Università degli Studi di Milano Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

Metodi del Calcolo Scientifico – Progetto 2

Compressione JPEG

Compressione di Immagini tramite DCT

Autori:

Giacomo Savazzi – 845372

Raffaele Cerizza – 845512

Andrea Assirelli - 820149

Anno Accademico 2021 – 2022

Sommario

Indice delle Figure

Introduzione

Questo progetto prevede due scopi principali. Il primo consiste nel verificare l'efficienza di due implementazioni della trasformata discreta del coseno in due dimensioni (in seguito: ***DCT2***) in ambiente open-source: una che implementi un generico algoritmo per la ***DCT2*** senza utilizzare specifiche librerie esterne; e un'altra che sfrutti un'apposita libreria. Il secondo consiste nello studio degli effetti di un algoritmo di compressione come ***JPEG*** su immagini in toni di grigio.

Si procederà con il seguente ordine. In primo luogo verrà fornito un inquadramento generale sulla trasformata discreta del coseno. In secondo luogo verranno forniti dettagli sul linguaggio di programmazione e sulle librerie utilizzate per il progetto. Verrà poi illustrato il confronto delle predette due implementazioni della ***DCT2***. Infine verrà presentato l'algoritmo di compressione delle immagini e verranno commentati alcuni esempi applicativi.

Discrete Cosine Transform (DCT)

La ***DCT*** è una funzione che si presta a essere utilizzata all'interno di algoritmi per la compressione di immagini. In particolare questi algoritmi utilizzano la ***DCT*** per trasformare i valori dell'immagine in frequenze, cercando poi di conservare poche frequenze. L'operazione inversa alla ***DCT*** è la ***DCT*** inversa (in seguito: ***IDCT***). Questa funzione viene utilizzata per ricostruire l'immagine sulla base delle frequenze. In questo caso si parla di decompressione.

Più formalmente possiamo definire la funzione ***DCT*** come:

$$\boldsymbol{DCT}: \{\vec{f}\} \rightarrow \{\vec{c}\},$$

dove $\vec{f}$ è un vettore di numeri reali che negli algoritmi di compressione delle immagini rappresentano i valori puntuali dell'immagine, mentre $\vec{c}$ è il vettore delle frequenze.

Analogamente possiamo definire la funzione ***IDCT*** come:

$$\boldsymbol{IDCT}: \{\vec{c}\} \rightarrow \{\vec{f}\}.$$

Le operazioni di ***DCT*** e ***IDCT*** si possono scrivere in forma matriciale:

- Consideriamo i vettori $\vec{f} = (f_0, \dots, f_{N-1})$ e $\vec{c} = (c_0, \dots, c_{N-1})$;
- Consideriamo la matrice di trasformazione $\boldsymbol{D} \in \mathbb{R}^{N \times N}$ che sia tale che:

  $$D_{l,j} = \alpha_l^N \cos\left(l\pi \frac{2j+1}{2N}\right),$$

  dove:

  $$\alpha_l^N = \begin{cases} \frac{1}{\sqrt{N}} \; se \; l = 0 \\ \sqrt{\frac{2}{N}} \; altrimenti \end{cases}$$

- In questo quadro possiamo eseguire la ***DCT*** e calcolare il vettore $\vec{c}$ come:

  $$\vec{c} = D\vec{f}$$

- Inoltre possiamo eseguire la ***IDCT*** e calcolare il vettore $\vec{f}$ come:

  $$\vec{f} = D^T\vec{c}$$

È possibile estendere la ***DCT*** e la ***IDCT*** al caso bidimensionale. In questo caso si parla di ***DCT2*** e ***IDCT2***. Un generico algoritmo per l'esecuzione della ***DCT2*** prevede di calcolare (*i*) prima la ***DCT*** sulle colonne e (*ii*) poi la ***DCT*** sulle righe del risultato dell'operazione precedente. Lo stesso generico algoritmo può essere utilizzato per calcolare la ***IDCT2*** *mutatis mutandis*.

Il costo computazionale della ***DCT2*** e della ***IDCT2*** implementate come appena descritto è $\boldsymbol{O(N^3)}$. Esistono però algoritmi più efficienti per la ***DCT2*** e la ***IDCT2***. In particolare vi sono

algoritmi che sfruttano la trasformata di Fourier veloce (in seguito: ***FFT***). In questo caso il costo computazionale è $\boldsymbol{O}(\boldsymbol{N^2}\,\mathbf{log}(\boldsymbol{N}))$.

Linguaggio di programmazione e librerie

Per il presente progetto è stato utilizzato un linguaggio di programmazione open source. In particolare si è scelto di utilizzato Python. Si tratta di un linguaggio di programmazione ad alto livello molto diffuso e particolarmente adatto alla computazione numerica. Le sue caratteristiche principali sono le seguenti:

- è un linguaggio multi-paradigma. In particolare supporta: la programmazione orientata agli oggetti; la programmazione procedurale; e la programmazione funzionale;
- è un linguaggio interpretato. Quindi le istruzioni vengono eseguite senza necessità di una previa compilazione in linguaggio macchina;
- è un linguaggio debolmente tipizzato. Questo significa che non è necessario descrivere il tipo di una variabile in fase di dichiarazione;
- Python è mantenuto frequentemente.

In particolare per il presente progetto è stato necessario installare e utilizzare diverse librerie. Le librerie principali sono le seguenti:

- ***SciPy***: è una libreria open source per Python [1]. In particolare, ***SciPy*** fornisce una collezione di algoritmi e strumenti matematici e scientifici in senso lato. Più precisamente della libreria ***SciPy*** è stato utilizzato il *package* ***fft*** [2]. Questo *package* fornisce diverse funzioni per il calcolo della ***FFT***. Inoltre fornisce apposite funzioni per il calcolo della ***DCT***, della ***DCT2***, della ***IDCT*** e della ***IDCT2*** che sfruttano la ***FFT***. La libreria ***SciPy*** è frequentemente mantenuta e ben documentata.
- ***NumPy***: è una libreria open source per Python [3]. In particolare, questa libreria fornisce un'ampia collezione di funzioni matematiche e facilita la gestione di matrici e array di qualsiasi dimensione.
- ***Matplotlib***: è una libreria open source di supporto al linguaggio Python che permette di creare grafici statici, animati e interattivi [4]. In particolare, questa libreria è stata utilizzata per mostrare graficamente i risultati delle diverse implementazioni della ***DCT2***.
- ***TkInter***: è una libreria open source per Python che consente di realizzare interfacce grafiche [3]. Ai fini del presente progetto è stata utilizzata per realizzare una semplice applicazione in grado di comprimere immagini in toni di grigio tramite ***DCT2*** e ***IDCT2***.
- ***Pillow (PIL)***: è una libreria open source per Python deputata alla gestione di immagini [4]. Questa libreria è stata utilizzata per convertire immagini in formati compatibili con ***TkInter***.

- ***OpenCV-Python (CV2)***: è una libreria open source per Python che permette il caricamento di una gamma di immagini con estensione .bmp maggiore rispetto a ***PIL*** [7].
- ***Imageio***: è un'altra libreria open source per Python di supporto alla gestione delle immagini [8]. Questa libreria è stata utilizzata per salvare in memoria le immagini compresse tramite ***DCT2*** e ***IDCT2***.

Parte 1: confronto dei tempi di esecuzione

Nel paragrafo introduttivo si è anticipato che il progetto esposto nella presente relazione si compone di due parti. La prima parte ha come obiettivo il confronto dei tempi di esecuzione di due implementazioni della ***DCT2***. In particolare, per questa prima parte si è proceduto a:

- implementare una versione semplice della ***DCT2*** in Python;
- implementare una versione della ***DCT2*** che sfrutti la ***FFT*** fornita dal *package* ***fft*** di ***SciPy***;
- confrontare i tempi di esecuzione di queste due implementazioni della ***DCT2*** su matrici quadrate $N \times N$ e verificare che i tempi siano proporzionali rispettivamente a $\boldsymbol{N^3}$ e a $\boldsymbol{N^2 \log(N)}$.

Per l'esposizione del lavoro svolto si seguirà questo ordine: anzitutto verrà descritta la semplice implementazione della ***DCT2*** in Python; dopodiché verranno forniti alcuni dettagli sull'implementazione della ***DCT2*** tramite il *package* ***fft*** di ***SciPy***; infine verranno presentati e commentati i confronti fra le implementazioni della ***DCT2***.

Per la raccolta dei dati oggetto di confronto è stata utilizzata una macchina con processore Intel(R) Core(TM) i7-6700K a 4.00GHz con 8 *core*, con 16 GB di RAM e con una scheda video NVIDIA GeForce GTX 1070. Il sistema operativo utilizzato è stato Microsoft Windows.

Implementazione semplice della DCT2

Cominciamo con la presentazione dell'implementazione della ***DCT2*** che non sfrutta il *package* ***fft*** di ***SciPy***. In particolare, per l'implementazione della ***DCT2*** si è proceduto a definire tre distinte funzioni: una prima funzione calcola la matrice di trasformazione ***D***; una seconda funzione calcola la ***DCT***; e una terza funzione calcola la ***DCT2***. Vengono ora mostrate le implementazioni di ciascuna di queste funzioni.

1. Implementazione della matrice di trasformazione ***D***.

   Il codice per questa implementazione è riportato nella Figura 1.

```
def compute_D(n):
    '''
    Funzione per calcolare la matrice di trasformazione D.

    :param n: numero di elementi del vettore f per il calcolo della DCT1
    :return D: matrice di trasformazione
    '''

    # Calcola i valori alfa utilizzati per la matrice di trasformazione
    alpha_vect = np.zeros(n)
    alpha_vect[0] = 1.0 / sqrt(n)
    alpha_vect[1:n] = sqrt(2.0 / n)

    # Calcola la matrice di trasformazione
    D = np.zeros((n, n))
```

```
    for i in range(n):
        for j in range(n):
            D[i,j] = alpha_vect[i] * cos(pi * (2 * j + 1) * i / (2 * n))
    return D
```

Figura 1 - Implementazione semplice matrice D

In particolare questa funzione: (*i*) prima calcola i valori α che verranno utilizzati come fattore moltiplicativo nella computazione dei valori della matrice ***D***; e (*ii*) poi calcola ogni valore della matrice di trasformazione ***D***. È agevole notare come questa implementazione costituisca una mera traduzione in Python della definizione della matrice ***D*** riportata in precedenza. Il tempo di esecuzione di questa funzione è $\boldsymbol{O(N)^2}$ dato che viene calcolato un valore per ciascun elemento della matrice $\boldsymbol{D} \in \mathbb{R}^{N \times N}$.

2. Implementazione della ***DCT***.

Il codice per questa implementazione è riportato nella Figura 2.

```
def my_dct1(f):
    '''
    Implementazione della DCT1.
    :param f: vettore dei valori puntuali
    :return c: vettore delle frequenze
    '''
    # Imposta i valori di f in formato float in 64 bit
    f = f.astype('float64')

    # Calcola il numero di elementi di f
    n = len(f)

    # Calcola la matrice di trasformazione
    D = compute_D(n)

    # Moltiplica la matrice di trasformazione e f
    c = np.dot(D, f)

    return c
```

Figura 2 – Implementazione semplice della DCT

Questa funzione si limita a moltiplicare la matrice di trasformazione ***D*** per il vettore $\vec{\boldsymbol{f}}$. In questo caso è stato necessario convertire i valori puntuali del vettore $\vec{\boldsymbol{f}}$ in valori in virgola mobile con doppia precisione. In mancanza di questa conversione la ***DCT*** implementata prendeva in considerazione (non valori in virgola mobile, ma) solo valori interi; questo portava a una perdita di accuratezza durante i calcoli; e il risultato finale non soddisfaceva il livello di qualità richiesto. In ogni caso anche questa implementazione è coerente con la definizione della ***DCT*** riportata in precedenza.

3. Implementazione della ***DCT2***.

   Il codice per questa implementazione è riportato nella Figura 3.

```
def my_dct2(f):
    '''
    Implementazione della DCT2.

    :param f: matrice dei valori puntuali
    :return c: matrice delle frequenze
    '''

    # Calcola il numero di righe (n) e colonne (m)
    n, m = np.shape(f)

    # Copia f in c utilizzando valori float in 64 bit
    c = np.copy(f.astype('float64'))

    # Esegue la DCT1 sulle colonne
    for j in range(m):
        c[:,j] = my_dct1(c[:,j])

    # Esegue la DCT1 sulle righe
    for i in range(n):
        c[i,:] = my_dct1(c[i,:])

    return c
```

Figura 3 – Implementazione semplice della DCT2

La funzione che implementa la ***DCT2*** si limita ad eseguire la ***DCT*** prima sulle colonne e poi sulle righe della matrice ***f*** ricevuta in input. Anche in questo caso è stato necessario convertire i valori di ***f*** in valori in virgola mobile a doppia precisione. E in ogni caso anche questa implementazione è coerente con la definizione della ***DCT2*** riportata in precedenza.

Implementazione della DCT2 tramite SciPy.fft

Passiamo ora alla presentazione dell'implementazione della ***DCT2*** che sfrutta il *package* ***fft*** di ***SciPy***. In questo caso è stato sufficiente richiamare la funzione ***dct*** del predetto *package* come segue:

```
def scipy_dct2(f):
    '''
    Implementazione della DCT2 tramite scipy.fft.
    '''
    return dct(dct(f.T, norm='ortho').T, norm='ortho')
```

Figura 4 - Implementazione della DCT2 con SciPy

La sintassi per il calcolo della ***DCT2*** è riportata nella documentazione della libreria. Si noti che in questo caso si è scelto di impostare il parametro *norm='ortho'*. Con questo parametro la funzione ***dct*** considera anche i valori α (come definiti in precedenza) nel calcolo della matrice di trasformazione ***D***. In mancanza di questo parametro il calcolo non utilizza questi valori α.

Inoltre si noti l'utilizzo dell'attributo ***T*** nella chiamata della funzione. Questo attributo contiene semplicemente la trasposizione del relativo array. Attraverso l'utilizzo di questo attributo la ***DCT*** viene eseguita prima sulle colonne e poi sulle righe del relativo risultato. In questo modo risulta replicato esattamente il procedimento illustrato in precedenza e implementato anche nella versione semplice che non sfrutta il *package* ***fft*** di ***SciPy***.

Al fine di verificare il corretto funzionamento della funzione ***dct*** si è proceduto a implementare anche la ***DCT*** come segue:

```
def scipy_dct1(f):
    '''
    Implementazione della DCT1 tramite scipy.fft.
    '''
    return dct(f.T, norm='ortho')
```

Figura 5 - Implementazione della DCT tramite SciPy

Verifica della correttezza delle implementazioni della DCT2

Lo scaling effettuato dalle funzioni che implementano la ***DCT*** (e la ***DCT2***) può essere diverso. A questo proposito è richiesto dalla consegna del progetto di verificare che le implementazioni della ***DCT*** e della ***DCT2*** rispettino un determinato scaling. In questo quadro si è proceduto a implementare un'apposita funzione per questa verifica.

In particolare questa funzione permette di verificare il risultato della trasformazione di un blocco di 8×8 valori sia tramite la ***DCT*** che tramite la ***DCT2***. Il controllo della correttezza dei risultati viene svolto manualmente tramite una semplice comparazione dei risultati ottenuti.

Si è proceduto a eseguire questa verifica sia per le implementazioni della ***DCT*** e della ***DCT2*** offerte dal *package* ***fft*** di ***SciPy*** sia per le versioni più semplici che non sfruttano questo *package*.

Il codice della funzione di test è riportato nella Figura 6. I risultati ottenuti sono riportati invece nella Figura 7.

```
def test():
    '''
    Funzione per verificare se le funzioni implementate per
    calcolare DCT1 e DCT2 rispettano i valori specificati
    nella consegna.
    '''

    # Array con i valori utilizzati per il test
    test_dct1 = np.array([231,  32, 233, 161,  24,  71, 140, 245])
    test_dct2 = np.array([[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]])

    # Stampa dei risultati. Il controllo della correttezza va svolto
    # manualmente
    print("Verifica correttezza di scipy_dct1: ")
    result_scipy_test_dct1 = scipy_dct1(test_dct1)
    print(result_scipy_test_dct1)
    print()
    print("Verifica correttezza di my_dct1: ")
    result_my_test_dct1 = my_dct1(test_dct1)
    print(result_my_test_dct1)
    print()
    print("Verifica correttezza di scipy_dct2: ")
    result_scipy_test_dct2 = scipy_dct2(test_dct2)
    print(result_scipy_test_dct2)
    print()
    print("Verifica correttezza di my_dct2: ")
    result_my_test_dct2 = my_dct2(test_dct2)
    print(result_my_test_dct2)
    print()
```

Figura 6 - Funzione di verifica della correttezza delle implementazioni di DCT e DCT2

```
Verifica correttezza di scipy_dct1:
[ 401.9902051     6.60001991  109.16736544 -112.78557857   65.40737726
  121.83139804  116.65648855   28.80040722]

Verifica correttezza di my_dct1:
[ 401.9902051     6.60001991  109.16736544 -112.78557857   65.40737726
  121.83139804  116.65648855   28.80040722]

Verifica correttezza di scipy_dct2:
[[ 1.11875000e+03  4.40221926e+01  7.59190503e+01 -1.38572411e+02
   3.50000000e+00  1.22078055e+02  1.95043868e+02 -1.01604906e+02]
 [ 7.71900790e+01  1.14868206e+02 -2.18014421e+01  4.13641351e+01
   8.77720598e+00  9.90829620e+01  1.38171516e+02  1.09092795e+01]
 [ 4.48351537e+01 -6.27524464e+01  1.11614114e+02 -7.63789658e+01
   1.24422160e+02  9.55984194e+01 -3.98287969e+01  5.85237670e+01]
 [-6.99836647e+01 -4.02408945e+01 -2.34970508e+01 -7.67320594e+01
   2.66457750e+01 -3.68328290e+01  6.61891485e+01  1.25429731e+02]
 [-1.09000000e+02 -4.33430857e+01 -5.55436908e+01  8.17347083e+00
   3.02500000e+01 -2.86602437e+01  2.44149822e+00 -9.41437025e+01]
 [-5.38783591e+00  5.66345009e+01  1.73021519e+02 -3.54234494e+01
   3.23878249e+01  3.34576728e+01 -5.81167864e+01  1.90225615e+01]
 [ 7.88439693e+01 -6.45924096e+01  1.18671203e+02 -1.50904840e+01
  -1.37316928e+02 -3.06196663e+01 -1.05114114e+02  3.98130497e+01]
 [ 1.97882438e+01 -7.81813409e+01  9.72311860e-01 -7.23464180e+01
  -2.15781633e+01  8.12999035e+01  6.37103782e+01  5.90618071e+00]]

Verifica correttezza di my_dct2:
[[ 1.11875000e+03  4.40221926e+01  7.59190503e+01 -1.38572411e+02
   3.50000000e+00  1.22078055e+02  1.95043868e+02 -1.01604906e+02]
 [ 7.71900790e+01  1.14868206e+02 -2.18014421e+01  4.13641351e+01
   8.77720598e+00  9.90829620e+01  1.38171516e+02  1.09092795e+01]
 [ 4.48351537e+01 -6.27524464e+01  1.11614114e+02 -7.63789658e+01
   1.24422160e+02  9.55984194e+01 -3.98287969e+01  5.85237670e+01]
 [-6.99836647e+01 -4.02408945e+01 -2.34970508e+01 -7.67320594e+01
   2.66457750e+01 -3.68328290e+01  6.61891485e+01  1.25429731e+02]
 [-1.09000000e+02 -4.33430857e+01 -5.55436908e+01  8.17347083e+00
   3.02500000e+01 -2.86602437e+01  2.44149822e+00 -9.41437025e+01]
 [-5.38783591e+00  5.66345009e+01  1.73021519e+02 -3.54234494e+01
   3.23878249e+01  3.34576728e+01 -5.81167864e+01  1.90225615e+01]
 [ 7.88439693e+01 -6.45924096e+01  1.18671203e+02 -1.50904840e+01
  -1.37316928e+02 -3.06196663e+01 -1.05114114e+02  3.98130497e+01]
 [ 1.97882438e+01 -7.81813409e+01  9.72311860e-01 -7.23464180e+01
  -2.15781633e+01  8.12999035e+01  6.37103782e+01  5.90618071e+00]]
```

Figura 7 - Risultati della verifica della correttezza delle implementazioni di DCT e DCT2

È agevole riscontrare che i risultati ottenuti sono quelli desiderati.

Confronto

Sin qui sono state presentate le implementazioni della ***DCT2*** da confrontare. Ora è possibile illustrarne il confronto. Procediamo con ordine.

- **Scopi**. Il confronto ha due scopi. Il primo è misurare i tempi di esecuzione delle diverse implementazioni della ***DCT2***: quella semplice e quella che sfrutta il *package* ***SciPy.fft***. Il secondo è verificare che questi tempi siano proporzionali rispettivamente a N^3 per l'implementazione base e a $N^2 \log(N)$ per l'implementazione che sfrutta la libreria ***SciPy***. In questo caso N è il numero di righe (o colonne) di matrici quadrate $N \times N$.
- **Oggetto**. I tempi di esecuzione raccolti per il confronto riguardano il tempo impiegato dalle due versioni della ***DCT2*** per eseguire la trasformazione su una serie di matrici quadrate $N \times N$. In particolare si è scelto di utilizzare 18 matrici con una dimensione di N crescente da 50 a 900. Il valore di N viene incrementato di 50 dopo ogni esecuzione.
  Le matrici utilizzate contengono valori random compresi tra 0 e 255. Si è evitato di utilizzare matrici con pattern particolari in modo da ottenere risultati più generici e imparziali rispetto alle implementazioni degli algoritmi.
- **Codice**. Il codice per la raccolta dei dati è riportato nella Figura 8.

```
def get_data():
    '''
    Funzione per calcolare i tempi di esecuzione delle implementazioni
    della DCT2.

    :return times_scipy_dct: tempi di esecuzione della DCT2 di scipy
    :return times_my_dct: tempi di esecuzione della DCT2 implementata
     in questo file
    :return matrix_dimensions: dimensioni utilizzate per creare
     matrici NxN
    :return n3: tempi di esecuzione di n^3
    :return n2_logn: tempi di esecuzione di n^2 * log(n)
    '''

    # Istanziazione degli array per contenere i valori dei tempi
    times_scipy_dct = []
    times_my_dct = []
    matrix_dimensions = []
    n3 = []
    n2_logn = []

    # Ripetizione dei calcoli dei tempi di esecuzione per più matrici
    # range(start, stop, step)
    for n in range(50, 901, 50):
        print("Dimension: ", n)
        matrix_dimensions.append(n)
```

```
        # Creazione di una matrice random
        np.random.seed(10)
        matrix = np.random.uniform(low=0.0, high=255.0, size=(n, n))

        # Calcolo del tempo di esecuzione con scipy.fft
        time_start = time.perf_counter()
        result = scipy_dct2(matrix)
        time_end = time.perf_counter()
        times_scipy_dct.append(time_end - time_start)

        # Calcolo del tempo di esecuzione con la DCT2 implementata in
        # questo file
        time_start = time.perf_counter()
        result = my_dct2(matrix)
        time_end = time.perf_counter()
        times_my_dct.append(time_end - time_start)

        # Calcolo del tempo di esecuzione con n^3
        n3.append(np.power(n, 3))

        # Calcolo del tempo di esecuzione con n^2 * log(n)
        n2_logn.append(np.power(n, 2) * np.log(n))

    return times_scipy_dct, times_my_dct, matrix_dimensions, n3,
           n2_logn
```

Figura 8 - Codice per la raccolta dei tempi di esecuzione

- **Risultati**. Si mostrano ora i risultati ottenuti. Nella Figura 9 viene riportato un prospetto dei tempi di esecuzione registrati dalle due implementazioni della ***DCT2***. Nella Figura 10 invece vengono rappresentati graficamente i risultati per una loro migliore comprensione.

N	Tempi DCT2 base (s)	Tempi DCT2 SciPy (s)	N^3	(N^2) * log(N)
50	0,1830626	0,0001495	125000	9780,057514
100	1,3453343	0,0003711	1000000	46051,70186
150	4,0999953	0,0004617	3375000	112739,2941
200	9,9072294	0,0007813	8000000	211932,6947
250	19,0533304	0,001107	15625000	345091,3074
300	33,0368316	0,0016036	27000000	513340,4227
350	52,4890566	0,0030937	42875000	717596,8114
400	77,2011424	0,0032163	64000000	958634,3275
450	111,4987498	0,0034749	91125000	1237122,636
500	151,8757074	0,004412	125000000	1553652,025
550	206,3708672	0,0060974	166375000	1908750,279
600	262,9908453	0,0065759	216000000	2302894,676
650	336,1633171	0,0113224	274625000	2736520,823
700	420,2869141	0,008981	343000000	3210029,364
750	540,2124185	0,0115353	421875000	3723791,179
800	624,8959697	0,0109796	512000000	4278151,506
850	742,8144144	0,0169016	614125000	4873433,263
900	884,4590864	0,0134796	729000000	5509939,758

Figura 9 - Prospetto tempi di esecuzione

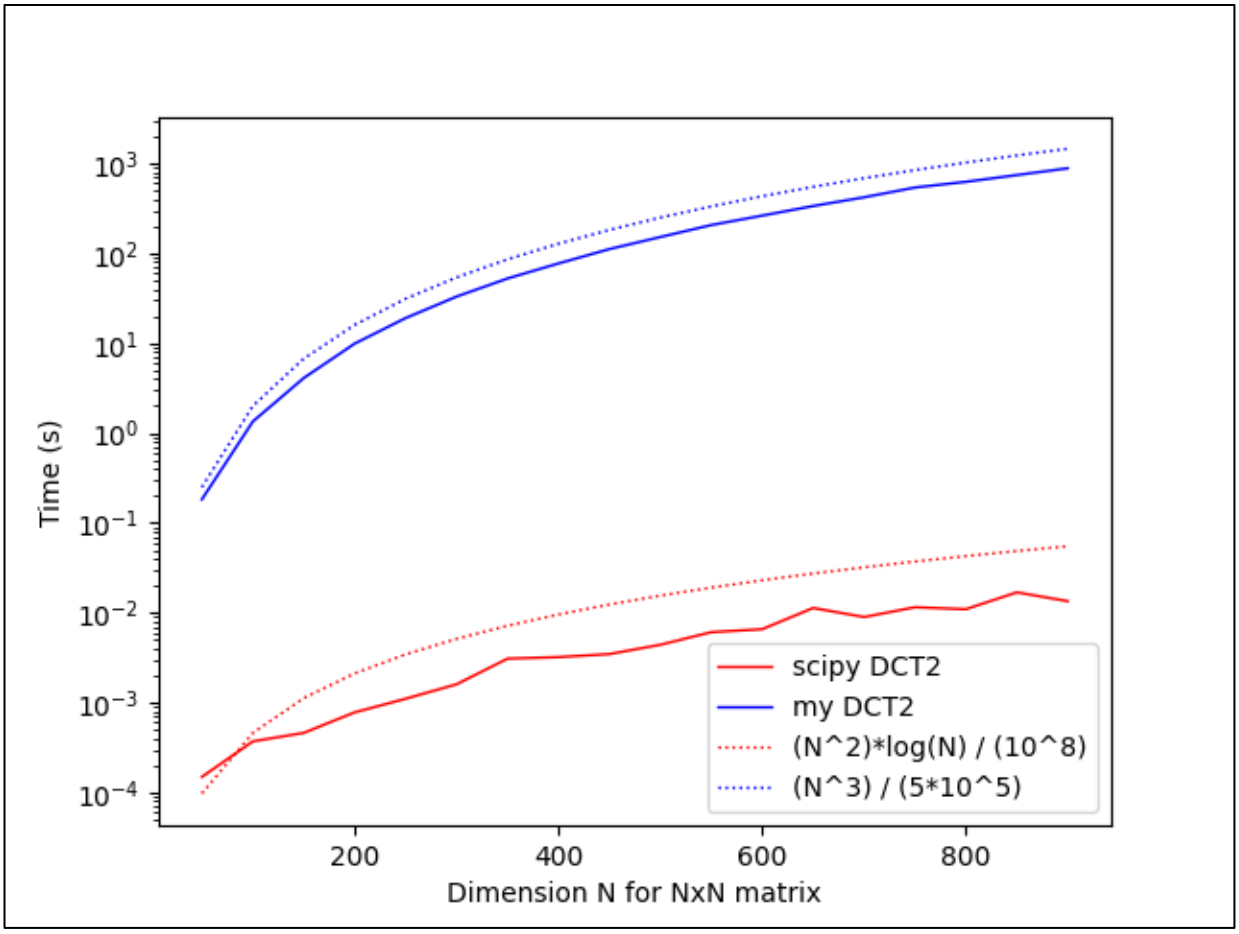

Figura 10 - Grafico dei tempi di esecuzione

Nella Figura 10 l'asse delle ascisse mostra i valori crescenti di ***N*** utilizzati come dimensione per le varie matrici. Sull'asse delle ordinate vengono invece riportati in scala logaritmica i tempi di esecuzione misurati in secondi. Ora, in questa Figura vengono mostrati sia i tempi di esecuzione delle due implementazioni della ***DCT2*** sia i tempi di esecuzione corrispondenti a $\boldsymbol{N^3}$ e a $\boldsymbol{N^2 \log(N)}$.

Queste due ultime serie di tempi di esecuzione sono state divise per un opportuno numero in modo da rendere più agevole la verifica della loro proporzionalità ai tempi di esecuzione delle due implementazioni della ***DCT2***. In particolare, risulta evidente come la serie dei tempi di esecuzione della ***DCT2*** base sia proporzionale a $\boldsymbol{N^3}$, mentre la serie dei tempi di esecuzione della ***DCT2*** di ***SciPy.ftt*** risulta proporzionale a $\boldsymbol{N^2 \log(N)}$. In questo quadro i tempi di esecuzione delle implementazioni confermano le proporzionalità attese e confermano come l'utilizzo della ***FFT*** velocizzi sensibilmente l'esecuzione della ***DCT2***.

Parte 2: compressione JPEG

La seconda parte del progetto ha avuto come obiettivo lo studio degli effetti di un algoritmo di compressione come ***JPEG*** su immagini in toni di grigio. In particolare, per questa seconda parte si è proceduto a realizzare un'applicazione che:

- presenti un'interfaccia per selezionare immagini dal *file system*;
- permetta all'utente di scegliere l'ampiezza dei macro-blocchi della ***DCT2*** e la soglia di taglio delle frequenze;
- suddivida le immagini in macro-blocchi partendo in alto a sinistra e scartando gli avanzi;
- applichi la ***DCT2*** a ogni macro-blocco eliminando le frequenze in base alla soglia predetta;
- applichi successivamente la ***IDCT2*** arrotondando i valori agli interi più vicini;
- ricomponga l'immagine utilizzando i risultati della ***IDCT2***;
- mostri a schermo affiancate l'immagine originale e quella ottenuta a seguito dell'applicazione di ***DCT2*** e ***IDCT2***.

L'esposizione di questa seconda parte del progetto procederà come segue. In primo luogo verrà presentata l'applicazione e la sua implementazione. In secondo luogo verranno mostrati e commentati alcuni confronti fra immagini originali e immagini compresse.

Applicazione

In questo paragrafo verrà presentata l'applicazione realizzata. In primo luogo verrà mostrata e descritta l'interfaccia utilizzata dall'applicazione. In secondo luogo verranno forniti i dettagli implementativi dell'algoritmo di compressione.

Cominciamo con l'interfaccia grafica. L'interfaccia grafica è mostrata nella Figura 11.

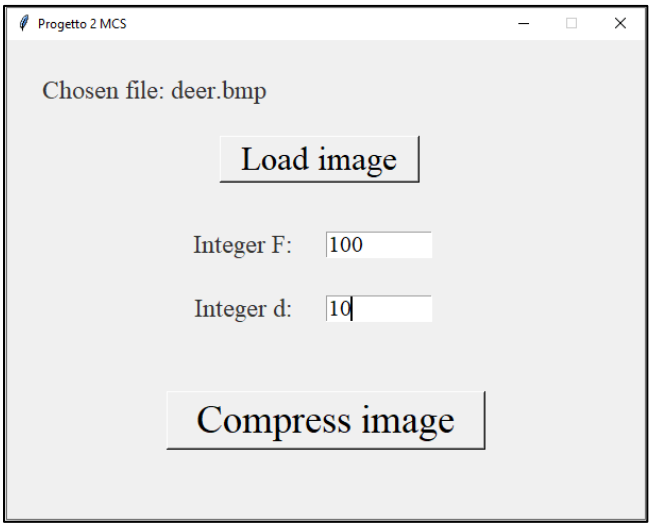

Figura 11 - Interfaccia grafica

L'interfaccia presenta i seguenti elementi:

- l'indicazione dell'immagine scelta per la compressione;
- un bottone per scegliere l'immagine da comprimere;
- un widget deputato all'accettazione di un numero intero come valore ***F*** (rappresentante l'ampiezza dei macro-blocchi);
- un widget deputato all'accettazione di un numero intero come valore ***d*** (rappresentante la soglia di taglio delle frequenze);
- un bottone per avviare la compressione dell'immagine.

Una volta effettuata la compressione vengono mostrate una a fianco all'altra l'immagine originale e quella ricostruita a seguito della compressione.

Passiamo ora all'implementazione. La parte principale dell'implementazione riguarda la funzione di compressione (e decompressione) dell'immagine. Questa funzione è riportata nella Figura 12.

```
def compress_command(self):
        '''
        Funzione per la compressione di immagini in toni di grigio.

        :param self: variabili e funzioni della classe App
        '''

        # Controlla se gli input sono corretti
        correct_inputs = self.check_inputs()

        # Se gli input sono corretti si procede alla compressione dell'immagine
        if (correct_inputs):

            # Legge l'immagine originale dal path memorizzato
            image = cv2.imread(self.chosen_image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

            # Converte l'immagine in scala di grigi
            image_pil = Image.fromarray(image)
            image_greyscale = image_pil.convert('L')
            image_matrix = np.array(image_greyscale)

            # Recupera altezza e larghezza dell'immagine
            height = image_matrix.shape[0]
            width = image_matrix.shape[1]

            # Se l'altezza non è multiplo di f, si tronca al maggior valore multiplo
            # di f
            if (height % self.f_value != 0):
                height = height - (height % self.f_value)

            # Se la larghezza non è multiplo di f, si tronca al maggior valore multiplo
            # di f
            if (width % self.f_value != 0):
```

```
            width = width - (width % self.f_value)

        # Comprime l'immagine in base ad altezza e larghezza troncate
        image_matrix_truncated = image_matrix[0:height , 0:width]
        height = image_matrix_truncated.shape[0]
        width = image_matrix_truncated.shape[1]

        # Crea blocchi F x F
        blocks = self.create_blocks(height, width, image_matrix_truncated)

        # Trasforma i blocchi con DCT2 e IDCT2
        compressed_blocks = self.compress_blocks(blocks)

        # Ricostruisce la matrice con i valori calcolati
        image_matrix_compressed = self.rebuild_image_matrix(height, width,
                                  image_matrix_truncated, compressed_blocks)

        # Trasforma la matrice in un'immagine
        image_compressed = Image.fromarray(image_matrix_compressed.astype('uint8'))
        image_compressed = image_compressed.convert('L')

        # Salva l'immagine compressa
        index = self.chosen_image_path.rfind("/") + 1
        path = self.chosen_image_path[:index]
        self.compressed_image_path = path + "compressed_image.bmp"
        imageio.imsave(self.compressed_image_path, image_compressed)

        # Mostra a schermo l'immagine originale e quella compressa
        self.show_images()
```

Figura 12 - Funzione di compressione dell'immagine

La funzione di compressione dell'immagine si avvale di diverse funzioni di supporto. La divisione dell'implementazione in moduli separati facilita la manutenzione del codice e il debugging. Le funzioni di supporto sono le seguenti:

1. Funzione per il controllo degli input.
   Questa funzione verifica la correttezza dei dati forniti in input all'applicazione. In particolare:
   - deve essere caricata un'immagine per la compressione e la decompressione;
   - deve essere specificato un valore ***F*** che deve essere un intero maggiore o uguale a 0 che non superi le dimensioni di altezza e larghezza dell'immagine;
   - deve essere specificato un valore ***d*** che deve essere un intero compreso fra 0 e $(2F - 2)$.

   Qualora i dati forniti in input non fossero corretti, viene mostrato un apposito messaggio di errore.
2. Funzione per la creazione di blocchi $F \times F$. Questa funzione è riportata nella Figura 13.

```
    def create_blocks(self, height, width, image_matrix_truncated):
        '''
        Funzione per la creazione di blocchi F x F.

        :param self: variabili e funzioni della classe App
        :param height: altezza dell'immagine originale
        :param width: larghezza dell'immagine originale
        :param image_matrix_truncated: matrice dell'immagine troncata
        :return blocks: blocchi F x F
        '''

        # Crea blocchi F x F
        blocks = []
        for i in range(0, height, self.f_value):
            for j in range(0, width, self.f_value):

                # Prende i valori di un blocco F x F
                values = image_matrix_truncated[i:i+self.f_value,
                                                j:j+self.f_value]

                # Crea un blocco F x F con i precedenti valori
                block = np.array(values).reshape(self.f_value,
                                                 self.f_value)

                # Aggiunge il blocco alla lista dei blocchi
                blocks.append(block)

        return blocks
```

Figura 13 - Funzione per la creazione di blocchi F×F

Questa funzione riceve in ingresso una matrice che rappresenta l'immagine originale. Questa matrice risulta già troncata in modo che sia l'altezza che la larghezza siano multipli di ***F***. Gli elementi in eccesso sono eliminati. Questa funzione crea una lista di blocchi $F \times F$. I blocchi sono creati partendo da in alto a sinistra. I blocchi sono inseriti nella lista nell'ordine in cui sono creati.

3. Funzione per la compressione dei blocchi. Questa funzione è riportata nella Figura 14.

```
    def compress_blocks(self, blocks):
        '''
        Funzione per la compressione e decompressione dei blocchi.

        :param self: variabili e funzioni della classe App
        :param blocks: blocchi F x F
        :return compressed_blocks: blocchi F X F ottenuti con DCT2 e
        IDCT2, con frequenze tagliate in base al valore d, e con i
        valori arrotondati all'intero più vicino compreso tra 0 e 255
        '''

        # Trasforma i blocchi con DCT2 e IDCT2
        compressed_blocks = []
        for k in range(len(blocks)):
```

```
                # Recupera un singolo blocco dalla lista dei blocchi
                block = blocks[k]

                # Esegue la DCT2 sul singolo blocco
                c = dct(dct(block.T, norm='ortho').T, norm='ortho')

                # Elimina le frequenze per i + j >= d
                for i in range(0, self.f_value):
                    for j in range(0, self.f_value):
                        if (i + j >= self.d_value):
                            c[i][j] = 0

                # Esegue la IDCT2 sul singolo blocco
                f = idct(idct(c.T, norm='ortho').T, norm='ortho')

                # Arrotonda i valori di f all'intero più vicino
                f = np.rint(f)

                # Mette a 0 i numeri negativi e a 255 quelli maggiori
                di 255
                for i in range(0, self.f_value):
                    for j in range(0, self.f_value):
                        if (f[i][j] < 0):
                            f[i][j] = 0
                        if (f[i][j] > 255):
                            f[i][j] = 255

                # Aggiunge il blocco compresso alla lista dei blocchi
                compressi
                compressed_blocks.append(f)

        return compressed_blocks
```

Figura 14 - Funzione per la compressione dei blocchi

Questa funzione rappresenta il cuore della compressione. In particolare questa funzione riceve in ingresso la lista di blocchi $F \times F$ appena specificata. Questa funzione opera le seguenti operazioni su ogni blocco:

- applica la ***DCT2*** sul blocco;
- elimina le frequenze c_{kl} tali che $k + l \geq d$;
- applica la ***IDCT2*** sul blocco;
- arrotonda tutti i valori del blocco all'intero più vicino;
- imposta a 0 i valori negativi e a 255 i valori maggiori di 255.

4. Funzione per la ricostruzione dell'immagine. Questa funzione è riportata nella Figura 15.

```
    def rebuild_image_matrix(self, height, width,
                             image_matrix_compressed, blocks):
        '''
        Funzione per ricostruire la matrice di blocchi F x F
```

```
        in vista della composizione dell'immagine compressa.

        :param self: variabili e funzioni della classe App
        :param height: altezza dell'immagine originale
        :param width: larghezza dell'immagine originale
        :param image_matrix_compressed: matrice dell'immagine da
         restituire
        :param blocks: blocchi F x F compressi
        :return image_matrix_compressed: matrice dell'immagine con i
         blocchi compressi
        '''

        # Ricostruisce la matrice dell'immagine con i valori calcolati
        k = 0
        for i in range(0, height, self.f_value):
            for j in range(0, width, self.f_value):
                image_matrix_compressed[i:i+self.f_value,
                                        j:j+self.f_value] = blocks[k]
                k += 1
        return image_matrix_compressed
```

Figura 15 - Funzione per la ricostruzione dell'immagine

Questa funzione si limita a ricostruire una matrice a partire dalla lista dei blocchi $F \times F$ ottenuta a seguito di compressione e decompressione tramite ***DCT2*** e ***IDCT2***.

5. Funzione per mostrare l'immagine originale e l'immagine ricostruita affiancate.
   Infine questa funzione mostra in una nuova finestra l'immagine originale e quella ricostruita una a fianco all'altra. Qualora le immagini siano troppo grandi, vengono ridimensionate.

Risultati

Sin qui è stata presentata l'applicazione utilizzata per la compressione di immagini in toni di grigio. Si procede ora a mostrare i risultati di alcuni esperimenti effettuati con questa applicazione.

- Primo esperimento. Il primo esperimento riguarda la compressione di un'immagine con un valore ***d*** pari a 0. Il risultato è mostrato nella Figura 16.

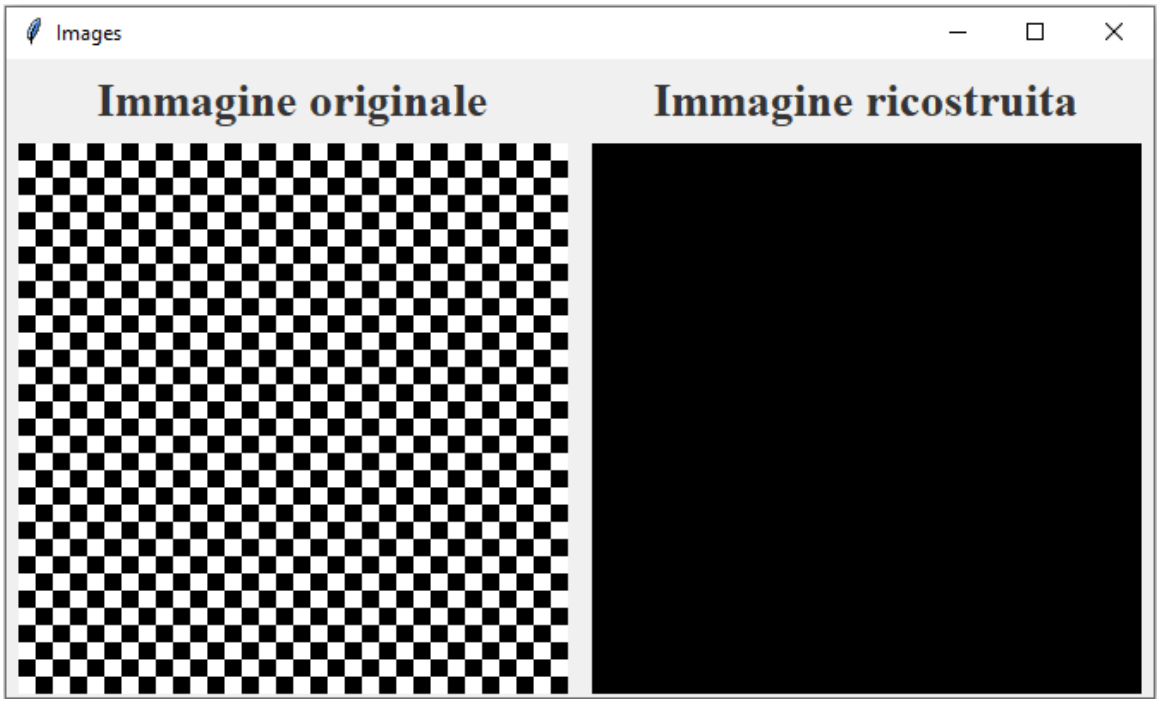

Figura 16 - Compressione immagine con d=0

In questo caso è stata utilizzata un'immagine a scacchiera di dimensioni 320×320. Inoltre sono stati utilizzati un valore ***F*** pari a 320 e un valore ***d*** pari a 0. Il risultato ottenuto dopo la compressione e la decompressione è un'immagine completamente nera. Questo risultato è coerente con le aspettative. Infatti se ***d*** è pari a 0, allora tutte le frequenze ottenute tramite ***DCT2*** vengono tagliate, ovvero vengono poste uguali a 0. E al valore 0 è associato il colore nero. Pertanto il risultato finale non può che essere un'immagine completamente nera.

- Secondo esperimento. Il secondo esperimento riguarda invece la compressione di un'immagine con un valore ***d*** pari a $(2F - 2)$, ovvero il valore di ***d*** più alto possibile. Il risultato è mostrato nella Figura 17.

Figura 17 - Compressione immagine con valore massimo di d

In questo caso è stata utilizzata un'immagine di dimensioni 1011×661. Inoltre sono stati utilizzati un valore ***F*** pari a 10 e un valore ***d*** pari a 18. In questo caso l'immagine ricostruita è indistinguibile a occhio nudo dall'immagine originale. Anche questo risultato è coerente con le aspettative. Infatti avendo tagliato il minor numero possibile di frequenze, la qualità dell'immagine ricostruita è la migliore possibile.

- Terzo esperimento. Il terzo esperimento riguarda la compressione di un'immagine con un valore ***F*** che non sia multiplo delle dimensioni di altezza e/o larghezza dell'immagine originale. Il risultato è mostrato nella Figura 18.

Figura 18 - Compressione con F non multiplo delle dimensioni dell'immagine

In questo caso è stata utilizzata un'immagine di dimensioni 1011×661. Inoltre sono stati utilizzati un valore ***F*** pari a 515 e un valore ***d*** pari a 515. In questo caso si nota chiaramente

che l'immagine ricostruita risulta tagliata. Infatti poiché ***F*** è pari a 515, è stato possibile ottenere solo un blocco di dimensioni 515×515 a partire dalla parte in alto a sinistra dell'immagine originale. Il resto dell'immagine originale è stato tagliato.

- Quarto esperimento. Il quarto esperimento riguarda la compressione di un'immagine che presenta contrasti forti utilizzando un valore ***d*** alto. Il risultato è mostrato nella Figura 19.

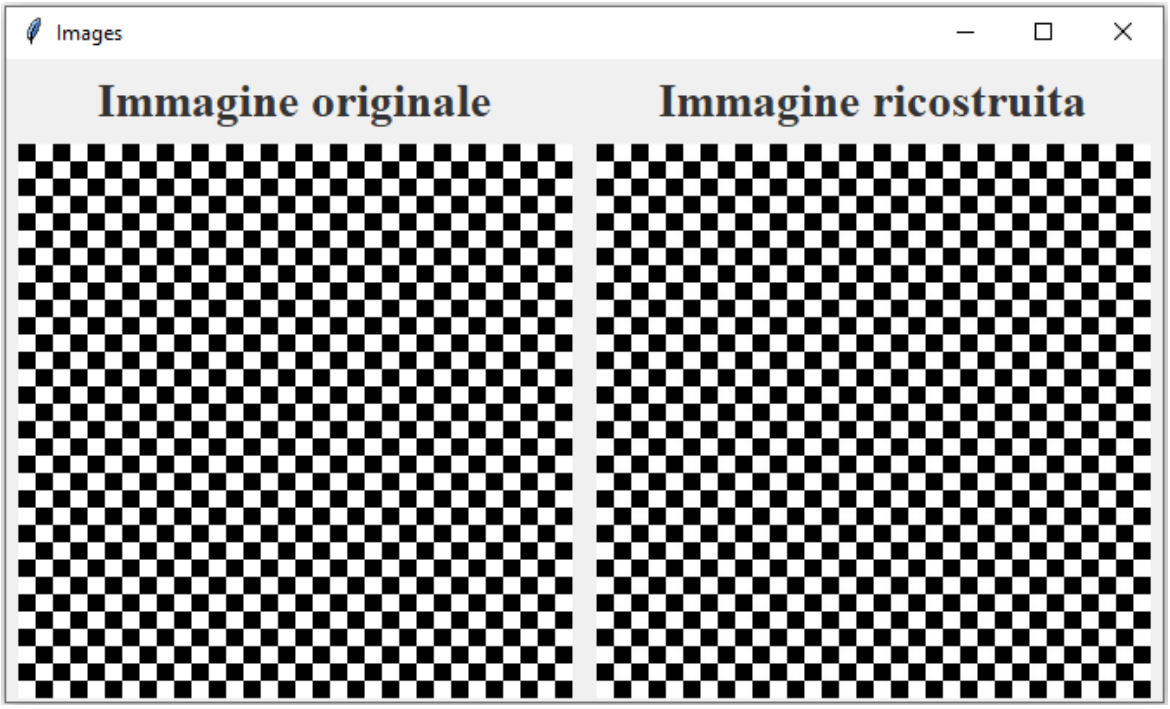

Figura 19 - Compressione immagine con contrasti forti e valore d alto

In questo caso è stata utilizzata un'immagine di dimensioni 320×320. Inoltre sono stati utilizzati un valore ***F*** pari a 40 e un valore ***d*** pari a 70. L'immagine ricostruita è indistinguibile da quella originale a occhio nudo.

- Quinto esperimento. Il quinto esperimento riguarda la compressione di un'immagine che presenta contrasti forti utilizzando un valore ***d*** basso. Il risultato è mostrato nella Figura 20.

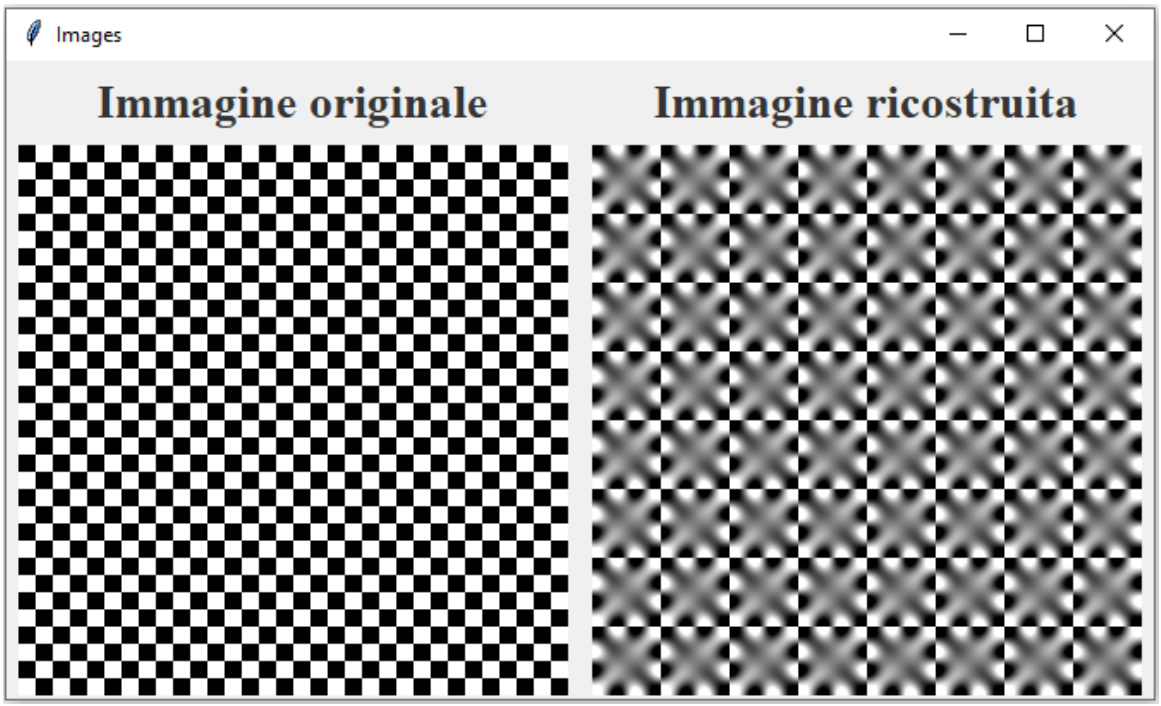

Figura 20 - Compressione immagine con contrasti forti e valore d basso

In questo caso è stata utilizzata un'immagine di dimensioni 320×320. Inoltre sono stati utilizzati un valore ***F*** pari a 40 e un valore ***d*** pari a 7. In questo caso l'immagine ricostruita è molto differente da quella originale. In particolare questa differenza è evidente in corrispondenza dei punti in cui l'immagine passa dal bianco al nero. Questa differenza è dovuta al fatto che la funzione che rappresenta i valori puntuali dell'immagine presenta delle discontinuità di prima specie (dei "salti") in corrispondenza dei passaggi dal bianco al nero e viceversa. In questo caso in corrispondenza di questi punti di discontinuità la ***DCT2*** produce valori intermedi tra il bianco e il nero. Per questo motivo si nota una forte presenza di grigi nell'immagine ricostruita: grigi che nell'immagine originale erano assenti.

- Sesto esperimento. Il sesto esperimento riguarda la compressione di un'immagine che presenta contrasti deboli utilizzando un valore ***d*** basso. Il risultato è mostrato nella Figura 21.

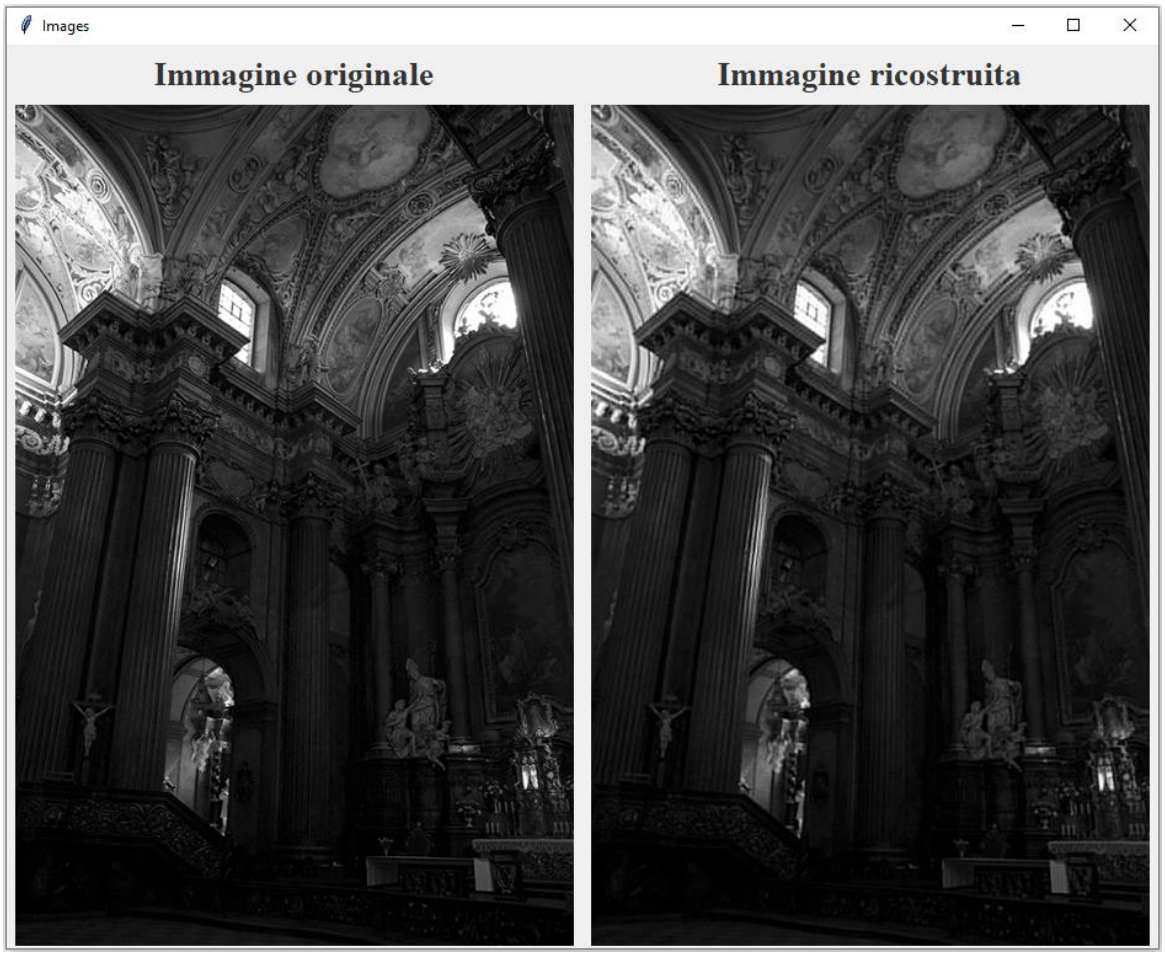

Figura 21 - Compressione immagine con contrasti deboli e valore d basso

In questo caso è stata utilizzata un'immagine di dimensioni 2000×3008. Inoltre sono stati utilizzati un valore ***F*** pari a 40 e un valore ***d*** pari a 7. A differenza dell'immagine precedente, in questo caso si notano differenze meno marcate dell'immagine ricostruita rispetto a quella originale, nonostante i parametri ***F*** e ***d*** siano identici. Il motivo è che l'immagine utilizzata in questo caso presenta contrasti meno forti; e presenta un vasto uso di grigi. Pertanto non vi sono discontinuità tali da generare valori dell'immagine ricostruita completamente diversi da quelli originali. In ogni caso il basso valore di ***d*** comporta che l'immagine ricostruita presenti diversi punti di differenza rispetto all'immagine originale. Pertanto anche a occhio nudo si possono cogliere alcune differenze. E naturalmente si colgono nei punti di maggior contrasto tra le diverse gradazioni di grigio.

- Settimo esperimento. Il settimo esperimento invece riguarda la compressione di un'immagine che presenta contrasti forti utilizzando un valore ***d*** basso e ***F*** pari alla dimensione dell'immagine. Il risultato è mostrato nella Figura 22.

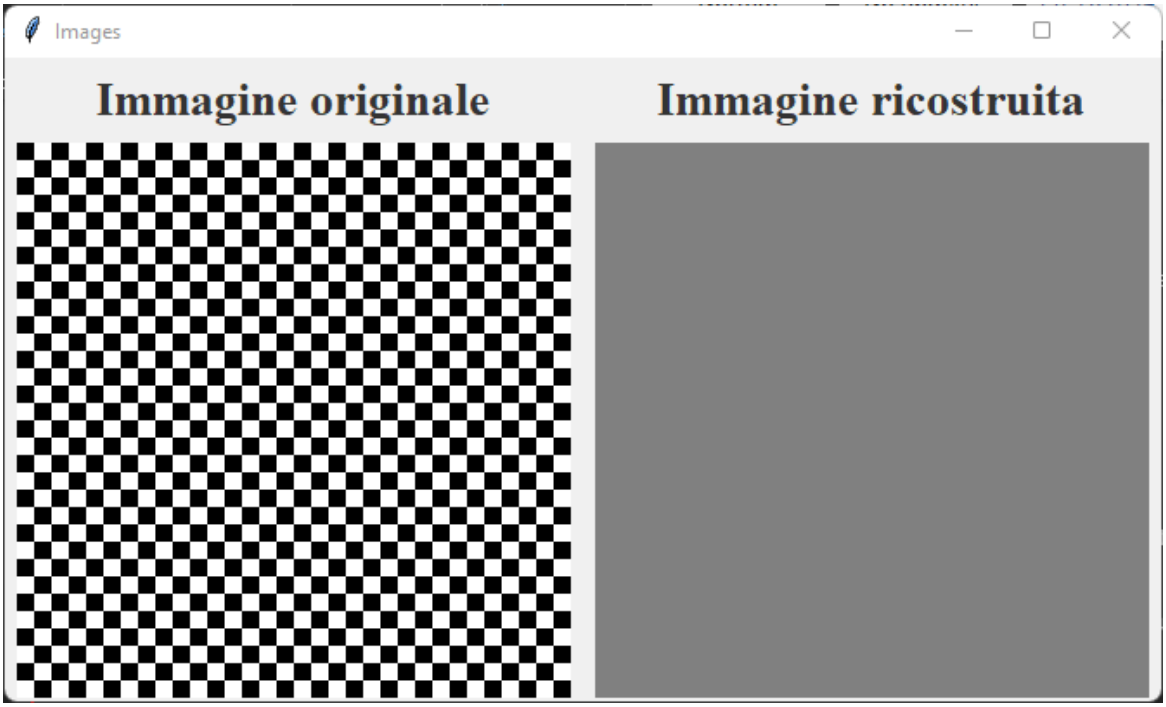

Figura 22 - Compressione immagine con contrasti forti, valore d basso e valore di F massimo

In questo caso è stata utilizzata un'immagine di dimensioni 320×320. Sono stati utilizzati come riportato precedentemente un valore ***F*** pari alla dimensione (320) e un valore ***d*** pari a 5. In questo caso per coerenza ci aspettiamo che l'immagine ricostruita presenti sostanzialmente un'immagine totalmente grigia. A differenza del primo esperimento, dove l'immagine risultava completamente nera, in questo caso l'immagine risulta grigia in quanto il valore di ***d*** non è pari a 0 facendo sì che non tutte le frequenze vengano tagliate.

- Ottavo esperimento. L'ultimo esperimento riguarda la compressione di un'immagine che presenta sia contrasti forti che contrasti deboli utilizzando un valore ***d*** basso. Il risultato è mostrato nella Figura 23.

Figura 23 - Compressione immagine con contrasti sia forti che deboli e valore d basso

In questo caso è stata utilizzata un'immagine di dimensioni 1011×661. Inoltre sono stati utilizzati un valore ***F*** pari a 90 e un valore ***d*** pari a 15. In questo caso si notano differenze marcate tra le due immagini in corrispondenza dei contrasti più forti, e differenze meno marcate in corrispondenza dei contrasti più deboli. Naturalmente i contrasti più deboli si riscontrano in corrispondenza dei grigi e dei blocchi caratterizzati da colore uniforme.
Per rendere più evidente il risultato della compressione (e decompressione) in corrispondenza dei contrasti più forti, si riporta anche un dettaglio dell'immagine ricostruita con un maggior ingrandimento. Questo dettaglio è mostrato nella Figura 24.

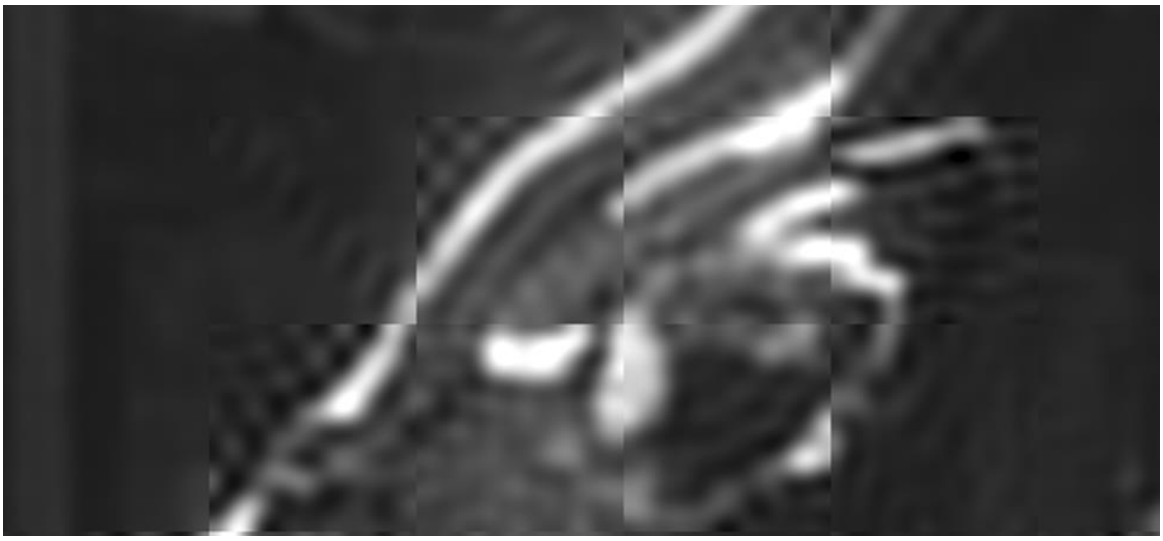

Figura 24 - Dettaglio compressione su contrasto forte con valore d basso

Riferimenti

[1] SciPy. [Online]. Available: https://scipy.org/. [Consultato il giorno 28 giugno 2022].

[2] SciPy.fft. [Online]. Available: https://docs.scipy.org/doc/scipy/tutorial/fft.html. [Consultato il giorno 28 giugno 2022].

[3] NumPy. [Online]. Available: https://numpy.org/. [Consultato il giorno 28 giugno 2022].

[4] Matplotlib. [Online]. Available: https://matplotlib.org/. [Consultato il giorno 28 giugno 2022].

[5] TkInter. [Online]. Available: https://wiki.python.org/moin/TkInter. [Consultato il giorno 28 giugno 2022].

[6] Pillow. [Online]. Available: https://pillow.readthedocs.io/en/stable/. [Consultato il giorno 28 giugno 2022].

[7] OpenCV-Python. [Online]. Available: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html. [Consultato il giorno 28 giugno 2022].

[8] Imageio. [Online]. Available: https://imageio.readthedocs.io/en/v2.19.3/index.html. [Consultato il giorno 28 giugno 2022].