Compressione di immagini tramite la DCT

Lorenzo Olearo `l.olearo@campus.unimib.it`

Alessandro Riva `a.riva86@campus.unimib.it`

A.A. 2022-2023

Lo scopo di questo progetto è di utilizzare l'implementazione della trasformata DCT2 in un ambiente open source e di studiare gli effetti di un algoritmo di compressione di tipo JPEG (senza utilizzare una matrice di quantizzazione) sulle immagini in toni di grigio. Comprende l'implementazione di un codice e la scrittura di una relazione da consegnare al docente.

Tutto il codice sorgente per la realizzazione del progetto comprendente consegne, test, esperimenti, grafici, dati e relazione è stato pubblicato al seguente repository GitHub: `https://github.com/LorenzoOlearo/cosine-image-compression`. Il repository rimarrà privato fino alla data dell'esame, per avere l'accesso, le chiediamo per ragioni tecniche di comunicarcelo via mail.

Indice dei contenuti

1 Confronto tra DCT diretta e veloce

Nella prima parte del progetto si richiede di confrontare le prestazioni della DCT2 come spiegata a lezione nella sua forma diretta, con quella di una libreria open source a scelta che si presuppone essere nella sua versione *fast*.

La trasformata DCT2 diretta è stata implementata in Python mentre per la sua versione *fast* è stata utilizzata la libreria Python `fftpack` di `scipy`.

La trasformata DCT che è stata implementata è la DCT di tipo II con riferimento alla seguente definizione:

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

Dove, α_k^N è il fattore di normalizzazione ed è definito come:

$$\begin{cases} \alpha_k^N = \sqrt{\frac{1}{N}} & per \quad k = 0 \\ \alpha_k^N = \sqrt{\frac{2}{N}} & per \quad k = 1, \cdots, N-1 \end{cases}$$

in modo che le basi dello spazio dei coseni siano ortonormali.

L'implementazione Python della DCT come definita sopra è la seguente:

```python
def dct(vector):
    N = len(vector)
    cosine_vector = np.zeros(N)

    for k in range(N):
        for n in range(N):
            cosine_vector[k] += vector[n] * np.cos((np.pi * k * (2 * n + 1)) / (2 * N))

        if k == 0:
            scaling_factor = np.sqrt(1/N)
        else:
            scaling_factor = np.sqrt(2/N)

        cosine_vector[k] *= scaling_factor

    return cosine_vector
```

Per calcolare la DCT2 su matrici si è calcolata la DCT, come definita sopra, prima sulle righe e poi sulle colonne della matrice.

```python
def dct2(matrix):
    cosine_matrix = np.zeros(matrix.shape)

    for i in range(matrix.shape[0]):
        cosine_matrix[i] = dct(matrix[i])
    for i in range(matrix.shape[1]):
        cosine_matrix[:, i] = dct(cosine_matrix[:, i])

    return cosine_matrix
```

Per confrontare i tempi di esecuzione delle due implementazioni sono state create una serie di matrici di interi di dimensioni crescenti, da 2x2 a 1024x1024, con valori casuali compresi tra 0 e 255. La scelta dell'upper-buond dei test dipende soltanto da i limiti computazionali dei calcolatori a nostra disposizione.

Ogni matrice è stata poi trasformata con entrambe le implementazioni della DCT2 tenendo traccia dei rispettivi tempi di esecuzione. I risultati sono stati riportati su un grafico che mette in relazione le dimensioni delle matrici con i tempi di esecuzione degli algoritmi in scala logaritmica.

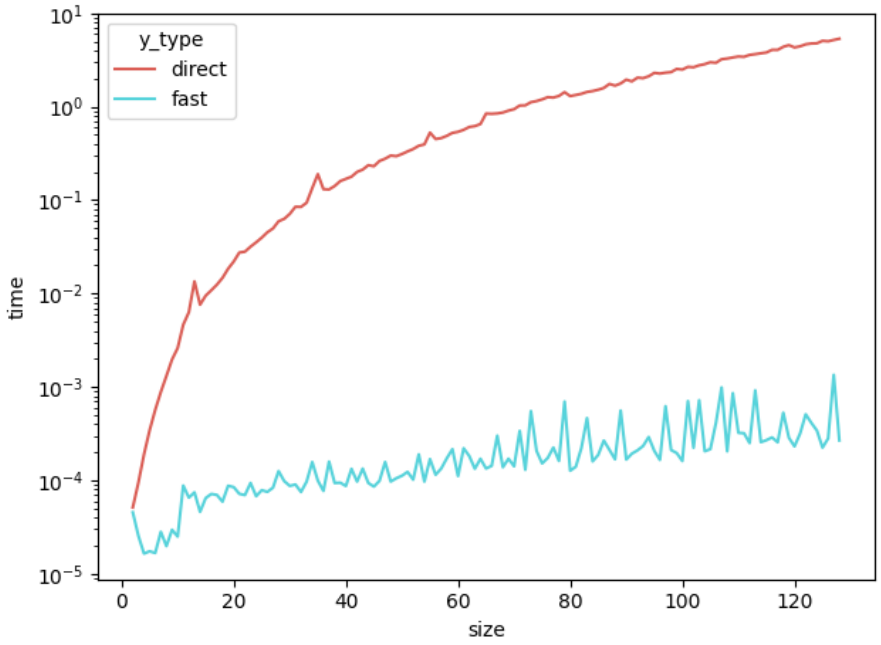

Figure 1: Confronto tra DCT diretta e veloce, sull'asse delle ascisse la dimensione delle matrici, sull'asse delle ordinate il tempo di esecuzione in scala logaritmica.

Come si può osservare dal grafico in Figura 1, il tempo computazione della DCT calcolata tramite il metodo diretto cresce in maniera decisamente più rapida rispetto a quello della DCT calcolata tramite la libreria `fft` di `scipy`.

La DCT2 calcolata tramite il metodo diretto presenta tempi proporzionali a N^3 rendendola quindi inutilizzabile per matrici di grandi dimensioni, la DCT calcolata tramite la libreria `fft` di `scipy` invece presenta tempi di esecuzione decisamente migliori.

Siccome la libreria `fft` di `scipy` utilizza la FFT, si è scelto di mettere a confronto i tempi di esecuzione della traformatata DCT2 su input di dimensioni pari a potenze di 2 crescenti.

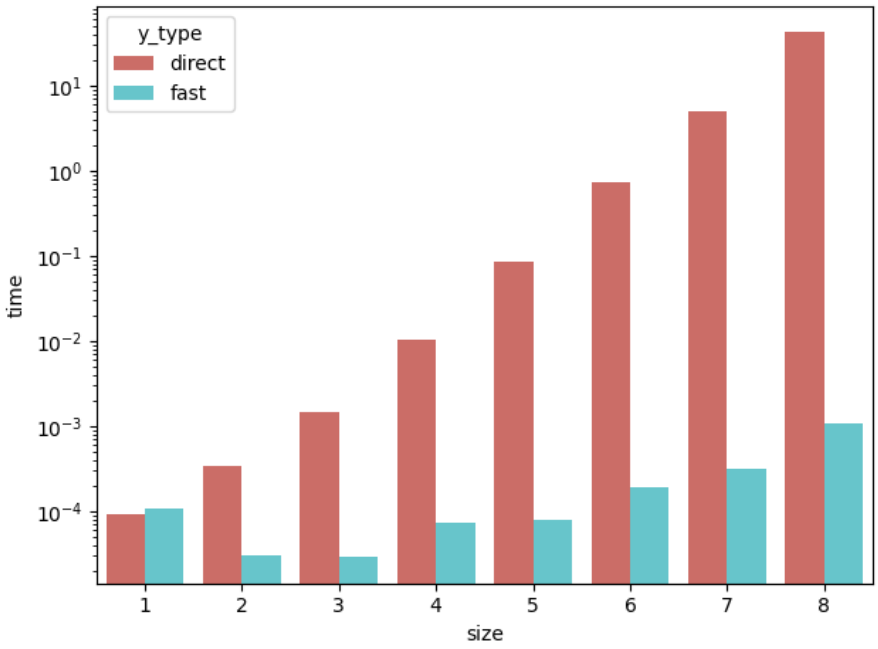

Figure 2: Confronto tra DCT diretta e veloce su input di dimensione pari a potenze di 2. Sull'asse delle ordinate il tempo in scala logaritmica, sull'asse delle ascisse le potenze di due delle dimensioni delle matrici.

Anche in questo caso, come si può osservare dal grafico in figura 2, la DCT implementata tramite il medoto diretto risulta essere considerevolmente più lenta rispetto a quella implementata tramite la libreria `fft` di `scipy`.

Tramite il software sviluppato è possibile ripetere i test qui presentati. Per effettuare il confronto tra la DCT diretta che è stata implementata e quella *fast* di `scipy` su matrici di dimensione incrementali comprese tra un valore di lower e upper buond, si può eseguire il programma con i seguenti argomenti:

```
> python main.py --nogui \
   --performance --incremental --lower <value> --upper <value>
```

Il software sviluppato permette inoltre di eseguire il confronto sulle prestazioni delle due trasformate su matrici di dimensioni pari a multipli di 2 tramite i seguenti argomenti:

```
> python main.py --nogui \
   --performance --order --lower <value> --upper <value>
```

Siccome è stato necessario implementare la trasformata coseno diretta, al fine di poterne verificare la correttezza è stato necessario introdurre dei metodi di test appositi. Si è scelto di esporre questi metodi tramite il parametro `test`.

E' possibile quindi eseguire solo il test della corretta implementazione della trasformata coseno con i seguenti parametri:

```
> python main.py --nogui --test
```

Per avere tutti i parametri con le loro descrizioni è possibile usare il parametro `help` come segue:

```
> python main.py --help
   Cosine Image Compression,
   Written By Lorenzo Olearo and Alessandro Riva, 2023

   options:
     -h, --help            Show this help message and exit
     --nogui               Run program with no GUI
     --test                Run test
     --performance         Run performance test
     --slow                Use direct cosine transform [...]
     --order               Test DCT on matrix on order of 2 [...]
     --incremental         Test DCT on matrix on incremental [...]
     --lower LOWER         Lower bound for performance tests
     --upper UPPER         Upper bound for performance tests
     --iterations          Iteration of the same tests
     --load                Load pre-computed data in order to [...]
```

2 Compressione di immagini

La seconda parte della consegna richiede la realizzazione di un'interfaccia grafica che permetta di caricare un'immagine in formato `BMP` in toni di grigio e di applicare un algoritmo di compressione JPEG senza però utilizzare una matrice di quantizzazione.

Anche per la realizzazione di questa seconda parte si è scelto di utilizzare Python, in particolare, l'interfaccia grafica è stata realizzata tramite la libreria `tkinter`.

Il software implementato permette all'utente di caricare un'immagine `BMP` in scala di grigi dal proprio filesystem sfruttando il *file picker* di `tkinter`.

Una volta caricata l'immagine all'interno dell'applicazione, l'utente è in grado di specificare:

- un intero F corrispondente alla dimensione in pixel dei *macro-blocchi* su cui effettuare la DCT2.
- un intero d compreso tra 0 e $2F - 2$ rappresentante il valore di taglio delle frequenza

2.1 Backend

Una volta specificati questi parametri tramite l'interfaccia grafica, la matrice corrispondente all'immagine `BMP` caricata dall'utente viene passata all'algoritmo di compressione.

L'algoritmo di compressione viene invocato dal modulo controller tramite il metodo `bitmap_to_jpeg`

```
def bitmap_to_jpeg(bitmap, macro_size, freq_cut, fast = True):
    macro_blocks = extract_macro_blocks(bitmap, macro_size)
    compressed_macro_blocks = np.zeros(macro_blocks.shape)

    for i in range(macro_blocks.shape[0]):
        for j in range(macro_blocks.shape[1]):
            compressed_macro_blocks[i, j] = macro_block_compression(macro_blocks[i, j], freq_cut, fast)

    return recompose_macro_blocks(compressed_macro_blocks)
```

Per ogni macroblocco si applicano ora una serie di funzioni nel seguente ordine:

1. `dct2`: applica la DCT2 al macroblocco, i pixel in eccesso vengono eliminati
2. `frequency_cut`: applica il taglio delle frequenze
3. `idct2`: applica l'antitrafrormata al macroblocco
4. `clamp_macro_block`: arrotonda i valori all'intero più vicino, mettendo a zero quelli negativi e a 255 quelli superiori di 255

Infine, tutti i macroblocchi vengono ricomposti nell'ordine corretto per formare l'immagine compressa dal metodo `recompose_macro_blocks`.

L'intero codice sorgete è pubblicato al repository GitHub:
`https://github.com/LorenzoOlearo/cosine-image-compression`.

2.2 Frontend

L'interfaccia grafica presenta una barra di controllo mostrata in figura 3 dove sono presenti due pulsanti, uno per selezionare l'immagine da utilizzare e uno per avviare il processo di compressione con DCT. Dopo il caricamento dell'immagine il pulsante per avviare la compressione viene abilitato insieme ai campi per l'input dei parametri F e d.

Una volta trasfrormata l'immagine, l'applicazione mostra fianco a fianco l'immagine originale e quella su cui è stato eseguito l'algoritmo di compressione delle frequenze. L'applicazione permette inoltre `zoom` e `pan` delle due immagini simultaneamente.

Figure 3: Barra di controllo dell'applicazione.

L'inserimento del parametro F, è limitato ai valori tra 1 e la dimensione dell'immagine, e lo slider per il parametro d, è limitato tra 0 e $2F - 2$. Alla pressione del pulsante

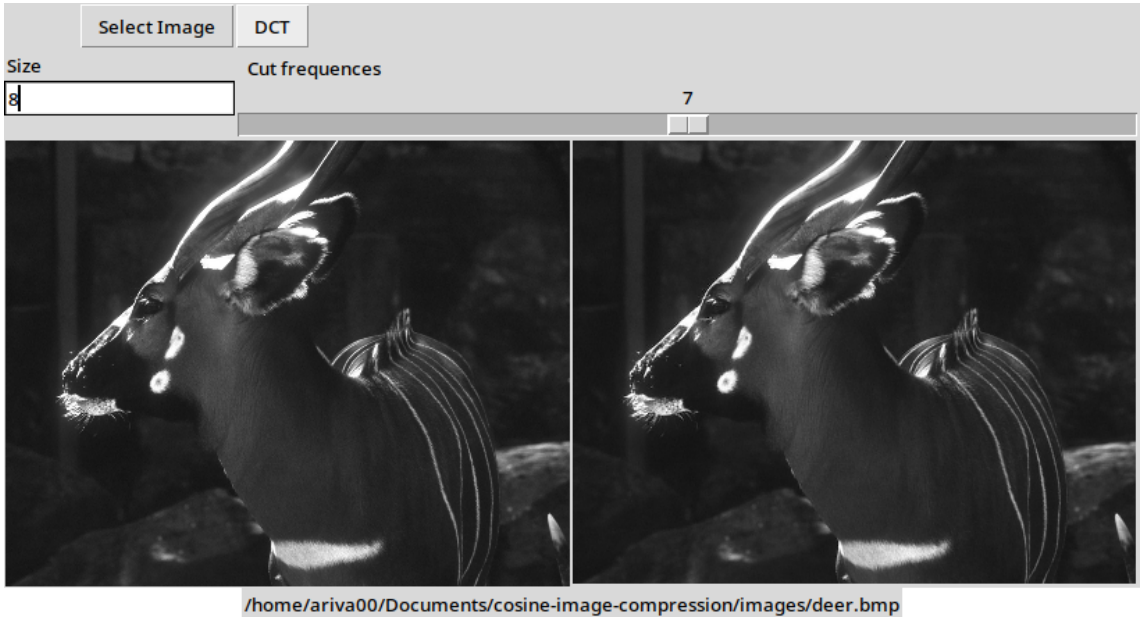

Figure 4: L'applicazione mostra l'immagine originale e quella trasformata.

DCT viene eseguita la trasformata e viene mostrato il risultato della compressione affiancato all'immagine originale come mostrato in figura 4.

Per meglio visualizzare l'immagine è possibile utilizzare la rotella del mouse per scalarla e il tasto sinisto per trascinarla. Così facendo è possibile apprezzare la differenza tra le due immagini come in figura 5.

Figure 5: Zoom su un dettaglio dell'immagine.

Lo zoom e il pan sono implementati tramite il modulo `PIL` di Python che viene utilizzato per tagliare e ridimensionare le immagini prima di mostrarle, le trasformazioni utilizzano un criterio nearest neighbour, in modo da non introdurre artefatti durante lo zoom.

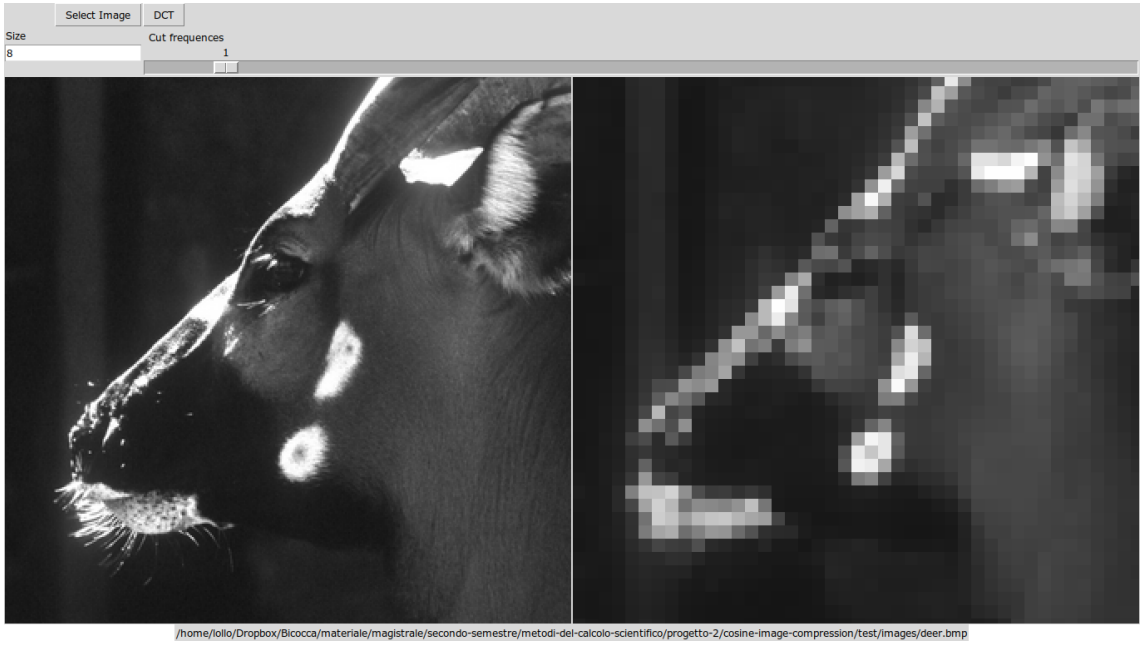

Figure 6: DCT con valore di taglio delle frequenze ad 1, in questo modo solo il valore medio non viene eliminato, dal confronto infatti, si può notare come in ogni macroblocco sia la media dell'intensità dei pixel che lo compongono

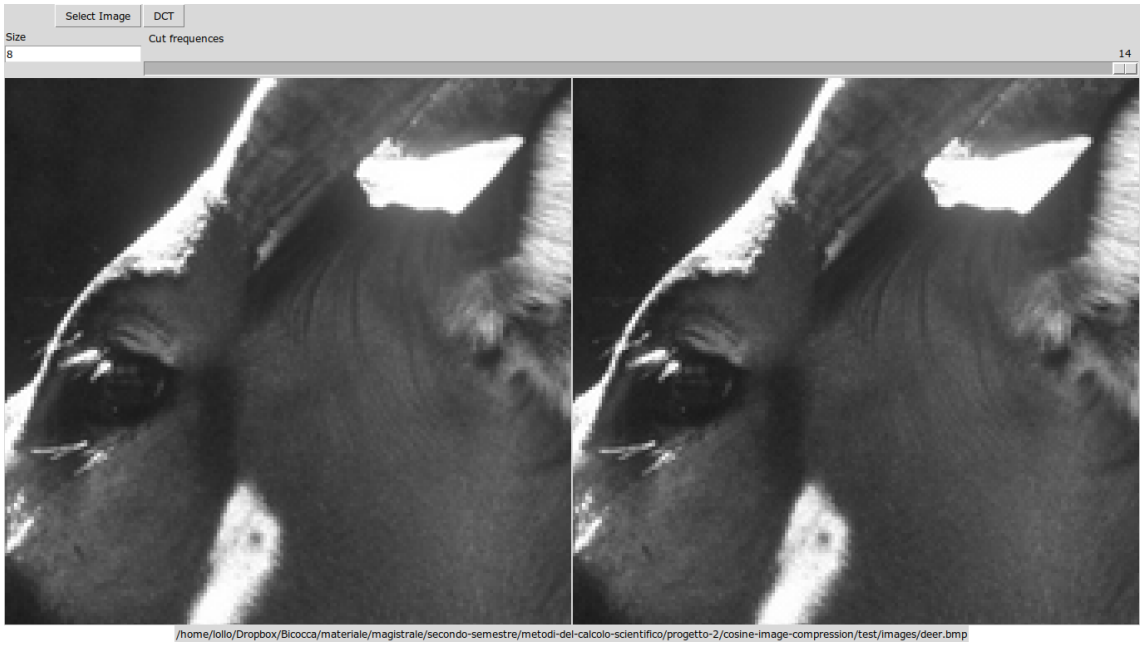

Figure 7: DCT con il valore massimo di taglio delle frequenze, si può notare come in questo caso non venga persa informazione in frequenza nell'immagine