e 255) Istruttiva è l'istogramma
possiamo anche risavere il valore massimo in base al quale potranno
normalizzare l'immagine (basta impostare 1 altre proporzione
rispetto a quella che abbiamo scritto con il valore 100)



In questo esempio viene evidenziato che riflettendo e scomponendo l'immagine l'istogramma relativo non cambia affatto.

Lena



Gli histogrammi delle due immagini sono identici - μ e σ sono gli stessi.

C'è da dire però che per confrontare 2 immagini gli histogrammi devono essere confrontabili (se vogliamo confrontarle con questi metodi) devono quindi avere la stessa scala.

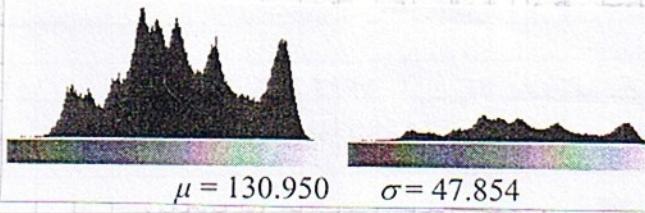
Il primo dei due histogrammi è quindi quello assoluto, il secondo è un confronto; l'istogramma relativo ad un altro per il confronto (normalizzato salt & pepper) (???)

vediamo un po' di esempi di histogrammi:

• NEGAZIONE DELL'IMMAGINE

(è proprio una not dell'immagine) abbiamo visto che vuol dire una not in scale di grigio (è il complemento di 256).

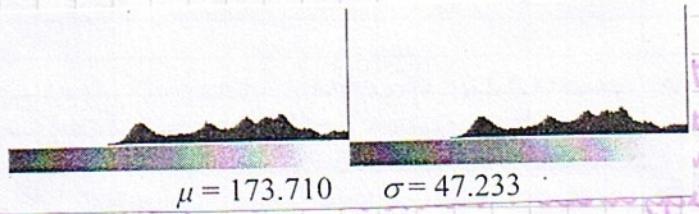
Gli histogrammi dell'immagine complementata sono (gratissimi) identici a quelli originali ma come se fossero verticale allo specchio. Inoltre σ è identica.



• AUMENTO DELLA LUMINOSITÀ

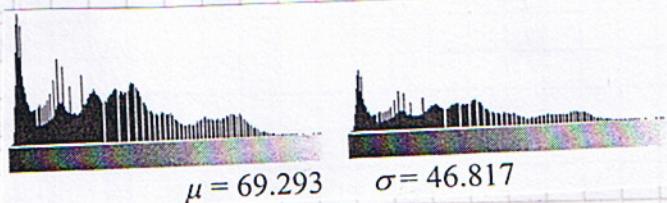
(di 50 in questo esempio) per aumentare la luminosità significhi scrivere di 50 vuol dire aumentare di 50 il livello di grigio di ogni pixel, trattandosi avviamente da 255 (che doveva non potevamo più aumentare).

Quanto genera dei picchi. Inoltre possiamo notare che l'istogramma sembra spostato (lo è) di 50 verso destra (tipico saltato con picco finale).



● QUADRATO DELLI IMMAGINI

se le linee verticali bianche rappresentano proprio la totale concentrazione di pixel di quel colore (livello di grigio)



- RADICE QUADRATA → si snaccia. Troviamo linee alte e basse. (Per via dei picchi) - slide 167 -

- USO DEL FILTRO MEDIA → slide 168.

Notiamo che gli istogrammi relativi alle immagini in cui abbiamo applicato il filtro media sono molto simili a quelli delle immagini originali, ma le fondamentali differenze fra i picchi qui appaiono smussati, appiattiti. Le grafiche tendono ad uniformarsi.

- DENOISING → Alcuni pixel sopravvivono.

Avranno più pixel di bianchi e non più e quindi più picchi.

- RIDIMENTONAMENTI → (slide 170-171)

Gli istogrammi ottenuti ridimensionando l'immagine da 512 a 128 e poi di nuovo a 512 con nearest neighbor o con bilineal / nearest sono molto simili (tra loro). Con bicubico e bicubic lo sono ancora di più.

- EROSIONE → la media scende molto (slide 171)

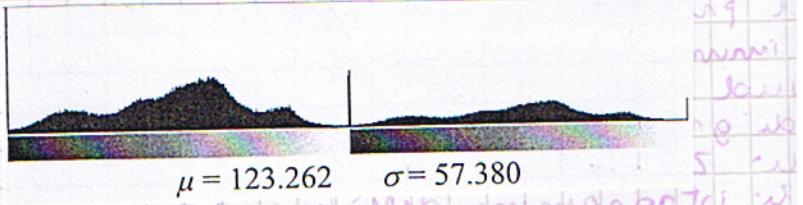
- DILATAZIONE → la media sale molto (slide 172)

- APERTURA / CHIUSURA → qualche picco in più

● AUMENTO DEL CONTRASTO

Sembra tutto più uniforme. d'immagini di riferimento contrastata con K5

(slide 173), dove le medie sono molto simili (anche se le variazioni di luminosità), mentre la deviazione standard è passata da 47 a 57.



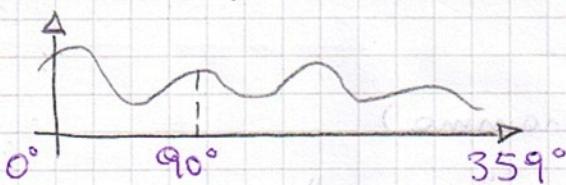
In genere quindi prima ancora di confrontare gli istogrammi di 2 immagini potremmo confrontare le loro medie per avere informazioni, e dovremo fare un'interpretazione qualcosa anche solo leggendo i numeri di questi.

L'ATTIVITÀ ALLE ORE NUOVE

(esempio di un potrebbe essere DataBank pittorica per cercare immagini simili)

ammiraglia) uno molti ammiraglie in mare. (ad esempio 5000 (altri 1000) articolati con 0.2: 10 (100) chilometri orari)

Sempre seguendo alle bandette di immagini, potremmo tirare fuori il cosiddetto **EDGE HISTOGRAM**, che è in pratica un histogramma in cui vengono evidenziati gli andamenti degli angoli di una immagine. Pendo quindi l'immagine e ne tiro fuori i contorni (la magnitudo del gradiente e l'angolo dei vari contorni) in un grafico di questo tipo:



(Per le ricerche in database pittorici)



Un plot come questo dell'esempio vuole ad esempio che abbiano un picco a 90°, cioè un picco a 90°, in pratica una linea verticale nell'immagine.

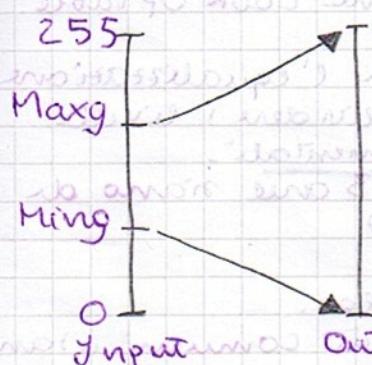
Un plot con tanti picchi a 90° e a 180° corrisponde ad una immagine con molti contorni verticali (tipo una immagine di vari grattacieli e palazzi).

STRETCHING (istogramma)

Lo stretching è la tecnica che (dal nome stesso) dilata l'istogramma ma in modo da coprire l'intera gamma dei grigi, anche se introduce dei "buchi" nel nuovo istogramma.

```
function out=mystretch(img)
% esempio: imshow(mystretch(lena));
ming=double(min(img(:)));
maxg=double(max(img(:)));
out=uint8(round(255*(double(img)-ming)/(maxg-ming)));
```

(Un esercizio su questo codice di stretching potrebbe essere quello di eliminare gli outlier (solo quelli di saturazione a 0 e 255) per non soffocare il miglioramento del contrasto)



Maxg = Massimo livello di grigio nell'immagine di input
(Ming è il minimo)

In pratica prendiamo il massimo valore e lo portiamo a 255 nelle nuove immagini, mentre il ming lo portiamo a 0 (cioè nero).

Per gli altri valori basta impostare una semplice proporzione.

Per ogni livello di grigio sottraggo ming, avrei imposto una proporzione

$$\rightarrow \text{ming} : 0 = \text{Maxg} : 255$$

cioè:

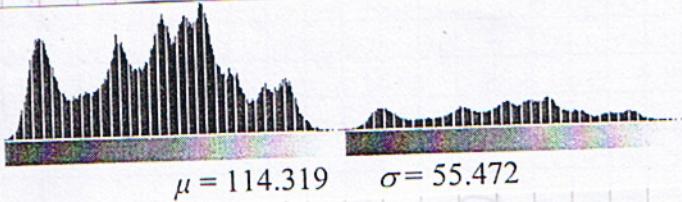
$$\left[\frac{g - \text{ming}}{\text{Maxg} - \text{ming}} \cdot 255 \right]$$

In output avrò l'immagine mostra più contrastata.

Il numero di livelli di grigio sono = $\text{Maxg} - \text{ming} + 1$

d'istogramma relativo allo stretching sono:

In output ci saranno infatti i livelli di grigio non utilizzati e questo spiega i continui buchi nell'istogramma (dei binari bianchi nel plot)



EQUALIZATION (istogramma)

d'equalizzazione è una tecnica che "dovrebbe" migliorare la visualizzazione delle immagini come per lo stretching, modifica i livelli di grigio dell'immagine di input secondo una Look Up Table che copre l'intero range di variabilità $[0, 255]$. Inoltre, produce un nuovo istogramma più uniformemente distribuito.

```
function out=myeq(img)
% esempio: equalized=myeq(lena);
s=255/prod(size(img));
lut=uint8(round(s*cumsum(uint8(img),256)));
out=lut(img);
```

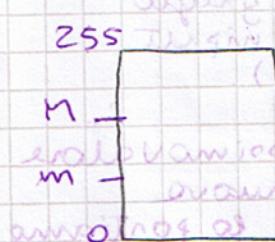
(un esempio su questo codice potrebbe essere quello di definire qualche altro algoritmo che permette di distribuire uniformemente l'istogramma)

l'immagine risulta ancora più contrastata.

Lo scopo è di distribuire meglio i livelli di grigio per sfruttare al meglio la luminosità.

d'Equaliz. difatta sempre i livelli di grigio ma a differenza dello stretching è come se fosse il doppio dell'alto in orizzontale e verticale.

Tutte queste operazioni nascondono ovviamente una Look Up Table.



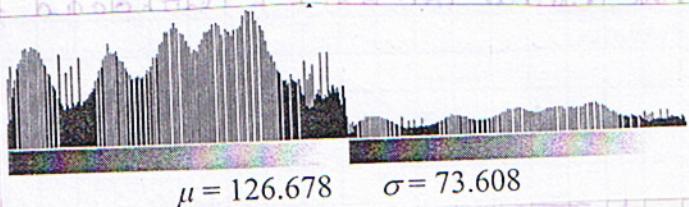
Ci sono diversi algoritmi per l'equalizzazione. Dobbiamo immaginare di dividere i livelli di grigio in 3 aree fondamentali, in modo inoltre che queste 3 aree siano di grandezza simile (o almeno quanto possibile).

Questo codice funziona in Matlab.

Le formule negli algoritmi più comuni usano una notazione ad esempio le sommatorie

d'istogramma relativo all'equalization sono:

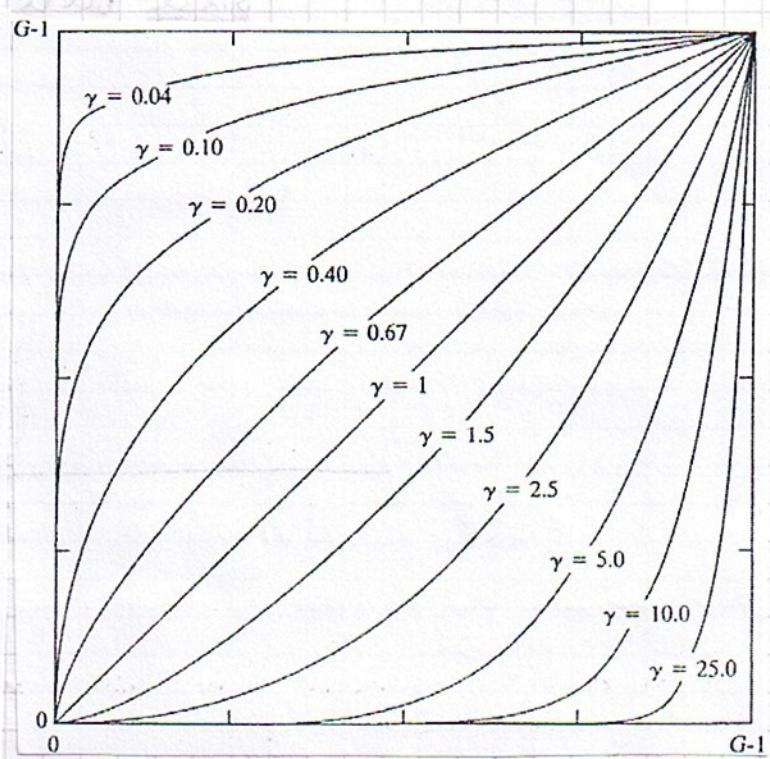
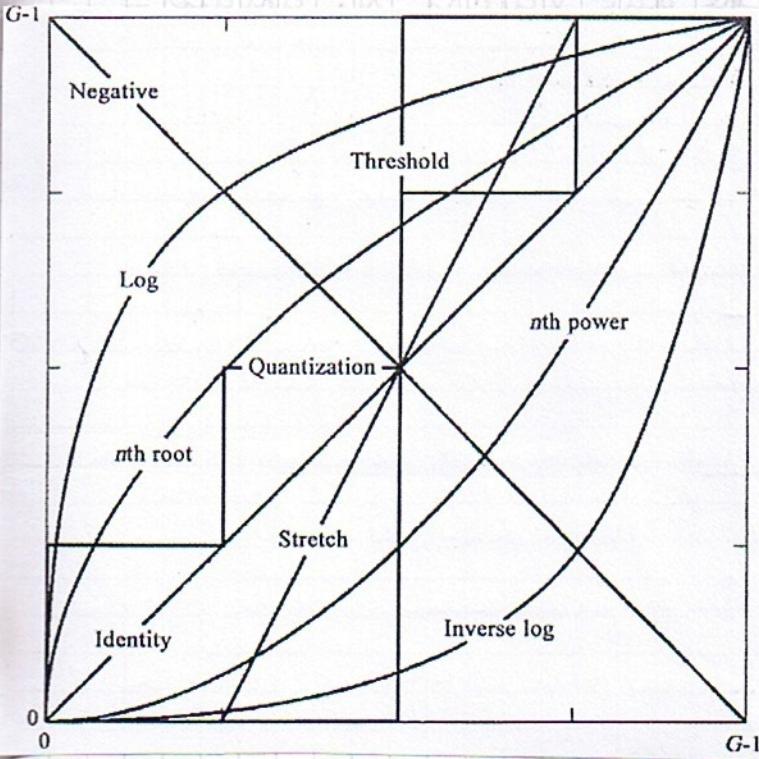
Il contrasto è aumentato e ho buchi sempre più ampi: esistono buchi non ci sono anotende mentre come invece c'è il stretching (il plot è quindi meno uniforme)



→ Normalizzato e appiattito

(Gli istogrammi sono sempre relativi alle stesse immagini)

Vediamo dei grafici che mi quadrano un po' le tecniche in termini un po' più sistematici:
(e schematici)



Esempi di LUT
(come funzione non come tabella)

I valori di γ (che sono standard nei pc) -
conoscere il livello di γ vuol dire mantenere lo stesso livello di luminosità. Anche le periferiche hanno un γ con un livello impostato.

Per questo ad esempio, se andiamo a stampare delle foto, e le vogliamo esattamente come ci piacciono sul nostro pc, potremmo comunicare il valore di γ al fotografo. (.... no comment!)

THRESHOLD (Sogliazione)

Il threshold o sogliazione è la tecnica secondo la quale tutto ciò che sta al di sotto di un certo valore di soglia t va ad assumere un determinato valore, mentre ciò che sta al di sopra ne assume un altro.

E questi valori assunti potrebbero essere 0 e 255 nelle immagini a scala di grigio. L'output sarà infatti una sorta di immagine in binario. Potremmo scrivere noi stessi un algoritmo che trovi uno o più valori di sogliazione, data un'immagine di input.

In generale il valore di sogliazione t dipende non solo dal livello di grigio g del singolo pixel p , ma anche dalle posizioni (x,y) e dall'intorno N del pixel. Potremmo definire "brutale" il threshold ad un solo valore t .

Ci sono comunque 3 tipi di threshold:

- LOCALE**: $t = f(g(p))$ (dipende dai valori di grigio de funzione f che restituisce il valore t di sogliazione)
- REGIONALE**: $t = f(g(p), R)$ (dipende dai valori di grigio de funzione f che restituisce il valore t di sogliazione)
- GLOBALI**: $t = f(g(p), R, I)$ (dipende dai valori di grigio de funzione f che restituisce il valore t di sogliazione)

• LOCALE : $t = f(p(x,y), g(p))$
 (dipende anche dalla posizione del pixel oltre che dai valori di grigio - Il valore di soglia t dipende anche dalla posizione quindi)

• DINAMICO : $t = f(p(x,y), g(p), N(p))$
 (dipende anche dall'intorno N del pixel)

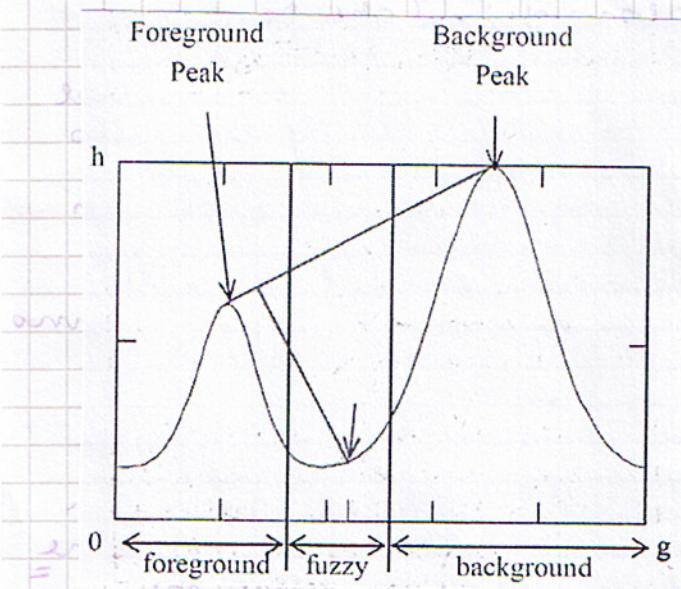
Nel caso di threshold più semplice, siamo appunto nel caso in cui vogliamo binarizzare l'immagine -
 Supponiamo di avere un soggetto e uno sfondo. Potremmo binarizzarne il senso che potremmo rendere nero il soggetto e bianco lo sfondo - ovvero.

THRESHOLD OTTIMALE

Nel caso di un istogramma bimodale (cioè con due picchi), che è il più semplice degli istogrammi; è possibile separare il primo piano (foreground) dallo sfondo (background).

Ovviamente più oggetti ci saranno, più picchi avremo; (ogni oggetto avrà un picco).

Cioè non si chiamerà bimodale magari ma potremmo sempre "binarizzarla".



Un valore di soglia per i istogrammi bimodali può essere ricevuto geometricamente, partendo dal segmento di massima distanza tra le curve dell'istogramma e la tangente congruente i suoi picchi.

Cioè:

(in figura)

il punto **OPTIMUM**

i due picchi quindi. Da questa immagine siamo di partire negli spazi di colore che quindi interseca le curve dell'istogramma spazzolandolo. Ci sarà solo un punto in cui questa perpendicolare avrà lunghezza massima, lo tocchiamo. (se non fosse bimodale ovviamente ce ne sarebbero più di uno) A questo punto basta fare le proiezioni di questo punto (che è l'intersezione (in figura)) sui valori di grigio e trovando proprio il valore ottimale di threshold - "elbow" (in alto a destra).

In questo modo segmentare l'immagine su cui i oggetti sarebbero facili, basterà segmentare geometricamente nel modo descritto sopra. Abbiamo quindi effettuato il threshold attraverso un

algoritmo basato sui segmenti geometrici

→ **ALGORITMO BASATO SUL SEGMENTI GEOMETRICI**

Spesso però l'istogramma non è bimodale, oppure è bimodale ma mascherato da rumore -
Partendo dall'assunzione che l'illuminazione è uniforme almeno localmente, si può calcolare una soglia ture adattiva:

$$t = [\mu]; \quad t = \text{median}; \quad t = \left[\frac{\mu + \text{median}}{2} \right]$$

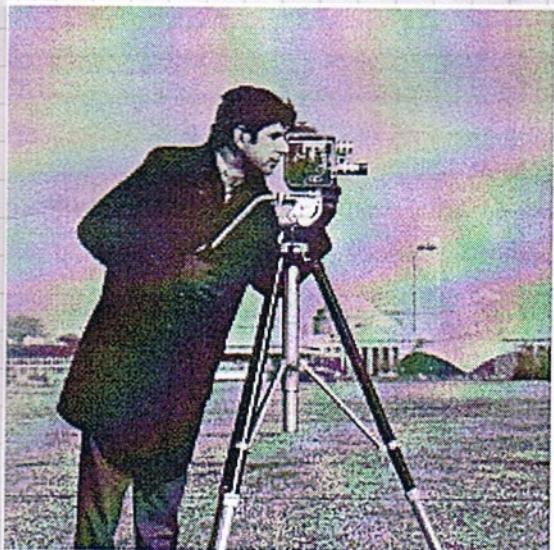
(potremo inoltre eliminare il rumore con delle tecniche che vedremo nelle wavelet)

Media e Mediane quindi. (Più grande è il raggio, più sono globali).

ESEMPIO:

$$t = 119$$

(valore di media)



Input

$$t = 144$$

(valore di mediana)



$$t = 131$$

(valore medio di media e mediana)



THRESHOLD ITERATIVO (detto anche ALGORITMO DI OTSU)

Per un dato istogramma, relativo ad un'immagine con due oggetti, un valore di sogliatura sufficientemente buono puo' essere facilmente ricavato minimizzando iterativamente la varianza all'interno dei due insiemi (foreground e background).

Initialmente l'immagine è segmentata arbitrariamente in due parti e sono calcolati i rispettivi valori medi \bar{X}_1 e \bar{X}_2 .

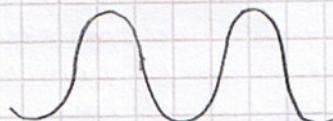
Il nuovo valore t di sogliatura si posto uguale alle media di \bar{X}_1 e \bar{X}_2 e la segmentazione è ricalcolata.

Il procedimento termina quando il valore di t si stabilizza.

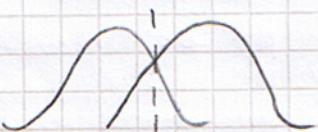
Esercizio: Estendere questo metodo quando sono presenti tre componenti nell'immagine.

Quindi questo caso iterativo è stato visto per il caso bimodale ma si può generalizzare -

Il caso ottimale sarebbe:



Ma potremmo anche avere:



(e sarebbe qui più complicato avere dei valori)

In pratica, parto da un valore di sogliatura t arbitrario (ad esempio 128, che è a metà) e assumo che i due insiemi (foreground e background) siano separati da questo t .
Metiamo che però questo non va bene -

Allora faccio la media delle 1ª classe $\rightarrow \bar{X}_1$
e faccio " " " " " " 2ª " $\rightarrow \bar{X}_2$

(Per estenderlo a K-classi, k-means)

Faccio quindi le medie delle due medie e trovo un nuovo valore di sogliatura t -

Assumo questo valore come buono -

Ma, di nuovo, metiamo che manche questo mi vade bene -

Faccio di nuovo le medie delle due classi e, insomma, ripeto il procedimento finché il valore di sogliatura non si stabilizza.

Non è però garantito che quest'algoritmo termini -

Potrebbe infatti looppare tra i 2 valori di t -

Per ovviare a questa situazione potremmo fissare un numero massimo di iterazioni -

THRESHOLD DINAMICO

Possiamo descrivere i passi dell'algoritmo nel seguente modo:

1- Suddividere l'immagine in blocchi di 7×7 pixel -

Per i blocchi che presentano un istogramma bimodale puo' essere applicato l'algoritmo precedente per ricavare un insieme di valori di sogliature -

2 - I valori di sogliazione dei blocchi che non presentano un istogramma bimodale sono ricavati per interpolazione de 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 sogliazione ricavati ancora una volta tramite interpolazione.

Ora, sorgono due problemi:

Intanto non c'è dubbio che l'immagine si presti ad essere suddivisa in blocchi de 7×7 . In questi casi bisogna cercare un metodo per adattare l'immagine, aggiustarla, magari aggiungendo pixel (PUDDING)

Ma se non trovo subito t vuol dire che l'immagine non è bimodale.
Allora, per riempire questi "blocchetti", calcolo le medie.

Quella lo assegno al pixel centrale del blocchetto in questione.

Ma se i blocchi sono 7×7 , vuol dire che tra i 2 centri di due blocchi adiacenti avremo 6 pixel di cui trovare il valore t, che troviamo per interpolazione.

In questo modo posso trovare un valore di sogliazione t per ogni pixel. (Abbastanza complicato come metodo) (???)

LEZIONE 7

COMPRESSEIONE

21/05/2007

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 allo scopo di ottenere una memorizzazione che richiede minor uso di memoria.

Le tecniche e gli algoritmi di compressione si dividono in 2 fondamentali categorie:

1- LOSSY

→ con perdita di informazione (comprimono i dati attraverso un processo con perdita di informazione che sfrutta la ridondanza nel utilizzo dei dati).

2- LOSSLESS

→ Senza perdita di informazione (comprimono i dati attraverso un processo senza perdita di informazione che sfrutta la ridondanza nella codifica dei dati).

Tecniche con perdita di informazione ottengono delle compressioni molto spinte sui file a scopo dell'integrità del file stesso. Le file prima della compressione e quelli dopo sono simili ma non identici. Normalmente viene utilizzata per comprimere file multimediali. Questi sono spesso troppo grandi per essere facilmente trasmessi o memorizzati quindi si preferisce avere una piccola riduzione delle qualità ma nel contempo file molto più leggeri, un esempio sono le immagini JPEG (per la compressione di immagini) o il formato MPEG (per dati video/audio).

Tecniche lossless invece si occupano 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 in 1 di questi formati, è sempre possibile ricostruire esattamente il file d'origine.

Se infatti abbiamo un'immagine Truecolor in formato TIF non compresso, e la comprimiamo, perderemo informazioni che non potremo più recuperare. Molti infatti non potranno più essere ripristinati.

Per questo se dobbiamo lavorare su une immagini, conviene lasciare il formato TIF finché non abbiamo finito di lavorarci, e solo allora utilizzare JPEG.

Gli algoritmi di compressione vengono detti anche CODEC.

Per quanto riguarda le immagini non compresse abbiamo il tipo

RAW = Formato di memorizzazione delle immagini usato da alcune fotocamere digitali, senza perdita di informazioni sui dettagli dell'immagine proveniente dal sensore, come invece avviene per il formato JPEG. All'unità di misura di questi formati è il "bpp" = "Bit per pixel".

Quanto spazio occupa un'immagine in formato RAW?

occupa:

$$(Larghezza \times Altezza) + Intestazione$$

byte

cioè si dice la matrice
dell'immagine più l'intestazione
che l'h e w, ma non è detto
che per questi 16 byte sia
sufficiente)

Di solito il numero di byte
occupati da file non
compresso sono: $(h \times w) + 4$

Vediamo degli esempi di utilizzo della memoria:
(ad es. in una immagine 100×100)

TrueColor: $100 \times 100 \times 3 + 4$ // byte. 3 per il Truecolor, 4 è la costante additiva per h e w vista prima per l'intestazione, ma trascurabile

Color: $100 \times 100 \times 1 + 4 + LUT$ // byte. Ogni pixel è una terna di colori (3×256)

16 colori: $\frac{100 \times 100}{2} + 4 + \text{tavolozza}$ // perché ogni byte codifica 2 pixel (2 pixel consecutivi sono codificati con un byte) da tavolozza sarebbe 3×16

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

do → Bit Per Pixel: $bpp = \frac{C}{N}$

Rapporto di compressione: ratio = $\frac{KN}{C}$

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. Di solito le informazioni aggiuntive blabla non sono incluse nei calcoli.

MISURE DELL'ERRORE: Con gli algoritmi di compressione si introduce quindi un errore, ad esempio, se nel ruo ci sono quelle di tipo Lossy, perdiamo informazioni che vengono introdotte artificiali. Ma questo possiamo permettere anche accadendo in immagini o in file multimediali; ma se abbiamo ad esempio una foglio di word non possiamo perdere informazioni.

d'errore (distorsione) introdotto dagli algoritmi di compressione (lossy) è quantitativamente misurabile tramite:

• MEAN ABSOLUTE ERROR: (MAE)

$$\frac{1}{N} \sum_{i=1}^N |g_i - g'_i|$$

→ g'_i è il livello di grigio del pixel corrispondente nell'immagine compressa.

a sinistra: Rumore Medio Assoluto

da formula ci dice come misurare questo valore.

N = numero dei pixel

ci sono in g'_i = livello del grigio di quel pixel

il cui valore consigliato

DPI = Unità di misura (dots per Inch), punti per pollice; quantità di informazioni grafiche che possono essere rese da una pittura. Più elevato è il valore, più elevata è la risoluzione.

→

81

MEAN SQUARE ERROR (MSE)

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

PEAK TO PEAK SIGNAL TO NOISE RATIO (PSNR)

$$10 \cdot \log_{10} \left[\frac{255^2}{MSE} \right]$$

Queste misure di "disturbo (errore) valgono sempre, non solo per la compressione".

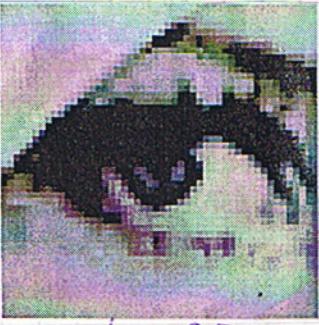
Queste misure però, non coincidono con le normale valutazioni personali. Ad esempio, l'occhio umano non confronta i singoli.



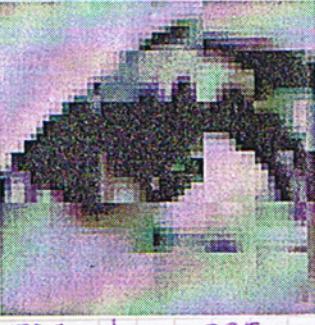
Input: bpp = 8.00
MSE = 0.00



JPG: bpp = 1.00
MSE = 17.26



JPG: bpp = 0.50
MSE = 33.08



JPG: bpp = 0.25
MSE = 79.11

pixel, ma permette una stima qualitativa della distorsione globale. Inoltre, queste misure non confrontano artefatti come i blocchi o le spaccature.

In pratica, si due immagini differiscono per un certo numero k di pixel, e questo è piccolo abbastanza, il nostro occhio le percepisce come uguali. Ma queste misure ci dicono che non è così (a dire del singolo pixel).

Non sono infatti misure per blocchi ma per pixel, guardano l'errore puntuale e non globale o locale come il nostro occhio.

MSE misura in dB (decibel) e così l'errore viene calcolato rispetto all'originale.

$bpp = 1.00 \rightarrow$ significa che l'immagine occupa 118 rispetto all'originale.

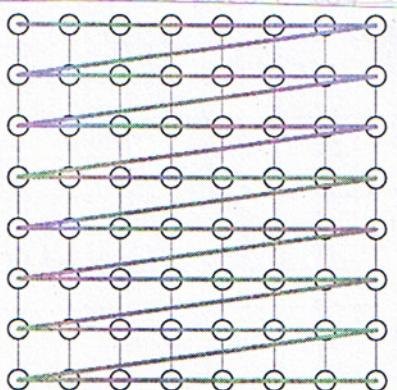
$bpp = 0.50 \rightarrow$ Ogni pixel ha mezzo bit

$bpp = 0.25 \rightarrow$ utilizziamo 114 di bit (ogni bit codifica 4 pixel)

ZIGZAG-ING E' molto importante il modo in cui le immagini vengono lette per compressione.

Il modo dipende infatti dall'ordine in cui vengono letti gli scambiati pixel dell'immagine, se ci sono variazioni di

(Rallent And Code \rightarrow RLC)



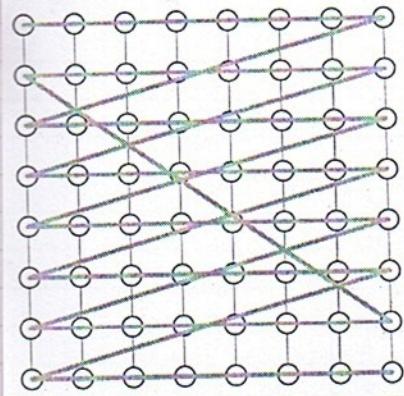
SCANSIONE RASTER

Potiamo quindi caso dall'astrazione che i pixel siano uniformi.

L'effetto desiderato è che chi vede l'immagine vede via via già vista o comunque avendone bisogno.

Per questo ci sono altri metodi migliori per leggere le immagini avendo solo

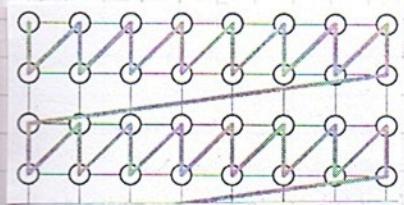
solo quando avremo ricevuto tutti i pixel.



→ SCANSIONE INTERLACCIATA

Vengono visualizzate prime le righe dispari e in questo modo ci facciamo già un'idea dell'immagine, poi continui con le restanti.

(le immagini GIF possono sfruttare questo principio nella loro scansione)



→ E' migliore del raster pure come modo di effettuare la scansione ma possiamo ancora trovare metodi migliori.

In questo modo l'immagine appare dall'alto verso il basso, come si può intuire.

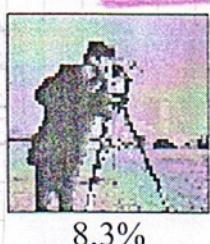
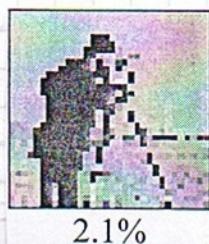
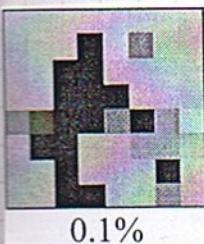
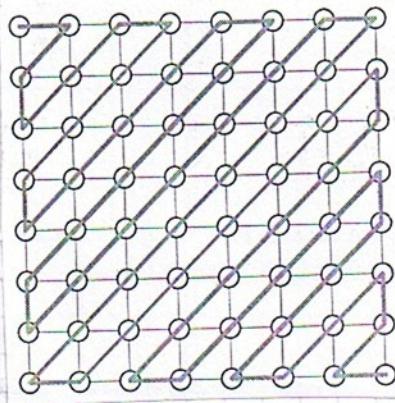
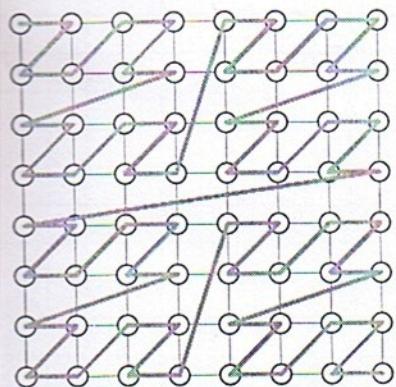


immagine scritta, decidere se ci interessa e prosegui oppure no.



SCANSIONE JPG →

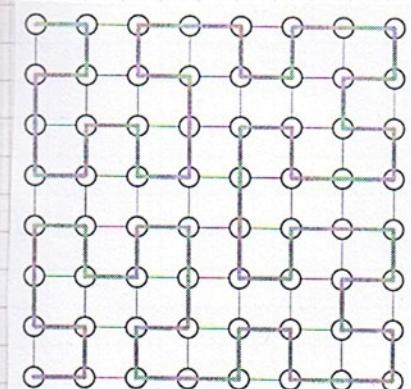
Con la scansione JPG vediamo che già con l'8.3% della scansione possiamo già copiare circa



→ SCANSIONE DI MORTON

Nella scansione di Morton così come in quelle di Peano-Hilbert non parte innanzitutto dall'assunzione del principio di località (ovvero in linea di massima possiamo supporre che pixel vicini abbiano un livello di grigio simile). Ma nella scansione raster ad esempio, perdiamo il principio di località ogni volta che

"andiamo a capo".



→ SCANSIONE DI PEANO-HILBERT

Questi due metodi tengono invece conto del principio di località.

Nella scansione di Morton i pixel vicini restano ancora vicini durante la scansione, sia a livello locale che puntuale.

(i blocchi adiacenti quindi restano tali)

La scansione di Peano-Hilbert non può definire ricorsivamente. Questa sfrutta ancora meglio il principio di località, e

gli pixel vicini restano vicini.

Solo in casi particolari abbiamo peggioramento anti che un miglioramento nell'utilizzare questo metodo.

CODICE GRAY

Vediamo quanto le codice gray da quello binario, come funziona e come si costruisce (già visto in sistemi di elaborazione e Architetture I).

0 0 0 0 0	0 0 0 0 0
1 0 0 0 1	1 0 0 0 1
2 0 0 1 0	2 0 0 1 1
3 0 0 1 1	3 0 0 1 0
4 0 1 0 0	4 0 1 1 0
5 0 1 0 1	5 0 1 1 1
6 0 1 1 0	6 0 1 0 1
7 0 1 1 1	7 0 1 0 0
8 1 0 0 0	8 1 1 0 0
9 1 0 0 1	9 1 1 0 1
10 1 0 1 0	10 1 1 1 1
11 1 0 1 1	11 1 1 1 0
12 1 1 0 0	12 1 0 1 0
13 1 1 0 1	13 1 0 1 1
14 1 1 1 0	14 1 0 0 1
15 1 1 1 1	15 1 0 0 0

codifica binaria

0 0 0 0 0	0 0 0 0 0
1 0 0 0 1	1 0 0 0 1
2 0 0 1 0	2 0 0 1 1
3 0 0 1 1	3 0 0 1 0
4 0 1 0 0	4 0 1 1 0
5 0 1 0 1	5 0 1 1 1
6 0 1 1 0	6 0 1 0 1
7 0 1 1 1	7 0 1 0 0
8 1 0 0 0	8 1 1 0 0
9 1 0 0 1	9 1 1 0 1
10 1 0 1 0	10 1 1 1 1
11 1 0 1 1	11 1 1 1 0
12 1 1 0 0	12 1 0 1 0
13 1 1 0 1	13 1 0 1 1
14 1 1 1 0	14 1 0 0 1
15 1 1 1 1	15 1 0 0 0

codifica Gray

Un codice gray ad n-bit si costruisce attraverso un algoritmo ricorsivo abbastanza semplice. Si parte dal primo bit (quello meno significativo (destra)), si mette uno zero sopra ed un uno sotto, e abbiano fatto lo 0 e l'1 che coincidono con quelli in binario.

Al passo successivo, si mette una riga al di sotto dell'uno, come se fosse uno specchio, e si ricopiano le cifre invertendo l'ordine, con la riga che funge da specchio, appunto.

Si termina mettendo uno 0 davanti alla sequente costruita se questa è sopra la riga, altrimenti si aggiunge un 1.

Ora siamo arrivati ad un codice a 2bit.

Ittando i passi appena scritti, cioè si mette la riga, si specchia (ribalta) le sequenze e si aggiunge il bit più significativo, si costruiscono codici a ad n-bit.

Siamo vedendo questo codice perché di solito i bit plane delle codifiche di Gray sono più comprimibili.

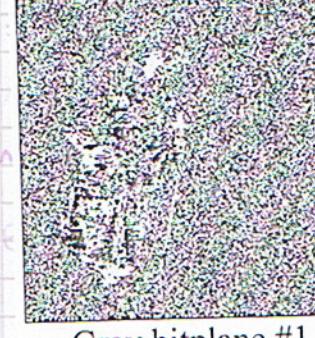
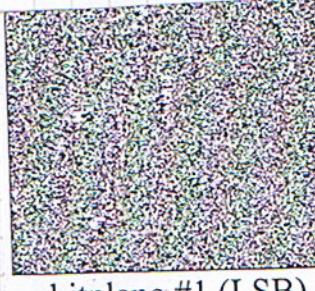
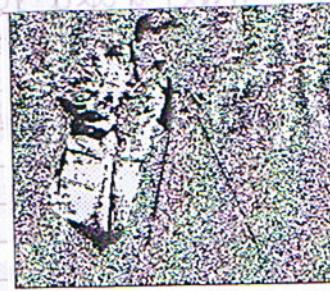
(vengono compressi meglio quindi singoli bit plane)

Da un numero al suo successivo infatti questa codifica stravolge 1 solo bit.

Non in tutte le versioni di Matlab troviamo le funzioni predefinite per le codifiche gray (anzi, solo nelle ultime versioni), ma le funzioni sono ugualmente facili da realizzare, come possiamo vedere.

```
function g=dec2gray(d)
g=bitxor(d,bitshift(d,-1));
```

```
function d=gray2dec(g)
d=g;
for i=[-32,-16,-8,-4,-2,-1]
    d=bitxor(d,bitshift(d,i));
end;
```



MSB → Most Significant Bit
(Bit più significativo)

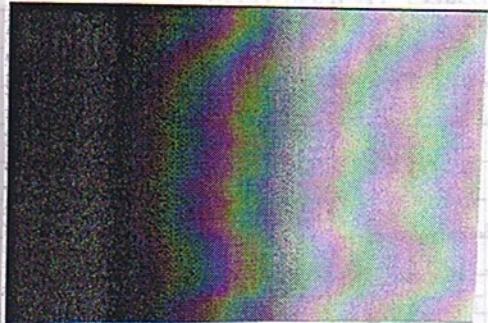
Andando avanti con i bitplane vediamo che perdiamo molto.

Converting da binario in gray ho quindi 8 bitplane -
comprimendo con il codice gray ottengo un migliore fattore di compressione.

LSB → lowest significant Bit

SEGMENTAZIONE

segmentazione automatica è uno dei compiti più ardui dell'analisi di immagini digitali; poiché spesso presupone la conoscenza dell'immagine stessa.



→ Quante componenti sono presenti in questa immagine?

E' difficile rispondere a questa domanda, perché in realtà la cosa è del tutto soggettiva, in questa immagine come in altri o in altri contesti.

In questo caso quindi non è possibile definire a priori il numero di segmenti sulla luminosità - In alternativa potremmo decidere noi stessi un numero arbitrario di segmenti (ad esempio 3 per questa figura - ma si potrebbe anche dire infiniti ...)

Nella compressione la segmentazione vale nel senso che potremmo pensare di comprimere definendo varie regioni, cioè segmentando l'immagine per blocchi (o anche per colonne).

Formalmente la segmentazione di un'immagine è un partizionamento dell'immagine I in regioni: R_1, \dots, R_i tali che:

- le regioni sono uniformi: $P(R_i) = \text{true}$
(abbiamo N regioni R_i)

Uniforme vuol dire ad esempio una regione tutta bianca, tutta rossa etc, oppure con una qualsiasi proprietà che sia la stessa per tutti i pixel della regione.

Potremmo definire anche il predicato di uniformità sulla texture o sulla luminosità.

- le regioni confinanti hanno proprietà differenti: $P(R_i) \neq P(R_j)$ con $i \neq j$

- il ricoprimento è completo: $\bigcup R_i = I$

- le regioni sono a due a due disgiunte: $R_i \cap R_j = \emptyset$ con $i \neq j$

Notiamo quindi che non può bastare il binario. Vale anche qui il teorema dei 4 colori: data una superficie divisa in regioni connesse sono sufficienti 4 colori per colorare ogni regione facendo in modo che regioni adiacenti non abbiano lo stesso colore. (85)

Nel nostro caso intendiamo almeno 4 colori, ma se so che ne fanno anche di più, ma di sicuro non due.

L'unione delle regioni (segmenti) } → E' UNA PARTIZIONE
deve restituire l'intera immagine.

Un banale algoritmo di segmentazione ad esempio, per decidere il numero di segmenti da usare, potrebbe essere quello di "chiedere" fisicamente ad un tot di persone, e poi fare una media del numero di segmenti che ognuno vede.

BACCHETTA MAGICA → Questo strumento di Photoshop si basa appunto sul principio della segmentazione. Si clicca su un pixel x che rappresenta il cosiddetto "pixel sema" e si decide un livello di tolleranza "toleranza", ovvero l'ampiezza della regione in cui ancora quel pixel lo meno prossimo a seconda delle tolleranza) pixel, o con quelle qualità.

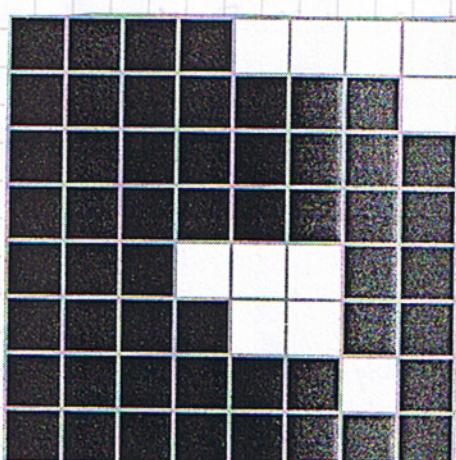
Abbiamo ad esempio un sema "g" e una tolleranza "10", vuol dire che i quattro vicini adiacenti hanno un valore compreso tra $(g-10)$ e $(g+10)$ vengono presi come pixel della regione o nel modo dei segmenti.

Questo viene fatto iterativamente per un certo numero di passi. Questo algoritmo può essere in 2 modi:

- ASSOLUTO → Confronta i pixel sempre con il sema (Photoshop)
- RELATIVO → Confronta sempre con i altri pixel dell'intorno che sto considerando (ma spande troppo).

Il formato JPEG2000 ha tutta la segmentazione (è una struttura dati pietrificata)
ANTIALIASING = Non la vediamo, ma è una tecnica per ridurre l'effetto di aliasing (scalamento) quando un segnale ad alta risoluzione viene mostrato a bassa risoluzione. Questa tecnica ammorbi disce le linee smussando i bordi e omogenizzando l'immagine.

QUADTREE



I quadtree sono strutture "piramidali" in cui l'immagine è divisa in 4 quadranti ad ogni livello, fino al raggiungimento di zone uniformi.

Facciamo l'esempio di aver l'immagine come mi figura, una griglia 8×8 pixel (tutti della stessa dimensione).

Dovendo dividere ad ogni passo ricorrendo in 4 parti le mia griglie.

(per comodità uso potenze di 2 per le dimensioni di una griglia).

In pratica divido in 4 parti e controllo che in ogni parte sia "uniforme". Se non lo è, divido ulteriormente in altre 4 parti ed effettuo lo stesso controllo.

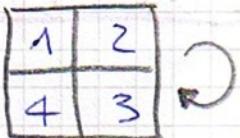
Nel nostro esempio dopo qualche passo avremo:

Continuando da questo passo si puo' anche arrivare alla dimensione del singolo pixel e in questo modo avra' soddisfatto la definizione stessa di Quadtree.

Esiste una corrispondenza tra quadtree e i 4-alberi:

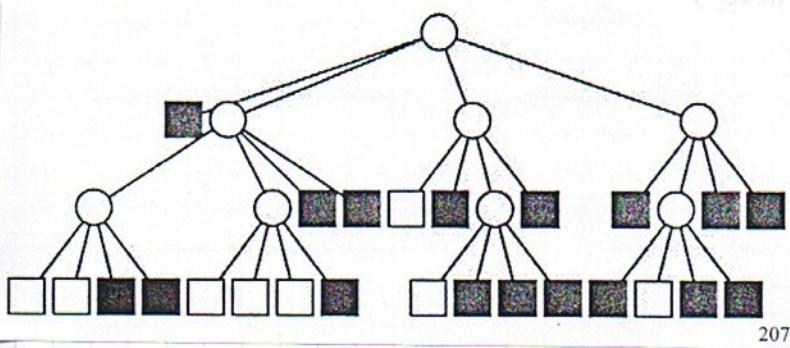
In pratica possiamo rappresentare i quadtree con i 4-alberi come possiamo vedere in figura.

Il modo di suddividere e numerare volta per volta i 4 quadranti e il seguente



Quando arriviamo alle foglie vuol dire che non dovrà più suddividere, e il colore della foglia sarà quello del quadrante uniforme (che non ha quindi più bisogno di essere suddiviso)

Indichiamo le foglie con quadrati.



$\square \rightarrow$ Non c'è più bisogno di suddividere

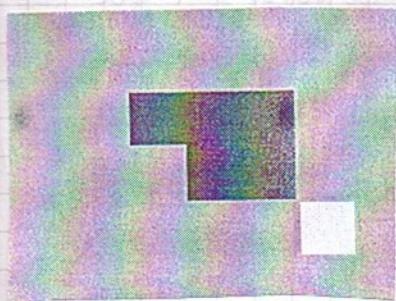
$\circ \rightarrow$ Nodo interno, il processo di quadtree va ancora applicato su quelle griglie

Proprio dei 4-alberi il processo dei quadtree prende il nome di struttura "piramidale".

Ma a che può servire questo processo per la segmentazione?

Potrebbe servire ad esempio perché una volta che ho il quadtree posso riconoscere insieme i quadranti che hanno le stessa luminosità secondo la 4-connectività (merge dei blocchi).

(potremmo vedere anche secondo la 8-connectività \rightarrow stesso colore per chi adiacenti)



\rightarrow I quadtree possono quindi essere usati per la segmentazione: si fondono le regioni 4 adiacenti con le medesime proprietà P

I quadtree possono essere utili per rappresentare le immagini binarie in un codice in genere non particolarmente efficiente... Da 64 bit per la griglia "raw" riusciamo a passare meglio a 58 bit per l'albero quadtree.

$\circ \rightarrow 0$	$\} \text{ Codifica}$
$\blacksquare \rightarrow 10$	
$\square \rightarrow 11$	

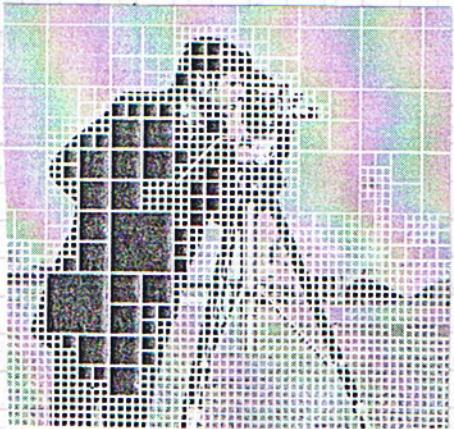
Per leggere l'albero con una visita BFS e ottengo una stringa del tipo:

BFS $\rightarrow 0\ 10\ 0\ 0\ 0\ 0\ 0\ 10\ 10\ 11\ 10\ 0\ 10\ 10\ 0\ 10\ 10\ 11\ ...$
(etc etc per l'albero)

Codice Prefisso → Codice non ambiguo, di una sola parola.
Per essere univoco un codice prefisso deve essere
completo e non ambiguo fino alla fine; in pratica finché ogni prefisso non
è composto da più di 4 cifre (tipi i numeri di telefono)
Quindi secondo le istruzioni prioritaria in lunghezza (BFS) otengo il
codice scritto prima (e non completato)

Da 8×8 pixel → 64 bit che con questo metodo diventano 58
(non è un gran che come codice)

E con i livelli di grigio? Per estendere i quadtree alle immagini
con 8 livelli di grigio, si definisce un
"criterio di uniformità" basato sulla
varianza dei pixel nel medesimo blocco:
se superiore ad una data soglia, il blocco
deve essere suddiviso.



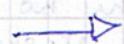
In pratica dobbiamo fissare un valore di
soglia (tipically threshold) e, se il valore
medio è al di sopra di questo valore di
soglia allora il quadtree vuol dire che non
è già uniforme e va suddiviso secondo
i quadtree.

Ovviamente dipende da quale valore di soglia
scelgiamo otterremo dei quadtree più o meno numerosi.

Se minimizziamo di suddividere un'immagine con i quadtree
assegnando una deviazione σ (deviation standard)



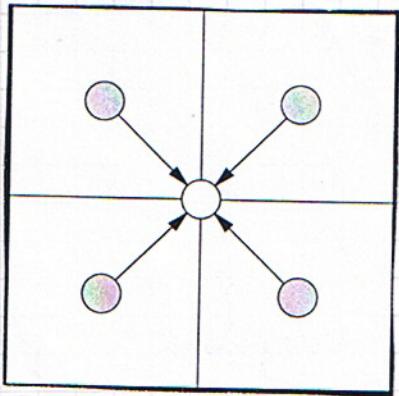
(Immagine originale)



(Quadtree $\sigma = 20.00$)

All'aumentare della deviazione standard di minimi come i blocchi e quindi con meno blocchi per la stessa immagine saranno più evidenti gli artefatti.
Con $\sigma = 5.00$ sembrano di avere l'immagine originale.

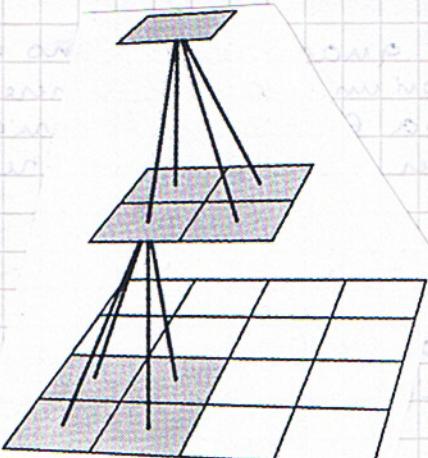
Vediamo il perché della struttura piramidale:



È come la suddivisione ad albero.
cioè ogni griglia è potenzialmente divisibile in 4 altre parti.

Graficamente: →

→ Ogni 4 pixel ne ottieniamo uno



86) I quadtree sono strutture piramidali.

I quadtree sono utili anche per rappresentare le immagini con differenti risoluzioni. I dettagli sono memorizzati alla base delle piramidi, mentre i livelli superiori permettono un'analisi più veloce. I livelli di grigio possono passare dai figli al padre tramite, ad esempio, le funzioni and, xor, min, max, mean, median, ...

La dimensione la si dice dal livello ℓ , ma non so quanto sono grandi i blocchi (lo vedo delle coordinate) e in questo modo so da dove partire per il blocco successivo.

Possiamo usare i quadtree per rappresentare le immagini in multi risoluzione:

Proprietà:

- le fogli rappresentano i pixel
- ogni nodo di livello ℓ contiene le coordinate del primo pixel in alto a sinistra
- Indichiamo con $(i, j, 2^\ell)$ la sottomatrice di dimensione $2^\ell \times 2^\ell$ identificata dal pixel (i, j)

Tra i principali problemi dei quadtree si raggruppano:

- dipendenza delle dimensioni dell'immagine (assumiamo 2^K) quindi dovo aggiungere o togliere a ricordare di "cessi".
- perdita di committitività
- immagini simili (anche versioni leggermente modificate della stessa immagine) possono avere quadtree completamente differenti

$2^\ell \rightarrow$ livello coordinate

Diviso in blocchi (magon zone) - quadranti uniforme

Ovviamente basta avere 1 solo pixel diverso e i quadtree sono diversi (perché il blocchetto che cambia si propaga a cercata)

LIVELLO 0: La radice rappresenta l'intera immagine $(0, 0, 2^K)$.

I suoi figli rappresentano in senso orario le sottomatrici di dimensione 2^{K-1}

$$(0, 0, 2^{K-1}) \quad (0, 2^{K-1}, 2^{K-1}) \quad (2^{K-1}, 2^{K-1}, 2^{K-1}) \quad (2^{K-1}, 0, 2^{K-1})$$

LIVELLO ℓ : I figli di $(i, j, 2^\ell)$ sono rappresentati da:

$$(i, j, 2^{\ell-1}) \quad (i, j+2^{\ell-1}, 2^{\ell-1}) \quad (i+2^{\ell-1}, j, 2^{\ell-1}) \\ (i+2^{\ell-1}, j+2^{\ell-1})$$

Il livello 0 (nell'albero) \rightarrow modo più interno è a K -profondità $\log_2(\text{dim}) \rightarrow$ profondità

Vediamo una funzione che mi permette di passare dai figli al padre e viceversa.

La funzione L restituisce le posizioni del padre di ogni nodo interno:

$$L: I_\ell \rightarrow I_{\ell-1} \text{ con } 0 < \ell \leq K$$

$$L: (i, j) \rightarrow \left(\left\lfloor \frac{i}{2} \right\rfloor, \left\lfloor \frac{j}{2} \right\rfloor \right)$$

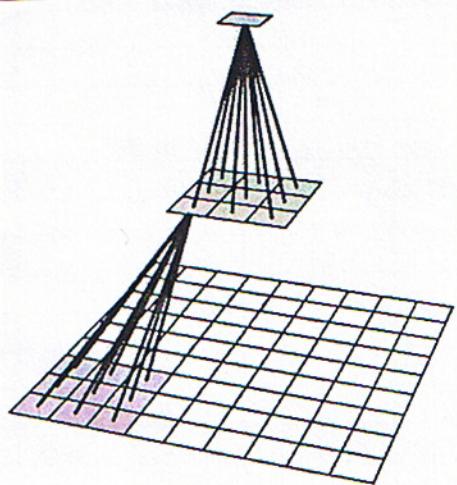
e viceversa:

$$L^{-1}: I_{\ell-1} \rightarrow I_\ell \text{ con } 0 \leq \ell < K$$

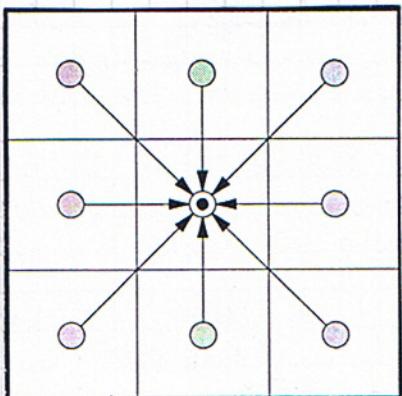
$$L^{-1}: ((2i, 2j), (2i, 2j+1), (2i+1, 2j), (2i+1, 2j+1)) \rightarrow (i, j)$$



In effetti, possiamo generalizzare i quadtree in modo che ogni blocco non sia necessariamente suddiviso in 4 sotto-bloccchi.



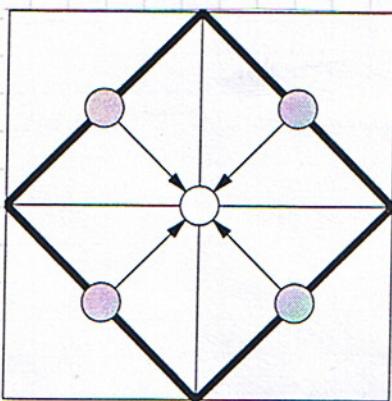
In pratica il disco so
piramide si può
estendere
(per le dimensioni dei
bloccetti)
Ad esempio 9 pixel a
volta



OCTREE = Inoltre è possibile
estendere il quadtree alle 3
dimensioni, si ottiene il
cosiddetto octree, ogni cubo viene
suddiviso in 8.

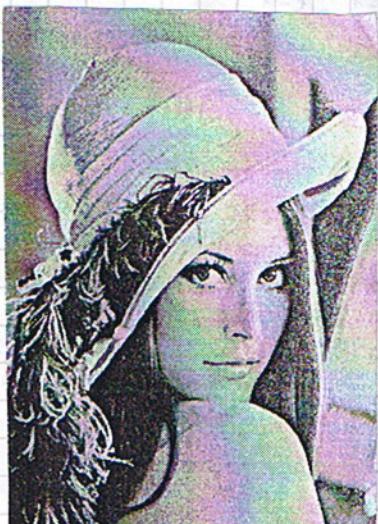
Per il resto, l'octree è identico al quadtree.
Divido quindi il volume in $2 \times 2 \times 2$.
L'octree è quello che ad esempio viene utilizzato nei videogiochi per vedere se un oggetto ad esempio è davanti o dietro l'altro.

altre strutture piramidali regolari

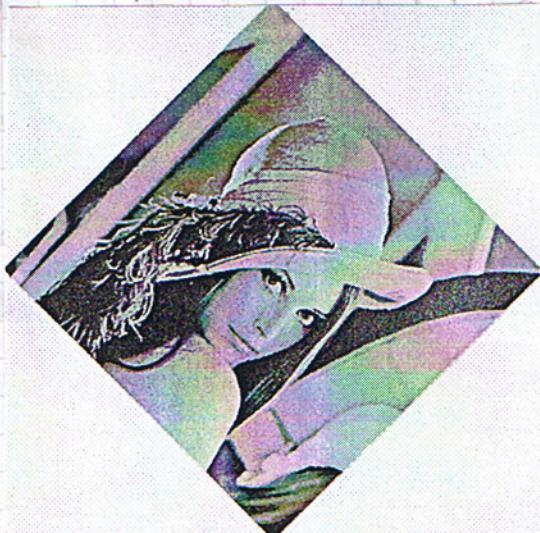


Per non perdere la connettività possiamo decidere
di dividere non per 2 ma per $\sqrt{2}$ ad esempio.
→ Ogni 4 pixel ne otengo 2 me ruotati
di 45° .

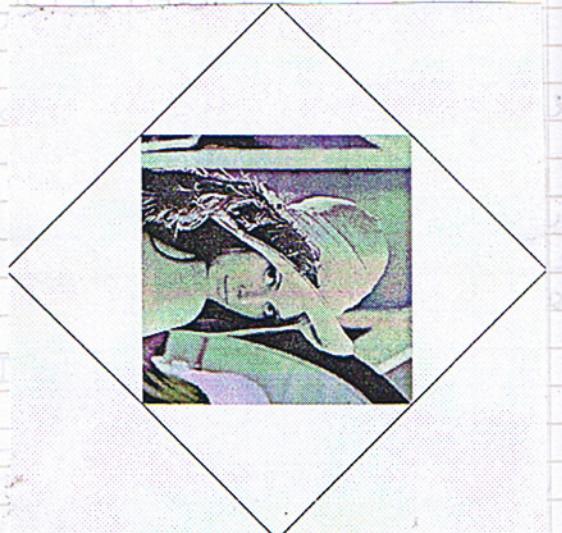
In questo modo si mantiene la connettività.
Avremo una piramide regolare in cui blocchi
distinti mantengono le connessioni da un
livello all'altro (in pratica 2 pixel condividono
qualcosa da blocchi adiacenti precedenti).



Livello 0 (512x512 pixel)



Livello 1 (362x362 pixel)



Livello 2 (256x256 pixel)



MISURE DI SIMILARITÀ Vogliamo confrontare ancora le immagini, vediamo qualche formula per le misure di similarità.

$$E = \sqrt{\sum_{q \in T} (I(p+q) - T(q))^2}$$

notazione vettoriale

$$R = \sum_{q \in T} I(p+q)T(q)$$

prodotto simile alla convoluzione

$$S = \sum_{q \in T} |I(p+q) - T(q)|$$

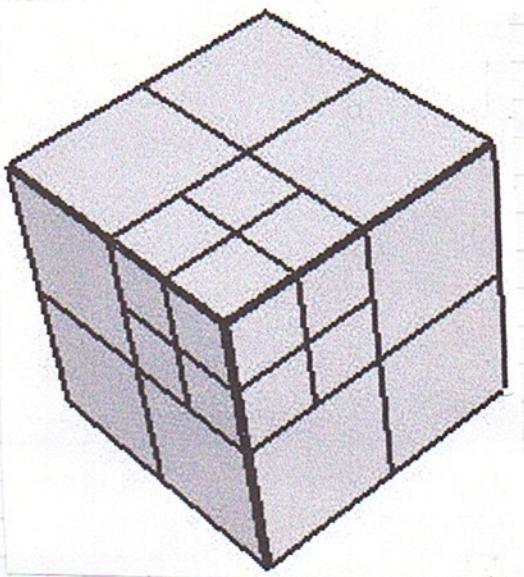
$$NR = \frac{R}{\sqrt{\sum_{q \in T} I(p+q)^2} \sqrt{\sum_{q \in T} T(q)^2}}$$

Osserviamo meglio

SPLIT E MERGE

consiste nel suddividere inizialmente l'immagine in un insieme di "regioni" arbitrarie e disgiunte, quindi procedere allo dividere le regioni con il fine di ottenere un partizionamento dell'immagine in regioni disgiunte ed interamente omogenee e connesse.

Esistono diversi algoritmi che usano questa idea di base - la maggior parte di essi sono sostanzialmente variazioni dell'algoritmo Quadtree Decomposition mirante ad ottimizzare le prestazioni ottenibili. Questi algoritmi di segmentazione sono appunto i Quadtree.



TRASFORMATA DISCRETA COSENO (DCT)

La Trasformata discreta coseno (DCT) di una sequenza $C(u)$ di lunghezza N , è una sequenza $f(n)$, di lunghezza N , definita dalla seguente formula:

1D DCT \rightarrow

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

u = segnale, frequenza

$f(n)$ = segnale di ingresso

dove

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{se } u=0 \\ \sqrt{\frac{2}{N}} & \text{se } u=1, 2, \dots, N-1 \end{cases}$$

CASO MONODIMENSIONALE

1D DCT

mentre la trasformata inversa coseno (IDCT) di una sequenza $f(x)$ di lunghezza N , è una sequenza $C(u)$, di lunghezza N , data dalle seguenti formule:

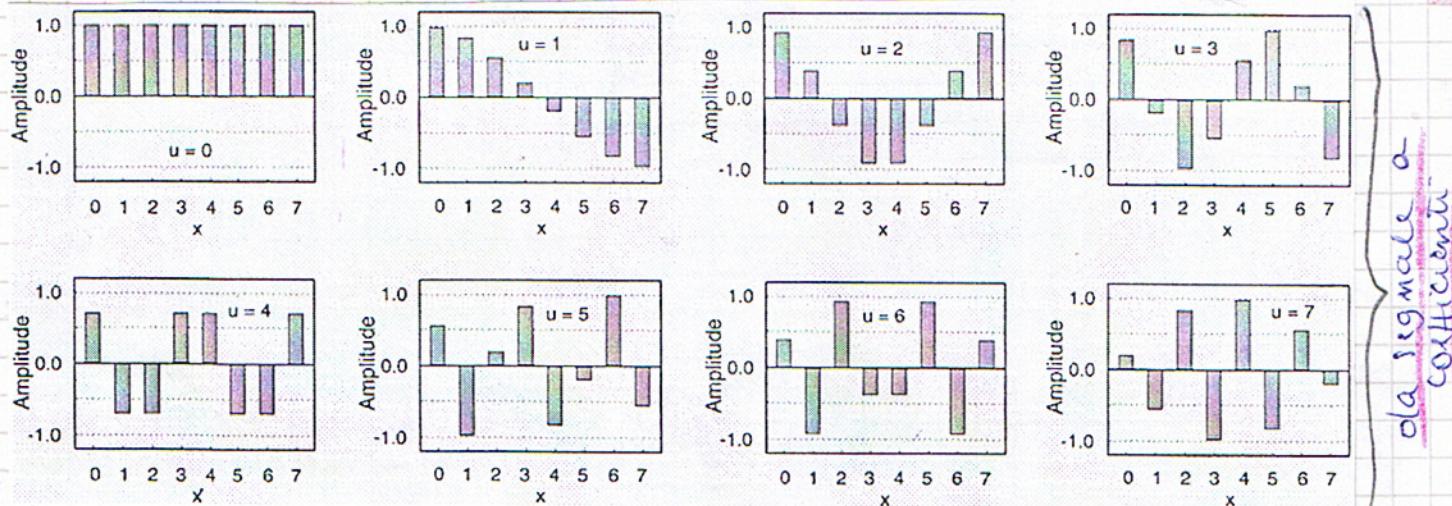
IDCT \rightarrow

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

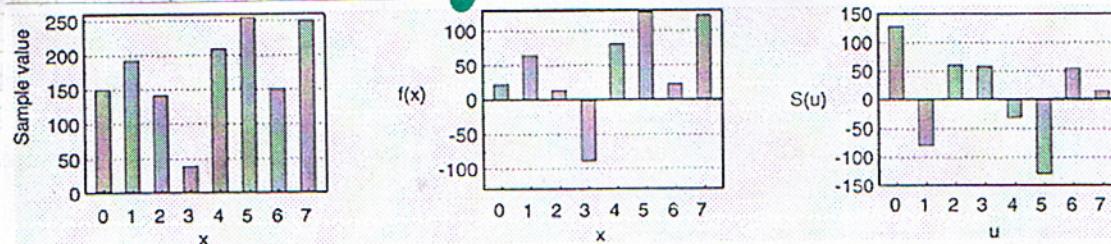
Osserviamo che $\alpha(u)$ e $C(u)$ servono per mantenere costante l'energia. Infatti la proprietà di conservazione dell'energia è sfruttata nei moderni metodi per trasmissione o memorizzazione di immagini (standard JPEG), usate nelle reti internet, nelle telecomunicazioni, televisioni digitali, etc...

con $N=8$ (che mettiamo nelle ascisse x da 0 a 7) osserviamo l'andamento della DCT al variare di u (anch'esso da 0 a 7 sostituendo secondo il sistema per le formule $\alpha(u)$)

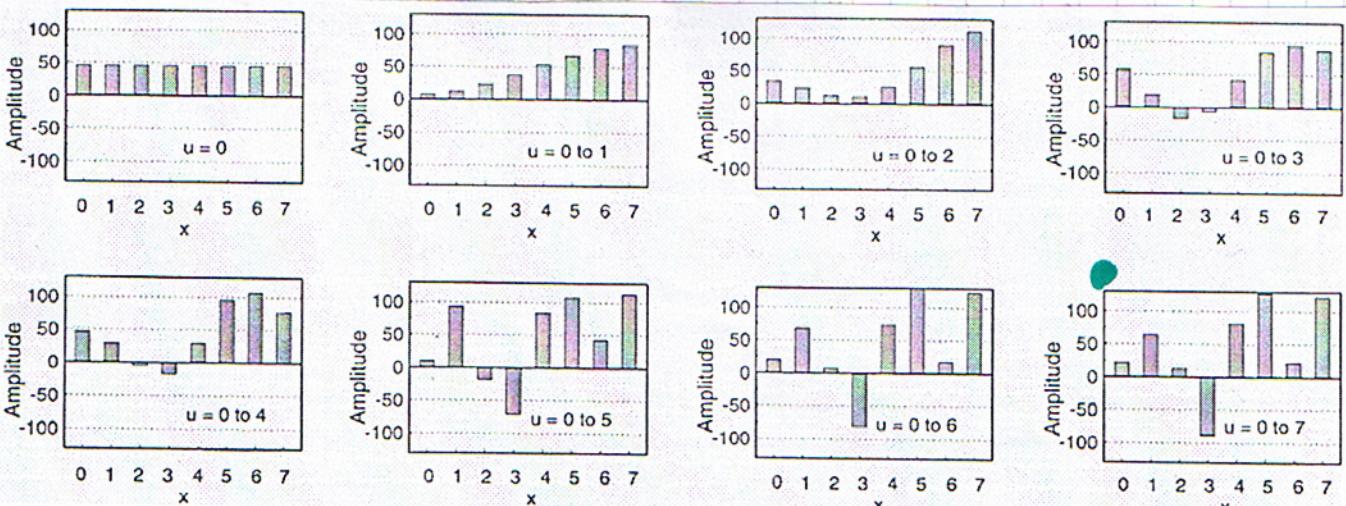
* Ved
per



In pratica ho il segnale f in 8 coefficienti, che possiamo immaginare come la curva del coseno al variare di u.



Come possiamo vedere nella 2a immagine riportiamo i valori da -128 a 128 (per centrare sullo zero).



da coefficienti

L'obiettivo è effettuare la convoluzione con i vari coefficienti, che si ottengono applicando $C(u)$, al fine di ottenere il segnale di ingresso $f(n)$.

Osserviamo infatti che f rappresenta lo stesso segnale $f(x)$. Con quest'ultimo procedimento in pratica, convoliamo i segnali di ingresso con quelli delle u .

Ad ogni passo effettua una somma con il passo precedente e

ottengo una variazione. I coefficienti ci dicono l'ampiezza del segnale, dobbiamo vedere il loro valore (supponendo come esempio di calcolandolo attraverso le formule).

→ Ne risulta che se conosciamo il segnale ci possiamo ricavare i coefficienti, viceversa, se conosciamo i coefficienti possiamo ricostruire il segnale.

1DDCT è quindi la base nel caso monodimensionale e ci indica l'andamento dei coefficienti e segnali al variare di N . (nel nostro esempio sono 8)

Obiettivo = ottenere il segnale di ingresso traslato a meno di approssimazioni.

2D DCT (& IDCT) - CASO BIDIMENSIONALE - * vedi pag. 93

E' il caso bidimensionale della trasformata discreta coseno. Vediamo le formule relative, analoghe a quelle del caso monodimensionale:

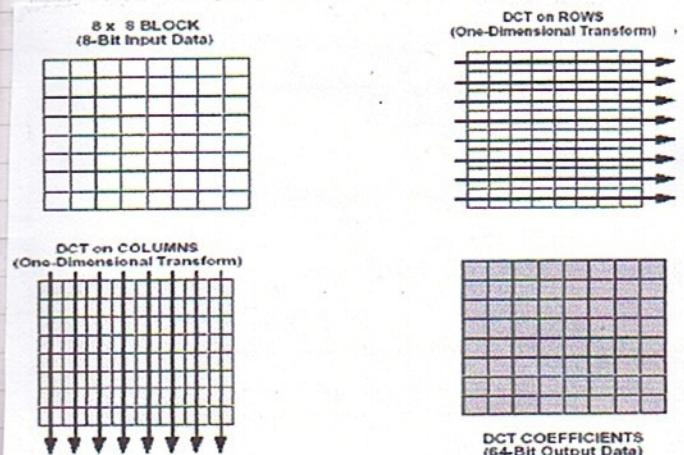
$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cdot \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cdot \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

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

$$\alpha(u) = \begin{cases} \sqrt{1/N} & \text{for } u=0 \\ \sqrt{2/N} & \text{for } u=1, 2, \dots, N-1 \end{cases}$$

Siccome la 2D DCT è separabile nelle 2 dimensioni, possiamo applicare una volta per x e una volta per y le formule del caso monodimensionale.

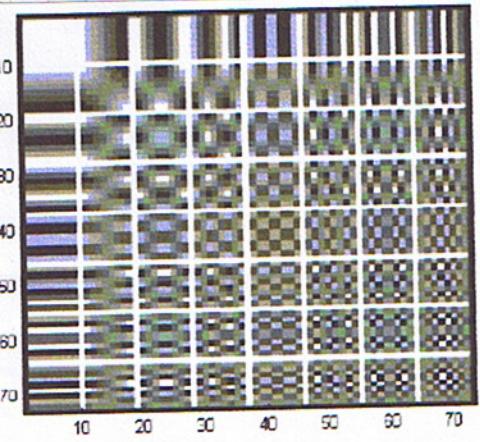
In pratica la 2D DCT lavora scomponendo l'immagine in blocchi di 8×8 pixels e si applica ad ogni singolo blocco.



Per ogni blocco, come si può vedere, si "1D-trasformano" prima le righe poi le colonne.

In pratica, dato che la 2D DCT può essere ottenuta come una combinazione di due 1D DCT, per questo si dice che la 2D DCT è separabile nelle due dimensioni.

avorando quindi su blocchi di 8×8 pixel, avremo 64 funzioni base, che sono quindi fatte come nella seguente immagine:



→ valori dei coseni nella 2D DCT. Nell'immagine qui accanto vengono mostrati i valori dei coseni ordinati al variare dei coefficienti u e v della trasformata.

Questi sono i valori che moltiplicano i pixel dell'immagine originale.

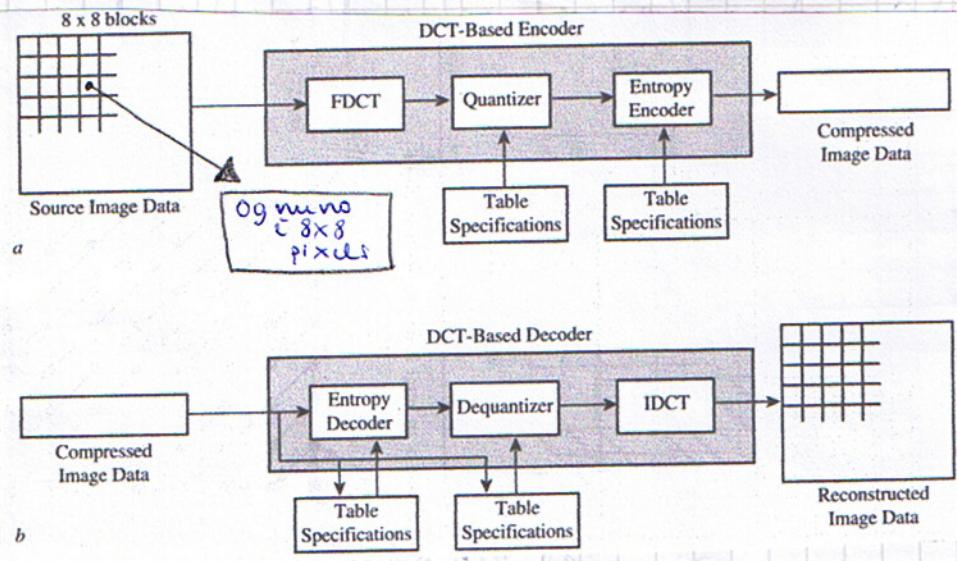
→ DCT Basis

Come vediamo dalle formule che è possibile passare dal segnale ai coefficienti (come in figura) possiamo immaginare un'immagine in cui passiamo dai coefficienti al segnale (e sarebbe simile a questa di partenza) - DCT reconstruction -

Per quanto riguarda i colori, il -128 (e toni scuri) indica un segnale basso, all'inverso lo zero è il grigio medio e 128 (e toni chiari) indica un segnale alto.

Lungo le diagonali → Kernel simmetrica.

JPG (Encoding & Decoding)



JPEG è uno standard che sta per Joint Photographic Experts Group ed è uno standard per la compressione di immagini statiche a toni di grigio. Ovvero è stato pensato per codificare immagini fotografiche, in contrapposizione ad immagini sintetiche.

generate al calcolatore.

Come noto, è uno standard molto diffuso.

Non è un singolo algoritmo, ma utilizza diverse tecniche di codifica:

- Codifica tramite trasformata coseni discrete (DCT)
- Run length Encoding
- Codifica di Huffman

4 diverse modalità di funzionamento:

- Lossless JPEG
- Sequential JPEG
- Progressive JPEG
- Hierarchical JPEG

Ogni immagine JPEG contiene varie frequenze di colore, le basse frequenze corrispondono a colori che cambiano in modo lento e graduale, le alte a cambiamenti fini e particolareggiati. La DCT si vuole quindi necessaria per trovare un modo di rappresentare le immagini (JPEG) in termini delle frequenze di cui sono composte.

Per interpretare l'immagine precedente approfondiamo il concetto generale di DCT.

DCT Il sistema visivo dell'occhio umano dipende considerabilmente dalla frequenza spaziale: essere in grado di decomporre l'immagine in insieme di forme d'onde elementari, ognuna con una particolare frequenza, significa poter separare ciò che l'occhio può realmente vedere da ciò che è invece impercettibile. La DCT è la procedura matematica che fornisce una buona approssimazione di questa decomposizione e sulla quale si basa la costruzione dei blocchi JPEG. Essa permette di ottenere dei coefficienti non correlati, ognuno dei quali può essere pertanto trattato indipendentemente senza perdita di efficienze nella compressione.

* Consideriamo un segmento di 8 campioni adiacenti in scala di grigi rappresentati su 8 bit (valori tra 0 e 255); Dopo aver sottratto ai campioni il valore 128 (valori tra -128 e 127) come richiesto del JPEG, possiamo decomporre questa sequenza in insieme di forme d'onde di frequenze spaziali crescente, per esempio 8 diverse funzioni coseno di ampiezze uniformi ognuna campionata su 8 punti: $f(x) = \cos(ux)$ con u che va da 0 a π .

Queste forme d'onde sono fra loro ortogonali; quindi indipendenti, e possono essere utilizzate per ottenere gli 8 campioni tramite combinazioni lineari: gli 8 coefficienti moltiplicativi rappresentano l'uscita della DCT a 8 punti.

Il coefficiente con $u=0$ è chiamato "coefficiente DC", gli altri "coefficienti AC"; questi nomi derivano dall'uso storico della DCT per analizzare correnti elettriche con componenti sia continue che alternative. E' da notare che il coefficiente DC rappresenta la media degli 8 campioni.

Il processo di decomposizione in insieme di funzioni di base coseno è chiamato FDCT.

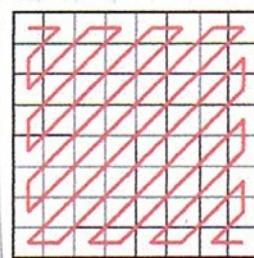
* La procedura per il calcolo della DCT uni-dimensionale può essere estesa per l'applicazione nelle immagini bidimensionali: le funzioni in base coseno sono ottenute moltiplicando le 8 funzioni base uni-dimensionali orientate orizzontalmente con le stesse orientate verticalmente: le prime rappresentano le frequenze orizzontali, le altre quelle verticali.

Moltiplicate x coefficienti opportuni queste 64 funzioni di base possono essere usate per rappresentare i valori dei campioni del blocco 8×8 , esattamente come nel caso uni-dimensionale.

I 64 valori così ottenuti vengono ordinati in 1

array seguendo una sequenza di zigzag, come mostrato nella figura seguente: si codifica e zigzag utilizzata dal JPEG ordine approssimativamente le funzioni di base con frequenze spaziali crescenti.

Purché le funzioni coseno bidimensionali si



ottengono dal prodotto di due uni-dimensionali; l'unica funzione di base continua è quella corrispondente al primo coefficiente più alto a sinistra dello stesso, amalgamando al caso unidimensionale, il coefficiente DC; nella codifica a zigzag questo sarà il primo coefficiente poi seguiranno i coefficienti AC.

Analizziamo i singoli passi dell'immagine per Encoding/Decoding.

Da Originale a Compresso: Dividiamo l'immagine originale in 8x8 blocchi a loro volta divisi in 8x8 pixel.

Applichiamo la decomposizione DCT (pagina precedente).

A questo punto applichiamo la quantizzazione.

La quantizzazione è l'operazione che ci permette di ridurre l'accuratezza con la quale i coefficienti DCT sono rappresentati quando vengono annotati a numeri interi. In pratica, dopo aver applicato la DCT ad un blocco 8×8 , ogni elemento del blocco viene diviso per un coefficiente di quantizzazione distinto ed annotato all'interno più vicino (questo procedimento formalmente porta alla perdita di informazione).

→ de Matrice contenenti i 64 coefficienti di quantizzazione (QuantizationTables), potrebbero essere definite a piacere, ma dopo alcuni esperimenti sono state rese note le tabelle che danno i risultati migliori e sono appunto queste in figura. (questi valori comunque sono prossimi alla soglia delle visibilità ed l'immagine ricostruita usando queste tabelle mostra comunque artefatti quando vista su schermi ad alta qualità; se tutti i valori vengono divisi per due l'immagine risultante sarà indistinguibile).

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

tavella 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

tavella di quantizzazione per crominanza

Il segnale di ingresso è un intero positivo mentre i coefficienti sono floating point quindi a' abbiamo puro.

Visibilmente l'immagine ricostruita usando queste tabelle mostra comunque artefatti quando vista su schermi ad alta qualità; se tutti i valori vengono divisi per due l'immagine risultante sarà indistinguibile dall'originale).

A questo punto è la fase dell' Entropy Encoder. Infatti i coefficienti del blocco 8×8 ottenuti applicando DCT e quantizzazione vengono codificati con una tecnica di tipo RLE (Run Length Encoding) in cui più alcuna perdita di informazione, grazie alla quale si riescono a rappresentare in modo efficiente le lunghe sequenze di zeri che si ottengono inevitabilmente con la quantizzazione. Ma noi sappiamo che dato un insieme di simboli, l'entropia H è definita come l'informazione media di ogni simbolo, ed è una buona misura delle prestazioni di una codifica.

JPEG usa due tecniche per l'entropy coding, quella di Huffman e quelle aritmetiche.

Attraverso questo procedimento si passa da una immagine JPEG sorgente ad una compressa.

Da Compressa a Originale: E' il processo inverso.

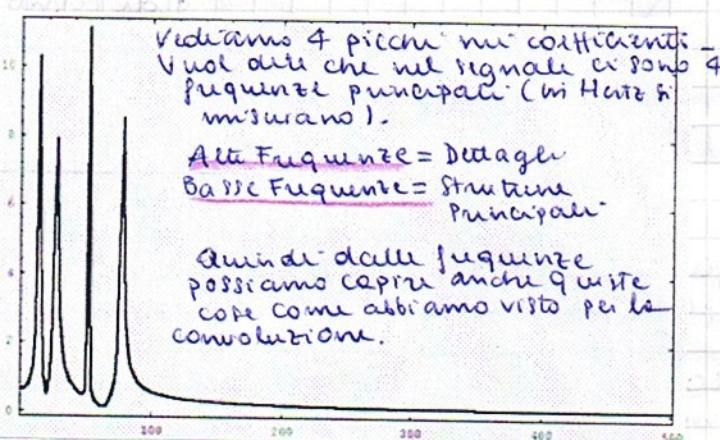
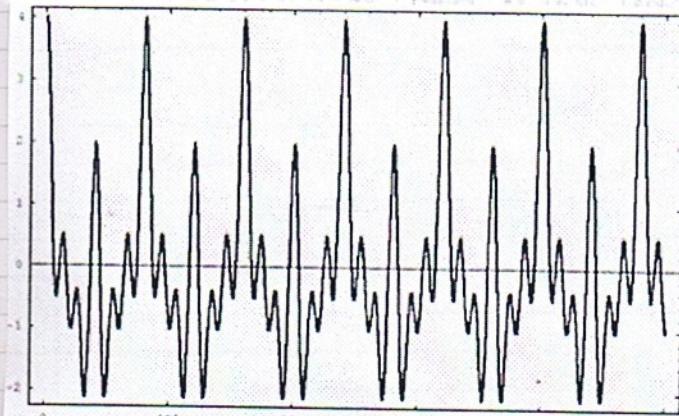
Si inizia con la fase di Entropy Decoder (primo in Encoder).

Poi si applica la dequantizzazione (sempre con delle tavole opportune), e infine si applica la trasformazione inversa discreta Coseno (IDCT) e così si ottiene

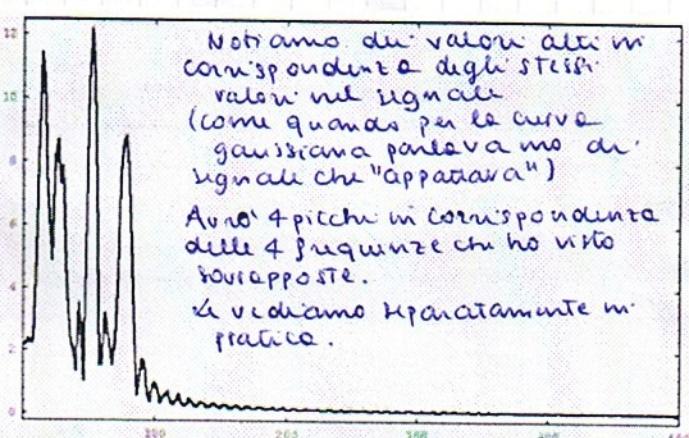
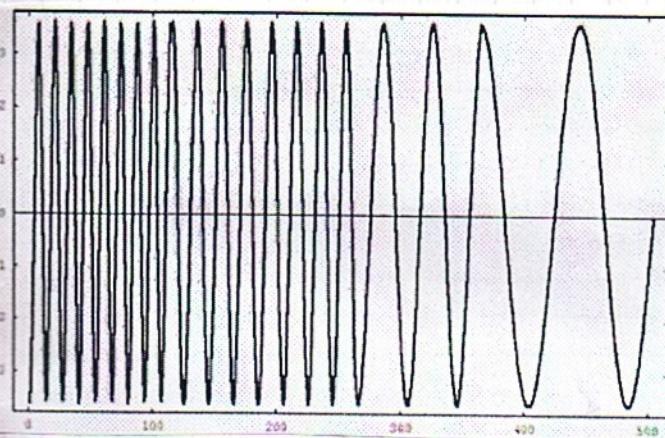
l'immagine ricostruita.

Ma vediamo alcuni esempi di DCT:

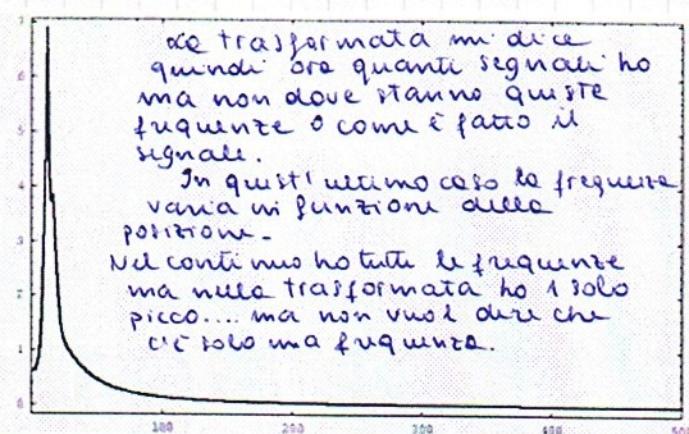
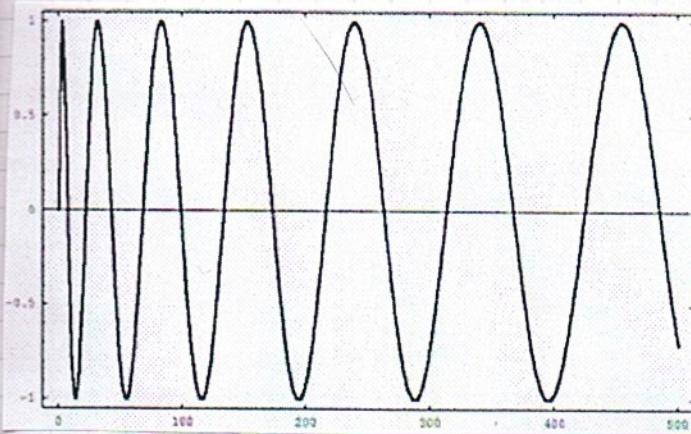
- Un segnale stationario e le sue trasformata di Fourier



- Un segnale composto e le sue trasformata di Fourier



- Un segnale non stazionario e le sue trasformata di Fourier



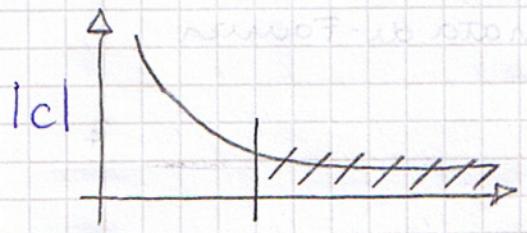
→ da originale a compressa quindi applico FDCT e fino a questo momento si può dire che ci ho perso. Ma applicando la quantizzazione secondo le opportunità della quantizzazione torniamo a valori interi. Con un processo di Entropy Encoder dividiamo i coefficienti meno significativi (sempre secondo una tabella specifica).

In pratica perdiamo dettagli poiché trasmettiamo meno coefficienti.

Se trasmettiamo tutti e 64 i coefficienti ricostruiamo il segnale ma non avremo compressamente.

Se ricostruiamo il segnale introduciamo quindi un errore, e meno coefficienti usiamo, più dettagli perdiamo.

Graficamente potremmo immaginare come un grefie fatto in questa maniera:



Se guardo l'ampiezza di $|I_{cl}|$ (ampiezza dei coefficienti), vediamo che per le alte frequenze abbiamo alti valori, poi tendono a zero.

Per comprimerli introduciamo quindi un valore di soglia e li tagliamo in modo da trasmettere solo alcuni coefficienti e gli altri considerarli nulli.

Tagliamo tutte le altre frequenze.

Se sposto la soglia verso sinistra avrò un segnale minore.

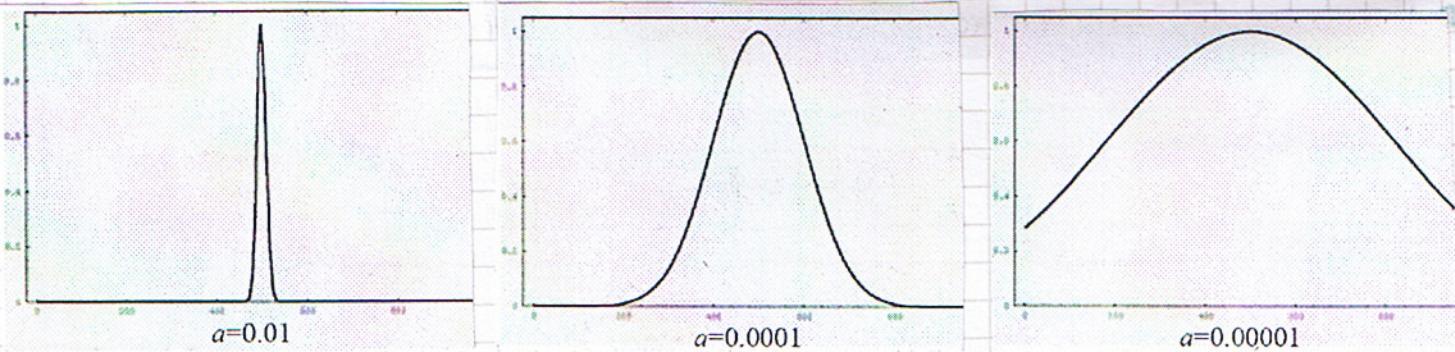
La compressione JPEG viene effettuata fondamentalmente in 2 modi:

- Per numero di coefficienti (come sopra)
- Per qualità (Photoshop)

In conclusione, la DCT è una trasformazione bidimensionale che consiste nell'applicazione di un operatore lineare, quindi invertibile, al segnale da elaborare in modo da renderlo il più possibile diagonale. La sua matrice di autocorrelazione è quindi concentrare l'informazione su un minor numero di coefficienti (in sostanza, si fanno comparire meno zeri nelle codifiche).

La DCT tipicamente viene implementata su blocchi di 8×8 pixels (quello vuol dire che l'immagine originale deve essere suddivisa in una griglia di piccole sottoimmagini quadrate di 64 elementi).

STFT (Short Time Fourier Transform)



La trasformata di Fourier a tempo breve fornisce uno insieme complesso di pesi delle diverse frequenze nel segnale in esame, in tutta la sua durata.

Per i segnali non stazionari (nella figura DCT in cui ad un solo picco corrispondevano più frequenze - problema -) occorre uscire nella trasformazione, una dipendenza del tempo; il modo più immediato consiste nel rendere locale la trasformata di Fourier non operando su tutto il supporto del segnale (quindi non più da $-\infty$ a $+\infty$) ma su porzioni di esso (intervalli).

Per questo si chiama "Short Time".

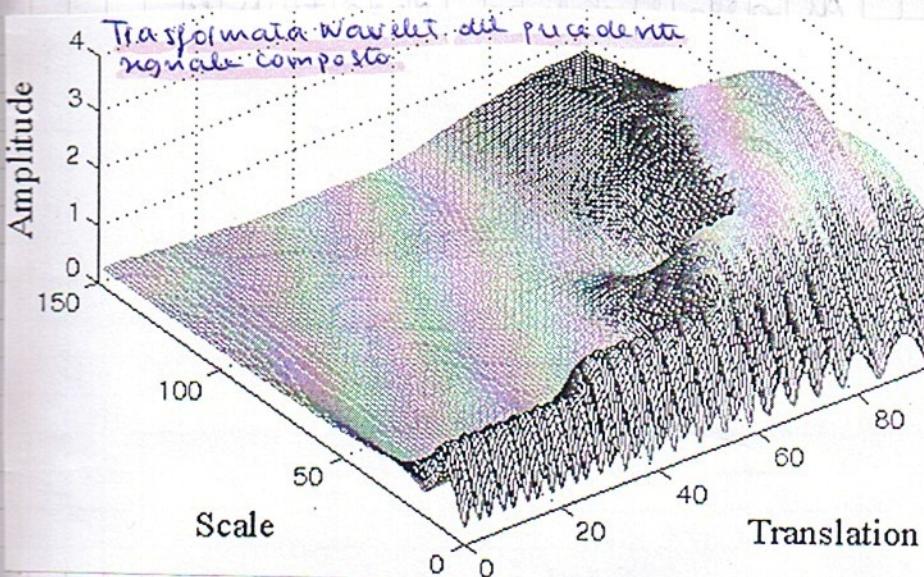
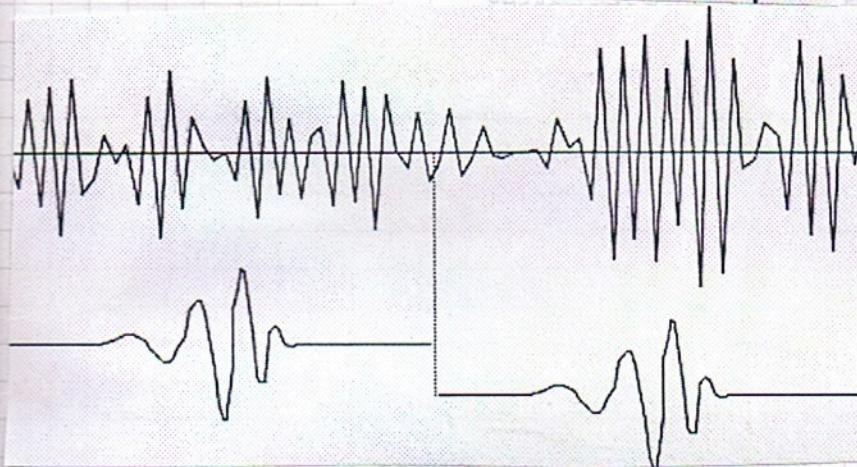
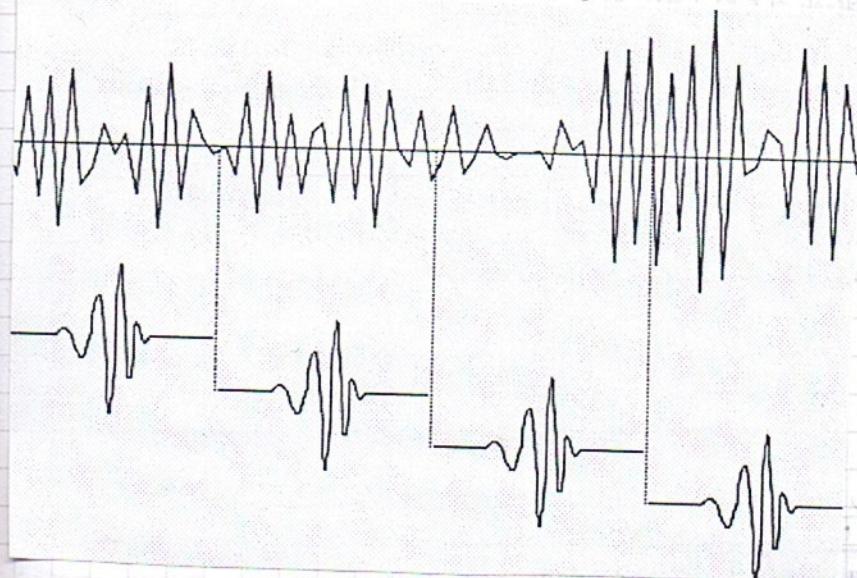
Nella STFT, anziché convolare con la curva del careno, convolviamo con la gaussiana (detta anche curva di Gabor). La a delle immagini rappresenta lo sigma della gaussiana.

Se vogliamo una frequenza più bassa attribuiamo un valore più piccolo alle a, viceversa, altri valori \rightarrow alte frequenze.

Agendo localmente con le STFT possiamo conoscere esattamente

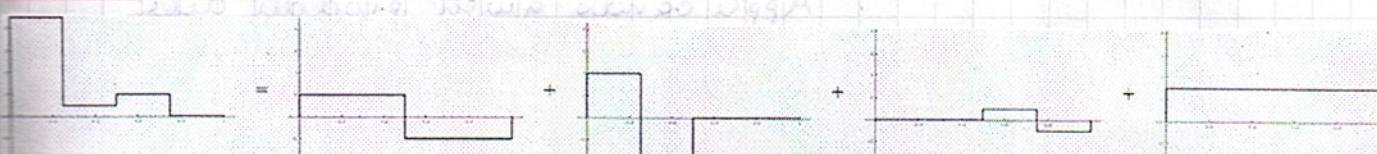
l'ascissa sulla quale agisce la frequenza, mentre al di fuori di questo intervallo la gaussiana assumeva valore nullo.

TRASFORMATA DISCRETA WAVELET (DWT)



La più semplice trasformata Wavelet è la trasformata di Haar.

HAAR



La trasformata Wavelet rappresenta una valida alternativa alla trasformata coseno.

Il motivo basilare è che, applicando la DWT ad una immagine, moltissimi coefficienti sono prossimi a zero, e qui non i quantizzatori a precisione variabile operano nel loro ambiente ideale.

Come possiamo vedere nelle immagini, abbiamo il segnale di ingresso in alto e sotto di esso troviamo le cosiddette wavelet madre, che sono delle gaussiane che possiamo immaginare (per usare un paragone) come i kernel nel caso delle trasformate.

Effettuiamo quindi la convoluzione ottenendo dei picchi le cui posizioni corrispondono (come visto per le gaussiane).

Quindi il più grande vantaggio delle DWT è che non solo mide la frequenza ma anche dove è localizzata.

E' un processo di convoluzione che ha come output un segnale tridimensionale (vedi figura) e prende il meglio dei entrambi metodi visti in precedenza.

Abbiamo parlato di trasformate Wavelet, ma in realtà sarebbe stato più corretto parlare di "famiglia" di trasformate, in quanto ne esistono numerosi tipi che rispondono a precise caratteristiche come la continuità, le simmetrie, etc.

da Wavelet principale (la cosiddetta "madre" di tutte le wavelet) è la cosiddetta Haar Wavelet (trasformata di Haar) che produce risultati ricorrenti con insiemisimili come le immagini.

Come possiamo vedere nell'immagine, abbiamo che il segnale viene moltiplicato (1a figura) e dato dalla somma delle trasformate di Haar (modificano le ampiezze e coseno) e dalla portante (ultima immagine).

Osserviamo inoltre che mentre prima il segnale si propagava, ora rimane locale.

Prendiamo le Wavelet Madre della trasformata di Haar

In pratica il nostro scopo è trovare i coefficienti tali che, moltiplicandoli per il segnale (dato dalla trasformata) ottieniamo il segnale di origine.

Questi coefficienti vengono trovati algebricamente attraverso il metodo delle semisomme e semi-differenze.

Vediamolo nello specifico.

Esempio: Prendiamo i valori del segnale f e facciamo sommazioni e semidifferenze.

$$\begin{array}{cccc} 9 & 1 & 2 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ -4 & 1 & - & \\ +5 & & 1 & + \\ \downarrow & \downarrow & \downarrow & \downarrow \\ -2 & & 3 & \\ + & & & \end{array}$$

Ad ogni passo continuo le semisomme e semidifferenze a partire dalle semisomme del passo precedente.

Al passo finale prenderò tutte le semidifferenze e l'ultima semisomma, quindi:

$$\rightarrow 4 \ 1 \ 2 \ 3$$

L'ultima semisomma rappresenta il coefficiente da dare alla portante (quindi 3).

L'ultima semidifferenza è l'ampiezza della wavelet.

Se otteniamo valori negativi per i coefficienti avevamo \lceil antichi

Da queste due immagini, possiamo osservare lo stesso procedimento con operazioni matriciali.

Applicando questi prodotti alle matrici ottieniamo gli stessi coefficienti della trasformata di Haar.

(98) Esempio di scomposizione matriciale della Trasformata di Haar

$$\frac{1}{2} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} 9 \\ 1 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ 1 \\ 1 \end{pmatrix} \text{ is a high pass filter}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 4 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ 4 \\ 1 \end{pmatrix} \text{ is a permutation filter}$$

$$\frac{1}{2} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 5 \\ 1 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 4 \\ 1 \end{pmatrix} \text{ is a low pass filter}$$

$$\frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 \end{pmatrix} \begin{pmatrix} 9 \\ 1 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 4 \\ 1 \end{pmatrix}$$

The Haar matrix multiplication

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 9 \\ 1 \\ 2 \\ 0 \end{pmatrix}$$

Esempio di scomposizione matriciale della trasformata di Haar

In questo modo però lavoriamo con matrici sparse e non dense (molti zero, vedi prima immagine) ma possiamo comprimere ridefinendo il prodotto matriciale (come mostra le seconde immagini)

I 3 passaggi dell'immagine precedente diventano un unico prodotto matriciale \rightarrow Ampliata trasformata

$$W = \log_2 (\text{lunghezza del segnale})$$

FILTER BANK

Ponendo le semi somme come filtri passa-basso e le semi differenze come filtri passa alto otteniamo quello che dagli ingegneri è definito "Filter Bank"

Come si evince dalle figure il segnale di ingresso viene scomposto nei due filtri ottenendo con il filtro passa alto h i coefficienti wavelet w_1 , e con il filtro passa basso g i coefficienti c_1 che sono la versione smussata del segnale di ingresso (precedenti)

DECOMPOSIZIONE STANDARD

Osserviamo ora come poter applicare Haar nel caso bidimensionale.

Secondo la decomposizione standard (immagine qui a fianco) data l'immagine in input applico Haar per

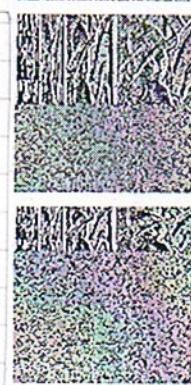
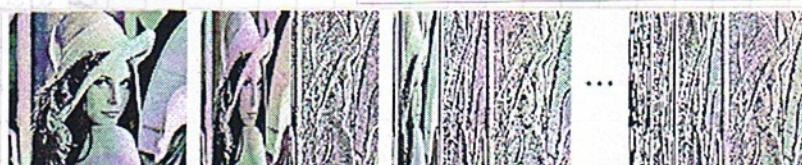
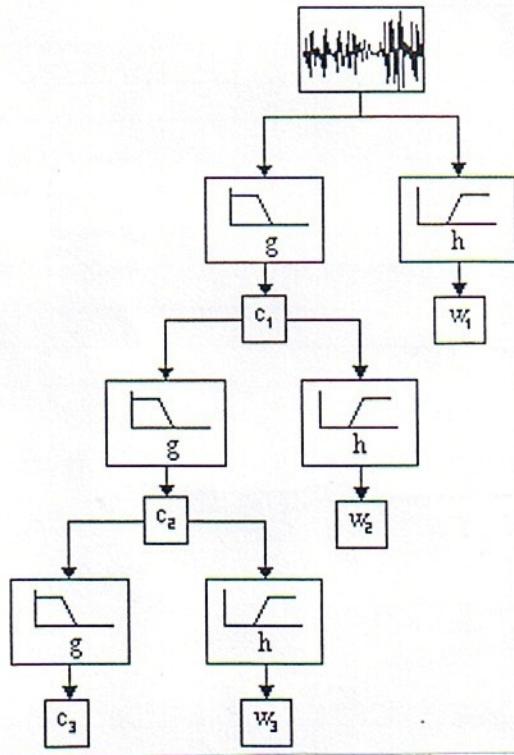
sulle righe ottenendo nella prima metà la versione smussata dell'immagine mentre nell'altra i coefficienti wavelet.

Dopo che ripeto il procedimento per le righe lo applico anche per le colonne

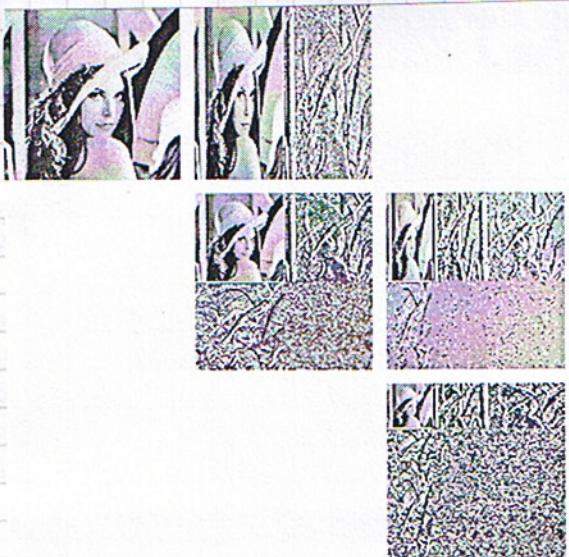
non

il caso di decomposizione standard invece il procedimento è visibile dall'immagine nella pagina successiva

partendo dalla riga \rightarrow



DECOMPOSIZIONE NON STANDARD



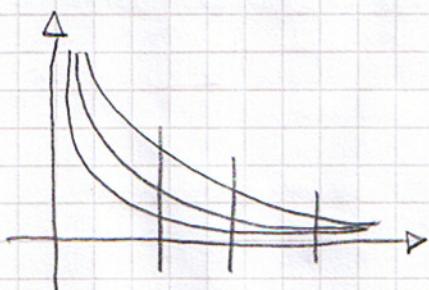
In questo caso invece attorno continuamente righe e colonne.

Quindi ho valori e coefficienti wavelet per ricostruire l'immagine

Si osserva che a puntate oh taglio perdiamo meno informationi con la decomposizione non standard, cioè comprimiamo di più perdendo meno informationi



Nota = In base al segnale viene scelta una wavelet o un'altra in modo da far andare i coefficienti più vicinamente a zero



In questo modo riesco a comprimere di più perdendo meno informationi

Quindi in base al segnale sceglio la wavelet madre adeguata

LEZIONE 8 À TROUS

$$c_i(k) = \frac{1}{2^i} \left\langle f(x), \phi\left(\frac{x-k}{2^i}\right) \right\rangle = \sum_{n=-\infty}^{+\infty} h\left(\frac{n-k}{2^{i-1}}\right) c_{i-1}(n)$$

$$c_0(k) = c_p(k) + \sum_{i=1}^p w_i(k)$$

$$h(n) = \frac{1}{2^{s+1}} \begin{pmatrix} s+1-n \\ 2 \\ s+1 \end{pmatrix}$$

$$\left. \begin{array}{l} w_i(k) = \sum_{n=-\infty}^{+\infty} g(n) c_{i-1}(k+n2^{i-1}) \\ w_i(k) = c_{i-1}(k) - c_i(k) \end{array} \right\}$$

$$\phi(x,y) = \phi(x)\phi(y)$$

E' l'altro algoritmo riguardante le wavelet.
Attualmente A Trous significa a buchi; vediamo il perché; dal modo di operare.

Come visto prima i coefficienti rappresentano il filtro passa alto mentre la versione smussata del segnale è il filtro passa basso (Filter

versione smussata del segnale di ingresso

Bonk), nel caso dell'algoritmo A Trous, facendo la convolution tra l'immagine di ingresso e il kernel

$$I \otimes k_1 \rightarrow I_1$$

(versione smussata del filtro di puntate (passa basso))

Per trovarne quello passa alto facciamo $I - I_1$ (metto in evidenza i dettagli).

Sull'immagine I_1 applichiamo K_1 (il primo kernel) e otteniamo una versione smussata (K) etc e ripetiamo ricorsivamente ottenendo i coefficienti Wavelet un certo numero di volte.

Alla fine mettiamo insieme l'ultima versione smussata del segnale e i coefficienti wavelet dati dal filtro passa basso e otteniamo i coefficienti della trasformata A trous dell'immagine.

Inoltre con questo metodo otteniamo 1 numero maggiore di coefficienti, ad esempio se considero un segnale con 32 valori dei coefficienti, con la trasformata di Haar ottengo 32 valori dei coefficienti Wavelet.

Con A trous ne ottengo al 1° passo 32 sia per il passa-alto che per il passa-basso, la stessa cosa al 2° passo etc.

Il kernel che utilizziamo inizialmente è

116	118	116
118	114	118
116	118	116

(Kernel 3×3 per interpretazione lineare)

Ma, ci chiediamo, dove sono i "buchi" se uso il kernel K_1 ?

In realtà, a differenza del metodo Haar, con il quale il segnale (l'immagine) diventava via via più piccole, con il metodo A trous invece, non siamo a mantenere l'immagine della stessa grandezza poiché ad allargarsi è il kernel ottenendo così maggiori dettagli.

Ma dovendo ingrandire il kernel, dobbiamo per forza introdurre dei buchi.

Vediamo il kernel migrando K_2 (dove 2 è il raggio) di dimensione 5×5 :

116	0	118	0	116
0	0	0	0	0
118	0	114	0	118
0	0	0	0	0
116	0	118	0	116

Potrei provare ad allargare il kernel raddoppiandomi volta per volta il raggio (questo metodo è detto trasformazione diadice).

Più migrando di sotto i kernel più abbasso le frequenze (ottengo strutture più grandi).

Un piccolo esempio di procedimento al raddoppio è:

Raggio kernel (Poteremo anche non andare al raddoppio e avere tutti i raggi)

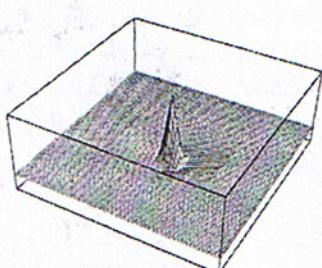
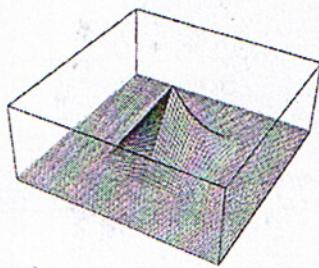
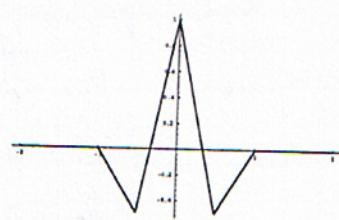
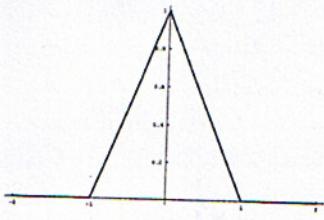
1	3×3
2	5×5
4	9×9
8	17×17
16	33×33

Così è ovvio, aumentando la grandezza del kernel, aumenta la complessità della convoluzione, e quindi si risolve questo problema.

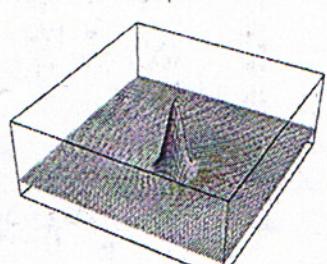
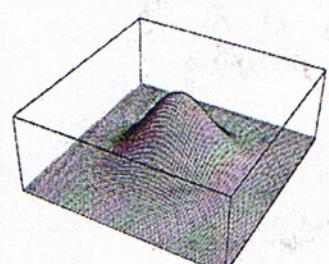
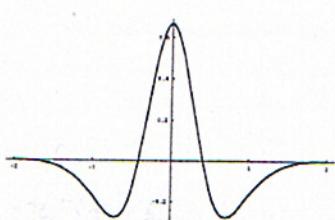
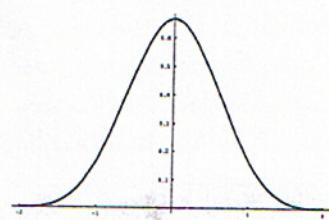
Ma possiamo notare subito, che i buchi vengono introdotti con degli zeri, e che questi sono sistemati nel kernel in modi e posizioni opportune.

Possiamo quindi eseguire un metodo per evitare di effettuare le moltiplicazioni degli zeri, riuscendo a mantenere costante la velocità dell'algoritmo, che sarà quindi equivalente alla velocità per un kernel 3×3 .

$$B_1(x) = \frac{1}{2}(|x-1| - 2|x| + |x+1|)$$



$$B_3(x) = \frac{1}{12}(|x-2|^3 - 4|x-1|^3 + 6|x|^3 - 4|x+1|^3 + |x+2|^3)$$



B₁-Spline scaling function ϕ and wavelet ψ

B₃-Spline scaling function ϕ and wavelet ψ

Poiché sia le $B_1(x)$ e le $B_3(x)$ sono a variabili separabili, invece di effettuare la convoluzione con un kernel 3×3 , le posso effettuare con un kernel 1×3 prima orizzontalmente e poi verticalmente, ottenendo risultati più veloci.

B_1 e B_3 sono funzioni rispettivamente bilineare e bicubica, in cui possiamo notare a sinistra la scala e a destra le wavelet.

La parte negativa di entrambi i grafici serve per annullare la luminosità (ad esempio per sopprimere l'effetto dei pallini bianchi con anello nero nelle foto, ad esempio la volta celeste).

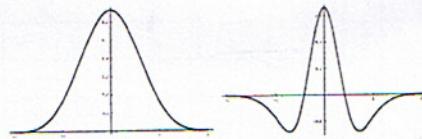
Vediamo alcuni esempi di funzioni di scala e rispettive wavelet:



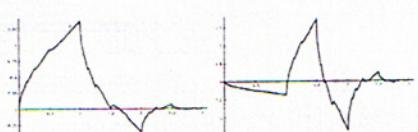
Haar scaling function ϕ and wavelet ψ .



B₁-spline scaling function ϕ and wavelet ψ .



B₃-spline scaling function ϕ and wavelet ψ .



Daubechies scaling function ϕ and wavelet ψ .



Meyer scaling function ϕ and wavelet ψ .



Battle-Lemarié scaling function ϕ and wavelet ψ .

JPEG VS JPEG2000

Il successo dello standard di compressione JPEG si puo' assumere innanzitutto dalle cifre: è stato calcolato infatti che l'80% circa delle immagini presenti su Internet siano dei file JPEG.

Tuttavia esiste già il successore di questo "fortunato" standard, ed è appunto il JPEG2000.

Ma come funziona precisamente JPEG2000?

E quali sono le principali differenze rispetto al "fratello maggiore" JPEG?

Vediamole.

Le differenze sono numerose e la principale è senz'altro il cambiamento dello strumento matematico alla base dell'algoritmo.

di compressione.
mentre, infatti, JPEG usa la trasformata DCT, JPEG 2000 usa la DWT, quindi all'immagine da input è applicata la trasformata wavelet -
la stessa parola "wavelet" infatti, che significa "piccole onde", identifica un sistema matematico che consente di superare il difetto principale delle immagini tratte secondo il normale standard JPEG, cioè la presenza di:

JPG (In evidenza gli artefatti a blocchi)



JPG (bpp = 0.2; MSE = 320; PSNR = 23.1dB)

JPG2000



JPG2K (bpp = 0.2; MSE = 113; PSNR = 27.6dB)

qui blocchi: quadrettati che sono il prezzo da pagare per la maggior compressione ottenuta -

Con l'uso della DWT invece, il contenuto del file originale non viene più suddiviso in blocchi de 8x8 pixel, ma è l'immagine nella sua totalità ad essere analizzata e successivamente scomposta, per ottenere alla fine un file compresso dove l'eventuale degrado dell'immagine è significativamente inferiore a quello ottentibile, a parità di compressione, con il JPEG tradizionale.

Vedremo quindi un "degrado" che si manifesta ora non più attraverso i famosi blocchi quadrettati (immagine a sinistra), ma con un aspetto più o meno sfocato dei contorni degli oggetti presenti nell'immagine -

Nelle nostra immagini di esempio codifichiamo con bpp = 0,2 cioè ogni 2 bit codifichiamo 10 pixel -

Come è evidente, e come possiamo vedere da MSE e PSNR, a parità di compressione l'errore si è meno che dimezzato, è praticamente un tetto.

Possiamo dire allo stesso modo che, a parità di "qualità" ed errore tra i due formati → con JPEG2000 (e quindi con le wavelet)

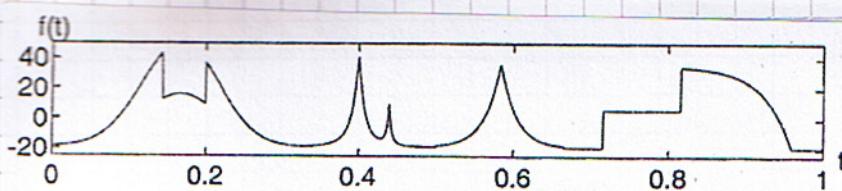
Comprimiamo MOLTO di più -

(In realtà però il formato JPEG2000 è ancora in sperimentazione quindi non esiste nei nostri computer)

WT (TRASFORMATA WAVELET)

Concludiamo adesso il discorso sulle wavelet vedendo degli esempi di segnale codificati con coefficienti di wavelet e formie.





→ B₃-Spline wavelet approximation

→ SEGNALE D'INGRESSO

Nell'immagine qui a fianco osserviamo le corrispondenze del segnalet d'ingresso con i coefficienti wavelet.

Come è visibile, all'aumentare della frequenza avremo più coefficienti per costruire meglio il segnale.

In questa prima fase, i coefficienti vengono posti in modo equidistante.

Vediamo adesso lo stesso segnale di partenza ricostituito nell'ordine primi con Fourier Transform e poi con Wavelet.

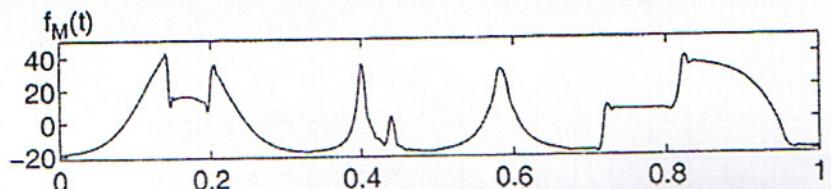
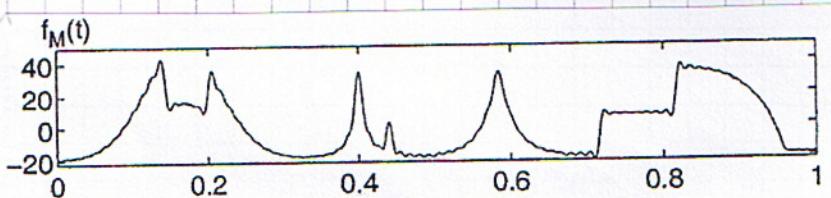
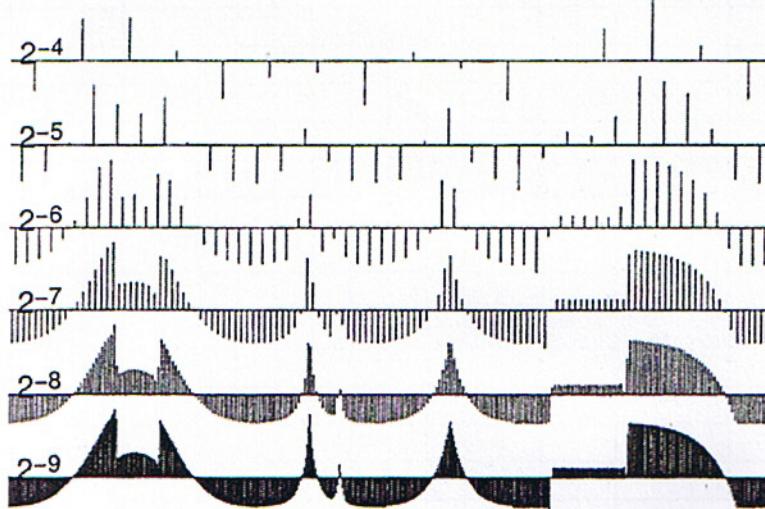
Possiamo subito osservare che, a parità del numero di coefficienti usati per ricostituire il segnale, mentre in Fourier abbiamo più oscillazioni (in quanto servono più cosini e seni) il segnale ricostituito con wavelet è più simile a quello di partenza.

Ma, avendo coefficienti equidistanti ci porta ad avere molti coefficienti e anche in punti di continuità (che non coincidono), potremmo quindi migliorare il modo di scegliere questi coefficienti (come possiamo vedere delle figure accanto), ponendone di più nei punti relativi alle discontinuità.

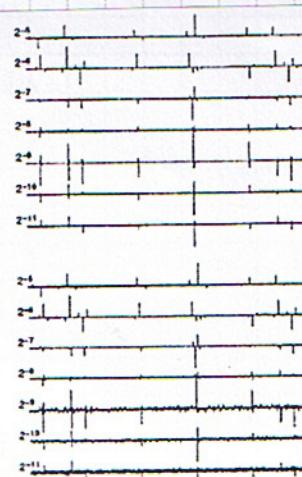
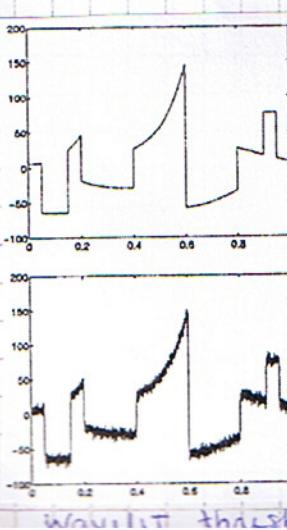
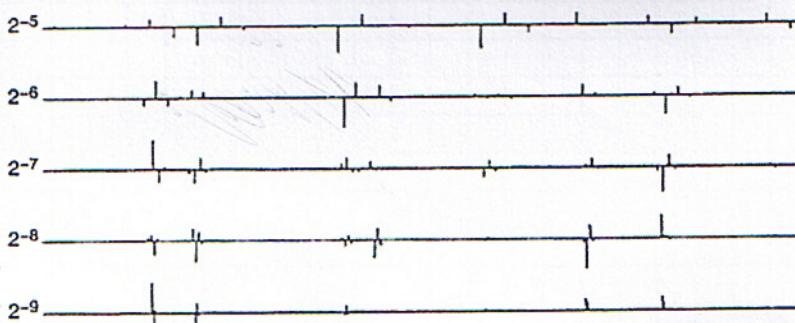
In questo modo otterremo segnalet molto più puliti.

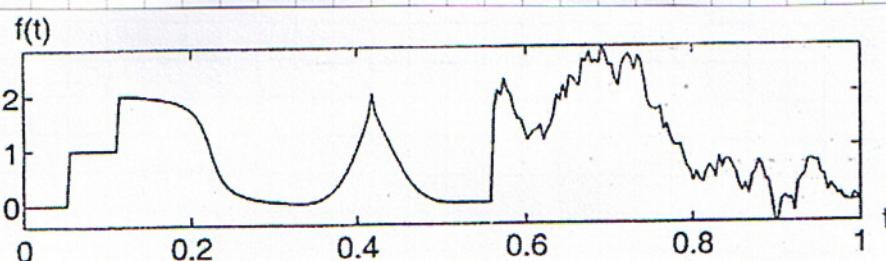
Si deduce che le wavelet ci permettono anche di individuare le discontinuità nei segnali.

→ Esempi di segnali con relativi coefficienti wavelet



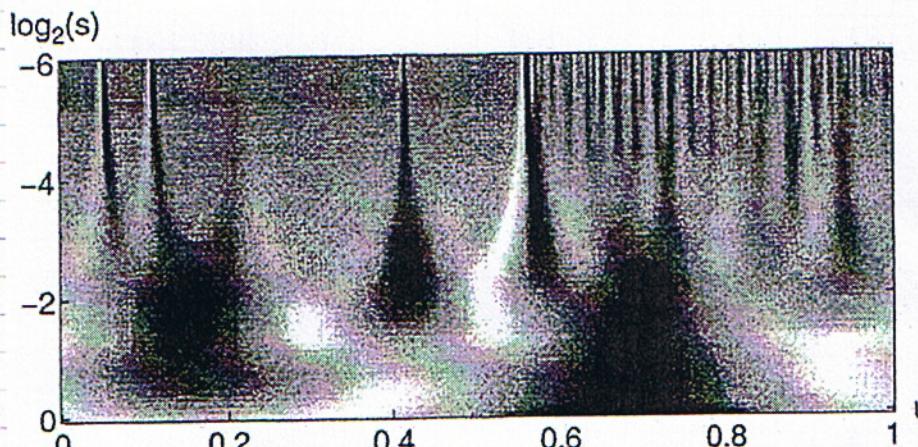
(FT and WT linear approximation with 0.15N coefficients)





→ SCALOGRAMMA

Le due immagini qui al lato corrispondono allo stesso segnale.



La figura in basso mi è la rappresentazione grafica in cui viene assegnato un colore alle varie frequenze.

Ad esempio, il nero indica una bassa frequenza, il bianco una alta e il grigio una media.
Un cuspidone sarà una struttura tutta nera.

(Scalogramma (altro modo di rappresentare il wanda)) Vengono messe in evidenza sia le discontinuità che il loro tipo.

A ciascuna discontinuità corrisponde un canale colorato opportunamente. Si possono anche evincere "a occhio" posizione e grandezza del segnali.

FORMATO BITMAP

L'estensione di un file formato bitmap è ".bmp" e si tratta di un formato non compresso, ma che permette di essere visualizzato.

(de chi possiede windows) senza l'uso di programmi particolari. Il formato supporta i metodi di colore RGB, scala di colore, scala di grigio e Bitmap, ma non supporta i canali alpha (per la trasparenza).

Il bitmap è un formato rawdata (senza perdita di informazioni). Ricordiamo brevemente che questo formato (rawdata) funziona con un solo numero di pixel, non fornisce le dimensioni in pixel dell'immagine, quanti bpp e cosa rappresenta in valore numerico.

I formati bitmap variano molto tra di sé, ma condividono la stessa struttura generale.

Un file bitmap è organizzato secondo componenti base e componenti specifiche. Le componenti base sono presenti in tutti i formati di tipo bitmap, mentre le componenti specifiche dipendono dalla complessità del formato.

Troviamo una descrizione completa di come è codificata una bitmap nelle slide.