Elaborazione di Segnali Multimediali

Rappresentazione delle immagini

L.Verdoliva, D.Cozzolino

In questa prima lezione vedremo come si rappresentano diversi tipi immagini digitali in Python. Vedremo anche come caratterizzare le immagini digitali in Python tramite media e varianza e ne valuteremo l'istogramma.

Per questa esercitazione useremo diverse liberie Python, in particolare Scipy e Scikit-Image. Di seguito le istruzioni che ci permettono di importare i moduli python che useremo:

```
# attiva la modalita interattiva di matplotlib
%matplotlib qt

import numpy as np              # importa Numpy
import matplotlib.pyplot as plt # importa Matplotlib
import scipy.ndimage as ndi     # importa Scipy per le immagini
import skimage.io as io         # importa il modulo Input/Output di SK-Image
```

1 Immagini su scala di grigio

Un'immagine digitale (monocromatica) può essere rappresentata in Python come una array bidimensionale il cui generico campione $x[m, n]$ (*pixel*) varia nel range $[0, K-1]$. Le immagini sono quindi rappresentate con K livelli di grigio, e ogni pixel è codificato con $\log_2 K$ bit. Scriviamo i comandi Python per leggere da file l'immagine su scala di grigi dorian.jpg e che ci consentono di memorizzarne i valori in una matrice:

```
x = io.imread('dorian.jpg');    # leggiamo il file
(M,N) = x.shape                 # memorizziamo le dimensioni in M e N

# cosa abbiamo adesso in memoria?
%whos
```

Attraverso il comando `io.imread` è possibile leggere immagini con formato standard come JPEG, TIFF, GIF, BMP ed altri. Con il comando `help(io.imread)` potete leggere la guida inline della funzione.

Se l'immagine invece è in formato *grezzo* (cioè il file non ha un header, per cui l'estensione può essere qualsiasi) è necessario conoscerne le dimensioni e il formato. Per esempio, se l'immagine è 512×512 e il tipo di dato è intero senza segno su 8 bit, la lettura avviene con i seguenti comandi:

```
x = np.fromfile('house.y', np.uint8) # lettura dei dati dal file

%whos
# quali sono le caratteristiche di x? x è un vettore

x = np.reshape(x, (512,512))    # reshape permette di costruire una matrice
```

Fate attenzione al fatto che le immagini che volete leggere devono trovarsi nella directory corrente o nel path; in alternativa potete dare il percorso completo. Notate (usando `%whos`) come il comando `io.imread` restituisce direttamente la matrice, mentre il comando `np.fromfile` fornisce un vettore e quindi va applicato `np.reshape` per costruire una matrice. Usualmente la funzione `io.imread` restituisce un array di interi senza segno su 8 bit (`np.uint8`). Dopo la lettura conviene effettuare una conversione in float (`np.float32` o `np.float64`) per evitare errori di arrotondamento durante l'elaborazione dell'immagine. La conversione si ottiene facilmente col comando `x = np.float64(x)`.
Per quanto riguarda invece la visualizzazione si può usare il seguente comando:

```
plt.figure(1);                          # apre la figura 1
plt.imshow(x, clim=[0,255], cmap='gray'); # visualizza l'immagine
```

Se $x[m,n] = k$, il comando `plt.imshow` disegna nel punto $[m,n]$ un pixel col k-esimo colore disponibile. Con il parametro `clim=[low,high]` si indica il range di visualizzazione, mentre `cmap='gray'` indica che l'immagine verrà visualizzata in scala di grigi. Nell'esempio precedente, il valore 0 viene visualizzato come nero, il valore 255 come bianco e tutti i valori intermedi come grigi.

E' anche possibile modificare l'associazione dei livelli di grigio e fare in modo che il nero sia associato al valore minimo e il bianco al valore massimo dell'immagine nel seguente modo:

```
plt.imshow(x, clim=None, cmap='gray'); # usa il range [min(x),max(x)]
```

Se non si forniscono indicazioni sul valore minimo e massimo, allora i dati vengono scalati settando low e high al minimo e massimo valore presenti. Provate a visualizzare sia l'immagine dorian.jpg che granelli.jpg utilizzando le diverse opzioni del comando `plt.imshow`. Come mai la visualizzazione varia così tanto solo per la seconda immagine?

In Python dopo aver elaborato un'immagine e' possibile salvarla in un file con un ben determinato formato con il comando `io.imsave`. Nell'ipotesi in cui i valori da scrivere su file siano memorizzati nella variabile `x`:

```
x = np.uint8(x)            # tipo di dato supportato da JPEG
io.imsave('immagine.jpg',x, quality = 70)
```

In questo modo si sta anche effettuando automaticamente una compressione JPEG con fattore di qualità 70 (tale parametro varia da 0 (qualità peggiore) a 100 (qualità migliore)). Provate a salvare un'immagine di prova a diversi fattori di qualità e visualizzatela.

Se però l'immagine che volete salvare è double e non volete perdere informazione la cosa migliore è salvare i dati con `tofile` senza un header, nel seguente modo:

```
np.uint8(x).tofile('immagine.y') # scrittura del file in uint8
```

Un altro modo di salvare una variabile in Python è quello di usare il comando `np.save`:

```
np.save('immagine.npy', x)  # salva la variabile x
```

Il comando `np.load` permette di caricare nuovamente la variabile nella console.

```
x = np.load('immagine.npy')  # carica la variabile x
```

2 Immagini a colori

Un'immagine a colori in formato standard (JPG, BMP, TIF ...) può facilmente essere importata in Python con il comando `io.imread`. In questo modo si crea una variabile che è una matrice di dimensioni $M \times N \times 3$, le cui componenti coincidono con le componenti di rosso, verde e blu dell'immagine (spazio RGB). Le tre immagini possono essere visualizzate singolarmente su scala di grigi e definiscono le quantità di rosso, verde e blu da mescolare per ottenere il pixel a colori sullo schermo.

```
x = io.imread('fragole.jpg')
plt.figure(); plt.imshow(x);

R = x[:,:,0]
plt.figure();
plt.imshow(R, clim=[0,255], cmap='gray');
plt.title('componente di rosso');

G = x[:,:,1]
plt.figure();
plt.imshow(G, clim=[0,255], cmap='gray');
plt.title('componente di verde');

B = x[:,:,2]
plt.figure();
plt.imshow(B, clim=[0,255], cmap='gray');
plt.title('componente di blu');
```

Il comando `io.imshow` visualizza un'immagine a colori, operando in modo diverso se i dati sono di tipo uint8 oppure float. Nel primo caso i valori della variabile devono essere compresi tra 0 e 255. Nel secondo caso, invece, devono essere normalizzati tra 0 e 1.

Estraendo poi le tre immagini da `x` è possibile visualizzare le componenti RGB (Red, Green, Blue): i valori più luminosi si ottengono in corrispondenza delle componenti di colore che sono presenti maggiormente nell'immagine. Facciamo un esperimento annullando la prima componente (R) e poi ricostruiamo l'immagine:

```
M = x.shape[0]               # numero di righe di x
N = x.shape[1]               # numero di colonne di x
R = np.zeros((M,N), x.dtype) # annullamento della componente di rosso
y = np.stack((R,G,B), -1)
plt.figure();
plt.imshow(y);
```

Provate ad annullare singolarmente anche le altre due componenti e visualizzate l'immagine ricostruita.

2.1 Esercizi proposti

1. *Lettura e visualizzazione di un'immagine*. Provate a scrivere una funzione in grado di visualizzare le immagini sia in formato JPEG che in formato grezzo. Il prototipo delle funzioni deve essere rispettivamente: `vediJPG(nomefile)` e `vediRAW(nomefile,nRighe,nColonne,tipo)`.

2. *Rappresentazione in falsi colori.* La rappresentazione in pseudocolori (o falsi colori) consiste nel visualizzare a colori un'immagine monocromatica. L'obiettivo è quello di migliorare l'interpretazione di un'immagine su livelli di grigio o semplicemente visualizzarla più facilmente. Infatti, l'occhio umano è in grado di discriminare migliaia di variazioni di colore rispetto a qualche decina di livelli di grigio.

   Un esempio di rappresentazione in falsi colori riguarda le immagini multispettrali. Queste sono un insieme di immagini, ognuna delle quali è stata rilevata dal satellite in una diversa banda spettrale. Considerate le 4 immagini telerilevate di Washington, in cui è presente una regione con il fiume Potomac. Le prime tre immagini sono state rilevate nelle tre bande del visibile, mentre la quarta è nel vicino infrarosso. Provate a visualizzare le prime tre immagini come immagine a colori e poi sostituite alla componente R, la quarta immagine, quindi confrontate le due figure. Noterete come nella seconda immagine il fiume (vegetazione) è più facilmente discriminabile dalla città. Infatti il vicino infrarosso è molto sensibile alle parti di una scena contenenti biomasse, e l'immagine mostra effettivamente in maniera chiara le differenze tra le componenti naturali (in rosso) e le costruzioni, composte principalmemte da asfalto e cemento (tendenti al blu).

3 Caratteristiche delle immagini

Prima di realizzare qualsiasi tipo di elaborazione sulle immagini, dopo averle lette, convertitele in float per evitare errori. La libreria Numpy fornisce i seguenti comandi per il calcolo della media e della deviazione standard di un'immagine:

```
xmed = np.mean(x)    # media
xstd = np.std(x)     # deviazione standard
var  = np.var(x)     # varianza = xstd **2
```

Queste grandezze possono anche essere valutate localmente all'immagine. Scegliete un'immagine tra quelle disponibili e supponete di voler stimare l'immagine delle medie locali, cioè l'immagine che in ogni punto presenta il valor medio calcolato in un vicinato 3×3 di ogni pixel. Il codice è il seguente (N.B. per evitare il problema ai bordi, lasciamo invariati i pixel nella cornice più esterna):

```
MED = np.zeros((M-2,N-2))
for i in range(M-2):
    for j in range(N-2):
        MED[i,j] = np.mean(x[i:i+3,j:j+3])

plt.figure(); plt.imshow(MED, clim=[0,255], cmap='gray');
```

Per conoscere i tempi di elaborazione si può utilizzare il modulo `time` nel seguente modo:

```
from time import time
tic = time()
# il tuo codice
toc = time()
print(toc-tic, 'sec')
```

Il calcolo delle statistiche locali può essere realizzato anche usando la funzione `generic_filter` di Scipy nel seguente modo:

```
y = ndi.generic_filter(x, np.mean, (3,3))

plt.figure();
plt.subplot(1,2,1); plt.imshow(x, clim=[0,255], cmap='gray');
plt.subplot(1,2,2); plt.imshow(y, clim=[0,255], cmap='gray');
```

La funzione `ndi.generic_filter` realizza l'operazione specificata attraverso una finestra scorrevole su tutta l'immagine. Usando il modulo `time`, cronometrate il tempo di esecuzione delle due elaborazioni.

Un'analisi più approfondita delle caratteristiche di un'immagine è fornita dal suo istogramma, che rappresenta la frequenza di occorrenza di ogni livello di grigio:

```
n, b = np.histogram(x, np.arange(257));  # istogramma
plt.figure; plt.bar(np.arange(256), n);  # grafico a barre
plt.axis([0,255,0,1.1*np.max(n)]);       # estremi per ascisse e ordinate
```

La funzione `np.histogram` ha in ingresso l'array `x` dei campioni di luminanza, e un vettore che indica le soglie degli intervalli da considerare. Se per esempio, le soglie sono `[0, 1, 2, 3]` allora il range dei valori è diviso in tre intervalli dove il primo intervallo è `[0, 1[` (0 incluso, ma 1 escluso), il secondo intervallo è `[1, 2[` e l'ultimo intervallo è `[2, 3]`, con 3 incluso. La funzione `np.histogram` restituisce due vettori `n` e `b`. Il vettore `b` riporta le soglie utilizzate mentre il vettore `n` indica le occorrenze cioè `n[i]` è il numero di volte che $x[m,n]$ appartiene all'i-esimo intervallo, in questo caso quindi il numero di volte in cui $x[m,n] \in [i, i+1[$. Provate a visualizzare gli istogrammi di alcune delle immagini che avete a disposizione. Provate poi a visualizzare l'istogramma usando i comandi `stem` e `plot` al posto di `bar`.

3.1 Esercizi proposti

1. *Immagine delle medie*. Scrivete una funzione dal prototipo `medie(x,K)` che fornisce l'immagine delle medie locali su finestre $K \times K$. Fate poi un esperimento in cui calcolate l'immagine delle medie al

variare di $K = 3, 5, 7, 9$ e visualizzate il risultato. Che osservazioni potete fare?

2. *Immagine delle varianze*. Scrivete adesso una funzione dal prototipo `varianze(x,K)` per calcolare l'immagine delle varianze locali su finestre $K \times K$. Usate questa funzione per valutare e visualizzare l'immagine delle varianze dell'immagine `filamento.jpg` usando blocchi 3×3. Che tipo di informazioni vi dà sull'immagine?

3. *Istogramma*. Matplotlib fornisce la funzione `plt.hist` per visualizzare un istogramma, usate l'help in linea per conoscerne le funzionalità e provate ad utilizzarlo.

4 Suggerimenti

Alcuni suggerimenti per l'uso di Spyder.

- In Spyder è possibile impostare un'istruzione da eseguire all'avvio della console, per esempio per importare i moduli che si utilizzano più spesso. Dal menù a tendina *Tools* scegliete *Preferences*, quindi selezionate *IPython console* dall'elenco a sinistra e cliccate sulla linguetta *Startup*. Inserite nella casella *Lines* l'istruzione da eseguire all'avvio ad esempio:

```
import numpy as np; import matplotlib.pyplot as plt; import skimage.io as io
```

- Gli script e moduli che scrivete devono sempre trovarsi nella directory corrente affinché possiate utilizzarle. In realtà è possibile modificare il *PYTHONPATH* di Spyder, cioè aggiungere il percorso in cui si trovano i vostri moduli al percorso di default in cui Python cerca i package standard. In questo modo non sarà necessario trovarsi nella directory corrente per richiamare le funzioni. Dal menù a tendina *Tools* scegliete *PYTHONPATH manager*, quindi usare il tasto *Add path* per inserite il percorso in cui si trovano i vostri moduli, infine cliccate *Synchronize*.

- Nelle impostazioni di Spyder è possibile anche selezionare la modalità di *matplotlib* evitando di specificarla ogni volta all'avvio della console. Dal menù a tendina *Tools* scegliete *Preferences*, quindi selezionate *IPython console* dall'elenco a sinistra e cliccate sulla linguetta *Graphics*. Selezionate nella casella *Backend* la modalità voluta, ad esempio *Qt5* per l'interattiva, infine cliccate *Apply*.

- E' possibile realizzare il *debug* del codice che scrivete usando tasto debug di Spyder.