

Relazione del progetto 2 di Metodi del Calcolo Scientifico

Studente: Daniel Scalena - 844608 - d.scalena@campus.unimib.it

Corso: Metodi del calcolo scientifico

Contents

1 Overview	2
1.1 Breve introduzione alla DCT	2
1.2 Applicazione della DCT	2
1.3 Linguaggio di programmazione e libreria selezionata	4
2 Prima parte	6
2.1 Test e hardware impiegato	6
2.2 Implementazione e risultati	7
3 Seconda parte	11
3.1 Implementazione	11
3.2 Risultati ottenuti	13
4 Conclusioni	24

1 Overview

Viene di seguito affrontata una breve introduzione e descrizione dei problemi presentati, nonché un breve resoconto riguardo la documentazione ufficiale dei tool e delle librerie utilizzate sia per la prima che per la seconda parte del progetto.

1.1 Breve introduzione alla DCT

La DCT, abbreviazione di Discrete Cosine Transform mira ad esprimere una determinata sequenza come una sommatoria di coseni oscillanti a diverse frequenze. Il vantaggio di questa trasformazione sta nella possibilità di eliminare alcune di queste frequenze, comprimendo di fatto un qualsiasi media in un formato più piccolo, cercando allo stesso tempo di non perdere un'eccessiva qualità dello stesso. Gli impieghi più comuni per questa transformazioni sono le immagini (generalmente per formati di JPEG e HEIF, eliminando piccoli componenti ad alta frequenza), video (per formati MPEG e H26x,), audio (MP3 e AAC) e molti altri formati, generalmente relativi alla trasmissione di dati.

Nell'ambito di questo progetto la DCT viene applicata per la compressione di immagini, prendendo i singoli pixel e trasformandoli in frequenze. Si parla di compressione dato che, alcune delle frequenze ottenute possono essere cancellate, comprimendo di fatto la dimensione in memoria dell'immagine in questione. Nel concreto, essendo l'immagine un media di tipo bidimensionale (rappresentabile con una matrice), viene applicata la DCT2, ovvero l'estensione bidimensionale della DCT fatta sulle colonne e sulle righe della matrice di pixel. È possibile applicare la funzione inversa della DCT (o DCT2) per poter ricostruire l'immagine fatta di pixel partendo dalle frequenze salvate (eventualmente minori rispetto all'originale in caso di compressione), potendo quindi decomprimere l'immagine.

1.2 Applicazione della DCT

L'obiettivo della DCT è il cambiamento della base dal vettore di partenze $f = (f_0, \dots, f_{N-1})$ dei componenti rispetto alla base canonica $\{e_0, \dots, e_{N-1}\}$ per ottenere il vettore dei coefficienti (c_0, \dots, c_{N-1}) nella base ortonormale $\{w_0, \dots, w_{N-1}\}$.

Formalmente:

$$\text{DCT}: \{f_i\} \mapsto \{c_k\}$$

$$f = \sum_{i=0}^{N-1} f_i e_i = \sum_{k=0}^{N-1} c_k w_k$$

Dove w_k è il k -esimo componente della base ortonormale $\{w_0, \dots, w_{N-1}\}$. Sfruttando l'ortonormalità di entrambe le basi è possibile trasformare il vettore f nei coefficienti c moltiplicando scalarmemente per w_l

$$\sum_{i=0}^{N-1} f_i (e_i \cdot w_l) = \sum_{k=0}^{N-1} c_k (w_k \cdot w_l)$$

Avendo $e_i \cdot w_l$ come la componente i -esima di w_l dove $w_l = \alpha_l^N \cos(\pi l t_i) = \alpha_l^N \cos(\pi l \frac{2i+1}{2N})$; nella pratica passo da f_i , nei punti t_i calcolati come N intervalli da $\frac{2i+1}{2N}$, alla nuova base ortonormale le cui entrate corrispondono alla valutazione della funzione $\cos(\pi k t)$ e avendo $w_k \cdot w_l = \delta_{kl} = 0$ se $k \neq l$ o $= 1$ se $k = l$ posso scrivere:

$$\sum_{i=0}^{N-1} f_i \alpha_l^N \cos\left(\pi l \frac{2i+1}{2N}\right) = c_l, \quad l = 0, \dots, N-1$$

riscrivibile come la formula generale per la **DCT**:

$$c_k = \alpha_k^N \sum_{i=0}^{N-1} f_i \cos\left(\pi k \frac{2i+1}{2N}\right), \quad k = 0, \dots, N-1$$

Avendo quindi f rappresentato come una combinazione lineare di coseni valutati nei punti t_i .

È possibile definire l'operazione inversa della DCT che, dal vettore dei coefficienti, riporta al vettore f originale. L'operazione inversa è chiamata **IDCT** ed è formalmente definita come:

$$\text{IDCT: } \{c_k\} \mapsto \{f_i\}$$

ovvero:

$$f_i = \sum_{k=0}^{N-1} c_k \alpha_k^N \cos\left(\pi k \frac{2j+1}{2N}\right), \quad j = 0, \dots, N-1$$

L'operazione di DCT è riscrivibile in forma compatta matriciale definendo una matrice D | $c = Df$ dove:

$$c_k = \sum_{i=0}^{N-1} D_{ki} f_i$$

e con

$$D_{ki} = \alpha_k^N \cos\left(\pi k \frac{2i+1}{2N}\right)$$

e ovviamente l'inversa operazione $f = D^{-1}c$ con:

$$f_i = \sum_{k=0}^{N-1} D_{ik}^{-1} c_k$$

e con

$$D_{ik}^{-1} = \alpha_k^N \cos\left(\pi k \frac{2i+1}{2N}\right)$$

con $D^{-1} = D^T$ dato che D è ortonormale, avendo una rappresentazione più efficiente considerando la bassa complessità di calcolare una matrice trasposta se confrontata con la complessità di calcolare l'inversa.

1.3 Linguaggio di programmazione e libreria selezionata

Come per il primo progetto si è scelto di proseguire l'implementazione su Python, ampiamente utilizzato nella comunità scientifica per la sua flessibilità e semplicità. Anche la libreria rimane la medesima con SciPy che mette a disposizione un pacchetto dedicato alle trasformate di Fourier chiamato fftpack, dove all'interno vengono implementate `dct` e la sua inversa `idct`.

Seguendo quanto visto a lezione viene mantenuto il parametro `norm = 'ortho'` per quanto scritto di seguito nella documentazione:

For norm="ortho" both the dct and idct are scaled by the same over-all factor in both directions. By default, the transform is also orthogonalized which for types 1, 2 and 3 means the transform definition is modified to give orthogonality of the DCT matrix [...].

Vengono quindi testate le funzioni secondo quanto indicato nella consegna. In 1 è presente l'output della verifica per il vettore dato, in 2 per la matrice data.

4.1e+02	6.6E+00	1.1E+02	-1.1E+02	6.5E+01	1.2E+02	1.2E+02	2.8E+01
---------	---------	---------	----------	---------	---------	---------	---------

Table 1: Output per la verifica della dct monodimensionale

```

1 d = np.array(dct(np.array([231, 32, 233, 161, 24, 71, 140, 245]),
      norm = "ortho"))
2 ["%E" % ele for ele in d.flat]

```

Listing 1: Codice python per la verifica della funzione dct sul vettore fornito

```

1 matrix = np.array(
2     [
3         [231, 32, 233, 161, 24, 71, 140, 245],
4         [247, 40, 248, 245, 124, 204, 36, 107],
5         [234, 202, 245, 167, 9, 217, 239, 173],
6         [193, 190, 100, 167, 43, 180, 8, 70],
7         [11, 24, 210, 177, 81, 243, 8, 112],
8         [97, 195, 203, 47, 125, 114, 165, 181],
9         [193, 70, 174, 167, 41, 30, 127, 245],
10        [87, 149, 57, 192, 65, 129, 178, 228]
11    ]
12 )
13 dct(dct(matrix.T, norm="ortho").T, norm="ortho")

```

Listing 2: Codice python per la verifica della funzione dct sulla matrice fornita

+1.1e+03	+4.4e+01	+7.5e+01	-1.3e+02	+3.5e+00	+1.2e+02	+1.9e+02	-1.0+02
+7.7e+01	+1.1e+02	-2.1e+01	+4.1e+01	+8.7e+00	+9.9e+01	+1.3e+02	+1.0+01
+4.4e+01	-6.3e+01	+1.1e+02	-7.6e+01	+1.2e+02	+9.5e+01	-3.9e+01	+5.8+01
-6.9e+01	-4.0e+01	-2.3e+01	-7.6e+01	+2.6e+01	-3.6e+01	+6.6e+01	+1.2+02
-1.1e+02	-4.3e+01	-5.5e+01	+8.1e+00	+3.0e+01	-2.8e+01	+2.4e+00	-9.4+01
-5.3e+00	+5.7e+01	+1.7e+02	-3.5e+01	+3.2e+01	+3.3e+01	-5.8e+01	+1.9+01
+7.8e+01	-6.5e+01	+1.1e+02	-1.5e+01	-1.3e+02	-3.0e+01	-1.0e+02	+3.9+01
+1.9e+01	-7.8e+01	+9.7e-01	-7.2e+01	-2.1e+01	+8.1e+01	+6.3e+01	+5.9+00

Table 2: Output per la verifica della dct bidimensionale

I risultati sono molti simili, eccezion fatta per alcuni piccoli errori di approssimazione dovuti a specifici flag interni alla libreria.

2 Prima parte

Viene chiesto di implementare una DCT2 fatta in casa per come è stata vista durante le lezioni ed effettuare un confronto con l'implementazione fornita da una libreria open source, nella versione FFT (fast).

Successivamente alla fase di test viene chiesto di illustrare i tempi di risoluzione, ipotizzando che questi siano proporzionali a $O(N^3)$ per la DCT2 costruita manualmente e $O(N^2 \log(N))$ per la versione fast, dove N è la dimensione della matrice quadrata.

2.1 Test e hardware impiegato

I test sono stati effettuati su diverse matrici con diverse dimensioni (N, N) . Nel dettaglio vengono riprese matrici a scacchiera simili a quelle presenti nella cartella immagini del progetto. Vengono ulteriormente testate matrici "particolari" costruite appositamente viste durante il corso delle lezioni, non solo relativamente alla seconda parte sulla DCT. La prima di questa è una matrice a scacchiera, con elementi random e 0 alternati; la seconda è a prevalenza diagonale con una struttura sparsa, dove la stessa è composta da uni sulla diagonale e 3 negli elementi della sotto-diagonale e sopra-diagonale. Le ultime due matrici testate sono una incrementale dove, seguendo l'ordine riga-colonna, viene aumentato di uno il valore fino al raggiungimento della fine della matrice, e la seconda dove vengono inseriti valori random. In figura 1 sono presenti gli spy delle relative matrici descritte. Si è quindi cercato di comprendere se la struttura della matrice o la presenza di molti 0 influisca in qualche modo le prestazioni della DCT, sia per quella costruita a mano che per quella fornita dalla libreria.

L'hardware impiegato per il test è descritto in 3. Tutti i test sono stati eseguiti con una versione di Python pari a 3.9 e l'ultima versione di SciPy nativa per l'architettura ARM di Apple Silicon.

Nome: MacBook Pro 16" 2021

CPU: Apple Silicon M1 Max - 10 Core (2 efficency, 8 performance)

GPU: Apple Silicon M1 Max - 32 Core (w/Metal)

RAM: 32 GB

SSD: 1 TB

Table 3: Specifiche tecniche sul laptop usato per i benchmarks

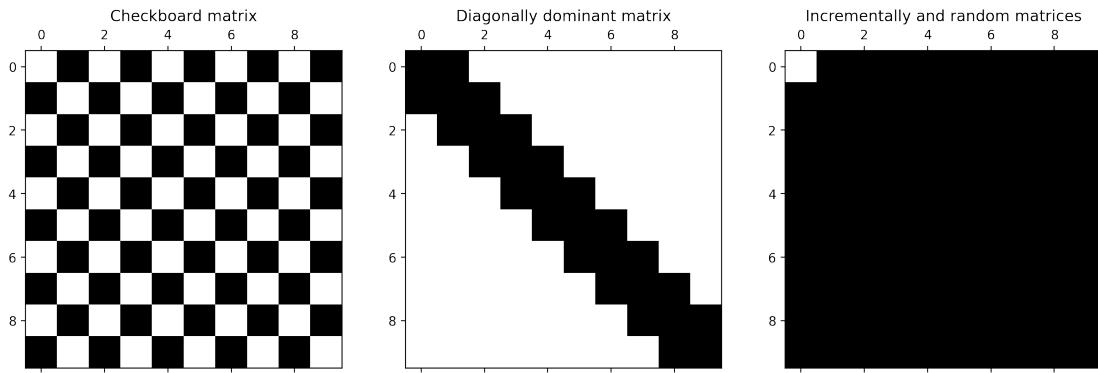


Figure 1: Matrici utilizzate per il test della dct costruita a mano vs quella inclusa nella libreria

2.2 Implementazione e risultati

Si è scelto di implementare la DCT seguendo la sua forma matriciale per questioni di efficienza in termini di calcolo e semplicità implementativa.

Nel seguente listato di codice è presente il ciclo per la creazione D rappresentante la trasformazione DCT, ovvero l'operazione più complessa dal punto di vista temporale. Essendo due cicli innestati, ognuno dei quali scorre N volte, la complessità totale è di $O(N^2)$.

```

1   N = np.array(f_vect).shape
2
3   a = np.zeros(N)
4   a[0] = pow(N, -0.5)
5   a[1:N] = pow(N, -0.5)*np.sqrt(2)
6   D = np.array([np.zeros(N) for i in range(0, N)])
7
8   for r in range(0, N):
9       for c in range(0, N):
10          D[r, c] = a[r]*np.cos((r)*np.pi*(2*c+1)/(2*N))
11 np.dot(D, f_vect)

```

Listing 3: Codice per la creazione della matrice D e la successiva operazione per effettuare la DCT

```

1   from scipy.fftpack import fft, dct
2   # m -> matrice sulla quale effettuare dct su righe e colonne
3   dct(dct(m.T, norm="ortho").T, norm="ortho")

```

Listing 4: implementazione della DCT2 dalla libreria

Allo stesso modo, per quanto riguarda la DCT2 vengono sfruttate le stesse operazioni viste precedentemente, con lo stesso calcolo della matrice D, ma effettuando l'operazione prima sulle righe e poi sulle colonne. Questo porta la complessità pari a $O(2N \cdot 2^N)$ approssimabile a $O(N^3)$.

I risultati ottenuti sono illustrati nelle figure 2, 3, 4 e 5. È possibile quindi concludere che, a parità di dimensione della matrici e indipendentemente dal tipo di matrice utilizzata, i risultati appaiono simili a meno di piccole approssimazioni o variazioni dovute a fattori esterni.

Risulta quindi confermato che l'implementazione fatta manualmente della DCT2 può essere approssimata con un andamento pari a $O(N^3)$ mentre la DCT2 fornita dalla libreria open source ha un andamento pari a circa $O(N^2)$, verificando quanto visto e richiesto dalle ipotesi iniziali.

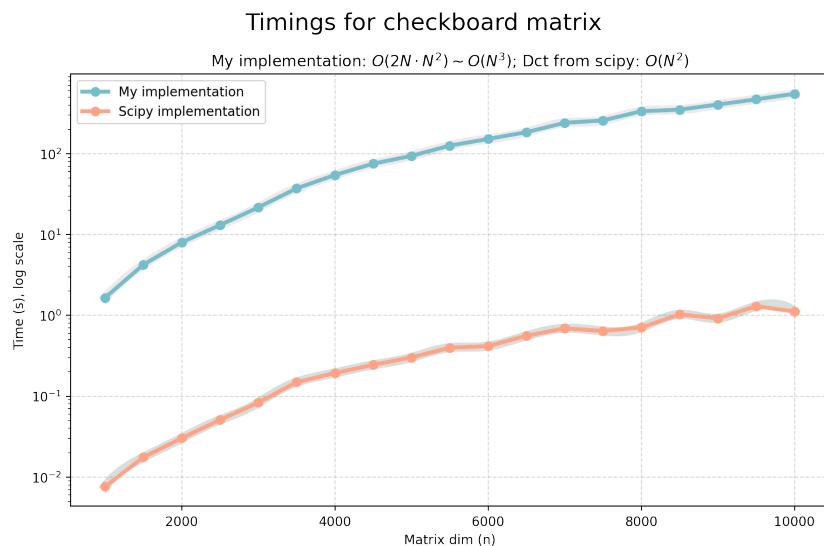


Figure 2: Benchmark algoritmo DCT fatto a mano VS implementazione della libreria SciPy (Checkboard matrix)

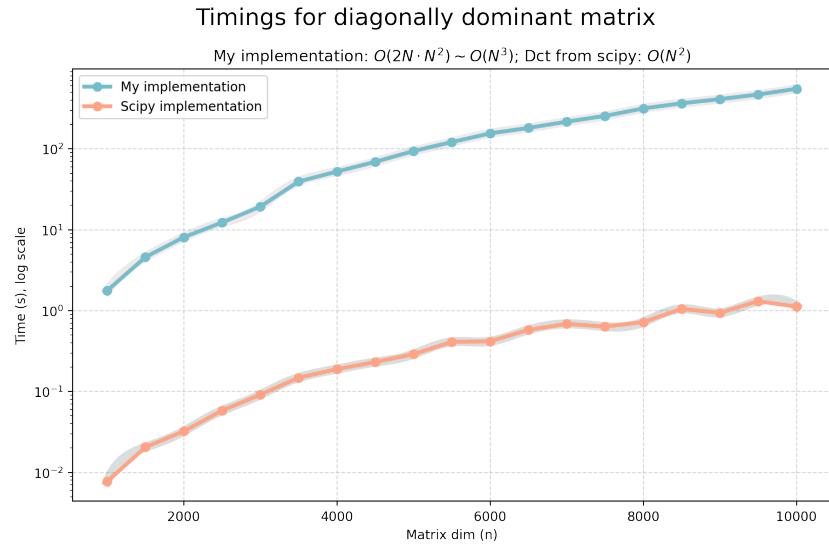


Figure 3: Benchmark algoritmo DCT fatto a mano VS implementazione della libreria SciPy (Diagonally dominant matrix)

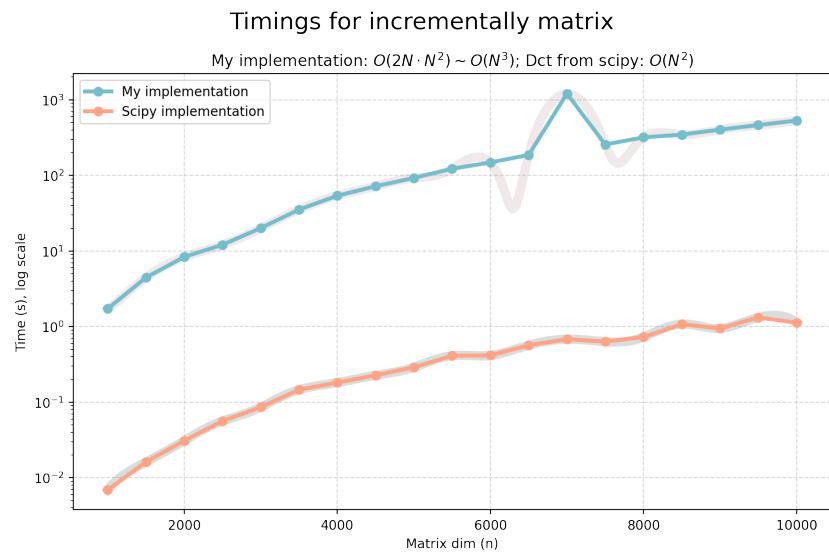


Figure 4: Benchmark algoritmo DCT fatto a mano VS implementazione della libreria SciPy (Incrementally matrix)

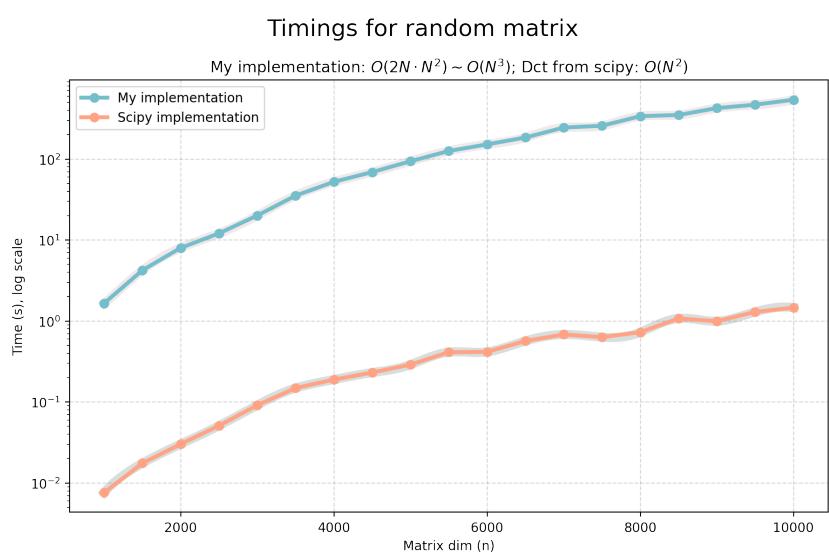


Figure 5: Benchmark algoritmo DCT fatto a mano VS implementazione della libreria SciPy (Random matrix)

3 Seconda parte

La seconda parte del progetto comprende la compressione e la seguente decompressione delle immagini utilizzando le funzioni di DCT e IDCT presentate precedentemente nella libreria di SciPy in Python.

3.1 Implementazione

Come primo passo viene costruita una semplice interfaccia grafica che permette di scegliere un'immagine interna alla memoria del sistema. Viene ulteriormente fornita un'anteprima della stessa e vengono messi a disposizione due slider dove, come indicato nella consegna, è possibile scegliere i valori di F e di d relativi alla successiva compressione. È importante sottolineare che, in base al parametro F selezionato il paramentro d avrà un upper bound dinamico impostato a $2F - 2$. Uno screenshot del programma in questione è fornito in figura 6.

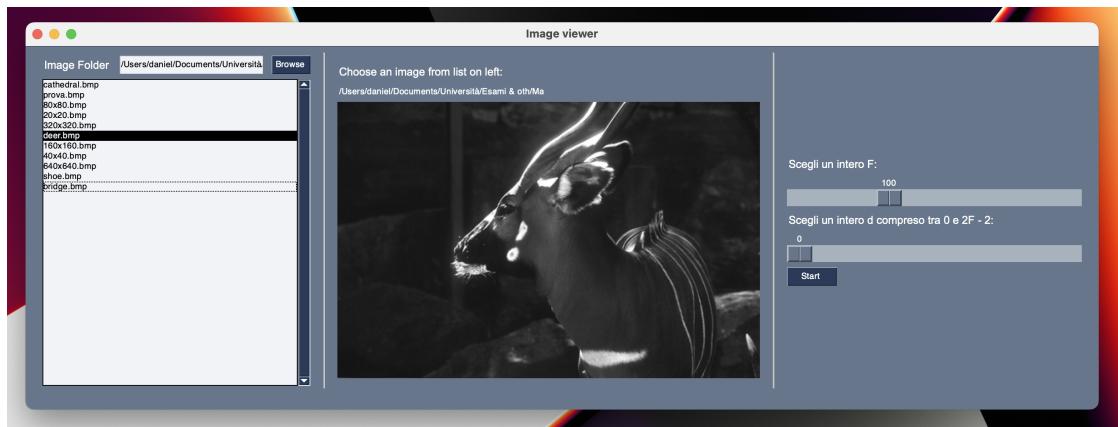


Figure 6: Screenshot dell'interfaccia grafica del programma di selezione dell'immagine

Essendo tutte le immagini presenti nella raccolta in bianco e nero viene fornita un'implementazione standard per la loro lettura attraverso la libreria Open Source PIL; se l'immagine caricata non è in bianco e nero o è in un formato diverso viene comunque effettuata una conversione *lossless* della stessa nel formato compatibile per la compressione con DCT2.

Successivamente l'immagine viene convertita in una matrice e compressa semplicemente passando il suo contenuto alla funzione array di NumPy. SciPy

e NumPy infatti condividono molto del codice sottostante, rendendo quasi tutte le loro strutture dati compatibili in entrambe le direzioni. Quanto detto è espresso nel seguente listato di codice.

```
1     img = Image.open(path_image).convert('L')
2     img_c = compress(np.array(img), F = F, d = d)
```

Listing 5: Codice la lettura e la trasformazione in matrice dell'immagine selezionata dall'interfaccia grafica

Il formato finale della matrice relativa all'immagine perderà quindi tutte le informazioni RGB o RGBA e conterrà soltanto dei valori compresi tra 0 e 256 indicanti la tonalità di grigio di un singolo pixel.

Come visto anche i parametri F e d vengono passati alla funzione `compress` che si occuperà di effettuare la vera e propria DCT2 sulla matrice rappresentante l'immagine selezionata. Nel dettaglio vengono raccolte le dimensioni della matrice, calcolando quindi, in base alla dimensione dei blocchi F selezionata, quanto questa debba essere tagliata; nel seguente modo, prendendo i pixel fino al punto `hc` per l'altezza e `hw` per la larghezza:

```
1     hc = F * (image.shape[0] // F)
2     hw = F * (image.shape[1] // F)
3
4     image = image[:hc, :hw]
```

Viene successivamente effettuata la DCT2 sul singolo blocco, per ogni blocco in cui la matrice originale è stata definita, analogamente al codice visto nella prima parte del progetto affrontato nel capitolo 2. Vengono anche tagliate le frequenze in accordo con quanto osservato nell'esempio nella consegna in base al parametro d selezionato nella fase iniziale.

```
1     for r in range(0, hc, F):
2         for c in range(0, hw, F):
3             dct2_matrix[r:r+F, c:c+F] = dct(dct(image[r:r+F, c:c+F]
4 ].T, norm="ortho").T, norm="ortho")
5
6             for x in range(F):
7                 lb = d - x
8                 # qui vengono tagliate le frequenze
9                 if x > d: lb=0
10                for y in range(lb, F):
11                    dct2_matrix[x+r, y+c] = 0
```

Procedimento analogo viene eseguito per la IDCT2, effettuando quindi l'operazione inversa della DCT e ottenendo dalle frequenze l'immagine originale compressa.

Il ciclo, uguale a quello precedente, chiama al posto della `dct` la `idct` come di seguito:

```
1  for r in range(0, hc, F):
2      for c in range(0, hw, F):
3          idct2_matrix[r:r+F, c:c+F] = idct(idct(dct2_matrix[r:r+F, c:c+F].T, norm="ortho").T, norm="ortho")
```

La compressione si conclude quindi con un arrotondamento di tutti i valori che superano 255 o che sono sotto lo 0.

3.2 Risultati ottenuti

Vengono di seguito messi a confronto diverse immagini prima e dopo la compressione, impostando diversi valori per poterne osservare i cambiamenti. Gli esempi presentati riguardano solo alcuni degli esperimenti effettuati; in ogni caso è consigliato guardare le immagini risultanti attraverso l'interfaccia grafica fornita dal programma stesso per poter verificare le differenze con la qualità massima disponibile.

Immagine 20x20 Partendo dall'immagine più semplice proposta è possibile effettuare delle prove per controllare come la compressione gestisce i salti di colore repentina.

Nel primo caso presentato in figura 7 è possibile notare come, nonostante il parametro di eliminazione delle frequenze d sia impostato a 1, la compressione non sembra avere nessun effetto, né di ridimensionamento né di imperfezione dell'immagine. Questo accade perché la dimensione del blocco, determinata dal parametro F è impostata a 2, ovvero un sottomultiplo della dimensione del blocco nero o bianco nell'immagine (10, 10). Questo fa sì che all'interno del blocco (F, F) non ci siano salti di colore da dover approssimare, rendendo la compressione efficace senza nessuna perdita di qualità.

Medesimo discorso si applicherebbe per qualsiasi sottomultiplo di 10, come mostrato in figura 8 con un F scelto pari a 10.

Impostando un valore di F non sottomultiplo di 10, come ad esempio 3 in figura 9, è possibile notare sia il ridimensionamento dell'immagine, sia la presenza di artefatti lungo i lati dei quadrati neri e bianchi. Questo accade perché ci sono dei blocchi (F, F) che al loro interno hanno più colori e, approssimandoli, creano degli artefatti visibili eliminando il taglio netto del colore presente nell'immagine originale.

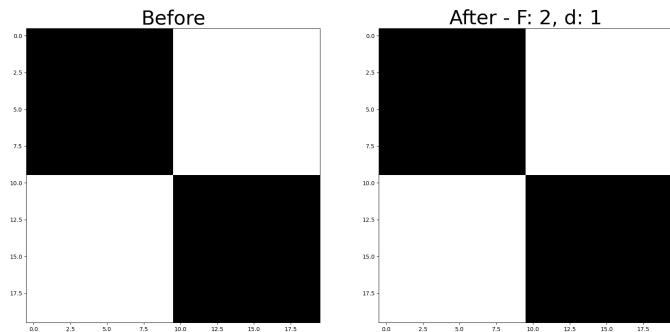


Figure 7: 20x20.bmp - F = 2, d = 1

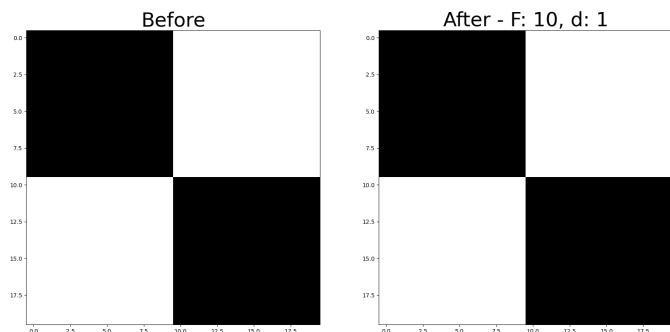


Figure 8: 20x20.bmp - F = 10, d = 1

In questo caso si hanno due diverse tonalità di grigio sfumate, quella più chiara ha un blocco in prevalenza bianco (1 pixel nero e 2 bianchi), quella più scura, al contrario, ha nel blocco più nero (2 pixel neri e 1 bianco).

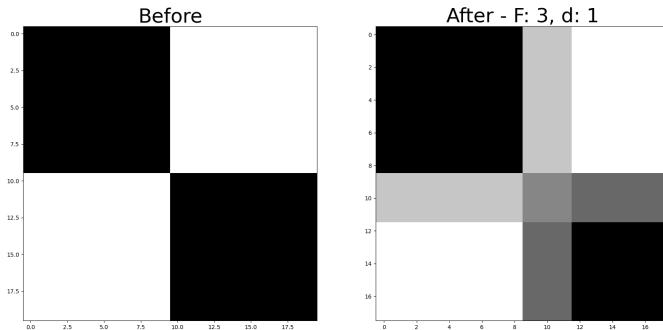


Figure 9: 20x20.bmp - $F = 3$, $d = 1$

Immagine 80x80 Analogamente a quanto visto precedentemente, viene scelta l’immagine a blocchi di dimensione totale (80, 80) per ulteriori test sulla compressione. Nel dettaglio il parametro F viene impostato a 6, piccolo abbastanza da evitare un ridimensionamento eccessivo della stessa e in modo che i blocchi non corrispondano ai quadrati neri e bianchi nell’immagine. Vengono presentati quindi diversi valori per il parametro d indicante la quantità di frequenze da tagliare per blocco.

Nei primi due esempi, in figura 10 e 11 è possibile osservare come la compressione sia decisamente evidente, dovuto al taglio delle frequenze interne ai blocchi della DCT2. In questo caso i salti di colore sono mal rappresentati e gli artefatti rimangono ben visibili.

Importante notare come, relativamente alla figura 11, sulla quale viene effettuato uno zoom in figura 12, il salto di colore tra il secondo blocco nero e il secondo blocco bianco venga ben compresso vista la sua posizione nella colonna 30 di pixel, multiplo del valore 6 scelto come input per F , coerentemente con quanto osservato precedentemente.

Con parametri di d superiori a 5 (figura 13) si iniziano ad ottenere buoni risultati ma anche con d pari a 7, come in figura 14 è comunque possibile notare la presenza di artefatti se si guarda più da vicino i singoli blocchi di colore.

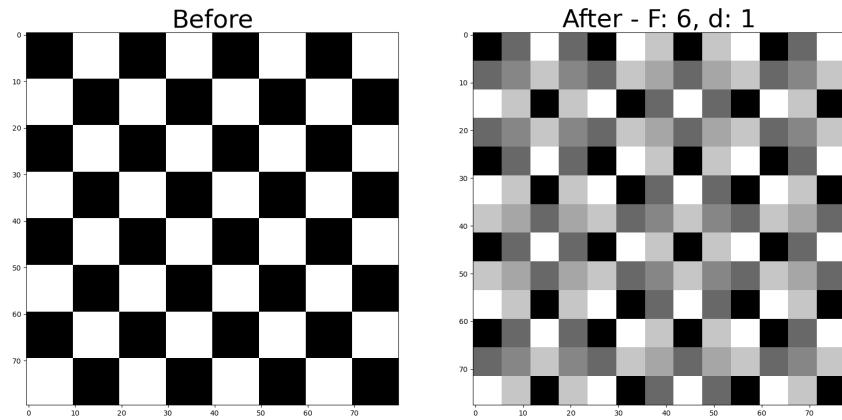


Figure 10: 80x80.bmp - $F = 6, d = 1$

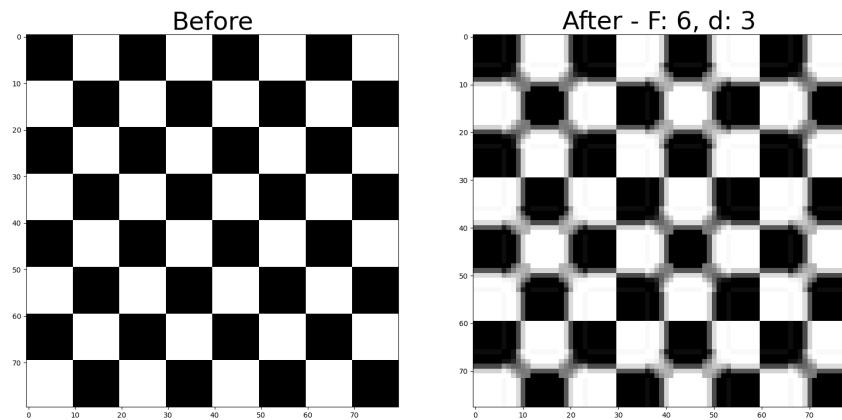


Figure 11: 80x80.bmp - $F = 6, d = 3$

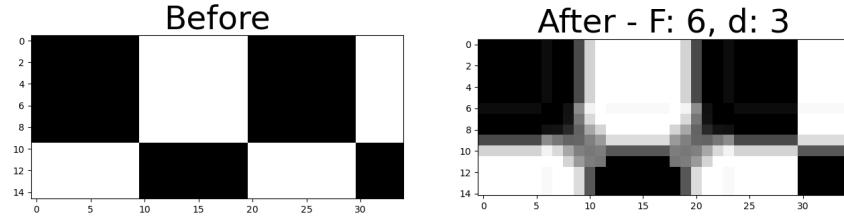


Figure 12: 80x80.bmp - $F = 6$, $d = 3$, zoommed

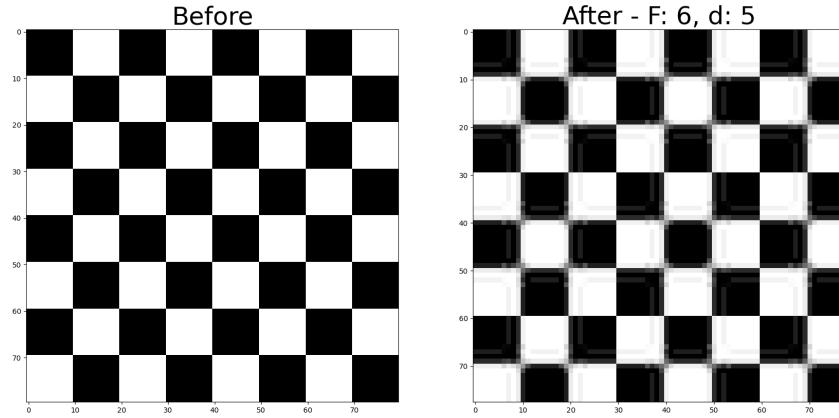


Figure 13: 80x80.bmp - $F = 6$, $d = 5$

Bridge Viene successivamente selezionata un’immagine più grande e complessa contenente diverse sfumature dal bianco al nero non presenti nei blocchi precedenti. Sono presenti anche più dettagli, avendo quindi la possibilità di osservare come la compressione si comporta su quest’ultimi.

In questo esperimento viene scelto un parametro F pari a 60, vista la più grande dimensione dell’immagine. Con valori di d molto piccoli come 1 o 5, l’immagine in output alla compressione risulta essere molto sgranata e di bassa qualità, come evidente in figura 15 e 16.

Se si aumenta il parametro d invece, fino a 11 come in figura 17 si osserva un

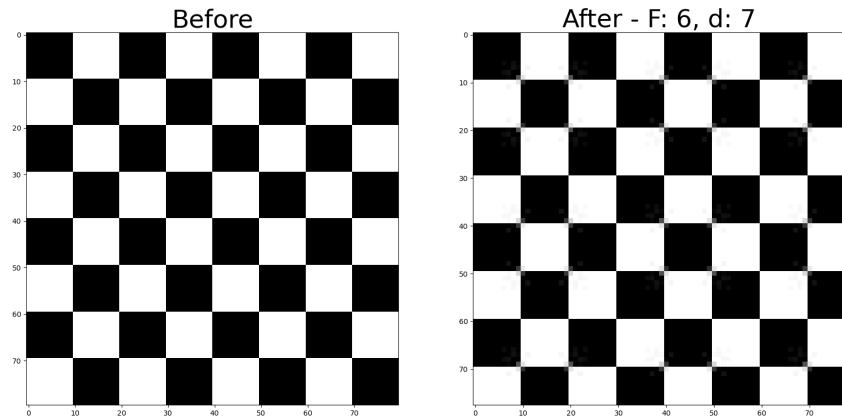


Figure 14: 80x80.bmp - F = 6, d = 7

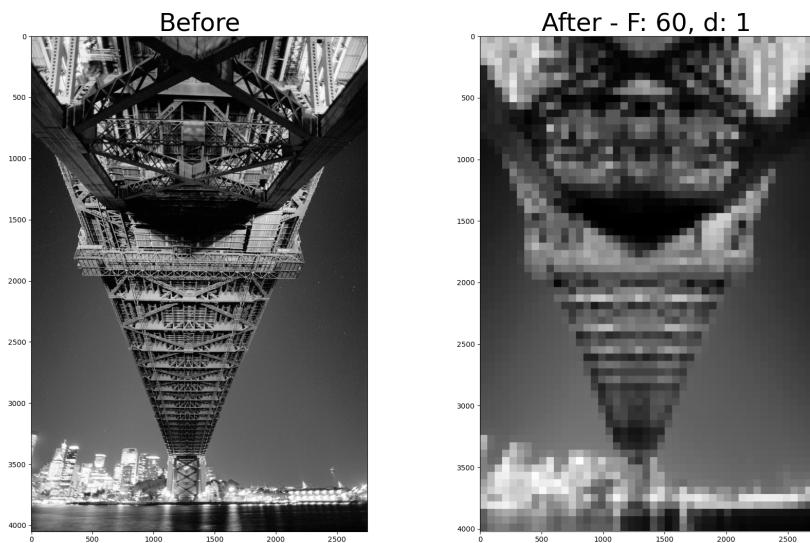


Figure 15: Bridge.bmp - F = 60, d = 1

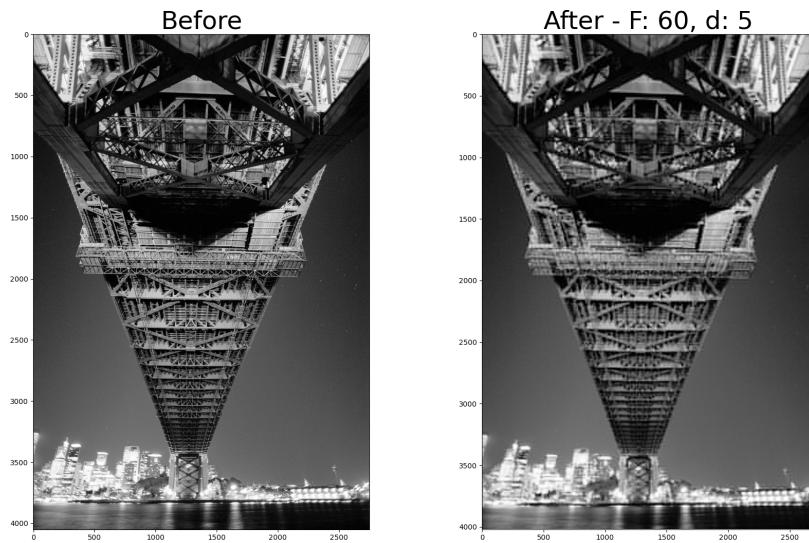


Figure 16: Bridge.bmp - F = 60, d = 5

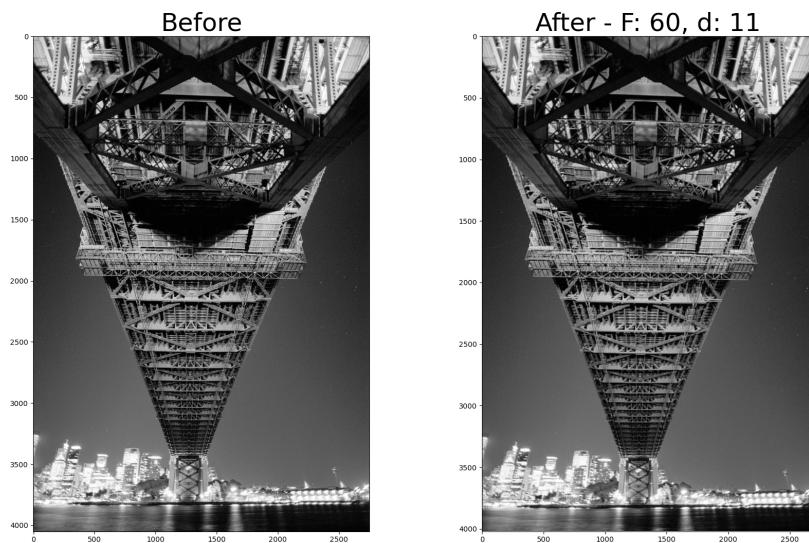


Figure 17: Bridge.bmp - F = 60, d = 11

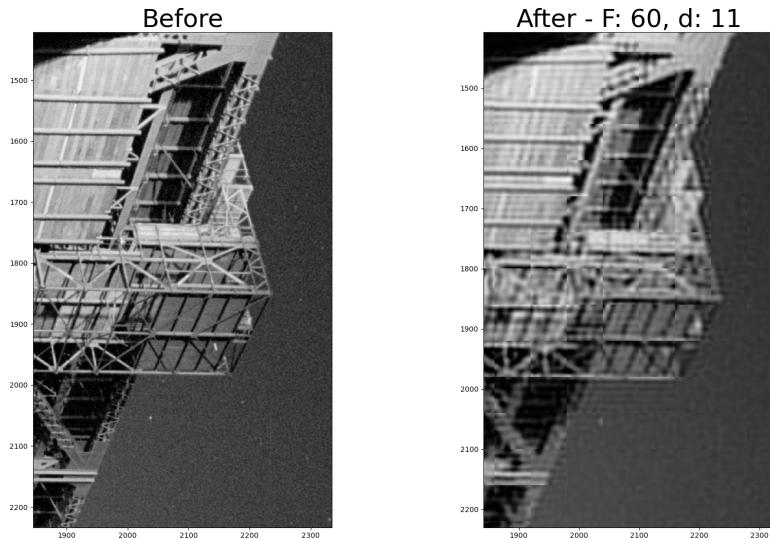


Figure 18: Bridge.bmp - $F = 60$, $d = 11$, zoommed

miglioramento della compressione ma, effettuando uno zoom, come evidente in figura 18, è possibile notare ancora gli artefatti soprattutto in presenza di pattern più complicati e salti di colore relativi all'architettura del ponte; al contrario, se si osserva il cielo sullo sfondo, risulta essere di buona qualità vista la presenza di un colore uniforme per tutto il blocco di dimensione (F, F) .

L'ultima prova viene effettuata per un parametro di d decisamente più alto, pari a 40, e viene presentato uno zoom in figura 19, dove la compressione, seppur minima, risulta essere efficace senza far perdere troppa qualità all'immagine.

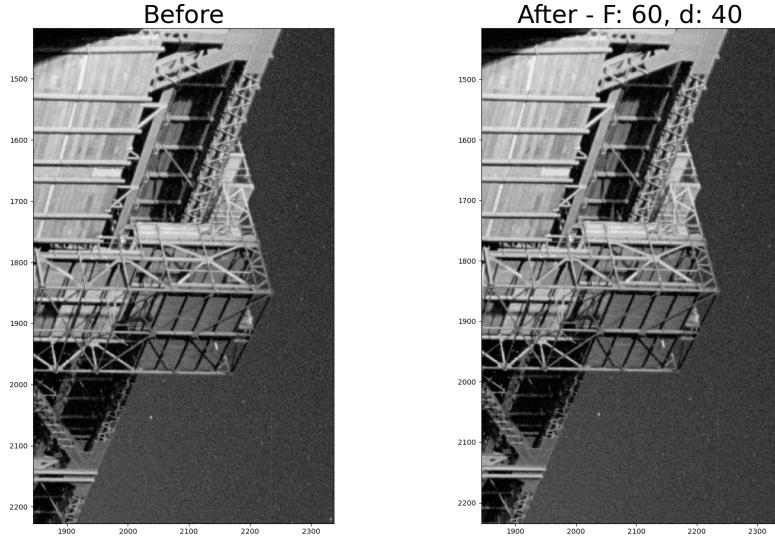


Figure 19: Bridge.bmp - $F = 60$, $d = 40$, zoommed

Webb's First Deep Field Viene infine scelta un'immagine molto più dettagliata e risoluta (4537, 4630) colma di piccoli dettagli utili per osservare il comportamento della compressione sui piccoli particolari dell'immagine. L'immagine presentata, nella versione bianca e nera, presenta una forte prevalenza del colore nero dove però i dettagli delle stelle e delle galassie sono illuminati di bianco; alcuni corpi celesti sono più grandi mentre altri sono più piccoli, nell'ordine di qualche pixel, cercando di mettere in difficoltà il processo di compressione soprattutto causato dall'eliminazione delle frequenze.

Viene scelto un parametro di F pari a 100 e un d iniziale pari a 10. Come osservabile in figura 20, e successivamente in una porzione ingrandita in figura 21, si perdono tanti dettagli più piccoli mentre le galassie e le stelle più grandi presentano un'evidente perdita di dettaglio. Sono altresì evidenti i blocchi (F, F) dove viene applicata la DCT2.

Nella figura 22 il parametro d viene aumentato a 20, migliorando quindi la qualità dell'immagine e riuscendo a conservare gran parte dei dettagli anche più piccoli ma perdendo molta qualità e definizione in quest'ultimi. I corpi più grandi invece risultano essere leggermente più definiti e meglio approssimati con una qualità generale apprezzabile in assenza di ingrandimento.

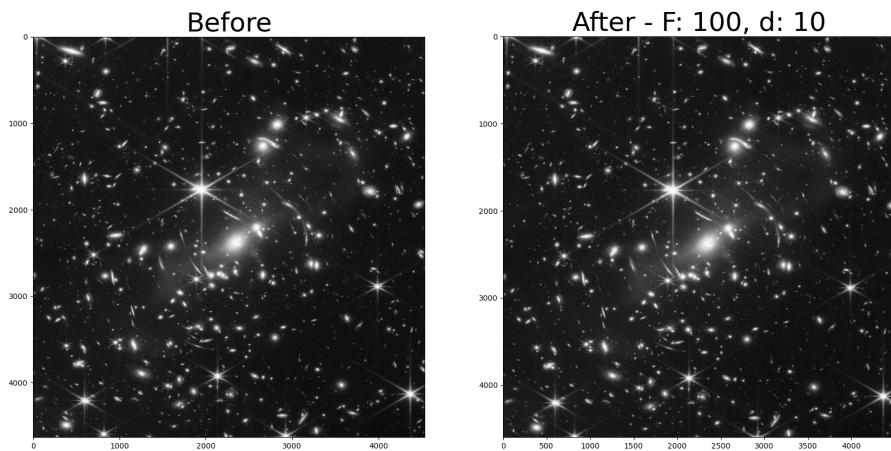


Figure 20: Webb telescope first image - $F = 100$, $d = 10$

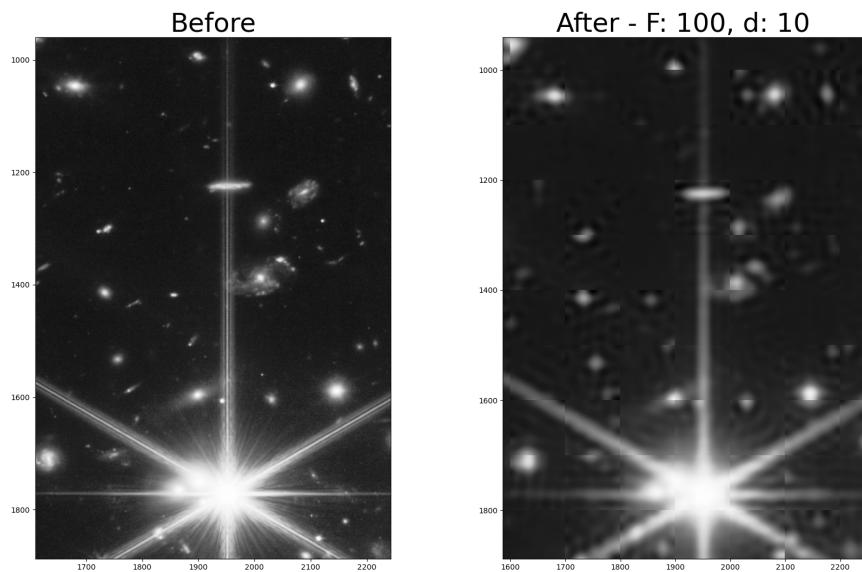


Figure 21: Webb telescope first image - $F = 100$, $d = 10$, zoommed

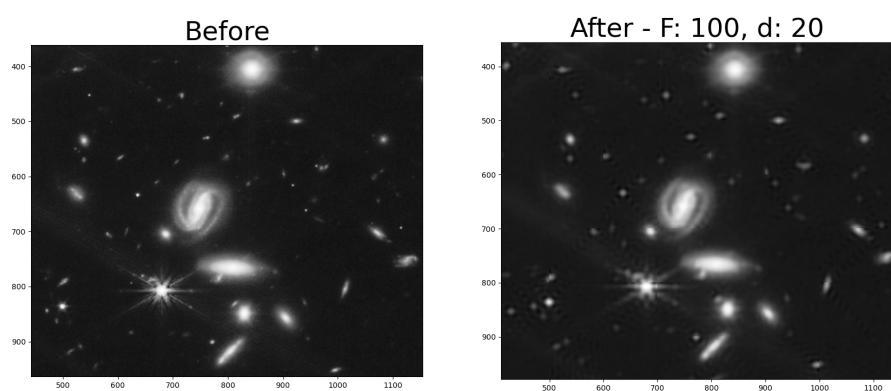


Figure 22: Webb telescope first image - $F = 100$, $d = 20$, zoommed

4 Conclusioni

Il progetto sviluppato implementa le operazioni di DCT e, conseguentemente, di DCT2 in un ambiente open source. Vengono dapprima mostrate le differenze tra un'implementazione manuale grezza e una fornita dalla libreria SciPy per calcoli scientifici su Python. I risultati dei test effettuati riprendono le ipotesi iniziali riguardo la complessità temporale delle due implementazioni dove, per quanto riguarda l'implementazione grezza, si ottiene un tempo proporzionale a $O(N^3)$, mentre la DCT2 implementata dalla libreria ha una complessità pari a $O(N^2 \log(N))$, con N riferito alla dimensione della matrice quadrata.

Nelle fasi successive si è implementata una semplice interfaccia grafica che permette all'utente di scegliere una qualsiasi immagine nel sistema. Viene data ulteriormente la possibilità di scegliere dei parametri per la compressione della stessa seguendo quanto visto con DCT2. Le immagini in questione vengono suddivise in blocchi sui quali viene effettuata la DCT2 e vengono successivamente tagliate delle frequenze per comprimere effettivamente la quantità di dati contenuta nell'immagine. Successivamente viene usata la funzione inversa della DCT2, la IDCT2, per poter decomprimere l'immagine e mostrare il risultato finale.

Per ultimo vengono effettuati una serie di esperimenti con diverse tipologie di immagini, mostrando dove la compressione è efficace e dove invece presenta delle criticità. Si sono potute osservare infatti delle problematiche per quanto riguarda la compressione in presenza di forti salti di colore o di elementi molto piccoli da dover conservare. Nei casi limite si è mostrato come si necessario mantenere una buona parte dei coefficienti per preservare una qualità dell'immagine accettabile.