Metodi del Calcolo Scientifico
Relazione Progetto 2

A.A. 2019-2020, Appello 04/06/2020

Componenti gruppo:

- Christian Bernasconi 816423
- Gabriele Ferrario 817518
- Riccardo Pozzi 807857
- Marco Ripamonti 806785

Link al repository: `https://gitlab.com/m.ripamonti/mcs_projects`

Indice

1 Introduzione

La relazione di questo progetto si compone di tre parti principali. Per cominciare, Python viene presentato al capitolo 2 con una descrizione delle librerie utilizzate per lo sviluppo di quanto richiesto. A seguire, al capitolo 3 si trova la spiegazione di quanto fatto per la prima parte del progetto, ossia quella relativa all'implementazione della DCT2. Infine, al capitolo 4 è presentata la seconda parte del progetto, corredata da alcuni esperimenti a dal commento dei loro risultati.

2 Python

Il linguaggio di programmazione che è stato scelto per sviluppare entrambe le parti del progetto è Python.
Python è un popolare linguaggio di programmazione ad alto livello, interpretato e orientato agli oggetti [1] nato negli anni 90 [2], completamente gratuito e distribuito con licenze open source GPL-compatibili [2]. Una delle sue peculiarità è quella di essere semplice e di facile apprendimento, ma allo stesso tempo è in grado di essere impiegato in svariati campi, grazie all'enorme quantità di moduli e librerie che è possibile integrare all'interno dei propri script. Oltre alle librerie di computazione numerica sono disponibili anche librerie per lo sviluppo di GUI, motivo per cui è stato scelto anche per la seconda parte del progetto.

2.1 Librerie utilizzate

Di seguito vengono elencate le librerie utilizzate per le due parti del progetto.

Prima parte - Implementazione DCT2

- NumPy: libreria per la computazione numerica ed analisi scientifica. Tra le sue principali caratteristiche possiamo notare la gestione di array N-dimensionali, funzioni per operazioni di algebra lineare e per la generazione di numeri pseudo-casuali [3].

- SciPy library: libreria che fornisce diverse efficienti routines per operazioni di numerica, interpolazione, ottimizzazione, algebra lineare e statistica [4]. Da questa libreria sono state utilizzate le funzioni per computare la DCT2 con cui poi avverrà il confronto con la funzione da noi implementata.

- Pandas[1]: libreria per manipolazione e analisi dei dati. In particolare, offre i cosiddetti dataframe, ovvero strutture dati tabulari (righe e colonne), mutabili in dimensione ed eterogenee.

Tutte e tre le librerie sono parte di SciPy: un "ecosistema basato su python rivolto alla matematica, alle scienze e all'ingegneria" [5]. L'utilizzo è gratuito e sono distribuite sotto licenza opensource (BSD 3-Clause) [6, 7].

Seconda parte - Implementazione algoritmo di compressione

- NumPy: libreria utilizzata principalmente per la gestione delle matrici.

[1] https://pandas.pydata.org/

- SciPy library: libreria utilizzata per effettuare le computazioni di DCT2 ed IDCT2 sulle matrici rappresentanti le immagini su cui avverrà la compressione.
- PySide2: libreria utilizzata per lo sviluppo dell'interfaccia utente [8]. Tale libreria, descritta dettagliatamente nella sezione 4.3.1, permette l'integrazione della famosa libreria Qt[2] di C++.
- OpenCv[3]: libreria utilizzata per leggere le immagini da disco.
- Humanize[4]: libreria di importanza secondaria utilizzata per mostrare lo spazio occupato dall'immagine.

[2] `https://www.qt.io/`
[3] `https://opencv.org/`
[4] `https://github.com/jmoiron/humanize`

3 Implementazione DCT2 (Parte 1)

3.1 Analisi libreria DCT2 SciPy

SciPy è una libreria che offre funzioni per operazioni di analisi numerica offrendo efficienti routines per algebra lineare, ottimizzazione ed analisi statistica. In particolare è stato usato il pacchetto *SciPy.fft* che raccoglie le funzioni *FFT* (Fast Fourier Transform) che permettono di computare le *DFT* (Discrete Fourier Transform) tra cui *DCT* e *DST* (Discrete Sin and Cosine Transform) migliorandone notevolmente la complessità computazionale. Di seguito vengono riportate alcune delle funzioni offerte da questo pacchetto[5]:

- fft: computa la *Discrete Fourier Transform* monodimensionale
- ifft: computa la *Inverse Discrete Fourier Transform* monodimensionale
- fft2: computa la *Discrete Fourier Transform* bidimensionale
- ifft2: computa la *Inverse Discrete Fourier Transform* bidimensionale
- **dct**: computa la *Discrete Cosine Transform* monodimensionale
- **idct**: computa la *Inverse Discrete Cosine Transform* monodimensionale

Per questo progetto è stata necessaria in particolare la funzione *dct*. Occorre sottolineare che non viene offerta alcuna funzione per computare direttamente la *dct2* (bidimensionale), ma é sufficiente applicare la funzione dct una volta sulle colonne e una volta sulle righe agendo sul parametro *axis*.
SciPy implementa quattro degli otto tipi di dct[6]: è possibile specificarne uno di essi attraverso il parametro *type*, altrimenti di default viene computata la dct di tipo 2.
Segue la formula per il calcolo di ciascun coefficiente della dct di tipo 2:

$$y_k = 2 \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi k \left(2n+1\right)}{2N}\right)$$

È stato infine necessario specificare il parametro *norm* con valore *ortho* in modo da normalizzare ciascun y_k per un fattore f:

[5] `https://docs.scipy.org/doc/scipy/reference/fft.html`

[6] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.dct.html#scipy.fft.dct`

$$f = \begin{cases} \sqrt{\frac{1}{4N}} & \text{con } k = 0 \\ \\ \sqrt{\frac{1}{2N}} & \text{altrimenti} \end{cases}$$

Questo passaggio è stato necessario a rendere la matrice dei coefficienti ortonormale riconducendola alla forma vista a lezione.
Viene riportata di seguito l'istruzione per effettuare la dct2 correttamente normalizzata tramite SciPy, dove M è la matrice di partenza e A quella ottenuta dalla dct2.

```
A = dct( dct( M, axis=1, norm='ortho' ), axis=0, norm='ortho' )
```

Come ultima nota è importante sottolineare che la dct di SciPy fa uso della *Fast Fourier Transform*; ciò la rende notevolmente più efficiente rispetto alla nostra implementazione, come si vedrà nella sezione 3.3.

3.2 Struttura del codice

Il codice sorgente di questa parte del progetto è disponibile all'indirizzo: `https://gitlab.com/m.ripamonti/mcs_projects/-/tree/master/proj2/python/part1` ed è suddiviso nei seguenti quattro file python: *main.py*, *libraryMCS.py*, *utils.py* e *verifyDCT.py*. Per eseguirlo è necessario aver installato python 3 e i requirements definiti nel file *requirements.txt* reperibile al seguente link: `https://gitlab.com/m.ripamonti/mcs_projects/-/blob/master/proj2/python/requirements.txt`.

3.2.1 libraryMCS.py

Questo file è composto dalle seguenti funzioni:

- **getAlpha(k, N)**: prende in input un intero positivo k e un intero positivo N (rappresentante la dimensione della matrice), in base al valore di k computa e ritorna il valore α_k seguendo la seguente regola:

$$\alpha_k = \begin{cases} N & \text{if } k = 0 \\ \\ \frac{N}{2} & \text{if } k \neq 0 \end{cases}$$

- **dct1(v)**: prende in input un vettore v e computa la dct1 (monodimensionale); ritorna un vettore c le cui componenti sono calcolate tramite la seguente formula:

$$c_k = \frac{\sum_{i=0}^{N-1} v_i \cdot \cos(\pi k \frac{2i-1}{2N})}{\sqrt{\alpha_k}}$$

- **dct2V1(Mat)**: prende in input una matrice Mat di dimensione N x M; ritorna una matrice C ottenuta tramite l'applicazione della funzione dct1 una volta sulle righe e una volta sulle colonne.

- **dct2V2(Mat)**: prende in input una matrice Mat di dimensione N x M; ritorna una matrice C le cui componenti sono calcolate tramite la seguente formula:

$$c_{k,l} = \frac{\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} Mat_{i,j} \cdot \cos(\pi k \frac{2i-1}{2N}) \cos(\pi l \frac{2j-1}{2M})}{\sqrt{\alpha_k \alpha_l}}$$

3.2.2 verifyDCT.py

In questo file è riportata la matrice D di test indicata nella traccia del progetto per testare lo scaling e il funzionamento di dct2 e dct1. Le funzioni sviluppate per effettuare le verifiche sulla matrice sono le seguenti:

- **verifyBlockDCT2(flag, norm)**: prende in input un flag (default 0) e una variabile norm (default ortho). Se il valore di flag è 0 restituisce sullo standard output la matrice D a cui è stata applicata la DCT2 tramite l'applicazione della dct di SciPy una volta sulle righe e una volta sulle colonne; altrimenti restituisce la matrice D a cui è stata applicata la dct2V1.

- **verifyRowDCT1(flag, norm)**: prende in input un flag (default 0) e una variabile norm (default ortho). Se il valore di flag è 0 restituisce sullo standard output la prima riga della matrice D a cui è stata applicata la dct di SciPy; altrimenti restituisce la prima riga della matrice D a cui è stata applicata la dct1 implementata.

Queste funzioni permettono quindi all'utente di effettuare una rapida verifica visuale dei risultati ottenuti dalla trasformazione.

3.2.3 utils.py

In questo file è definita la seguente funzione:

- **generateRandomMatrix(N, M)**: prende in input due interi positivi N e M; restituisce una matrice di dimensione N x M composta da interi random.

3.2.4 main.py

In questo file è stato scritto uno script per testare tutte le funzioni appena descritte. Tramite un menù interattivo permette le operazioni seguenti:

- verifyBlockDCT2
- verifyRowDCT1
- check dct2 implementation
- compare dcts

3.3 Confronto tra DCT2 implementata e DCT2 di Scipy

Innanzitutto è stato effettuata una verifica sui valori ottenuti dalla dct di nostra implementazione rispetto al vettore fornito nella traccia del progetto.
Dopodiché, prima di confrontare le performance della dct2 di *libraryMCS* rispetto a quella fornita da *SciPy.fft*, è stato eseguito un controllo sulla consistenza dei valori ottenuti dalle due librerie a partire da matrici di test, in modo da poter confermare che la dct computata con *SciPy.fft* operasse allo stesso modo rispetto a quanto visto a lezione.

Il confronto sulle performance prende in considerazione il tempo impiegato da *libraryMCS.dct2V1*, *libraryMCS.dct2V2* e dalla doppia applicazione di *SciPy.fft.dct* per calcolare la trasformazione su 20 matrici quadrate generate casualmente, le cui dimensioni variano da 10x10 a 200x200 con un incremento di 10x10.

In figura 1 è possibile osservare il comportamento dalle tre funzioni al crescere delle dimensioni dell'input. Le ascisse rappresentano le dimensioni delle matrici, mentre le ordinate, in scala logaritmica, corrispondono al tempo impiegato. Le curve nel grafico confermano le previsioni riguardo la complessità computazionale delle diverse implementazioni.
Il metodo *dct2V2*, la cui complessità è pari a $O(N^4)$, risulta il meno performante.

Questo metodo è stato volutamente limitato alle prime 10 matrici a causa dei notevoli tempi richiesti.
Il metodo *dct2V1*, di complessità $O(N^3)$, mostra un comportamento nettamente migliore del precedente.
Infine *SciPy.fft.dct*, la cui complessità è ridotta a $O(N^2 log(N))$ per merito della Fast Fourier Transform, ottiene il risultato migliore in assoluto.
I tempi effettivi non in scala logaritmica sono consultabili nella tabella 1.

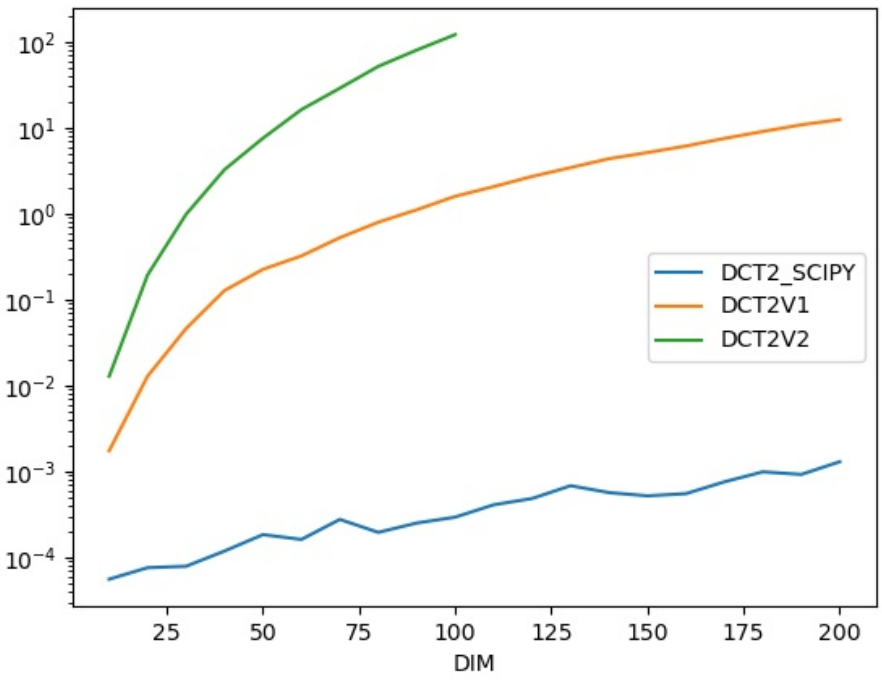

Figura 1: Tempo impiegato dalle diverse implementazioni di dct2 (sulle y in scala logaritmica) all'aumentare delle dimensioni dell'input (sulle x)

SciPy	**dct2V1**	**dct2V2**	**Dim.**
5.569999999999187e-05	0.0017336000000000018	0.012775300000000045	10
7.580000000007026e-05	0.012813999999999992	0.19321499999999991	20
7.869999999998711e-05	0.04581029999999997	0.9775666000000001	30
0.00011830000000001561	0.12703189999999998	3.2447316	40
0.0001838000000002893	0.22438480000000016	7.5267007	50
0.0001614999999990374	0.3212653000000003	16.198178	60
0.0002761000000006675	0.5238236000000001	28.738181400000002	70
0.0001952000000028404	0.7929004000000006	51.56870620000001	80
0.0002507999999892263	1.1066644999999937	79.8446131	90
0.000293400000003885	1.588044999999994	120.83413380000002	100
0.00040869999997994455	2.0552627000000143	-	110
0.00048359999999547654	2.703120599999977	-	120
0.0006816000000071654	3.4227569000000244	-	130
0.000566400000025169	4.343924700000002	-	140
0.0005194999999957872	5.121972200000016	-	150
0.0005508999999506159	6.102163299999972	-	160
0.0007562000000120861	7.4770121000000245	-	170
0.0009878000000185239	9.013939600000015	-	180
0.0009223000000133652	10.770630900000015	-	190
0.0012959999999679894	12.3887158	-	200

Tabella 1: Tempi di computazione delle diverse implementazioni di dct2

4 Implementazione algoritmo di compressione (Parte 2)

4.1 Struttura del codice

Il codice sorgente di questa parte del progetto è disponibile all'indirizzo `https://gitlab.com/m.ripamonti/mcs_projects/-/tree/master/proj2/python/part2/src`.
Per essere eseguito è necessaria l'installazione di python 3 e dei requirements definiti nel file *requirements.txt* reperibile al seguente link: `https://gitlab.com/m.ripamonti/mcs_projects/-/blob/master/proj2/python/requirements.txt`. La struttura di organizzazione del codice è la seguente:

- **assets**
- **components**
  - content.py
  - main_window.py
  - toolbar.py
- **shared**
  - imageCompressor.py
  - worker.py
- main.py

La cartella *assets* ha lo scopo di raccogliere file (es: logo, font) che verranno utilizzati dalle componenti grafiche. *components* contiene i file dell'implementazione dell'interfaccia grafica. *shared*, invece, contiene la logica dell'algoritmo di compressione e una classe per costruire un oggetto thread utilizzata per migliorare l'interfaccia grafica. In particolare, quest'ultima cartella ha lo scopo di raccogliere tutti i file le cui funzioni possono essere riutilizzate all'interno del progetto anche in più componenti. Infine, allo stesso livello delle cartelle si trova il file *main.py* che rende possibile l'avvio dell'applicazione.
I dettagli di quanto implementato saranno discussi nelle apposite sezioni 4.2 e 4.3.

4.2 Algoritmo di compressione

Di seguito viene descritto l'algoritmo di compressione implementato mediante la funzione *compressImage* contenuta in *imageCompressor.py*[7].

4.2.1 Definizione di *compressImage*

```
compressImage(mat, F, d, [progress_callback])
```

Input:

- mat: matrice corrispondente all'immagine selezionata in input. L'immagine, essendo stata letta in scala di grigi, avrà la corrispondenza di ogni pixel con un elemento della matrice a cui è associato un intero compreso tra 0 e 255.
- F: parametro che individua la dimensione di un singolo blocco su cui avverranno le operazioni di compressione. Ad esempio con F pari a 8, la matrice rappresentante l'immagine in input verrà divisa in blocchi 8x8.
  N.B: F deve essere un intero positivo il cui valore massimo corrisponde al minimo tra il numero di righe e il numero di colonne della matrice.
- d: il *Frequency Threshold*; i coefficienti ckl di ogni blocco tali che $k + l \geq d$ vengono azzerati.
  N.B: d deve essere un intero compreso tra 0 e (2F - 2).
- progress_callback: parametro opzionale che consente di passare alla funzione il riferimento al *signal progress* del worker (thread) che la sta eseguendo e permettere quindi un *emit* dell'avanzamento. Questo parametro è stato utilizzato per per mostrare una progressbar.

Output:

- compressedImage: in output viene fornita la matrice dell'immagine compressa.

4.2.2 Fasi dell'algoritmo di compressione

L'algoritmo si compone principalmente di due fasi:

[7] `https://gitlab.com/m.ripamonti/mcs_projects/-/blob/master/proj2/python/part2/src/shared/imageCompressor.py`

1. Divisione della matrice in blocchi: la matrice rappresentante l'immagine viene divisa in blocchi quadrati la cui grandezza varia a seconda del parametro F scelto dall'utente. È importante notare che nel caso le dimensioni della matrice non siano multipli di F, i pixel in eccesso vengono semplicemente eliminati. Ne risulta quindi una riduzione delle dimensioni dell'immagine.

2. Operazioni su un singolo blocco: per ogni blocco vengono eseguite le seguenti operazioni:

   (a) Viene computata la *dct2* (bidimensionale) producendo un nuovo blocco *newBlock* di coefficienti *ckl*.

   (b) Tutti i *ckl* con $k + l \geq d$, con d scelto dall'utente vengono azzerati.

   (c) I dati restanti del blocco vengono ora riportati alla forma originale tramite la funzione inversa (idct2 bidimensionale).

   (d) Dopodiché i coefficienti sono arrotondati all'intero più vicino e i valori esterni all'intervallo [0,255] sono a loro volta sostituiti con l'estremo più vicino.

4.3 Interfaccia

In questa sezione viene riportata una spiegazione dettagliata dell'interfaccia con la libreria utilizzata per la sua realizzazione, le sue componenti e il suo funzionamento.

4.3.1 PySide2

Per la realizzazione dell'interfaccia è stata utilizzata la libreria *PySide2*, prodotto del progetto *Qt for Python* che ha l'obiettivo di fornire il completo supporto e porting del modulo *Qt 5* in *C++*.
La libreria offre un framework per lo sviluppo di interfacce utente con design di alto livello per la realizzazione di applicazioni di alta qualità.
La documentazione della libreria è ben aggiornata e dettagliata con casi d'uso ed esempi. La facilità è aumentata dal fatto di avere la stessa sintassi e semantica di *Qt* per *C++* quindi, per chi proviene da questa piattaforma, l'uso è praticamente immediato.

4.3.2 Componenti Interfaccia

L'interfaccia (Figura 2) comprende tre componenti principali. Il componente principale *MainWindow* che a sua volta ne contiene altri due: *Toolbar* e *Content*.

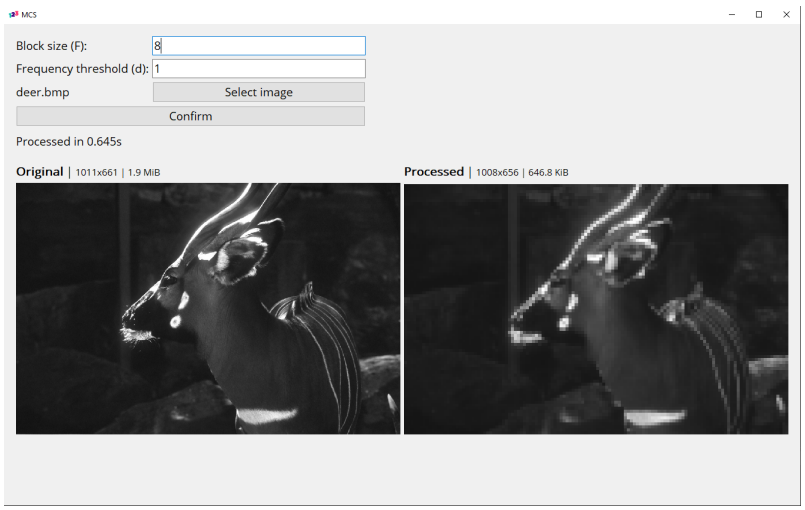

Figura 2: Interfaccia utente

La **Toolbar** è il 'core' dell'interfaccia; contiene un form per inserire i campi che fanno riferimento alle variabili F (Block Size) e d (Fequency Threshold). Inoltre è presente un bottone per la selezione di un'immagine dal filesystem in formato *bmp* con la label corrispondente al nome dell'immagine selezionata.
È importante sottolineare che grazie all'utilizzo della libreria *OpenCv* è possibile leggere un'immagine in scala di grigi in cui ogni pixel è costituito da un byte che può assumere valori compresi tra 0 e 255. Se non specificato diversamente, i valori dei pixels vengono convertiti alla profondità di 8 bit[8]. Infine è presente un bottone *Confirm* per eseguire l'algoritmo di compressione dell'immagine in base ai parametri scelti: premendolo, se i parametri sono validi, una progressbar mostra l'avanzamento dell'algoritmo e al suo termine viene riportato il tempo impiegato per il completamento.
Il processo di compressione dell'immagine e aggiornamento della barra di caricamento avviene in un thread di lavoro differente dal thread principale per evitare il *freeze* dell'interfaccia utente.
In figura 3 e in figura 4 vengono mostrate rispettivamente la toolbar durante l'esecuzione dell'algoritmo di compressione e una volta terminata la computazione.

[8] `https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html`

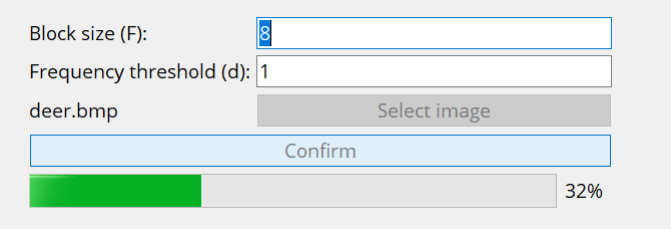

Figura 3: Toolbar durante l'esecuzione dell'algoritmo di compressione

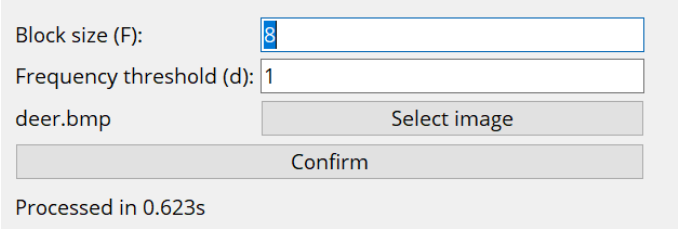

Figura 4: Toolbar che mostra il tempo impiegato dall'algoritmo di compressione

Il componente **Content** si occupa di visualizzare l'immagine originale selezionata dal filesystem e la stessa una volta compressa. Vengono inoltre riportate alcune misure tra cui le dimensioni dell'immagine e lo spazio occupato su disco (figura 5).

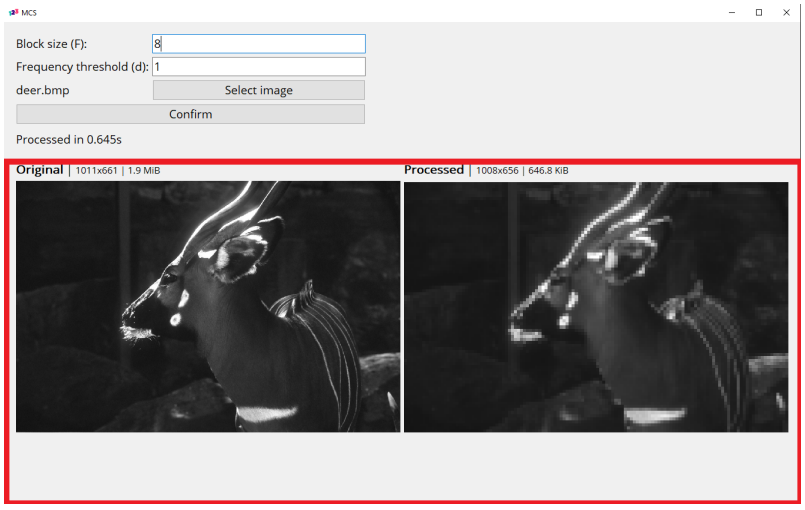

Figura 5: Content

4.4 Applicazione dell'algoritmo di compressione ad alcune immagini

Una volta portato a termine lo sviluppo di questo progetto, sono stati effettuati degli esperimenti su alcune immagini per poter osservare a livello visivo i risultati ottenuti operando sulle matrici. In seguito sono mostrate due figure di esempio che mostrano il confronto tra immagine originale e processata.

Figura 6: Esempio 1 - Compressione con F=100, d=5

Figura 7: Esempio 2 - Compressione con F=15, d=10

Nel caso della figura 6 è stata impostata su un'immagine 2000x3008 la dimensione di 100 per i macro-blocchi (F) e il valore di 5 (d) come soglia di taglio delle frequenze. Questa configurazione porta al raggruppamento della matrice originale in 30 righe e 20 colonne[9] di blocchi 100x100. La soglia di taglio bassa porta all'eliminazione di gran parte delle frequenze generando una forte compressione che abbassa di molto la qualità dell'immagine originale.
Nella figura 7, invece, vediamo un esempio di minor compressione. In questo caso vengono utilizzati su un'immagine 512x512 i parametri 15 per la grandezza dei macro-blocchi e 10 per la soglia di taglio, portando alla formazione di una matrice quadrata con righe e colonne divise in 34 blocchi. Rispetto a prima la porzione di frequenze tagliate è proporzionalmente molto più contenuta. Nel confronto si può osservare che l'immagine processata non presenta rilevanti differenze visive rispetto all'originale.

Un'ulteriore sperimentazione è stata quella di far variare sulla stessa immagine prima il parametro della dimensione dei blocchi mantenendo fissa la soglia di taglio, per poi fare il viceversa. L'immagine su cui sono state svolti i test è quella in figura 8.

[9]Si ricorda che i pixels in eccesso rispetto ai blocchi vengono direttamente scartati causando un eventuale ridimensionamento dell'immagine

Figura 8: Immagine originale sequenza esempi - 1011x661 — 1.9Mib

La sequenza delle figure 9-10-11 mostra i risultati ottenuti fissando a 10 la soglia di taglio delle frequenze e facendo variare la dimensione dei blocchi tra i valori 10-50-100. Come si può vedere la qualità dell'immagine compressa va peggiorando, in quanto l'aumentare della dimensione dei blocchi non seguito da un aumento anche della soglia di taglio porta a scartare per ogni blocco una porzione sempre più ampia di frequenze. Un'osservazione che si può fare è che, dovendo processare meno blocchi, anche i tempi di esecuzione si abbassano assieme alla qualità. Le immagini sono state computate rispettivamente in 1.110s, 0.325s e 0.265s.

Figura 9: Esempio fissata soglia - F=10, d=10; 1010x660 — 653.3Kib — 1.110s

Figura 10: Esempio fissata soglia - F=50, d=10; 1000x650 — 635.8Kib — 0.325s

Figura 11: Esempio fissata soglia - F=100, d=10; 1000x600 — 587.0Kib — 0.265s

Con la sequenza di figure 12-13-14 sono mostrati i risultati ottenuti fissando a 50 la dimensione dei blocchi e facendo variare la soglia di taglio tra i valori 1-10-30. In questo caso si può notare che il comportamento è opposto rispetto a quello della precedente sequenza di immagini. La causa del miglioramento della qualità è che per ogni blocco vengono preservate sempre più frequenze.
Contrariamente a quanto succede in precedenza, in questo caso i tempi impiegati restano molto simili tra loro perché il numero di blocchi da processare resta il medesimo in tutti e tre i casi. Le immagini sono state processate rispettivamente in 0.320s, 0.321s e 0.330s.

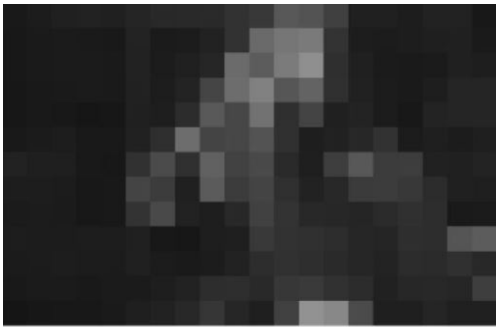

Figura 12: Esempio fissata soglia - F=50, d=1; 1000x650 — 635.8Kib — 0.320s

Figura 13: Esempio fissata soglia - F=50, d=10; 1000x650 — 635.8Kib — 0.321s

Figura 14: Esempio fissata soglia - F=50, d=30; 1000x650 — 635.8Kib — 0.330s

Riferimenti bibliografici

[1] Python Software Foundation, "What is python? executive summary." https://www.python.org/doc/essays/blurb/.

[2] Python Software Foundation, "History and license." https://docs.python.org/3/license.html.

[3] NumPy developers, "Numpy." https://numpy.org/.

[4] SciPy developers, "Scipy library." https://www.scipy.org/scipylib/index.html.

[5] SciPy developers, "Scipy." https://www.scipy.org/.

[6] SciPy developers, "Scipy license." https://www.scipy.org/scipylib/license.html.

[7] NumPy developers, "Numpy license." https://numpy.org/license.html.

[8] Qt, "Qt for python." https://wiki.qt.io/Qt_for_Python.