Elaborazione delle Immagini

Laboratorio 5

Obiettivi:

- Segmentazione
- Clustering
- Descrittori delle immagini

Ricordate: per processare le immagini è sempre conveniente trasformale in valori double tra 0 e 1 con **im2double**.

Ricordate: imshow visualizza le immagini in modo corretto se hanno valori tra 0 e 255 (uchar8), se hanno valori tra 0 e 1 (double) o sono valori logici.

Ricordate: se volete saperne di più sulle funzioni Matlab usate, consultate l'help o la documentazione con i seguenti comandi da console:
help <funzione>
doc <funzione>

Scrivete il codice di ogni esercizio in uno script separato (labX_1.m, labX_2.m, ...)

A1. Aprite il file **"compute_local_descriptors.m"** e analizzate il codice. La funzione permette di calcolare dei descrittori passando una funzione apposita (*compute_funct*) come parametro. L'immagine è divisa in tasselli quadrati di dimensione data (*tsize*) e su ogni tassello è calcolato un descrittore. I tasselli possono essere sovrapposti o meno settando opportunamente il passo di creazione dei tasselli *tstep*. I descrittori sono poi raccolti, per riga, in un unico array. L'output è una struttura con tre informazioni: i descrittori (*descriptors*), il numero di tasselli per riga (*nt_rows*) e il numero di tasselli per colonna (*nt_cols*).

B1. Scrivete una funzione **"compute_average_color.m"** per calcolare il colore medio di un tassello. L'input è una immagine, l'output è un vettore riga con i valori medi dei tre canali [R,G,B].

In uno script a parte:

C1. Caricate l'immagine "**sweets.png**" in **im** e visualizzatela.

D1. Chiamate la funzione **compute_local_descriptors** passando in input l'immagine, la dimensione di un tassello (es. 5), un passo (es. 5 per avere tasselli non sovrapposti) e la funzione **compute_average_color.** Per passare una funzione come parametro dovete premettere al nome della funzione il carattere @. Mettete i risultati in una variabile **out.**

E1. Usando la funzione Matlab **kmeans** raggruppate (clusterizzate) i descrittori in 5 gruppi omogenei. L'output della funzione (**labels**) sono le etichette assegnate a ciascun descrittore.

F1. Costruite l'array bidimensionale delle etichette, **img_labels,** usando la funzione **reshape** e il numero di tasselli per riga e per colonna che sono contenuti in **out.** In pratica dovete costruire una immagine delle etichette. L'immagine ottenuta è più piccola dell'originale (dipende dalla dimensione del tassello e dal passo). Scalatela con **imresize** alle dimensioni corrette usando il metodo nearest.

G1. Visualizzate in una unica figura, **im** usando **imshow** e **img_labels** usando **imagesc** (non dimenticate l'opzione axis image). Confrontate il risultato con l'immagine originale. Cosa notate? Rieseguite più volte lo script con gli stessi parametri. Cosa notate?

H1. Provate a cambiare la dimensione del tassello e/o il passo e/o il numero di gruppi che l'algoritmo **kmeans** deve cercare. Analizzate i risultati.

Il clustering è un algoritmo non supervisionato che organizza una serie di dati in gruppi omogenei (clusters). Ogni dato è assegnato ad uno dei cluster creati mediante una etichetta. Dati simili hanno assegnata la stessa etichetta. Nell'esercizio noi usiamo il clustering per identificare le regioni che sono simili tra loro in termini di colore. Le caramelle con lo stesso colore hanno ricevuto la stessa etichetta. L'immagine è analizzata a tasselli e quindi le regioni sono scalettate. L'immagine delle etichette può essere usata con un algoritmo di labeling per determinare le singole regioni e quindi segmentare l'immagine per colore.

A2. Caricate l'immagine "**segmentation.png**" nella variabile **segm** e provate a segmentarla come nell'esercizio 1. Cosa notate? Come potete ottenere un risultato migliore?

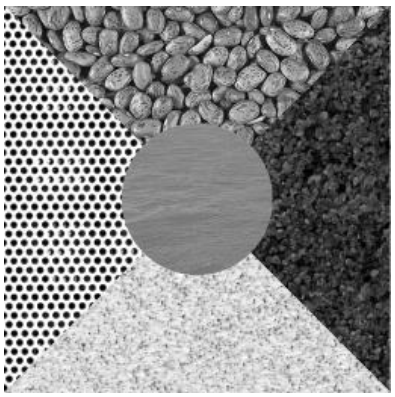
B2. Provate a creare un'altra funzione che calcola un diverso descrittore colore e testatela. Che risultati avete?

A3. Caricate l'immagine "**text1.png**" in **text1**. Provate a segmentarla per colore (luminosità). Che risultati ottenete?

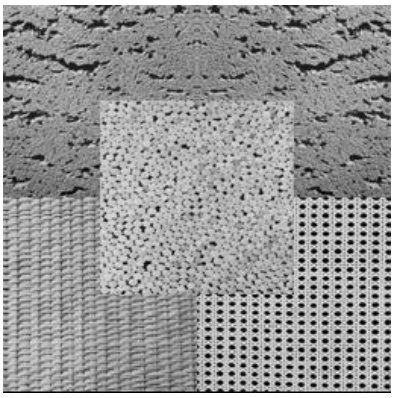
B3. Create una funzione che calcoli un descrittore di texture (es. deviazione standard). Usatela per segmentare l'immagine **text1**. Che risultati ottenete?

C3. Caricate l'immagine "**text2.png**" in **text2** e provate a segmentarla per texture. Che risultati ottenete?

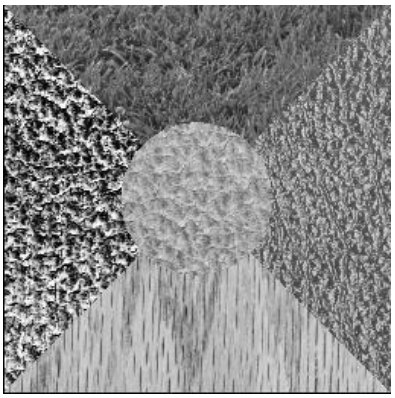
D3. Caricate l'immagine "**text3.png**" in **text3** e provate a segmentarla per texture. Che risultati ottenete?



Text1



Text2



Text3

A4. Il file "**compute_lbp.m**" contiene la funzione **compute_lbp** che calcola il descrittore di texture LBP: Local Binary Patterns histogram. E' un istogramma particolare che codifica in una serie di 8bit la struttura locale andando a confrontare il valore di un pixel p con quelli del suo vicinato. ATTENZIONE: questo descrittore vuole immagini a livelli di grigio e a valori tra 0 e 255.

B4. Provate a segmentare le immagini **text1**, **text2**, e **text3** usando questo descrittore di texture.

example

7	6	5
7	7	4
7	9	8

threshold

1	0	0
1		0
1	1	1

weights

1	2	4
128		8
64	32	16

Pattern=11110001

Decimal=1+16+32+64+128=241

I Local Binary Patterns hanno diverse varianti. Per saperne di più guardate il pdf allegato **LBP-chapter.pdf**.

(5)

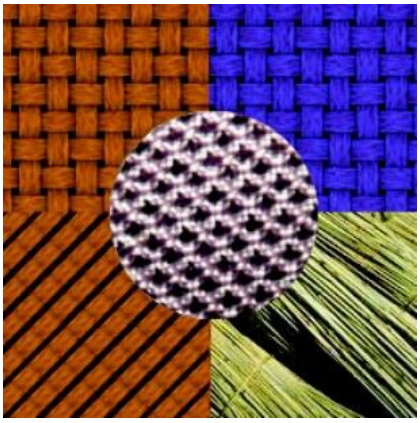
A5. Caricate l'immagine '**colortext.png**' in una variabile **im.**

B5. Provate a segmentarla per colore

C5. Provate a segmentarla per texture

D5. Create una funzione che calcola un descrittore composito di colore e texture. In pratica create una funzione che concatena i descrittori creati da altre due funzioni già scritte: una per il colore e una per le texture.

E5. Usando la nuova funzione, provate a segmentare l'immagine **coltext**



---

(6)

A6. Caricate l'immagine '**flowerbed.png**' in **im**.

B6. Provate a segmentarla per colore e texture combinate.

A7. Caricate l'immagine '**cosamanca-0.png**' in **im** e visualizzatela. L'obiettivo è quello di ottenere, separatamente, le immagini delle 4 vignette (quattro immagini in output).

B7. Scrivete un algoritmo per trovare <u>automaticamente</u> le regioni delle 4 vignette e mettete le immagini in un array di celle **V**: **V{1}**, **V{2}**, **V{3}** e **V{4}** Un array di celle è indicizzato con le graffe. Per creare un array di celle vuoto usate **V={}.** Potete poi indicizzare direttamente una cella e assegnargli l'immagine della vignetta i-esima: **V{1}**=vignetta1, **V{2}**=vignetta2,... Matlab crea automaticamente le celle di posizione indicata.

C7. Una volta trovate le vignette, usando Matlab, trovate... cosa manca nelle vignette **V{2}**, **V{3}** e **V{4}** rispetto alla **V{1}**. Potete usare un ciclo for per processare con il vostro algoritmo le coppie di vignette **V{1}** e **V{j}** con j=2,3,4.