Università degli Studi di Milano-Bicocca

Scuola di Scienze Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea Magistrale in Informatica

Progetto di Metodi del Calcolo Scientifico

MCS-progetto-2

Anno Accademico 2024-2025

Relazione a cura di:

Oleksandra Golub

Matricola:

856706

Sommario

1. Introduzione

Lo scopo del progetto è implementare e confrontare due versioni della Trasformata Coseno Discreta 2D (DCT-II):

- una **naïve** basata su moltiplicazioni matriciali ($O(N^3)$);
- una **fast** basata su FFT ($O(N^2 \log N)$);

Per poi sviluppare un **compressore a blocchi** che elimina le frequenze alte secondo il criterio **k+ℓ≥d**. La valutazione copre: correttezza numerica, tempi/complessità, qualità percettiva (PSNR) e rapporto di compressione.

1.1 Scelte tecnologiche

L'implementazione è in **Python**. Si usano:

- **NumPy** per le operazioni vettoriali e matriciali di base.
- **SciPy.fft** per le trasformate DCT/IDCT veloci (type=2, norm='ortho').
- **Pillow** per l'I/O delle immagini.
- **Matplotlib** per la generazione di grafici.
- **Tkinter** per la GUI.

Le implementazioni principali sono state organizzate come segue:

- **utils.py**: DCT2 naïve (matrice **D**, **C = DxD^T**) e versioni DCT2/IDCT2 veloci.
- **parte1_dct_comparison.py**: test comparativo che esegue più varianti di un algoritmo nelle stesse condizioni per confrontarne prestazioni.
- **parte2_compressor.py**: interfaccia grafica del compressore.
- **analisi_compressione.py**: analisi e visualizzazioni dei risultati.
- **esperimenti_finali.py**: esperimenti sistematici con esportazione dei risultati (CSV/HTML con PSNR e percentuale di compressione).

1.2 Procedura di validazione

Per verificare la correttezza e l'efficacia delle implementazioni sono state previste quattro fasi:

- **Correttezza**
  È stato confrontato il risultato della DCT2 naïve con quello della versione veloce sul blocco **8×8** fornito nel PDF della consegna. Inoltre, è stata verificata la coerenza tra DCT2 e IDCT2: l'errore massimo riscontrato nella ricostruzione risulta dell'ordine di **10^(-10)**, quindi trascurabile.

- **Benchmark**
  Sono stati misurati i tempi di esecuzione su matrici casuali di dimensione **N×N**, con **N variabile tra 4 e 1024**. I risultati sono stati riportati su grafico in scala logaritmica sull'asse delle y, in modo da evidenziare la differenza di crescita tra l'approccio naïve e quello veloce.

- **Compressione a blocchi**
  Le immagini in scala di grigi vengono suddivise in blocchi di dimensione **F×F**. Su ciascun blocco si applica la DCT2, seguita dall'azzeramento dei coefficienti con indici **k+ℓ≥d**. Infine, si applica l'IDCT2 per ricostruire l'immagine compressa. Questo approccio riproduce il principio base della compressione JPEG.

- **Metriche di valutazione**
  - **PSNR (Peak Signal-to-Noise Ratio):** misura la fedeltà dell'immagine ricostruita rispetto all'originale.
  - **Percentuale di compressione teorica:** calcolata come

$$1 - \frac{\#\text{coefficienti tenuti}}{F^2}$$

    che rappresenta la quota di coefficienti eliminati.

  - **Percentuale di pixel scartati:** dovuta al taglio delle dimensioni dell'immagine a multipli di **F** (riportata nelle tabelle degli esperimenti).

2. Architettura della libreria

Il progetto è organizzato come una piccola libreria per la compressione di immagini basata sulla trasformata discreta del coseno bidimensionale (DCT2).

La libreria si compone di un modulo di supporto (utils.py), che contiene le implementazioni fondamentali della DCT2 e delle sue varianti, e di diversi script dedicati a benchmark, GUI e analisi dei risultati sperimentali.

2.1 utils.py

Il modulo utils.py raccoglie le routine di base su cui si fonda l'intero progetto.

- **_dct_matrix(N)** genera la matrice ortonormale della DCT-II, seguendo la definizione vista a lezione. È il "mattone" matematico usato per implementare la versione naïve.

- **dct2_naive(X)** e **idct2_naive(C)** implementano rispettivamente la DCT2 e la IDCT2 in modo diretto, tramite moltiplicazioni matriciali (**C=D X D^**T).
  Hanno complessità cubica **O(N^3)**, quindi sono utili solo a fini didattici e di verifica.

- **dct2_fast(block)** e **idct2_fast(C)** sono le versioni veloci basate su **scipy.fft.dct** e **idct**. L'implementazione è coerente con la definizione ortonormale, quindi i risultati sono confrontabili con la versione naïve, ma con complessità molto più bassa (in pratica quasi lineare).

- **measure_time(func, *args, n_iterations=3)** è una piccola utility per misurare il tempo medio di esecuzione di una funzione, utile per i test comparativi (benchmark).

In sintesi, **utils.py** fornisce sia le versioni "teoriche" della DCT2/IDCT (a scopo di validazione), sia le versioni ottimizzate da usare in pratica, oltre a strumenti di supporto per il confronto delle prestazioni.

2.2 test_dct.py

Questo modulo è dedicato esclusivamente alla **validazione delle implementazioni** della DCT2 e IDCT2 fornite in **utils.py**.

Uso: **python test_dct.py**

Lo scopo è dimostrare che:

1. la versione naïve (basata su moltiplicazioni matriciali) e quella veloce (basata su SciPy) producono risultati equivalenti;
2. la ricostruzione tramite IDCT2 riporta il blocco originale con errore numerico trascurabile;
3. le uscite sono coerenti con i valori attesi riportati nel materiale della consegna.

Il test è strutturato in due parti principali:

- **(A) Verifica 2D su un blocco 8×8 di riferimento:**
  - si calcola la DCT2 con entrambe le versioni (naïve e fast) e si confronta la differenza elemento per elemento;
  - si controlla la coerenza numerica applicando l'IDCT2 per ricostruire il blocco originale (errore massimo ≈ 1e−10);
  - si confronta la prima riga della DCT2 calcolata con quella riportata nel PDF, per un riscontro "visivo" dei valori.
- **(B) Verifica 1D sulla prima riga:**
  - viene applicata la DCT monodimensionale (scipy.fft.dct) alla prima riga del blocco;
  - i valori ottenuti vengono confrontati con quelli attesi dal PDF;
  - si usa una tolleranza ragionevole, perché i valori del documento sono già arrotondati.

Esito:

Il test stampa a video le differenze numeriche e conferma se l'implementazione è coerente. Se entrambe le verifiche (ricostruzione e DCT 1D) rientrano nelle soglie di errore, l'esito è "TEST PASSED". In caso contrario, viene segnalata la necessità di controllare scalatura, tipo di DCT o arrotondamenti.

2.3 parte1_dct_comparison.py

Questo modulo esegue un **benchmark controllato** per confrontare le prestazioni della DCT2 **naïve** (moltiplicazioni matriciali **O(N^3)**) con la DCT2 **veloce** basata su FFT (**O(N^(2)log N)**).

L'obiettivo è duplice:

- mostrare sperimentalmente la diversa crescita dei tempi al variare di N;
- produrre un **grafico** riassuntivo e una **stampa** dei rapporti attesi/ottenuti, utile a discutere la complessità;

Uso: **python parte1_dct_comparison.py**

Struttura del file:

- **benchmark_dct_implementations()** esegue l'intero esperimento (generazione dati, timing, grafico, analisi) e restituisce le serie misurate.

- **create_comparison_plot(...)** costruisce il grafico semilog con i dati misurati e le curve teoriche, e lo salva su disco.

- **analyze_complexity(...)** stampa l'analisi numerica delle crescite (rapporti attesi vs. osservati) e lo speedup.

- **create_results_folder()** crea la cartella risultati/ se non esiste.

- __**main**__ crea la cartella, lancia il benchmark e riepiloga l'esito.

Note:

- La **naïve** viene **fermamente limitata** a **N ≤ 128** per evitare tempi eccessivi; per la **fast** si spinge **fino a 1024**.
- L'asse y in **log** permette di visualizzare range di tempi molto diversi sulla stessa figura.
- L'uso di molte iterazioni per la fast stabilizza la misura su durate molto piccole (rumore di timing).

2.4 parte2_compressor.py

Questo modulo fornisce una **GUI Tkinter** per provare la compressione a blocchi basata su DCT2.

L'utente può caricare un'immagine BMP in scala di grigi, scegliere i parametri della compressione e visualizzare **anteprima**, **PSNR** e **percentuale teorica di compressione**. È anche possibile salvare l'immagine compressa.

Uso: **python parte2_compressor.py**

Struttura del file:

- **class DCTImageCompressor** incapsula la GUI e la logica, dove:
  - **__init__** + **setup_gui()** costruiscono i pannelli (controlli, anteprima originale/compressa, status bar).
  - **_update_d_max()** mantiene coerente il range di **d** quando cambia **F**.
  - **load_image()**: file dialog, conversione in scala di grigi, anteprima e status.
  - **compress_image()**: valida i parametri, avvia la compressione e aggiorna anteprima + statistiche.
  - **dct_compress(image, F, d)** è il core della compressione a blocchi.
  - **display_image(...)** ridimensiona e mostra un array come immagine nella GUI.
  - **show_compression_stats(F, d)** calcola coeff. tenuti/scartati, compressione %, MSE/PSNR (sull'area ritagliata).
  - **save_compressed()** salva su disco l'immagine ricostruita.
- **main()** avvia l'app Tkinter.

Note:

- Maschera "triangolare" semplice (criterio **k+ℓ<d**): serve a mostrare il trade-off qualità/compressione; non usa quantizzazione JPEG né zig-zag.

2.5 analisi_compressione.py

Questo modulo serve a **studiare e visualizzare** gli effetti della compressione DCT sui contenuti d'immagine.

Produce due risultati principali:

1. una **griglia di immagini** che mostra come cambia la qualità al variare della soglia **d** (con PSNR e % di compressione),
2. una **mappa delle frequenze mantenute** per diversi valori di ddd (per capire visivamente il taglio "triangolare" **k+ℓ<d**).

Uso: **python analisi_compressione.py**

Struttura del file:

- **compress_image_dct(image, F, d)**
  - Divide l'immagine in blocchi **F×F** (ritagliando eventuali "avanzi").
  - Per ogni blocco:
    1. DCT2;
    2. azzera i coefficienti con k+ℓ≥d;
    3. IDCT2;
    4. arrotonda [0,255];
  - Restituisce l'immagine compressa (uint8) con dimensioni multiple di FFF.
  - Scopo: funzione **riutilizzabile** per gli esperimenti.

- **analyze_compression_effects()**
  - Carica una piccola **suite di immagini di test** (bridge.bmp, cathedral.bmp, gradient.bmp, 640x640.bmp).
  - Fissa **F=8** (scelta classica stile JPEG) e valuta **più soglie d∈{1,2,4,8,12,14}**.
  - Per ciascuna immagine: mostra l'originale e, accanto, le versioni compresse per ogni **d**.
  - Per ogni compressione calcola:
    - **PSNR** (sull'area effettivamente compressa, cioè dopo il ritaglio ai multipli di **F**);
    - **Compressione teorica (%)** = **100*{1−[#coeff. tenuti/(F^2)]}**, con **#coeff. tenuti=|{(k,ℓ):k+ℓ<d}|**.
  - Salva la figura complessiva in **risultati/analisi_compressione_completa.png**.

- **plot_frequency_spectrum()**
  - Per **F=8** costruisce, per vari **d**, una **maschera F×F** che evidenzia quali coefficienti DCT vengono **tenuti** (1) e **scartati** (0).
  - Mostra titoli con il numero di coefficienti mantenuti, assi etichettati come "frequenza orizzontale/verticale" e una griglia per leggere meglio l'indice.
  - Salva la figura in **risultati/frequenze_tagliate.png**.

- **__main__**
  - Crea la cartella **risultati/** (se non esiste).
  - Esegue prima la visualizzazione delle frequenze, poi l'analisi effetti.
  - Stampa un breve riepilogo in console.

Note:

- Maschera "triangolare" semplice (criterio **k+ℓ<d**): serve a mostrare il trade-off qualità/compressione; non usa quantizzazione JPEG né zig-zag.
- Ritaglio ai multipli di **F**: la ricostruzione (e il PSNR) si riferiscono all'area effettivamente compressa; eventuali bordi non multipli di **F** vengono scartati.

2.6 esperimenti_finali.py

Questo modulo esegue una **campagna di esperimenti** strutturata per quantificare il trade-off tra **qualità** (PSNR) e **compressione teorica** al variare di:

- l'immagine di input (suite di test);
- la dimensione del blocco $F \in \{4,8,16\}$;
- la soglia **d** espressa come **percentuale di**

$$d_{max} = 2F - 2 \quad (d/d_{max} \in \{25\%, 50\%, 75\%, 100\%\})$$

Genera una **tabella** (CSV + HTML) con le metriche per ogni combinazione, più un riepilogo in console (filtrato su F=8F=8F=8 e statistiche globali).

Uso: **python esperimenti_finali.py**

Struttura del file:

- **compress_image(image, F, d)** implementa la compressione DCT a blocchi con maschera **k+ℓ<d** e ritorna l'immagine compressa uint8.
- **run_experiments()** raccolta risultati in lista per salvare files CSV/HTML + stampa riepiloghi.
- **__main__** avvia **run_experiments().**

2.7 immagini/

Contiene le immagini di test utilizzate negli esperimenti di compressione DCT (da pattern sintetici a fotografie reali).

Permette di testare la libreria in condizioni molto diverse, evidenziando:

- la capacità della DCT di catturare frequenze basse/alte;
- l'effetto del parametro di soglia **d** sulla qualità visiva;
- le differenze di compressione su contenuti strutturati vs. naturali;

2.8 risultati/

Raccoglie gli output generati automaticamente dagli script di analisi e di esperimento.

In particolare:

- **analisi_compressione_completa.png** è una figura generata da **analisi_compressione.py**: mostra per ogni immagine di test gli effetti della compressione con soglie diverse **d**, riportando accanto a ciascuna il valore di compressione teorica e il PSNR.
- **confronto_dct_tempi.png** è un grafico prodotto da **parte1_dct_comparison.py**: confronto tra i tempi di esecuzione della DCT naïve e della DCT fast, con sovrapposte le curve teoriche **O(N^3)** e **O(N^2\log N)**.
- **frequenze_tagliate.png** è una figura generata da **analisi_compressione.py**: visualizza le maschere di frequenze mantenute/scartate per diversi valori di soglia **d**, evidenziando i coefficienti della DCT che vengono conservati.
- **tabella_esperimenti.csv** è una tabella dei risultati numerici degli esperimenti sistematici (**esperimenti_finali.py**), salvata in formato CSV per analisi o apertura con Excel.
- **tabella_esperimenti.html** è la stessa tabella dei risultati esportata in formato HTML, utile per la visualizzazione diretta in browser.

Questa cartella rappresenta quindi la **documentazione quantitativa e visiva** dei test, e costituisce il materiale di supporto per la relazione.

3. Risultati

In questo capitolo vengono presentati i risultati sperimentali ottenuti con gli script sviluppati per il progetto di compressione tramite DCT.

L'obiettivo è duplice:

- **Verificare la correttezza delle implementazioni** (confronto tra versioni naïve e veloci, coerenza DCT/IDCT, confronto con valori attesi dal materiale didattico).
- **Analizzare prestazioni e qualità** su immagini reali e sintetiche, valutando il compromesso tra compressione teorica e qualità percepita (PSNR).

I file di input provengono dalla cartella **immagini/**, mentre i risultati numerici e grafici sono stati salvati automaticamente nella cartella **risultati/**.

3.1 Test di correttezza

Lo scopo di questo test è dimostrare che:

- la **DCT2 naïve** (implementata via moltiplicazioni matriciali (O(N^3)) e la **DCT2 veloce** (basata su FFT, O(N^2\log N)) producono risultati numericamente coincidenti;
- la **IDCT2 veloce** ricostruisce correttamente i dati originali;
- la nostra implementazione è coerente con i **valori attesi del PDF** (blocco **8×8** fornito nel materiale del corso).

```
(.venv) PS C:\Users\aleqs\mcs_progetto_2> python test_dct.py
============================================================
TEST DCT IMPLEMENTATION
============================================================

1) DCT2 Naive vs Fast
   Max |naive-fast| = 1.435e-12
2) IDCT2 Fast -> max rec err = 1.137e-13, mean = 4.641e-14

3) Prima riga DCT2 (nostra vs attesa PDF):
   nostra : [1118.75   44.02   75.92 -138.57    3.5   122.08  195.04 -101.6 ]
   attesa : [1110.    44.    75.9 -138.     3.5  122.   195.  -101. ]

4) DCT 1D della prima riga (type=2, norm='ortho'):
   nostra : [ 401.99    6.6   109.17 -112.79   65.41  121.83  116.66   28.8 ]
   attesa : [ 401.     6.6  109.  -112.    65.4  121.   116.    28.8]
   max |diff| = 9.902e-01

TEST PASSED (scalatura e implementazione coerenti)
(.venv) PS C:\Users\aleqs\mcs_progetto_2>
```

Osservazioni principali:

- La differenza massima **|naive−fast| ≈ 10^−12** conferma che le due implementazioni sono equivalenti dal punto di vista numerico.

- L'errore massimo di ricostruzione **≈ 10^−13**, media **≈ 10^−14**, indicando che la trasformata inversa ricostruisce l'input con **precisione numerica a doppia precisione**.

- Confronto con valori attesi (PDF):

```
[1118.75   44.02   75.92 -138.57    3.5   122.08  195.04 -101.6 ]
```

1.11e+03	4.40e+01	7.59e+01	-1.38e+02	3.50e+00	1.22e+02	1.95e+02	-1.01e+02
7.71e+01	1.14e+02	-2.18e+01	4.13e+01	8.77e+00	9.90e+01	1.38e+02	1.09e+01
4.48e+01	-6.27e+01	1.11e+02	-7.63e+01	1.24e+02	9.55e+01	-3.98e+01	5.85e+01
-6.99e+01	-4.02e+01	-2.34e+01	-7.67e+01	2.66e+01	-3.68e+01	6.61e+01	1.25e+02
-1.09e+02	-4.33e+01	-5.55e+01	8.17e+00	3.02e+01	-2.86e+01	2.44e+00	-9.41e+01
-5.38e+00	5.66e+01	1.73e+02	-3.54e+01	3.23e+01	3.34e+01	-5.81e+01	1.90e+01
7.88e+01	-6.45e+01	1.18e+02	-1.50e+01	-1.37e+02	-3.06e+01	-1.05e+02	3.98e+01
1.97e+01	-7.81e+01	9.72e-01	-7.23e+01	-2.15e+01	8.13e+01	6.37e+01	5.90e+00

  la prima riga dei coefficienti DCT calcolati coincide con quelli pubblicati, a meno di differenze di arrotondamento.

- DCT 1D della prima riga:

```
DCT 1D della prima riga (type=2, norm='ortho'):
nostra : [ 401.99    6.6   109.17 -112.79   65.41  121.83  116.66   28.8 ]
attesa : [ 401.     6.6  109.  -112.    65.4  121.   116.    28.8]
max |diff| = 9.902e-01
```

  differenza massima con i valori del PDF pari a **≈ 10^−0**, imputabile agli arrotondamenti del materiale fornito.

3.2 Benchmark DCT: tempi ed efficienza

Dopo aver verificato la correttezza numerica, il passo successivo è valutare le prestazioni delle due implementazioni della trasformata discreta del coseno bidimensionale (DCT2):

- la versione **naïve**, basata sulla moltiplicazione con matrici di base, con complessità teorica **O(N^3)**;
- la versione **fast**, basata su FFT (SciPy), con complessità **O(N^2logN)**;

Il test è stato condotto generando matrici casuali **N×N**, con **N variabile da 4 a 1024**. Per ogni dimensione sono stati misurati i tempi di esecuzione medi, ripetendo più iterazioni per ridurre la variabilità dovuta al sistema operativo.

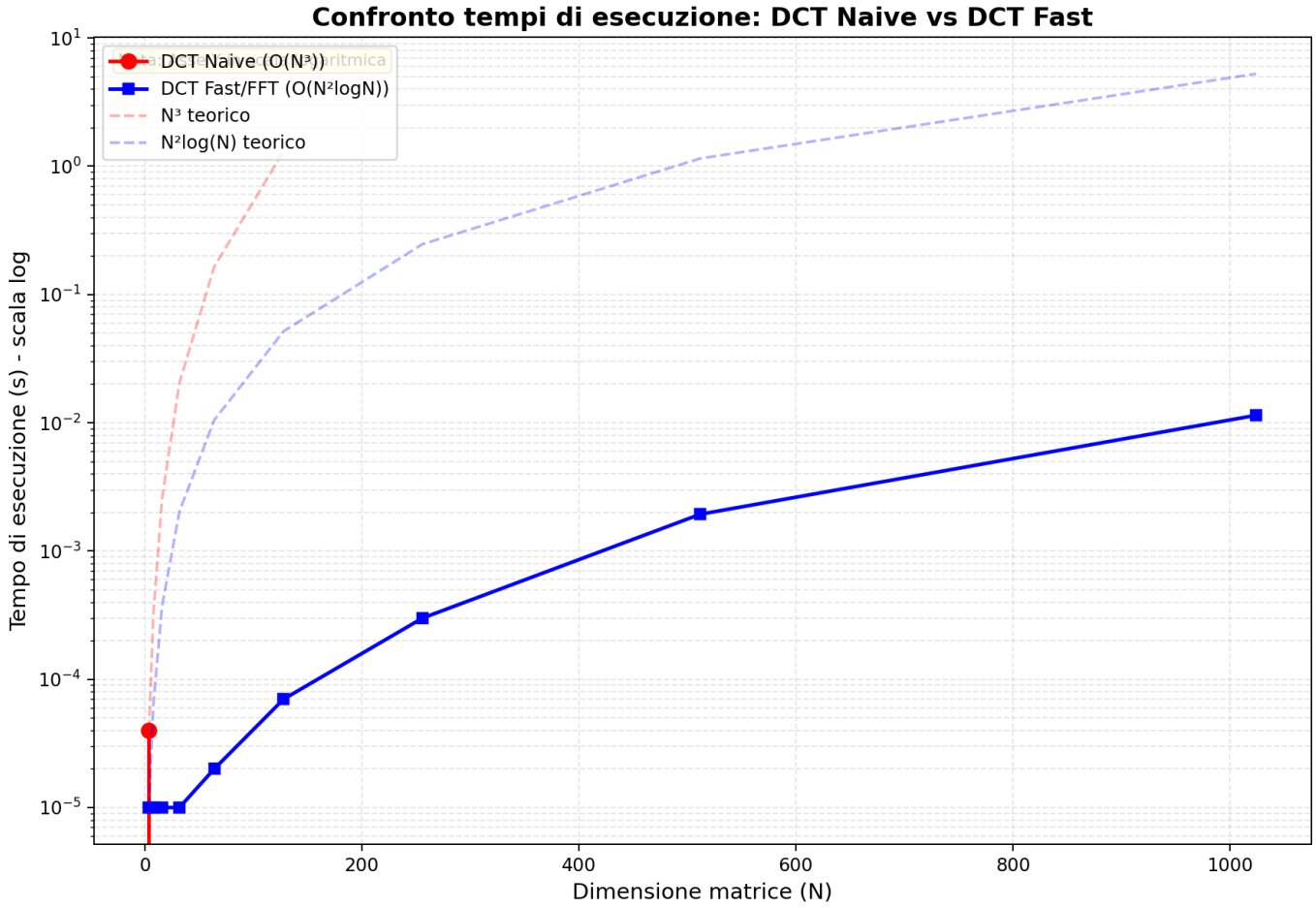
```
(.venv) PS C:\Users\aleqs\mcs_progetto_2> python parte1_dct_comparison.py
============================================================
PARTE 1: BENCHMARK DCT IMPLEMENTATIONS
============================================================

 Testing DCT Naive (questo richiederà qualche minuto)...
   Dimensione 4x4...
      Naive: 0.000040s | Fast: 0.000010s
   Dimensione 8x8...
      Naive: 0.000000s | Fast: 0.000010s
   Dimensione 16x16...
      Naive: 0.000000s | Fast: 0.000010s
   Dimensione 32x32...
      Naive: 0.000000s | Fast: 0.000020s
   Dimensione 64x64...
      Naive: 0.000000s | Fast: 0.000030s
   Dimensione 128x128...
      Naive: 0.000000s | Fast: 0.000080s

 Testing DCT Fast per dimensioni maggiori...
   Dimensione 4x4...
      Fast: 0.000010s
   Dimensione 8x8...
      Fast: 0.000010s
   Dimensione 16x16...
      Fast: 0.000010s
   Dimensione 32x32...
      Fast: 0.000010s
   Dimensione 64x64...
      Fast: 0.000020s
   Dimensione 128x128...
      Fast: 0.000070s
   Dimensione 256x256...
      Fast: 0.000300s
   Dimensione 512x512...
      Fast: 0.001936s
   Dimensione 1024x1024...
      Fast: 0.011425s
```

Osservazioni principali:

- L'output del terminale mostra chiaramente come già per matrici di piccola dimensione (es. 32×32) la DCT naïve diventi significativamente più lenta della fast.

- La **fast DCT** rimane molto efficiente fino a **1024×1024**, con tempi sempre sotto il centesimo di secondo.

- La **naïve** è stata fermata a **128×128** per l'elevato costo computazionale (tempi non più gestibili).



Il grafico in scala logaritmica mette in evidenza l'andamento teorico e sperimentale:

- la curva **rossa** (naïve) cresce rapidamente in modo cubico (**N^3**);
- la curva **blu** (fast) cresce molto più lentamente, seguendo **N^2 log N**;

Conclusione:

- La **DCT naïve** ha valore puramente didattico (per mostrare la costruzione matriciale), ma **non è utilizzabile in pratica**;
- La **DCT fast** è quella effettivamente **usata nei sistemi reali** (ad esempio JPEG), perché **consente la compressione anche su immagini di grandi dimensioni**;

3.3 Compressione DCT delle immagini

In questa sezione si analizza il comportamento della trasformata discreta del coseno (DCT) applicata a immagini reali.

Attraverso l'interfaccia grafica sviluppata, è possibile caricare immagini BMP, scegliere la dimensione del blocco **F×F** e il parametro di soglia **d** che determina quanti coefficienti in frequenza vengono mantenuti.

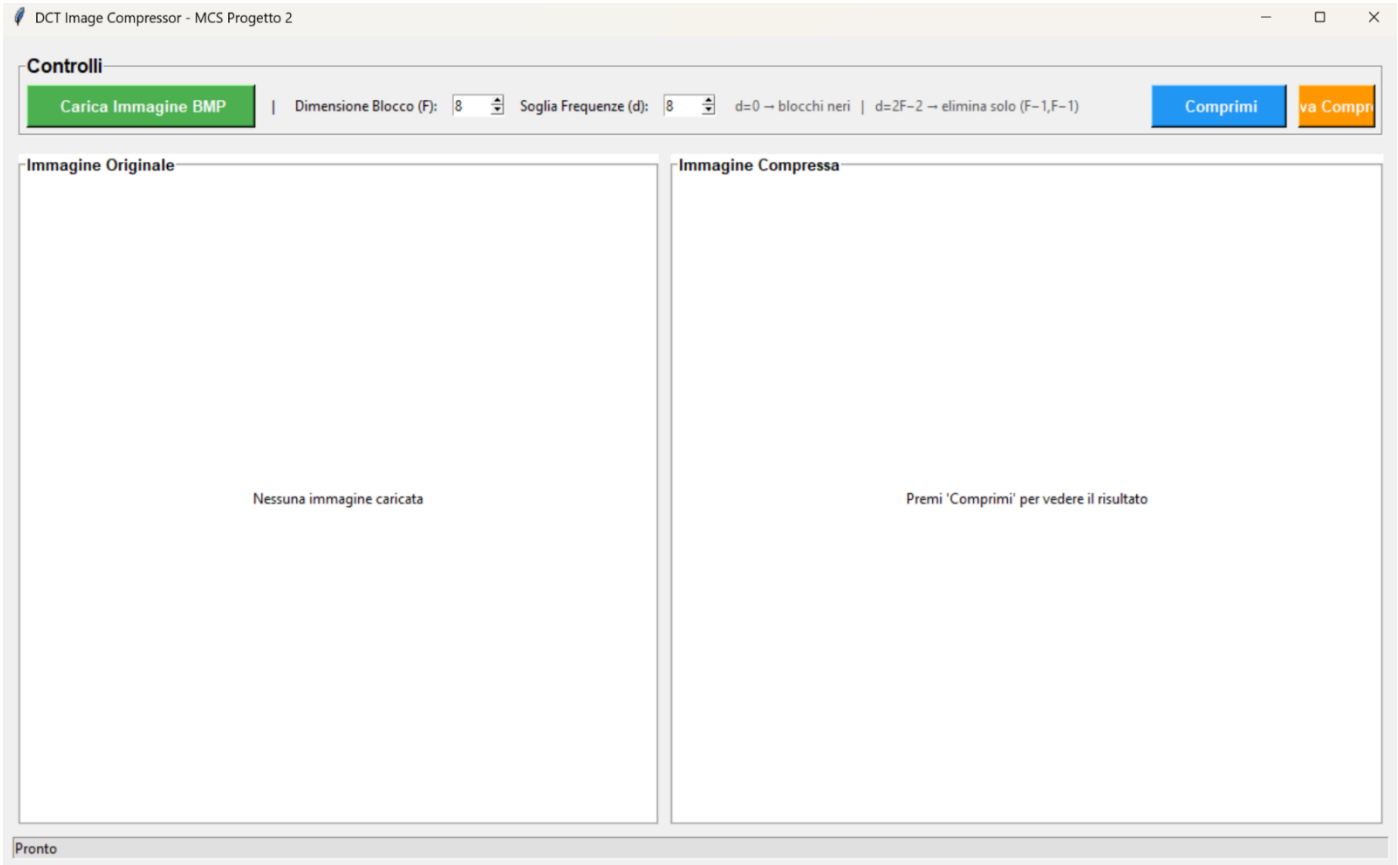
L'obiettivo è valutare l'impatto di tali parametri sulla qualità visiva (misurata con PSNR) e sul grado di compressione, osservando come varia la fedeltà della ricostruzione al crescere della perdita di informazione.

Nei paragrafi seguenti vengono presentati dapprima i risultati sperimentali ottenuti su immagini di test (3.3.1), e successivamente una visualizzazione grafica del meccanismo di taglio delle frequenze (3.3.2), utile per interpretare i fenomeni osservati.

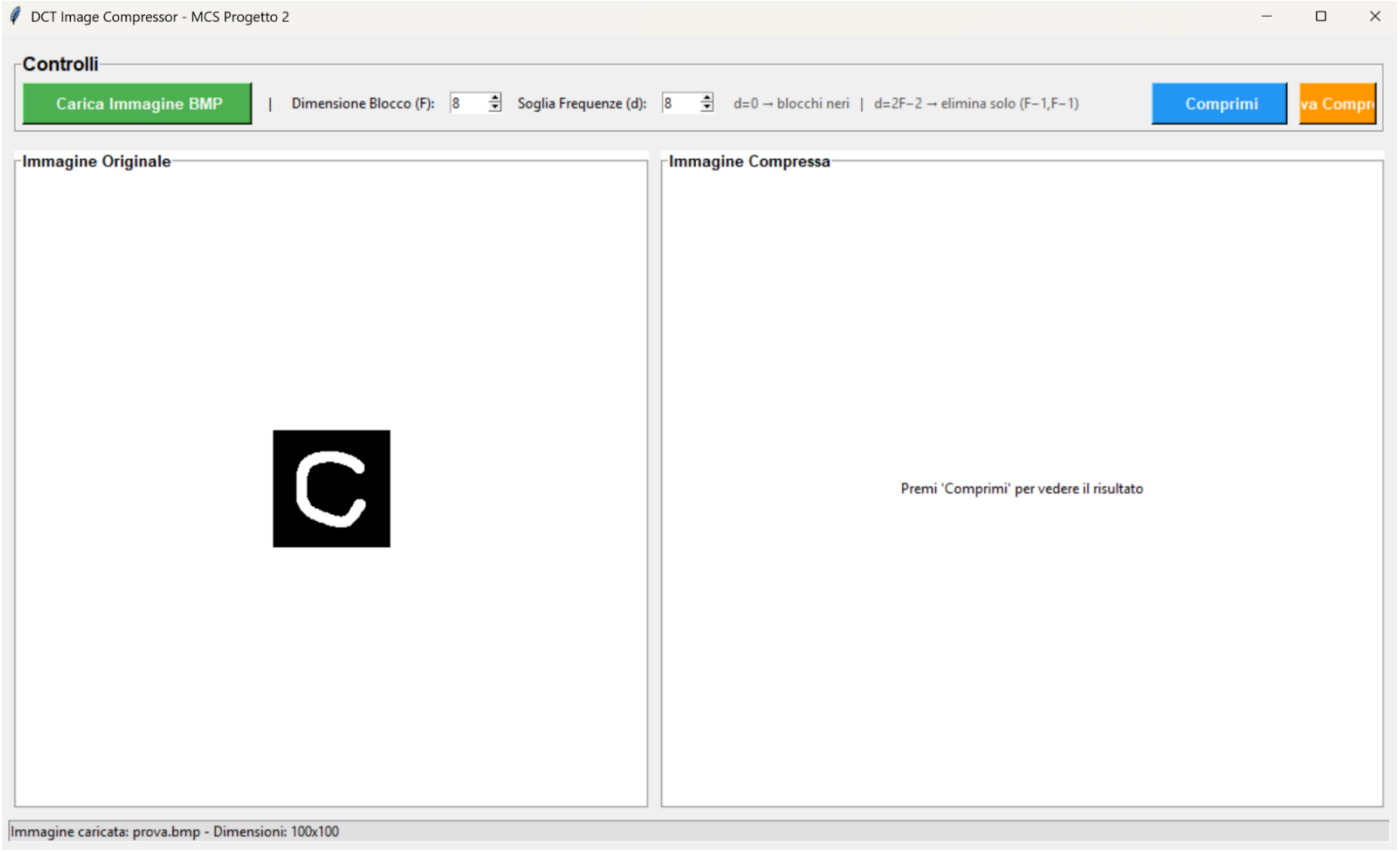
3.3.1 Risultati su immagini di test

In questa prima analisi sperimentale vengono mostrati i risultati su alcune immagini di esempio, osservando visivamente come variano la qualità e il grado di compressione al variare dei parametri **F** e **d**.

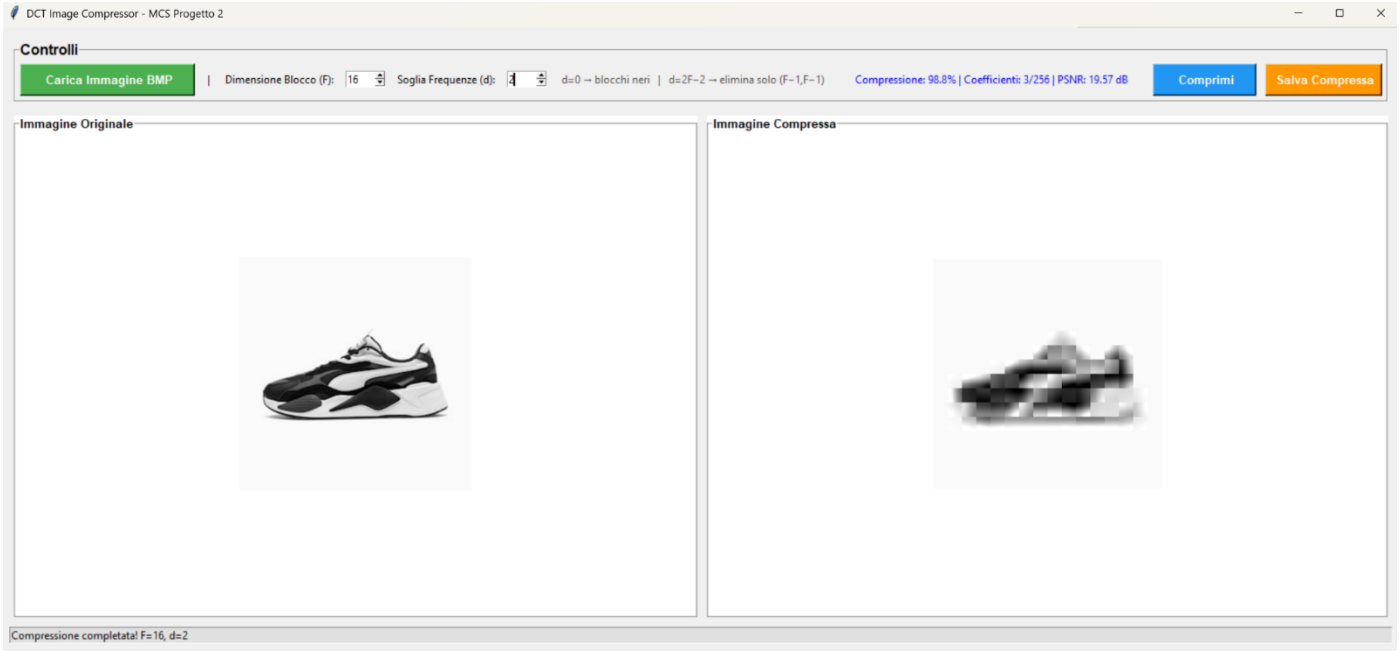
- Interfaccia vuota: il programma permette di caricare un'immagine BMP, impostare i parametri (F, d) e vedere a sinistra l'originale e a destra la compressa.



- Immagine caricata: questa schermata illustra l'input di partenza, che serve come riferimento per i confronti successivi.

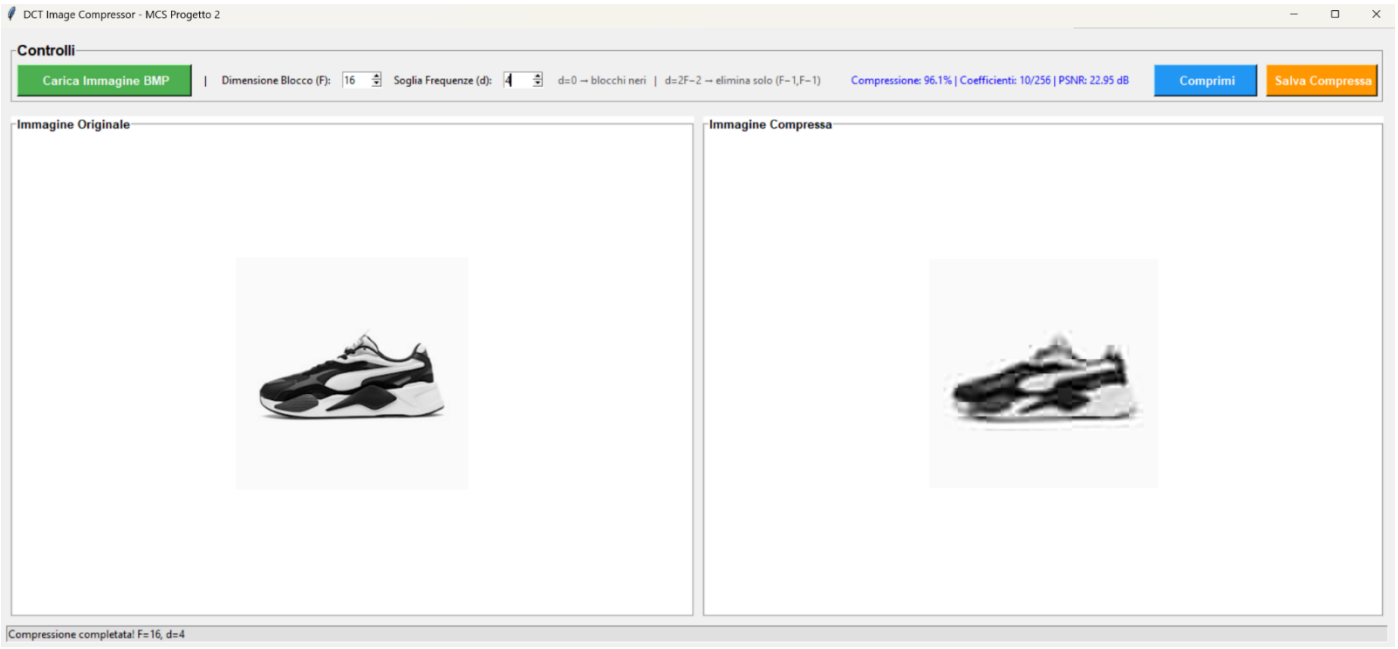


- Configurazione 1: **F = 16, d = 2**



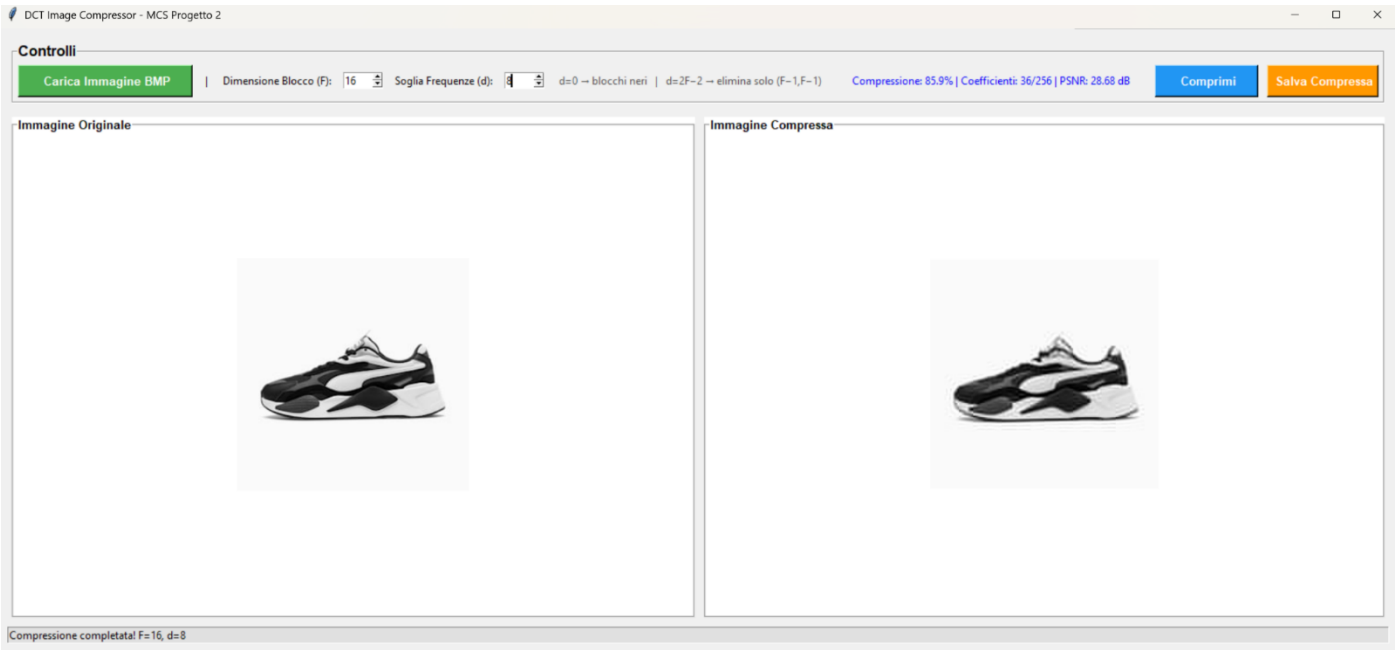
Compressione: 98,8% (3 coefficienti su 256)
PSNR: ~19,57 dB
Osservazioni: l'immagine compressa è molto degradata e quasi irriconoscibile. Con un numero così ridotto di coefficienti mantenuti, vengono preservate solo le frequenze molto basse, che catturano la struttura grossolana ma non i dettagli.

- Configurazione 1: **F = 16, d = 4**



Compressione: 96,1% (10 coefficienti su 256)
PSNR: ~22,95 dB
Osservazioni: qualità leggermente migliore rispetto al caso precedente, ma i dettagli della scarpa restano molto sfocati. Si riconosce la forma generale, ma non le texture fini.

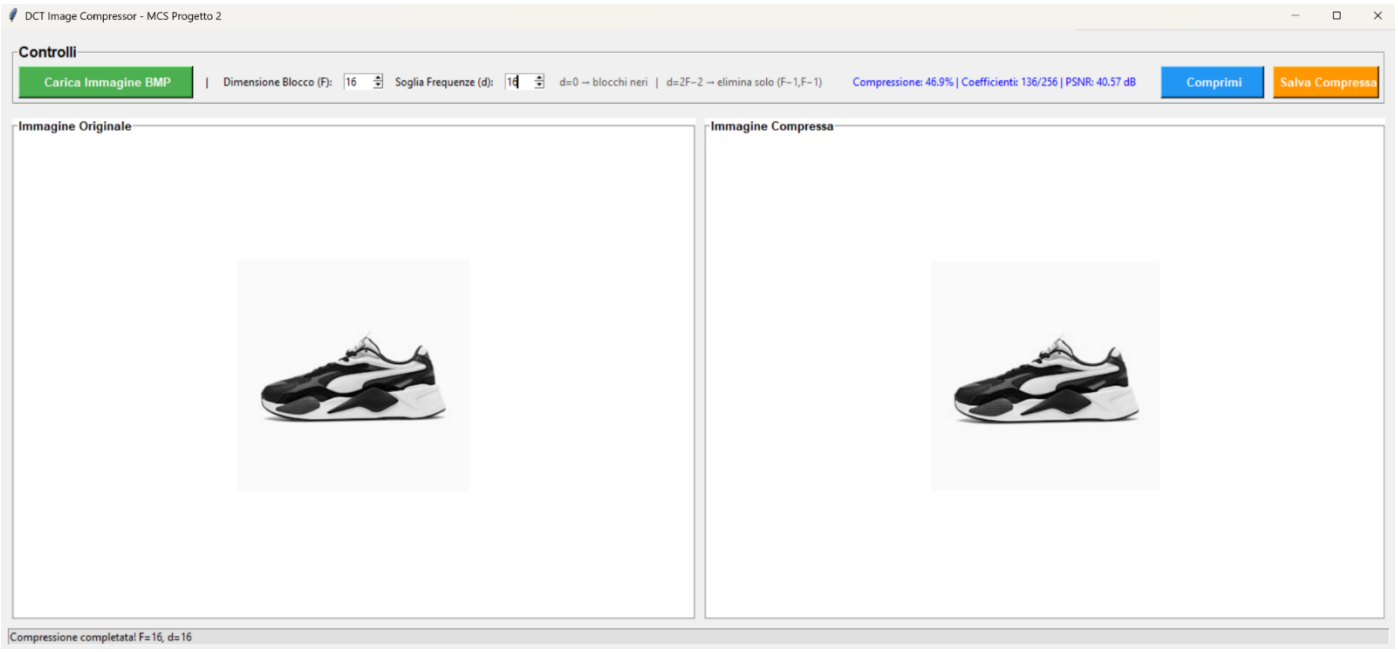
- Configurazione 1: **F = 16, d = 8**



Compressione: 85,9% (36 coefficienti su 256)
PSNR: ~28,68 dB

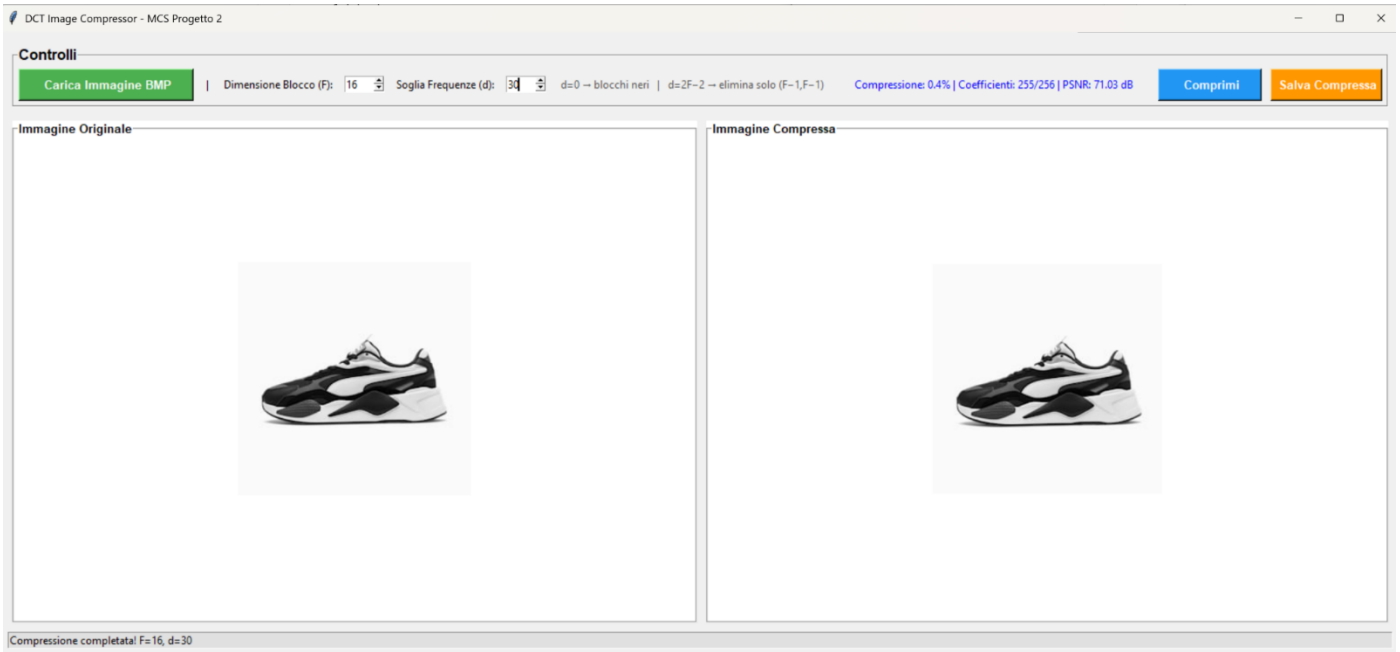
Osservazioni: compromesso accettabile tra qualità e compressione. L'immagine compressa mantiene una buona leggibilità della scarpa, con perdita percettibile solo nei dettagli più fini.

- Configurazione 1: **F = 16, d = 16**



Compressione: 46,9% (136 coefficienti su 256)
PSNR: ~40,57 dB
Osservazioni: la qualità è quasi indistinguibile dall'originale. Si tratta di una compressione molto leggera, con rapporto qualità/dimensioni ottimale.

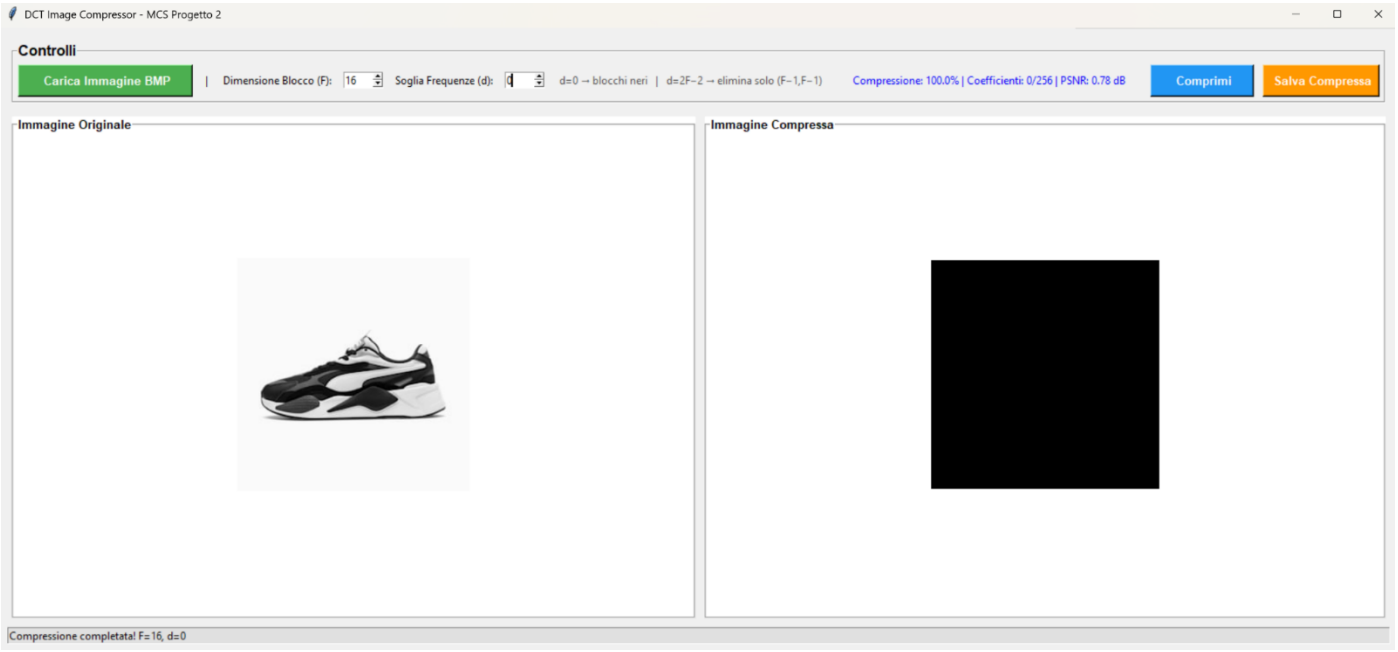
- Configurazione 1: **F = 16, d = 30**



Compressione: 0,4% (255 coefficienti su 256)

PSNR: ~71,03 dB
Osservazioni: l'immagine compressa è identica all'originale, poiché praticamente tutti i coefficienti sono stati mantenuti. In questo caso la compressione è minima, quindi poco utile in termini di riduzione dimensionale.

- Configurazione 1: **F = 16, d = 0**



Compressione: 100% (0 coefficienti su 256)
PSNR: ~0,78 dB
Osservazioni: viene eliminata ogni informazione, ottenendo un blocco nero. È il caso limite che mostra cosa succede se non si mantengono coefficienti.

3.3.2 Visualizzazione del taglio delle frequenze

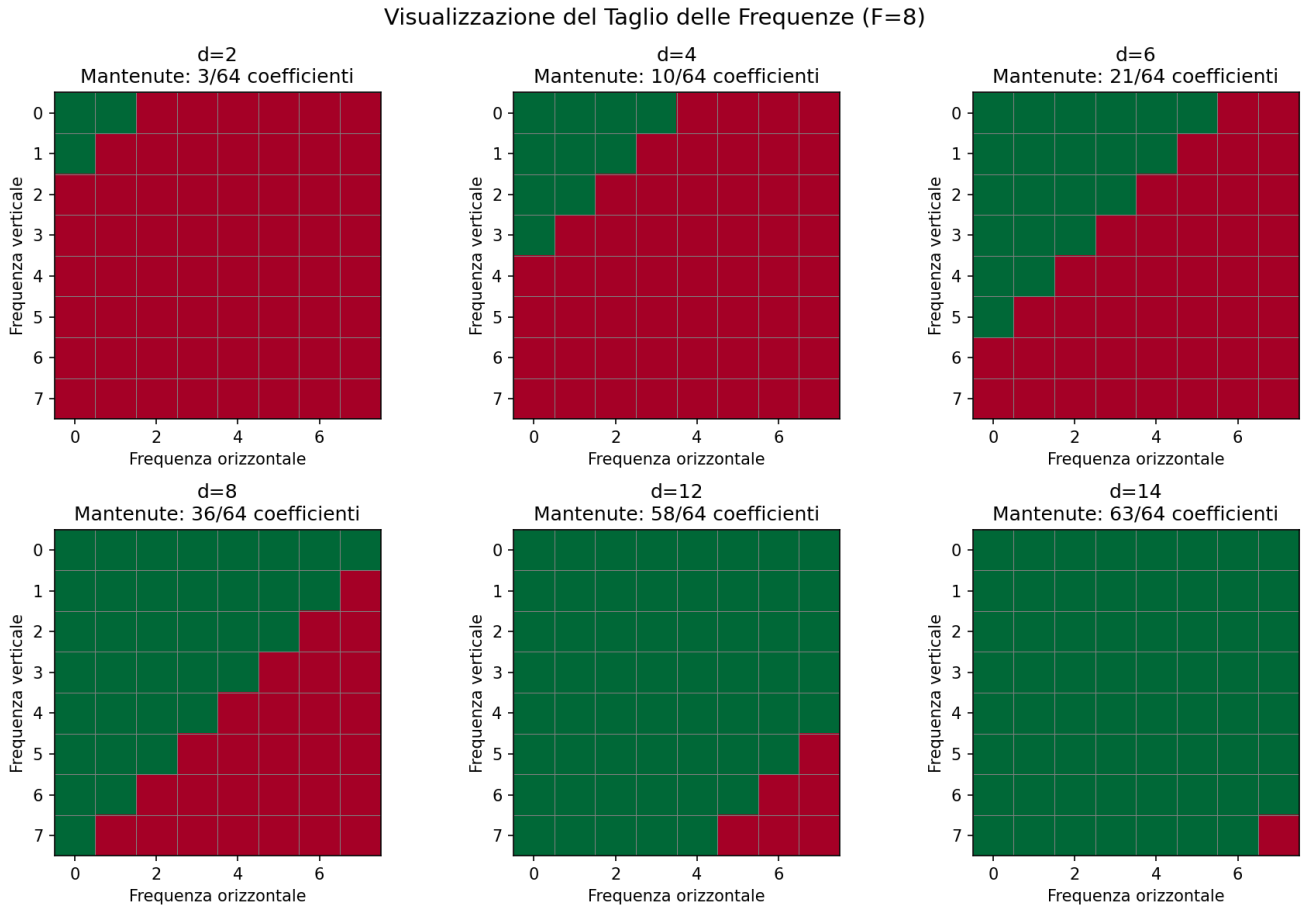
Per interpretare meglio i risultati appena osservati, viene ora mostrata una rappresentazione grafica dei coefficienti DCT mantenuti (in verde) ed eliminati (in rosso) in funzione della soglia **d**.

La figura, riferita a blocchi di dimensione **F = 8**, permette di capire perché al crescere di d l'immagine recuperi progressivamente qualità: aumentando il numero di frequenze preservate, si mantengono più dettagli visivi.

Quindi, ogni blocco **8×8** dell'immagine originale viene trasformato in frequenze (assi orizzontale e verticale della griglia).

- I coefficienti **in alto a sinistra** rappresentano le **basse frequenze**, che contengono l'informazione visiva più importante (forma, intensità generale).

- I coefficienti **in basso a destra** rappresentano le **alte frequenze**, cioè i dettagli fini e il rumore.



Analisi dei casi:

- **d = 2 (3/64 coefficienti mantenuti)**
  Solo le frequenze più basse vengono preservate. L'immagine risultante appare molto sfocata, con struttura generale ma priva di dettagli.
- **d = 4 (10/64 coefficienti mantenuti)**
  Aggiungendo qualche frequenza in più, si recupera un po' di dettaglio, ma resta una perdita evidente.
- **d = 6 (21/64 coefficienti mantenuti)**
  La qualità migliora sensibilmente: la struttura dell'immagine è chiara, ma con meno precisione nelle texture.
- **d = 8 (36/64 coefficienti mantenuti)**
  Buon compromesso: circa metà dei coefficienti vengono salvati, garantendo un equilibrio tra qualità visiva e compressione.
- **d = 12 (58/64 coefficienti mantenuti)**
  Quasi tutte le frequenze sono mantenute: l'immagine compressa è molto simile all'originale, con perdita quasi impercettibile.

- **d = 14 (63/64 coefficienti mantenuti)**
  Compressione minima: l'immagine è praticamente indistinguibile dall'originale.

In conclusione, la visualizzazione del taglio delle frequenze conferma sperimentalmente quanto osservato nei test: **basse soglie d** producono **immagini molto compresse ma degradate**, mentre **valori alti preservano la qualità riducendo però l'efficacia della compressione**.

3.3.3 Esperimenti finali e valutazione complessiva

A completamento dell'analisi, sono stati eseguiti esperimenti sistematici su un insieme di immagini di test, variando i parametri **F** e **d** in modo automatico.

Le immagini selezionate coprono diversi scenari, così da testare la compressione DCT su contenuti eterogenei:

- **bridge.bmp** (4049×2749): immagine ad alta risoluzione con molti dettagli fini e strutture geometriche complesse.
- **cathedral.bmp** (3008×2000): immagine architettonica con contrasti e texture ricche.
- **gradient.bmp** (600×1000): immagine artificiale a gradiente uniforme, utile per valutare la resa su transizioni morbide.
- **640x640.bmp** (640×640): immagine sintetica con pattern ad alta frequenza, particolarmente difficile da comprimere.
- **shoe.bmp** (260×260): immagine a risoluzione medio-bassa, rappresentativa di scenari più semplici.

Il programma fornisce per ciascun caso i valori di:

- **Compressione (%)**, cioè la riduzione della quantità di coefficienti mantenuti;
- **PSNR (dB)**, indicatore della qualità visiva;
- **Classificazione qualitativa** della ricostruzione (da *Scarsa* a *Eccellente*).

Tabella di sintesi:

La tabella seguente riporta nel dettaglio i valori per ogni immagine testata, consentendo di osservare in modo sistematico come variano compressione e qualità al variare dei parametri **F** e **d**.

Esempi con F=8 (standard JPEG):

Immagine	d	Compressione %	PSNR (dB)	Qualità
bridge	3	90.6	32.39	Buona
bridge	7	56.2	41.68	Eccellente
bridge	10	23.4	48.90	Eccellente
bridge	14	1.6	68.26	Eccellente
cathedral	3	90.6	33.89	Buona
cathedral	7	56.2	43.34	Eccellente
cathedral	10	23.4	49.97	Eccellente
cathedral	14	1.6	69.15	Eccellente
gradient	3	90.6	54.62	Eccellente
gradient	7	56.2	57.65	Eccellente
gradient	10	23.4	65.39	Eccellente
gradient	14	1.6	100.00	Eccellente
640x640	3	90.6	12.80	Scarsa
640x640	7	56.2	20.99	Accettabile
640x640	10	23.4	32.05	Buona
640x640	14	1.6	46.44	Eccellente
shoe	3	90.6	25.66	Accettabile
shoe	7	56.2	36.72	Buona
shoe	10	23.4	59.12	Eccellente
shoe	14	1.6	76.61	Eccellente

Oltre ai risultati puntuali, il programma produce anche un riepilogo generale:

- N totale di esperimenti: **60**
- PSNR medio: **47,61 dB**
- Migliore risultato: **100 dB** (caso limite, qualità identica all'originale)

```
Statistiche Generali:
 - Numero totale esperimenti: 60
 - PSNR medio: 47.61 dB
 - Migliore PSNR: 100.00 dB
 - Peggiore PSNR: 10.46 dB
```

- Peggiore risultato: **10,46 dB** (qualità scarsa, immagine quasi irriconoscibile)

Conclusioni:

- **Relazione compressione–qualità (PSNR):**

  All'aumentare della compressione (percentuale alta, pochi coefficienti mantenuti) il PSNR cala e la qualità peggiora.

  Viceversa, quando la compressione è minima (molti coefficienti mantenuti), il PSNR cresce e la qualità diventa quasi indistinguibile dall'originale.

  Tutto cio' conferma il compromesso intrinseco tra risparmio di memoria e fedeltà visiva.

- **Ruolo della soglia d:**

  Valori bassi di **d** (es. 3) portano ad alta compressione (~90%) ma qualità scarsa/buona.

  Valori medi di **d** (es. 7–10) rappresentano il miglior compromesso, con compressione accettabile (20–60%) e qualità buona o eccellente.

  Valori alti di **d** (es. 14) producono qualità eccellente ma con compressione minima, quindi poco utile.

- **Dipendenza dall'immagine di input:**

  Immagini semplici o regolari (es. *gradient.bmp*) mantengono un PSNR molto alto anche con forte compressione (più facili da comprimere).

  Immagini complesse con dettagli ad alta frequenza (es. *640x640.bmp*) crollano di qualità già con compressione moderata (più difficili da comprimere).

- **Qualità soggettiva vs metrica PSNR:**

  PSNR > 30 dB corrisponde generalmente a immagini percepite come ***Buone/Eccellenti***, mentre **valori < 20 dB** corrispondono **a qualità scarsa o appena accettabile**.

  La classificazione qualitativa risulta coerente con la metrica numerica.

4. Conclusioni

Il progetto ha permesso di studiare e confrontare due implementazioni della trasformata discreta del coseno bidimensionale (DCT-II), una **naïve** basata su moltiplicazioni matriciali e una **fast** basata su FFT, oltre a sviluppare un compressore a blocchi che applica una maschera di frequenze secondo il criterio **k+ℓ<d**.

I risultati sperimentali hanno mostrato che:

- **Correttezza numerica**:
  Le versioni naïve e fast della DCT2 producono **risultati numericamente equivalenti**, con differenze massime dell'ordine di 10^{-12}. La ricostruzione tramite IDCT2 restituisce il segnale originale con **errore trascurabile**, confermando la correttezza delle implementazioni.
- **Prestazioni**:
  I benchmark hanno evidenziato che la versione naïve, pur utile a fini didattici, diventa rapidamente impraticabile già per matrici di dimensioni modeste (oltre 128×128). Al contrario, la **versione fast** mostra **tempi ridottissimi anche per matrici 1024×1024**, seguendo la complessità attesa **O(N^(2)log N)**.
- **Compressione delle immagini**:
  La compressione a blocchi ha messo in evidenza il classico compromesso tra qualità visiva e grado di compressione:
  - per soglie **basse** (pochi coefficienti mantenuti), la **compressione è molto elevata** (>90%) ma con **qualità scadente** (PSNR < 20 dB);
  - per soglie **intermedie**, si ottiene il **miglior equilibrio**, con PSNR tra 30–40 dB e compressione ancora significativa (20–60%);
  - per soglie **alte**, la **qualità è eccellente** (PSNR > 40 dB), ma la **compressione è minima** e quindi poco vantaggiosa.
- **Dipendenza dai contenuti**:
  Le immagini **semplic**i (es. gradienti) mantengono **buona qualità** anche **con alta compressione**, mentre **immagini complesse** e ricche di dettagli **ad alta frequenza degradano rapidamente**. Ciò conferma l'importanza della natura del contenuto per valutare l'efficacia della compressione.

In conclusione, il progetto ha evidenziato come la **DCT fast sia l'unica implementazione utilizzabile in scenari reali**, e come la compressione DCT mostri un compromesso intrinseco tra riduzione dei dati e fedeltà visiva. Le analisi confermano che **valori medi della soglia d** rappresentano la **scelta ottimale** per **bilanciare qualità** e **risparmio di memoria.**