



Progetto 2 Metodi del Calcolo Scientifico

Compressione di immagini tramite la DCT

AA 2024/2025

di

⌚ Andrea Falbo

a.falbo7@campus.unimib.it

⌚ Ruben Tenderini

r.tenderini@campus.unimib.it

Università degli Studi di Milano-Bicocca

Indice

1	Introduzione	1
1.1	Obiettivo	1
1.2	Contesto applicativo	1
1.3	Strumenti e Linguaggi Utilizzati	1
1.4	Repository	2
2	Parte I: Implementazione e Analisi delle Prestazioni della DCT2	3
2.1	Verifica Iniziale	3
2.2	DCT2 Custom	4
2.3	DCT2 SciPy	5
2.4	Confronto delle Prestazioni	5
2.5	Gestione dati e Modularità	6
2.6	Conclusioni e Sviluppi Futuri	6
3	Parte II: Compressione di Immagini tramite la DCT2	8
3.1	Struttura del Software	8
3.2	Algoritmo di Compressione	9
3.3	Esperimenti su Immagini di Test	10
3.4	160x160	11
3.5	Gradient	13
3.6	Cathedral	15
3.7	Discussione dei Risultati	17
3.8	Conclusioni e Sviluppi Futuri	17
3.9	Sviluppi Futuri	17
A	Codice Parte I	19
A.1	main.py	19
A.2	runner.py	19
A.3	plotter.py	21
A.4	CSVLogger.py	23
A.5	functions.py	24
A.6	constants.py	25
B	Codice Parte II	27
B.1	main.py	27
B.2	gui.py	29

Capitolo 1

Introduzione

1.1 Obiettivo

Lo scopo di questo progetto è di utilizzare l'implementazione della DCT2 in un ambiente open source e di studiare gli effetti di un algoritmo di compressione tipo jpeg sulle immagini in toni di grigio. Il progetto è suddiviso in due parti:

- la prima prevede l'implementazione manuale della DCT2 e un confronto prestazionale con la versione ottimizzata fornita da librerie scientifiche.
- la seconda consiste nello sviluppo di un software per la compressione di immagini tramite eliminazione delle frequenze DCT.

1.2 Contesto applicativo

La compressione delle immagini è una delle applicazioni fondamentali della trasformata DCT, utilizzata in particolare nel formato JPEG. Questo progetto si propone di replicare, semplificandolo, il processo di compressione JPEG, senza utilizzare matrici di quantizzazione, ma limitandosi a manipolare i coefficienti DCT.

1.3 Strumenti e Linguaggi Utilizzati

Tutto il progetto è stato sviluppato in linguaggio **Python** (versione 3.11.11), sfruttando l'ecosistema scientifico open-source. Le librerie principali utilizzate includono:

- **NumPy**: per la gestione di array multidimensionali;
- **SciPy**: in particolare il modulo `scipy.fft`, per l'utilizzo delle trasformate coseno (DCT) e inverse (IDCT);
- **Matplotlib**: per la generazione di grafici e la visualizzazione dei risultati sperimentali;
- **Pillow (PIL)**: per il caricamento e la gestione delle immagini in vari formati;
- **Tkinter**: per la costruzione di un'interfaccia grafica utente.
- **Pandas**: per leggere i dati in formato csv.

Lo sviluppo e i test sono stati eseguiti su una macchina che ospita come sistema operativo Windows 11 e con le seguenti caratteristiche hardware:

- **Processore**: Intel Core i7-10700F, 8 core, 16 thread
- **Memoria RAM**: 32 GB DDR4 3600 MHz

1.4 Repository

Per completezza e trasparenza, tutti i file sorgenti, inclusi codice, grafici e documentazione, sono disponibili pubblicamente nella seguente repository GitHub:

<https://github.com/LilQuacky/dct2-image-compression>

La repository contiene:

- Codice Python per l'implementazione della DCT2 manuale e via libreria;
- Script per il confronto dei tempi di esecuzione e generazione dei grafici;
- Software per la compressione delle immagini eseguibile tramite linea di comando o interfaccia;
- La seguente relazione, la specifica di consegna e il README.

Capitolo 2

Parte I: Implementazione e Analisi delle Prestazioni della DCT2

2.1 Verifica Iniziale

Per garantire che la DCT2 della libreria open source `scipy` usi lo scaling visto a lezione, è stato utilizzato un blocco di test noto 8×8 , fornito come riferimento, per cui si conosce l'output esatto della trasformata:

$$\begin{bmatrix} 231 & 32 & 233 & 161 & 24 & 71 & 140 & 245 \\ 247 & 40 & 248 & 245 & 124 & 204 & 36 & 107 \\ 234 & 202 & 245 & 167 & 9 & 217 & 239 & 173 \\ 193 & 190 & 100 & 167 & 43 & 180 & 8 & 70 \\ 11 & 24 & 210 & 177 & 81 & 243 & 8 & 112 \\ 97 & 195 & 203 & 47 & 125 & 114 & 165 & 181 \\ 193 & 70 & 174 & 167 & 41 & 30 & 127 & 245 \\ 87 & 149 & 57 & 192 & 65 & 129 & 178 & 228 \end{bmatrix}$$

La trasformata DCT2 attesa è:

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

Analogamente, la prima riga del blocco:

$$[231, 32, 233, 161, 24, 71, 140, 245]$$

deve produrre il seguente vettore:

$$[4.01e+02, 6.60e+00, 1.09e+02, -1.12e+02, 6.54e+01, 1.21e+02, 1.16e+02, 2.88e+01]$$

Per verificare la correttezza, è stato implementato un test automatico che:

- Applica la `dctn` di SciPy al blocco completo;
- Applica la `dctn` alla sola prima riga;

- Confronta ciascun risultato con i valori attesi, misurando l'errore massimo assoluto;
- Conferma che tale errore sia inferiore a una soglia di tolleranza personalizzabile.
- Nel nostro caso specifico, abbiamo scelto **TOLERANCE = 10.0**.

Tutti i test sono stati superati con successo, validando l'utilizzo della funzione

```
scipy.fft.dctn(..., type=2, norm='ortho')
```

per i confronti successivi.

2.2 DCT2 Custom

Per comprendere il funzionamento della Trasformata Discreta del Coseno bidimensionale (DCT2), è stata sviluppata una versione custom partendo dalla definizione teorica. In seguito, si è scelto un approccio **separabile**, che prevede l'applicazione della DCT monodimensionale (DCT) prima lungo le righe e poi lungo le colonne di una matrice bidimensionale.

La DCT unodimensionale è stata implementata secondo la seguente formula:

$$X_k = \sqrt{\frac{2}{N}} \cdot \alpha(k) \sum_{n=0}^{N-1} x_n \cdot \cos \left[\frac{\pi}{2N} (2n+1)k \right]$$

dove $\alpha(k) = \frac{1}{\sqrt{2}}$ se $k = 0$, altrimenti 1.

Nello specifico la funzione `dct1d_custom` implementa la formula appena riportata, calcolando la DCT su un vettore monodimensionale di input.

```

1 N = len(signal)
2 result = np.zeros(N)

3
4 def alpha(u): return 1 / np.sqrt(2) if u == 0 else 1
5
6 for u in range(N):
7     sum_val = sum(signal[x] * np.cos((2 * x + 1) * u * np.pi / (2 *
8         N)) for x in range(N))
9     result[u] = np.sqrt(2 / N) * alpha(u) * sum_val
return result

```

Mentre la funzione `dct2_separable` applica `dct1d_custom` a ciascuna riga della matrice e, successivamente, a ciascuna colonna del risultato.

```

1 rows_dct = np.apply_along_axis(dct1d_custom, axis=1, arr=image)
2 return np.apply_along_axis(dct1d_custom, axis=0, arr=rows_dct)

```

Per maggiori dettagli sull'implementazioni delle funzioni riferirsi alla Sezione A.5 dell'Appendice A.

La trasformazione è quindi ottenuta in due passaggi indipendenti, sfruttando la proprietà di separabilità della DCT2. In termini di complessità, questa strategia riduce i costi

computazionali rispetto a un'implementazione diretta della definizione bidimensionale, passando da $O(N^4)$ a $O(N^3)$.

2.3 DCT2 SciPy

Per il confronto prestazionale, è stata utilizzata la funzione `scipy.fft.dctn`, un'implementazione altamente ottimizzata della DCT n-dimensionale basata su FFT e simmetrie:

- La chiamata `dctn(image, type=2, norm='ortho')` applica la DCT bidimensionale ortonormale separatamente lungo ciascun asse.
- Sebbene la funzione accetti array di qualsiasi dimensione, nel caso bidimensionale si comporta come una DCT2 separabile.
- La complessità computazionale è $O(N^2 \log N)$, e quindi più efficiente rispetto alla nostra implementazione custom $O(N^3)$.

2.4 Confronto delle Prestazioni

Per valutare l'efficienza delle due implementazioni, è stato condotto un benchmark su matrici casuali di dimensioni crescenti, tutte di forma $N \times N$, con N scelto come potenza di 2.

In particolare, sono stati testati i seguenti range:

- **Per la DCT2 custom:** $N \in \{8, 16, 32, \dots, 2048\}$;
- **Per la DCT2 di scipy:** $N \in \{8, 16, 32, \dots, 32768\}$.

Per ciascun valore di N , l'esperimento è stato ripetuto per un numero fissato di iterazioni (tramite il parametro `ITERATIONS`), e sono stati calcolati i tempi medi di esecuzione tramite la funzione (`time.perf_counter()`). La varianza dei tempi registrati è risultata trascurabile, per cui un numero anche molto ridotto di ripetizioni avrebbe prodotto risultati sostanzialmente equivalenti.

I dati ottenuti sono stati salvati in formato CSV e visualizzati tramite un grafico semi-logaritmico, riportato in Figura 2.1.

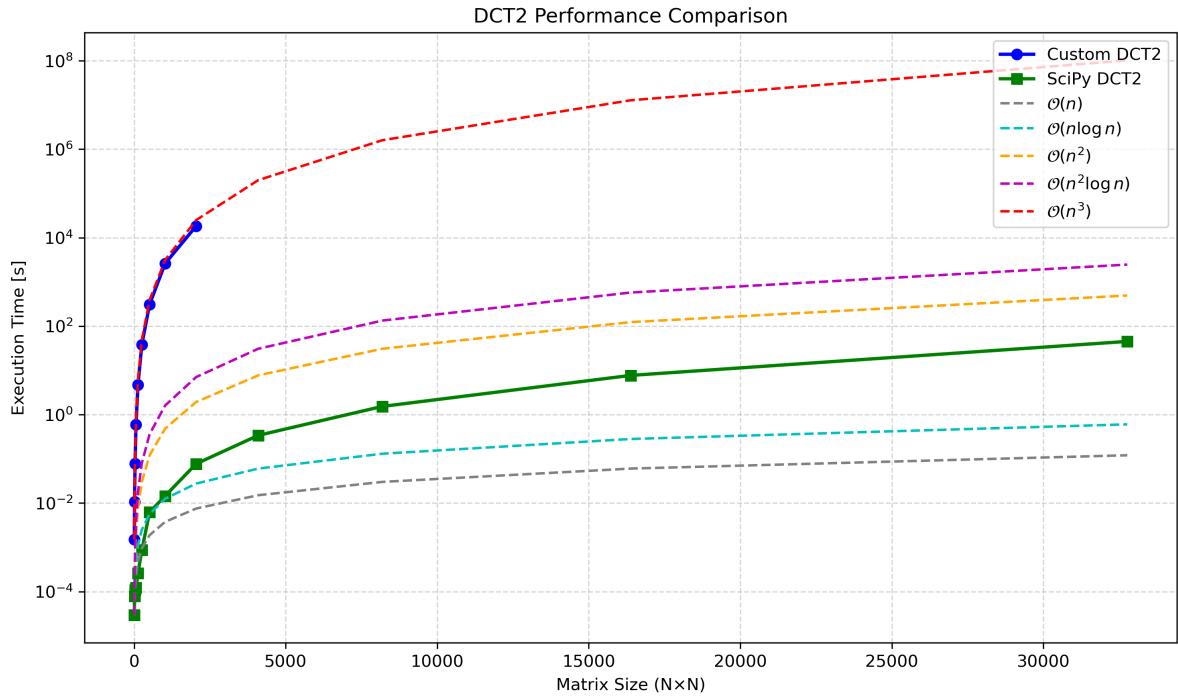


Figura 2.1: Confronto dei tempi di esecuzione: DCT2 custom vs SciPy

Come atteso, la DCT2 custom presenta una scalabilità peggiore rispetto alla versione ottimizzata, con un divario sempre più marcato all'aumentare di N .

2.5 Gestione dati e Modularità

Tutti i moduli del progetto sono stati realizzati con un approccio **modulare e scalabile**, al fine di facilitare il riutilizzo del codice e la possibilità di confrontare più implementazioni.

Il benchmarking è stato gestito tramite una classe dedicata `DCTRunner`, che riceve in input la lista delle funzioni da testare e le dimensioni delle matrici.

Altri componenti ausiliari, come `CSVLogger`, `Plotter` e i file di `constants`, sono stati separati logicamente in moduli distinti, migliorando la leggibilità e la manutenibilità del codice.

Per ogni test eseguito, i dati relativi ai tempi di esecuzione sono stati salvati in file CSV separati. Analogamente, ogni grafico è stato generato e salvato con un nome univoco, evitando la sovrascrittura di risultati precedenti.

Per maggiori informazioni riguardanti il codice sviluppato, riferirsi all'Appendice A.

2.6 Conclusioni e Sviluppi Futuri

La nostra implementazione custom della DCT2 ha permesso di osservare in modo diretto l'impatto della complessità algoritmica sulla scalabilità delle prestazioni. In particolare:

- La DCT2 custom presenta complessità $O(N^3)$ confermata empiricamente. Per matrici 2048×2048 , il tempo di esecuzione ha superato le 5 ore, rendendo impraticabile spingersi oltre.
- Per la DCT2 fornita da `scipy.fft`, invece, è stato necessario proseguire fino a matrici 32768×32768 , in quanto fermandosi a 2048×2048 (come per la versione custom), la differenza rispetto a $O(N^2)$ non sarebbe risultata evidente. Una volta testate sulle nuove matrici, è stata osservata una crescita coerente con la complessità $O(N^2 \log N)$.
- È stato tentato un test con dimensioni fino a 65536×65536 e potenze di 2 superiori, ma è stato impedito dai limiti di memoria del sistema (32 GB di RAM).

Per sviluppi futuri, si prevede:

- l'integrazione di nuove implementazioni DCT (ad esempio GPU-based) nel framework esistente;
- l'esecuzione di benchmark distribuiti su macchine dotate di maggiore RAM e potenza computazionale, per estendere il confronto anche a dimensioni superiori a 32768×32768 .

Capitolo 3

Parte II: Compressione di Immagini tramite la DCT2

3.1 Struttura del Software

Architettura del programma

Il software si compone di due file:

- `main.py`: punto di ingresso del programma, si occupa della logica di compressione;
- `gui.py`: definisce l’interfaccia utente e gestisce le interazioni come il caricamento dell’immagine e la selezione dei parametri.

Interfaccia utente per caricamento immagine e scelta parametri

L’interfaccia, sviluppata con Tkinter, permette all’utente di:

- Caricare un’immagine BMP tramite un file selector;
- Selezionare la cartella in cui salvare le immagini;
- Impostare la dimensione dei blocchi $F \times F$;
- Regolare la soglia d per controllare il grado di compressione;
- Scegliere se mostrare la nuova immagine e l’originale al termine.

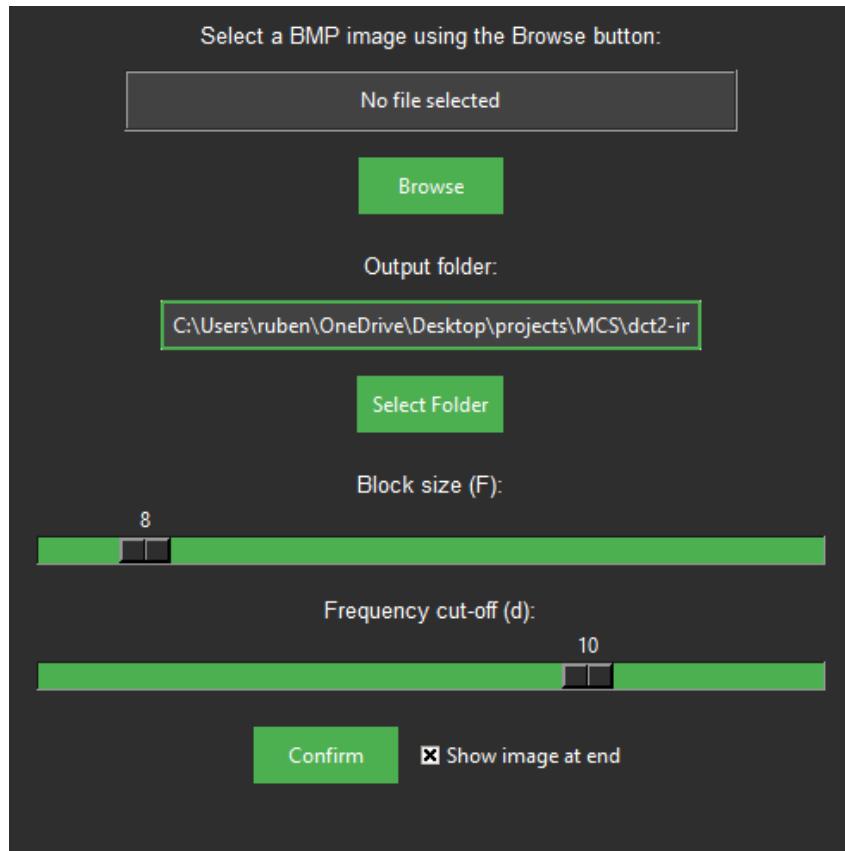


Figura 3.1: Screenshot dell’interfaccia grafica

3.2 Algoritmo di Compressione

Suddivisione in blocchi $F \times F$

L’immagine caricata viene convertita in array NumPy e suddivisa in blocchi quadrati non sovrapposti di dimensione $F \times F$. I blocchi vengono elaborati indipendentemente.

Applicazione della DCT2

A ciascun blocco viene applicata la trasformata discreta del coseno bidimensionale tramite `scipy.fft.dctn` con parametri `type=2` e `norm='ortho'`, la quale assicura una trasformazione ortonormale, compatibile con l’inversa.

Criterio di soglia sulle frequenze $k + \ell \geq d$

Ogni coefficiente $C_{k,\ell}$ della matrice DCT di ciascun blocco viene esaminato: se $k + \ell \geq d$, il coefficiente viene annullato. Questo criterio consente di conservare le frequenze basse, responsabili delle componenti principali dell’immagine, eliminando le alte frequenze (dettagli e rumore).

Ricomposizione dell’immagine

Viene applicata l’inversa DCT2 con `scipy.fft.idctn`, usando gli stessi parametri di normalizzazione. Il risultato viene:

- **Arrotondato** e convertito in interi per ricostruire l'immagine finale.
- **Clippato** all'intervallo $[0, 255]$ per evitare artefatti;

Per concludere, i blocchi elaborati vengono ricomposti per formare l'immagine compressa.

3.3 Esperimenti su Immagini di Test

Le immagini utilizzate negli esperimenti sono state fornite dai docenti come parte integrante del progetto. Si tratta di file BMP in scala di grigi, con risoluzioni e contenuti differenti:

- **Immagini sintetiche**: scacchiera di dimensioni 20×20 , 40×40 , 80×80 , 160×160 , 320×320 , 640×640 .
- **Immagini reali**: `bridge`, `cathedral`, `deer`, `shoe`.
- **Immagini artificiali**: `prova` (lettera "C" bianca su sfondo nero), `gradient` (gradiante orizzontale da nero a bianco).

Per analizzare l'effetto della compressione DCT, sono stati considerati tre valori per la dimensione del blocco DCT, indicata con F :

$$F \in \{8, 16, 32\}$$

Per ciascun valore di F , è stato variato il parametro d , ovvero il numero di coefficienti DCT da mantenere per blocco. I valori scelti per d sono:

$$d \in \{1, 5, 10, 20, 50\}$$

Nel dettaglio:

- Per $F = 8$, i coefficienti selezionabili sono 15, quindi abbiamo usato solo $d \in \{1, 5, 10\}$;
- Per $F = 16$, i coefficienti selezionabili sono 31, quindi abbiamo usato $d \in \{1, 5, 10, 20\}$;
- Per $F = 32$, tutti i valori di d scelti sono validi, quindi $d \in \{1, 5, 10, 20, 50\}$.

Sono state selezionate tre immagini rappresentative, una per ciascuna categoria, al fine di valutare il comportamento dell'algoritmo in scenari differenti:

- `160x160`, come esempio di immagine con **pattern regolare** ad alta frequenza;
- `cathedral`, rappresentativa di un'immagine con **dettagli complessi**;
- `gradient`, come immagine con **variazione graduale** dell'intensità.

Tutte le immagini originali fornite per gli esperimenti erano in formato `.bmp`, ma sono state convertite in formato `.png` per essere correttamente visualizzate all'interno della relazione. Questo perché L^AT_EX, e in particolare l'ambiente Overleaf utilizzato per la stesura del documento, non supporta nativamente il formato `.bmp`. Il formato `.png`, essendo compresso ma lossless, non altera i contenuti visivi e garantisce una compatibilità completa con i pacchetti grafici comunemente utilizzati in L^AT_EX.

3.4 160x160

Analisi Soglia di Taglio

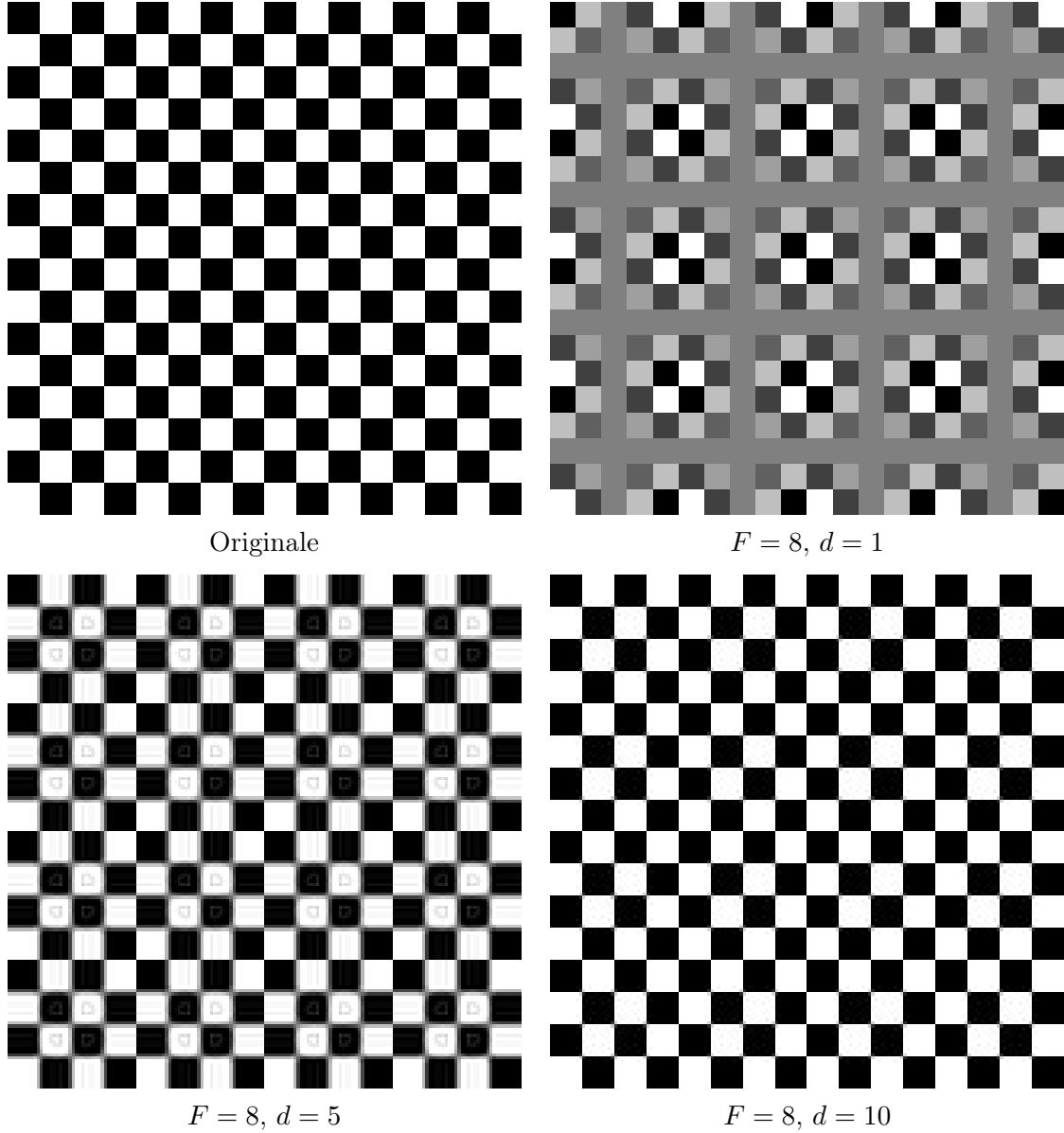


Figura 3.2: Confronto tra compressioni con $F = 8$ e diversi valori di d sull’immagine 160x160.

Mantenendo costante la dimensione dei macro-blocchi ($F = 8$), si osserva come la variazione della soglia di taglio d influenzi significativamente la qualità dell’immagine ricostruita.

- Con $d = 1$, la soglia molto bassa comporta un taglio drastico delle frequenze, mantenendo solo i coefficienti DCT di ordine più basso e risultando in un’immagine fortemente degradata.
- Con $d = 5$, si ottiene un migliore bilanciamento tra compressione e qualità: l’immagine mantiene le caratteristiche principali del pattern a scacchiera, pur presentando ancora alcuni artefatti di compressione. È possibile osservare il fenomeno di Gibbs, visibile come ondulazioni nei pressi dei bordi netti tra i blocchi.

- Aumentando la soglia a $d = 10$, si conserva un numero maggiore di coefficienti DCT, risultando in una ricostruzione di qualità superiore con dettagli più definiti. Si osserva come il bianco subisca degradazioni maggiori rispetto ai blocchi neri, che risultano parecchio simili agli originali.

Analisi Ampiezza Finestre

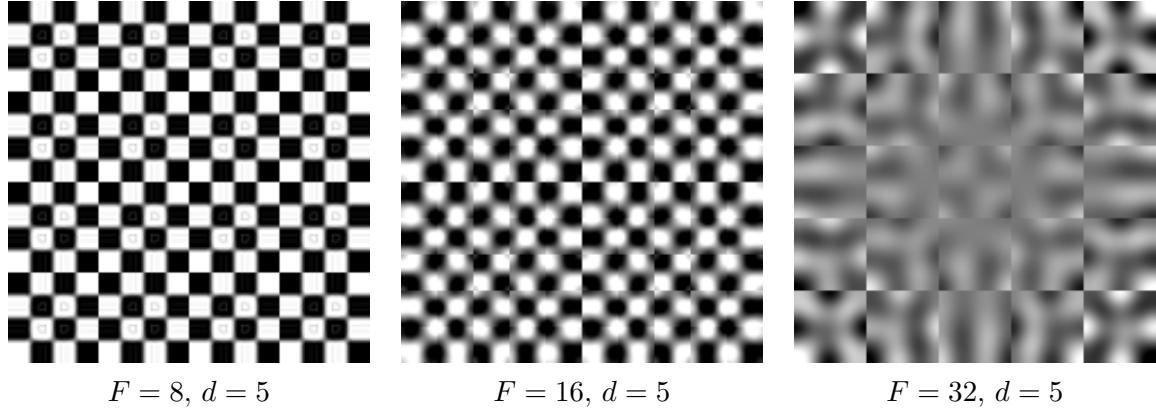


Figura 3.3: Confronto tra compressioni con $d = 5$ e diversi valori di F sull'immagine 160x160.

Mantenendo ora costante la soglia di taglio ($d = 5$), l'analisi dell'effetto della dimensione dei macro-blocchi F rivela comportamenti interessanti:

- Con $F = 8$, l'immagine mantiene una buona definizione del pattern a scacchiera, anche se risultano visibili artefatti di compressione.
- Aumentando a $F = 16$, si osserva una evidente perdita di nitidezza, con i contorni che diventano meno definiti e l'introduzione di lievi sfumature nelle transizioni tra blocchi adiacenti.
- Con $F = 32$, l'immagine diventa irriconoscibile: il pattern a scacchiera perde la sua natura binaria e assume un aspetto sfocato con transizioni graduali tra le regioni. Le zone bianche e nere si fondono in tonalità intermedie di grigio, compromettendo significativamente la riconoscibilità del pattern originale.

3.5 Gradient

Analisi Soglia di Taglio

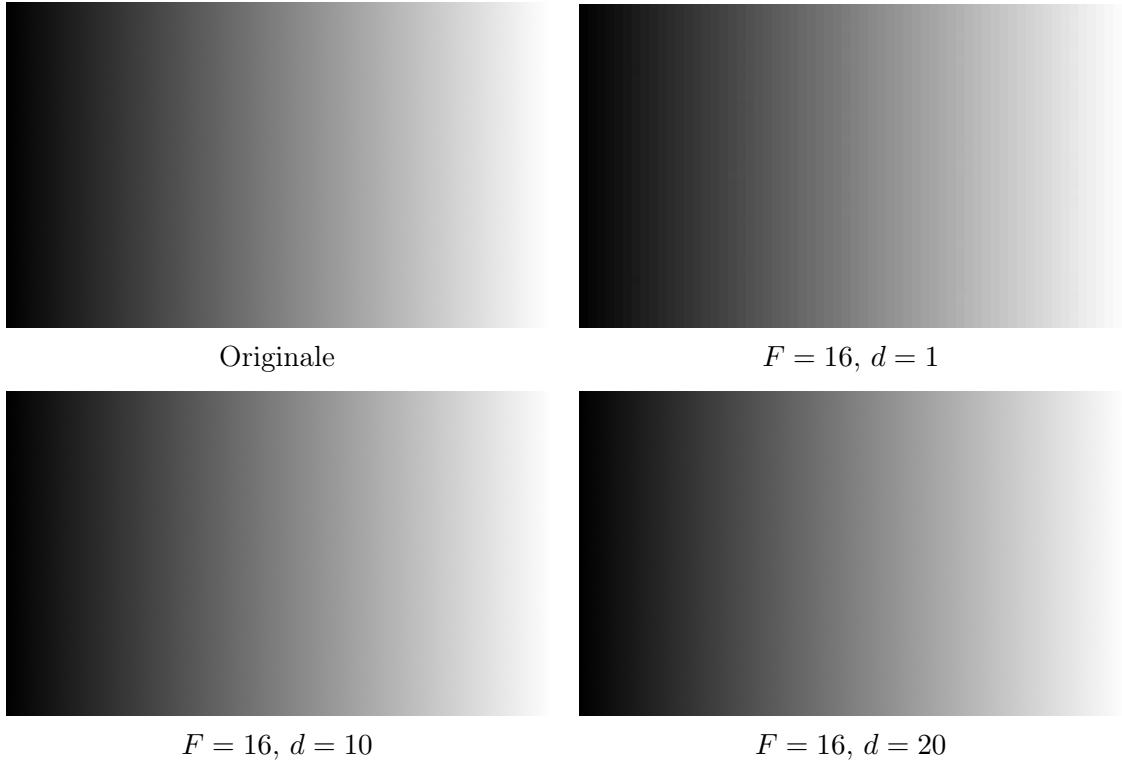


Figura 3.4: Confronto tra compressioni con $F = 16$ e diversi valori di d sull’immagine gradient.

Per immagini con variazioni tonali graduali come i gradienti, gli effetti della compressione DCT risultano molto meno percettibili rispetto ai pattern ad alto contrasto. Mantenendo costante $F = 16$ e variando la soglia di taglio d , si osserva:

- Con $d = 1$ è facilmente osservabile la blocchettatura in colonne dell’immagine.
- Aumentando progressivamente da $d = 10$ a $d = 20$, le differenze visive rispetto all’immagine originale diventano quasi impercettibili, indicando che per contenuti a basso contrasto anche soglie di taglio moderate sono sufficienti per preservare la qualità percettiva.

Analisi Ampiezza Finestre

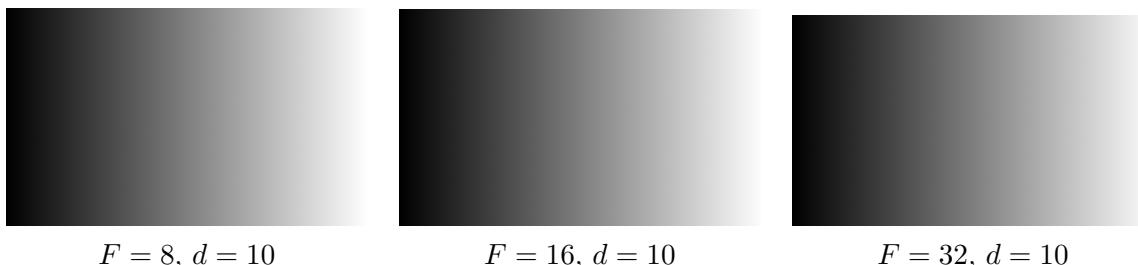


Figura 3.5: Confronto tra compressioni con $d = 10$ e diversi valori di F sull’immagine gradient.

Analogamente, la variazione della dimensione dei macro-blocchi F su contenuto gradiente conferma la robustezza di questo tipo di immagini alla compressione DCT. Le tre configurazioni testate producono risultati visivamente indistinguibili, con il gradiente che mantiene la sua smoothness e continuità tonale indipendentemente dalla dimensione dei blocchi utilizzati per la trasformata.

3.6 Cathedral

Analisi Soglia di Taglio

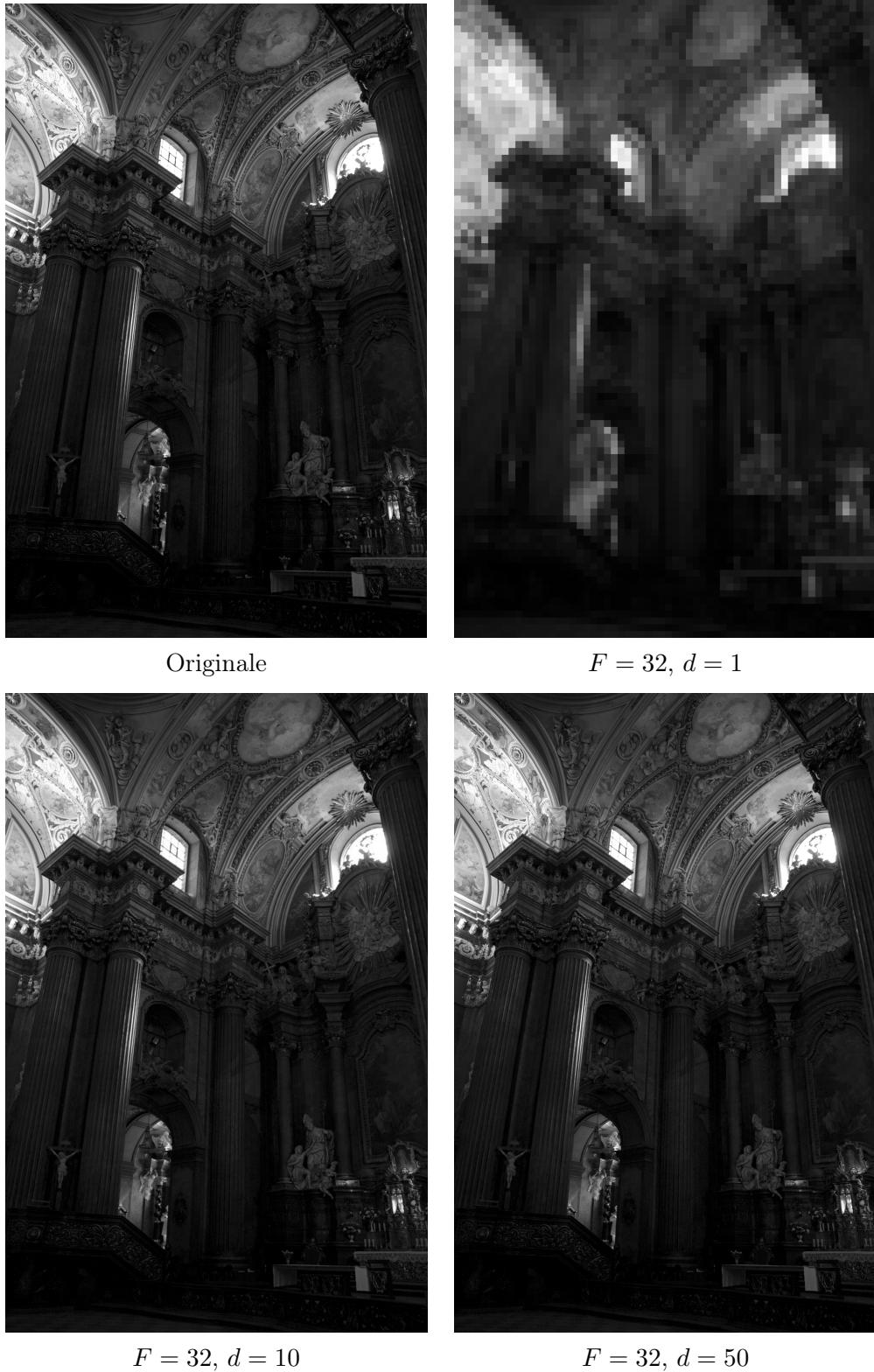


Figura 3.6: Confronto tra compressioni con $F = 32$ e diversi valori di d sull'immagine *cathedral*.

L'applicazione della compressione DCT su un'immagine reale con ricchi dettagli architettonici rivela comportamenti diversi rispetto ai contenuti sintetici precedentemente analizzati:

- Con $d = 1$, l'immagine subisce una degradazione drastica: i dettagli fini degli ornamenti, delle colonne e degli affreschi vengono completamente persi, sostituiti da grossolani blocchi che conferiscono un aspetto fortemente pixelato e rendono difficile il riconoscimento delle strutture architettoniche.
- L'aumento della soglia a $d = 10$ produce un miglioramento significativo della qualità, con il recupero della maggior parte dei dettagli strutturali, pur mantenendo ancora una leggera perdita di nitidezza nei particolari più fini.
- Con $d = 50$, l'immagine raggiunge una qualità molto elevata, praticamente indistinguibile dall'originale, dimostrando come per contenuti ricchi di dettagli sia necessario conservare un numero considerevole di coefficienti DCT per mantenere la fedeltà visiva.

Analisi Ampiezza Finestre



Figura 3.7: Confronto tra compressioni con $d = 10$ e diversi valori di F sull'immagine *cathedral*.

La variazione della dimensione dei macro-blocchi F mostra differenze più sottili ma comunque percettibili:

- Con $F = 8$, l'immagine mantiene una buona definizione dei dettagli architettonici, con bordi relativamente netti e una preservazione accettabile delle texture delle superfici decorate.
- Passando a $F = 16$, si osserva una leggera riduzione della nitidezza generale, particolarmente evidente nei dettagli più fini come gli ornamenti e le decorazioni, che perdono parte della loro definizione pur rimanendo riconoscibili.

- Con $F = 32$, la perdita di dettaglio diventa più marcata: le texture si appiattiscono e i contorni assumono un aspetto leggermente più morbido, pur mantenendo un livello di qualità sufficientemente buono.

3.7 Discussione dei Risultati

L’analisi sperimentale condotta su tre tipologie di immagini rappresentative ha evidenziato come l’efficacia della compressione DCT dipenda fortemente dalle caratteristiche del contenuto dell’immagine e dalla configurazione dei parametri F e d .

- Le **immagini sintetiche** con pattern ad alto contrasto (come la scacchiera 160×160) si sono rivelate particolarmente vulnerabili alla compressione, mostrando artefatti di blocchettatura evidenti già con soglie di taglio moderate.
- Al contrario, le **immagini con variazioni graduali** (come il gradiente) hanno dimostrato una notevole robustezza, mantenendo qualità visiva accettabile anche con parametri di compressione aggressivi.
- Le **immagini reali** (cathedral) hanno presentato un comportamento intermedio, richiedendo soglie di taglio più elevate per preservare i dettagli.

L’analisi ha confermato l’esistenza di un trade-off fondamentale tra efficienza di compressione e qualità dell’immagine ricostruita. Il parametro d si è dimostrato critico per la qualità finale: valori troppo bassi ($d \leq 1$) producono degradazioni inaccettabili per la maggior parte delle applicazioni, mentre valori intermedi ($d = 5-10$) offrono un compromesso ragionevole per contenuti non critici. Il parametro F influisce principalmente sull’efficienza computazionale e sulla capacità di preservare dettagli locali: blocchi piccoli ($F = 8$) mantengono meglio i dettagli ma richiedono maggiori risorse computazionali, mentre blocchi grandi ($F = 32$) sono più efficienti ma tendono a "spalmare" le variazioni locali.

3.8 Conclusioni e Sviluppi Futuri

La seconda parte del progetto ha fornito un’implementazione funzionale di un **sistema di compressione delle immagini** che replica, in forma semplificata, i principi alla base del formato JPEG. L’analisi sperimentale condotta ha evidenziato l’importanza del contenuto dell’immagine e della configurazione dei parametri della DCT.

3.9 Sviluppi Futuri

Il lavoro svolto apre diverse possibilità di estensione e miglioramento che potrebbero costituire oggetto di futuri sviluppi. Tra i tanti, uno è stato già apportato: nonostante l’applicativo sia stato sviluppato su richiesta per immagini in scala di grigi, esso è perfettamente in grado di gestire immagini a colori, trattando separatamente i tre canali RGB durante le fasi di compressione e ricostruzione. Altri sviluppi futuri potrebbero concentrarsi su:

- **Ottimizzazione adattiva dei parametri:** Sviluppo di algoritmi per la selezione automatica dei parametri F e d basata sulle caratteristiche locali dell’immagine.

- **Metriche di qualità oggettive:** Implementazione di metriche standardizzate di valutazione della qualità, come il Peak Signal-to-Noise Ratio (PSNR), per una caratterizzazione quantitativa delle performance di compressione.

Appendice A

Codice Parte I

A.1 main.py

```
1 from dct2_performance.utils.functions import dct2_separable,
2     dct2_scipy, test_correctness_scipy
3 from dct2_performance.utils.plotter import PerformancePlotter
4 from dct2_performance.utils.runner import DCTRunner
5
6 def main():
7     """
8     Main function to run the DCT2 performance benchmark.
9     """
10    #test_correctness_scipy()
11
12    runner = DCTRunner(
13        [dct2_separable, dct2_scipy],
14        [8, 16, 32, 64, 128, 256, 512],
15        "benchmark/"
16    )
17    log_file = runner.run()
18    plotter = PerformancePlotter(log_file)
19    plotter.save_performance_plot(dct2_scipy.__name__)
20
21
22 if __name__ == "__main__":
23     main()
```

A.2 runner.py

```
1 from typing import Collection, Callable
2
3 import numpy as np
4 import time
5
6 from dct2_performance.utils.CSVLogger import CSVLogger
7 from dct2_performance.utils.constants import ITERATIONS
8
9
10 class DCTRunner:
11     """
```

```

12     Class to easily run benchmarks on dct2 functions.
13     """
14
15     def __init__(self, functions: Collection[Callable],
16                  benchmark_sizes: Collection[int], logs_path: str = ".\\"):
17         """
18             DCTRunner constructor
19             :param functions: list of callables to test
20             :param benchmark_sizes: sizes to run the benchmark on
21             :param logs_path: path to save the logs to
22             """
23
24         self.functions = functions
25         self.benchmark_sizes = benchmark_sizes
26         self.logger = CSVLogger("benchmark", logs_path)
27
28     def run(self) -> str:
29         """
30             Method to start the benchmark procedure
31             :return: path to the logs file
32             """
33
34         print("Starting benchmark...")
35
36         results = {}
37         for N in self.benchmark_sizes:
38             results["size"] = N
39             for func in self.functions:
40                 results[func.__name__] = self._run_single(func, N)
41
42             self.logger.write_row(results)
43
44         print("Benchmark completed.")
45
46         return self.logger.log_file
47
48     def _run_single(self, func: Callable, N: int) -> float:
49         """
50             Method to test a single function with a specified size
51             :param func: callable to test
52             :param N: size to test
53             :return: total test time
54             """
55
56         print(f"Running function: {func.__name__} with size {N}")
57
58         matrix = np.random.rand(N, N) * 255
59         tot_time = 0
60
61         for _ in range(ITERATIONS):
62             start = time.perf_counter()
63             func(matrix)
64             end = time.perf_counter()
65
66             tot_time += end - start

```

```

62
63     return tot_time / ITERATIONS

```

A.3 plotter.py

```

1  from typing import Optional
2
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import os
6  import pandas as pd
7
8  from dct2_performance.utils.constants import PLOT_PATH
9
10
11 class PerformancePlotter:
12     """
13         Class to plot the performance of DCT2 functions based on logged
14         data.
15     """
16
17     def __init__(self, log_file: str, save_path: str = PLOT_PATH) ->
18         None:
19             """
20                 PerformancePlotter constructor.
21                 :param log_file: Path to the log file containing benchmark
22                     data
23                 :param save_path: Path to save the performance plot
24             """
25
26             self.log_file = log_file
27             self.save_path = save_path
28
29
30     def save_performance_plot(self, func_to_norm: Optional[str] =
31         None):
32             """
33                 Method to save the performance plot based on the logged data
34
35                 .
36
37                 :param func_to_norm: Function name to normalize the
38                     complexity curves against. If None, uses the first
39                     function.
40             """
41
42             os.makedirs(os.path.dirname(self.save_path), exist_ok=True)
43
44             df = pd.read_csv(self.log_file)
45             sizes = df['size'].values
46             df.drop(columns=['size'], inplace=True)
47
48             if not func_to_norm:
49                 func_to_norm = df.columns[0]
50
51             n = sizes.astype(np.float64)

```

```

39     n1 = n
40     nlogn = n * np.log2(n)
41     n2 = n ** 2
42     n2logn = n ** 2 * np.log2(n)
43     n3 = n ** 3
44
45     # Normalize the complexity curves to the first value to
46     # match the scale of the plot
47     n1_norm = n1 * (df[func_to_norm][0] / n1[0])
48     nlogn_norm = nlogn * (df[func_to_norm][0] / nlogn[0])
49     n2_norm = n2 * (df[func_to_norm][0] / n2[0])
50     n2logn_norm = n2logn * (df[func_to_norm][0] / n2logn[0])
51     n3_norm = n3 * (df[func_to_norm][0] / n3[0])
52
53     """
54     # Hardcoded section to normalize the first 5 functions based
55     # on scipy and the last one based on the custom function
56     n1_norm = n1 * (df["dct2_scipy"][0] / n1[0])
57     nlogn_norm = nlogn * (df["dct2_scipy"][0] / nlogn[0])
58     n2_norm = n2 * (df["dct2_scipy"][0] / n2[0])
59     n2logn_norm = n2logn * (df["dct2_scipy"][0] / n2logn[0])
60     n3_norm = n3 * (df["dct2_separable"][0] / n3[0])
61     """
62
63     plt.figure(figsize=(10, 6))
64
65     for func, time in df.items():
66         plt.semilogy(sizes, time, label=func, linewidth=2,
67                       markersize=6)
68
69         plt.semilogy(sizes, n1_norm, '--', color='gray', label=r'$\mathcal{O}(n)$')
70         plt.semilogy(sizes, nlogn_norm, '--c', label=r'$\mathcal{O}(n \log n)$')
71         plt.semilogy(sizes, n2_norm, '--', color='orange', label=r'$\mathcal{O}(n^2)$')
72         plt.semilogy(sizes, n2logn_norm, '--m', label=r'$\mathcal{O}(n^2 \log n)$')
73         plt.semilogy(sizes, n3_norm, '--r', label=r'$\mathcal{O}(n^3)$')
74
75         plt.xlabel('Matrix Size (N N)')
76         plt.ylabel('Execution Time [s]')
77         plt.title('DCT2 Performance Comparison')
78         plt.legend()
79         plt.grid(True, which='both', linestyle='--', alpha=0.5)
80         plt.tight_layout()

    file_name = "plot_" + os.path.basename(self.log_file)[:-4] +
               ".png"
    full_path = os.path.join(self.save_path, file_name)

```

```

81     plt.savefig(full_path, dpi=300)
82     print(f"Plot saved at: {os.path.abspath(full_path)}")
83
84     plt.close()

```

A.4 CSVLogger.py

```

1 import csv
2 import os
3 from datetime import datetime
4
5
6 class CSVLogger:
7     """
8         Class to simplify logging the runs data to .csv files.
9     """
10    def __init__(self, filename: str = None, path: str = "") -> None:
11        :
12        """
13            CSVLogger constructor.
14            :param filename: Log file name
15            :param path: Path to save the file to
16        """
17        self._log_file = self._create_log_file_path(filename, path)
18
19    @property
20    def log_file(self) -> str:
21        return self._log_file
22
23    def _create_log_file_path(self, filename: str, path: str) -> str:
24        :
25        """
26            Method to create the logs file path.
27            :param filename: Name of the log file
28            :param path: Path to save the file to
29            :return: Full path file
30        """
31
32        if filename is None:
33            filename = 'dct2_benchmark'
34
35        filename += " " + datetime.now().strftime("%Y_%m_%d_%H%M%S")
36        + ".csv"
37        log_file = os.path.join(path, filename)
38
39        os.makedirs(path, exist_ok=True)
40        return log_file
41
42    def write_row(self, data: dict) -> None:
43        """
44

```

```

40     Method to write a row to the logs file. If the file is empty
41         , it writes the header first
42     :param data: dict of data to write
43     """
44
45     with open(self._log_file, 'a', newline='') as f:
46         writer = csv.DictWriter(f, fieldnames=data.keys())
47         if os.stat(self._log_file).st_size == 0:
48             writer.writeheader()
49         writer.writerow(data)

```

A.5 functions.py

```

1 import numpy as np
2
3 from scipy.fft import dctn
4 from dct2_performance.utils import constants
5
6
7 def dct1d_custom(signal: np.ndarray) -> np.ndarray:
8     """
9         Custom implementation of the 1D Discrete Cosine Transform (DCT
10            Type II).
11        :param signal: Input signal (1D array)
12        :return: Transformed signal (1D array)
13        """
14
15    N = len(signal)
16    result = np.zeros(N)
17
18    def alpha(u): return 1 / np.sqrt(2) if u == 0 else 1
19
20    for u in range(N):
21        sum_val = sum(signal[x] * np.cos((2 * x + 1) * u * np.pi /
22            (2 * N)) for x in range(N))
23        result[u] = np.sqrt(2 / N) * alpha(u) * sum_val
24    return result
25
26
27 def dct2_separable(image: np.ndarray) -> np.ndarray:
28     """
29         Custom implementation of the 2D Discrete Cosine Transform (DCT
30            Type II) using separable DCT.
31        :param image: Input image (2D array)
32        :return: Transformed image (2D array)
33        """
34
35    rows_dct = np.apply_along_axis(dct1d_custom, axis=1, arr=image)
36    return np.apply_along_axis(dct1d_custom, axis=0, arr=rows_dct)

```

```

36     Wrapper for the scipy's dctn function to compute the 2D DCT Type
37         II.
38     :param image: Input image (2D array)
39     :return: Transformed image (2D array)
40     """
41
42
43 def test_correctness_scipy() -> None:
44     """
45     Test to verify the correctness of the DCT2 implementation
46         against scipy's dctn.
47     """
48
49     scipy_result = dct2_scipy(constants.TEST_BLOCK.astype(float))
50     print("Testing correctness of scipy's dctn for DCT Type II...\n")
51
52     print("Expected values:")
53     print(constants.EXPECTED_DCT2)
54     print("\nActual values:")
55     print(scipy_result)
56
57     max_err_dct2 = np.max(np.abs(scipy_result - constants.
58                           EXPECTED_DCT2))
59     assert max_err_dct2 < constants.TOLERANCE, f"DCT2 max error {
60         max_err_dct2:.2f} exceeds tolerance"
61
62     row_result = dctn(constants.TEST_BLOCK[0, :].astype(float), type
63         =2, norm='ortho')
64
65     print("\nExpected values:")
66     print(constants.EXPECTED_DCT1D)
67     print("\nActual values:")
68     print(row_result)
69
70     max_err_dct1 = np.max(np.abs(row_result - constants.
71                           EXPECTED_DCT1D))
72     assert max_err_dct1 < constants.TOLERANCE, f"DCT1D max error {
73         max_err_dct1:.2f} exceeds tolerance"
74     print("\nCorrectness test passed for scipy's dctn DCT Type II
75         implementation.\n")

```

A.6 constants.py

```

1 import numpy as np
2
3 # Tolerance for correctness test
4 TOLERANCE: float = 10.0
5
6 # Input test matrix

```

```

7 TEST_BLOCK = np.array([
8     [231, 32, 233, 161, 24, 71, 140, 245],
9     [247, 40, 248, 245, 124, 204, 36, 107],
10    [234, 202, 245, 167, 9, 217, 239, 173],
11    [193, 190, 100, 167, 43, 180, 8, 70],
12    [11, 24, 210, 177, 81, 243, 8, 112],
13    [97, 195, 203, 47, 125, 114, 165, 181],
14    [193, 70, 174, 167, 41, 30, 127, 245],
15    [87, 149, 57, 192, 65, 129, 178, 228]
16])
17
18 # Expected DCT2 result (rounded)
19 EXPECTED_DCT2 = np.array([
20     [1.11e+03, 4.40e+01, 7.59e+01, -1.38e+02, 3.50e+00, 1.22e+02,
21      1.95e+02, -1.01e+02],
22     [7.71e+01, 1.14e+02, -2.18e+01, 4.13e+01, 8.77e+00, 9.90e+01,
23      1.38e+02, 1.09e+01],
24     [4.48e+01, -6.27e+01, 1.11e+02, -7.63e+01, 1.24e+02, 9.55e+01,
25      -3.98e+01, 5.85e+01],
26     [-6.99e+01, -4.02e+01, -2.34e+01, -7.67e+01, 2.66e+01, -3.68e
27      +01, 6.61e+01, 1.25e+02],
28     [-1.09e+02, -4.33e+01, -5.55e+01, 8.17e+00, 3.02e+01, -2.86e+01,
29      2.44e+00, -9.41e+01],
30     [-5.38e+00, 5.66e+01, 1.73e+02, -3.54e+01, 3.23e+01, 3.34e+01,
31      -5.81e+01, 1.90e+01],
32     [7.88e+01, -6.45e+01, 1.18e+02, -1.50e+01, -1.37e+02, -3.06e+01,
33      -1.05e+02, 3.98e+01],
34     [1.97e+01, -7.81e+01, 9.72e-01, -7.23e+01, -2.15e+01, 8.13e+01,
35      6.37e+01, 5.90e+00]
36])
37
38 # Expected DCT1 result
39 EXPECTED_DCT1D = np.array([401.0, 6.6, 109.0, -112.0, 65.4, 121.0,
40     116.0, 28.8])
41
42 # Paths for saving results
43 PLOT_PATH: str = "plot_benchmark/"
44
45 # Number of iterations for benchmarks
46 ITERATIONS: int = 1

```

Appendice B

Codice Parte II

B.1 main.py

```
1 import os
2 import platform
3 import subprocess
4 import numpy as np
5
6 from scipy.fftpack import dctn, idctn
7 from tkinter import filedialog, Tk
8 from PIL import Image
9
10
11 def dct2(block):
12     """
13     Wrapper of scipy dctn function.
14     """
15     return dctn(block, type=2, norm='ortho')
16
17
18 def idct2(block):
19     """
20     Wrapper of scipy idctn function.
21     """
22     return idctn(block, type=2, norm='ortho')
23
24
25 def select_image():
26     """
27     Function to select an image.
28     :return: path to the selected image
29     """
30
31     root = Tk()
32     root.withdraw()
33     return filedialog.askopenfilename(filetypes=[("Bitmap files", ".*.bmp")])
34
35
36 def compress_image(img_array, F, d):
37     """
38     Function to compress an image using dct2.
39     :param img_array: image to be compressed, in array format
     :param F: block dimension
```

```

40 :param d: frequency cutoff threshold
41 :return: compressed image in array format
42 """
43 h, w = img_array.shape
44 h_blocks = h // F
45 w_blocks = w // F
46
47 compressed = np.zeros((h_blocks * F, w_blocks * F))
48
49 for i in range(h_blocks):
50     for j in range(w_blocks):
51         block = img_array[i * F:(i + 1) * F, j * F:(j + 1) * F]
52         c = dct2(block)
53
54         for k in range(F):
55             for l in range(F):
56                 if k + l >= d:
57                     c[k, l] = 0
58
59         ff = idct2(c)
60         ff = np.round(ff)
61         ff = np.clip(ff, 0, 255)
62         compressed[i * F:(i + 1) * F, j * F:(j + 1) * F] = ff
63
64 return compressed.astype(np.uint8)
65
66
67 def save_compressed_image(compressed_img, out_path):
68 """
69 Function to save compressed image.
70 :param compressed_img: compressed image in array format
71 :param out_path: output path
72 :return: abs path to the saved image
73 """
74 output_abs_path = os.path.abspath(out_path)
75
76 Image.fromarray(compressed_img).save(output_abs_path)
77 print(f"Image saved at: {output_abs_path}")
78
79 return output_abs_path
80
81
82 def open_image(path):
83 """
84 Function to open an image.
85 :param path: path to the image to open
86 """
87 try:
88     # macOS
89     if platform.system() == "Darwin":
90         subprocess.run(["open", path])

```

```

91     # Windows
92     elif platform.system() == "Windows":
93         os.startfile(path)
94     # Linux
95     else:
96         subprocess.run(["xdg-open", path])
97 except Exception as e:
98     print(f"Error opening the image: {e}")
99
100
101 def dct2_compress(input_file, F, d, output_dir):
102     """
103     Function to start the compression process
104     :param input_file: path to the input image
105     :param F: block dimension
106     :param d: frequency cutoff threshold
107     :param output_dir: folder to save the plot_benchmark to
108     """
109     img = Image.open(input_file).convert('L')
110     img_array = np.array(img)
111
112     compressed_img = compress_image(img_array, F, d)
113
114     img_name = os.path.splitext(os.path.basename(input_file))[0]
115     compressed_img_name = f"{img_name}_compressed_F{F}_d{d}.bmp"
116
117     os.makedirs(output_dir, exist_ok=True)
118     output_path = os.path.join(output_dir, compressed_img_name)
119     output_file = save_compressed_image(compressed_img, output_path)
120
121     open_image(input_file)
122     open_image(output_file)
123
124
125 if __name__ == "__main__":
126     print("Select a .BMP image")
127     input_file = select_image()
128
129     if not input_file:
130         print("No image selected.")
131         exit()
132
133     F = int(input("Insert block dimension F: "))
134     d = int(input(f"Insert frequency cutoff threshold d (0 <= d < {2
135             * F - 1}): "))
136
137     dct2_compress(input_file, F, d, "output/")

```

B.2 gui.py

```

1 import os
2 import tkinter as tk
3
4 from tkinter import filedialog, messagebox
5 from tkinter import ttk
6 from main import dct2_compress
7
8
9 class DCT2App:
10     def __init__(self, root):
11         self.root = root
12         self.root.title("DCT2 Image Compression Tool")
13         self.root.geometry("500x500")
14         self.root.configure(bg="#2e2e2e")
15
16         self.file_path = tk.StringVar()
17         self.output_folder = tk.StringVar(value=os.getcwd() + "\\"
18                                         "output\\")
19         self.F = tk.IntVar(value=8)
20         self.d = tk.IntVar(value=10)
21
22         self.create_widgets()
23
24     def create_widgets(self):
25         style = ttk.Style()
26         style.theme_use('clam')
27         style.configure(
28             "TLabel",
29             background="#2e2e2e",
30             foreground="white",
31             font=('Helvetica', 10))
32
33         style.configure(
34             "TEntry",
35             fieldbackground="#424242",
36             foreground="white",
37             bordercolor="#4caf50",
38             lightcolor="#4caf50",
39             darkcolor="#2e2e2e",
40             padding=5)
41
42         style.configure(
43             "TButton",
44             padding=6,
45             relief="flat",
46             background="#4caf50",
47             foreground="white")
48
49         style.map(
50             "TButton",

```

```

50         background=[("active", "#45a049")]
51     )
52
53     ttk.Label(self.root, text="Select a BMP image using the
54         Browse button:").pack(pady=(10, 0))
55
56     self.file_label = tk.Label(
57         self.root,
58         text="No file selected",
59         bg="#424242",
60         fg="white",
61         relief="groove",
62         width=50,
63         height=2
64     )
65     self.file_label.pack(pady=10)
66
67     ttk.Button(self.root, text="Browse", command=self.
68         browse_file).pack(pady=5)
69
70     ttk.Label(self.root, text="Output folder:").pack(pady=(15,
71         0))
72     self.output_entry = ttk.Entry(self.root, textvariable=self.
73         output_folder, width=50, style="TEntry")
74     self.output_entry.pack(pady=10)
75     ttk.Button(self.root, text="Select Folder", command=self.
76         browse_output_folder).pack(pady=5)
77
78     ttk.Label(self.root, text="Block size (F):").pack(pady=(15,
79         0))
80     self.f_slider = tk.Scale(self.root, from_=1, to=64, orient=
81         tk.HORIZONTAL, variable=self.F,
82             command=self.update_d_slider, bg="
83                 #2e2e2e", fg="white",
84                 troughcolor="#4caf50",
85                 highlightthickness=0)
86     self.f_slider.pack(padx=20, fill="x")
87
88     ttk.Label(self.root, text="Frequency cut-off (d):").pack(
89         pady=(15, 0))
90     self.d_slider = tk.Scale(self.root, from_=0, to=14, orient=
91         tk.HORIZONTAL, variable=self.d, bg="
92                 #2e2e2e",
93                 fg="white", troughcolor="#4caf50",
94                 highlightthickness=0)
95     self.d_slider.pack(padx=20, fill="x")
96
97     ttk.Button(self.root, text="Confirm", command=self.submit).
98         pack(pady=20)
99
100    def update_d_slider(self, val):
101        f_val = int(val)

```

```

88     max_d = 2 * f_val - 2
89     self.d_slider.config(to=max_d)
90     if self.d.get() > max_d:
91         self.d.set(max_d)
92
93     def browse_file(self):
94         filename = filedialog.askopenfilename(filetypes=[("BMP files",
95             ".*.bmp")])
96         if filename:
97             self.file_path.set(filename)
98             self.file_label.config(text=os.path.basename(filename))
99
100    def browse_output_folder(self):
101        foldername = filedialog.askdirectory()
102        if foldername:
103            self.output_folder.set(foldername)
104
105    def submit(self):
106        try:
107            f_value = self.F.get()
108            d_value = self.d.get()
109            file_path = self.file_path.get()
110            output_folder = self.output_folder.get()
111
112            if not os.path.isfile(file_path):
113                raise ValueError("Invalid or no file selected.")
114            if not os.path.isdir(output_folder):
115                raise ValueError("Invalid or no output folder
116                                selected.")
117            if f_value <= 0:
118                raise ValueError("F must be a positive integer.")
119            if not (0 <= d_value <= 2 * f_value - 2):
120                raise ValueError(f"d must be between 0 and {2 *
121                                f_value - 2}.")
122
123            dct2_compress(file_path, f_value, d_value, output_folder
124                          )
125
126        except ValueError as e:
127            messagebox.showerror("Error", str(e))
128
129
130    if __name__ == "__main__":
131        root = tk.Tk()
132        app = DCT2App(root)
133        root.mainloop()

```