Compressione di immagini tramite la DCT

Davide Pietrasanta - 844824
Davide Mammarella - 865536
Andrea Premate - 829777

https://github.com/davidepietrasanta/Image-compression-via-DCT

Obiettivo - Parte 1

- Implementare ***DCT2 in ambiente Open Source*** e ***confrontare i tempi di esecuzione*** con la DCT2 ottenuta utilizzando la libreria dell'ambiente

Linguaggio

Abbiamo scelto **Python** per questo progetto.

Abbiamo scelto questo linguaggio pur sapendo non sia il linguaggio più performante in quanto di semplice utilizzo e noto a tutto il team.

Abbiamo scelto la libreria **opencv-python**, basata su C++.

Considerazioni Iniziali

Per la prima parte del progetto abbiamo provato a creare diverse dct per vedere quale fosse la più performante.

Una dimensione:

- dct: Approccio usato in classe per una sola dimensione (più veloce).
- dct matriciale: L'alternativa era di vedere la DCT come prodotto matrice (più lenta).

Due dimensioni:

- dct2 diretto: Approccio O(n^4).
- dct2 colonne e righe: Approccio visto a lezione O(n^3).
- dct2 colonne e righe matriciale: Approccio visto a lezione ma in forma matriciale (più veloce).

Codice - dct2 matriciale

```python
def dct2(matrix):
    """
        DCT for two dimension np.array.
        matrix: Is a np.array of one dimension [M:N]
        return: Discrete cosine transform of matrix
    """
    N = matrix.shape[0]
    M = matrix.shape[1]
    C1 = np.zeros( shape = (N,N) )
    C2 = np.zeros( shape = (M,M) )
    Z = np.zeros(matrix.shape)


    for j in range(N):
        C1[0, j] = np.sqrt(1/N)

    for k in range(1, N):
        for j in range(N):
            C1[k, j] = np.cos(k * np.pi * (2*(j+1) - 1)/(2*N)) * np.sqrt(2/N)

    for j in range(M):
        C2[j, 0] = np.sqrt(1/M)

    for k in range(1, M):
        for j in range(M):
            C2[j, k] = np.cos(k * np.pi * (2*(j+1) - 1)/(2*M)) * np.sqrt(2/M)

    Z = np.dot(C1, matrix)
    Z = np.dot(Z, C2)

    return Z
```

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

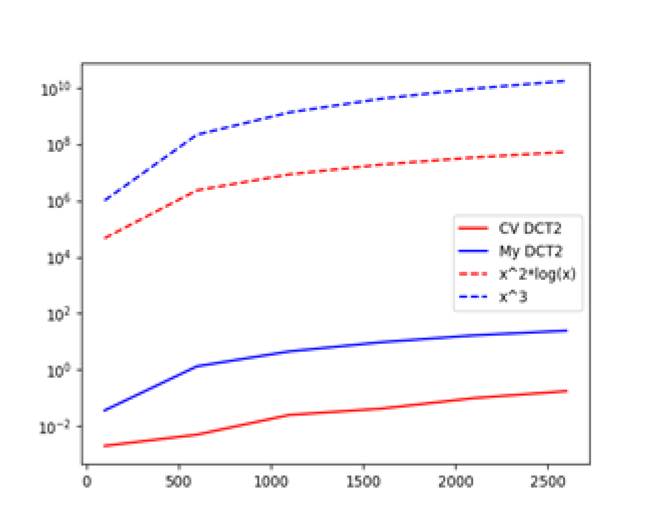
$$for\ u,v = 0,1,2,\ldots..N-1$$

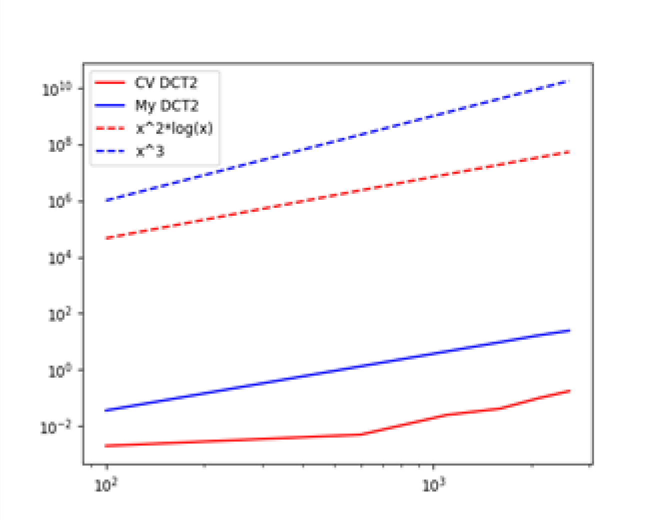
$$\alpha(u) = \begin{cases} \sqrt{\dfrac{1}{N}} & for\ u = 0 \\ \sqrt{\dfrac{2}{N}} & for\ u \neq 0 \end{cases}$$

$$\alpha(v) = \begin{cases} \sqrt{\dfrac{1}{N}} & for\ v = 0 \\ \sqrt{\dfrac{2}{N}} & for\ v \neq 0 \end{cases}$$

Misure a confronto

Per testare la dct più veloce e quella di libreria abbiamo creato varie matrici in modo casuale, con numeri da 0 a 255.





Obiettivo - Parte 2

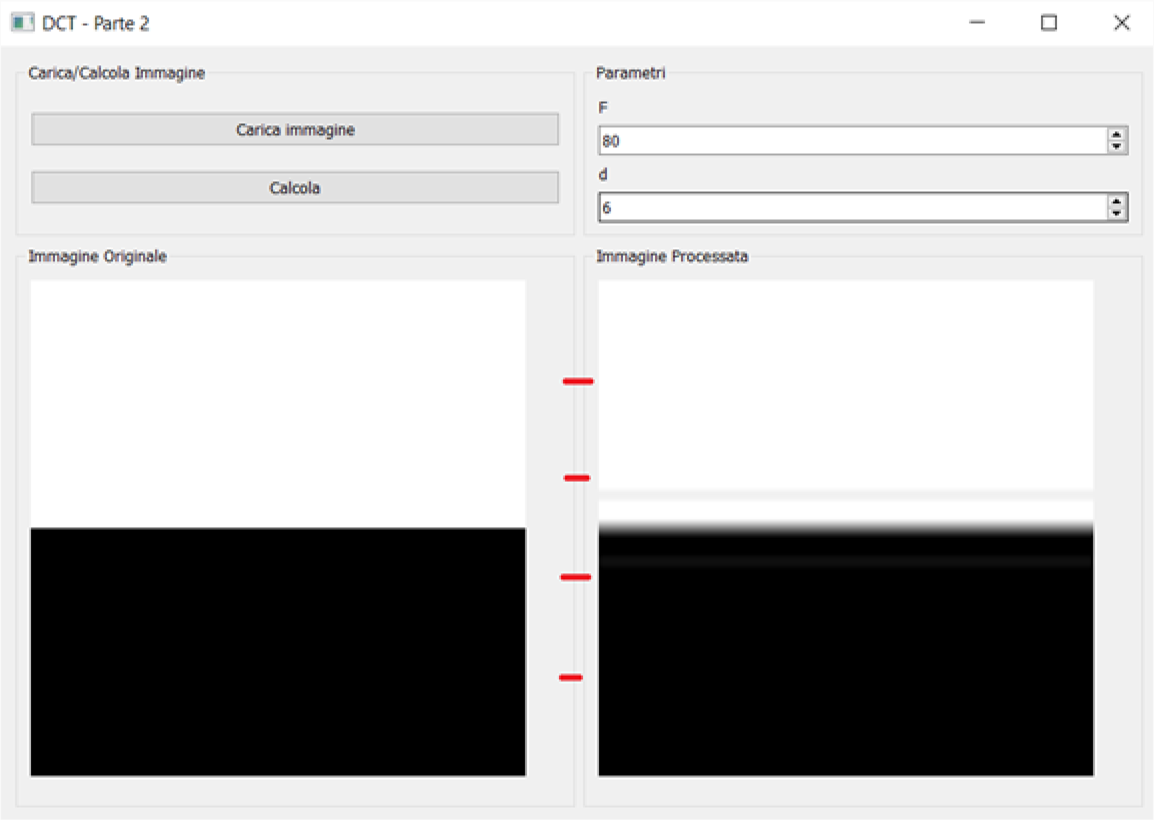
- Studiare gli ***effetti della compressione JPEG*** (senza utilizzare una matrice di quantizzazione) sulle immagini in toni di grigio. La compressione viene effettuata tramite:
  - DCT2
  - Eliminazione delle frequenze
  - IDCT2
  - Arrotondamento

- Librerie utilizzate:
  - **opencv-python**: implementazione DCT2
  - **PyQt5**: interfaccia grafica

Esempi - 1

- Le ***prime e ultime due righe*** di blocchi rimangono ***inalterate*** in quanto descrivibili con d=1 e quindi constanti.

- ***Fenomeno di Gibbs*** (linee grigie sopra e sotto la zona di transizione) osservabile in maniera parziale:
  - Nella zona nera (valore 0) si può osservare solo l'oscillazione positiva, in quanto quella negativa viene riportata a 0 (per essere visualizzata come immagine).
  - Discorso opposto per la zona bianca.

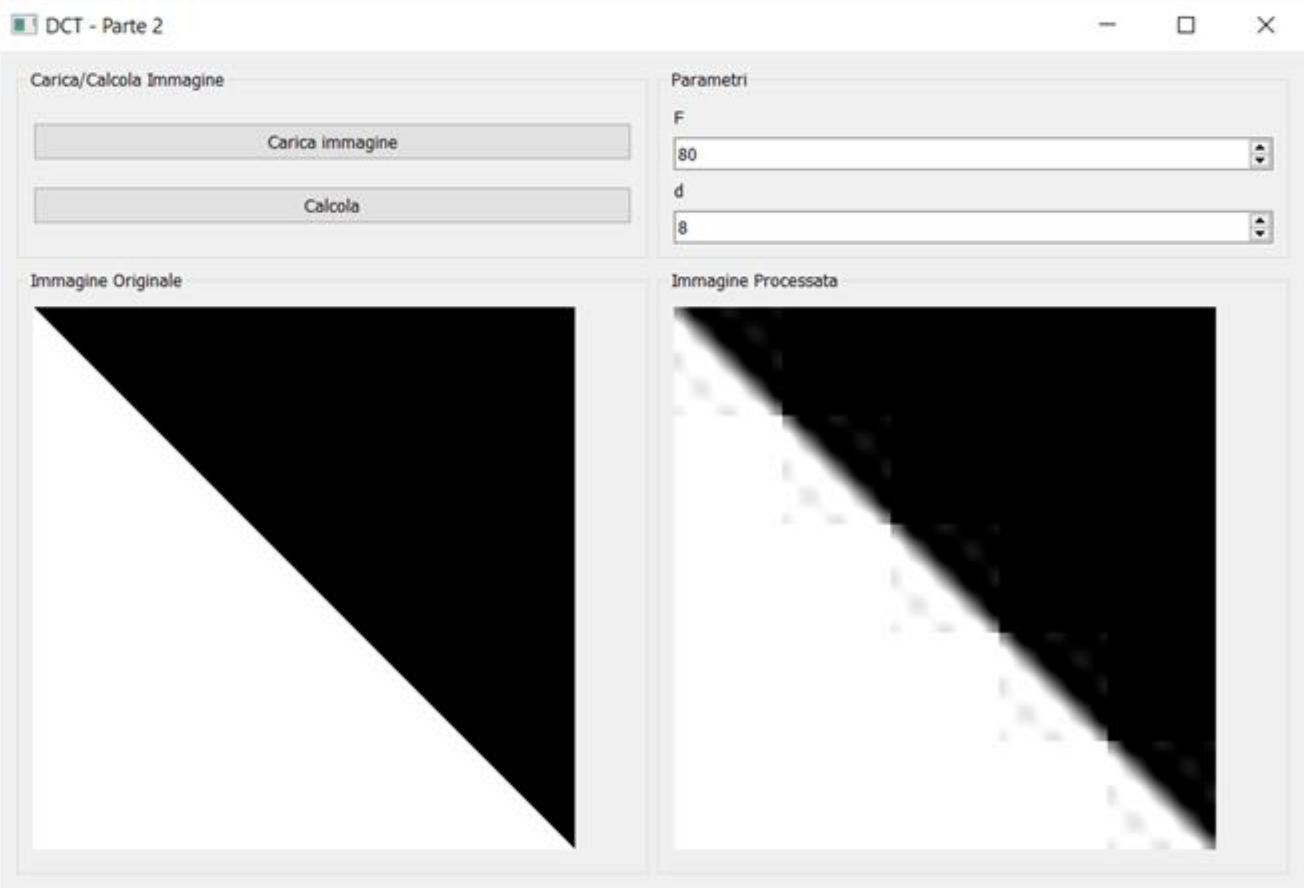


Esempi - 2

- ***Aumentando d*** (8 e 12) sempre più frequenze diventano disponibili e quindi vediamo il fenomeno di Gibbs riproporsi un numero maggiore di volte ma con intensità e larghezza sempre minore.
- La ***transizione centrale*** da bianco a nero diventa invece sempre ***più rapida***.

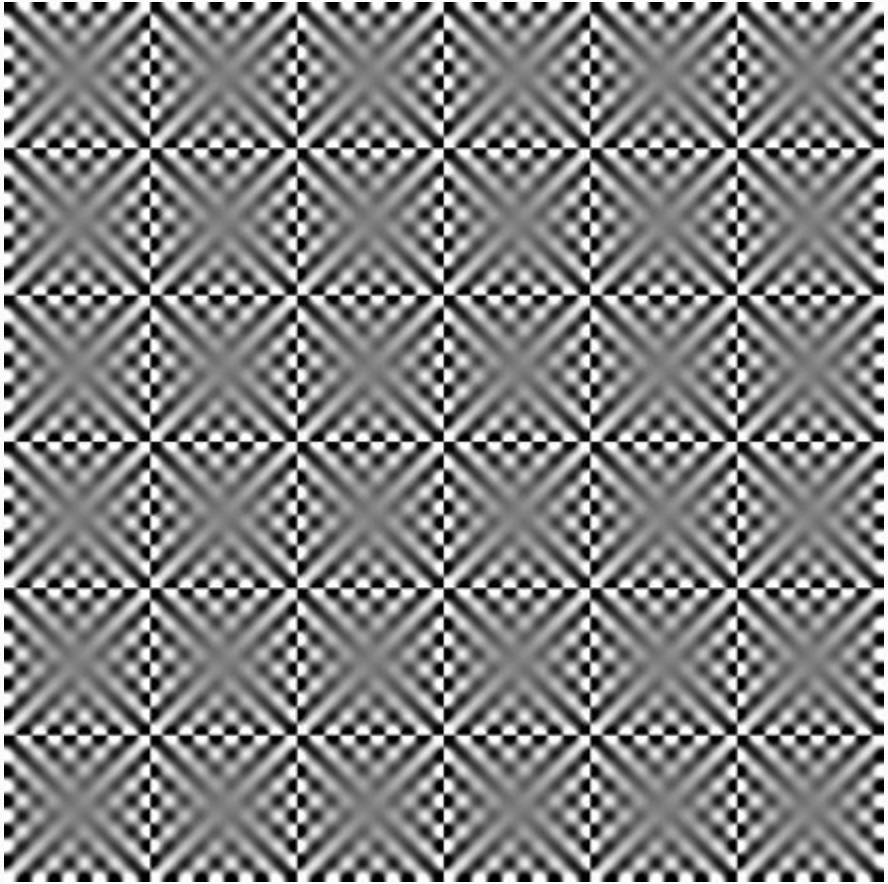
Esempi - 3

Utilizzando una ***transizione diagonale*** si può osservare lo stesso ***fenomeno sulle due dimensioni***.



Esempi - 4

La scacchiera 640x640 processata con F=100 e d=20. Le frequenze non sono sufficienti e lasciano quindi nella zona centrale di ogni blocco un grigio diffuso (blur).



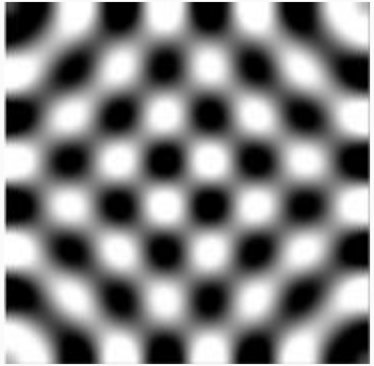
Esempi - 5

Facendo diverse prove si osserva che l'effetto di grigio nella parte centrale di ogni blocco, mantenendo F multiplo della grandezza delle caselle(10) svanisce quando d > 2*F/grand_caselle.

F=50, d=10

F=50, d=11

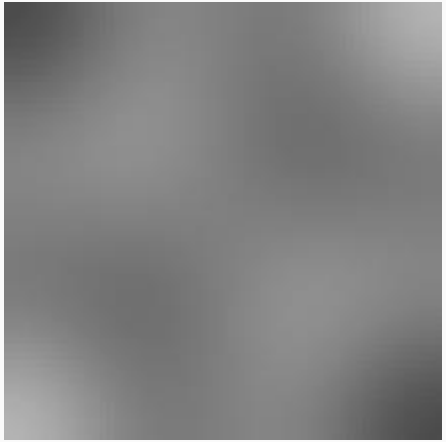
F=80, d=16



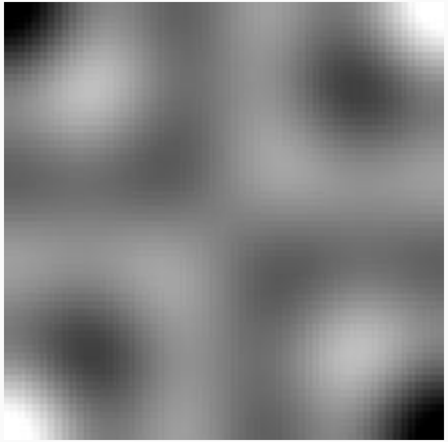
F=80, d=17

Esempi - 6

Similmente, se proviamo ad aumentare d partendo da 1, con d=F/grand_caselle+1 (e n°caselle bianche=n°caselle nere) si ottiene per la prima volta il nero e il bianco totale(negli angoli). Con quel valore di d infatti si ottiene la situazione in cui la costante grigia determinata dalla prima frequenza (0,0) viene eguagliata(o superata) in valore assoluto dalla somma di coseni, rendendo possibile la vista di nero(0) e bianco(255).



F=60, d=6



F=60, d=7

Esempi - 7

L'immagine del cervo processata con F=500 e d=40 risulta simile all'originale nelle zone di sfondo (luminosità simile), mentre perde di qualità sul primo piano, dove si ha una maggiore estensione sui valori di luminosità: si ottiene quindi il tipico effetto a onde e sono riconoscibili i blocchi.

Esempi - 8

Per migliorare la qualità è necessario quindi ridurre la dimensione dei blocchi o aumentare d. Il formato jpeg utilizza blocchi da 8x8. Inoltre tale formato non tronca di netto le frequenze più alte ad un certo d (come il nostro algoritmo), ma utilizza matrici di quantizzazione.

F=50, d=4

Conclusioni

Mantenendo uguale il rapporto d/F, quando abbiamo F grande è più facile vedere fenomeni di “ringing”(fenomeno di Gibbs), mentre quando abbiamo F piccolo è più facile vedere l’effetto quadrettatura, specie ingrandendo. Complessivamente però il risultato migliore sembra essere quello di mantenere F piccolo (come jpeg).