19-01-2023

LO STANDARD JPEG

L'algoritmo di compressione JPEG ha 8 passaggi. Essi richiamano tutti i concetti visti nel corso IEM. E' di tipo **LOSSY**, ovvero che quando si torna indietro (fase di **DECOMPRESSIONE**) non si ha più l'immagine di partenza.
In questo tipo di compressione la perdita è **CONTROLLATA**.

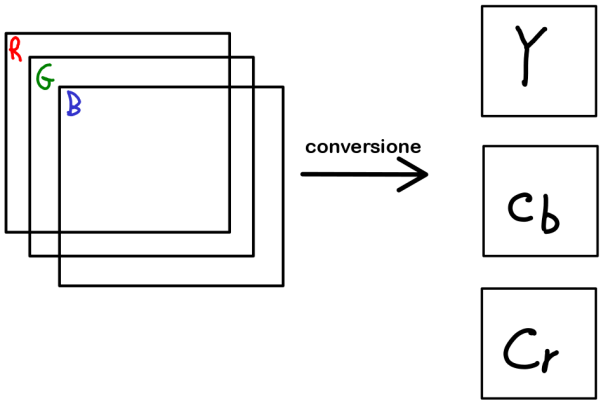
PASSI FONDAMENTALI DELLA CODIFICA JPEG

I passi fondamentali dell'algoritmo di compressione JPEG sono i seguenti:

- Pre-processing:
  1. **Conversione** dello spazio di colore ($RGB \rightarrow YC_bC_r$);
  2. **Sottocampionamento** della crominanza
  3. Partizione dell'immagine in **sottoimmagini** (8x8)
- Trasformazione:
  4. Discrete Cosine Transform (**DCT**);
  5. **Quantizzazione**;
- Codifica:
  6. Codifica dei coefficienti **DC**;
  7. Ordinamento Zig-zag dei coefficienti **AC**;
  8. **Huffman** (Entropy Coding)

Conversione spazio di colore

In questo passo si deve convertire lo spazio di colore da RGB a YC_bC_r suddividendo, quindi la luminanza (Y) dalla crominanza (C_bC_r).



- Questa operazione di passaggio è **REVERSIBILE**, quindi qui NON si ha alcuna perdita.

Preprocessing (i): da RGB a Y C_b C_r

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

La Y è la luminanza, gli altri due canali codificano i colori.

Si tratta di una trasformazione reversibile.

Il modello Y C_b e C_r è un modello di colori che permette di avvantaggiarsi della “debolezza” del sistema visivo umano.

Infatti…

-

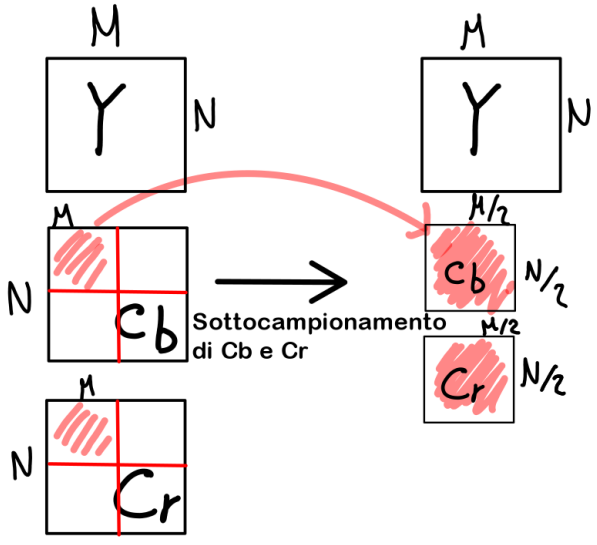
> *La prima riga corrisponde alla Y. $YCbCr$ ha tutti i numeri positivi (a differenza di YUV) e per tale motivo è stato scelto questo spazio di colore.*

In particolare le formule di conversione sono:

- $Y = 0.299R + 0.587G + 0.114B$
- $C_b = 0.596R + -0.275G + -0.321B$
- $C_r = 0.212R + -0.523G + 0.311B$

Sottocampionamento della crominanza

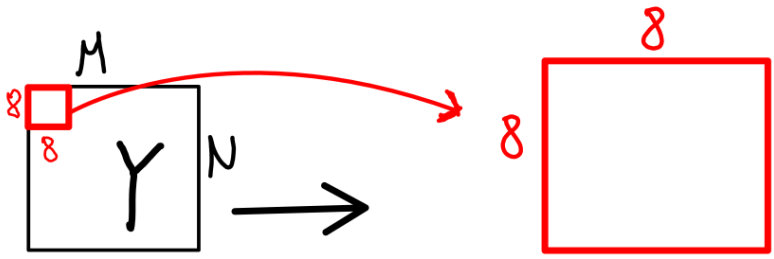
- La matrice di Y rimane **INVARIATA**.
- Le dimensioni di C_b e C_r vengono **DIMEZZATE** perchè la crominanza rappresenta un'informazione meno importante per l'occhio umano



> *Questo passo è **IRREVERSIBILE**. Quando si decomprime, quindi si ingrandiscono le matrici, sicuramente farò una* replication *e non avrò più i valori originali.*

Partizione dell'immagine in sottoimmagini

Suddivido l'immagine che ottengo (cioè le matrici di Y, di Cb e Cr) in **blocchetti** senza sovrapposizioni e ogni blocchetto(finestra) ha **dimensioni 8x8 (64 pixel totali)**.



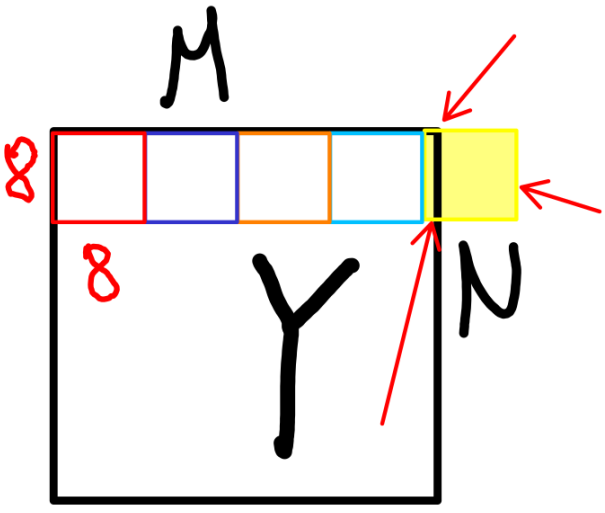
> *L'operazione è uguale per i 3 canali dello spazio di colore e ne verrà mostrata solo una.*

Quindi ora considero la finestra 8x8.

> *Ciò che devo fare per una finestra(blocco) vale per tutti i blocchi creati.*

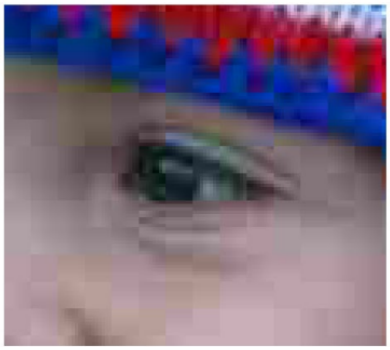
Possibili problematiche

- La dimensione dell'immagine può comportare un blocco 8x8 fatto da meno di 64 pixel, quindi:
  - O riempio di zeri i posti mancanti (quindi i bordi)
  - oppure uso immagine che abbia dimensioni multiple di 8.



- **Non si tiene mai conto di cosa succede nel blocco adiacente**.
  - Quindi stavolta non si fa (a differenza degli operatori locali che guardano l'intorno di un punto) e questo è **IL PUNTO DEBOLE DI JPEG**
  - Infatti i blocchi **si vedono sconnessi** l'uno dall'altro se si ingrandisce l'immagine perchè non si valutano quelli vicini. Il $JPEG2000$ non fa questa divisione in blocchi 8x8

ma considera tutta l'immagine



> Nella compressione JPEG questo è un **difetto minimo** che si può mantenere.
> *Visto che le operazioni successive valgono sia per Y che per Cb e Cr ne descriviamo solo uno di dimensioni 8x8.*

- Quindi si hanno 64 pixel all'interno del blocco 8x8 all'interno del quale tutti i numeri sono positivi perchè si è scelto proprio YCbCr come spazio di colore. La scelta è dovuta al fatto che i numeri negativi verranno "creati" in modo controllato e non in modo "casuale".

- Si prendono tutti i numeri presenti nella matrice e si **sottrae 128**.

- Si ottengono numeri positivi (se gli originali erano >128) e negativi (se gli originali erano <128)

> Con quest'operazione si sta spostando **l'origine dell'immagine** che prima era $[0, 255]$. ottengo numeri fra $[-127, 128]$ e lo 0 si trova al **centro**.

- In generale quindi si sottrae 2^{n-1}.

- Se si lavora con 8 bit, è $-(2^7)$

Esempio:

$$\begin{matrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 66 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{matrix}$$

$$\begin{matrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -62 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{matrix}$$

Note

- Nella figura soprastante ci sono tanti numeri negativi.
- Il blocco che ottengo è quello a destra.

Discrete Cosine Transform (DCT)

Al blocco risultante applico la **Trasformata Discreta del Coseno(DCT)** quindi si passa da un **dominio spaziale** ad un **dominio delle frequenze**.

> *La DCT è simile a quella di Fourier: prendo i 64 pixel e tutti servono per fare ognuno degli 64 valori in output.*
> *Dopo tale operazione avrò ancora una matrice 8x8 dove, ogni posizione, è stata calcolata prendendo in on considerazione tutti i valori di input.*

La formula della **DCT** è:

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

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

where:

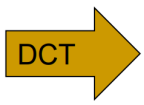
$$C(u)=\frac{1}{\sqrt{2}}\quad \text{for } u=0;\ C(u)=1\quad \text{otherwise}$$

$$C(v)=\frac{1}{\sqrt{2}}\quad \text{for } v=0;\ C(v)=1\quad \text{otherwise}$$

dove:

- f = immagine nel dominio spaziale;
- F = immagine in dominio delle frequenze;
- Si fa su **immagini quadrate**, infatti N=8.
  - L'antitrasformata è f(x,y) =2/N ... ecc.. (cioè la seconda formula)
  Visto che non ci sono numeri complessi come nella Trasformata di Fourier, il risultato è **un numero reale**.

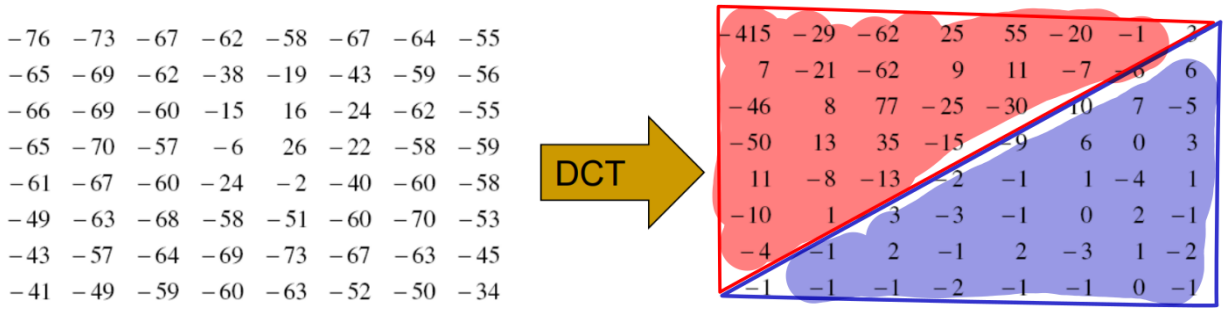
−76	−73	−67	−62	−58	−67	−64	−55
−65	−69	−62	−38	−19	−43	−59	−56
−66	−69	−60	−15	16	−24	−62	−55
−65	−70	−57	−6	26	−22	−58	−59
−61	−67	−60	−24	−2	−40	−60	−58
−49	−63	−68	−58	−51	−60	−70	−53
−43	−57	−64	−69	−73	−67	−63	−45
−41	−49	−59	−60	−63	−52	−50	−34



−415	−29	−62	25	55	−20	−1	3
7	−21	−62	9	11	−7	−6	6
−46	8	77	−25	−30	10	7	−5
−50	13	35	−15	−9	6	0	3
11	−8	−13	−2	−1	1	−4	1
−10	1	3	−3	−1	0	2	−1
−4	−1	2	−1	2	−3	1	−2
−1	−1	−1	−2	−1	−1	0	−1

Note e funzionamento DCT

- Vedendo la matrice che crea la Trasformata DCT si nota subito che la matrice risultante è "divisa" in due matrici triangolari:
  -

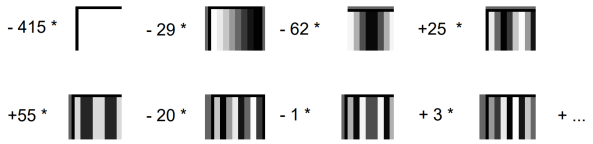


  - Quella che contiene numeri "più alti in valore assoluto" (matrice rossa) e quella che contiene numeri "più bassi in valore assoluto" (matrice blu).

> *Perchè il triangolo superiore è fatto cosi?*
> Il blocco superiore è come se fosse con una **base** (non canonica):
> -415 che moltiplica una base. -29 che moltiplica una base ecc...

- La BASE è grande **8x8**. Ogni base (*che non so com'è fatta*) la chiamo **base della DCT**. Ripeto tale procedimento per tutte le posizioni, quindi moltiplicando le basi per i rispettivi coefficienti
  - Di conseguenza ho 64 coefficienti che moltiplicano 64 basi (che sono 8x8).
  - E **sommando** queste 64 moltiplicazioni **ottengo la matrice di prima** (immagine soprastante a sinistra (*prima della DCT*)
- **In questo modo è possibile tornare nel dominio spaziale dal dominio delle frequenze.**

> Quindi i coefficienti che si ottengono dalla DCT sono da moltiplicare per una base (già nota) **per poter tornare indietro.**

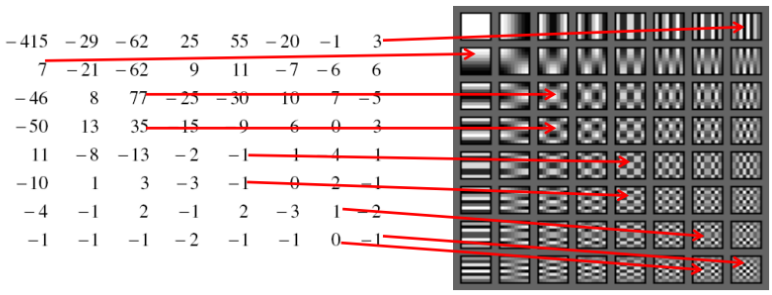


Nell'immagine soprastante ogni "*quadrato colorato*" è una base:

- si sta lavorando su delle **righe verticali** (se si guardano le foto delle basi) quindi su colonne.

> Se il **coefficiente che abbino è BASSO** il numero allora ci sono **POCHE** irregolarità

Lo stesso procedimento va ripetuto per tutti i coefficienti risultanti dalla DCT nel modo seguente:



Si nota che nella 2° riga ci sono irregolarità orizzontali

> Le basi della DCT (Che sono matrici di numeri) ognuno dei quadratini sono 8x8 rappresenta una possibile regolarità dell'immagine

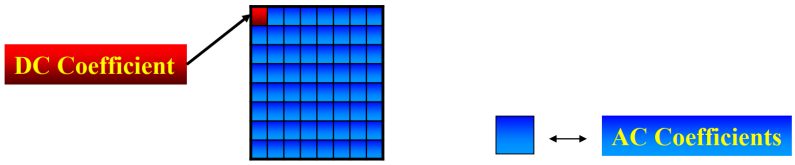
> *Se abbino un coefficiente BASSO allora non c'è irregolarità*
> *Se abbino coefficiente ALTO allora c'è irregolarità.*

- In una scacchiera ho molte regolarità e quindi ho i coefficienti più bassi di sotto.

Risultati della DCT

Prendendo in considerazione il coefficiente in posizione $(0,0)$, ovvero -415, esso prende il nome **coefficiente DC**.
Tutti gli altri (63) coefficienti sono detti **coefficienti AC.**

- **DC è il più importante perchè è proporzionale al VALORE MEDIO** (nella trasformata di Fourier)
  - tutti i coefficienti DC vengono trattati in **UN MODO**, mentre i coefficienti AC in un **ALTRO MODO.**



Quantizzazione

Per quantizzare si usa la seguente formula:

$$F_{quantizzato} = round(\frac{F}{Q})$$

dove Q è detto "*fattore di quantizzazione*".

- Se Q alto, allora nel complesso della matrice ottengo numeri "bassi" perchè ho una divisione
- Se Q è basso allora i valori tendono a mantenersi.

> *Di norma si prevedono **100 valori di Q**. Di conseguenza se è 1 allora non fa nulla, altrimenti sì.*

Scegliere Q è un grado di libertà. In questo passo si sceglie la **qualità dell'immagine**. In caso di $Q = 1$ allora la qualità rimane la stessa.

> ***In realtà** non si tratta dello stesso Q al denominatore ma **si tratta di una matrice**. Il JPEG è più complesso della formula di dove c'era Q perchè Q è una matrice. Si fa la divisione puntuale dove ho una matrice che contiene tutti gli elementi che vanno al denominatore.*

In base alla sottomatrice che sto analizzando, la matrice di Q può essere di diversi modi:

Tabella di quantizzazione luminanza

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

Tabella di quantizzazione crominanza

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

Si osservi che un fattore di compressione maggiore comporta una maggiore perdita di informazione.

L'utente del JPEG può scegliere il "grado" di quantizzazione da adottare fornendo un "quality factor" QF che va da 1 a 100. La tabella di quantizzazione adottata sarà una copia delle tabelle sopra i cui elementi sono divisi per QF.

Maggiore il QF, minore i fattori di quantizzazione e minore la perdita di informazioni.

> *Scegliere un Q da 1 a 100 equivale a scegliere una fra le 100 matrici.*

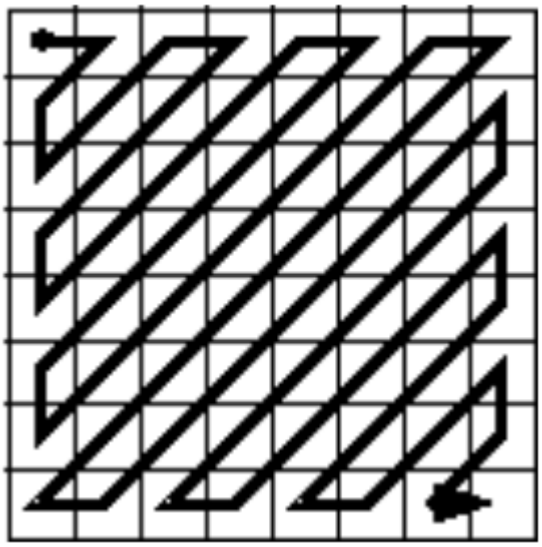
Quindi ciò che si fa è:

$$\begin{bmatrix} -415 & -29 & -62 & 25 & 55 & -20 & -1 & 3 \\ 7 & -21 & -62 & 9 & 11 & -7 & -6 & 6 \\ -46 & 8 & 77 & -25 & -30 & 10 & 7 & -5 \\ -50 & 13 & 35 & -15 & -9 & 6 & 0 & 3 \\ 11 & -8 & -13 & -2 & -1 & 1 & -4 & 1 \\ -10 & 1 & 3 & -3 & -1 & 0 & 2 & -1 \\ -4 & -1 & 2 & -1 & 2 & -3 & 1 & -2 \\ -1 & -1 & -1 & -2 & -1 & -1 & 0 & -1 \end{bmatrix} / \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

$$= \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\ 1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Ordinamento Zig-zag dei coefficienti AC

- Da questa matrice risultante 8x8 si vuole ottenere **UN VETTORE** di 64 elementi., quindi un array.
- L'obiettivo è avere **tantissimi 0 tutti vicini** perchè ci sono algoritmi RLE che funzionano bene quando la **RUN** (sequenza di 0) **è grande**.
- Il metodo utilizzato è quello **a serpentina** partendo dal valore **DC** in posizione (0,0).

- Il percorso da seguire è:
  - 

  E dall'esempio si ottiene:

  -26 -3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB

  dove:
- in `EOB` sono tutti zeri e "*me li dimentico*" perchè si sa a prescindere il loro valore.

> *Quanti vettori da 64 ho?* Dipende dall'immagine 8x8. Alcuni provengono da Y e altri da Cb e altri da Cr e questa informazione non mi serve più. (è stata utile solo per il sottocampionamento e la quantizzazione (tabella di quantizzazione).

Ho solo tanti vettori con al massimo 64 elementi, uno per ogni blocco 8x8 creato

- Ognuno di essi avrà `EOB` e tutti avranno un primo elemento cioè `DC`

Codifica differenziale dei coefficienti DC

Si prendono tutti i DC di tutti gli array ottenuti *come se fossero una cosa a parte* e si applica il seguente algoritmo che applico a 2 array generici (*non quello soprastante*):

- Il primo coefficiente DC è -26 e per esempio il secondo DC in questione è -17
- La differenza fra i due coefficienti è $\Delta = DC_1 - DC_2$
  - $\Delta = -26 - (-17) = 19$ (Quindi faccio una **CODIFICA DIFFERENZIALE**)
  - se poi il 3° DC sarebbe stato -3, allora avrei avuto $-17 - (-3) = -14$ e avrei trovato un altro valore per Δ

> Il primo DC **lo devo sempre portare dietro** nella catena di differenze quindi **il primo DC si memorizza SEMPRE**.

Come codifico il Δ che ottengo? Si fa uso di apposite tabelle:

Esame: Codificare una stringa seguendo una data tabella

La tabella che è nota è:

- Nella tabella delle categorie, -9 sta nella posizione n=SSSS=4.

SSSS	Δ
0	0
1	-1, 1
2	-3 -2, 2 3
3	-7 … -4, 4 … 7
4	-15 … -8, 8 … 15
5	-31 … -16, 16 … 31
6	-63 … -32, 32 … 63
7	-127 … -64, 64 … 127
8	-255 … -128, 128 … 255
9	-511 … -256, 256 … 511
10	-1023 … -512, 512 …1023
11	-2047 … -1024, 1024 … 2047

SSSS = categoria

Δ = differenza tra due coefficienti DC (evento)

n.b.: le tabelle complete sono disponibili su studium!

1. Cerco il range del Δ e vedo l'$SSSS$ corrispondente (tabella soprastante)
2. Ogni $SSSS$ ha un codice binario (tabella sottostante)

SSSS	Codice base
0	010
1	011
2	100
3	00
4	101
5	110
6	1110
7	11110
8	111110
9	1111110
10	11111110
11	111111110

n.b.: le tabelle complete sono disponibili su studium!

- Il **Codice base** nella tabella soprastante mi indica la **PRIMA PARTE** della codifica finale
- A tale codice devo aggiungere altri SSSS (4) numeri dopo il binario.
  - Es: $SSSS = 4$ è il numero di bit mancanti da aggiungere al 101

Vedendo meglio la prima parte del codice, esso ha dei **codici di lunghezza variabile**, **non ci sono codici prefissi.** La tabella in questione è quella di **HUFFMAN.**

> Il codice all'evento SSSS=3 è il più corto, quindi è il più frequente. Mentre 11 è il meno frequente.

In JPEG **NON** si calcola Huffman perchè le tabelle sono già NOTE.

- La prima parte di codice quindi si fa con Huffman. Della seconda parte so solo che mancano 4 bit. Quindi se la codifica che trovo in seguito avrà codifica più lunga allora so che devono essere considerati solo 4 bit.

- A questo punto occorre completare il codice.
- Per fare ciò si usa la seguente regola:
- Se Δ > 0 allora i bit da aggiungere sono gli n bit meno significativi del valore Δ in binario.
- Se Δ < 0 allora i bit da aggiungere sono gli n bit meno significativi del valore in binario di Δ (con complemento a due) ai quali occorre sottrarre il valore 1.
- Δ = 0 allora anche SSSS è uguale a zero, pertanto non viene aggiunto alcun bit.

- Nel nostro caso siamo $\Delta < 0$
  - 9 = 1001 (in biniaro)

- *complemento a 1* = 0110 *switcho 0 con 1*
- complemento a 2 = 0111 *switcho 0 con 1 e poi sommo 1*.
- Quindi si tratta di complemento a 1. E quindi trovo **0110**
- 0110 rappresenta la 2° parte del codice (ovvero "*prendo 4 bit mancanti*")
- codifica finale del coefficiente DC in questione= 101**0110**

> - Se $\Delta > 0$, per esempio $\Delta = 3$, $\Delta = 3 = SSSS = 2$ -> codifica base **100**
> - devo completare con n=2 bit.
> - $\Delta > 0$, quindi scrivo solo SSSS in binario **10**, aggiungo 1 e trovo **11**.
> - Codifica finale 10011

Codifica Run-Length dei coefficienti AC

Dopo aver codificato i coefficienti DC, posso passare alla parte di array che "succede" DC. Nei coefficienti di tipo AC, invece **codifico con coppie**:

- Poiché i coefficienti quantizzati AC sono spessissimo nulli, si usa una trasformazione in **skip-value**.
- Cioè, data una sequenza di valori, si memorizza il **numero degli zeri seguito dal primo valore non zero che si incontra**.

> Esempio: si debba codificare 0,0,0,0,11,0,0,0,3,0,0,0,0,0,0,0,0,12,17… memorizzo: (4,11),(3,3),(8,12),(0,17),… -> ovvero le coppie(4 zeri seguito da 11), (3 zeri seguito da 3)

-3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB

- Nel nostro caso sarebbe

(0,-3), (0,1), (0,-3),(0,-2), (0,-6), (0,2), (0,-4),
(0,1), (0,-4), (0,1), (0,1), (0,5), (1,2), (2,-1),
(0,2), (5,-1), (0,-1) …

- La prima coppia del tipo (0,v) da codificare è (0,-3).
- Il coefficiente -3 ha come SSSS il valore 2.

SSSS	Coefficiente AC
1	-1, 1
2	-3 -2, 2 3
3	-7 … -4, 4 … 7
4	-15 … -8, 8 … 15
5	-31 … -16, 16 … 31
6	-63 … -32, 32 … 63
7	-127 … -64, 64 … 127
8	-255 … -128, 128 … 255
9	-511 … -256, 256 … 511
A	-1023 … -512, 512 … 1023

n.b.: le tabelle complete sono disponibili su studium

- prendo il primo elemento della coppia (cioè 0) e SSSS corrispondente e creo la coppia **(0,2)**

Vedo la seconda tabella:

(run, category)	Codice base	Lunghezza codice completo
(0, 0)	1010 (= EOB)	4
(0, 1)	00	3
(0, 2)	01	4
(0, 3)	100	6
….	….	….
(F, 0)	111111110111	12
….	….	….
(F, A)	1111111111111110	26

n.b.: le tabelle complete sono disponibili su studium

Da essa trovo che **Codice base** che ottengo è **01** perchè guardo la nuova coppia (0,2) e devo scrivere altri 2 numeri in binario visto che il codice completo è 4 (*dalla tabella*)

- Alla coppia (0, 2) corrisponde il codice base 01, tale codice sarà completato dall'aggiunta di un numero di bit fino a raggiungere il numero totale di bit che è riportato nell'ultima colonna della tabella.
- I bit che completano il codice base sono scelti con lo stesso criterio enunciato per la codifica dei coefficienti DC, in base al valore v del coefficiente AC (v > 0, v < 0, v = 0).
- Se v > 0 allora i bit da aggiungere sono i bit meno significativi del valore v in binario.
- Se v < 0 allora i bit da aggiungere sono i bit meno significativi del valore in binario di v (con complemento a due) ai quali occorre sottrarre il valore 1.
- v = 0 allora anche SSSS è uguale a zero, pertanto non viene aggiunto alcun bit.

Nel nostro caso $v < 0$ quindi:

- scrivo -3 in binario e sottraggo 1
  - 3=011 -> 100 e prendo i 2 meno significativi
  Quindi la codifica della coppia (0,-3) è **0100**

> E lo stesso ragionamento vale per tutti gli altri AC

- La sequenza finale sarà

-26 -3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB

1010110 0100 001 0100 0101 100001 0110 100011 001 100011 001 001 100101 11100110 110110 0110 11110100 000 1010

> **EOB** è gia definito e vale **1010**

Considerazioni JPEG

- La sequenza che si ottiene è la sequenza che codifica il blocco 8x8.
- Non avrò ambiguità perchè Huffman lo garantisce

Confronto tra blocchi

- Attenzione, il Jpeg è lossy! Quindi il blocco ricostruito è diverso da quello in input.

52	55	61	66	70	61	64	73
63	59	66	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

58	64	67	64	59	62	70	78
56	55	67	89	98	88	74	69
60	50	70	119	141	116	80	64
69	51	71	128	149	115	77	68
74	53	64	105	115	84	65	72
76	57	56	74	75	57	57	74
83	69	59	60	61	61	67	78
93	81	67	62	69	80	84	84

> I blocchi si assomigliano ma non sono uguali proprio perchè si hanno perdite di informazioni(lossy)

- Nella quantizzazione si è scelto un fattore di qualità Q. Piu aumenta Q piu ho dettagli. (a differenza di quello che ho scritto prima durante la spiegazione dei passi, quindi correggere l'appunto soprastante)

JPEG2000

Questo algoritmo di compressione è stato creato per migliorare questi risultati e **PER NON AVERE ARTEFATTI**.

- Non si fanno blocchi 8×8 ma si considera **L'INTERA IMMAGINE** come unica immagine.

- Servirebbero basi della DCT grandi quanto l'immagine $M \times N$ ma questo approccio non conviene.

- Per tale allora si usa la **Trasformata di Wavelet** che consiste nel dividere l'immagine in 4 quadranti. In particolare:

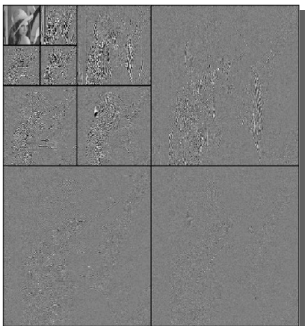
  - Crea un'immagine grande quanto quella iniziale e dividila in **4 PARTI**: sottocampionamento(in alto a sinistra), shift di una riga (in basso a sinistra), shift di una colonna (in alto a destra) e shift di una riga e una colonna(in basso a destra).

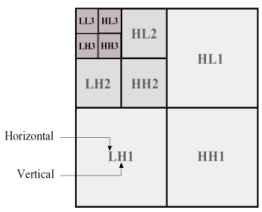
- Ricorsivamente si altera tale algoritmo prendendo come **nuova immagine di input l'immagine sottocampionata.**

- L'algoritmo si **ferma** quando, nell'ultima immagine sottocampionata, si ha solamente **UN PIXEL**.

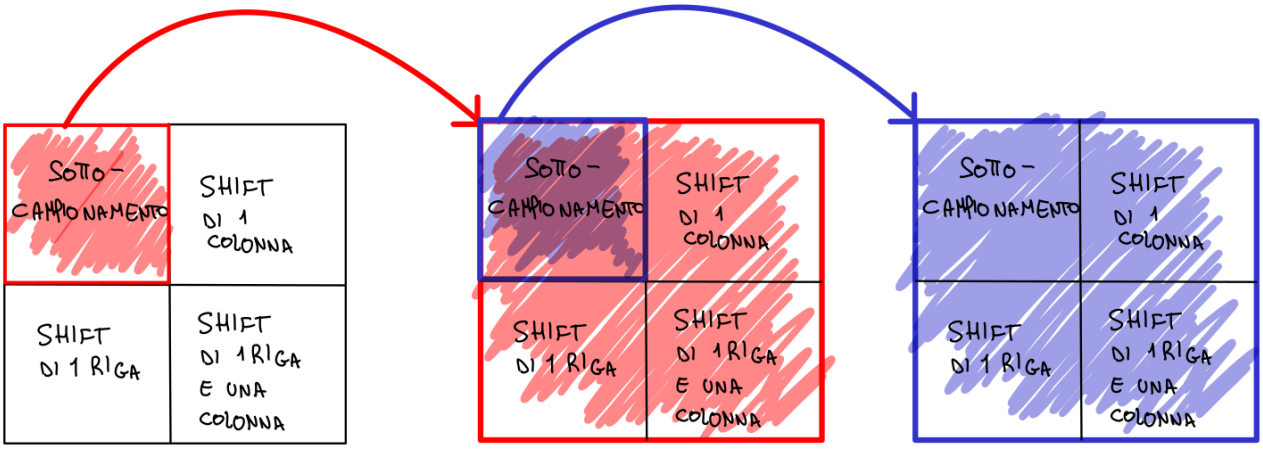
> *Quindi si ha sottocampionamento di sottocampionamento di sottocampionamento di sottocampionamento ecc..*

Graficamente si ha:





ovvero:



Da questo passi in poi si usa questa per fare la **CODIFICA DIFFERENZIALE**(differenza fra i DC) o SKIP-VALUE (ovvero con le coppie di 0 e numero di volte che lo 0 si presenta)

In tutte le trasformate, nel primo coefficiente c'è DC che rappresenta **IL VALORE DI MEDIA**. dopo la trasformata di wavelet,

DIFFERENZE FRA JPEG E JPEG2000

Le uniche due differenze fra questi algoritmi di compressione sono:

- In JPEG2000 non si divide più in quadrotti 8x8 ma **SI CONSIDERA TUTTA LA MATRICE**;
- Si usa la **Trasformata di Wavelets** e non più la Trasforamata Discreta dei Coseni.