Metodi del Calcolo Scientifico

Progetto 2 - Compressione di immagini con la DCT

Andrea Angiolillo 761678
Corrado Ballabio 764452

July 12, 2016

Lo scopo di questo progetto è di studiare una estensione della compressione jpeg introducendo, rispetto allo standard, la possibilità di utilizzare blocchetti di 8N $\times$ 8N pixel con N scelto dall'utente invece che semplicemente 8 $\times$ 8.

Assunzioni(?)

Scelte implementative

Per quanto riguarda la scelta del linguaggio di programmazione da utilizzare, si è deciso di sviluppare il codice con *Python*. Questo perchè rappresenta uno dei più diffusi e versatili linguaggi di programmazione open-source, supportato da una ampia e sempre attiva community on-line e dispone di numerosissime librerie che lo rendono adattabile a qualsiasi contesto.
Le librerie usate per questo progetto sono quattro:

- **NumPy**
  Si tratta della libreria fondamentale usata per tutte le operazioni matematico-scientifiche nell'ambiente di Python. Permette tra le molte funzionalità di gestire facilmente vettori n-dimensionali e di sfruttare diverse utilità dell'algebra lineare. In questo progetto è stato sfruttato principalmente per la gestione di tutte le matrici e degli arrays.

- **scipy**
  È una libreria open-source di ambito scientifico che fornisce importanti strumenti scientifici per più scopi. Nel progetto in questione è stato utilizzato solo un suo pacchetto, denominato *fftpack*. Questo contiene

tutte le funzioni e gli strumenti per la gestione della trasformata discreta di Fourier.

- **PIL - *Python Image Library***
  Questa libreria aggiunge all'interprete Python le funzionalità necessarie per la gestione e manipolazione delle immagini. È stato qui utilizzato il pacchetto *Image*, che fornisce utili capacità come la apertura e la creazione di immagini supportando diversi tipi di estensione di file.

- **PiQt4**
  La libreria in questione ha permesso lo sviluppo dell'interfaccia grafica, attraverso diversi metodi che rendono possibile e facilmente intuibile la creazione di finestre, buttons e tutti gli altri widgets.

Implementazione

Sono state utilizzate alcune strutture dati fisse, utili come variabili d'appoggio che sarebbero poi servite per calcoli futuri:

- **matrice *q*** ovvero la matrice di quantizzazione, dato che i suoi valore sono fissi, si è creata una variabile elencando uno per uno i valori di essa.

- **matrice *matrix_of_128s*** è una matrice di dimensione 8N $\times$ 8N che viene sfruttata in alcuni step del processo di compressione.

La prima parte dell'implementazione comprende tutta la fase preliminare di preparazione dei dati, mentre la seconda comprende il vero e proprio processo di conversione dell'immagine da formato *.bmp* a *.jpg*, e lavora su piccole porzioni del file alla volta.

All'interno del costruttore della classe vengono subito settate le variabili di istanza con gli opportuni valori scelti tramite l'interfaccia.
I più importanti di questi sono *quality*, che indica il livello di qualità della compressione, e n, la dimensione dei blocchi in cui verrà divisa l'immagine. Viene pure impostato il valore di *qf*, a seconda del caso in cui *quality* sia maggiore o minore di 50. Di seguito si imposta la matrice *q1*, che viene ricavata da q e aumentata di dimensioni fino al valore indicato dal parametro n; viene infine moltiplicata per il valore di *qf*. Come ultima azione viene recuperata l'immagine tramite il percorso del file, e attraverso alcuni metodi (vedi documentazione) l'immagine viene convertita in scala di grigi e castata in una matrice.

Si lavora ora direttamente sull'immagine, il processo di conversione e tutti i suoi passi sono racchiusi nel metodo *convert_img*. Viene ora spiegato il funzionamento di ciascun passo alla volta.

- **PASSO 0**
  Vengono "aggiustate" le dimensioni originali dell'immagine attraverso il metodo *resize_img*, al fine di renderla perfettamente divisibile in blocchi 8N $\times$ 8N. È ora possibile scomporre attraverso due cicli for l'immagine in blocchi delle dimensioni adatte, e lavorare su ognuno di essi sequenzialmente attraverso i passi 1-8 e attuare così la compressione.

- **PASSO 1**
  Viene semplicemente detratto ciascun valore del blocco di 128 unità.

- **PASSO 2**
  Viene applicata la funzione *get_2d_dct* (vedi documentazione) appartenente al package fftpack.

- **PASSO 3**
  Si divide il blocco con la matrice *q1* e si arrotonda poi il risultato.

- **PASSO 4**
  Ora si rimoltiplica sempre per la matrice *q1*.

- **PASSO 5**
  Viene applicata la funzione *get_2d_idct*, (vedi documentazione) che performa la dct inversa sul blocco.

- **PASSO 6**
  Si somma ad ogni elemento del blocco le 128 unità prima sottratte, e si procede poi ad arrotondare il risultato.

- **PASSO 7**
  In ogni elemento si abbassano gli elementi eccessivamente grandi di modo che siano tutti minori di 255, e quelli eccessivamente piccoli per essere tutti maggiori di 0.

- **PASSO 8**
  In quest'ultimo passo si procede a copiare i valori così ottenuti nel blocco all'interno dell'immagine originale; ha così inizio un nuovo ciclo e si ripetono tutti i passaggi 1-8 fino ad aver elaborato l'immagine intera.

Il metodo si conclude con la chiamata al metodo *get_reconstructed_image* (vedi documentazione) che a partire dalla matrice a valori ricostruisce il file immagine.

Documentazione

- **library PIL - package Image**
  - *Image.open(file)* - convert_to_jpeg.py - riga 87
    Permette di aprire e identificare l'immagine specificata da *file* attraverso una stringa che indica il filepath e il filename. È una operazione di tipo *lazy*, la funzione legge il file header ma l'immagine viene effettivamente caricata solo appena si cerca di processare i dati.
  - *image.convert(mode)* - convert_to_jpeg.py - riga 88
    Converte un'immagine in una altra modalità *mode*. Nel progetto in questione è stata usata la modalità *"L"*, che trasforma *image* in una corrispondente immagine in scala di grigi utilizzando la trasformata ITU-R 601-2 luma; alternativamente è possibile utilizzare i parametri *"RGB"* e *"CMYK"*. Ritorna infine una copia dell'immagine come matrix.
  - *Image.fromarray(image)* - convert_to_jpeg.py - riga 135
    Crea in memoria e ritorna un'immagine a partire dall'oggetto matrice *image*, attraverso il protocollo dell'interfaccia array *buffer*.
- **library Scipy - package fftpack**
  - *dct(img, norm)* - convert_to_jpeg.py - riga 126
    Prende come parametri l'immagine sotto forma di array *img* e la modalità di normalizzazione *norm*, che può essere "none" oppure "ortho". Ritorna infine l'array preso in input trasformato.
  - *idct(img, norm)* - convert_to_jpeg.py - riga 130
    Opera allo stesso modo di *dct()* ma effettua la trasformata inversa sull'array passato come paramentro.

Avvio Programma

Per avviare il programma bisogno aprire eclipse e selezionare il file *convert_to_jpeg.py*. Una volta selezionato il corretto file, premendo il tasto *Run*, comparirà l'interfaccia mostrata nella seguente figura.

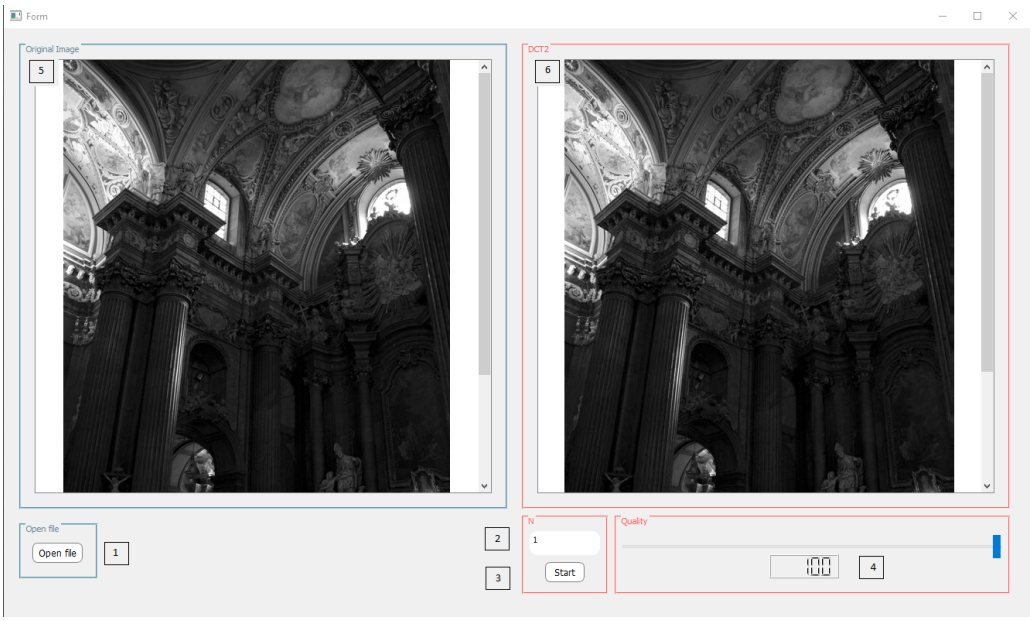

Figure 1: GUI

La *GUI* è composta dai seguenti elementi:

1. *Open File*: Permette di selezionare l'immagine da utilizzare per la *DCT2*.

2. *N*: In questa casella di testo è possibile definire la variabile *N* che verrà utilizzata dall'algoritmo per impostare la grandezza dei quadranti.

3. *Start*: Una volta selezionata l'immagine, scelta la *N* e la *Quality*, questo tasto permette di avviare l'algoritmo.

4. *Quality*: La barra scorrevole permette di selezionare il livello di *Quality* desiderato, il valore è compreso tra 1 e 100.

5. *Original Image*: In questo riquadro viene mostrata l'immagine in input all'algoritmo.

6. *DCT2*: In questo riquadro viene mostrato l'output del nostro algoritmo. L'immagine oltre ad essere mostrata a video viene salvata sul Desktop per permettere all'utente di visualizzarla nella sua interezza.