Elaborazione delle Immagini

Laboratorio 6

Obiettivi:

- Classificazione

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.

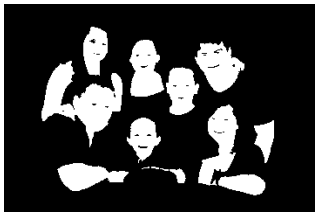
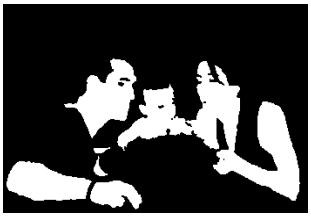
Ricordate: se volete saperne di più sulle funzioni Matlab usate, consultate l'help o la documentazione con i seguenti comandi da console:
help <funzione>
doc <funzione>

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

L'obiettivo è quello di **classificare** i pixel delle immagini in due categorie: pixel di pelle e pixel di non-pelle. Le immagini **testX.jpg** contengono le immagini che devono essere processate (test images). Le immagini **testX-gt.jpg** contengono le maschere binarie ideali corrispondenti alle regioni dei pixel di pelle (groundtruth images).





Bisogna processare le test images (riga sopra) per ottenere delle maschere binarie il più simili possibili alle groundtruth images (riga sotto).

Per addestrare un classificatore a riconoscere i pixel di pelle da quelli di non pelle, è necessario creare due set di dati che contengono esempi di pixel di pelle e di pixel di non pelle.

La prima cosa da fare è recuperare un **dataset,** cioè l'insieme dei dati che rappresentano pixel di pelle e pixel di non pelle. Nella cartella **data/** trovate due sotto-cartelle: **skin/** e **noskin/**. Queste due cartelle contengono delle immagini che possono essere usate per raccogliere pixel di pelle e di non pelle in due strutture dati. Analizzate queste cartelle.

NOTA: le immagini delle persone sono state recuperate dal sito http://humanae.tumblr.com/ © Angélica Dass.

Dato che a noi interessano i pixel, possiamo utilizzare direttamente due immagini che contengono solo pixel di pelle '**skin.png**' e solo pixel di non pelle '**noskin.png**' come strutture dati. Queste due immagini sono già state create facendo un collage (con un programma di editing di immagini) di regioni di pelle e di non pelle prelevate dalle immagini che stanno nelle cartelle del dataset. Analizzate queste due immagini.

(1)

Scriviamo uno script **classify_rule** che applica un classificatore a regole (o a parallelepipedo) per trovare i pixel di una immagine che sono di pelle. Il classificatore usa solo i dati provenienti da **skin.png.**

A1. Caricate l'immagine di test "**test1.jpg**" in una variabile **image**. Caricate l'immagine "**skin.png'**" in una variabile **skin.**

B1. Calcolate media e deviazione standard dei valori RGB dei pixel di skin (media e deviazione standard di ogni canale RGB). Mettete le medie in una variabile **m** e le deviazioni standard in una variabile **v.** Potrebbe essere utile ristrutturare l'immagine in un array che ha una terna RGB per riga.

C1. Settate una variabile **k** a 1. Questa variabile sarà un fattore moltiplicativo per le deviazioni standard.

D1. Supponendo che le medie dei pixel di skin siano m=[mr,mg,mb] e le deviazioni standard siano v=[vr,vg,vb], il classificatore a regole, classifica un pixel p=[r,g,b] come pixel di pelle se e solo se valgono contemporaneamente le seguenti condizioni:

$$mr - k * vr \leq r \leq mr + k * vr \quad \text{and}$$

$$mg - k * vg \leq g \leq mg + k * vg \quad \text{and}$$

$$mb - k * vb \leq b \leq mb + k * vb$$

Applicate le regole ad ogni pixel dell'immagine di input ottenendo una maschera binaria da mettere nella variabile **predicted**. Visualizzate il risultato con la funzione fornita **show_result** passando l'immagine di input e la maschera. Che cosa notate?

E1. Caricate l'immagine di groundtruth "**test1-gt.png"** in una variabile **gt**. Trasformate i valori di **gt** in valori logici (0 e 1 booleani).

F1. Usate la funzione **confmat** per valutare la bontà del classificatore. La funzione vuole in input **gt** e **predicted**. L'output della funzione è una struttura che contiene la matrice di confusione (*cm_raw*), la matrice di confusione con i valori in percentuale (*cm*), le etichette delle righe e colonne della matrice di confusione (*labels*, uguali a 0 e 1 nel nostro caso) e l'accuratezza del classificatore (*accuracy*). Confrontate il valore di accuratezza con i valori della matrice di confusione cm.

G1. Provate a modificare il valore di k e analizzate i risultati.

La funzione **confmat** calcola la matrice di confusione. In pratica, per ogni classe, conta quante volte un valore di classe i nella **gt** è stato riconosciuto tale in **predicted**. E conta quante volte un valore di classe i nella **gt** è stato riconosciuto di classe j nella **predicted** (i<>j). La diagonale della matrice rivela quante volte il classificatore ha azzeccato la risposta giusta. Le altre celle indicano gli errori commessi.

Per saperne di più: https://en.wikipedia.org/wiki/Confusion_matrix

Scrivete uno script **classify_mindist** che applica un classificatore a minima distanza per trovare i pixel di una immagine che sono di pelle. Il classificatore deve usare i dati provenienti da **skin.png** e quelli che provengono da **noskin.png.**

A2. Caricate l'immagine "**skin.png'**" in una variabile **skin.** Caricate l'immagine "noskin.png" in una variabile **noskin**. Trasformate tutte le immagini con **im2double**.

B2. Calcolate la media dei valori RGB dei pixel di skin e noskin (media di ogni canale RGB). Mettete le medie in una variabile **ms** per i pixel di skin e in **mns** per i pixel di noskin**.**

C2. Caricate l'immagine di test "**test1.jpg**" in una variabile **image**. Ristrutturate in un nuovo array **pixs** i pixel dell'immagine in modo tale da avere una terna RGB per ogni riga dell'array.

D2. Supponendo di voler classificare un pixel p, il classificatore a minima distanza (sotto certe condizioni, vedi sotto), assegna p alla classe di skin se e solo se vale la seguente regola:

$$dist(p, ms) < dist(p, mns)$$

Con dist la distanza Euclidea. Usate la funzione Matlab **pdist2** per calcolare la distanza Euclidea tra ogni pixel in **pixs** e la media **ms**. Mettete il risultato in una variabile **dist_s.** Fate la stessa cosa con **mns** e mettete il risultato in una variabile **dist_ns**.

E2. Create un vettore **predicted** che conterrà in posizione i-esima il valore logico vero (1) se **dist_s(i)** < **dist_ns(i)**.

F2. Ristrutturate **predicted** per trasformarlo in un array bidimensionale delle stesse dimensioni dell'immagine di input e visualizzate il risultato con **show_result** passandogli **image** e **predicted**.

G2. Caricate l'immagine di groundtruth "**test1-gt.png"** in una variabile **gt**. Trasformate i valori di **gt** in valori logici (0 e 1 booleani).

H2. Calcolate le performance del classificatore con **confmat** e analizzate i valori ottenuti. Ci sono differenze con il risultato dell'esercizio precedente? Se sì, quali?

Il classificatore a minima distanza confronta un pixel con il rappresentante dei pixel di pelle (**ms**) e con quello dei pixel di non pelle (**mns**). Il pixel è poi etichettato come pelle se è più vicino al rappresentante dei pixel di pelle rispetto a quello dei pixel di non pelle. Altrimenti è etichettato come pixel di non pelle. Il classificatore a minima distanza è un classificatore Bayesiano che opera sotto le seguenti condizioni:

1. Descrittori distribuiti con uguale varianza all'interno delle classi.
2. Descrittori statisticamente indipendenti.
3. Probabilità a priori uguali per tutte le classi.

Scrivete uno script **classify_bayesian** che applica un classificatore Bayesiano per trovare i pixel di una immagine che sono di pelle. Il classificatore deve usare i dati provenienti da **skin.png** e quelli che provengono da **noskin.png.**

A3. Caricate l'immagine "**skin.png'**" in una variabile **skin.** Caricate l'immagine "noskin.png" in una variabile **noskin**. Trasformate tutte le immagini con **im2double**.

B3. Ristrutturate **skin** e **noskin**, in modo tale da avere una singola terna RGB per riga.

C3. Concatenate, per riga, in un array **train_values** gli array **skin** e **noskin**. Create un array **train_labels** dove le prime n righe (n è pari al numero di righe dell'array **skin**) hanno valore 1 e le altre m righe (m è pari al numero di righe dell'array **noskin**) hanno valore 0.

D3. Usando la funzione Matlab **fitcnb** create un classificatore Bayesiano **classifier_bayes,** passando alla funzione gli array **train_values** e **train_labels.** Il classificatore viene automaticamente addestrato sui dati che gli sono stati passati e sarà in grado di classificare nuovi dati.

E3. Salvate la variabile **classifier_bayes** su file usando il comando Matlab **save**('classifier_bayes', 'classifier_bayes'). Potete poi recuperare la variabile con il comando Matlab **load**('classifier_bayes'). Vi servirà più avanti.

F3. Caricate l'immagine di test "**test1.jpg**" in una variabile **image**. Ristrutturate i pixel di **image** in un array **test_values** con un pixel per riga. Usate la funzione Matlab **predict** per classificare questi valori. La funzione vuole in input la variabile **classifier_bayes** con il classificatore e l'array **test_values**. La funzione ritorna un array di etichette (una per terna di valori) che dovete memorizzare in una variabile **test_predicted**.

G3. Ristrutturate **test_predicted** per trasformarlo in un array (**mask_predicted**) delle stesse dimensioni dell'immagine di input e visualizzate il risultato con **show_result** passandogli **image** e **test_predicted**.

H3. Caricate l'immagine di groundtruth "**test1-gt.png"** in una variabile **gt**. Trasformate i valori di **gt** da uint8 a double.

I3. Calcolate le performance del classificatore con **confmat** (passategli **gt** e **test_predicted**) e analizzate i valori ottenuti. Ci sono differenze con il risultato dell'esercizio precedente? Se sì, quali?

L3. Analizzate le proprietà del classificatore creato. (doc fitcnb)

E’ possibile anche valutare quanto un classificatore ha “imparato bene” andando a valutare i risultati di training. In pratica si devono passare i dati di training (train_values) al classificatore e vedere come le etichette predette (train_predicted) combaciano con le etichette di training (train_labels). In pratica le istruzioni sono queste:

```
train_predicted = predict(classifier_bayes, train_values);
performance = confmat(train_labels,  train_predicted);
```

Sui dati di training si dovrebbe avere idealmente un 100% di accuratezza. In realtà un classificatore difficilmente raggiunge un tale livello di accuratezza. Le performance di training sono in generale molto più elevate di quelle di test.

Il processo di training di un classificatore è fatto off-line. Una volta che il classificatore è stato creato, può essere riutilizzato per classificare nuovi dati. E’ importante quindi salvare il classificatore per poterlo riutilizzare in seguito senza dover rifare ogni volta il training.

Il processo d’uso di un classificatore è quindi questo:

- Creazione dei dati di training
- Creazione del classificatore (sui dati di training)
- Applicazione del classificatore su dati nuovi (test)

Per saperne di più: http://artint.info/html/ArtInt_181.html

Scrivete un script **classify_cart** che applica un classificatore CART (Classification And Regression Tree) per trovare i pixel di una immagine che sono di pelle. Il classificatore deve usare i dati provenienti da **skin.png** e quelli che provengono da **noskin.png.**

A4. Creare una copia dello script del precedente esercizio (classify_bayesian.m) e rinominatelo in **classify_cart.m**.

B4. Cambiate nome alla variabile del classificatore in **classifier_cart.** Sostituite la funzione Matlab **fitcnb** con la funzione Matlab **fictree.** Aggiornate le istruzioni sul salvataggio del classificatore. Eseguite lo script. Il classificatore costruisce un albero binario di regole. Ad ogni bivio applica una regola su un valore del descrittore di input per decidere come classificare i dati.

C4. Analizzate i risultati del classificatore nella fase di test dell'immagine. Confrontateli con i precedenti.

D5. Analizzate i risultati del classificatore in fase di training. Cosa notate?

E4. Usate la funzione Matlab **view** per vedere le regole create dal classificatore (basta passargli il classificatore). Le variabili X1, X2 e X3 corrispondono alle tre feature usate (i valori RGB del pixel rispettivamente).

F4. Provate a riaddestrare il classificatore dandogli solo 1000 dati di skin e 1000 dati di noskin (dovete anche ricreare le label secondo la nuova cardinalità). Confrontate i risultati ottenuti ora rispetto a quelli di prima.

G4. E' possibile vedere graficamente l'albero delle decisioni. Usate il seguente comando Matlab **view**: view(classifier,'Mode','Graph'). La figura che si apre mostra l'albero con i vari split generati e le rispettive regole usate.

In generale, per un classificatore, più dati sono usati per il training e più il classificatore è in grado di apprendere le differenze e generalizzare le regole per la classificazione. Ma se molti dei dati che gli sono passati sono molto simili o identici, il classificatore non impara nulla di più.

E' molto importante che i dati siano il più possibile rappresentativi della realtà. Per il nostro problema di classificazione di pixel in skin e noskin, la classe skin è concettualmente ben definita e i pixel hanno ragionevolmente delle caratteristiche simili tra loro. La classe di noskin invece è potenzialmente infinita ed estremamente eterogenea.

Per saperne di più: http:// www.stat.wisc.edu/~loh/treeprogs/guide/wires11.pdf

Scrivete uno script **classify_knn** che applica un classificatore k-Nearest Neighbor per trovare i pixel di una immagine che sono di pelle. Il classificatore deve usare i dati provenienti da **skin.png** e quelli che provengono da **noskin.png.**

A5. Creare una copia dello script dell'esercizio 4 (classify_bayesian.m) e rinominatelo in **classify_knn.m**.

B5. Cambiate nome al classificatore in **classifier_knn**. Sostituite la funzione Matlab **fitcnb** con la funzione Matlab **ficknn.** Aggiornate le istruzioni sul salvataggio del classificatore. Eseguite lo script. Il classificatore costruisce la struttura dati necessaria per la classificazione. Di default viene considerato 1 solo vicino per la classificazione. Quindi un dato da classificare è confrontato con tutti quelli di training e gli viene assegnata l'etichetta del più simile. NOTA: se la classificazione dovesse essere troppo lenta, passate al classificatore meno dati (es. 5000 per classe).

C5. Analizzate i risultati del classificatore nella fase di test dell'immagine. Confrontateli con i precedenti.

D5. Analizzate i risultati del classificatore in fase di training. Cosa notate?

E5. Riaddestrare il classificatore prendendo solo 5000 dati di skin e 5000 dati di noskin. Cambiate anche la chiamata alla funzione **fitcknn** come segue:

classifier = fitcknn(training_values, training_labels, 'NumNeighbors', N);

N è una variabile che andrà settata prima della chiamata e indica il numero di campioni simili da usare per la classificazione. In questo caso un dato riceve l'etichetta che occorre maggiormente tra gli N campioni di training simili al dato. Provate con N=3, 7, 11, 15. Analizzate i risultati. Cosa notate?

Il classificatore knn è un classificatore non parametrico dove la classificazione avviene per verosimiglianza di una dato con quelli usati per il training. Se il dato da classificare è in una regione dello spazio delle feature popolata da feature con etichetta X allora anch'esso avrà verosimilmente etichetta X.

Per saperne di più: http://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/

Fino ad ora abbiamo usato l'immagine 'test1.jpg' per gli esercizi e come features di classificazione i valori RGB dei pixel.

A6. Provate a classificare i pixel di skin delle immagini **test2.jpg**, **test3.jpg** e **test4.jpg** in RGB. Potete recuperare i classificatori già salvati e usateli per classificare le nuove immagini.

B6. Provate a classificare i pixel di skin delle immagini **test1.jpg**, **test2.jpg**, **test3.jpg** e **test4.jpg** usando altri spazi colore o altre feature. Quando cambiate le feature dovete riaddestrare tutti i classificatori con i nuovi dati.