Elaborazione di Segnali Multimediali

Elaborazione di immagini a colori

L.Verdoliva, D.Cozzolino

In questa esercitazione vedremo come si elaborano le immagini a colori in Python estendendo le tecniche di enhancement e filtraggio definite per immagini monocromatiche.

1 Lo spazio RGB

Il numero di bit usati per rappresentare ogni pixel nello spazio RGB è detta *profondità*; se per esempio ognuna delle tre componenti è a 8 bit, ogni pixel a colori ha una profondità di 24 e l'immagine è detta *full color*. In tal caso i possibili colori che si possono visualizzare sono $(2^8)^3 = 16777216$. Eseguite lo script allegato *color_cube.py* per visualizzare il cubo dei colori, il cubo mostrato si limita a 16 colori differenti per lato al fine di avere una visualizzazione veloce. I vertici del cubo in coordinate (0,0,0) e (1,1,1) corrispondono rispettivamente al nero e al bianco, mentre gli altri vertici sono rappresentati dai colori primari (R, G, B) e secondari (C, M, Y). Nella seguente tabella si mostrano i relativi valori RGB:

Colore	**Valore RGB**
Nero	[0,0,0]
Blu	[0,0,1]
Verde	[0,1,0]
Ciano	[0,1,1]
Rosso	[1,0,0]
Magenta	[1,0,1]
Giallo	[1,1,0]
Bianco	[1,1,1]

Potete tramite il mouse cambiare l'angolo di vista da cui l'osservatore guarda il grafico tridimensionale. Notate che il cubo è pieno, cioè i punti rappresentativi di colori validi sono sia sulla sua superficie che al suo interno.

Per la visualizzazione delle immagini a colori useremo la funzione `plt.imshow()` che considera l'immagine a colori nello spazio RGB. Inoltre, fate molta attenzione al formato dei dati, in particolare la visualizzazione di un'immagine a colori richiede che le componenti RGB abbiano dinamica $[0, 255]$ se il dtype usato è `np.uint8`, mentre la dinamica è $[0, 1]$ per le rappresentazioni a virgola mobile (`np.float32`, `np.float64`). Per le trasformazioni di spazio di colore useremo le funzioni di SK-Image del modulo `skimage.color`. Tutte le funzioni di questo modulo considerano che l'immagine sia normalizzate nel range $[0, 1]$.

1.1 Esercizi proposti

1. *Lo spazio CMY e CMYK.* Se le componenti RGB sono state normalizzate nel range $[0, 1]$ è molto facile ottenere le componenti nello spazio CMY (Cyan, Magenta, Yellow), dato che queste componenti sono le complementari di quelle RGB:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

   Provate a scrivere una funzione con il seguente prototipo: `rgb2cmy(x)`, che legge un'immagine a colori e determina la rappresentazione CMY dell'immagine fragole.jpg visualizzandone le componenti.

   Scrivete poi una funzione dal prototipo: `rgb2cmyk(x)` che legge un'immagine a colori e determina la rappresentazione CMYK (Cyan, Magenta, Yellow, Black) di un'immagine visualizzandone le componenti. Tenete presente che quando si usano i pigmenti, il nero si ottiene usando uguali quantità dei pigmenti ciano, magenta e giallo, e quindi la componente K è pari al minimo fra C, M ed Y, e le nuove componenti dei pigmenti si ottengono sottraendo K, cioè C'=C-K, e così via.

2. *Lo spazio HSI.* La rappresentazione HSV si ottiene con `skimage.color.rgb2hsv`. Il modello HSV è leggermente diverso da quello HSI, dato che il solido di riferimento è una piramide rovesciata, con la cima nell'origine a differenza del caso HSI in cui il modello è una doppia piramide. Per questo motivo per il passaggio nello spazio HSI usate le funzioni disponibili sul sito del corso nella sezione materiale didattico.

   Visualizzate le componenti HSI dell'immagine fragole.jpg e quelle dell'immagine cubo.jpg che rappresenta proprio il cubo dei colori. In quest'ultimo caso si possono fare alcune interessanti considerazioni. Nell'immagine di tinta, si può notare la forte discontinuità lungo la linea a 45^o sul piano frontale del cubo, che è quello del rosso: si ha infatti lungo questa linea la transizione brusca tra valori alti (360^o) e valori bassi (0^o) della tinta, dovuta alla sua rappresentazione circolare. L'immagine di saturazione mostra valori più scuri verso il vertice del bianco, dove i colori diventano progressivamente meno saturi. Infine, nell'immagine di intensità, ogni pixel è semplicemente la media dei valori RGB del pixel corrispondente nell'immagine a colori.

2 Tecniche per l'elaborazione

I metodi di elaborazione di immagini su scala di grigi descritti nelle precedenti esercitazioni possono essere applicate facilmente alle immagini a colori, lavorando sulle singole componenti definite dallo spazio di colore in cui ci troviamo. Tuttavia, il risultato dell'elaborazione di ogni singola componente non è di solito equivalente all'elaborazione congiunta di tutti i piani (elaborazione vettoriale). Affinché il processing per componente e quello vettoriale siano equivalenti devono essere soddisfatte due condizioni:

1. l'algoritmo deve potersi applicare sia a scalari che vettori;
2. l'operazione su ogni componente del vettore deve essere indipendente dalle altre.

Ovviamente non sempre tali condizioni sono verificate, tuttavia di seguito focalizzeremo l'attenzione su quelle tecniche che operano sulle singole componenti dell'immagine. In teoria, ogni trasformazione può essere realizzata in uno qualunque degli spazi di colore, in pratica però alcune operazioni si adattano meglio a modelli specifici. Supponiamo per esempio di voler semplicemente modificare l'intensità di un'immagine. Nello spazio HSI basta modificare solo la componente I dell'immagine, lasciando le altre due inalterate. Invece, nello spazio RGB, così come in quello CMY, è necessario modificare tutte e tre le componenti. Sebbene la trasformazione nello spazio HSI coinvolga il minor numero di operazioni, ciò va barattato con il costo computazionale di convertire l'immagine da RGB a HSI. Di seguito analizzeremo le trasformazioni del colore (*color mapping*),

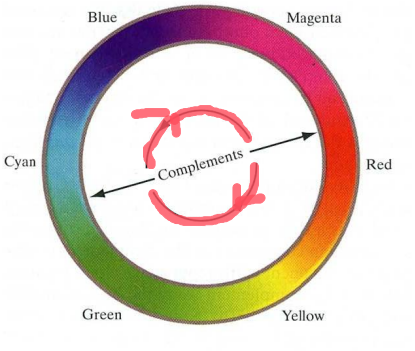

Figura 1: Cerchio dei colori.

in cui si elaborano i pixel di ogni piano di colore basandosi esclusivamente sul loro valore (sono in effetti le corrispondenti delle operazioni puntuali viste per le immagini monocromatiche) e le elaborazioni spaziali, in cui si effettua il filtraggio spaziale di ogni piano di colore.

2.1 Negativo

Nel cerchio mostrato in fig.1 i colori che sono l'uno opposto dell'altro si dicono complementari tra loro. L'interesse nei complementari deriva dal fatto che sono analoghi ai negativi delle immagini su scala di grigio. Sono quindi utili per enfatizzare i dettagli nascosti nelle regioni scure delle immagini a colori. Per ottenere il negativo dell'immagine basta effettuare il negativo di ogni componente RGB (nello spazio HSI invece bisogna utilizzare trasformazioni diverse per le tre componenti). Provate allora a visualizzare il negativo dell'immagine fragole.jpg, noterete come somigli al negativo dei film a colori: il rosso è sostituito col ciano (suo complementare), il bianco con il nero e così via.

2.2 Correzione di toni e colori

Le trasformazioni usate per modificare i toni di un'immagine sono selezionate in modo interattivo, nel senso che sperimentalmente si trova l'operazione che migliora la luminosità o il contrasto di un'immagine. Negli spazi RGB e CMY ciò significa che ognuna delle componenti sarà soggetta a tale trasformazione, mentre nello spazio HSI solo l'intensità sarà modificata. Per esempio, nel caso in cui si voglia migliorare il contrasto dell'immagine colori.jpg si può realizzare l'operazione potenza:

```
x = io.imread('colori.jpg')
x = np.float64(x)/255 # convertiamo nel range [0,1]
y = x ** 4

plt.figure();
plt.subplot(1,2,1); plt.imshow(x); plt.title('Immagine orignale');
plt.subplot(1,2,2); plt.imshow(y); plt.title('Immagine elaborata');
```

La normalizzazione è resa necessaria dal fatto che um'immagine a colori in formato double viene visualizzata correttamente solo se normalizzata. Modificate il valore di γ e osservate gli effetti sull'immagine. Si consideri poi l'immagine montagna.jpg, che presenta un aspetto complessivamente molto scuro. Si determini il valore di

γ che consente di migliorarne il contrasto. Provate a realizzare la stessa operazione nello spazio HSI (lavorando solo sull'intensità) e confrontate i risultati.

2.3 Equalizzazione

Ricordiamo che l'equalizzazione permette di ottenere un'immagine con istogramma approssimativamente piatto. In generale non ha molto senso equalizzare indipendentemente le componenti di un'immagine a colori, dato che viene modificata la proporzione con cui sono stati mescolati i colori primari. La modifica indipendente delle tre componenti cambia la tinta e la saturazione di ogni pixel introducendo una distorsione cromatica (gli istogrammi relativi alle tre immagini sono diversi quindi diversa risulterà la trasformazione applicata). L'approccio corretto è quello di elaborare solo l'intensità del colore, lasciando tinta e saturazione intatti. Da questo punto di vista è corretto rappresentare l'immagine nello spazio HSI ed equalizzare solo la componente di intensità (I). In realtà se si volesse tener conto anche della percezione del colore da parte del sistema visivo umano (più sensibile al verde che al rosso e al blu) si potrebbe utilizzare il modello YUV o YCbCr in cui la luminanza Y è data da una combinazione lineare dei primari, con pesi non uguali, a differenza del modello HSI, in cui I pesa ugualmente le componenti RGB.

Considerate l'immagine a colori `volto.tiff` e provate ad effettuare l'equalizzazione (utilizzando la funzione `skimage.exposure.equalize_hist`) sia nello spazio RGB su ognuna delle componenti che nello spazio HSI solo sulla componente I (ricordate di ritornare nello spazio RGB per la visualizzazione). Confrontate le immagini ottenute con i due approcci. Cosa potete osservare?

2.4 Color balancing

Spesso può essere utile realizzare un bilanciamento dei colori (*color balancing*) allo scopo di regolare le componenti di colore di un'immagine. Ci sono diversi possibili approcci, in ogni caso bisogna sempre tener conto che ogni operazione influenza l'equilibrio globale dei colori, dato che la percezione di un colore dipende anche da quelli circostanti. La proporzione di un colore può essere aumentata diminuendo la quantità di colore opposto. Allo stesso modo può essere diminuita aumentando la proporzione dei due colori immediatamente adiacenti oppure diminuendo la percentuale dei colori adiacenti al complementare. Se per esempio c'è troppo magenta in un'immagine allora può essere diminuito o rimuovendo del rosso e del blu oppure aggiungendo del verde.

Considerate l'immagine foto.jpg, che presenta un'elevata quantità di ciano, e, dopo aver effettuato la conversione nello spazio CMY, realizzate un bilanciamento delle componenti di colore cercando di riottenere l'immagine originale (memorizzata in foto_originale.tif). Se lavorate sulle componenti normalizzate tenete presente che per $\gamma > 1$ diminuite il peso del colore nell'immagine, il contrario accade per $\gamma < 1$.

2.5 Filtraggio spaziale

Supponiamo di considerare il filtro che realizza la media artitmetica dei valori che appartengono alla maschera, per cui se $\mathbf{x}(m, n, k)$ è l'immagine 3D, il risultato dell'elaborazione sarà:

$$\mathbf{y}(m, n, k) = \frac{1}{K} \sum_{m,n \in Mask} \mathbf{x}(m, n, k)$$

con K numero di pixel appartenenti alla maschera. Dalla formula è facile riconoscere che ogni componente dello spazio RGB ottenuta in uscita non è altro che la media aritmetica dell'elaborazione realizzata su ogni singola componente. Per realizzare il filtro media aritmetica nello spazio RGB di un'immagine a colori memorizzata nella variabile `x` bisogna realizzare i seguenti passi:

```
R = x[:,:,0]; G = x[:,:,1]; B = x[:,:,2];
fR = ndi.uniform_filter(R, (k,k))
fG = ndi.uniform_filter(G, (k,k))
fB = ndi.uniform_filter(B, (k,k))
y = np.stack((fR,fG,fB),-1)
```

In realtà le funzioni nel modulo `scipy.ndimage` possono operare direttamente nello spazio 3D, ma si deve definire opportunamente la maschera del filtro che deve essere 3D. Ad esempio il filtro media aritmetica nello spazio RGB si può eseguire con un singolo comando:

```
y = ndi.uniform_filter(x, (k,k,1))
```

Notate che il secondo parametro deve essere composto da 3 elementi che indicano le dimensioni del filtro 3D, inoltre la terza dimensione è pari ad 1 in modo che il filtraggio operi separatamente su ogni singola componente. Se si volesse realizzare lo filtro media aritmetica solo sulla componente I nello spazio HSI:

```
from color_convertion import rgb2hsi, hsi2rgb

w = rgb2hsi(x)
H = w[:,:,0]; S = w[:,:,1]; I = w[:,:,2];
fI = ndi.uniform_filter(I, (k,k))
w = np.stack((H,S,fI),-1)
y = hsi2rgb(w)
```

Si consideri allora l'immagine lenac.jpg, utilizzate un filtro di smoothing di dimensioni 5×5 operando su ogni singola componente RGB, provate poi a realizzare questa stessa operazione nello spazio YUV solo sulla componente di intensità. Confrontate le immagini ottenute sia visivamente che mostrando a video la differenza. Noterete che le due immagini non sono perfettamente identiche, ciò è dovuto al fatto che nello spazio RGB ogni pixel è pari al color medio dei pixel nella finestra 5×5, mentre mediare solo le intensità non altera il colore originale dei pixel (dato che tinta e saturazione non sono stati modificati). Questo effetto aumenta al crescere della dimensione del filtro. Ripetete l'esperimento usando l'immagine fiori.tif con una finestra di dimensioni 25×25.

Infine provate a realizzare un esperimento in cui effettuate l'enhancement di un'immagine a colori mediante un filtro di sharpening. Applicate il filtro laplaciano all'immagine fiori.jpg, lavorate sia su RGB che su I, e confrontate i risultati. Per eseguire il filtraggio nell'immagine RGB tramite un singolo comando dovete estendere la maschera 2D in 3D tramite la funzione `np.expand_dims()`:

```
h = np.array([[0,-1,0],[-1,4,-1],[0,-1,0]], np.float64) # maschera 2D
h = np.expand_dims(h, -1)  # maschera 3D
```

2.6 Esercizi proposti

1. *Variazione del colore*. Considerate l'immagine Azzurro.jpg, dove è presente un accappatoio azzurro, e generate l'immagine Rosso.jpg, in cui solo l'accappatoio diventa rosso. A tal fine, passate nello spazio di colore HSI, individuate la regione dell'immagine occupata dall'accappatoio in base ai suoi valori di tinta, saturazione e luminanza, e solo in tale regione operate una opportuna variazione della sola tinta.

2. *Enhancement*. Si vuole realizzare l'enhancement dell'immagine a colori primopiano.jpg allo scopo di migliorarne sia la luminosità che il contrasto. Scrivete il codice python in cui effettuate tutte le operazioni che ritenete necessarie (giustificando le scelte) sia nello spazio RGB che in quello HSI e stabilite qual è lo spazio più conveniente in cui lavorare. Infine, ruotate l'immagine in modo che le linee che si intravedono nello sfondo risultino perfettamente orizzontali.

3. *Filtraggio in frequenza*. Data l'immagine foto_originale.tif, si vuole realizzare il filtraggio dell'immagine nel dominio della frequenza mediante il seguente filtro:

$$H(\mu,\nu) = \begin{cases} 1 & |\mu| \leq 0.10\,,\ |\nu| \leq 0.25, \\ 0 & \text{altrimenti} \end{cases}$$

   Scrivete il codice relativo e mostrate l'immagine filtrata.

4. *Canny edge-detector*. Per realizzare l'algoritmo di Canny è possibile usare: `skimage.feature.canny`:

```
from skimage.feature import canny
mappa = canny(x, sigma, low_threshold, high_threshold)
```

   Applicate il canny edge detector all'immagine casa.tif con $\sigma = 1$, $T_{low} = 0.0325$ e $T_{high} = 0.13$. Confrontate la mappa dei contorni con quelle che si ottengono con le tecniche precedenti. Provate poi a modificare i parametri dell'algoritmo, come le dimensioni del filtro di smoothing, ovvero la sua deviazione standard, e i valori delle due soglie e osservate gli effetti sulla mappa prodotta. Testate l'algoritmo anche sull'immagine headCT.tif, (fate attenzione a settare opportunamente le due soglie).

3 Tecniche class-based

Le tecniche class-based si basano su un'elaborazione globale dell'immagine per realizzare la segmentazione. Gli approcci che esamineremo usano operazioni di thresholding o clustering per produrre la mappa di etichette, i cui valori identificano la regione associata ai pixel dell'immagine.

Data l'estrema semplicità di implementazione, la segmentazione basata su threshoding è molto utilizzata in varie applicazioni. Cominciamo col considerare un'immagine monocromatica costituita da un oggetto (foreground) su uno sfondo (background), in modo che i pixel dell'oggetto e quelli dello sfondo presentino livelli di intensità raggruppati in due modi dominanti. Un modo ovvio per estrarre gli oggetti dallo sfondo è quello di selezionare una soglia T che separi i due modi. Pertanto ogni pixel in posizione (m,n) per cui risulta $x(m,n) \geq T$ è classificato come *object point* altrimenti come *background point*. La mappa delle etichette, $y(m,n)$, si ottiene quindi nel modo seguente:

$$y(m,n) = \begin{cases} 1 & x(m,n) \geq T \\ 0 & x(m,n) < T \end{cases}$$

Quando T è costante si parla di thresholding globale; in tal caso un modo per selezionare il valore della soglia è quello di osservare l'istogramma dell'immagine oppure procedere in modo interattivo considerando diverse soglie fin quando il risultato diventa soddisfacente per l'osservatore.

Una procedura automatica può essere realizzata applicando l'algoritmo K-means, che è disponibile nel package SKLearn, e potete applicare all'immagine monocromatica `x` nel seguente modo:

```
from sklearn.cluster import k_means
d = np.reshape(x, (-1, 1))
centroid, idx, sum_var = k_means(d, K)
y = np.reshape(idx, x.shape)
```

dove `K` è il numero di regioni (dette anche *cluster*), `centroid` contiene il valore medio di ogni cluster, `idx` il vettore contenente le etichette di ogni pixel, e `sum_var` e la somme delle K varianze calcolate per ogni cluster. Applicate questo algoritmo, al variare del numero di classi K, all'immagini granelli_riso.tif.

L'algoritmo K-means può essere applicato anche ad immagini a colori. In tal caso il clustering va effettuato sulla base dei vettori che determinano lo spazio di rappresentazione dei pixel dell'immagine. Applichiamo K-means utilizzando lo spazio di colori RGB. Leggiamo quindi un'immagine a colori e facciamo attenzione a fornire correttamente l'ingresso, che deve essere una matrice bidimensionale in cui ogni riga è la rappresentazione del colore dei pixel dell'immagine:

```
from sklearn.cluster import k_means
L = x.shape[-1]
d = np.reshape(x, (-1, L))
centroid, idx, sum_var = k_means(d, K)
y = np.reshape(idx, x.shape[:-1])
```

Applicate l'algoritmo alle immagini a colori Fiori256.jpg e lenac.jpg, scegliendo opportunamente il valore da assegnare a K.

3.1 Esercizi proposti

1. *Thresholding locale*. Si vuole segmentare l'immagine yeast.tif ed individuare gli anelli grigio chiaro che circondano i cerchi bianchi; a tale scopo segmentate l'immagine usando una soglia variabile basata sulle proprietà statistiche locali dell'immagine stessa. Scrivete la funzione `thresholding_locale(x)` in cui:

   (a) si calcola l'immagine delle deviazioni standard locali, $\sigma_L(m,n)$, usando una finestra 3×3;

   (b) si ottiene la mappa binaria usando una soglia diversa per ogni posizione (m,n) come:

   $$g(m,n) = \begin{cases} 1 & x(m,n) > a\sigma_L(m,n) \ \text{ AND } \ x(m,n) > b\, m_G \\ 0 & \text{altrimenti} \end{cases}$$

   dove m_G è la media globale, e si pone $a = 30$ e $b = 1.5$.