Metodi del Calcoli Scientifico

Compressione di immagini tramite DCT

Progetto 2 / Parte 2

Edoardo Silva 816560
Bryan Zhigui 816335
Davide Marchetti 815990

A.A.: 2019/2020

1 Abstract

Si vuole presentare un software che implementa una compressione delle immagini basata sulla DCT.

Il programma deve permettere all'utente di scegliere da filesystem un'immagine bitmap in toni di grigio, applicarne la compressione secondo i parametri specificati e visualizzarne l'output.

2 Implementazione

L'applicazione è scritta in Python utilizzando PyQT5 che agisce da wrapper per QtGUI: un framework per la scrittura di interfacce grafiche in C++.

Inoltre, PyQT5 è multipiattaforma e tramite Python è possibile creare applicazioni molto rapidamente senza perdere i benefici della velocità del C++. I moduli utilizzati dalla nostra applicazione sono:

`QtCore:` definisce una serie di classi utilizzabili per definire stili e comportamenti dei widget.

`QtGUI:` contiene utility per l'elaborazione di dati che interagiscono con componenti dei widget.

`QtWidgets:` racchiude tutta una collezione di widget grafici come pulsanti, label, slider, controlli per form e molto altro.

L'interfaccia grafica è disegnata con l'ausilio di **QtDesigner**, uno strumento facente parte del framework stesso.

Questo permette di ridurre notevolmente la complessità della creazione di un'applicazione, in quanto è necessario solamente implementare i metodi di gestione degli eventi e connetterli ai rispettivi componenti.

3 Interfaccia dell'applicazione

La fig. 1 riporta l'interfaccia dell'applicazione proposta all'utente presentando:

- un pannello diviso in due sezioni per permettere di visualizzare affiancate: l'immagine originale e la rispettiva versione compressa.
- un'area per scegliere un immagine da utilizzare come input con una checkbox per visualizzare l'immagine adattandola all'area predisposta.
- una sezione nella quale regolare i parametri di compressione:
  1. F: definisce la dimensione del blocco. L'immagine sarà suddivisa in blocchi $F \times F$ durante il processo di compressione.
  2. d: soglia di taglio delle frequenze di ogni blocco.

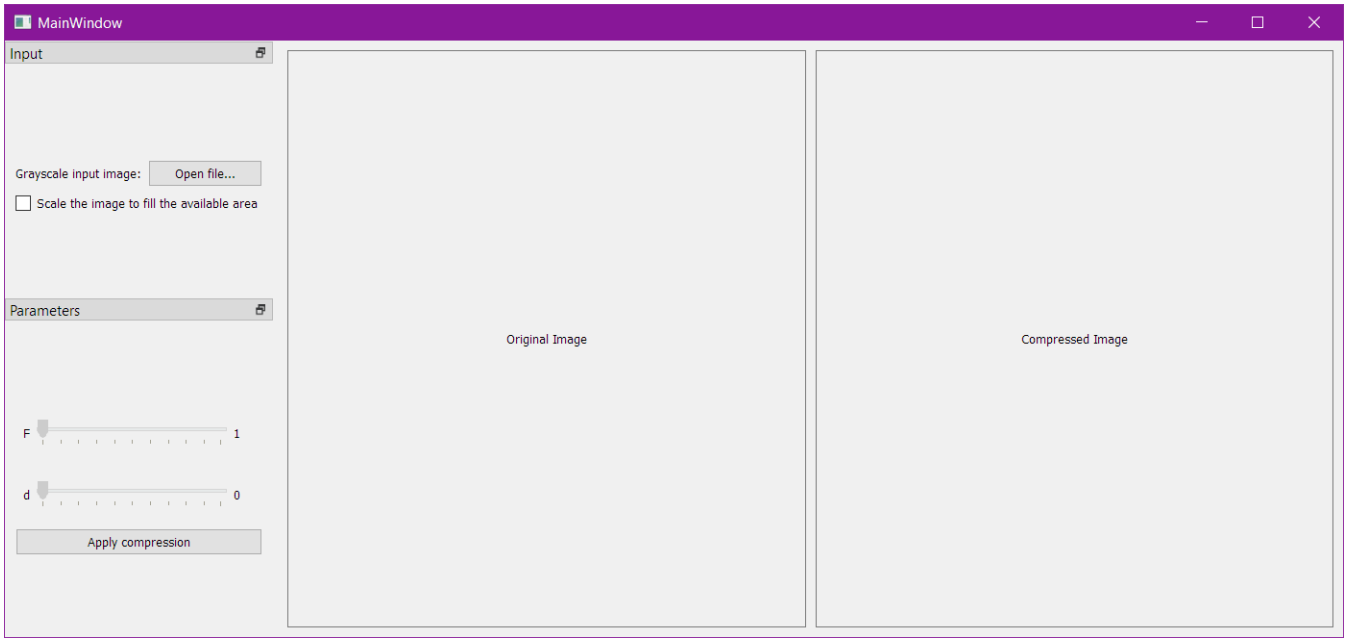

Figura 1: GUI dell'applicazione

Event handling

Il listato elenco 1 riporta la definizione del collegamento degli eventi dei widget dell'interfaccia al relativo gestore.

Listing 1: Event-handling dei segnali dei widget

```
def connectSignals(self):
    self.openFile.clicked.connect(self.ask_image)

    self.scaleImage.stateChanged.connect(self.updateImageScaling)

    self.fSlider.valueChanged.connect(self.updateFLabel)
    self.fSlider.valueChanged.connect(self.updateMaxDValue)

    self.dSlider.valueChanged.connect(self.updateDLabel)

    self.applyButton.clicked.connect(self.compress_image)

def updateImageScaling(self):
    for target in [self.originalImage, self.compressedImage]:
        target.setShouldScale(self.scaleImage.isChecked())

def updateFLabel(self):
    self.fValue.setText(str(self.fSlider.value()))

def updateDLabel(self):
    self.dValue.setText(str(self.dSlider.value()))

def updateMaxDValue(self):
    if self.fSlider.value() == 1:
        self.dSlider.setValue(0)
        self.dSlider.setEnabled(False)
        return

    limit = 2*self.fSlider.value() - 2

    self.dSlider.setMaximum(limit)
    self.dSlider.setTickInterval(limit / 10)
    self.dSlider.setEnabled(True)
```

4 Funzionalità

4.1 Richiesta immagine

Si richiede all'utente di selezione un'immagine `.bmp` in scala di grigi. Attraverso il pulsante `Open file` è possibile reperire delle immagini di esempio presenti nella cartella `./resources`, oppure sceglierne una personalizzata dal proprio dispositivo.

La finestra di dialogo è predisposta per filtrare le immagini selezionabili imponendo la scelta di immagini con estensione `.bmp`. All'apertura viene effettuato il controllo che l'immagine sia effettivamente in scala di grigi. In caso di fallimento, sarà visualizzata una message box per notificarne l'errore.

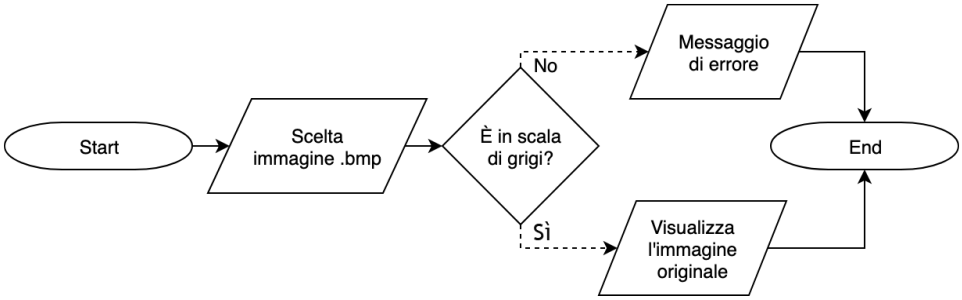

Figura 2: Flowchart di scelta dell'immagine

L'immagine viene caricata nella prima sezione del pannello permettendo di scalarla come in fig. 3.

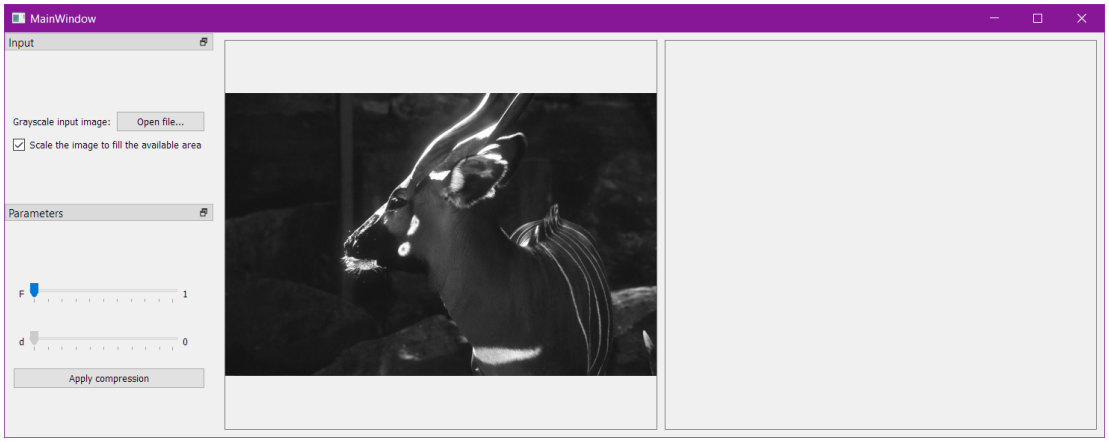

Figura 3: GUI: imagine caricata correttamente

4.2 Compressione dell'immagine

Per procedere con la compressione è necessario impostare i parametri F e d. Questo permetterà di studiare gli effetti della compressione tramite alcuni esempi riportati nella sezione 5.

Note sulla compressione

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

La traccia richiede di lavorare con immagini in toni di grigio. Queste verranno lette dall'applicazione come immagini a colori per le quali ogni pixel occupa 32bit (8 bit per ogni canale).

La fig. 4 illustra come la rappresentazione di una determinata tonalità di grigio comporti lo stesso valore nelle componenti R, G e B.

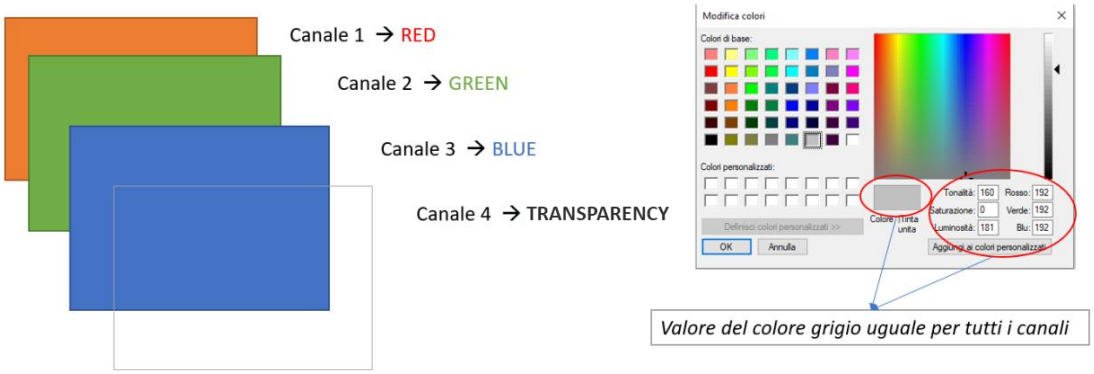

Figura 4: Schema canali RGBA

Il listato 2 riporta le istruzioni di lettura ed estrazione dei pixel utilizzate dal nostro algoritmo. Si può notare che dati i pixel di un immagine, si specifica il numero di byte presenti nella stessa ottenendo un buffer di dimensione `altezza` × `larghezza` × `canali`.

Il buffer di bit viene suddiviso in gruppi da 8bit (`uint8`) per ottenere i singoli byte di ciascuna componente dell'immagine. In conclusione, si estrae solamente il primo livello, equivalente alla componente `R` dell'immagine.

I valori del singolo canale pur essendo interi vengono convertiti in float, per evitare troncamenti o overflow nell'applicazione della DCT ed IDCT negli step successivi.

Listing 2: Estrazione del canale R dall'immagine

```
1 # Source rappresenta la mappa dei pixel dell'immagine
2 values = source.constBits()
3 values.setsize(height * width * 4)
4
5 pixels = np.frombuffer(values, np.uint8).reshape((height, width, 4))
      ↪ .copy()
6 pixels = pixels[:, :, 0].astype(np.float)
```

Algoritmo

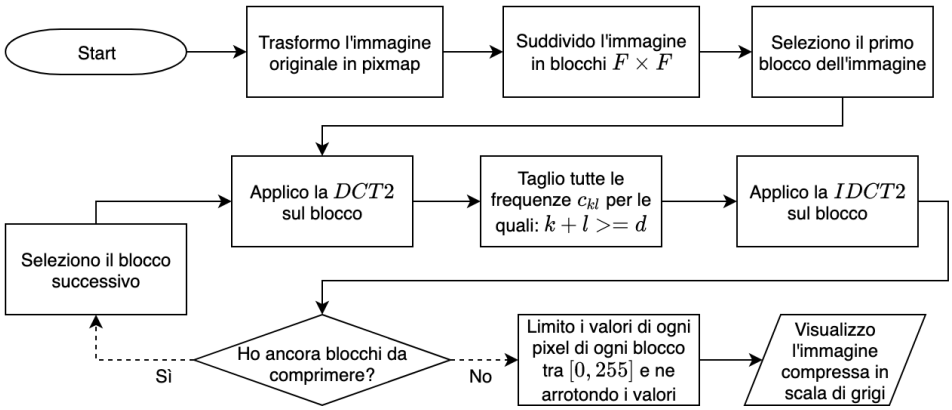

Figura 5: Flowchart di compressione dell'immagine

L'algoritmo implementato e riportato in fig. 5 e nel listato 3 lavora nel seguente modo:

1. L'immagine originale viene trasformata in `PixMap` dalla quale viene estratto uno tra i canali `R`, `G` e `B`
2. Vengono recuperati i valori di F e d immessi dall'utente
   (a) F definisce l'ampiezza dei macro-blocchi

(b) d regola da quale frequenza tagliare il blocco in fase di compressione

3. L'immagine viene divisa in blocchi z di dimensione $F \times F$ contenenti i rispettivi pixel dell'immagine originale, partendo dal primo blocco in alto a sinistra (fig. 6).

   Nota: Eventuali pixel in eccesso per i quali non è possibile costruire un quadrato che li contenga non verranno compressi e saranno copiati direttamente dall'immagine originale.

4. Per ogni blocco si effettuano le seguenti operazioni effettuando una sostituzione diretta nella matrice di pixel:

   (a) Si applica la $DCT2$ della libreria: $C = DCT2(z)$

   (b) Vengono azzerate le frequenze C_{kl} dove $k + l >= d$

   (c) Si applica $IDCT2$ sulla matrice C: $z = IDCT2(C)$

5. I valori della matrice di pixel vengono limitati tra $[0, 255]$ e arrotondati all'intero più vicino

6. La `PixMap` viene riconvertita in immagine e visualizzata nella sezione dedicata, affiancata all'immagine originale.

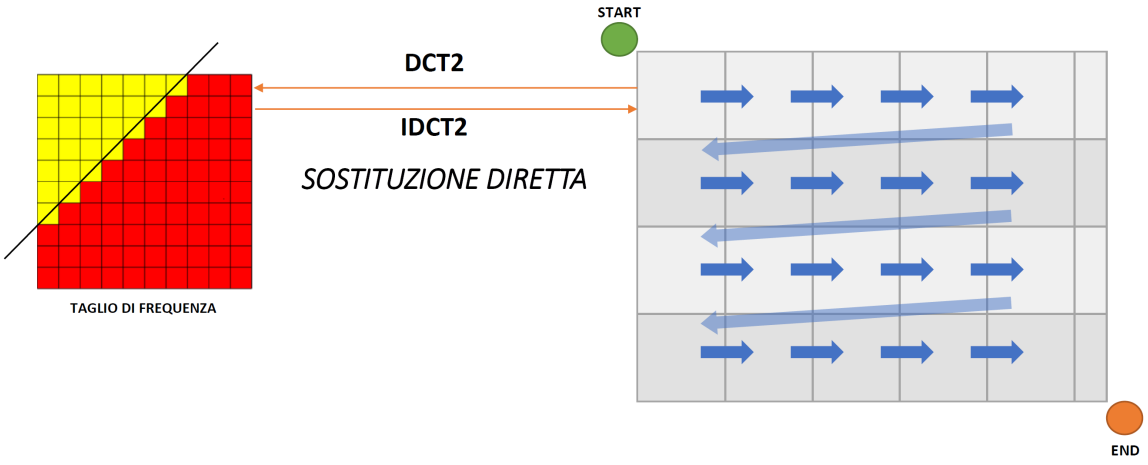

Figura 6: Compressione a blocchi

Listing 3: Implementazione in Python dell'algoritmo di compressione

```
def compress_image(self):
    source = self.originalImage.pixmap.toImage()
    width, height = [source.width(), source.height()]

    # bits() -> deep copy // constBits() -> references
    values = source.constBits()
    values.setsize(height * width * 4)

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

    F = self.fSlider.value()
    d = self.dSlider.value()

    height_blocks_count = int(height / F)
    width_blocks_count = int(width / F)

    for r in range(height_blocks_count):
        for c in range(width_blocks_count):
            block = dctn(pixels[r*F : (r+1)*F, c*F : (c+1)*F], norm='
                ↪ ortho')

            for k in range(F):
                for l in range(F):
                    if k + l >= d:
                        block[k,l] = 0

            pixels[r*F : (r+1)*F, c*F : (c+1)*F] = idctn(block, norm=
                ↪ 'ortho')

    pixels = pixels.clip(0, 255).round().astype(np.uint8)

    target = QImage(bytes(pixels), pixels.shape[1], pixels.shape[0],
        ↪  int(pixels.nbytes/height), QImage.Format_Grayscale8)
    self.compressedImage.setPixmap(QPixmap.fromImage(target))
```

5 Risultati

Negli esempi proposti, la parte in eccesso derivante dalla divisione in blocchi dell'immagine è copiata nell'immagine finale senza compressione di alcun tipo (fig. 7), questo per non alterare la dimensione finale dell'immagine.

Figura 7: Parametri: $F = 30$ e $d = 10$

Il primo confronto, rappresentato in fig. 8 e 9, mostra la variazione del parametro d, cioè il numero delle frequenze tagliate durante la compressione.

Figura 8: Parametri: $F = 124$ e $d = 138$

Pur eliminando quasi la metà delle frequenze in fig. 8, questa risulta indistinguibile rispetto al file originale.

La fig. 9 dimostra invece come più viene applicata una compressione massiccia (parametro d basso), più sarà presente il delineamento della divisione in blocchi.

Figura 9: Parametri: $F = 124$ e $d = 10$

Fenomeno di Gibbs

Le immagini in fig. 10 e 11 rappresentano il fenomeno di **Gibbs** causato dal passaggio netto dal colore nero al colore bianco (rispettivamente con valori 0 e 255) o viceversa, manifestandosi come un aloni che seguono i contorni della C e del cervo.

Figura 10: Fenomeno di Gibbs / Parametri: $F = 100$ e $d = 98$

Dall'immagine in fig. 11 si può osservare un comportamento piuttosto interessante: il fenomeno di Gibbs si propaga esclusivamente all'interno del blocco nel quale si verifica. Proprio per questo motivo, la compressione JPG classica pur operando in modalità differente, utilizza blocchi di dimensione F molto piccoli (solitamente 8×8).

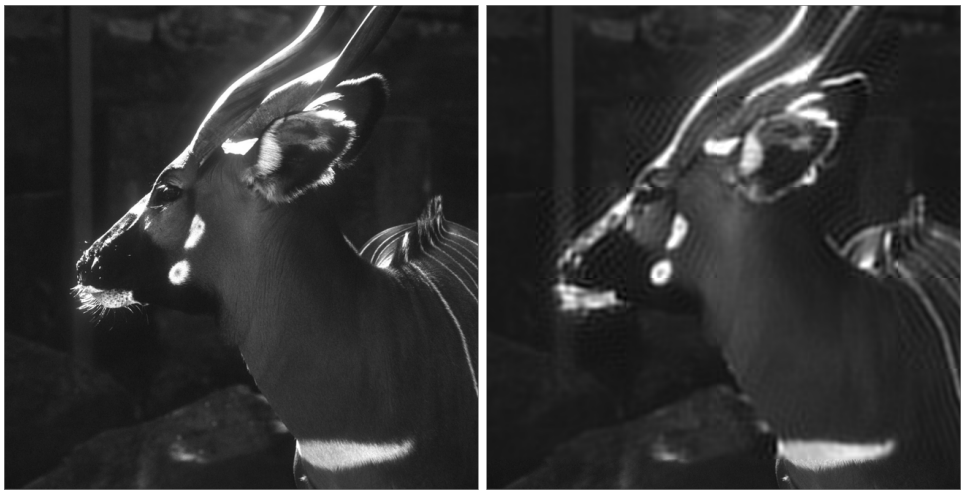

Figura 11: Fenomeno di Gibbs / Parametri: $F = 124$ e $d = 28$