

MCS 2020
Algebra lineare numerica
Compressione di immagini tramite la DCT

Silva Edoardo 816560, Zhigui Bryan 816335, Marchetti Davide 815990

21/05/2020

1 Abstract

Si vuole presentare un software che implementa una compressione delle immagini in modo tale che esegua i task descritti nella traccia.

Il software permette all'utente:

1. Scegliere dal filesystem un'immagine .bmp in toni di grigio riportando un messaggio d'errore qual ora scegliesse un formato diverso oppure a colori.
2. Scegliere un valore intero F che determina il valore massimo che può scegliere del valore d che né indicherà la soglia di taglio delle frequenze.
3. Visualizzare sullo schermo affiancante: l'immagine originale e l'immagine ottenuta post compressione.
4. Ridimensionare l'immagine per riempire l'area mancante rispetto a quella originale.

Il software è stato scritto in python per lo sviluppo della GUI moderna con PyQt5.

2 Implementazione

Il software inizialmente presenta un'interfaccia dotata con le seguenti caratteristiche:

1. Un pannello diviso a metà dedicando una sezione all'immagine originale e una sezione all'immagine compressa;
2. Una sezione di parametri nella quale sono suddivise in due parti:
 - A **Input:** permette di andare a reperire l'immagine e scalarla alle dimensioni disponibili con l'apposito check-box
 - B **Parameters:** definisce i parametri sulla qualità di compressione dell'immagine.

Il programma in **back-end** va a connettersi ai singoli eventi che deve avviare mano a mano che l'utente decide di effettuare un'operazione settando o facendo sparire alcune informazioni non più rilevanti.

3 Libreria PyQt5

PyQt5 è una libreria che consente di usare il framework QT5 GUI che serve per creare GUI application nel linguaggio C++.

PyQt5 è un toolkit multiplatforma che può essere eseguito su quasi tutti i sistemi operativi.

Usandolo con Python, è possibile creare applicazioni molto rapidamente senza perdere gran parte della velocità del C++

I moduli utilizzati in particolare sono:

- QtWidgets
- QtCore
- QtGUI
- QtWidgets

4 Richiesta immagine

Solitamente, un'immagine a colori è composta da quattro canali ben distinti, identificati con la sigla “**RGBA**”, ovvero Red, Green, Blue e Alpha (Trasparenza). La loro combinazione permette la definizione di un qualunque colore.

La traccia richiede di lavorare in toni di grigio nella quale si è fatta un'osservazione sulla sua espressione ovvero, in figura . . . si può notare che per rappresentare una determinata tonalità di grigio, il valore della componente R, G e B coincide.

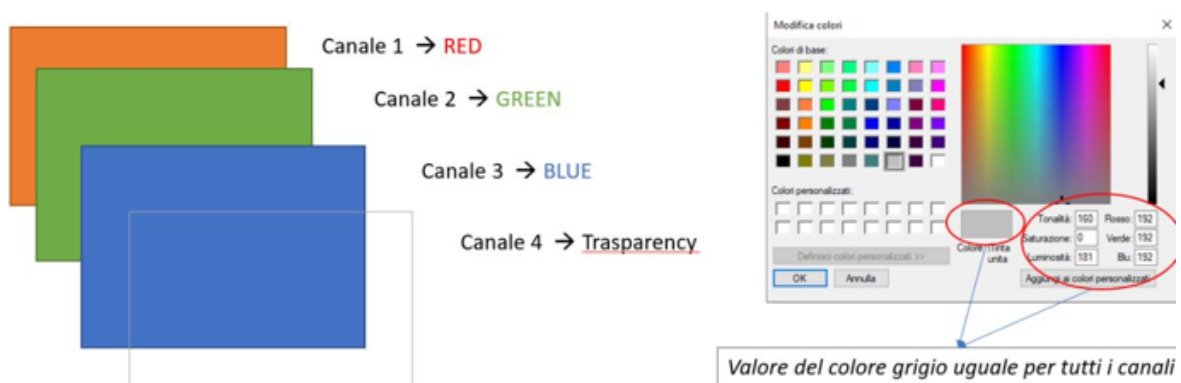


Figure 1: Schema canali RGBA

Un'immagine viene rappresentata in termini di pixel e sappiamo che:

- Un pixel equivale a 1Byte (ovvero 8 bit);
- 1 Byte rappresenta un numero tra 0-255.

Quindi, data un'immagine si va creare un buffer di grandezza data da altezza x larghezza x num_canali, contenente il numero totale di bit dell'immagine. A questo punto si va a definire la funzione (vedere in figura . . .) che raggruppa i bit di buffer in gruppi da 8, per ottenere i singoli Byte come intero senza segno (questo ci permetterà di non arrotondare il parametro **ff** all'intero più vicino).

Infine, si con i valori contenuti nel primo canale (Red).

```
pixels = np.frombuffer(values, np.uint8).reshape((height, width, 4)).copy()  
pixels = pixels[:, :, 0]
```

Figure 2: Funzione pixels

Si spiega l'algoritmo nei seguenti passi:

1. L'immagine originale viene trasformata in PixMap;
2. Si definiscono i valori per ogni pixel indicando il canale su cui lavorare;
3. Setta i valori di **f** e **d** immessi dall'utente;
 - (a) Dato **F** si definisce l'ampiezza dei macro-blocchi;
4. Quindi, l'immagine viene divisa in blocchi **f** di pixels di dimensione $F \times F$, partendo dal primo blocco in alto a sinistra;
5. Per ogni blocco si effettuano le seguenti attività:
 - (a) Si applica la dctn (DCT2 della libreria): $\mathbf{c} = \mathbf{DCT2}(\mathbf{f})$;
 - (b) Vengono eliminate le frequenze ckl con $k+l \geq d$;
 - (c) Si applica idct_n sulla matrice c: $\mathbf{ff} = \mathbf{IDCT2}(\mathbf{f})$
6. Trasforma da bit a immagine;
7. Riporta l'immagine compressa sulla sezione dedicata;

Viene riportata in figura... Un modello grafico dei passaggi 4 e 5 evidenziando che la ricomposizione dell'immagine non viene effettuata perché una volta finita la compressione sul blocco sarà sostituito direttamente nel blocco occorrente.

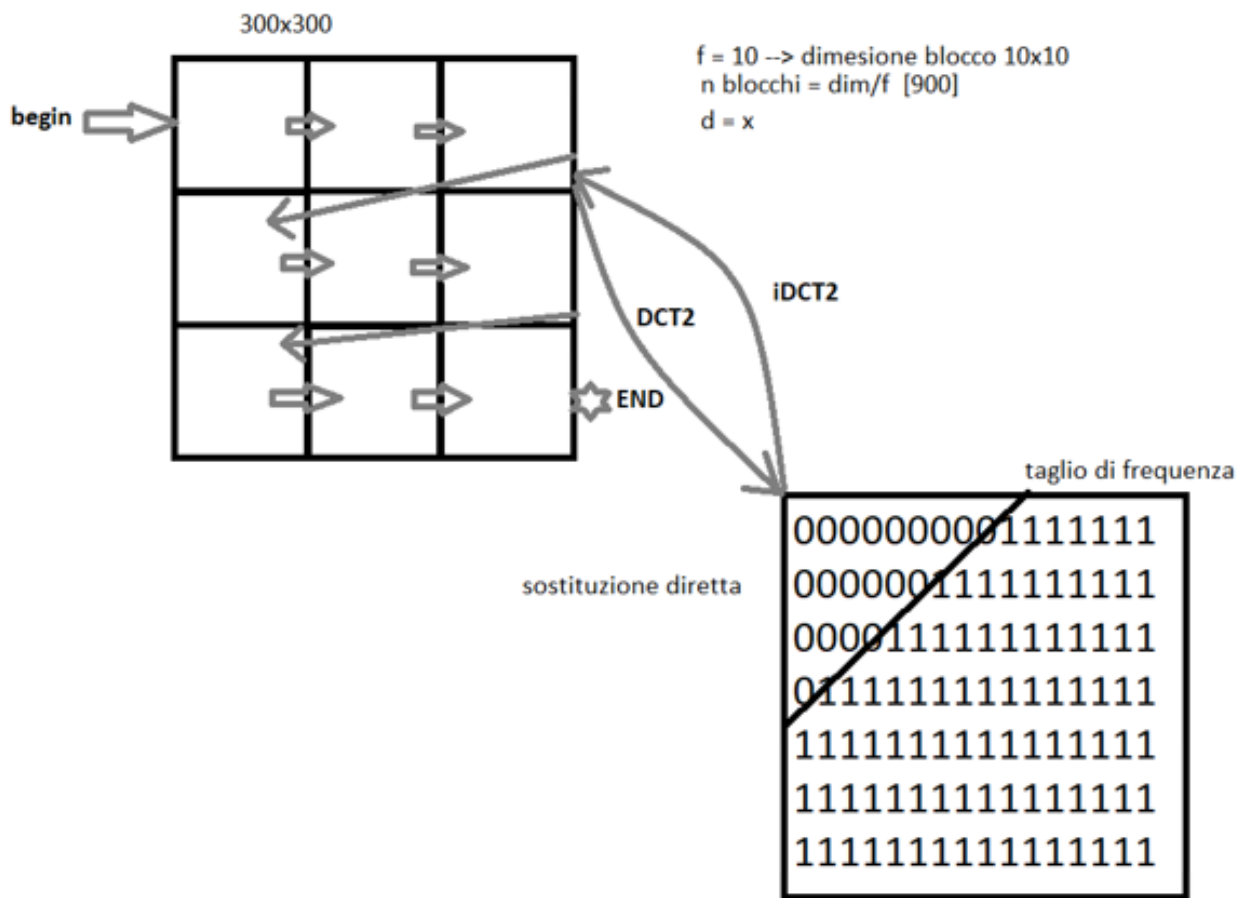


Figure 3: Schema funzionamento DCT

Di seguito verranno presentati tre esempi di compressione dell'immagine su immagini distinte.

L'applicazione richiede all'utente di andare a reperire un'immagine cliccando sul bottone "open file" che presenta inizialmente la cartella dove si trova il progetto perché contiene una cartella apposita `..\folder\resources` dove può reperire immagini di esempio, ma può anche sceglierne una in locale del suo pc.

Una volta presa l'immagine va a verificare:

1. Se il formato è .bmp, altrimenti presenta in message-box-error indicando che il formato dell'immagine è sbagliato;

2. Se l'immagine passata non sia a colori presentando un message-box-error indicando che l'immagine deve essere in toni di grigio.

L'immagine rappresentata contiene una buona qualità di compressione avendo scelto opportunamente i valori **f** e **d**. Dalla figura si può notare che sono quasi identiche ma sul contrasto dal colore tutto bianco al colore in tonalità di grigio mostra alcune discrepanze che sono minime visto i valori scelti.



Figure 4: Prima immagine /resources

L'immagine esposta rappresenta il fenomeno di **Gibbs** causato dal passaggio del colore nero su bianco e viceversa. Come ben sappiamo, il fenomeno si propaga ed è per questo che sono presenti “cerchi” molto grandi rispetto alle dimensioni dei blocchi f.



Figure 5: Seconda immagine /resources

5 Compressione immagine

Una volta che viene caricata l'immagine l'utente dovrà definire i valori di F e d in modo tale da studiare la qualità della compressione con diversi esempi che elencheremo alla fine.

Si spiega l'algoritmo nei seguenti passi:

1. L'immagine originale viene trasformata in formato pixmap;
2. Si definisce il numero totale di bit in rapporto con il numero di canali (RGBA= 4 canali);
3. Setta i valori di F e d immessi dall'utente;

A Dato F si definisce l'ampiezza dei macro-blocchi;

4. Quindi l'immagine viene divisa in blocchi f di pixels di dimensione $F \times F$, partendo dal primo blocco in alto a sinistra;

5. Per ogni blocco si effettuano le seguenti attività:
 - A Si applica la dctn (DCT2 della libreria): $c = \text{DCT2}(f)$;
 - B Vengono eliminate le frequenze ckl con $k+l \geq d$;
 - C Si applica idctn sulla matrice c: $ff = \text{IDCT2}(f)$
6. Trasforma da bit a immagine;
7. Riporta l'immagine compressa sulla sezione dedicata.

6 Osservazioni

1. Le immagini richiedono di lavorare con i 4 tipi di canale: RGBA che in combinazione definiscono il colore.

La traccia richiede di lavorare con immagini in toni di grigio, dalle analisi fatte (come in figura in alto a destra) si nota che il valore di *un qualunque grigio* è uguale per tutti i canali.

Di conseguenza si è deciso di lavorare con un solo canale (in questo caso il primo, ovvero Red).

1. Quindi l'immagine viene divisa in blocchi f di pixels di dimensione F x F, partendo dal primo blocco in alto a sinistra;
2. Per ogni blocco si effettuano le seguenti attività:
 - A Si applica la dctn (DCT2 della libreria): $c = \text{DCT2}(f)$;
 - B Vengono eliminate le frequenze ckl con $k+l \geq d$;
 - C Si applica idctn sulla matrice c: $ff = \text{IDCT2}(f)$

7 Conclusioni

L'app di compressione riscontra buoni risultati scegliendo i valori f e d opportuni per l'immagini in toni di grigio mentre per quelle definite solo in due colori ben distinti (black & white) mostra il fenomeno di Gibbs ma questo succede anche con MatLab.